# A prototype energy management system for a solar powered cycle

Full dissertation submitted for
MSc (Elec. Eng)



**Author:**
Gordon Andrew Webber
University of Cape Town

**Supervisor:**
Michel Malengret
Department of Electrical Engineering
University of Cape Town

**Submitted:**
August 2002

# Acknowledgements

Michel Malengret for his support and supervision.

Michel "The Mathemagician" Nlandu for his assistance in reducing the mathematics into something comprehensible.

Benaya Sebitosi for his critical thinking and friendship.

The Solar esCape Team for their support and encouragement.

Chris Wzoniak, Johan Smuts and Johan Malan for their assistance, patience and good humour.

Charles Loubser for spending the time finding the "peg line sections" of the R27.

Denzil Stickells for "Plan B" and reviewing this dissertation.

My loving parents for their patience, support and garage space.

# Declaration

I hereby declare that this dissertation does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.
I received much assistance and acknowledge it gratefully but none contributed directly to the material herein.

# Synopsis

The objective of this dissertation is to develop a functional prototype energy management system (EMS) for the prototype Solar esCape solar powered cycle (solar cycle).

It focuses on the development of the EMS from an optimal control theory perspective, with particular emphasis on the minimum time problem (MTP).

An EMS for a solar cycle has two objectives; firstly, it has to use the available energy to complete the day's stage in the minimum time possible. Secondly, it has to aid the team in making tactical decisions.

The EMS must respond to, and accommodate, a number of different parameters. It should also be able to predict the solar cycle's performance at any point during the stage. For the prototype, effort is focussed on the first objective and only the gradient and the uncertainty in the aerodynamic power losses due to wind are considered.

The MTP is solved via the shooting method, using three different methods to solve the initial value problem inherent in the shooting method. The best of the three methods was to be selected for use in the EMS.

Two formal numerical methods are used, the projected descent method and the Nelder-Mead method. In addition, a new method (Simple Method) is introduced. To provide two reference points for comparing the results of the optimal solutions produced, a constant power strategy and a constant velocity strategy are used to solve the MTP. The constant power strategy is the simplest solution to the problem statement. The constant velocity strategy is commonly regarded as the optimal solution.

The following insights and results were forthcoming:

Regeneration does not improve the performance of the solar cycle. In the presence of an unknown wind the constant velocity controller cannot meet the energy constraint. The Simple Method produces worse results than the constant velocity controller does. The projected gradient method places excessive computational demand on the PC used and was not able to solve the MTP. The Nelder-Mead method becomes attracted to a single solution that is worse than its starting point.

When faced with an unknown wind, the MTP cannot be minimised by the shooting method.

Thus far, no attempt to account for the effects of uncertainty has been made.

A two stage predictive controller (TSPC) is proposed as a means of compensating for the uncertainty. The TSPC generates a number of forecasts over a limited time horizon. After a short interval, the forecasts are evaluated and the control trajectory matching of the most accurate forecast is selected for use.

Each of the methods used to solve the MTP were then applied in the TSPC.

The Simple Method offers similar performance to the constant velocity controller, but suffered convergence problems.

The line search in the projected descent method fails at some point in the solution process. This renders the method useless in solving the MTP.

The Nelder-Mead solver gives a mixed set of results. It does not meet the energy constraint and is erratic.

In the light of the problems experienced with the methods above, a second new method is introduced. The Simple Solver 2 (SS2) is a simple constant power solver.

Its key feature is that it is robust. This robustness coupled to the TSPC's ability to handle uncertainty provides the closest match to the requirements of the EMS.

The SS2 results are worse than the constant velocity results, but the execution times are quicker. It does not meet the energy constraint. It was selected for use in the EMS.

As none of the methods used are adequate for the task, a more robust non-linear numerical method is needed to minimise the MTP. The difficulties of implementing the optimal control solver prompt speculation on alternatives.

The TSPC appears to handle uncertainty very well.

For implementation, the EMS was split into two parts. The battery control function was implemented in a DSP, and the TSPC function was implemented in a laptop. A serial connection linked the two devices. The DSP used is the Texas Instruments TMS320F243 control optimised device.

Two field tests were attempted to assess the EMS performance.

The EMS suffered from one confirmed electronic failure and at least one suspected fault or failure.

Two simulations were performed, the first was a look at the performance of the prototype had the wind not been present on the test runs, and the second examines the expected performance had the EMS been functional for the second test run.

The results for the simulation of the EMS reveal that the energy constraint was exceeded by 298%. This confirms that the SS2 solver is not adequate for the task.

Hence, the first design objective has not been achieved.

It is recommended that the electronics be rebuilt, an alternative means of solving the MTP is found and that the TSPC is retained for further development of the Solar esCape EMS.

To conclude, the optimal trajectory for a given set of conditions has still not been found. This is largely a numerical method issue. A robust numerical method that can handle non-linear problems is needed. As such, the question of the best approach to controlling the velocity of a solar cycle is unanswered.

The alternative approaches to designing a controller for the EMS need to be explored, while retaining the TSPC, as it is able to compensate for uncertainty.

The field tests suffered from electronic failures. The simulation results suggest that the EMS would not perform as expected had it functioned.

The design objective was not reached in this implementation of the EMS. Therefore, the objective of this dissertation was only partly met.

# Nomenclature

## Symbols

| | |
|---|---|
| $(\,)_i, (\,)_j$ | Indexing operators |
| $(\,)_{t_f}, [\,]_{t_f}$ | Term evaluated at the final time $t_f$. |
| $(\,)^T$ | Transpose operator |
| $a$ | Acceleration |
| $A$ | Area |
| $C_0, C_1, C_2$ | First, second and third coefficients of rolling resistance. |
| $C_d$ | Drag coefficient |
| $C_d A$ | Drag area (Product of drag coefficient and area) |
| $\delta(\,)$ | Variation operator |
| $D$ | Aerodynamic drag |
| $E_{batt}$ | Battery energy expended |
| $\dot{E}_{batt}$ | Time derivative of $E_{batt}$ |
| $f$ | Constraint equations |
| $f_u$ | Partial derivative of constraint equations w.r.t. the control vector $u$. |
| $f_x$ | Partial derivative of constraint equations w.r.t. the state vector $x$. |
| $G$ | Resistance due to the road gradient. |
| $H$ | Hamiltonian matrix |
| $H_u$ | Partial derivative of the Hamiltonian w.r.t. $u$. |
| $H_x$ | Partial derivative of the Hamiltonian w.r.t. $x$. |
| $J$ | Performance index formed by adjoining a given performance index, $L$, and the constraint equations $f$. |
| $\delta J$ | Variation of $J$ |
| $K$ | Gain term |
| $L$ | A given performance index. |
| $\lambda$ | Vector of Lagrange multipliers |
| $\dot{\lambda}$ | Time derivative of Lagrange vector |
| $\lambda^T$ | Transpose of Lagrange vector |
| $\lambda_E$ | Partial derivative of Lagrange vector w.r.t. $E_{batt}$. |
| $\lambda_s$ | Partial derivative of Lagrange vector w.r.t. $s$. |
| $\lambda_v$ | Partial derivative of Lagrange vector w.r.t. $v$. |
| $P_{aero}$ | Power required to overcome aerodynamic load at a given velocity |
| $P_{ava}$ | Total power available |
| $P_{batt}$ | Power drawn from battery |

| | |
|---|---|
| $P_{cyc}$ | Power delivered by cyclist |
| $P_{req}$ | Power required to maintain a given velocity, includes $P_{aero}$ |
| $P_{rolling}$ | Power required to overcome rolling resistance at a given velocity. |
| $P_{sol}$ | Power delivered by solar array |
| $\varphi$ | The evaluation of the performance index, $J$, at the final time $t_f$. |
| $\varphi_x$ | Partial derivative of $\varphi$ w.r.t. the state vector $x$. |
| $m$ | Mass |
| $M$ | Optimality condition |
| $\rho$ | Density of air |
| $R$ | Rolling resistance |
| $s$ | Distance travelled |
| $\dot{s}$ | Time derivative of distance travelled |
| $t_f$ | Final time |
| $t_0$ | Initial time |
| $u$ | Control vector |
| $\delta u$ | Variation of $u$ |
| $v$ | Velocity |
| $\dot{v}$ | Time derivative of velocity |
| $x$ | System state vector |
| $\delta x$ | Variation of $x$ |

**Acronyms**

| | |
|---|---|
| BVP | Boundary value problem |
| EMS | Energy management system |
| ISR | Interrupt service routine |
| IVP | Initial value problem |
| MPPT | Maximum power point tracker |
| MPT | Minimum time problem |
| SCI | Serial communications interface |
| SS2 | Simple Solver 2 |
| TSPC | Two stage predictive controller |
| WSC | World Solar Challenge |
| WSCC | World Solar Cycle Challenge |
| UCT | University of Cape Town |

# Table of contents

# Table of Figures

# Table of Tables

# 1 Introduction

The accepted term, within the solar powered vehicle racing fraternity, for a solar powered racing cycle is a solar cycle. The title is worded to avoid confusion with solar cycles in the astronomical sense. The term solar cycle is used in this dissertation with specific reference to the racing vehicles.

This dissertation covers the development of the energy management system (EMS) for the Solar esCape solar cycle. The emphasis is on the efficient utilisation of the available energy in terms of race strategy, as opposed to the efficient extraction of energy from available sources.

## *1.1 Objective of the dissertation*

The objective is to have a functional prototype system running in the prototype of the Solar esCape solar cycle.

## *1.2 Scope of the dissertation*

The dissertation focuses on the development of the EMS from an optimal control theory perspective.

The physical implementation of the EMS is carried out using the simplest and quickest means available.

For the software component of the implementation, neither the MATLAB code nor the DSP's C code has been optimised.

## *1.3 Structure of Dissertation*

The dissertation is structured in a graduated manner. The general context is presented first, and the discussion progresses to the more specific details of the topic.

**Section 1.** Introduction to the dissertation.

**Section 2.** Introduces solar cycles and solar cycle racing, and places the current work in context within the solar racing sphere.

**Section 3.** Presents an analysis of the factors governing the performance of solar cycles. It covers the dynamics of solar cycles and solar cycle racing.

**Section 4.** Discusses requirements of energy management for solar cycle racing.

**Section 5.** Places the dissertation in academic context by discussing the state-of-the-art in solar racing EMS and reviewing available literature.

**Section 6.** Discusses the philosophy behind the development of the EMS for Solar esCape.

**Section 7.** Is an introduction to the mathematics of optimal control theory, concentrating on those aspects relevant to the current problem. This serves to aid the reader unfamiliar with the field, and to place the specific problem presented by solar cycles in context within the field of optimal control theory. Included is a discussion of the numerical methods available for solving the type of problem under investigation.

**Section 8.** Applies the optimal control theory presented in section 7 to the energy management problem. A number of simulations were performed and the results are presented here. The development is continued to the design of an optimal controller. This section comprises the bulk of the theoretical work performed.

**Section 9.** Discusses the development of the prototype EMS in a prototype of Solar esCape. The prototype EMS incorporates the optimal controller designed in section 8. The section discusses the hardware and software developed.

**Section 10.** Reports on the results of the field trials conducted to test the prototype EMS. Included is a set of simulated results to compare the actual performance of the prototype EMS to the performance of the optimal controller.

**Section 11.** Presents some recommendations for further work.

**Section 12.** Presents the conclusions reached during the development of the prototype EMS.

# 2 Solar Cycle Racing

## 2.1 Background

Solar racing cycles are hybrid vehicles. Driven by human effort and electrical power derived from onboard batteries and solar panels.

Currently two international events are held, the World Solar Cycle Challenge in Australia, and the World Solar Rallye in Japan. The former is run on open highways, while the latter is run on a closed circuit track.

## 2.2 World Solar Cycle Challenge

The World Solar Cycle Challenge ® (WSCC) is held biannually through the Outback of Australia. The full text of the 2001 WSCC regulations is given in Appendix A.

The WSCC is split into several stages, with 200 – 300 km being covered each day. Each team has between two and six cyclists, who cycle in shifts of up to two hours.

Entrants in the WSCC are diverse, ranging from school teams, through university teams, to private teams and include many nationalities.

Each team must have a support vehicle travelling 10 to 50 metres behind their solar cycle while on the road. For Class C solar cycles, a voice radio link between the support vehicle and solar cycle is mandatory.


**Figure 1 Map of WSCC route.**

Figure 1 shows the route of the 1999 WSCC from Alice Springs to Adelaide.

### 2.2.1 History of WSCC

The WSCC was initiated in 1996 as an entry-level or "junior" competition to the World Solar Challenge (WSC) for solar cars. The overriding motive was to reduce the costs of participation.

The event was first run on a 3000 km West to East route. The race was won by the Zero To Darwin Project team with an average speed of 54 kph.

The WSCC was run again in 1997.

The organisers then decided to run the event in parallel to the WSC. Hence, the next event was held in 1999 and the most recent one in 2001. The route starts in Alice Springs and runs south to Adelaide.

The size and composition of the field has varied considerably. The 1999 event was the most popular with 21 vehicles on the start line. The 2001 event saw the field reduced by half.

The technical regulations governing the vehicles are constantly changing. Table 1 summarises the average speeds of the top three cycles for each event, except the 1997 event for which no results are available.

**Table 1 Top Three Solar Cycles per event.**

| WSCC 1996 | |
|---|---|
| Zero To Darwin Project | 54.9 kph |
| AeroVironment | 45.7 kph |
| Sunstrike | 30.4 kph |
| WSCC 1999 | |
| Reflex Southern Alliance | 44.3 kph |
| Spirit of Business | 41.1 kph |
| University of Southampton | 40.4 kph |
| WSCC 2001 | |
| Eastern Fleurieu School | 42.5 kph |
| University Tenega Nasional | 41.5 kph |
| Prince Alfred College | 41.1 kph |



What is apparent from the table is a convergence towards the 45 kph mark.

The Zero To Darwin Project team had a number of years experience in running solar cars. Hence their solar cycle had a drag area, $C_d A$, of 0.0689 and a solar array of 20% efficiency. Section 3.1.1 analyses the performance of solar cycles in more detail.

### 2.2.2 South African Solar Cycles

Only two local attempts are known, both linked to the University of Cape Town (UCT). Only one has competed in the WSCC.




**Figure 2 WestCape Solar Cycle**

### a. WestCape 1999

UCT fielded an entry in the 1999 event under the name WestCape.

#### *i. Design Philosophy*

The key motivation factor for the 1999 team was participation. On the technical side, no clear philosophy held sway.

#### *ii. Performance Expectations*

The cycle was designed to run at 45 kph, with the expectation that the team would average 40 kph.

#### *iii. Results*

The cycle and the team performed remarkably well in the given circumstances. Excluding time penalties for being late at the end of the fifth day, the team maintained an average speed of 36 kph. The team was placed 8$^{th}$ overall and 4$^{th}$ in the budget aerodynamic class.

#### *iv. Experience Gained*

The team learnt a great deal during the event. Some of the points relevant to this work are listed below:

- Wind (esp. cross winds) and its variability need greater consideration, from an aerodynamic and a control perspective.

- The maximum power point tracker (MPPT) for the solar array needs to cope with rapid changes in panel coverage.
- The cyclist needs as simple an instrument panel as possible.
- The cyclist should only have control over the battery output as an emergency measure.
- Braking is only at defined stops and in emergencies. Hence, regenerative braking is not worth considering.
- Proper data telemetry is essential for determining race strategy.
- Poor strategy loses time and causes time penalties.
- All systems should be as reliable and as robust as possible.
- Proper battery condition monitoring and recharging systems are essential.

### b. Solar esCape 2001

It was through participating in the design and development of the UCT cycle, and cycling it in the 1999 WSCC, that the author was inspired to attempt to develop a record-breaking cycle for the 2001 WSCC. Unfortunately, a lack of resources hampered the development process, and ultimately participation in the 2001 WSCC.

Figure 3 is an illustration of Solar esCape. It is derived from the CAD model.



**Figure 3 Solar esCape**

#### *i. Design Philosophy*

Solar esCape has been designed with one clear objective: to set a new average speed record for the WSCC.

This required a thorough analysis of the performance dynamics of solar cycles. The results of the analysis were used to focus development effort on the most sensitive areas and push for maximum effectiveness.

#### *ii. Performance Requirements*

To meet the objective the cycle needs to maintain an average speed in excess of 55 kph. Hence, the design speed should be in the range of 65 to 70 kph.

## 2.3 Engineering and Technology contribution

Participation in the WSCC presents the team with a number of challenges. One must complete the engineering design cycle within a short time frame. By starting with a competitive analysis, one establishes a set of relative performance requirements. By performing a detailed engineering analysis of the performance of a solar cycle, one can determine which aspect of the solar cycle will give the greatest performance return for engineering effort expended.

Solar cycles are closer to the ideal of solar powered transport than solar cars in terms of performance, cost and size. Their development highlights what is feasible and what areas need further development.

# 3 Analysis of Solar Cycle Performance

Three important areas affect the performance of solar cycles. They are the dynamic performance of the solar cycle, the race strategy and the sustained power output of the cyclists.

The first two are discussed in this section and the third is not considered.

## *3.1 Dynamic performance*

The dynamic performance of a solar cycle is influenced by a large number of parameters.

Those of concern to the engineer are ranked below:

1. Aerodynamic efficiency
2. Rolling resistance
3. Electrical efficiency

Other parameters affecting the performance are the gradient, cyclist changes, wind, cloud cover and temperature. While some can be anticipated, others are unpredictable.

### 3.1.1 Vehicle dynamics

Drawing from particle mechanics, the acceleration of the solar cycle, at any particular point in time, is determined by the excess power available, at that point. This is:

$$\dot{v} = a = \frac{P_{ava} - P_{req}}{mv} \qquad (3.1)$$

The available power, $P_{ava}$, comprises the power drawn from the battery, the converted solar power and the efforts of the cyclist.

The power required, $P_{req}$, is that to overcome aerodynamic drag, rolling resistance and the road gradient.

#### a. Aerodynamics

The aerodynamic drag seen by a vehicle is entirely due to the shape, size and configuration of the vehicle. This is best described by the drag area $C_d A$, the product of the drag coefficient of the vehicle and the area relative to which the coefficient is measured. The drag area can be determined using computational fluid dynamics (CFD) or empirically via wind tunnel tests or coast-down tests.

The drag area of Solar esCape is expected to be about 0.15165 square metres. Further discussion and a comparison of various vehicles is covered in Appendix B.

The power to overcome aerodynamic drag is determined using the formula:

$$P_{aero} = \tfrac{1}{2}\rho C_d A v^3, \quad \rho \approx 1.2 \qquad (3.2)$$

Where $\rho$ is the air density, $C_d A$ the drag area and $v$ the velocity.

#### b. Rolling Resistance

The rolling resistance of a solar cycle is dependant on the following parameters:

- The tyre, i.e. its profile, tread pattern, rubber composition and rubber thickness.

- The tyre loading, i.e. the portion of the loaded solar cycle mass carried by each tyre.
- The tyre inflation pressure
- The tyre diameter
- The suspension geometry, i.e. the camber and toe of each wheel.
- The quality of the road surface.
- The camber of the road.
- The temperature of the road and of the tyre.

The general form of the power required to overcome rolling resistance is:

$$P_{rolling} = mv(C_0 + C_1 v + C_2 v^2)$$

This equation assumes the suspension is set-up for zero caster, zero camber and zero toe. The best method to determine the rolling resistance is to fit the solar cycle in racing trim and perform a number of low speed tests on rough asphalt roads. This will be a close approximation to race conditions.

### c. Electrical Efficiency

The amount of solar power converted into electrical power is dependant on the following:

- Conversion efficiency of the solar cells.
- Placement and orientation of the solar cells on the solar cycle body.
- Area of solar array on the solar cycle.
- Temperature of the solar cells.
- Efficiency and effectiveness of the maximum power point tracker and DC-DC converter (MPPT).
- Time of day.
- Cloud cover.

Other components that affect the overall electrical efficiency of the solar cycle are:

- The battery DC-DC converter.
- The electric motor driving the solar cycle.
- The battery discharge characteristics.

For the purposes of this project, only the solar power was modelled. The following assumptions were made in the model:

- The array is flat and parallel to the ground.
- The array is built from a single type and model of solar cell.
- The array has a single MPPT of perfect efficiency.
- The solar cell performance is independent of temperature.

The motor and converter are taken as ideal.

## 3.2 Race strategy

Several key variables have been identified. The key variables can be split into two categories, environmental variables – which cannot be controlled, and the strategic variables – which the team can control.

The discussion of strategy here, and in the rest of this dissertation, assumes that the race route is the official WSCC route.

### 3.2.1 Environmental Variables

These variables describe the state of the environment. They only can be measured or estimated at the time of test.

#### a. Traffic State

##### *i. Normal automotive traffic*

Normal traffic is not a serious problem for most of the route. In general, the drivers are courteous and interested in the event.

##### *ii. Road Trains*

Road train is the Australian term for an articulated truck and trailer combination comprising of a heavy truck and two to five trailers. The length of a road train can be up to 80 metres, while they travel at an average speed of 80 kph. The greatest problem with road trains is their length and speed. At low speeds, the bow wave will cause a solar cycle to swing towards the road train, on approach from the rear, and then swing away from the road train when alongside. The low-pressure region behind the road train will also induce a similar set of yaw moments. The behaviour is dependant on the location of the aerodynamic centre of pressure of the solar cycle; the example above assumes that the centre of pressure is behind the centre of mass.

To minimise tyre scrub and unbalancing the solar cycle, the cyclist should steer with the motion of the solar cycle.

#### b. Road State

The prime objective is to maintain the as high an average speed as possible using the least amount of power, especially battery power.

##### *i. Flat road*

A flat, level road is the ideal steady state scenario. The velocity is kept constant.

##### *ii. Up Hill gradient*

An up hill road requires a greater use of energy to maintain the same speed. To reduce the amount of energy required, the solar cycle's velocity (momentum) can be increased before the hill and allowed to drop off as the solar cycle climbs the hill.

##### *iii. Down Hill gradient*

Three options are available on a downhill. The choice depends on the other environmental variables and the radii of any bends ahead.

The first option is to coast down the hill. This is best when the road is suitable for high speeds. When the required velocity is exceeded, no further power is fed to the motor.

Should the road not be amenable to high speeds, a safe velocity is maintained, coasting when required.

The second option is to use regenerative braking to maintain a steady velocity down the hill and recoup some energy.

The third option is to maintain a steady power input to the motor and gain some time.

#### *iv. Cattle Grids*

There are 84 cattle grids along the route. Most of these grids are 2m wide and consist of 9 pieces of railway track with 150mm gaps between them.

### c. Weather Conditions

The weather can be described by three states, namely, rain, cloud and wind. These states are interdependent, which complicates strategic decisions.

#### *i. Rain Conditions*

Solar esCape should not be susceptible to the ingress of water. The major effect of rain is to alter the handling of the solar cycle and the visibility for the cyclist. Rain reduces the coefficient of friction from 1.0 to about 0.3 – 0.5 depending on the road surface.

Heavy rain gives rise to puddles of various sizes. These present an aquaplaning hazard. Visibility may be reduced. The velocity should be sufficiently low so that only the largest puddles present the possibility of aquaplaning.

When it rains, braking distances need to be lengthened, and cornering speeds reduced. The available power will be slightly reduced depending on the cloud cover. Any oil on the road will form slick patches that are often not visible.

#### *ii. Cloud Coverage*

If cloud coverage appear to reduce along the route, it could be better to flee the cloud cover as fast as possible, then relax when in good sun.

Intermittent clouds provide the best morning and evening insolation levels. The sunlight is also more diffused throughout the day, to the benefit of curved solar panels. Sometimes the sun is clear, surrounded by scattered cloud. In this case, the insolation is greater than when the sky is clear, as the scattered clouds add a component of diffused sunlight.

Overcast conditions, with little or no direct sunlight, will reduce the available solar energy. The sunlight is also diffused.

A clear sky will provide the greatest insolation at noon, with a steeper rise and fall in the morning and afternoon. The noon peak is to the greatest benefit of flat horizontal panels.

#### *iii. Prevailing Wind*

Wind is a menace, particularly cross winds and small whirlwinds. The best defence is to design an aerodynamically stable vehicle. In general, a wind could be regarded as a crosswind when its major effect on the solar cycle is to increase the steering effort.

A constant headwind will retard the solar cycle. Some means of compensating for the increased load will be needed.

A tailwind is a boon.

Crosswinds increase tyre scrub by adding a side load. The cyclist must compensate with increased steering effort, which increases the rates of both physical and mental fatigue. Special caution needs to be exercised when passing large or fast vehicles, as the yaw moment exerted on the solar cycle will be reduced and then increased, or vice versa depending on the wind direction.

Road cuttings are another hazard, as the blast of the crosswind suddenly hits the vehicle as the vehicle leaves the cutting.

In certain conditions, the crosswind causes sufficient lateral lift to lift the windward wheel and accelerate the solar cycle. Steering control is difficult and braking tends to add to the tipping moment. Good aerodynamic design will address this problem.

A gusting wind will tax the cyclist's concentration and require more frequent cyclist changes.

#### d. Time of Day

As mentioned with regard to the cloud conditions, the time of day determines the solar flux incident on the panels, hence the solar power available. This is compounded by the shape and angle of the solar panel on the solar cycle.

##### *i. Mid-morning to Mid-afternoon*

This is the period of the greatest daily solar flux. Conversely, the temperature is higher, leading to quicker cyclist fatigue.

##### *ii. Late afternoon*

The sun is low and oblique. Hence, the available solar power is low. This case is only a consideration on long stages.

### 3.2.2 Strategic Variables

A few strategic variables, e.g. cyclist changes, are independent of the opposition, although they can be used to gain competitive advantage. Frequently it is preferable to gain a clear competitive advantage over an opponent and lose some time.

#### a. Overtaking manoeuvres

##### *i. Overtaking Others*

This is best achieved on a downhill, while increasing the vehicle speed. Slipstreaming the opponent can occur on the uphill. Long hills will enable the solar cycle to overtake the opponent's solar cycle and support vehicle simultaneously.

Psychological factors can alter the form of the overtaking manoeuvre.

##### *ii. Being Overtaken*

If one is sporting, one will pull to the side of the road, and ease up a fraction. If one were competitive, one would allow a large gap to form between ones solar cycle and support vehicle, forcing the opponent to overtake the two vehicles individually. Another option is to invite the opponent's solar cycle to overtake on an uphill.

#### b. Frequency of Cyclist Changes

The frequency of the cyclist changes allows one means of controlling the cyclist power output. Some cyclists are better at short duration sprints, while others are better suited to long, endurance rides.

The maximum cycling time is limited to two hours, so an endurance cyclist could be used during the midday shift, maximising the solar power used. A sprint cyclist could be used to catch and overtake another team, out run bad weather or establish a lead in the morning.

#### c. Psychological Tactics

Cycling lore is full of tactics that can be applied to solar cycle racing. Three examples are included here.

- Start on the back of the grid on the first day; lull the teams ahead into a false sense of security. Once out of town the objective is to sprint past the field as

quickly as possible. This should dismay the leading teams, leading them to attempt to chase your team and abandon their planned strategy.

- When gaining on a team, do so slowly, conserving battery and cyclist energy. Then overtake decisively and sprint into the distance. Typically, the opposing cyclist will attempt to chase and be exhausted earlier than the team had planned.
- If a team is reasonably close behind, but not a direct threat yet, allow them to slowly close the gap. Then stay just ahead using the cyclist and conserving the battery energy. After about half an hour, the chasing team should stop to change cyclists, putting in one of their sprinters. Ones team must also stop and change cyclists. If passed, perform the above manoeuvre, else if still ahead, sprint into the distance. Usually the opponent's sprinter will chase hard, requiring the opposing team to change cyclists again within an hour.

### d. Stopping Points

The best position to make a cyclist change is when the solar cycle's velocity is already low. Thus, the effort needed to regain race speed is a minimum. The duration should be kept as short as possible.

# 4 Energy Management Systems for Solar Cycles

An EMS for a solar cycle has two objectives; firstly, it has to use the available energy to control the velocity of the solar cycle in order to complete the day's stage in the minimum time possible. Secondly, it has to aid the team in making tactical decisions.

## *4.1 Energy Sources*

A solar cycle, being a hybrid, has three distinct sources of energy, viz. the physical effort of the cyclist, a chemical battery and an array of solar cells. The single largest source is the cyclist. The next largest is either the solar array or the battery depending on the aerodynamic compromises made. See Appendix B for a more detailed discussion. In the case of WestCape, the solar array produced more energy than the battery, while in Solar esCape the battery is expected to provide more energy than the solar array.

Nobody has yet devised a reasonable system for regulating a cyclist's power output. At least, not within solar cycle racing.

## *4.2 Role of Energy Management in Race Strategy*

Race strategy is vital to the success of a team. A good strategy can extract the best performance from an average team and solar cycle, whereas a single bad decision can destroy the hopes of the best team with the best solar cycle. Thus, everything that contributes to the strategic decisions needs to be accurate and reliable.[i]

The EMS has to respond to, and accommodate, a number of different parameters. It should also be able to predict the solar cycle's performance (i.e. time to complete the stage) at any point during the stage. This will allow the team to determine how to minimise the impact of weather conditions, gain competitive advantage through exploiting weather conditions and tactics, and the cost of gaining that competitive advantage.

## *4.3 Management of Solar Energy*

The most common approach in solar cycling racing is simply to attach the solar array directly to the battery and allow it to charge the battery if conditions are favourable. The more sophisticated approach is to use a maximum power point tracker (MPPT) to extract the greatest amount of power possible from the solar array in any given set of weather conditions. The only real choice is whether one uses excess solar energy (during a cyclist change, or coasting down a long, fast hill) to charge the battery or simply loses the energy. Given a series of five two minute stops and an exceptional solar panel of 352 W, one would have an excess of 58 Wh of energy, approximately 5% of the battery total. In the case of WestCape panel (224 W) the excess would be 37 Wh, or 3.7%. This problem warrants further investigation, but is beyond the scope of this dissertation.

The figure below illustrates the typical electrical system for a solar cycle.



[i] Aurora lost their chance of winning the 2001 WSC due to a battery current reading that under read by 2%.

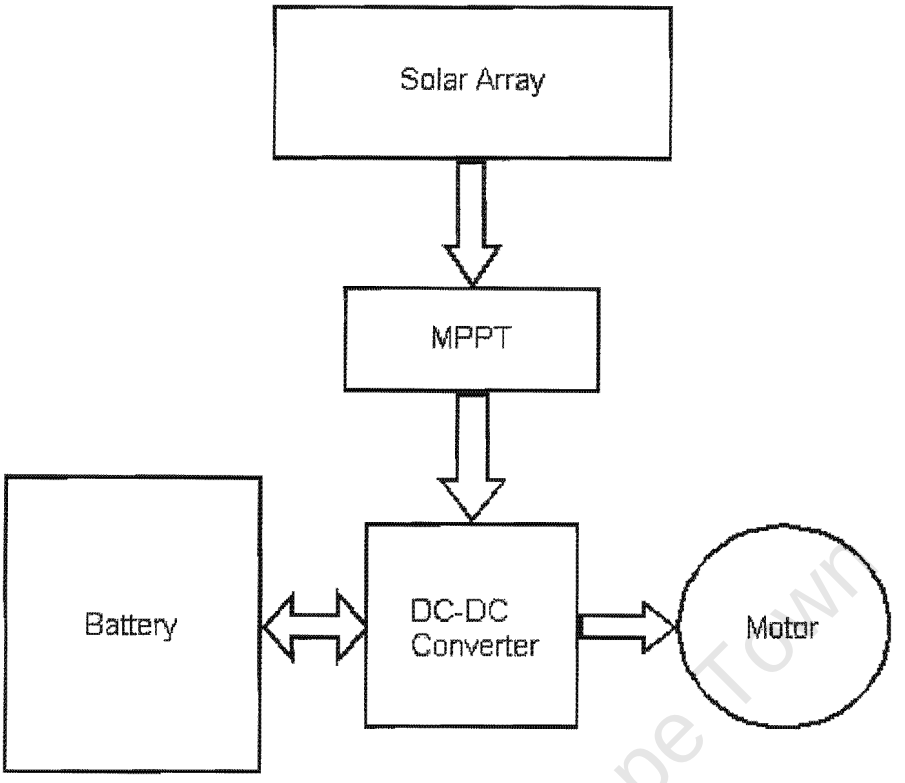



**Figure 4 Typical Solar Cycle Electrical System**

Some teams design and build their own MPPT devices, with mixed results. Others purchase proven solar racing MPPT devices from AERL, who have been supplying WSC teams for more than a decade, or the Engineering School of Biel.

The output of a MPPT can be connected either to the battery or to the system bus, the latter being more efficient.

So, the only source amenable to control is the battery.

## *4.4 Management of Battery Energy*

The battery is usually controlled with a DC-DC converter. This in turn controls either the motor voltage or the current fed to the motor. Frequently the voltage or current setpoint is under the control of the cyclist. More sophisticated solar cycles have a velocity feedback loop where the cyclist sets a velocity setpoint.

The battery state is estimated with the use of ampere-hour meters or simply by measuring the voltage under load. Both approaches need to be tested to determine the behaviour of a particular battery in racing conditions. For the purposes of the prototype system, it is assumed that the battery discharge profile is perfectly flat.

# 5 State of the Art of Solar Cycle EMS

## 5.1 Prior Art

### 5.1.1 Solar Car based Art

All the published works in the field of solar racing deal with solar cars. The first comprehensive work was published by Storey, Schinckel and Kyle[1], it covered all the races from 1987 to 1993. The work covers the technical aspects of the various cars and a discussion of the 1993 event. Included is a datasheet and picture of all the vehicles in the events to the date of publication.

This was followed by Roche *et al*[2], which included a discussion of the solar cycles in the 1996 WSCC. A passing comparison of the control strategies of solar cars and solar cycles is made. In their opinion, the systems for solar cycles are more complex than those for solar cars[3] and have not been not fully explored.

Tamai's text forms an aerodynamics design manual for solar cars, with extensive empirical data tables and performance estimation procedures[4]. A short discussion of how the aerodynamics affects the design of a solar cycle is given in Appendix B.

Late into the project, Pudney's PhD dissertation on the optimal control of solar cars became available[5].

### 5.1.2 WestCape 1999

The design of WestCape drew from the work published by Roche *et al*.

Using this work and deriving the basic vehicle dynamics, the author wrote a number of simulations covering a vehicle's performance in a given set of circumstances. The simulations confirmed the efficiency hierarchy listed earlier.

During the development of WestCape, the question arose on what strategy to follow. Simulations demonstrated that a constant velocity strategy was superior to a constant power strategy and a "squirt and coast" strategy. Since this is a near optimum strategy for solar cars[6,7], and the most popular, leant confidence to the decision to adopt it. However, time constraints prevented the implementation of the speed control loop.

While Ray Hemer provided the 1999 WSCC route data to teams prior to the event[8], it was not incorporated into the simulations at that stage.

## 5.2 Current Work

Prompted by the mention, by Roche *et al*[9], of General Motor's use of optimal control theory to determine the optimal speed of a solar car in a given set of conditions, the simulation was extended. The first extension was to run the simulation over a given route. This allowed the route data from the 1999 WSCC to be incorporated, yielding results that are more tangible. By wrapping the simulation in a minimising loop, the optimum constant velocity giving the minimum time to complete the stage was found.

To introduce some uncertainty the option to include random cloud cover was included. This feature demonstrated the shortcomings in the constant velocity strategy. It is not feasible to set the solar cycle velocity at a constant value and be able to predict when the battery will be drained. If the battery is drained some distance prior to the end of the stage, it is not possible to maintain the required constant speed and in all cases, the time to complete the stage is increased.

The question then arose: What is the best approach to control the velocity of a solar cycle?

# 6 Energy Management System for Solar esCape

## 6.1 Design Objectives

The objective of the Solar esCape EMS is to perform the role as described in section 4. In doing so, it would answer the question posed in 5.2.

For the prototype, only the gradient and the uncertainty in the aerodynamic power losses due to wind are considered. The gradient is readily measured beforehand and the aerodynamic losses can be estimated from the apparent wind seen by the solar cycle. The wind estimate is the dominant source of noise in the system.

## 6.2 Considered Options

The areas examined were:

- Classifier Systems: These systems are rule based, with hard decision points determining the actuation of the rules. Essentially an extended finite state machine.
- Learning Classifier Systems: These systems utilise a genetic algorithm to generate rules and adapt to a changing environment.
- Fuzzy Logic: A classifier system with soft rule actuation and a 'blending' of output results.
- Neural Networks: An extension of fuzzy logic with 'blended' inputs resulting in a 'blended' output.

The author has one simple objection to the use of the above options: none of them inherently considers the dynamics of the system being controlled.

A chemical engineer suggested that the problem could possibly be solved using model predictive control (MPC). Subsequent discussion yielded that MPC was a sub-field of optimal control.

After studying the textbooks on optimal control, a particular special case was discovered, the Minimum Time Problem. A detailed discussion of the problem is given in section 7.2.

A later literature review netted no published work in the field of solar cycle racing. By broadening the parameters, some of the work on solar cars was cited in a number of general articles, but no papers.

Purely perchance, Peter Pudney mentioned, on the WSC mailing list[10], that his PhD thesis was available on his web page. This was just prior to the publication, in *IEEE Spectrum,* of the article covering the 2001 WSC, in which Pudney's PhD is cited[11].

Pudney bases his work on optimal control. He does concede that alternative methods for searching for an optimal strategy can be used, e.g. genetic algorithms, but they do not provide any insight into the strategy.

## 6.3 Selected Approach

On the strength of the existence of the special case, and the inherent ability of the optimal control approach to handle dynamic systems, it was decided to use optimal control theory to attempt to answer the question posed in section 5.2.

# 7 Optimal Control Theory

This section deals with the necessary mathematics of optimal control. A brief introduction is given, followed by a more detailed description of the Minimum Time Problem.

A discussion of the various numerical methods available to solve optimal control problems is then presented.

## *7.1 Fundamentals of Optimal Control*

The description of optimal control in this section draws heavily on the work by Bryson and Ho[12]. The objective of this section is to introduce the reader to the field of optimal control and the notation used in this text. Any errors are by mishap of the author and readers are referred to the original source for clarification and further reading.

Dynamic systems are of great interest to engineers, as any device that interacts with its environment is a dynamic system. As such, there is great scope for optimising that interaction of the device.

Our discussion examines continuous dynamic systems and their optimisation.

Consider the system described by the non-linear differential equations:

$$\begin{aligned} &\dot{x} = f[x(t),u(t),t] \\ &x(t_0) \quad given, \quad t_0 \le t \le t_f \end{aligned} \tag{7.1}$$

Where $x(t)$, $n$-dimensional, is the state vector of the system and $u(t)$, $m$-dimensional is a control or decision vector. The equations, $f$, are referred to as the system equations or constraint equations.

Consider the scalar performance index of the form:

$$J = \varphi[x(t_f),t_f] + \int_{t_0}^{t_f} L[x(t),u(t),t]dt \tag{7.2}$$

The problem is to find $u(t)$ to minimise or maximise $J$. We adjoin the constraints (7.1) to $J$ with Lagrange multiplier functions $\lambda(t)$:

$$J = \varphi[x(t_f),t_f] + \int_{t_0}^{t_f} \left[L[x(t),u(t),t] + \lambda^T(t)\{f[x(t),u(t),t] - \dot{x}(t)\}\right]dt \tag{7.3}$$

We define the Hamiltonian, $H$ as:

$$H[x(t),u(t),\lambda(t),t] = L[x(t),u(t),t] + \lambda^T(t) f[x(t),u(t),t] \tag{7.4}$$

Also integrating the final term on the RHS of (7.3) we have:

$$J = \varphi[x(t_f),t_f] - \lambda^T(t_f)x(t_f) + \lambda^T(t_0)x(t_0) + \int_{t_0}^{t_f} \left[H[x(t),u(t),t] + \dot{\lambda}^T(t)x(t)\right]dt \tag{7.5}$$

We now consider the *variation* of $J$ due to the variation in the control vector $u(t)$ for fixed times $t_0$ and $t_f$.

$$\delta J = \left[\left(\varphi_x - \lambda^T\right)\delta x\right]_{t=t_f} + \left[\lambda^T \delta x\right]_{t=t_0} + \int_{t_0}^{t_f}\left[\left(H_x + \dot{\lambda}^T\right)\delta x + H_u \delta u\right]dt \qquad (7.6)$$

We choose the Lagrange multiplier functions $\lambda(t)$ to cause the coefficients of $\delta x(t)$ to vanish:

$$\dot{\lambda}^T = -H_x = -L_x - \lambda^T f_x, \qquad (7.7)$$

with boundary conditions

$$\lambda^T(t_f) = \varphi_x \qquad (7.8)$$

Equation (7.9) then reduces to

$$\delta J = \left[\lambda^T \delta x\right]_{t_0} + \int_{t_0}^{t_f} H_u \delta u dt \qquad (7.10)$$

For a stationary point, $\delta J$ must be zero for arbitrary $\delta u(t)$. This can only occur if

$$H_u = 0, \quad t_0 \le t \le t_f \qquad (7.11)$$

To determine whether a minimum exists, one expands (7.6) to second order:

$$\delta J = \left[\tfrac{1}{2}\delta x^T \frac{\partial^2 \varphi}{\partial x^2}\delta x\right]_{t_f} + \tfrac{1}{2}\int_{t_0}^{t_f}\left(\delta x^T \frac{\partial^2 H}{\partial x^2}\delta x + \delta u^T \frac{\partial^2 H}{\partial u^2}\delta u + 2\frac{\partial H}{\partial x \partial u}\delta x \delta u\right)dt + \frac{\vartheta(\varepsilon^2)}{\varepsilon^2}$$

$$\Rightarrow \delta J = \left[\tfrac{1}{2}\delta x^T \frac{\partial^2 \varphi}{\partial x^2}\delta x\right]_{t_f} + \tfrac{1}{2}\int_{t_0}^{t_f}\left(\begin{bmatrix}\delta x^T & \delta u^T\end{bmatrix}\begin{bmatrix}\frac{\partial^2 H}{\partial x^2} & \frac{\partial^2 H}{\partial u^2} \\ \frac{\partial H}{\partial x \partial u} & \frac{\partial H}{\partial u \partial x}\end{bmatrix}\begin{bmatrix}\delta x \\ \delta u\end{bmatrix}\right)dt + \frac{\vartheta(\varepsilon^2)}{\varepsilon^2} \qquad (7.12)$$

A minimum exists if the Hessian matrix is greater than or equal to zero.

$$\begin{bmatrix}\frac{\partial^2 H}{\partial x^2} & \frac{\partial^2 H}{\partial u^2} \\ \frac{\partial H}{\partial x \partial u} & \frac{\partial H}{\partial u \partial x}\end{bmatrix} \ge 0 \qquad (7.13)$$

In summary, to find a control vector $u(t)$ that produces a stationary value for the performance index $J$, we are faced with the two-point boundary value problem:

$$
\begin{aligned}
&\textit{Differential equations}: \\
&\dot{x} = f(x,u,t), \\
&\dot{\lambda} = -(f_x)^T \lambda - (L_x)^T, \\
&\textit{where } u(t) \textit{ satisfies} \\
&\frac{\partial H}{\partial u} = (f_u)^T \lambda + (L_u)^T = 0 \qquad (7.14) \\
&\textit{with split boundary conditions}: \\
&x(t_0) \textit{ given}, \\
&\lambda(t_f) = (\varphi_x)^T
\end{aligned}
$$

When the Hessian is positive definite, the value is a minimum.

In the case of problems with some state variables specified at an unspecified terminal time, a number of additional necessary conditions arise. It is convenient to regard $t_f$, the terminal time, as a control parameter to be found in addition to $u(t)$. The optimal choice of $t_f$ must satisfy the following condition:

$$\left(\varphi_t + \lambda^T f + L\right)_{t_f} = 0 \qquad (7.15)$$

The Minimum Time Problem is a special case of this type of problem.

## 7.2 The Minimum Time Problem

The nature of a Minimum Time Problem (MTP) is that the performance index is the time elapsed while the system progresses from its initial state to a specified terminal state. In this case, we have

$$\varphi = 0, \quad L = 1, \qquad (7.16)$$

which implies

$$J = t_f - t_0 \qquad (7.17)$$

This allows the two-point boundary value problem to be stated as:

$$
\begin{aligned}
&\dot{x} = f(x,u,t); \\
&\quad x(t_0) \textit{ is given. } (\textit{If } x_j(t_0) \textit{ is not specified, } \lambda_j(t_0) = 0) \ (n \textit{ initial conditions}) \\
&\qquad\qquad (7.18a) \\
&\dot{\lambda} = -(f_x)^T \lambda; \\
&\quad x_j(t_f) \textit{ specified}, \quad j = 1,\ldots,q; \quad \lambda_j(t_f) = 0, \quad j = q+1,\ldots,n \ (n \textit{ final conditions}) \\
&\qquad\qquad (7.18b)
\end{aligned}
$$

$$0 = f_u^T \lambda \quad (m \text{ optimaility conditions}) \tag{7.18c}$$

$$(\lambda^T f)_{t=t_f} = -1 \tag{7.18d}$$

Hence, there are $2n$ boundary conditions for the $2n$ differential equations, $m$ optimality conditions for the $m$ control variables, $u$, and a transversality condition for the terminal time $t_f$. The unspecified values of $\lambda_j(t_f)$ are part of the solution. Of course, at least one state variable must be specified at $t_0$ and $t_f$ for the problem to make any sense.

## 7.3 Numerical Methods for Solving Non-linear Two-point Boundary Value Problems

The literature presents two approaches for solving two-point boundary value problems, namely shooting methods and finite difference methods[13]. These methods are well established and still being researched to extend their applicability.

Two other methods are also discussed, as their focus is on optimal control problems. These are a gradient method suggested by Bryson and Ho[14], and the method of consistent approximations developed by Schwartz[15].

### 7.3.1 Finite Difference Methods

Finite difference methods are readily programmable and directly incorporate the boundary conditions of the problem. In essence, a difference equation is used to approximate the differential equation in the BVP. The resulting matrices can be solved directly using normal linear algebra, in the linear case. In the non-linear case, some iteration is required to approach the solution.

However, the method also assumes that the final time is known. This is implicit in the handling of the boundary conditions. Again, the method could be extended, with the same penalty.

### 7.3.2 Methods and functions available in MATLAB

MATLAB has a suite of ODE solvers, based on a number of methods. The most commonly used is `ode45`, based on a fourth order Runge-Kutte method. These solvers can be used to solve IVPs of varying stiffness. However, all the solvers require the specification of the final time.

MATLAB has a number of optimisation functions available. Of particular interest is `fmincon`, a minimising solver for constrained problems. When attempting to solve a MTP it returns the error: "No method available." The documentation does not list any description or explanation for the error. It is assumed that the solver was not able to solve for the final time.

It is mentioned in the documentation, that the solver uses a finite difference method[16].

### 7.3.3 Bryson and Ho's Gradient Method

A number of methods are proposed in their text. They specifically deal with problems where the terminal time is unspecified. On initial examination, the most amenable to MTP solving, appear to be the gradient methods. The methods do have a number of disadvantages, i.e. they do not satisfy the optimality conditions, or the boundary conditions. However, the system equations and influence equations are satisfied by these methods. A more complete description is given in Appendix C.

The method failed due to a number of the internal matrices becoming singular during the first iteration. This is independent of the initial conditions. Tweaking the various weighting parameters gave no improvement.

### 7.3.4 Schwartz's Method of Consistent Approximations

Schwartz claims to have developed a new robust method for solving the type of BVP that arises from optimal control problems. The method has been coded as a MATLAB toolbox for MATLAB 5.0, and is now out of date.

Attempting to extract the method from the text of Schwartz's PhD was abandoned, as it is not clearly presented.

The method is based on Euler's method and approximates the continuous problem with a number of discrete problems. It appears to be a shooting method.

### 7.3.5 Shooting Methods

Shooting methods replace the BVP with two initial value problems. By adjusting the initial conditions of one of the initial value problems (IVP), the termination point, or final point, of the two problems can be made to approach that of the BVP.

The method can handle non-linear systems of equations, in an iterative manner. The method used to solve the IVPs can be selected from a large range of methods, with a fourth order Runge-Kutte method being the most common.

The first problem encountered when attempting to solve a MTP is the specification of the termination, or final, conditions. While the state's value is known, the time at which that value is reached is unknown and must be determined as part of the solution.

It may be possible to extend the method to solve for the final time of an IVP, with a given initial condition, in an iterative manner, before progressing to evaluate the degree of "hit or miss" for the solution of the IVP with that particular initial condition. This nested idea would probably increase the computational load by an order of magnitude.

Another approach is to use a substitute one of the states for the time. This approach is explored further.

Faced with the complexity and non-performance of the above methods, effort focussed on using the shooting method, which can use a number of methods to solve the IVPs. Three such methods are investigated.

#### a. A simple, iterative method

This method was written in response to the complexity and sensitivity of the above methods. It focuses on the solution of the BVP as applied to the solar cycle MTP. In essence, the BVP is replaced by an IVP, and the independent variable is changed. That is, while the independent variable in the BVP is time, in the IVP we choose to use a state variable that has two given boundary conditions. In addition, we require another that another state variable also has two given boundary conditions, for use as an error metric.

The method is listed here.

Step 1: Estimate the control trajectory. Be certain to allow sufficient length to accommodate non-optimal solutions during the iterative procedure.

Step 2: Integrate the system and influence equations over the interval defined by the choice of state variable.

Step 3: Determine the difference in the calculated and given end boundary condition. Use this in a proportional-integral-derivative (PID) type control law: $du = -K\dot{x}$

Where $\dot{x}$ is a reduced vector of the state derivatives. $K$ is the gain parameter vector, with values typically between one and ten.

Step 4: Add $du$ to the control trajectory $u(t)$. Repeat from Step 2 until the difference in the end boundary condition is within tolerance.

Step 5: Integrate the influence equations backward.

Step 6: Check whether the optimality condition is met.

$$M = f_u^T \lambda$$

Where M is the deviation from optimality.

Thus the nominal solution satisfies following:

- System equations
- Influence equations
- Boundary conditions

The optimality condition may be met by a particular problem, but is not explicitly met by the method. However, a measure of the deviation from the optimality condition is available. No formal convergence proofs have been attempted.

This is a simple shooting method derived from the solar cycle MTP.

### b. Nelder-Mead Method

The Nelder-Mead method is a direct search method that does not require gradient information. As such, it is suitable for use on non-linear problems. Wood[17] gives an algorithm that is almost identical to the MATLAB function by Kelley[18]. The method has a number of parameters that allow it to be tuned to the application. In this instance, the standard set of parameters was used.

### c. Projected Gradient Method

The projected gradient method is a general case of the steepest descent method and is claimed to be more robust with similar convergence properties. It is a first order gradient method and requires the availability of the gradient of the function being minimised. Schwartz proposes a novel bounded algorithm[19]. Wolfe gives a description of the unconstrained algorithm[20]. Kelley gives a MATLAB function implementation of the unconstrained algorithm[21].

# 8 Application of Optimal Control to the Solar Cycle Racing

With the WSCC run as a series of stages over a fixed stage length, each day's racing can clearly be stated as a minimum time problem. The objective is simply to finish the day's stage as quickly as possible. In fact, the team's performance is measured in time to complete the stage, which echoes the performance index of the MTP.

The discussion starts with the definition of the problem statement, in the mathematical sense.

## *8.1 The Problem Statement*

This section describes the problem statement for a solar cycle for a known route. It is assumed that the length and gradient profile of the route are known.

Three states describe the solar cycle at any point along the route. They are the amount of battery energy that has been expended, the distance travelled and the velocity of the solar cycle. The advantage of this particular choice of states is that the initial conditions are all zero. Only the end condition for the velocity is unknown, as the distance travelled will be the length of the stage or route and the energy expended is the capacity of the battery.

There are three control variables which influence the solar cycle while on route. They are the power drawn from the battery, the converted solar power supplied to the motor and the cyclist's power output. Of these, the solar power and cyclist's power are not actually controls, but noise inputs, whose input is uncertain but can be measured in the case of solar power or predicted in the case of the cyclist's power. Malinowski suggests this treatment[22].

Drawing from our analysis of solar cycle performance in section 3.1, the system or dynamic equations are:

$$\dot{x} = f(x,u,t):$$

$$\Rightarrow \dot{x} = \begin{pmatrix} \dot{E}_{batt} \\ \dot{s} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} P_{batt} \\ v \\ \dfrac{P_{batt} + P_{sol} + P_{cyc} - \left(Dv^3 + Rv^2 + Gv\right)}{mv} \end{pmatrix} \tag{8.1}$$

Where:

- $E_{batt}$ is the battery energy expended and $\dot{E}_{batt}$ is the time derivative.
- $s$ is the distance travelled and $\dot{s}$ is the time derivative.
- $v$ is the velocity and $\dot{v}$ is the time derivative.
- The time derivative of battery energy extended is the power drawn from the battery, $P_{batt}$.
- The time derivative of the distance travelled is the velocity.
- The time derivative of the velocity is the acceleration. See eqn. (3.1).
- $m$ is the mass of the laden solar cycle, i.e. including the cyclist.

To recap, the necessary conditions are:

$$a: \dot{\lambda} = -(f_x)^T \lambda$$

$$where\ f_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & f_{3v} \end{pmatrix}$$

$$, f_{3v} = -\frac{1}{mv^2}\left(P_{batt} + P_{sol} + P_{cyc} - \left(Dv^3 + Rv^2 + Gv\right)\right) + \frac{1}{mv}\left(3Dv^2 + 2Rv + G\right)$$

$$and\ \lambda = \begin{pmatrix} \lambda_E \\ \lambda_s \\ \lambda_v \end{pmatrix}$$

$$b: (f_u)^T \lambda = 0$$

$$where\ f_u = \begin{pmatrix} 1 \\ 0 \\ \frac{1}{mv} \end{pmatrix}$$

$$c: (\lambda^T f)_{t_f} = -1$$

The boundary conditions are listed in the following table:

**Table 2 Boundary conditions for MTP**

| Variable | Initial condition, $t_0 = 0$ | End condition, $t_f$ |
|---|---|---|
| $E_{batt}$ | 0 | Capacity |
| $s$ | 0 | Stage length |
| $v$ | 0 | Unknown |
| $\lambda_E$ | Unknown | Unknown |
| $\lambda_s$ | Unknown | Unknown |
| $\lambda_v$ | Unknown | 0 |



### 8.1.1 Analytical Simplification of Necessary Conditions

It is possible, in this case, to simplify the necessary conditions. This will also simplify the numerical solution process.

Simplifying the influence equations:

$$From\ a:$$

$$\dot{\lambda} = \begin{pmatrix} 0 \\ 0 \\ -\lambda_s - f_{3v}\lambda_v \end{pmatrix} \qquad (8.2)$$

$$\Rightarrow \lambda_E = C_E,\ \lambda_s = C_s,\ \dot{\lambda}_v = -\left(\lambda_s + f_{3v}\lambda_v\right)$$

Where $C_E, C_s$ are constants of integration.

Simplifying the optimality condition:

$$From\ b:$$

$$(f_u)^T \lambda = \begin{pmatrix} 1 & 0 & 1/mv \end{pmatrix} \lambda = \lambda_E + \frac{\lambda_v}{mv}$$

$$\therefore \lambda_E + \frac{\lambda_v}{mv} = 0$$

$$At\ t = t_f : \lambda_E + \frac{\lambda_v(t_f)}{mv(t_f)} = \lambda_E + \frac{0}{mv(t_f)} = \lambda_E \qquad (8.3)$$

$$\Rightarrow \lambda_E = 0$$

$$\therefore \frac{\lambda_v}{mv} = 0$$

$$\Rightarrow \lambda_v = 0$$

Hence, the optimality condition is satisfied for this particular problem.

Simplifying the transversality condition:

$$From\ c:$$

$$\begin{pmatrix} \lambda_E & \lambda_s & \lambda_v \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} 0 & \lambda_s & 0 \end{pmatrix} \begin{pmatrix} P_{batt} \\ v \\ \dfrac{P_{batt} + P_{sol} + P_{cyc} - \left(Dv^3 + Rv^2 + Gv\right)}{mv} \end{pmatrix} = \lambda_s v$$

$$\therefore \left[\lambda_s v\right]_{t_f} = -1$$

$$\Rightarrow \lambda_s v(t_f) = -1$$

$$\Leftrightarrow \lambda_s = \frac{-1}{v(t_f)}$$

(8.4)

Hence, the transversality condition is satisfied.

## 8.1.2 Conclusion

With the necessary conditions being satisfied by the nature of the problem, the problem becomes a far simpler. It is now simply to find the solution to a set of non-linear ordinary differential equations over time, the normal initial value problem (IVP).

## 8.2 The Simulation of Optimal Performance

In the light of the problem statement, the question can be phased as: What is the optimal control trajectory for a solar cycle on a given route?

The control input now considered is the power drawn from the battery.

To provide two reference points for comparing the results of the optimal solutions produced, the two control strategies mentioned in section 4.1 are also simulated. The constant power strategy is the simplest solution to the problem statement. The constant velocity strategy is commonly regarded as the optimal solution, and is relatively easy to implement. In fact, it is the optimal solution for a constant set of inputs only. Pudney bases his analysis on this fact, developing a piece-wise approach to solving the problem[23]. Though he does not present his method, this approach of using discrete approximations echoes that of Schwartz.

All three of the numerical methods discussed in section 7.3.5 were used to solve the MTP. One is a new method; one is a gradient method and one a direct search method. Each method has advantages and disadvantages, which are discussed below.

The solutions were to establish a set of the near-optimal control trajectories for a given route with a defined set of reference conditions. The results were to be used as a benchmark for the controllers developed.

The development work for the entire project was done in MATLAB 6. All the functions, including some of the solvers, were developed from first principles. This dealt with a number of concerns, namely, robustness of solution, convergence of solvers, solution time, determinism and debugging.[ii] A comprehensive set of simulations and comparison of their results is given in Appendix C.

The simulations were performed on a PC with an Athlon 1.33 GHz processor, 256 MB SDRAM and using the Windows 2000 operating system. All calculation times in this section refer to the performance on the above PC.

This section presents five control simulations:

1. A constant battery power simulation
2. A constant velocity simulation
3. An optimal solution using the Simple Method
4. An optimal solution using the Projected Descent Method
5. An optimal solution using the Nelder-Mead Method

In all five cases, the simulations were run with and without an uncertainty component. The uncertainty component is a simple wind model that adds a velocity component to the velocity of the solar cycle. This increases or decreases the apparent air velocity seen by the solar cycle and alters the aerodynamic drag accordingly.



[ii] This practice of writing dedicated solvers and other functions from scratch, often duplicating the built-in MATLAB functions, is followed at Kentron, a Denel subsidiary who design and manufacture guided missiles. The tracking and interception problem of guided missiles falls within the scope of optimal control. Their rational is that debugging their own code was quicker and simpler than debugging the MATLAB solvers.

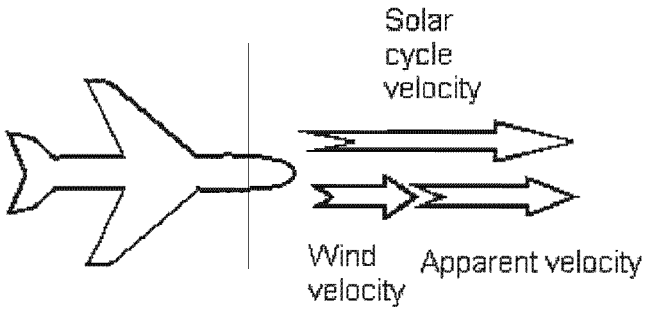



**Figure 5 Wind Model**

The above model is a simplification, as the apparent wind is usually at some angle off the direction of travel of the vehicle and the drag area, $CdA$ , changes, sometimes dramatically.

For prototyping purposes, the simple model is used, as it adds the necessary uncertainty component and is simpler to implement.

The model is set-up as a time series in the same manner as the solar and cyclist models. See Appendix D for a more detailed description of the models. To aid comparison the wind model was run once and the result used in all the simulations, referred to as the known wind.

The simulations used a short route of 56 km derived from the first day of the WSCC. The elevation profile is shown in figure 6. It is varied enough to test the solvers, but simple enough for the eye to follow. The battery capacity was fixed at 168.8 Wh to match the reduced route. A good cyclist was assumed.

During the development of the simple method simulations, the influence of the step size was explored. Using a 5 second step size gave results within 5% of the results with a one second step size. However, when developing the TSPC simulations, the step size was fixed at one second and all the simulations were performed using it.

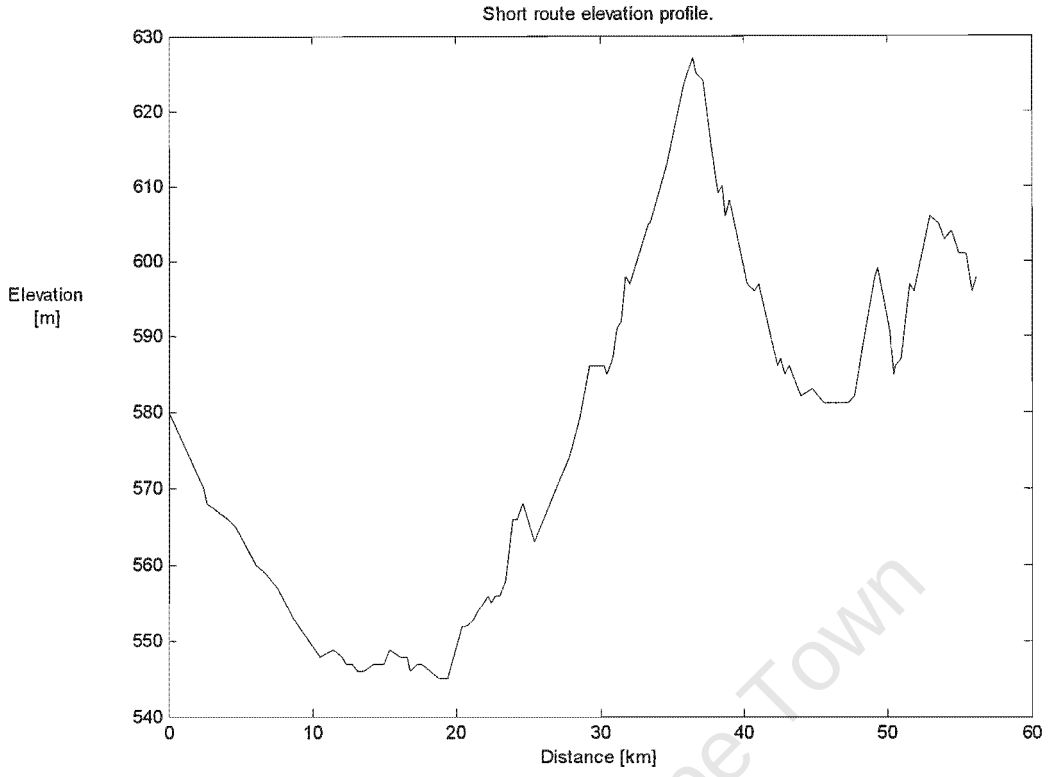



**Figure 6 Elevation Profile for Short Route**

### 8.2.1 A Constant Battery Power Controller

In the absence of energy or power constraints one can fly the solar cycle from start to finish in a matter of minutes. For the simulation to be of any value it must include a control constraint. The simplest constraint is an energy constraint, i.e. when the battery is flat no further energy can be drawn from it. This is also inherent in the system.

**Table 3 Constant Power Controller Results with No Wind**

| No Wind | Short Route | | |
|---|---|---|---|
| | Time [s] | Average velocity [m/s] | Execution time [s] |
| | 3210 | 17.482 | 2.07E+01 |
| | 3295 | 17.031 | 8.06E-01 |

**Table 4 Constant Power Controller Results with Known Wind**

| Known Wind | Short Route | | |
|---|---|---|---|
| | Time [s] | Average velocity [m/s] | Execution time [s] |
| | 3099 | 18.110 | 3.90E+00 |
| | 3044 | 18.438 | 3.64E+00 |

The results are illustrated in figures 7 and 8. The time to complete the route is 3295 seconds, or 54.92 minutes. The mean velocity is 17.031 $m.s^{-1}$ or 61.31 kph.

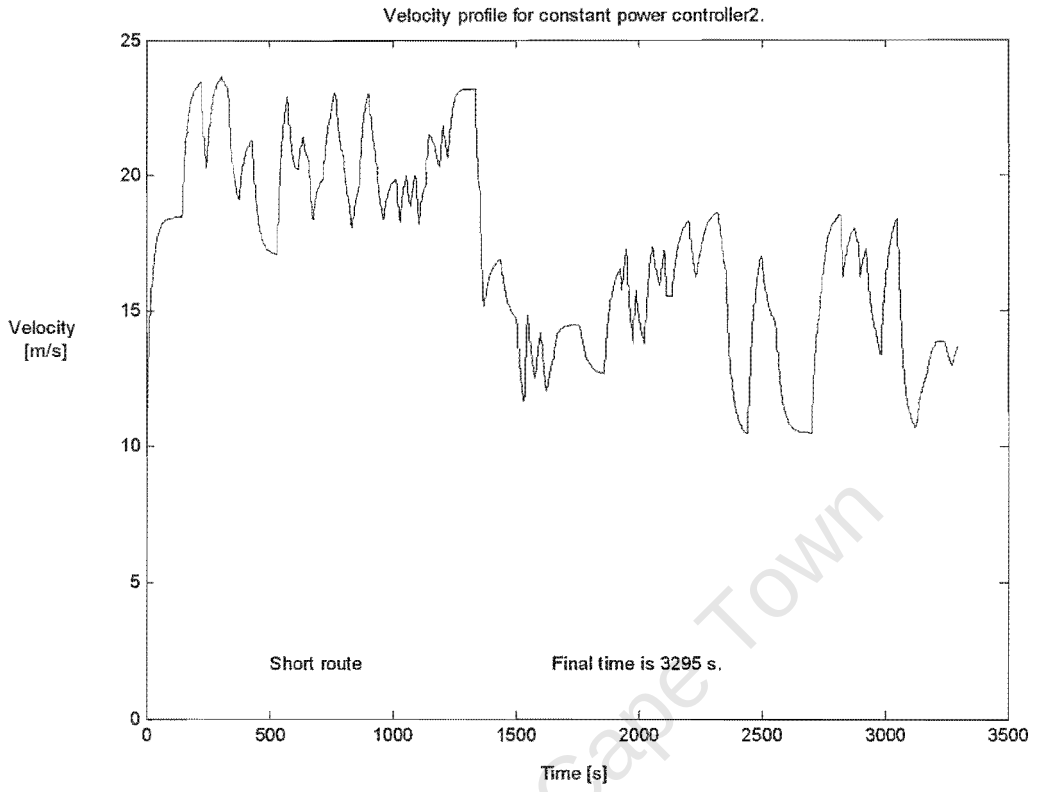



**Figure 7 Velocity Profile for Constant Power Controller**

The drop in the mean solar cycle velocity corresponds to the depletion of the battery, as can be seen in the figure below. It appears that the battery is depleted before the major incline has been travelled.

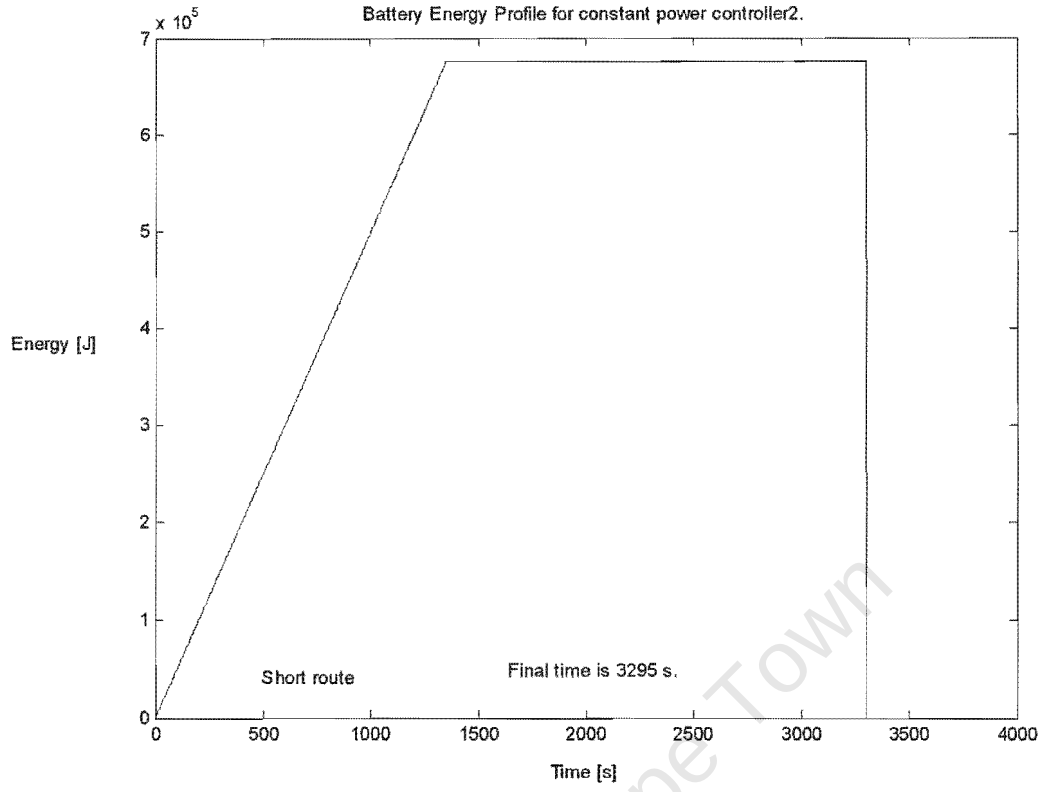



**Figure 8 Battery Energy Profile for Constant Power Controller**

As the gradient profile of the given route is known, it is possible to determine the optimal setting of the battery power level for the given route. This results in quicker completion of the stage and a smaller spread in the velocity. Figure 9 shows a velocity trajectory that is closer to being constant than the previous result.

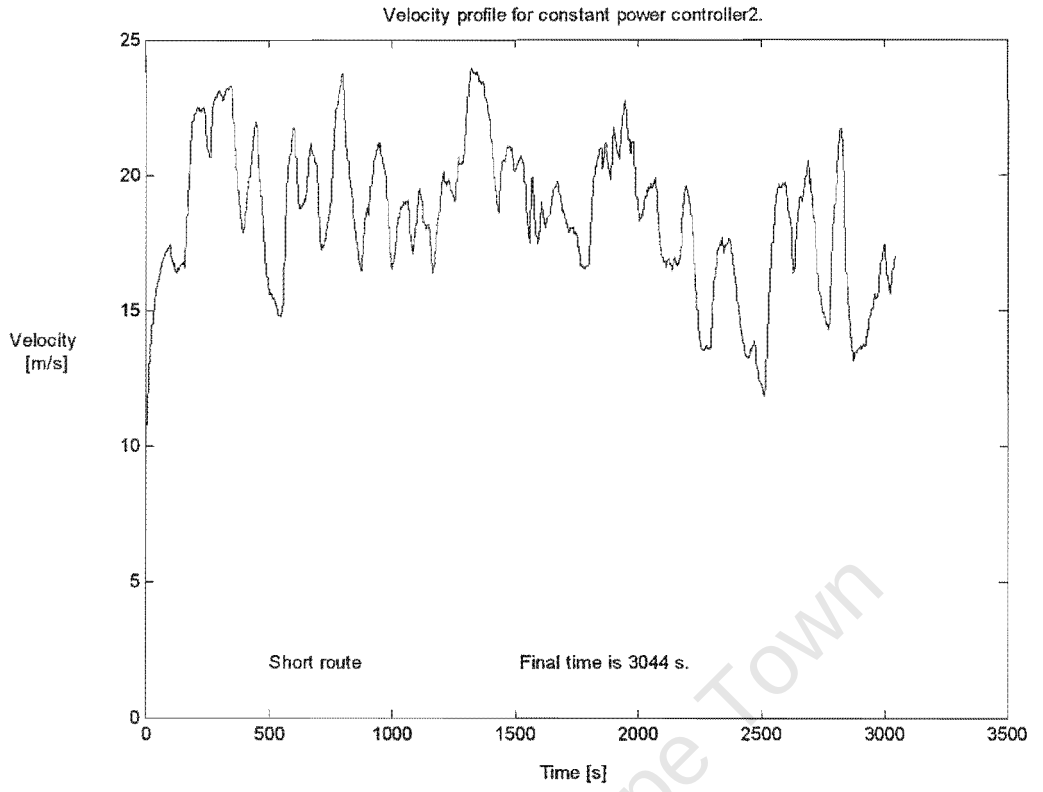



**Figure 9 Velocity profile for Constant Power controller, ideal case.**

The optimal battery power setting is 210.3587 watts. The time is four minutes quicker than the first power setting.

In the presence of a known wind, the results change somewhat. The solar cycle can be hindered or assisted by the wind. The figure below shows the simulation result for the 500 W power setting, including wind.

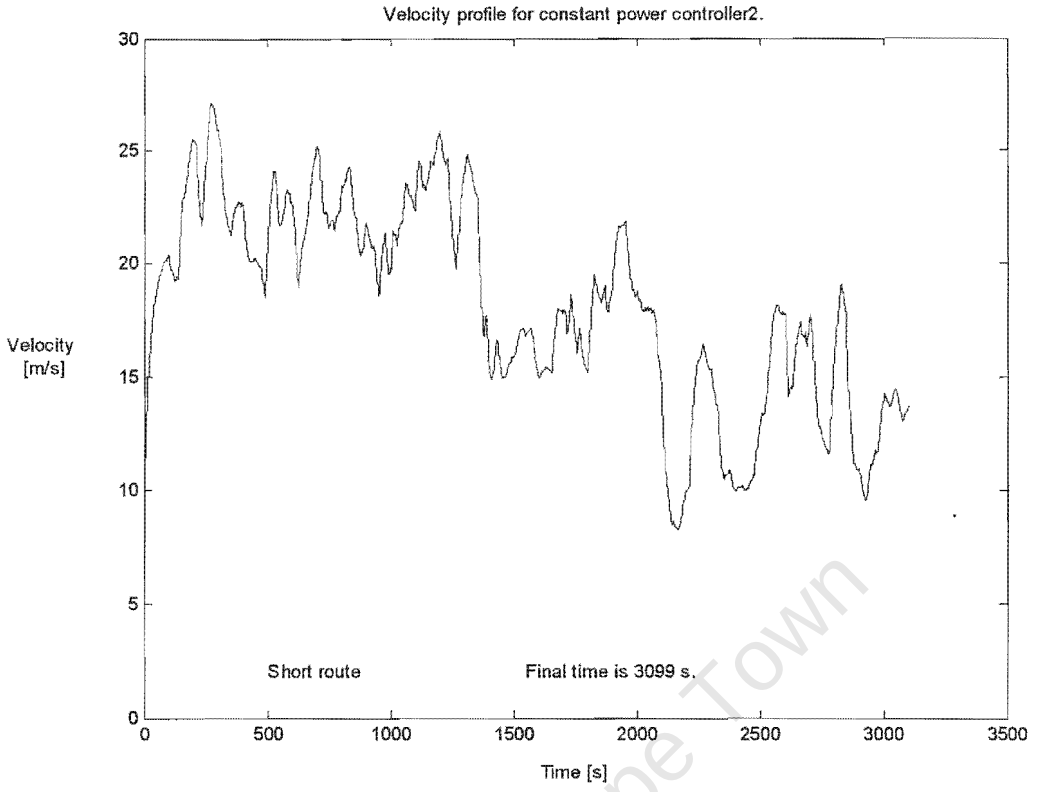



**Figure 10 Velocity Profile for Constant Power Controller, in presence of wind**

The drop in the mean velocity appears to be staggered. At three points, the velocity dips below 10 $m.s^{-1}$. The time, 3099 s, has improved, due to the assistance of the wind. The mean velocity of the known wind is –2.131 $m.s^{-1}$ for the first 4000 seconds. This is a significant tailwind of 7.62 kph.

For the setting of 210 W, the time, 3044 s, is still quicker than the 500 W setting, but the battery has not been fully utilised.

The emerging trend is that the best results occur when the battery energy is exhausted as the solar cycle reaches the end of the stage.

### 8.2.2 A Constant Velocity Controller

When considering the constant velocity case, one faces the choice of constraint again. The battery energy constraint is still the most sensible choice.

Table 5 lists the results for all the simulations run with the constant velocity controller, without the presence of wind.

**Table 5 Constant Velocity Controller Results with no wind**

| No Wind | Short Route | | | |
|---|---|---|---|---|
| | | Time [s] | Average velocity [m/s] | Execution time [s] |
| No power constraint | | 3278 | 17.120 | 2.32E+02 |
| Energy and Power constraint | | 3244 | 17.300 | 1.47E+02 |
| Stringent energy constraint | | 3177 | 17.669 | 1.15E+02 |
| Regeneration | 1 | 3185 | 17.624 | 3.59E+02 |
| | 2 | 3177 | 17.667 | 4.83E+02 |

### a. With energy constraint and no power constraint

By setting the reference velocity to 20 $m.s^{-1}$, the solar cycle maintains a fast pace until running the battery flat. As can clearly be seen in figure below, the battery is flat after about 1495 seconds. The result is almost as slow as the worst constant power result mentioned above.

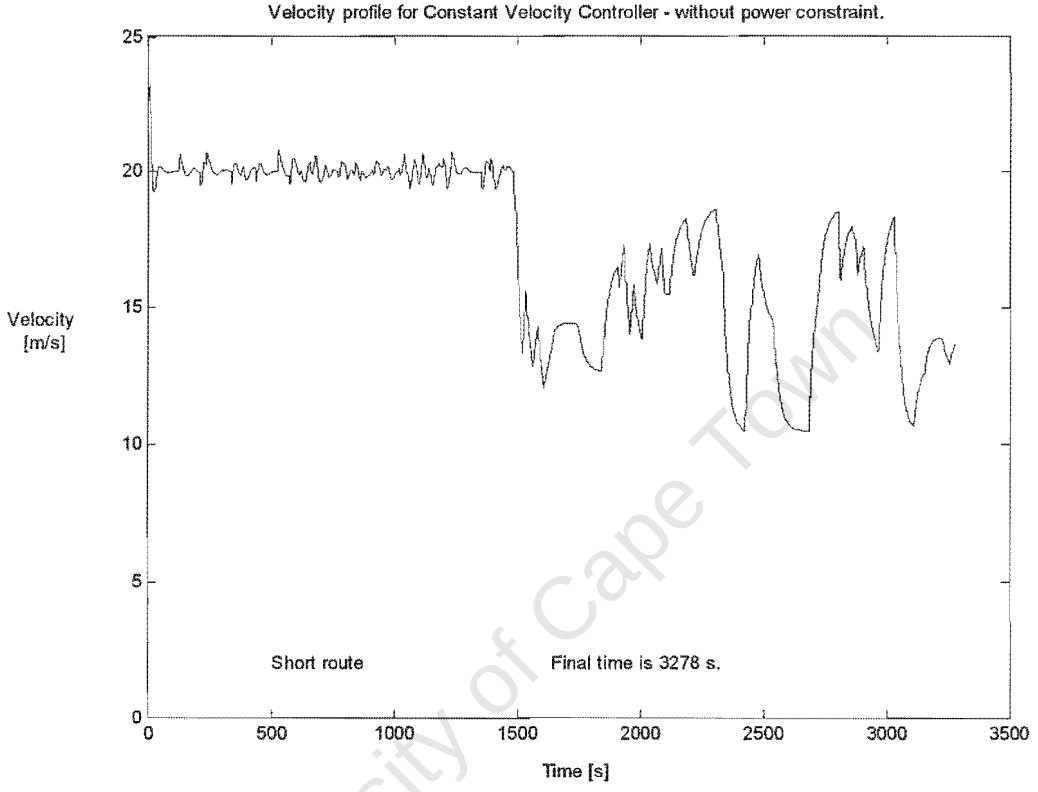



**Figure 11 Velocity Profile for Constant Velocity Controller, without power constraint**

A major problem with this approach can be seen in the next figure, where the peak power required is 20 kW to accelerate to the reference velocity. Even ignoring this point, the power required reaches 1 kW at a number of points.

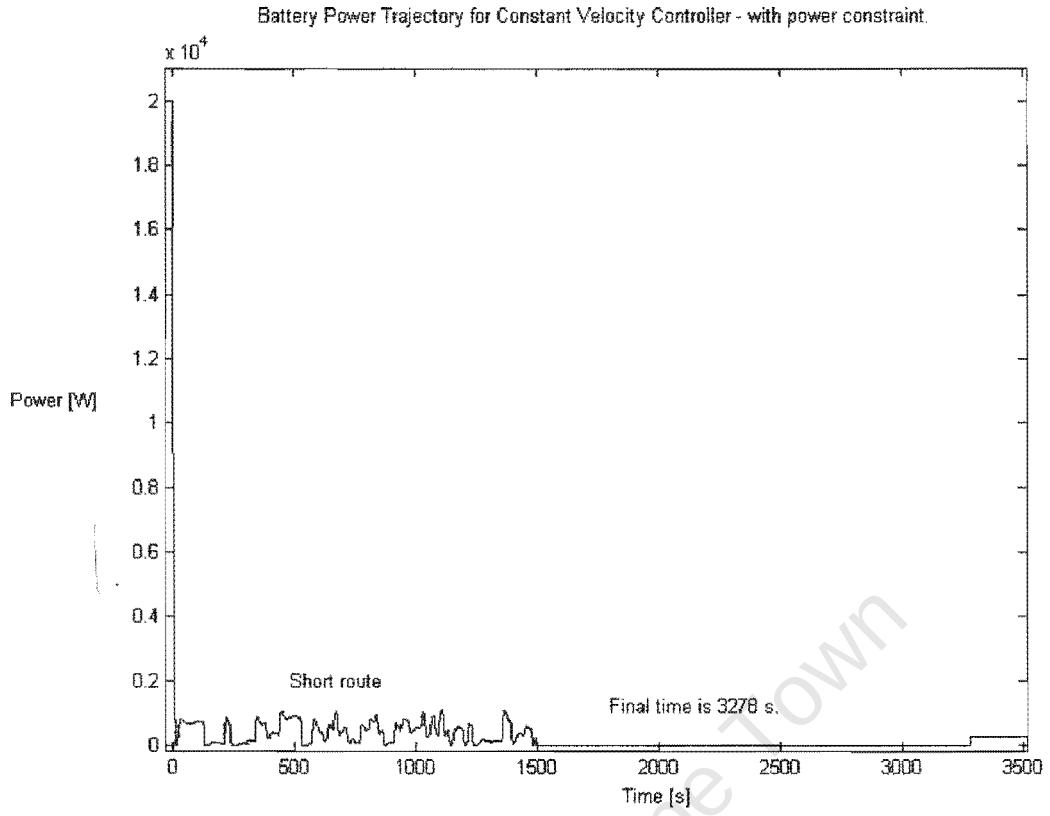



**Figure 12 Power profile for Constant Velocity Controller, without power constraint**

Some of the peaks, with half power points of about 400 W, span periods of more than a minute between half power points. This could lead to the prototype motor overheating. Another consideration is the effect of the high discharge rate on the battery and its capacity. It would be prudent to limit the peak power.

The step at 3279 s to 250 W is the residual of the initial control trajectory it is not part of the solution. Note the slight shift of the origin to better display the peak.

### b. With energy constraint and power constraint

In view of the preceding comments, the maximum power drawn from the battery is limited to 500 W. This is close to the continuous power rating of the motor.

It is interesting to note that the introduction of the power constraint improved the performance of the solar cycle. Inspection of the two figures below shows that the battery ran flat after about 1770 seconds. This confirms the trend mentioned in 8.2.1.

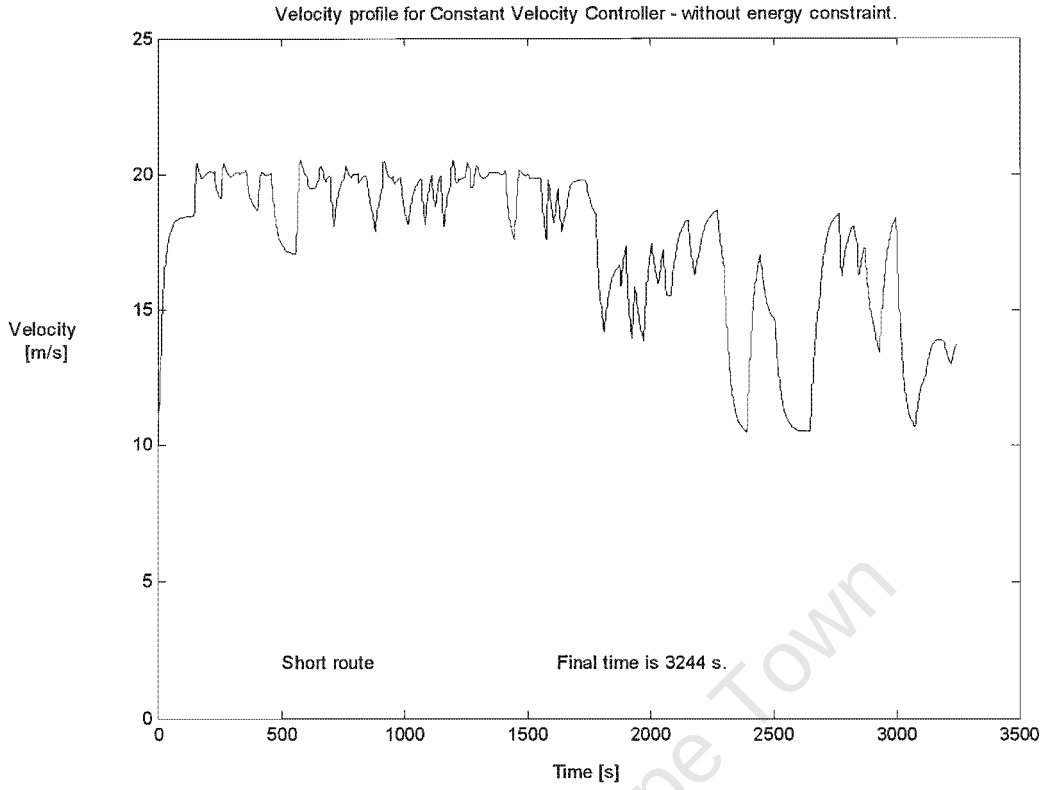



**Figure 13 Velocity Profile for Constant Velocity Controller, with power constraint**

The power limit clearly inhibits the solar cycle velocity in the first half of the velocity profile, as seen by the dips in the in the velocity at various points in the first 1500 seconds. These dips correspond to uphill sections of road where the power required to maintain the constant velocity exceeds what is available.

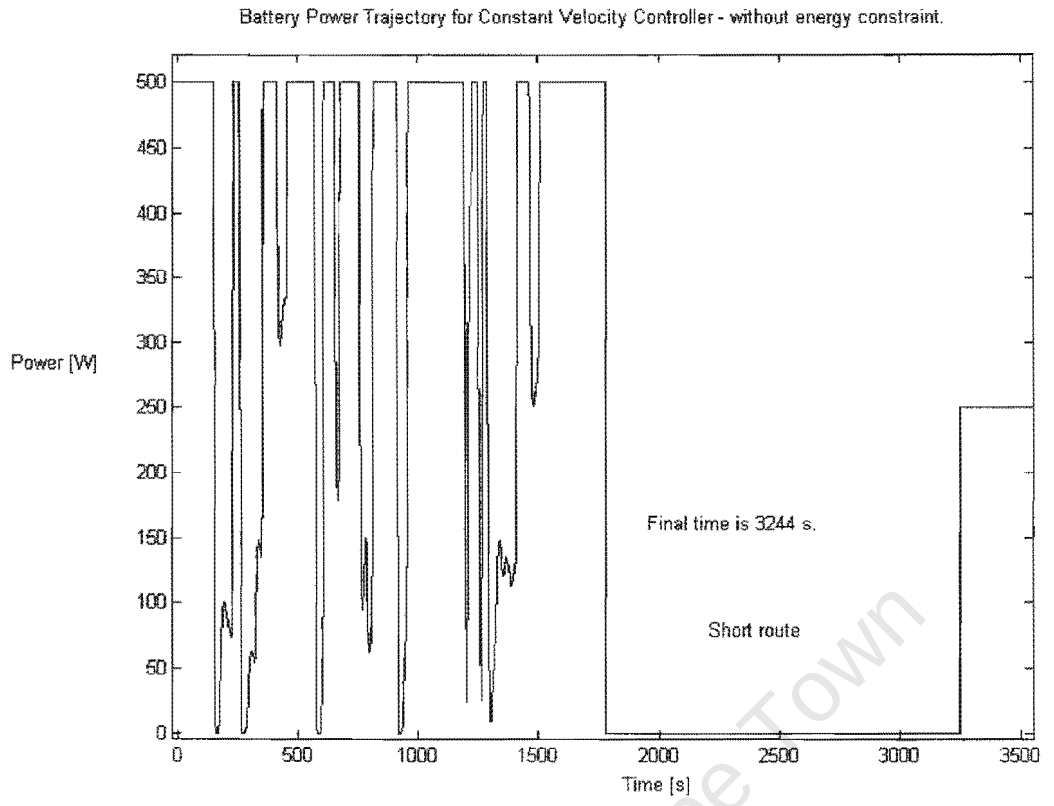



**Figure 14 Power Profile for Constant Velocity Controller, with power constraint**

The control trajectory clearly saturates at a number of points. The apparent ringing at 200, 1300 seconds is probably compensation for changes in the road gradient. Note the slight shift of the origin to better display the extremities of the trajectory.

### c. Crossing the line as the battery goes flat.

By adding a more stringent energy constraint, i.e. the battery energy is depleted as the solar cycle crosses the line; improves the performance. This matches the trend mentioned in 8.2.1. It is shown in figure below.

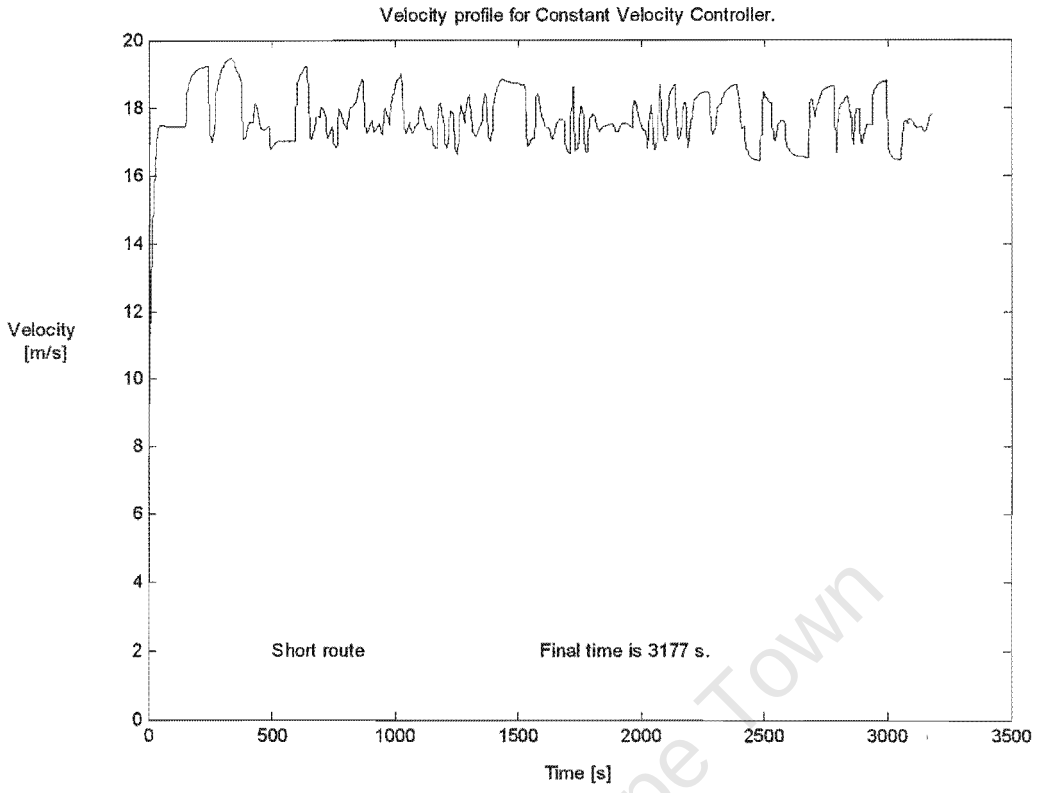



**Figure 15 Velocity Profile for Constant Velocity Controller**

The velocity profile is not flat because the solver was set to terminate when the standard deviation in the velocity became less than unity. To achieve a smaller deviation would require more tuning of the control law parameters, more simulations and more time. As it stands it serves its purpose.

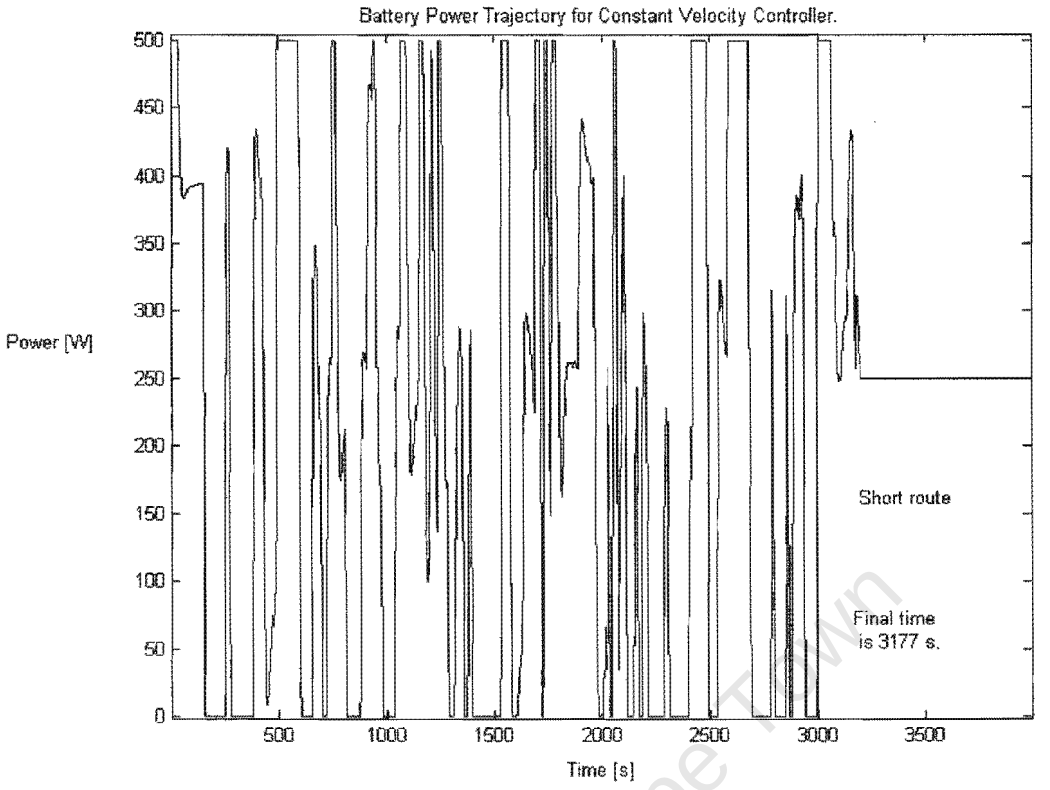



**Figure 16 Power Profile for Constant Velocity Controller**

As is shown in figure above, the battery delivers power until the end of the stage is reached. The power constraint is also reached at a number of points, which indicates where large deviations from the set velocity occur. These saturation periods are short compared to the previous case.

In the presence of a known wind, the results are similar to those in the above sections. If the mean wind velocity is positive (i.e. increases the apparent velocity) the simulation results in longer times and lower mean velocities, as expected when riding into a head wind. As expected, the results improve in the presence of a tail wind.

**Table 6 Constant Velocity Controller Results for Known Wind**

| Known Wind | Short Route | | |
|---|---|---|---|
| | Time [s] | Average velocity [m/s] | Execution time [s] |
| No power constraint | 3035 | 18.492 | 1.60E+02 |
| Energy and Power constraint | 3008 | 18.658 | 2.73E+02 |
| Stringent energy constraint | 2997 | 18.727 | 3.65E+02 |
| Regeneration | 3018 | 18.588 | 7.73E+01 |

### d. Adding Regeneration to the Constant Velocity Controller

Adding the capacity for regeneration – that is allowing power to flow into the battery when the solar cycle velocity is above the desired velocity – allows the simulations to answer one of the contentious points of debate among solar cycle engineers.

The results are somewhat surprising. Regeneration, in this form, does not improve the performance of the solar cycle with respect to a constant velocity strategy.

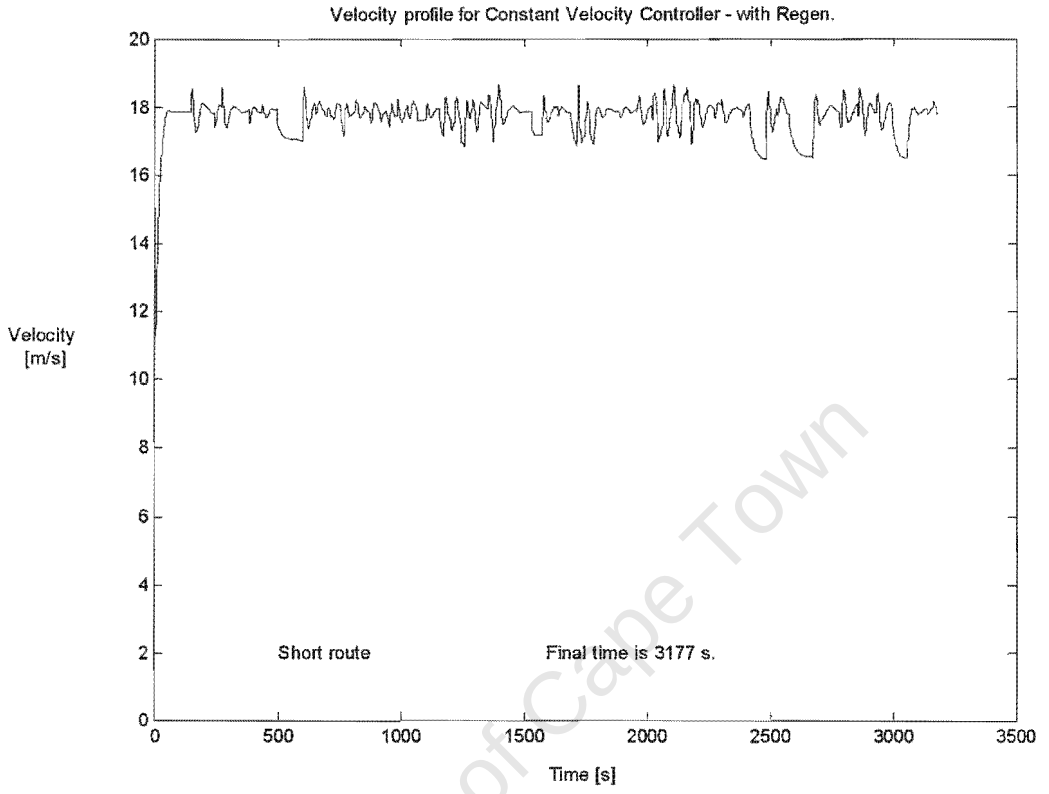



**Figure 17 Constant Velocity Controller with regeneration**

As figure 17 shows the velocity peaks are effectively capped, but some of the dips to have greater spread. The physical interpretation is that the solar cycle has less kinetic energy from a fast downhill section and now takes longer to travel uphill, which uses more battery energy than previously. In essence, the potential energy is converted to electrical energy in a solar cycle with regeneration instead of being converted to kinetic energy in a solar cycle without regeneration.

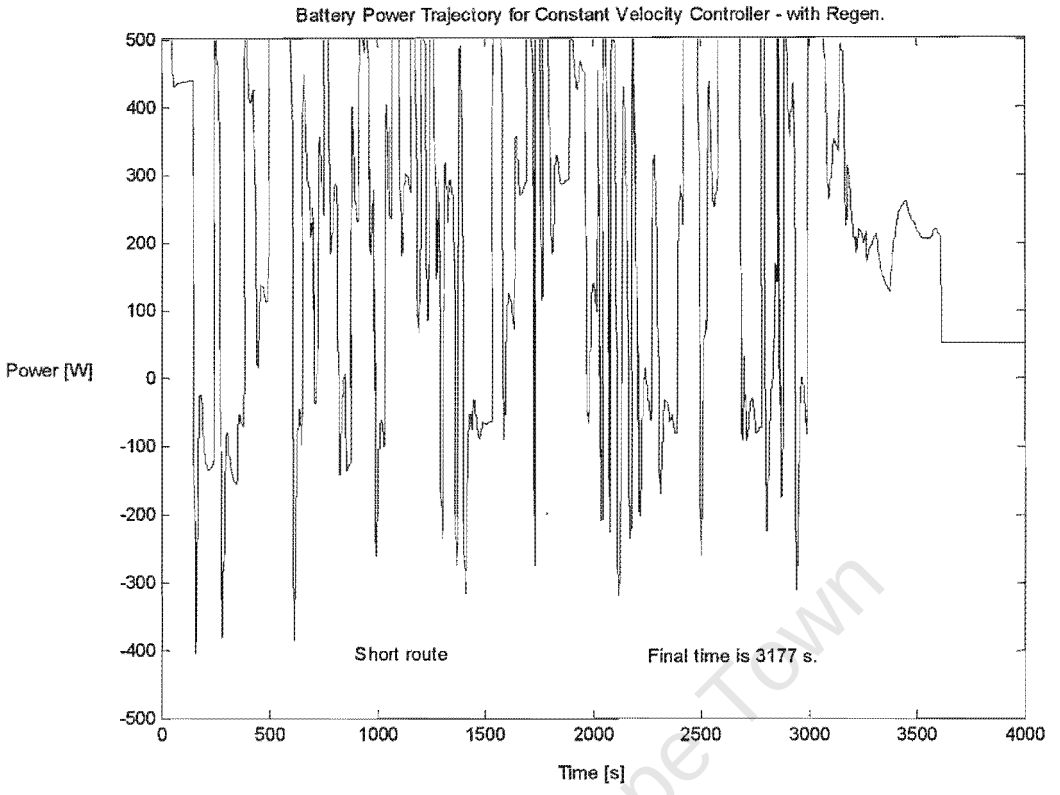



**Figure 18 Power profile for CV Controller with Regeneration**

Looking at the power flows, the power returned to the battery is never very much. If one considers a 100 kg solar cycle travelling at 18 $m.s^{-1}$ has 16.2 kJ (4.5 Wh) of kinetic energy and the same cycle travelling at 19 $m.s^{-1}$ has 1.85 kJ (0.513 Wh) more energy, the only point where a significant amount of energy will be returned to the battery is a cyclist change.

Given that the prototype has two freewheel couplings in the motor's drive train, the issue of regeneration is not considered further in this dissertation.

### e. An Unknown Wind

In the presence of an unknown wind profile, using the wind model to generate a random wind velocity, the best performance that a constant velocity controller can produce is similar to the results in section 8.2.2.3, where the battery is depleted prior to the end of the stage or is not completely utilised. This is because the constant velocity controller has no means of compensating for a noisy input and cannot meet the stringent energy constraint.

The best possible approach is to set the velocity assuming that no wind exists, i.e. that the average wind velocity will be zero.

### 8.2.3 Optimal Control – Simple Method

All the MPT results, without the presence of wind, are listed in table 7 below. The simulations for the field test route are included to show that the Simple Method can be adapted to different routes by tuning the gain terms.

**Table 7 MPT Results with No Wind**

| No Wind | | Short Route | | | | Field test route | | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] |
| Simple | 1 | 4160 | 13.489 | 2.42E+01 | -0.78% | 3091 | 13.75 | 1.49E+01 | 0.53% |
| | 2 | 4160 | 13.489 | 2.42E+01 | -0.78% | 3091 | 13.75 | 1.49E+01 | 0.53% |
| | 3 | 4160 | 13.489 | 2.42E+01 | -0.78% | 3091 | 13.75 | 1.49E+01 | 0.53% |
| Nelder-Mead | | | | | | | | | |
| 40 segment | 1 | 5099 | 8.7292 | 1.92E+03 | | | | | |
| | 2 | 5099 | 8.7292 | 2.50E+02 | | | | | |
| 100 segment | 1 | 5099 | 8.5325 | 6.10E+02 | | | | | |
| | 2 | 5099 | 8.6408 | 6.14E+02 | | | | | |
| 400 segment | 1 | 5099 | 8.7652 | 1.92E+03 | | | | | |

**Table 8 MPT Results for Known Wind**

| Known Wind | | Short Route | | | | Field test route | | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] |
| Simple | 1 | 2949 | 14.413 | 2.85E+01 | 0.74% | 4083 | 13.743 | 4.31E+01 | 0.75% |
| | 2 | 2949 | 14.413 | 2.93E+01 | 0.74% | 4083 | 13.743 | 4.31E+01 | 0.75% |
| | 3 | 2949 | 14.413 | 2.93E+01 | 0.74% | 4083 | 13.743 | 4.19E+01 | 0.75% |
| Nelder-Mead | | | | | | | | | |
| 40 segment | 1 | 5099 | 8.7292 | 1.92E+03 | | | | | |
| | 2 | 5099 | 8.7292 | 2.50E+02 | | | | | |
| 100 segment | 1 | 5099 | 8.5325 | 6.10E+02 | | | | | |
| | 2 | 5099 | 8.6408 | 6.14E+02 | | | | | |
| 400 segment | 1 | 5099 | 8.7652 | 1.92E+03 | | | | | |

Using the method described in section 7.3.6.1, a number of simulations were run to determine the effectiveness of the method.

The method is a shooting method, where the control trajectory is taken as an initial condition and the BVP solved as an initial value problem. However, in the MTP case one cannot use time as the independent variable, as is the usual case.

We use the distance travelled state variable, $s$, as the independent variable and the battery energy expended state, $E_{batt}$, as the error metric. The velocity, $v$, is the only state not known at the final time.

This selection makes a good deal of intuitive sense, as the solar cycle must complete the stage having, ideally, entirely depleted the battery as it reaches the end of the stage.

A complete listing of the MATLAB code is in Appendix D. Note that this implementation is based on the work of section 8.1.1, not the general case.

Using the same scenario as 8.2.2, the simple method produces somewhat worse results, 33 seconds slower.

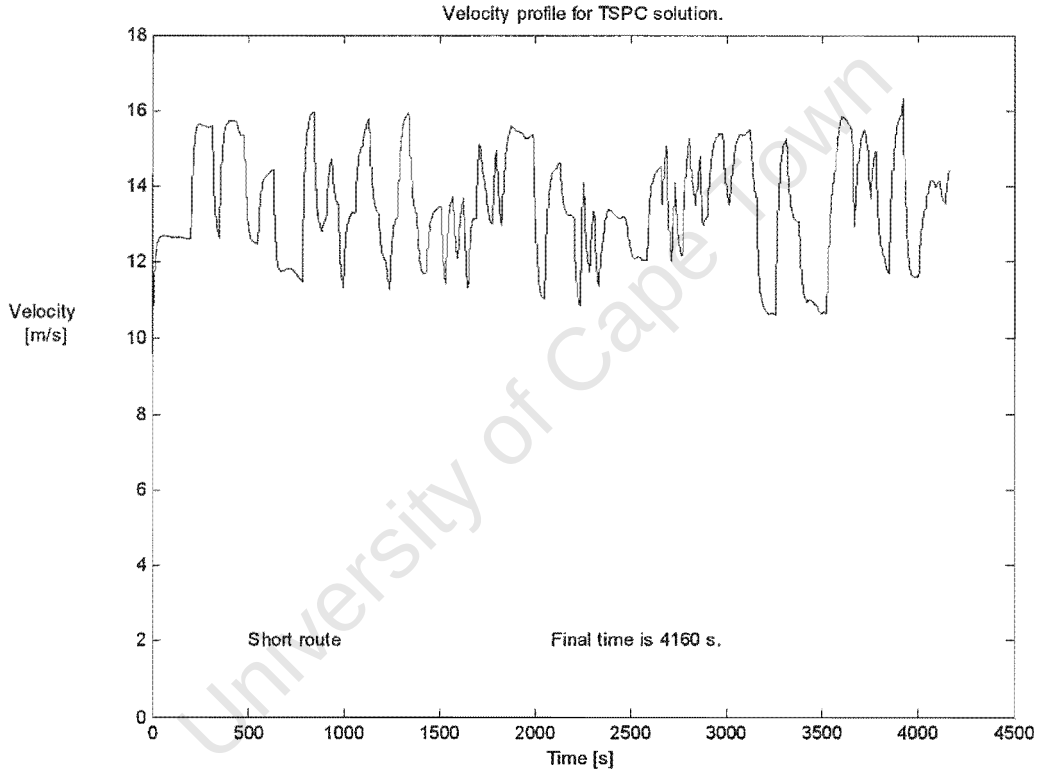





**Figure 19 Velocity Profile for Simple Method**

The velocity trajectory is flat, but with a high level of ripple. Looking at the battery power trajectory shows that it is rather low, below 250 W for the most part, when compared to the constant velocity solution where the power trajectory spans the whole range.

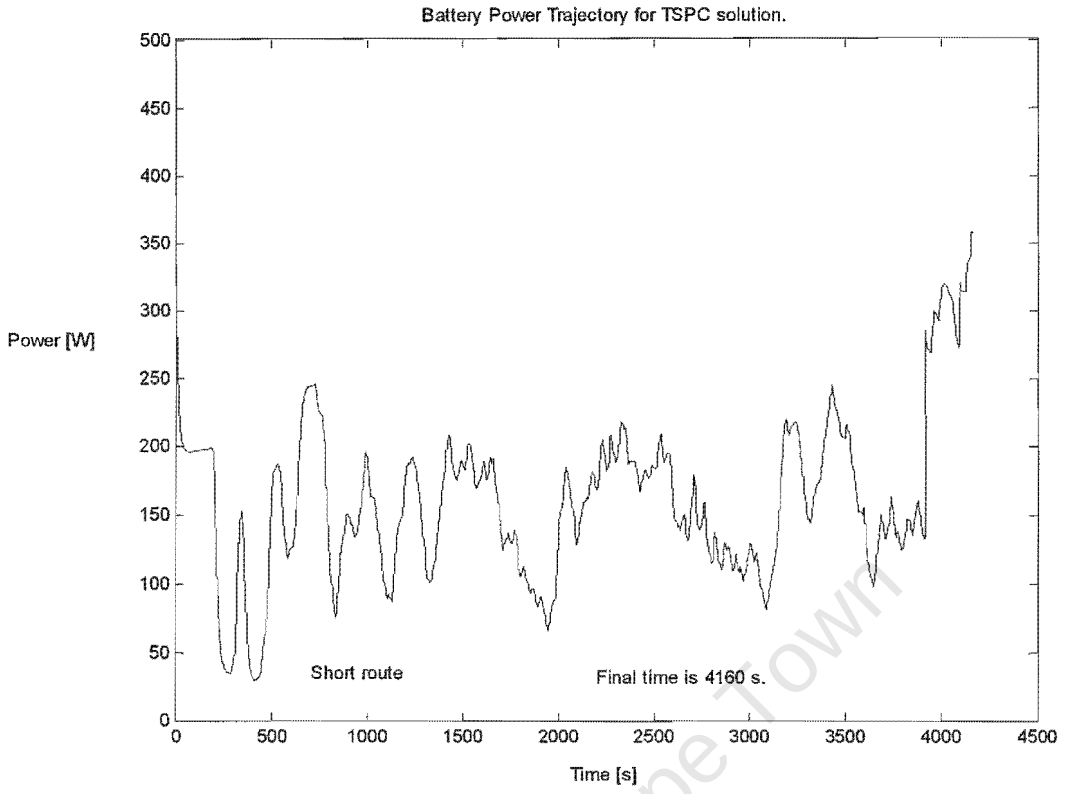



**Figure 20 Power Profile for Simple Method**

The method was tuned to converge within ten iterations; better results can be achieved by tuning the control law, but at the expense of more iteration.

The situation is actually better in the presence of a known wind. The power trajectory for the case with a known wind is shown below.

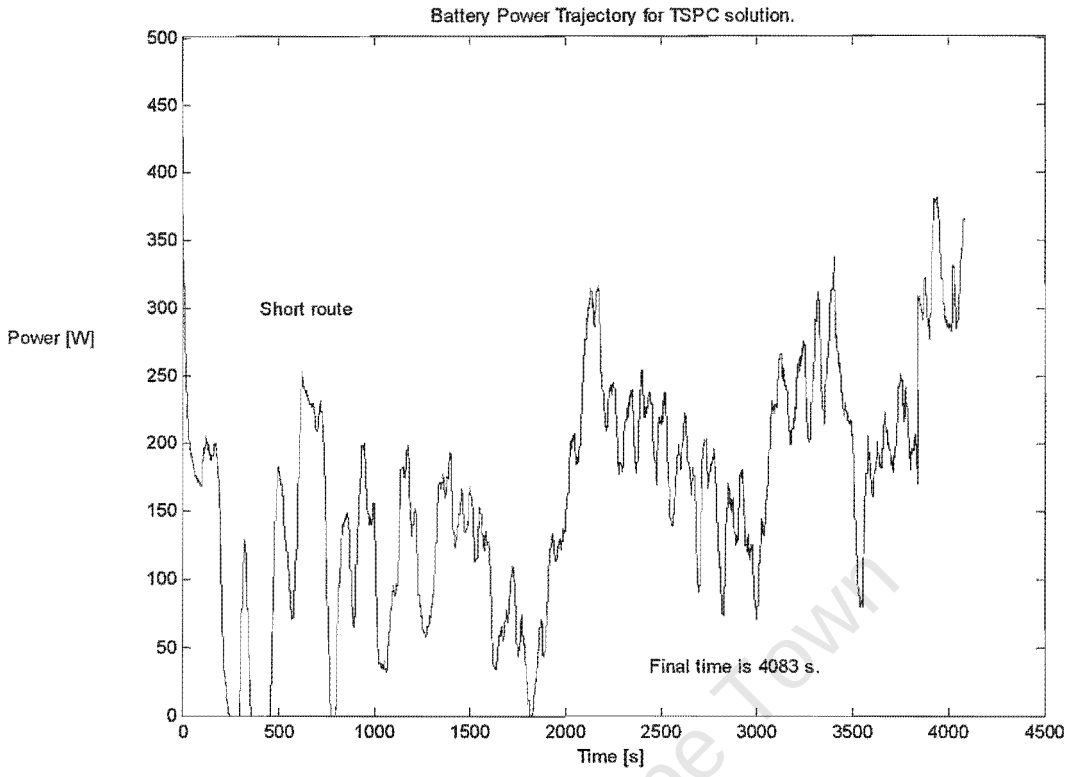



**Figure 21 Simple Method Power profile with wind**

The improvement is due to greater utilisation of the available power range.

### 8.2.4 Optimal Control – Projected Gradient Method

The projected gradient method was chosen as a representative of the gradient methods available. The oft-used steepest descent method is a special case of the projected descent method.

Schwartz's description of the bounded projected gradient method is incomplete. A number of elements are assumed known to the reader and their omission makes the algorithm unusable to a reader not entirely au fait with his field. Kelley has published a number of IVP solvers implemented as MATLAB functions on his web page. His projected gradient code[24] was modified to allow parameter passing.

One of the problems faced by gradient methods is the sheer size of the problem. The length of the control trajectory is worst-case time to complete the stage divided by the simulation time step. The worst-case time is taken as the time to complete the stage at an average velocity of 11 $m.s^{-1}$. For a short test route of 56 km, the time is 5091 seconds; the control trajectory contains 5091 elements. Some gradient methods, especially the second order methods, require the calculation of the Jacobian matrix, a 5091 by 5091 matrix. These calculations are time consuming. Even the simpler first order methods require the gradient of the function to be solved with respect to each control point or element. In solving the MTP, one cannot analytically derive the gradient of the performance function. The simplest approach is to calculate the numerical derivative, which requires 5091 function evaluations in this case. Given that each function evaluation takes about 3.3 seconds to compute, simply to calculate the gradient of the control trajectory will take approximately 4.6 hours.

In order to reduce the computational load, one can split the route into stages and solve them consecutively, although one will probably sacrifice some accuracy.

Due to the computational demands, the method is not considered further for the solution of the MTP simulations.

### 8.2.5 Optimal Control – Nelder-Mead Method

The Nelder-Mead method has two advantages over the gradient methods when faced with a non-linear problem. It does not require gradient information, and it has less stringent starting requirements[25]. The disadvantages are its tendency to stagnate at a non-stationary point, and its erratic convergence behaviour. In addition, its convergence has not been formally proven beyond the single dimension case[26].

In this implementation of the Nelder-Mead method, a further short cut was taken. The control trajectory was approximated by a cubic B-spline. The use of the B-spline allowed the reduction in the length of the control trajectory, by replacing it with the spline control points. At the extreme, the control trajectory could be replaced by a single segment spline with four control points. This vastly reduced the number of function evaluations that needed to be performed at the expense of optimality. It is simply not possible to approximate the optimal solution using a spline with fewer control points than elements in the control trajectory. An additional problem is the difficulty of representing the power constraint in such a way as to constrain the spline control points. This approach could be used for the projected gradient method, with similar difficulty in handling the constraints.

This is illustrated in the following two figures; the first is the resulting velocity profile for a forty-segment spline, and the second is the control trajectory, i.e. the forty-segment spline with the relevant constraints. This a typical solution found by the method for this particular route.

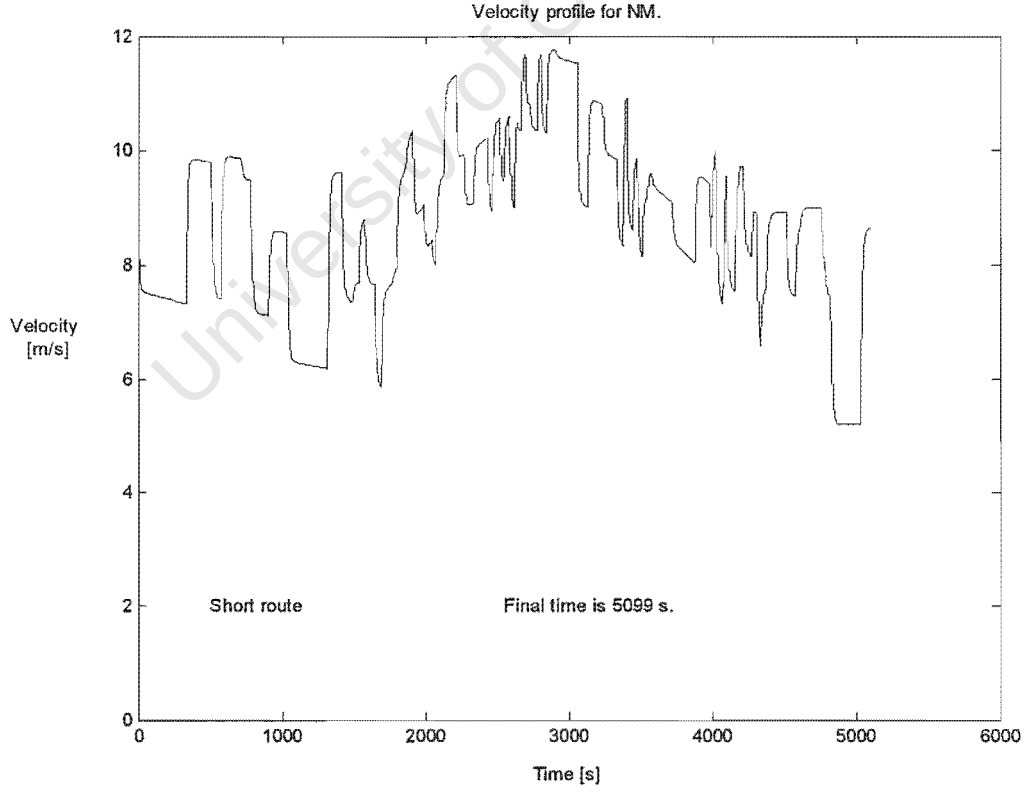



**Figure 22 Velocity Profile for Nelder-Mead solver**

For the short route, a local minimum of 5099 seconds appears to attract the solver regardless of the starting spline control points, which are randomly generated. This appears to apply for a large region and be independent of the number of segments used in the control trajectory, as seen in the results table in section 8.2.3.

Attempts to carefully set-up the B-splines did not minimise the extent of the excursions outside the constraints for a particular route. What is needed is a translation from the control trajectory space to the spline control point space of the constraints. This has not been attempted.

### 8.2.6 Optimal Control and an unknown wind

When faced with an unknown wind, the MTP cannot be minimised by the shooting method. Some form of adaptive controller is required.

### 8.2.7 Insights gained from the simulations

The battery energy must be depleted as the solar cycle reaches the end of the stage.

The constant velocity approach appears to be the closest to optimal.

The use of regeneration does not offer any advantage.

Considering that the constant velocity approach does not actually reach a constant velocity, an alternative method may produce better results.

## *8.3 Designing the Optimal Controller*

### 8.3.1 The influence of uncertainty

Thus far, no attempt to account for the effects of uncertainty has been made.

Using the predefined wind vector in the above simulations does not change the relative performance results. In addition, due to the prevailing tail wind, the times are quicker.

Even with the best metrological information and the skills to interpret it, one cannot predict the next day's wind behaviour with absolute certainty. At best, one could determine the prevailing wind, but not the local effects of hills, valleys and so forth.

To make a practical contribution to the success of a team, the controller must be able to accommodate this uncertainty.

The goal in designing the controller is to rapidly develop a functional prototype that can readily be extended to aid higher-level strategic decisions.

The discussion starts with an early attempt derived from a straightforward engineering approach to the problem. This attempt was made without reference to any formal theory or control philosophy.

The discussion then proceeds to the design of the controller to be implemented in the EMS. It takes advantage of all the previous theoretical and development work.

### 8.3.2 Energy Balance Controller

One of the early attempts to write an optimal controller used the idea of estimating the energy consumption for the rest of the day's stage assuming that the rest of the stage would be run at a constant speed. The constant speed value is adjusted until the estimated energy required matched the energy stored in the battery. The constant speed value is then used as the reference velocity for a PID battery power controller. The calculation period can be varied, typically being 1 second. This is, in essence, a shooting method.

The reference velocity is relatively flat until the end of the stage. This pick up is probably due to the approach of the end of the stage.

In contrast, the solar cycle velocity rises and dips in a similar manner to the earlier simulations. This is clearly due to the power constraint, as can be seen in the power trajectory.

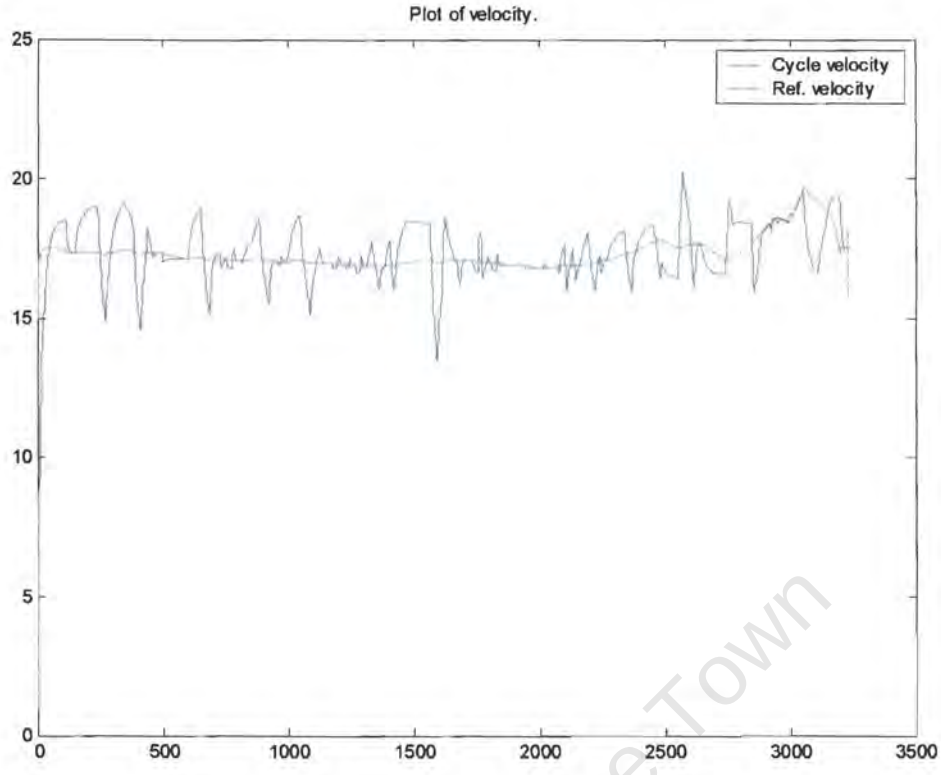



**Figure 23 Velocity Profile of Energy Balance Controller**

The power limit in this case is set slightly higher than 500 W. The power trajectory resembles that of the constant velocity simulation, although the results are worse.

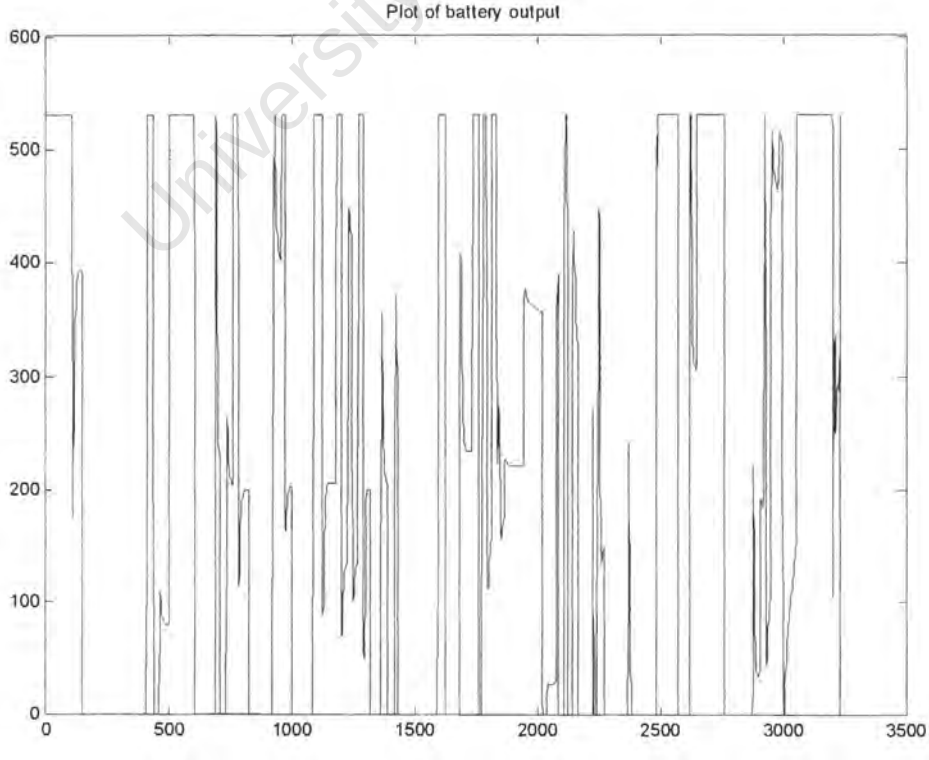



**Figure 24 Power Profile for Energy Balance Controller**

The problem with this controller is that it did not account for uncertainty in the wind in a direct manner. What it does is implicitly alter the available battery energy. This is what gives rise to the bump in the control velocity in Figure 23. A more direct, possibly predictive controller was needed.

At this stage, the system dynamics had not been fully analysed. The problem was regarded as one of setting the desired velocity and having the solar cycle respond, even though the controller contained a PID controller controlling the velocity using the battery power.

The PID controller did cause a number of problems, largely related to the integral term being wound up.

### 8.3.3 Two Stage Predictive Controller (TSPC)

The TSPC was written after all the theoretical work on optimal control had been completed. Malinowski[27] suggests the TSPC as a closed-loop predictive controller for handling the uncertainty in uncontrolled inputs. It is claimed to offer superior performance to more basic, open loop, controllers and classical, fixed rule controllers.

The TSPC generates a number of forecasts over a limited time horizon. After a short interval, the forecasts are evaluated and the control trajectory matching of the most accurate forecast is selected for use.

Malinowski's summarises his TSPC in the following steps:

1. Obtain state measurements, and update process model.
2. Generate $N$ forecasts of the uncontrolled inputs, $z^{ij}(t), t \in (t_i, t_{f_i}]$. Where $t_{f_i}$ is the end of the prediction horizon.
3. Decision mechanism:

$$\min_{m_i} \sum_{j=1}^{N} p^j \left[ \int_{t_i}^{t_{i+1}} q\left(x^j(t), m(t), z^{ij}(t)\right) dt + \min_{m^j_{i+1} \cdots m^j_{f_{i+1}}} \int_{t_{i+1}}^{t_{f_i}} q\left(x^j(t), m^j(t), z^{ij}(t)\right) dt \right]$$

Where $\sum_j p_j = 1,\ p_j \geq 0,\ m^j(t) = m(t)\ for\ t \in (t_i, t_{i+1})$

He also adds the warning that the TSPC is highly sensitive to the accuracy of the model and the forecast mechanisms used.

The TSPC seems an ideal approach to handling the uncertainties in the wind, gradient, cyclist output and insolation seen by a solar cycle. In addition, the inclusion of forecasts relating to tactical manoeuvres is readily apparent. A simplification of the decision mechanism is possible, and the following TSPC procedure is proposed for a solar cycle:

1. Obtain state measurements
2. Generate forecasts
3. Solve MTP for each forecast
4. Select the most accurate forecast
5. Send forecast trajectory to controller

Each step is discussed below.

### a. Forecasting Mechanism

When examining the basic dynamics of the uncontrolled inputs being considered, the following three trends were found to be adequate:

The conditions could improve, be static or deteriorate when viewed from the perspective of the influence of the conditions on the performance of the solar cycle.

By accommodating these trends, the forecasting would be adaptive to the prevailing conditions regardless of the starting conditions. The speed of response may need careful attention.

To recall, the uncontrolled inputs are aerodynamic losses due to wind, the road gradient and the cyclist's power output. The first can be estimated from the apparent wind direction as seen by the solar cycle, the gradient can be measured directly, as can the cyclist's output, albeit more demanding in terms of instrumentation.

By multiplication of choices, where there are three choices of three states, twenty-seven forecasts are required.

In terms of the prototype, the number of forecasts reduces to nine, as the cyclist's output is not being measured. Moreover, if the solar cycle is following a route with a known gradient profile, the gradient forecasts become predictable, and the number of forecasts reduces to three.

### b. TSPC solver

All three methods used in the previous simulations were used in the TSPC. In some cases, the methods required extensive tuning.

During the development of the TSPC solvers, it became apparent that none were adequate for implementation in the EMS. As a result, a new solver was developed – named the simple solver 2 (SS2).

#### *i. Simple Method solver*

While the solver used is the same as used previously, it does show a number of convergence problems.

Firstly, the gain parameters need to be tuned relative to the length of the forecast, as the battery capacity available varies according to it. If the gains are not correctly tuned, the solver will not converge to a solution within the required tolerance.

Secondly, sometimes the forecast spans a section of the route with a significant drop in elevation. This causes the solar cycle to increase its velocity regardless of the control input. This situation prevents the solver from converging.

To solve the first problem required the implementation of gain scheduling, where the pre-tuned gains were selected according to the length of the forecast.

The second problem was solved in a rather robust manner that prevented the TSPC from being locked into an endless loop. The number of iterations performed by the solver is counted and when it exceeds a preset value, again scheduled according to the forecast length, the iteration loop is terminated and the previous iteration is taken as the solution.

This worked with a long forecast horizon and long validation period (greater than 60 seconds). To reduce the computational load required the reduction of these lengths. Setting the forecast horizon length to 2 km, and the validation period to 20 seconds showed a significant decrease in iteration time. See tables 9 & 10 in subsection e below. However, the convergence of the method became a little more erratic.

### *ii. Projected Gradient Method solver*

The speculation in 8.2.4 was proved correct when the method was implemented in the TSPC, however the Armijo[28] condition is not always met. The method uses a line search to establish the step size of the current iteration; the Armijo condition is met when the best step size has been found. Modifying the method to terminate the loop searching for a solution when the condition was not met did not yield any improvement. It results in an imperfect solution for the point under consideration, but allows the method to continue running and find a solution trajectory. However, the resulting trajectory is very close to a zero trajectory and not a near-optimal solution. The results are not presented.

### *iii. Nelder-Mead Method solver*

When applied over shorter route sections, as per the TSPC procedure, the Nelder-Mead solver produces reasonable results. Stagnation of the method sometimes occurs, but it still yields a reasonable solution. The number of spline control points does not have a major effect on the solutions found.

The Nelder-Mead method comes close to matching the requirements of the EMS. However, the fact that it can become attracted to a non-minimal solution region diminishes its utility as a robust solver.

### *iv. Simple Solver 2*

The Simple Solver 2 is a simple constant power solver. It solves for the constant power required over a discrete time interval. Its key feature is that it is robust. As seen in the results, the constant power controller gave results that are within a 5% band of the other controllers. This robustness coupled to the TSPC's ability to handle uncertainty provides the closest match to the requirements of the EMS.

The basic procedure is:

1. Calculate initial velocity trajectory corresponding to the initial power trajectory.
2. If any point on the velocity trajectory exceeds the upper or lower bounds, decrease or increase corresponding point in the control trajectory.
3. Check the energy balance for the forecast horizon, and raise or lower the whole control trajectory as required to achieve the balance.
4. Recalculate the velocity trajectory using the mean of the power trajectory.

The initial power trajectory is set at a constant 250 W.

The solver was constructed to iteratively improve the control trajectory, but the convergence rate was too low, and the iterative loop was removed. Hence, the solver is now a single step procedure.

## c. Forecast Selection

Instead of using a weighted sum of all forecast trajectories, with the difficulty of establishing the weights, the most accurate forecast is selected.

The accuracy of the forecasts is determined by comparing the least mean squares (LMS) difference of the forecast velocity trajectory to the actual velocity trajectory over a validation period. The forecast with the least difference is then taken as the most accurate.

The validation period is used to extract the validation data from the most recent set of recorded data.

### d. TSPC Code

The MATLAB code is listed in Appendix D.

The simulations are launched from `tester_converg.m`, where one can select the route and the wind conditions. The selection of solver is made in `TSPC_SM_opt.m`. Each solver is implemented as a function in a separate file, as per MATLAB syntax, or a series of files as required. The solver function filed are designated `TSPC_solver_X.m`, where `X` refers to the method used, `PD2` is the projected gradient method, `opt` is the simple method, `NM` is the Nelder-Mead method and `SS2` is the second simple solver.

The structure is discussed further in the appendix.

### e. TSPC simulation results

For the simulations, the forecast horizon was set at 10 km. This is slightly excessive, but is used to determine the worst-case computational load. It could readily be cut to 6 or 7 minutes. Anything less would probably not allow sufficient time for the laptop computer (laptop) in the support vehicle to be rebooted should it crash or its battery fail. See section 9 for more details about the EMS implementation.

The TSPC algorithm needs a set of recorded data for the forecast selection step. The validation period was set at 100 seconds. To generate this data, the initial control trajectory is set at 250 W for the first 100 s. It is quite clear in the control (power) trajectories in the various figures below.

An additional measure of the solution fitness is introduced. It is the percentage difference in the battery energy utilisation. It is calculated by integrating the power trajectory. If a small amount of capacity still remains in the battery at the end of the route the energy utilisation is stated as -X% whereas if more energy has been used it is stated as X%.

**Table 9 TSPC Results with no wind**

| No Wind | | Short Route | | | | Field test route | | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] |
| Simple | 1 | 3314 | 16.931 | 2.56E+02 | 5.79% | 2667 | 15.936 | 1.76E+02 | 0.28% |
| | 2 | 3314 | 16.931 | 2.56E+02 | 5.79% | 2667 | 15.936 | 1.76E+02 | 0.28% |
| | 3 | 3314 | 16.931 | 2.56E+02 | 5.79% | 2667 | 15.936 | 1.76E+02 | 0.28% |
| SS2 | 1 | 3568 | 15.722 | 1.10E+01 | -15.70% | 2690 | 15.798 | 1.41E+01 | -11.80% |
| | 2 | 3568 | 15.722 | 1.10E+01 | -15.70% | 2690 | 15.798 | 1.41E+01 | -11.80% |
| | 3 | 3568 | 15.722 | 1.10E+01 | -15.70% | 2690 | 15.798 | 1.41E+01 | -11.80% |
| NM | 1 | 3607 | 15.557 | 5.45E+02 | -18.70% | 2744 | 15.492 | 4.18E+02 | -13.20% |
| 4points | 2 | 3651 | 15.371 | 5.52E+02 | -18.40% | 2741 | 15.508 | 4.16E+02 | -11.60% |
| | 3 | 3606 | 15.558 | 5.52E+02 | -18.80% | 2752 | 15.447 | 4.09E+02 | -13.40% |

**Table 10 TSPC Results with Known Wind**

| Known Wind | | Short Route | | | | Field test route | | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] | Time [s] | Average velocity [m/s] | Execution time [s] | Energy imbalance [%] |
| Simple | 1 | 3139 | 17.874 | 1.83E+02 | 4.32% | 2280 | 18.648 | 9.18E+01 | -3.29% |
| | 2 | 3139 | 17.874 | 1.83E+02 | 4.32% | 2280 | 18.648 | 9.18E+01 | -3.29% |
| | 3 | 3139 | 17.874 | 1.83E+02 | 4.32% | 2280 | 18.648 | 9.18E+01 | -3.29% |
| Nelder-Mead | 1 | 3065 | 18.304 | 4.49E+02 | -16.30% | 2335 | 18.207 | 3.30E+02 | -11.10% |
| 4point (2seg) | 2 | 3067 | 18.293 | 4.40E+02 | -3.47% | 2400 | 17.715 | 3.52E+02 | -8.61% |
| | 3 | 3118 | 17.996 | 4.62E+02 | -7.17% | 2381 | 17.852 | 3.32E+02 | -3.77% |
| SS2 | 1 | 3342 | 16.787 | 1.13E+01 | -24.40% | 2637 | 16.118 | 7.27E+00 | -38.10% |
| | 2 | 3342 | 16.787 | 1.13E+01 | -24.40% | 2637 | 16.118 | 7.27E+00 | -38.10% |
| | 3 | 3342 | 16.787 | 1.13E+01 | -24.40% | 2637 | 16.118 | 7.27E+00 | -38.10% |

### *i. Simple Method solver*

Using the same scenario and predefined wind vector as used in section 8.2, the Simple Method produces some interesting results.

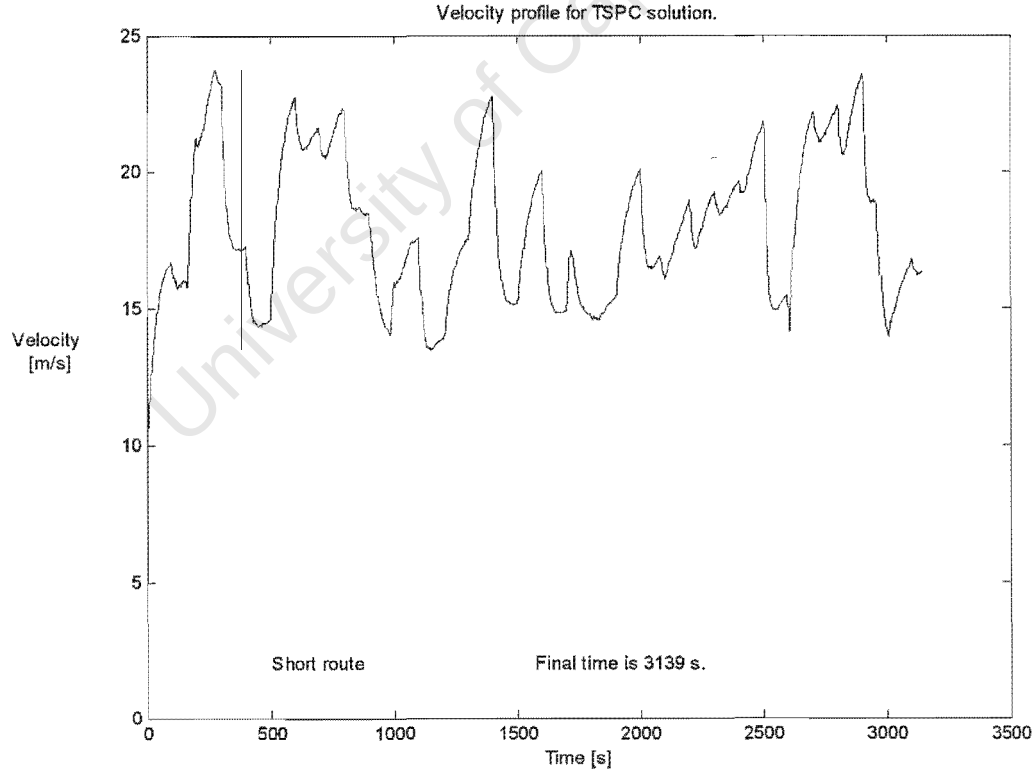



**Figure 25 Velocity Profile for Simple Method with known wind.**

The velocity trajectory is relatively free from high frequency noise, but not as smooth as some of the constant velocity results. What is notable is the quicker time, 3139 seconds, compared to 3177 seconds using the Constant Velocity controller. The result is plausible; the average speed is 17.87 $m.s^{-1}$, or 64.33 kph. The energy expended is within 5% of the available capacity.

Part of the explanation maybe observed in the battery power trajectory.

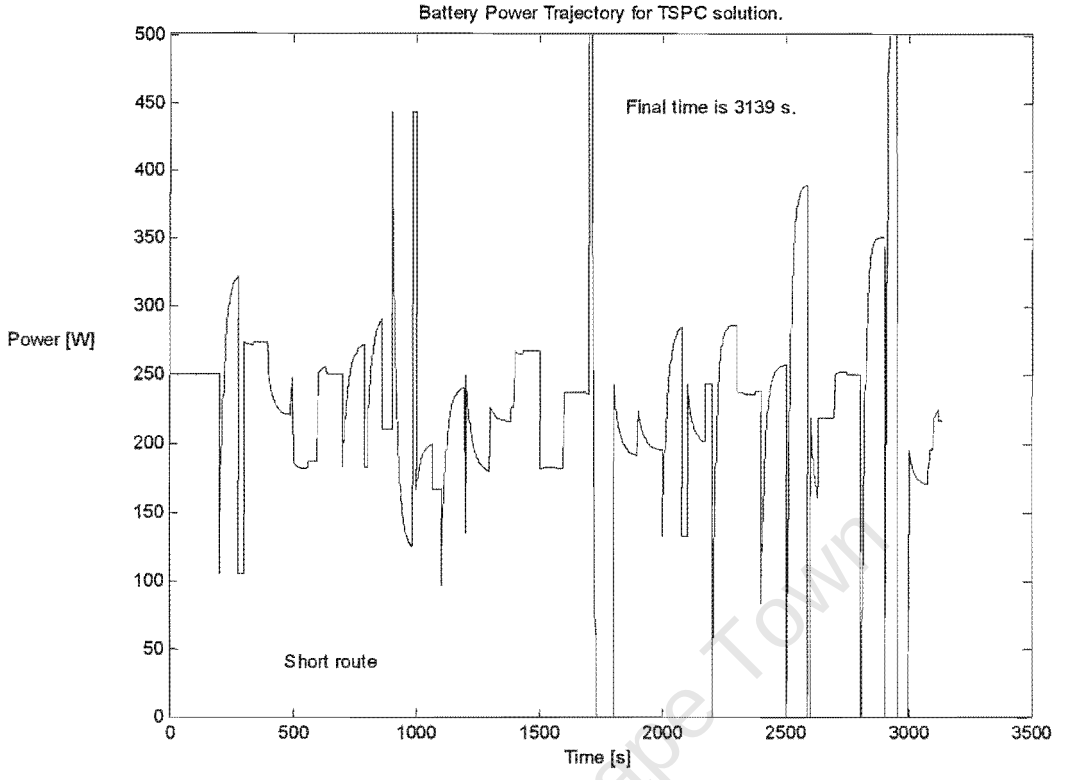



**Figure 26 Power profile for Simple Method with known wind.**

The power trajectory has fewer transients than the constant velocity power trajectories. This results in fewer accelerative impulses to the solar cycle and points towards better speed regulation occurring.

As a further check to determine whether a gross error exists, the energy expended is 4.32% greater than the available capacity. This is probably the cause of the improved final time. Considering this, the method produces a solution that is as good as the constant velocity controller solution.

### *ii. Nelder-Mead Method solver*

The Nelder-Mead solver gives a mixed set of results. For no wind, the final times are several minutes slower than the constant velocity result, although 18% of the battery energy has not been utilised.

In the known wind case, the final times are within a minute of the constant velocity times. The battery utilisation varies considerably, indicating that the solutions are not optimal.

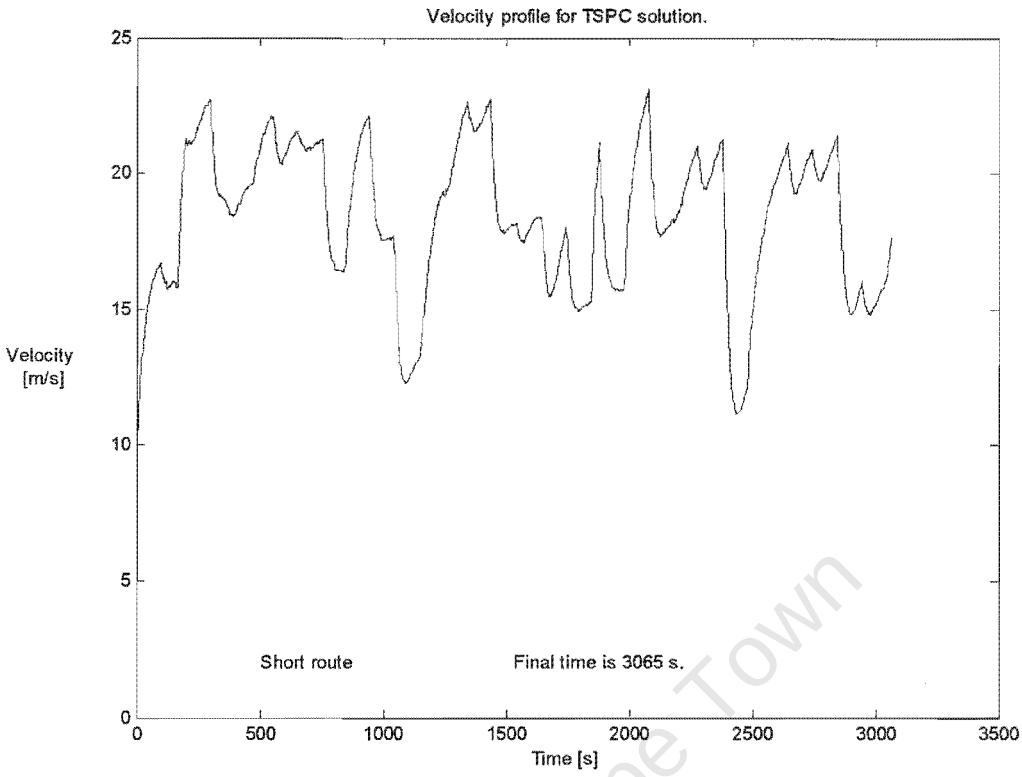



**Figure 27 Velocity Profile for Nelder-Mead solver with known wind.**

### *iii. Simple Solver 2*

The SS2 results are about 15% worse than the constant velocity results. The execution times are considerably quicker. The solver consistently under-utilises the battery energy.

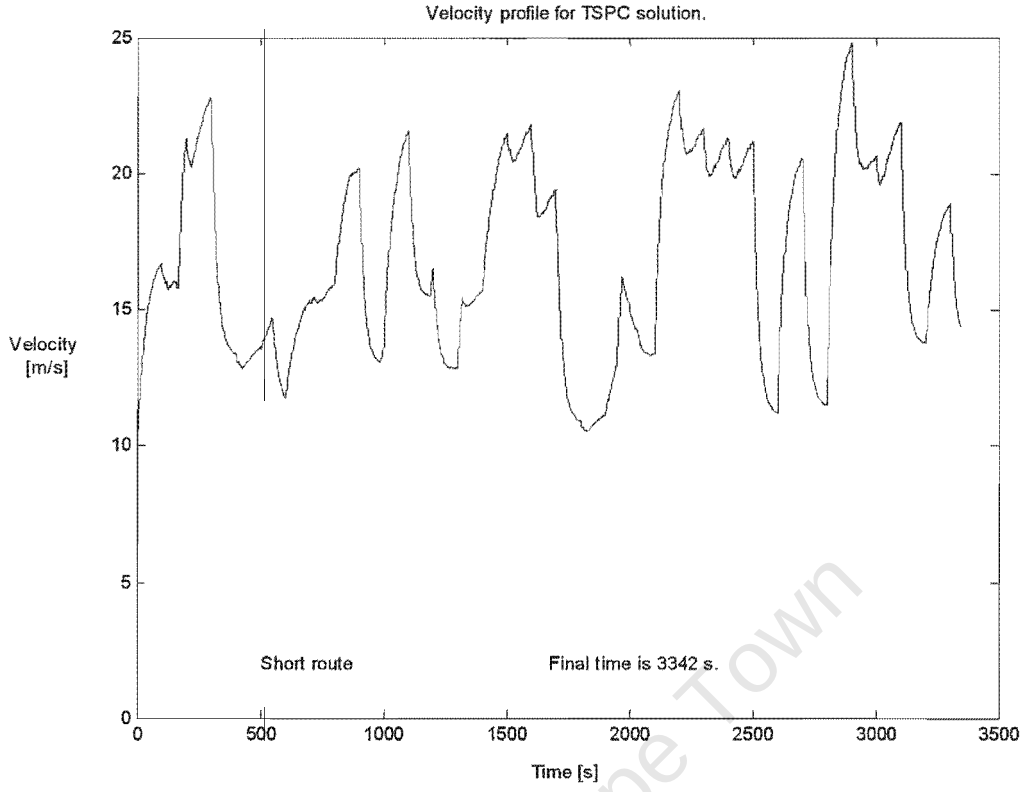



**Figure 28 Velocity profile of SS2**

Given the robust nature of SS2, and the good execution times it was selected for use in the prototype EMS. The assumption was that it would give a reasonable solution for any set of conditions met.

### 8.3.4 Insights gained during the design process

Algorithm convergence is a sizable issue. It became clear that a robust solver was preferable to a solver that could find an optimal solution in certain regions of the solution problem space but would not converge or give reasonable solutions in other regions. A robust solver would, at least give a reasonable solution for all regions of the problem space.

The Simple Method gives similar performance to the constant velocity controller for certain regions but requires the gain parameters be tuned to the particular region of interest.

The MTP is non-linear and gradient methods are not suitable for minimising the problem.

The Nelder-Mead method, incorporating the B-splines, is prone to being attracted to a false minimum in certain regions, while often stagnating in other regions.

The SS2 method fails to meet the energy constraint.

As none of the methods used are adequate for the task, a more robust non-linear numerical method is needed to minimise the MTP.

The TSPC appears to handle uncertainty very well.

Further insights are discussed in section 9.4, after discussing the implementation of the EMS.

# 9 The Implementation of the EMS

Only the first design objective was considered for the prototype EMS. Initially it was deemed desirable to host the entire EMS program within a DSP onboard the solar cycle. Commands would then be sent from a laptop in the support vehicle, via a radio RS-232 link, to the solar cycle. After seeing the times required to find a solution during the simulations, this approach was abandoned.

To speed development and ensure robustness, the EMS was split into two components. The lower level, time critical, functionality was embedded in the DSP, while the higher level, computational functionality was placed on the support laptop.

The discussion starts with the DSP side, before progressing to the laptop side and finishing with the electronics.

To extend the TSPC system beyond the simulation stage required the use of a near real-time data stream and some means of communicating with the TSPC code. The simplest means available is the RS-232 serial ports available on both the laptop and the DSP development board.

## *9.1 The DSP component of EMS*

The DSP used is the Texas Instruments TMS320F243 control optimised device. It was mounted on a MLT Drives development board. The device has a number of features that make it a good choice for the project:

- 6 PWM outputs
- 8 channel 10bit ADC
- 3 Capture inputs
- Flash program memory
- 20 MIPS performance
- Serial Communications Interface (RS-232 compatible)
- Flash boot loader
- ANSI C compiler

Further details are in the datasheet[29].

### 9.1.1 Functional Implementation of the DSP Component

This section presents the C program written to implement the required EMS functionally in the DSP.

#### a. Program Description

The algorithm structure follows that of two Texas Instruments (TI) Application Notes, which implemented, in C, a control system for a switched reluctance motor[30]. One used position sensors connected to the capture inputs and the other used a serial communications link to receive movement commands.

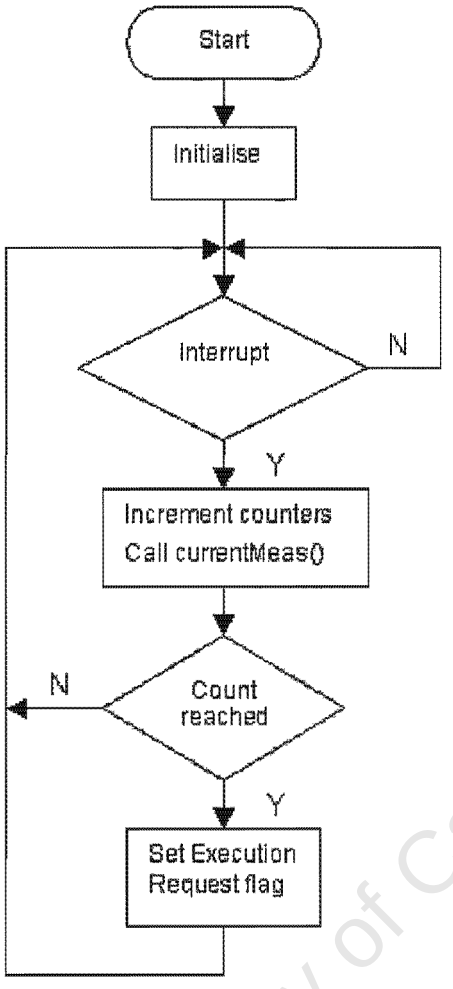



**Figure 29 Interrupt Service Routine flowchart**

A timing interrupt service routine (ISR) increments various task execution counters and calls the most frequent task. When a tasks counter has reached the desired count value a task execution request flag is set and the counter is cleared. Figure 29 shows the ISR flowchart from boot-up. It calls a single function the current measurement task, which is described below.

A scheduling routine runs in the background, and monitors the status of the task execution request queue and the task execution request flags. The scheduler will execute the task with the longest queue and then handle any task execution request flags. When a task execution request flag has been set, the scheduler increments the task execution queue and clears the task execution request flag. A detailed flowchart for the scheduler is in Appendix E.

This structure ensures that all requested tasks are executed. The scheduler could be flooded with execution requests if the execution cycle of a particular task is longer than the period between task requests. The motor current measurement task is the only one that could cause such a situation. By calling the current measurement task from the timing ISR, the problem is avoided.

The final HEX file is 11.7 KB.

### b. The Software Structure

This section describes the routines and file structure of the program. The program listings are given in Appendix E.

The structure used simplified the development of the program and allows the core structure to be used in another application with relatively little modification.

The following list shows the component files of the program and which routines are in each file. The header files `(.h)` are not discussed here, as they comprise prototype and data type declarations only.

Some of the files are modified versions of those used in the TI programs mentioned earlier. In such cases, the file headers clearly state the origin of the file and original author/s if available.

- `C243.h`
- `Scaffold.h`
- `Scaffold.c`
    - `Test1`
    - `GPT1_underflow`
    - `Test3`
    - `Test4`
    - `Test5`
    - `XINT2`
    - `ToggleLED`
    - `main`
    - `init`
    - `scheduler`
- `SCI.C`
    - `sci_init`
    - `sci_put`
    - `sci_get`
    - `send_packet`
    - `proc_cmd`
    - `prep`
- `Cycle.h`
- `Cycle.c`
    - `powerController`
    - `currentController`
    - `switch_mux`
    - `init_cycle`
    - `cutoff_motor`
- `Input.h`
- `Input.c`
    - `velocityMeas`
    - `currentMeas`
    - `read_a2d`

### *i. Scaffold.c*

`Scaffold.c` is the main file of the program.

The first six routines are ISRs.

`Test1` handles the interrupt generated when a character has been received by the serial port and placed in the receive buffer. The routine calls the `proc_cmd` routine to process the character.

`GPT1_underflow` is the timing ISR. On each underflow interrupt generated by timer one, it increments the task timing variables and sets any task execution request flags if the relevant timing variable has reached its threshold. The period of the timer is set in `init` and translates to a frequency of 10 kHz.

`Test3, Test4, Test5, XINT2` are null routines. Their declaration is required by the interrupt vector structure.

`ToggleLED` is a debugging routine. When called, it takes the byte passed to it and sends it to the digital I/O port, Port D. The interface board implemented Port D as an output port driving a LED array. This allowed sequential operations to be debugged.

`main` is the main program loop required in C syntax. It calls a number of initialisation routines before entering an infinite loop. The infinite loop runs in the background to the ISRs. It was used to debug numerous routines. In operational form, as listed, it calls the scheduling routine.

`init` is the first initialisation routine called by `main`. It sets the PWM registers, interrupt registers, timer registers, ADC registers and initialises the local variables used.

`Scheduler` is the scheduling routine that manages task execution. It checks the task execution request queue, implemented as an array, sets the execution flag for the highest priority outstanding request and decrements the queue. The relevant task is then called and the execution flag is cleared.

### *ii. SCI.C*

`SCI.C` contains the routines for handling the serial communications via the serial communications interface (SCI) of the DSP.

The communications consists of a data packet sent from the DSP to the laptop and a command string sent from the laptop to the DSP.

`sci_init` is the initialisation routine. The data format is set at eight data bits, odd parity and one stop bit. The baud rate is 19.2 kbaud.

`sci_put` places its calling argument, a character, in the transmit buffer when the buffer is empty. The character is then transmitted by the hardware.

`sci_get` retrieves a character from the receive buffer and returns the character to the calling function.

`send_packet` transmits the data packet to the laptop. When called it enables the SCI transmit output pin and calls `sci_put` for each character in the data packet.

`proc_cmd` handles the command strings. When called by the RX interrupt, it calls `sci_get` to obtain the received character. The character is passed to a finite state machine to determine if the character denotes the start of a command string. The next character in the command string is the command instruction and the following characters, if any, form the argument for the command.

`prep` refreshes the values of the data packet from the solar cycle state structure. It was also used while debugging to send other internal variables to the development PC.

**Table 11 Data packet structure**

| |
|---|
| STX |
| X |
| P |
| *Packet number* |
| A |
| B |
| *AB high byte* |
| *EF low byte* |
| C |
| D |
| *CD high byte* |
| *CD low byte* |
| C |
| A |
| *CA high byte* |
| *CA low byte* |
| C |
| B |
| *CB high byte* |
| *CB low byte* |
| E |
| F |
| *EF high byte* |
| *EF low byte* |
| ETX |

The italicised entries in the table are data values inserted by the `prep` function. The other entries are ASCII characters that add the needed structure to the packet. These characters are checked in the laptop component to determine whether the packet is valid.



### *iii. Cycle.c*

`cycle.c` implements the control functions for the solar cycle. Originally, the measurement functions in `input.c` were included here, but were moved to ease compilation.

The state of the solar cycle is held in a structure, which is passed to each function that can influence the solar cycle's state. The function will return the structure with the updated state.

`powerController` is the power control loop. Attempts to implement a PID controller failed, due to the computational load imposed by the floating-point arithmetic. A fixed-point solution did not yield accurate results. Finally, a simple gain term was used as the controller.

The output is kept within the limits of the allowed current.

`currentController` is the current control loop. Its development followed a parallel path to that of the power control loop.

The output is limited to the PWM period value used to set the PWM timer.

Included is the calculation of the power drawn from the battery.

`switch_mux` switches the ADC channel selection between three settings. It also sets the ADC start conversion flag.

`init_cycle` initialises the solar cycle state structure.

`cutoff_motor` terminates the PWM signals, resets the current reference values and the power reference value.

#### *iv. Input.c*

The measurement routines were moved to `input.c` from `cycle.c` to overcome a problem switching the ADC channel selection. While the first call to the switching routine works, a second call would not be implemented. The solution was to set the ADC control registers directly. However, compiling the code with symbolic debugging disabled removed the switching function call. Separating the routines from the rest of the control functions allowed the functions to be compiled using different compiler options.

`velocityMeas` measures the solar cycle velocity, the relative wind speed and the gradient.

`currentMeas` measures the current and voltage.

`read_a2d` returns the value from the relevant ADC FIFO buffer according to the its calling argument.

### 9.1.2 The Development Process of the DSP Component of the EMS

The development process was hampered by the lack of a debugger. One has to resort to toggling output pins and sending internal values to an 8-bit DAC. Later the situation was eased by using the serial communications link.

#### a. The Timer Interrupt Problem

After some difficulty in assigning the interrupt vector to accommodate the boot loader's reset interrupt and the interrupts used by the TI code, without losing the functionality of any of the interrupts, the author succeeded in finding a satisfactory assignment. However, the core functionality required, i.e. the regular timer interrupt was absent. No configuration of either Timer1 or Timer2 restored the interrupt. It was assumed that the fault lay in the interrupt vector assignment. As a result, the use of the TI code was abandoned.

Fortunately, Johan Malan and Johan Smuts of MLT drives had made some reference code available. Their interrupt vector performed as required.

#### b. The Capture Input Problem

Originally, the speed and gradient measurements were to use the capture inputs and associated timer.

Due to the asynchronous nature of the pulse trains, an ISR was required.

An unexpected difficulty arose when implementing the ISR.

Initially the ISR written to serve the capture interrupts typically failed to start correctly. On the occasions that it did start, the ISR performed as required. The need to press the reset button five to ten times in the attempt to start the capture ISR was deemed unacceptable. Much effort was expended attempting to find a functional configuration of the relevant registers.

A solution to the capture input initialisation was found. When the registers are initialised after the global interrupts have been enabled the ISR started correctly. Whether the problem was an initialisation conflict, interrupt conflict or something else is indeterminable.

The next difficulty arose as more demands were made on the DSP. The timing and execution of the speed and gradient tasks became erratic and yielded inaccurate results.

Finally an analogue solution was developed, in preparation of abandoning the direct digital approach.

### c. The ISR Location Issue

Another issue that became known during the development was the location of the ISR code. Originally the author had been attempting to list the ISR code in a separate C file with appropriate header file. The linker took exception to this approach and demanded that the ISR code be listed in the same file as the `main` function.

### d. Fixed-point Arithmetic Issues

As mentioned in 9.1.1.2.3, the use of floating point arithmetic placed a large burden on the DSP. The author was not able to develop accurate fixed-point routines to replace the floating-point routines. This compromised the performance of the controllers. However, the solar cycle is a high inertia load as seen by the motor, which mitigates the performance loss.

The current and voltage measurements followed the routine suggested by the TI application notes by using an infinite impulse response filter to smooth the readings.

### e. Compiler Issues

Care needed to be exercised when selecting the compiler options.

In particular, the optimisation options when more stringent than the default settings, stripped out the infinite loop that handles the non-critical tasks.

The option to enable program level optimisation is ignored by the compiler and optimiser.

The compiler options also had a direct effect on the ADC channel selections. Any level of optimisation removed the channel selection instructions from the assembled code. The ADC routines were listed in a separate file, `inputs.c`, and compiled separately with symbolic debugging enabled (`-g`). By enabling symbolic debugging a number of default optimisations are inhibited in the compilation process.

Frequently the compiler would omit a section of code, leaving the program without a certain function or partial implementation of the function. The usual fixes included altering the calling sequence of functions and routines, shuffling calculations and altering the compiler optimisation settings. The problem may have been a memory issue, i.e. the program was larger than the program memory space. This has not been confirmed.

## 9.2 *The Laptop Component of the EMS*

The laptop component was implemented on a Mecer 600 MHz Pentium 3 notebook computer with 128 MB RAM.

### 9.2.1 The Functional Implementation of the Laptop Component of the EMS

To perform the TSPC calculations, a reasonably powerful laptop was required. This is partly because of performing the calculations within the MATLAB environment and partly because of the computational load. The MATLAB environment was used to speed development, rather than spend considerable effort writing a dedicated Windows application. A laptop was hired for the duration of the field test. MATLAB was installed and the TSPC code copied across.

Of the suite of toolboxes and GUI environments available in MATLAB 6, only the Instrumentation Toolbox was used. It provides an interface to communicate with via GPIB, VISA or serial connections. The EMS used the serial component of the toolbox.

The algorithm and software structure were derived from those used in the simulations, discussed in section 8. As a result, the descriptions are similar. The differences are largely omissions of functionality in the files optimised (hence the `_opt` name extension) that was included in the simulation functions. Most of the omissions relate to the omission of solar power and scheduled cyclist changes, neither of which is applicable to the field test.

#### a. A Description of the Program

The TSPC algorithm has been discussed at length in section 8.3.3.

The basic steps are:

1. Measure the system states.
2. Generate forecasts.
3. Solve for optimal trajectories for each forecast.
4. Select most accurate trajectory.
5. Repeat from step 1.

What has been changed is the first step actually measures the states, as opposed to simulating them.

### b. The Software Structure

This section describes the routines and file structure of the program. The program listings are given in Appendix E.

The program structure is described in the following list. The list starts with the script file that launches the program. The functions that it calls are listed below it. The process is followed for each of the function files used.

MATLAB syntax dictates that each function with universal scope must be declared in a separate file. Sub-functions do not have scope beyond the file in which they are declared.

- `trial2.m`
  - `laptop_init.m`
  - `datalogger4.m`
    - `dataformat.m`
  - `TSPC_SS2_opt.m`
    - `TSPC_solver_opt2.m`
      - `cycle_opt.m`
        - `cycle_DE_wind_opt.m`
        - `cyclist_opt.m`
    - `LMSerror.m`
  - `laptop_term.m`

#### *i. trial2.m*

This MATLAB script file implements the TSPC. It starts by calling the laptop initiation script file `laptop_init.m`. Then the first data set is captured from the solar cycle and the solar cycle state trajectories are initialised.

The main loop starts by incrementing the dataset counter.

The solar cycle state trajectories, route parameters, time and wind state are passed to the TSPC solver shell, `TSPC_SS2_opt.m`, which solves for a single step and returns a control value and the expected remaining route.

The control value is transmitted to the DSP and the next data set captured.

The state trajectories are updated and the loop repeats until the end of the route has been reached.

The shutdown command is then transmitted and the serial link is terminated via `laptop_term`.

#### *ii. laptop_init.m*

This script file handles the initialisation of the serial link and some of the TSPC variables.

Firstly, the route is selected and loaded into memory. The route selection also determines the battery capacity used.

The initial control value is set and a number of other variables initialised.

The serial link initialisation comprises declaring the serial port object, setting the parameters and opening the object for communication.

The turn-on command is then transmitted, followed by a delay, and then the control value is transmitted.

### *iii. datalogger4.m*

The `datalogger4` function collects the data packets sent from the DSP and returns the solar cycle state trajectories, the wind state and the time duration of the trajectories.

After initialising the local variables, the function waits for fifty characters to be stored in the receive buffer. Ideally, these fifty characters will contain two data packets; otherwise, at least one will be in the fifty characters.

The receive buffer is then searched for the packet header, which is three characters long. When a header is found, the following twenty-five characters are read into a raw data set. This raw data set is passed to the `dataformat` function, which returns the data set allocated to the relevant variables and an error flag.

If the error flag is set the data packet is rejected, else the data set is appended to the local data logs. This process repeats until twenty data packets have been processed.

The local data logs are then assigned to the relevant return variables, e.g. the state trajectories.

As a failure tolerance mechanism, the local data logs are saved in a file with the filename determined by the dataset counter.

### *iv. TSPC_SS2_opt.m*

The `TSPC_SS2_opt` function implements the core of the TSPC algorithm.

The first and last steps of the algorithm are handled externally in `trial2.m`.

After some manipulation of the state variables and the generation of the forecasts, the optimal trajectories for the forecasts need to be found. The trajectories are limited to a subset of the route based on a forecast horizon of two kilometres. The trajectories are found using calls to the `TSPC_solver_opt2` solver.

To select the most accurate trajectory, the first twenty seconds of the forecast velocity trajectories are compared to the most recent twenty seconds of recorded solar cycle velocity trajectory. The comparison is made in `LMSerror`, which returns an index indicating which forecast produced the trajectory with the closest match. This forecast's control value is then returned to `trial2.m`.

The selection procedure is slightly different to that used in the simulation. It is a concession to the short time period (20s) of each calculation cycle (steps 1-5). This potential error source would be a problem if the solver had to find a control trajectory, but since the solver used finds a constant control value, a larger source of error is introduced.

### *v. laptop_term.m*

This script stops and closes the serial communications link and deletes the serial link object.

### *vi. TSPC_solver_opt2.m*

The `TSPC_solver_opt2` solver has been discussed previously, in sections 7.3.6.1 & 8.2.3. To recap, the basic procedure is as follows:

5. Calculate initial velocity trajectory using the `cycle_opt` function and expanding the given initial control value into a control trajectory.
6. If any point on the velocity trajectory exceeds the upper or lower bounds, decrease or increase corresponding point in the control trajectory.
7. Check the energy balance for the forecast horizon, and raise or lower the whole control trajectory as required to achieve the balance.

8. Recalculate the velocity trajectory using the `cycle_opt` function and the control trajectory derived above. Return the control trajectory to the calling function.

The solver was constructed to iteratively improve the control trajectory, but the convergence rate was too low and the iterative loop was removed. Hence, the solver is now a single step procedure.

#### *vii. LMSerror.m*

The differences in the forecast trajectories and the recorded trajectory are measured using the least mean squared error method (LMS). The LMS function is implemented as a sub-function.

#### *viii. cycle_opt.m*

`cycle_opt` returns the solar cycle's state trajectories given the control trajectory, route and initial conditions.

Since the current system ignores solar power, the control trajectory is added to the expected cyclist power trajectory, modelled in `cyclist_opt.m`, to obtain the trajectory of the total available power.

To calculate the trajectories, the state equations are integrated, using a simple Euler method, over the length of the total available power trajectory.

#### *ix. cycle_DE_wind_opt.m*

`cycle_DE_wind_opt` is the functional implementation of the solar cycle state equations. It returns the time derivatives of the states.

#### *x. cyclist_opt.m*

The cyclist's power output is modelled as a decaying exponential curve, with the starting value and decay rate dependant on the cyclist. In the field test optimised implementation, the model was restricted to using a single, good cyclist and the handling of cyclist changes was removed.

#### *xi. dataformat.m*

The raw packet data is in byte-sized pieces and needs to be placed in a more MATLAB friendly format. First, the data packet structure is checked for validity. If the structure is not correct, the error flag is set. The relevant data values are extracted from the data set and assigned to return variables.

### 9.2.2 The Development Process of the DSP component of the EMS

The bulk of the development process is described in section 8.

The remainder consisted of developing the serial communication link and integrating the link into the TSPC script.

#### a. Serial Communication Link

The MATLAB script `datalogger2.m` implements the communications protocol needed to extract the data from the received packets, and track the packet numbers. Initially, the data values from missing packets were replaced by zeros to aid debugging. The script also sends a command instruction to the DSP. As the command sending is implemented inside the main loop in the script, various waveforms can be sent to the DSP, e.g. ramp and step functions.

This development phase proved to be straightforward.

### b. Integration of TSPC and Serial Communication Link

The integration of the TSPC and serial link was handled in a progressive manner. First, the two script files were merged together while keeping the functionality separate. Then the serial link was altered to report the solar cycle state variables in the same manner as the simulated solar cycle state variables. The final step was to remove the simulated solar cycle functions and use the actual values.

The last step involved cycling the prototype around the block until MATLAB produced an error message and sounded a warning chime. The errors were corrected on the roadside and the ride continued.

## *9.3 Auxiliary Instrumentation and Power Electronics*

This section describes the electrical hardware built for the prototype. It includes the measurement instrumentation to the EMS, the battery, the motor and the motor's DC-DC converter.

On the DSP side, all the I/O signals are routed through an interface board. This board connects directly to the IDC headers on the DSP development board and provides the necessary signal conditioning for the various sensors and measurements. A number of the functions are performed on small, dedicated boards mounted close to the system with which they interact. Examples are the high side switch for the motor, the gradient sensor, the radio transceiver and the solar cycle velocity measurement.

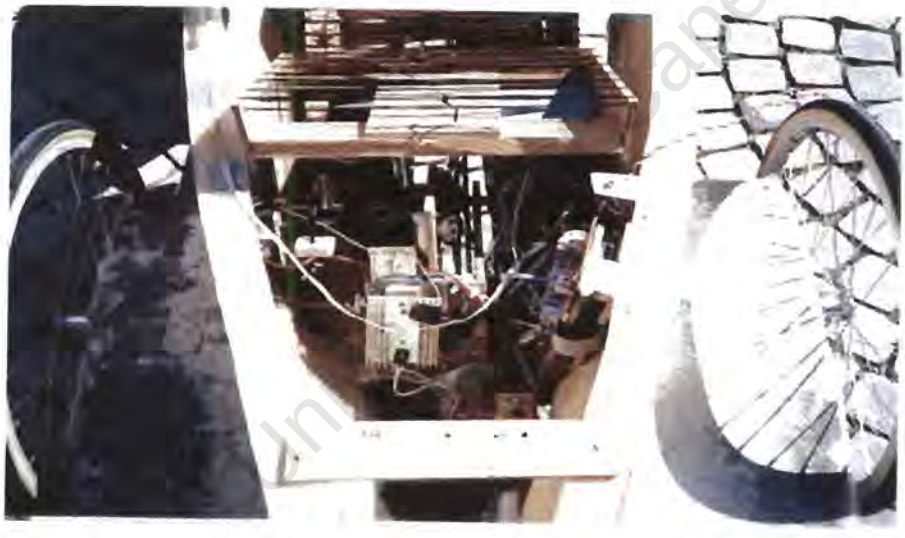


**Figure 30 Photo of EMS installation in Prototype**

The complete circuit diagrams are presented in Appendix G. The figures in this section are for illustration. Figure 30 shows the installation of the EMS, motor and batteries behind the seat in the Solar esCape prototype.

### 9.3.1 The Radio Telemetry Link

To run the prototype system with the laptop mounted on the solar cycle would be difficult and risky. The safest option, and the most suitable, is to implement the radio link that would be used in the final system. Then the laptop would be located in the support vehicle.

#### a. The Radio Transceivers

The radio transceivers selected are the nRF401 units from Nordic. They operate in the 433 MHz ISM band. A short list of their features is:

- 20 kbaud data throughput
- 2 channels available
- Very low external part count

For more detail, refer to the datasheet[31].

The transceivers are surface mount technology (SMT) devices and the reference application circuit uses SMT passive components. The reference circuit uses a loop antenna etched onto the PCB. Included in the reference application is the artwork for a two layer PCB and related bill of materials. The author used this design as the basis for a single layer PCB design, using the same SMT components. The artwork was drawn by hand using an etch-resistant pen directly onto the board. This was the quickest and simplest method available. Rigorous continuity testing was required to ensure the integrity of the PCB.

The one disadvantage of using these transceivers is that the communication becomes half-duplex, losing the full duplex capability of a wire serial link. Given that the data flow consists of discrete packets of information, with the larger packets going from the solar cycle to the laptop and small command packets in the opposite direction, one of each packet being sent per second, the loss of full duplex capability will not impact on the system performance. It is more of a hardware issue, in that the transceiver needs to be switched from transmit to receive mode and visa versa.

One problem experienced when implementing the radio link was getting the crystals on the radio transceiver ICs to oscillate. The problem is assumed to be stray capacitance in the etched PCB. Crystal oscillator modules were used instead.

The second problem was that the radio link did not work when tested. Not having the correct test equipment hampered faultfinding efforts and the radio link was abandoned.

#### b. The Laptop to Transceiver connection

To perform the transmit/receive mode switch, existing handshaking lines on the serial port on the laptop were used. The transceiver was also powered from the serial port, drawing 24 mA when transmitting.

#### c. The DSP to Transceiver connection

The transceiver is mounted on a separate board. To aid positioning of the antenna the lines are relatively long.

One of the digital output lines on port B was used to perform the transmit/receive mode switch. The signal, power, channel select and mode lines were all connected to the interface board.

### 9.3.2 The ADC Input Protection

The ADC inputs of the DSP are sensitive to voltage swings beyond the DSP supply rails. The inputs to the ADC are clamped with a 5.1 V zener diode and a schottky diode, to provide protection against transients and accidental shorts to the rails. The protection diodes can be seen in figures 31 & 32 below.

### 9.3.3 The Current Measurement Circuit

The current measurement was made using a hall-effect current transducer, a LEM LA-100P.

The transducer is positioned to monitor the return from the motor to the battery.

Originally, it was planned that the voltages and currents on both sides of the converter would be measured. This approach is intended to ease the later integration of the MPPT functionality in the DSP. However, the idea was abandoned when the difficulties with the ADC channel selection occurred.

Some analogue signal processing is required for the current signals output from the transducer. It is illustrated in figure 31. The output of the current transducer is first converted to a voltage using a resistor. It is then buffered using a unity gain inverter. The inverted signal is then inverted in an inverting amplifier and scaled to match the ADC input parameters. The signal is then sent to an integrator to determine the average current drawn.

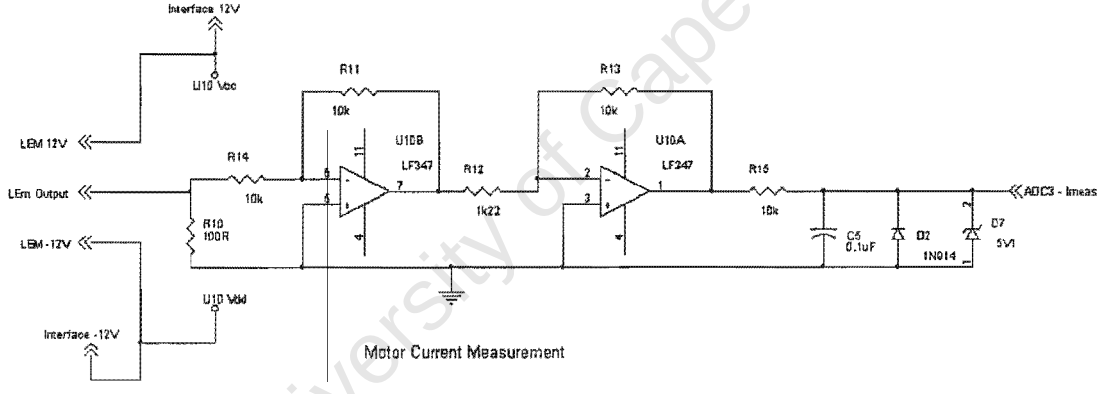


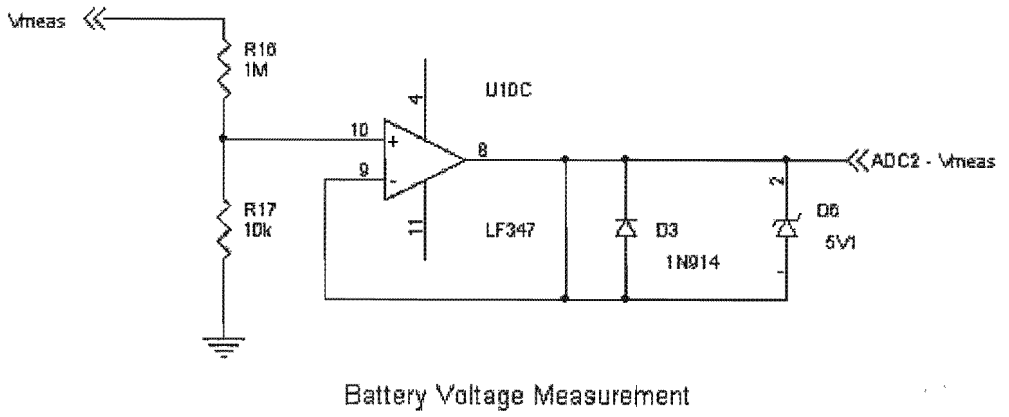

**Figure 31 Current Measurement Circuit**

At low signal levels, the ADC quantisation error caused large errors in the current levels indicated.

### 9.3.4 Voltage Measurements

The voltage measurement used voltage divider to reduce the voltage level, and a voltage follower to buffer the DSP from the battery.

The voltage measurement signals are taken directly to the interface board.




Battery Voltage Measurement

**Figure 32 Battery Voltage Measurement Circuit**

Figure 32 shows the circuit used.

### 9.3.5 The Gradient Measurement Circuit

The gradient measurement is made by a dual axis accelerometer from Analog Devices, the ADXL202. It has a range of +-2*g*, with a resolution of 5 m*g*. The bandwidth is adjustable up to 5kHz. It also features an analogue and a digital output for each axis.

The accelerometer performs two functions in the solar cycle. Its primary function is to measure the gradient and output it to the DSP. The second function is to measure the lateral acceleration of the solar cycle and warn the cyclist when the solar cycle is close to its limit when cycling through curves. This is purely a safety device.

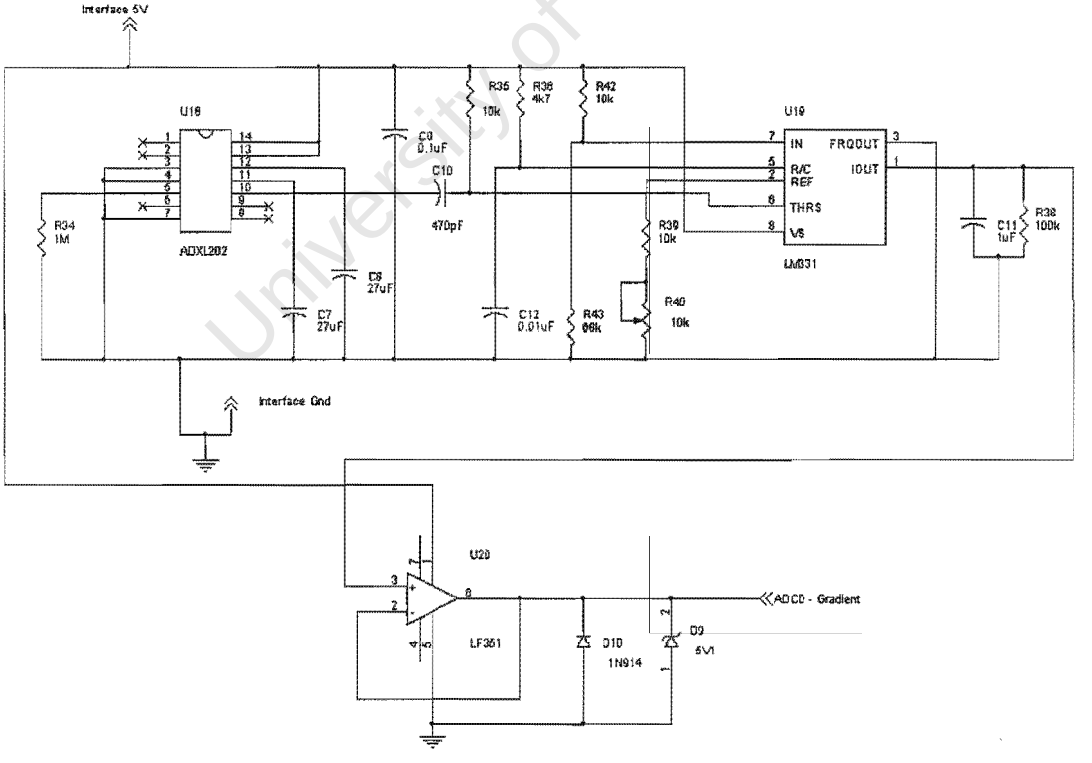



**Figure 33 Gradient Measurement Circuit**

The prototype has been tested to establish its behaviour. The limit where the inside wheel lifts is 0.5*g* of lateral acceleration, or 23 kph around a 5m radius turn. The

behaviour on the limit is quite stable and benign, a slight reduction of speed or opening of the turn radius causes the inside wheel to drop again.

The accelerometer is mounted such that one axis is aligned on the solar cycle's longitudinal axis and the other on its lateral axis.

Originally, it was planned to connect the both axis outputs directly to the capture inputs of the DSP, minimising additional components and generally being more elegant. However with the problems experienced with the capture inputs, an analogue approach was developed. A LM331 was configured as a frequency to voltage converter. The output was buffered by a voltage follower and the ADC protection diodes. This configuration is shown in figure 33.

The measurement frequency is set at 50Hz and smoothed inside the DSP.

The accelerometer is a SMT device and is mounted on a separate board. This allows it to be mounted as accurately as possible to minimise measurement offsets.

### 9.3.6 The Speed Measurement Circuit

The speed measurement is based on counting the number of spokes that pass through an optical limit switch. The frequency output is passed to a LM331 based frequency to voltage converter before being passed to the ADC input. As illustrated in figure 34, the output of the limit switch was first passed to a comparator with a small hysteresis on the input. This was to reduce chance of noise causing false pulses.

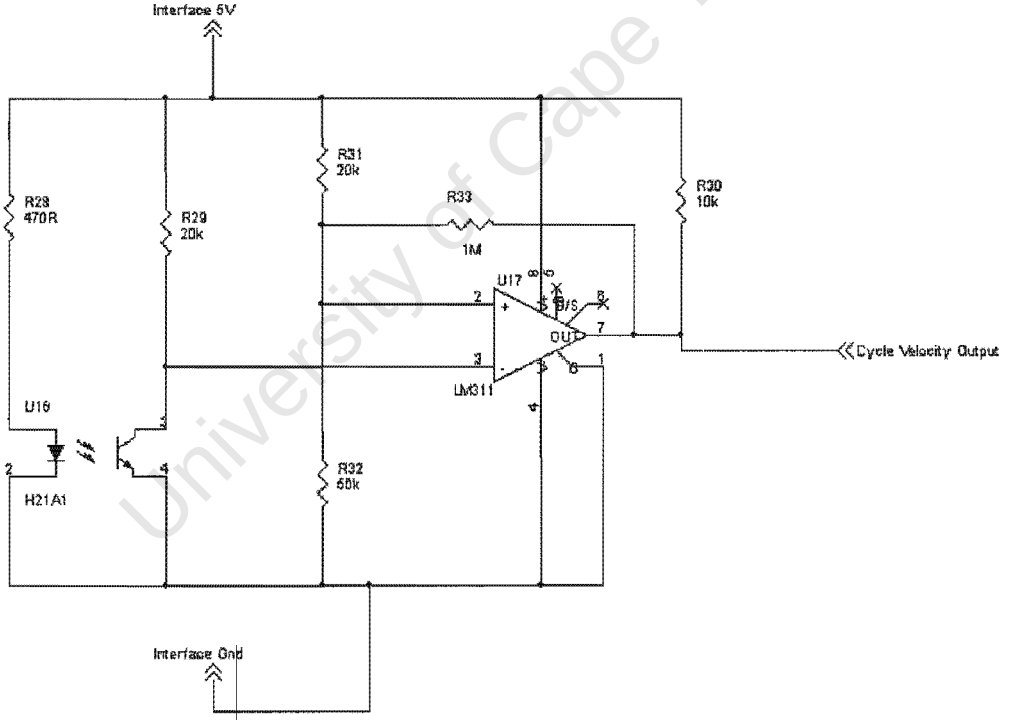



**Figure 34 Front End of Velocity Measurement**

Of necessity, the limit switch is split over two boards. To eliminate any possible drive train induced slip, the solar cycle velocity measurement is made at the front wheel. This is also the easiest mounting point on the prototype.

The actual frequency to voltage conversion circuit is located on the interface board. Figure 35 shows the LM331 circuit and the integrator on the output. The 555 is configured to generate a fixed width pulse when triggered by the rising edge of a pulse. This was intended to improve the linearity of the measurement.

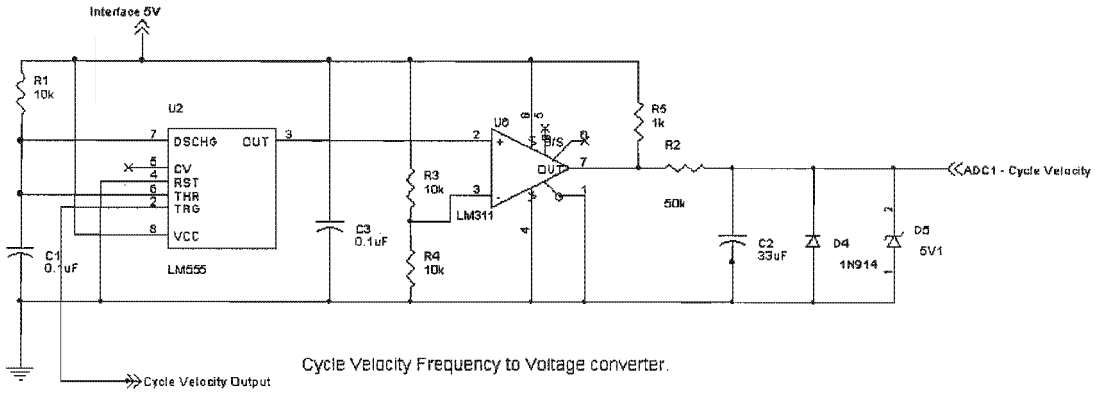



**Figure 35 Signal Processing of Velocity Signal**

The ADC protection diodes can be seen on the output, after the integrator.

Calibration attempts were not successful. The results appeared to be non-linear and erratic. The problem was later diagnosed as excessive vibration of the limit switch halves leading to spokes not being detected, or false pulses being created.

## 9.3.7 The DC-DC Converter

The DC-DC converter used in the prototype is a simple step-down converter without snubbing. The converter topology is suitable, since only single quadrant operation is required. Figure 36 shows the circuit, including the master switch, voltage measurement point and current transducer. The inductor is the same one used in WestCape its inductance has not been measured.

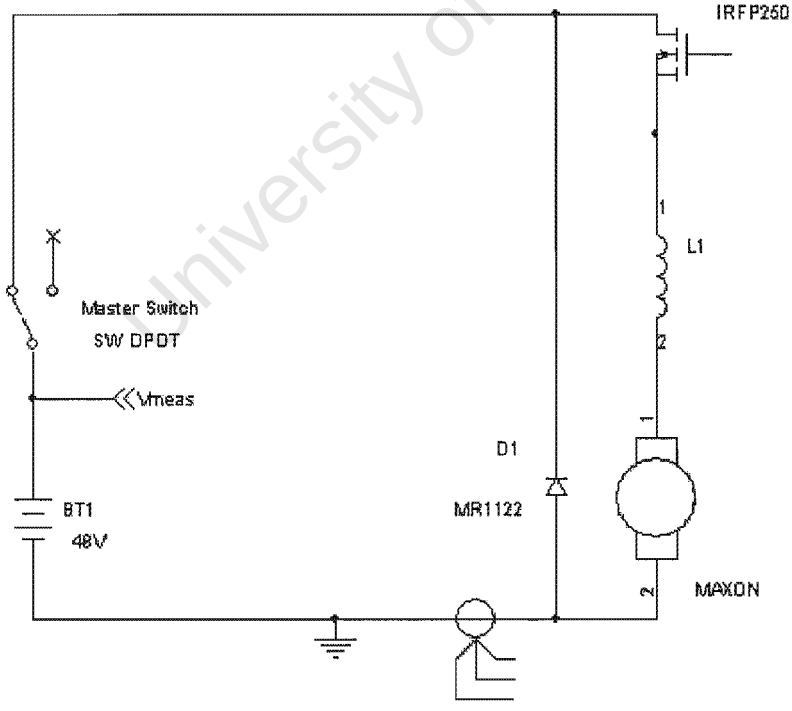



**Figure 36 DC-DC Converter Circuit**

The switching frequency is set in the DSP code.

The design and construction of a more efficient converter is beyond the scope of the project. It will probably only be finalised when the motor and battery selections for Solar esCape have been made.

The high side switch is an IRF250 power MOSFET capable of switching a 30 A load from a 200V rail. It has a low on resistance of 0.075 ohms. The integrated bypass diode is a fast recovery type.

The first high side driver selected was the IR2113 high and low side MOSFET driver. The IR2113 accepts standard TTL logic level inputs and bootstraps the high side internally.

Both the MOSFET and driver are ex-stock items chosen for their ease of use.

This simple solution did not operate as expected. The problem was the under-voltage lock out mechanism being activated at duty ratios above about 0.25 in the no-load condition (for the motor). This behaviour might have been tolerated, had it not given rise to another problem. When the UVLO was active, the gate voltage rose to about 4 V above the source, exceeding the gate-source threshold. This caused the MOSFET to conduct in its linear region and dissipate about 10 W of heat. This was deemed as unacceptable, and a floating high side supply was constructed.

The floating supply provided by a flyback converter using a TL494 PWM control IC. The TL494 has two integrated output transistors and output control allowing parallel or push-pull operation of these transistors. This allows a quick, simple implementation of a two-transistor flyback converter. The voltage feedback is achieved using two voltage dividers and an op-amp configured as a differential amplifier. When fly charged, the battery terminal voltage is 52 V, adding a possible 25 V (see below) from the floating supply gives a possible maximum voltage of 77 V, which is outside the ratings of most op-amps. Fortunately, complete electrical isolation is not required, allowing the use of the voltage dividers.

While the TL494 can be configured for current mode control, it was not implemented, as the regulation demands on the converter are very lenient. The IR2113 accepts a maximum high side supply voltage of 25 V above the MOSFET source voltage. The UVLO activates between 7.0 and 9.7 V depending on polarity of the voltage transition. With a $V_{GS}$ of 10.0 V, the MOSFET is saturated at the drain voltage and currents under consideration. Therefore, as long as the supply voltage exceeds this level, the switch will operate properly. On the bench, the floating supply functioned properly when driving a dummy, resistive load. When connected to the high side of the IR2113, retaining the dummy load for protection, the supply promptly destroyed the IR2113.

After this experience, the decision was made to construct a driver.

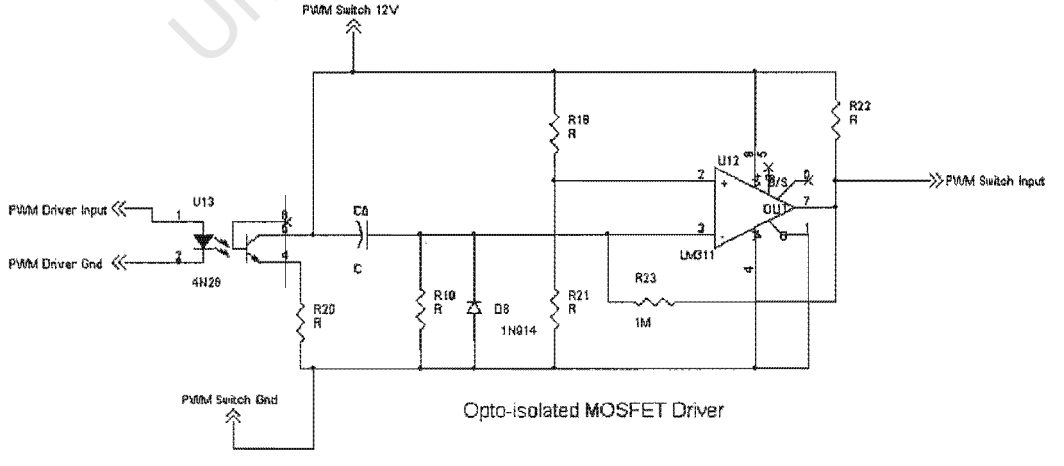



**Figure 37 Opto-isolated MOSFET Driver Circuit**

Mohan *et al* devote an entire chapter to the topic of gate and base drive circuits[32]. Their comparator based MOSFET gate drive with a totem-pole gate drive circuit[33] was then implemented, with a 4N26 optocoupler providing electrical isolation of the

control signal and the power supplied by the floating supply. The addition of a large feedback resistor provides a small hysteresis to the input. This configuration was functional from the start; it is illustrated in figure 37. It is limited in the switching speeds that it can maintain, as the output is of the open collector type. However, by adding a complementary output stage, the switching speeds and current capacity can be increased. The high pass filter and DC restoring diode were needed to ensure that the pulses returned to zero, allowing the low threshold value to be used.

### 9.3.8 The Battery

The battery used for the prototype is a series string of four sealed lead-acid batteries of 6Ah capacity. This results in a 48V battery with a nominal capacity of 288 Wh. This is 28.8% of the allowed capacity of 1 kWh.

Given that the average stage length of the WSCC is about 220 km, a field test route between 50 and 60 km in length would simulate racing conditions.

The length of the test route had been set at 45 km based on the nominal capacity of four sticks of ten NiCd D cells of 4Ah capacity in series. However, the NiCd cells would not retain their charge for more than a day. With the uncertainty in the winter weather, the sealed lead acid batteries offered greater flexibility in selecting when to run the field tests.

### 9.3.9 The DC Motor

The motor used in the prototype is the same motor used in WestCape, a brushed DC motor manufactured by Maxon.

In fact, the whole drive train was moved to the prototype, with a change of gear ratios. Apart from some initial alignment problems, the drive train functioned flawlessly.

## *9.4 Insights gained during the implementation process*

Project management of a project relies on prior experience, however when the entire project is a new development, estimating the time to complete tasks is difficult and uncertain.

By being systematic, and patiently testing each component and component system prior to integration and through each level of integration, the most of the errors and faults were found early and the least amount of time wasted. This held for both the hardware and the software implementation.

The difficulties of implementing the optimal control solver prompt speculation on alternatives. The problem is relatively simple in dynamics terms, but the constraints add a severe non-linear aspect to the problem.

Having gained good insight into the problem and the dynamics of the solar cycle, one can now focus attention on dealing with the non-linearity in the problem.

Rule-based systems, dismissed in section 6.2, could be structured to handle the problem. Both fuzzy logic and neural networks are touted as being able to handle non-linear problems.

Using a rule-based solver in the TSPC algorithm could yield the performance desired, with the same, or lower, computational load as the currently implemented system.

The development of a rule-based system would be facilitated by the body of simulations that provide benchmarks for the performance of the system and a functional code base.

The simple solver 2 (SS2) used in the functional implementation is the simplest beginnings of a rule-based system. It could be readily extended to a Mamdani type fuzzy logic system.

# 10 Field Testing the EMS

The plan was to thoroughly test the prototype solar cycle (prototype) and EMS prior to the test. The goal was to run one test only. As the test would require the participation of the number of Solar escape team members for a number of hours and the hire costs of the laptop and two-way radios. As a result, the simulation was performed after the field test.

Prior testing exposed a number of problems:

- Chain derailments
- Loose motor mounting
- Loose bearing mountings
- Incorrect seat position
- Flat NiCd batteries
- An offset in the steering
- Inadequate brakes
- Poor field of view for the cyclist
- Solar cycle velocity measurement error.

The first five items were remedied. The steering and brakes require the fitment of a new front fork. The field of view was deemed adequate for travelling in a straight line. The velocity measurement appeared to be erratic and non-linear, the problem had not been rectified before conducting the field tests.

## *10.1 Description and Route*

The battery pack has sufficient capacity for about fifty kilometres under race conditions, thus only one cyclist is required.

To simulate conditions close to those that are found on the first few days along the Stuart Highway in Australia, a relatively straight, flat road was needed. Several of the cyclists in the team suggested the West Coast Road (R27) between Cape Town and Springbok. It offered one advantageous feature, in that it has a wide tarmac shoulder.

The starting point selected was the intersection of the Koeberg Power Station road and the R27. The section of road just past the intersection was relatively level, allowing the prototype to stand without wheel chocks.

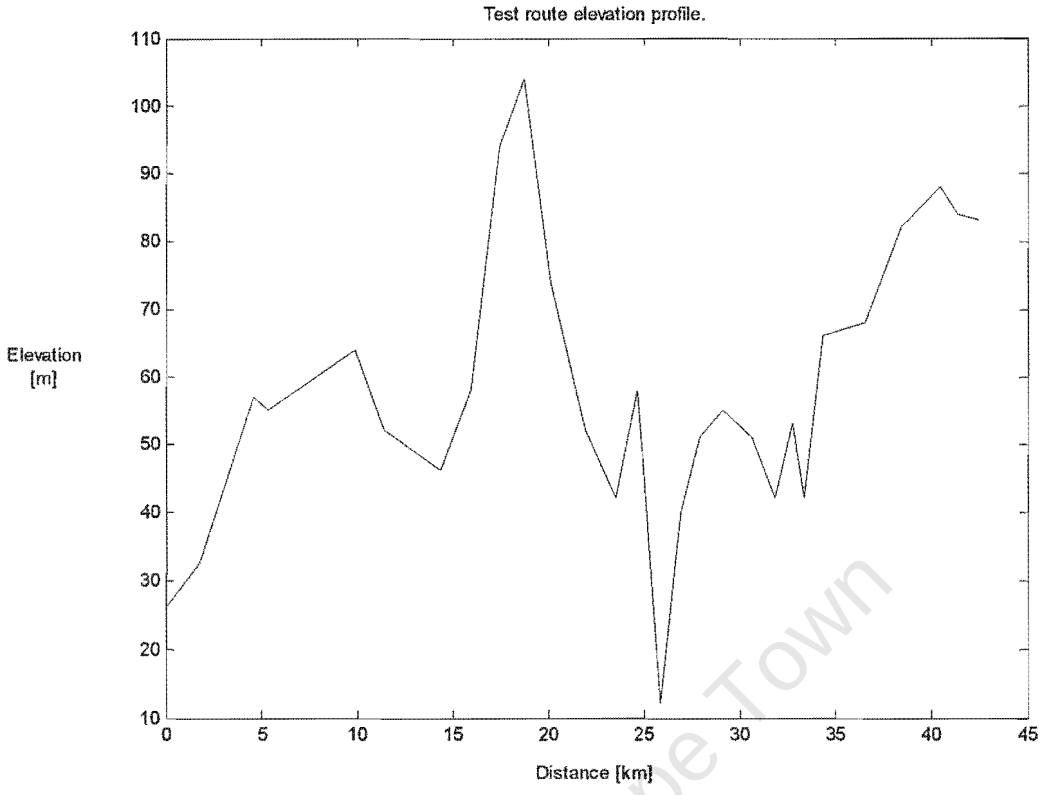



**Figure 38 Elevation profile of test route**

## *10.2 Empirical Results*

Two test runs were attempted to assess the EMS performance.

These were scheduled for the first week of August and were run on the two days of reasonable weather that occurred during the week.

The cyclist scheduled to ride the prototype withdrew on the morning of the first test, forcing the use of an unfit cyclist.

### 10.2.1 First Test

The first test run started at 10h00 Thursday 1 August 2002.

#### a. Weather

The forecast weather for the week showed two fine days[34].

```
WEST COAST OF WESTERN CAPE PROVINCE, LAMBERT'S BAY TO YZERFONTEIN
26 09/19 Partly cloudy. Wind: Strong northwesterly.
27 11/15 Cloudy, showers mainly morning (70%). Wind: Strong westerly.
28 06/16 Cloudy, rain overnight (50%). Wind: Fresh northwesterly.
29 07/15 Cloudy, showers mainly morning (50%). Wind: Fresh southwesterly.
30 06/16 Partly cloudy becoming fine. Wind: Moderate southerly.
31 05/18 Fine. Wind: Moderate northerly.
01 07/18 Fine. Wind: Moderate northwesterly.
```

The sky was clear with the temperature about 17 °C. The prevailing wind was about 20 kph from the northeast. The wind was gusty and increased in strength over the duration of the test.

### b. Cyclist's Notes

Traffic was light. The greatest problem was the strength sapping uphill section of the route. The start was relatively easy, but the speed dropped on the incline.

The low speed prevented the establishment of a comfortable cadence.

The prototype started well, but the power eased off within the first minute.

Wind gusts caused some yawing, but nothing of concern.

### c. Results

The Cateye cycling computer recorded the following:

- A time of 15 minutes 3 seconds
- An average speed of 11.3 kph
- A maximum speed of 17.1 kph
- A distance of 2.83 km

The figure below shows the recorded data form the first test run; the curves are scaled by their calibration factors. These variables are from the saved MATLAB workspace and include the data received from the prototype.

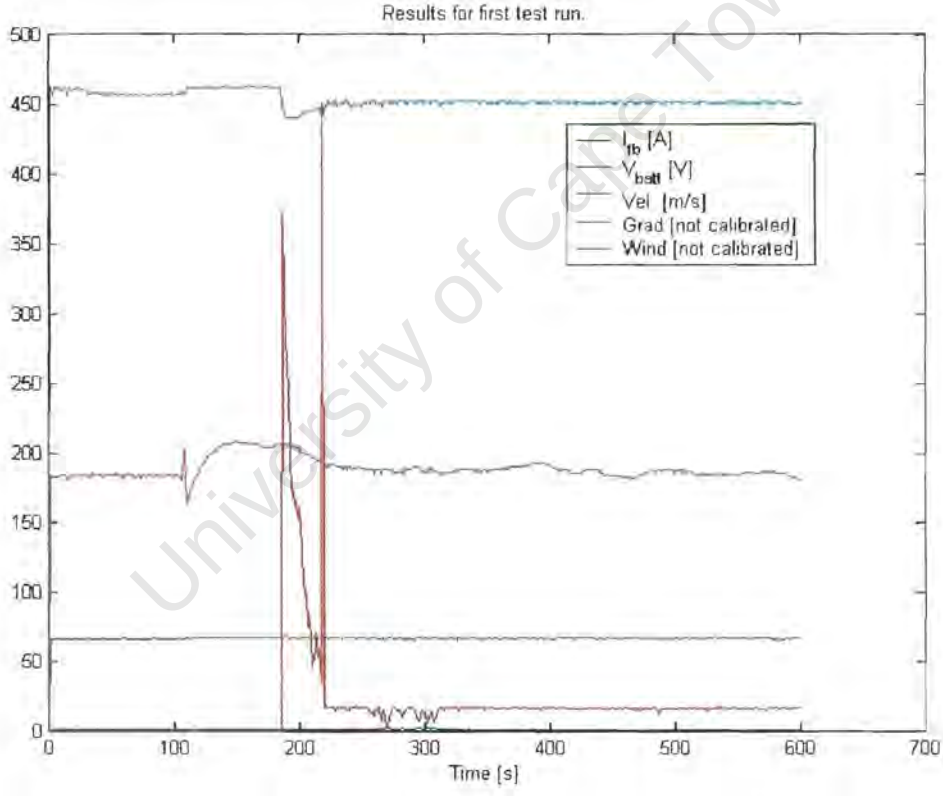



**Figure 39 Results for first test**

The DSP was turned on at zero seconds. The cyclist entered the prototype and then connected the wind speed measurement circuitry at 109 seconds.

The motor was turned on at 187 seconds, when a suitable gap appeared in the traffic.

The prototype reaches 17 kph within the first few seconds and drops thereafter. At 219 seconds the velocity reading spikes and drops to a low level. Up to that point, the velocity had been following an exponential curve for the previous 30 seconds, with the prototype travelling at about 12 kph. The gradient curve shows a small spike. The

velocity reading drops to almost zero at two points, however the prototype did not drop below 8 kph during the test run according to the cyclist. At 311 seconds, an event occurs that fixes the velocity at a constant value (with some noise). The motor current reading also drops off to its minimum level from this point.

The MATLAB component halted after 600 seconds when the Windows 98 power management function put the laptop on standby, despite being disabled.

After about fifteen minutes of cycling the test is terminated.

#### d. Discussion

The low assistance level from the motor cannot be explained by examining the data. The starting power level is 250 W for the first twenty seconds. However, the first twenty seconds are spent waiting for a gap in the traffic. Perhaps this caused a variable interval to the TSPC functions to saturate. The motor did assist at a few points, so it was clearly not an electrical fault.

In the workshop, inspection showed that the high side switch battery had worked loose from its mounting and been lost in transit. A replacement was strapped in using duct tape. The inspection was limited to checking the battery conditions, as the prototype was kept tied to the trailer in preparation for another test run.

### 10.2.2 Second Test

After the first test, it was decided to run a second test at the first opportunity provided by the weather. This decision was based on the short distance travelled and strong wind experienced in the first test.

The second test run was started at 10h43 Saturday 3 August 2002.

#### a. Weather

The weather was similar to the previous test, with a more favourable wind of about 15 kph from the east. The wind was steady for the duration of the test.

#### b. Cyclist's Notes

The motor hum was almost continuous; no check was made to determine whether the motor was actually providing any assistance.

The higher speed caused the canopy to jump around in spite of the locating pegs.

The traffic was considerably heavier.

The test run was halted after about 30 minutes. The uphill section was travelled at an average 20 kph and the downhill section at about 33 kph.

The uphill section was worse than any hill seen during the first three days of the WSCC.

#### c. Results

The Cateye cycling computer recorded the following:

- A time of 30 minutes 43 seconds
- An average speed of 22.8 kph
- A maximum speed of 35 kph
- A distance of 11.71 km

The data recovered from the saved data log files does not make much sense. The wind velocity and prototype velocity readings are the same. The gradient reading is a series of pulses. The current reading is two-thirds of the voltage reading. The figure below shows a section of all the curves, as reconstructed, without scaling.

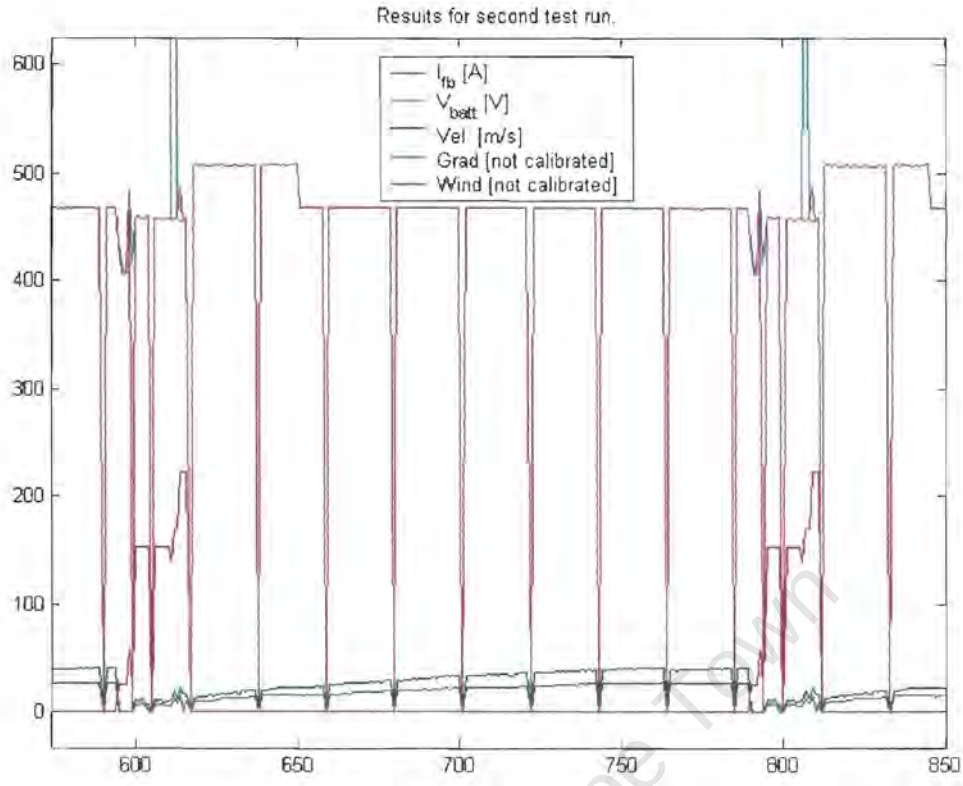



**Figure 40 Results for second test**

The spikes to zero are artefacts from combining the log files, as the first value in each file is a zero and any missed data packets result in a zero. They are retained to keep the waveforms synchronised.

Clearly, something was faulty.

### d. Discussion

After halting the test run, the laptop failed to switch on the LCD display when the finger pad was activated, a key pressed or the power button pressed. It was assumed that the Windows 98 power management system had activated again and was not responding to the wake-up instruction. The battery was removed and inserted and the laptop booted up again. This resulted in the MATLAB workspace data being lost and the only available data being the recorded data files from the prototype.

On return to the workshop, it was noticed that the emitter part of the prototype velocity measurement limit switch had been torn off. This explains the velocity reading problems in the first test run and the difficulty in calibrating the velocity measurement. The limit switch halves were attached to the nail heads using an epoxy adhesive. The detector part had fallen off in one of the previous short tests and been reattached. It is possible that the levels of vibration seen by the limit switch were sufficient to cause the emitter and detector to swing out of alignment and generate a false pulse. The spike noticed in the gradient curve coincides with the spike in the velocity curve. This bump probably caused the emitter to start working loose, as seen by the lower velocity readings, before falling off and being torn out if circuit by the spokes. This would explain the sudden levelling of the velocity reading. The velocity measurement circuitry then settled to a steady state value.

This raises the possibility that another vibration induced failure or fault prevented the system from functioning correctly.

No further tests or diagnostic checks have been made to determine if and where such a fault or failure exists.

## *10.3 Simulated Results*

Due to the incomplete bodywork, the prototype's aerodynamics is not as good as expected when running the previous simulations. The prevailing weather and time constraints had not allowed the measurement of the actual CdA of the prototype. The CdA was estimated from the data of the first test run and used in these simulations.

### 10.3.1 Estimation of Drag Area

By changing the cyclist model, using the gradient profile and using the forecast wind velocity, it is possible to estimate the power required to maintain a given speed in the prototype. The figure below shows the road load or total power requirement for one set of estimates for the CdA and the cyclists' ability.

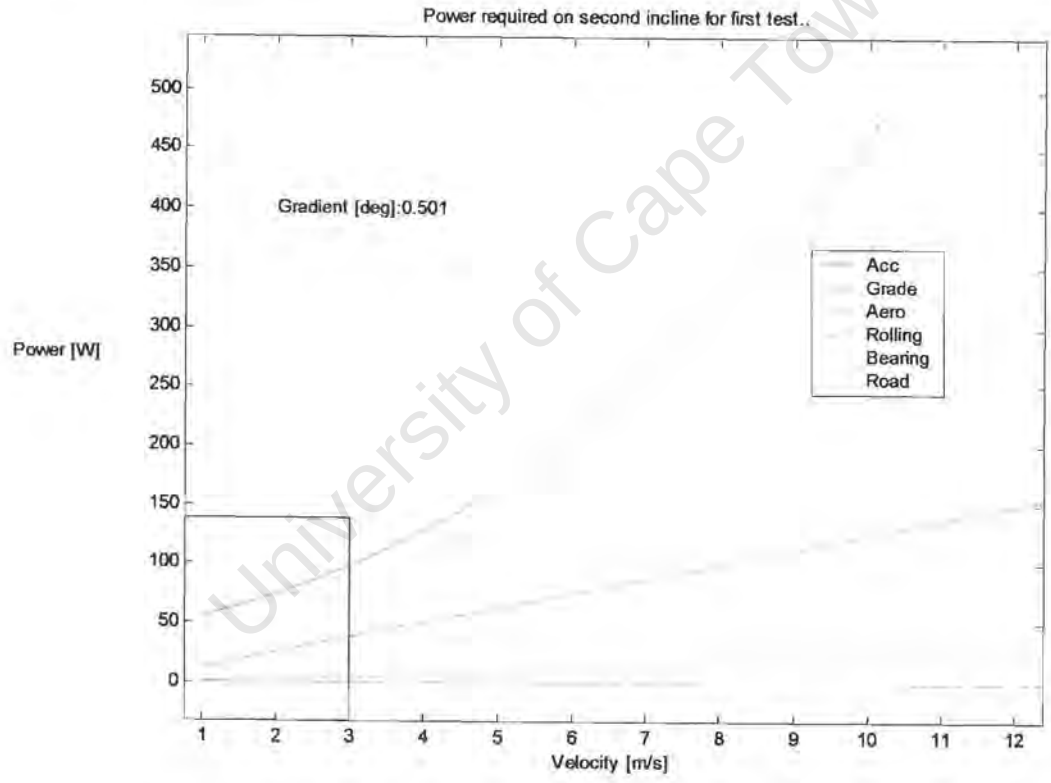



**Figure 41 Power required for various velocities on second incline of first test run.**

For a CdA of 0.20 and the cyclist able to generate a constant power of 125 W the prototype will be able to travel at about 3 $m.s^{-1}$ or 10.8 kph. This includes the 25 kph NE wind and no assistance from the motor.

Using the same set of estimates with the wind conditions on the second test run produces a consistent result, as illustrated in the following figure.

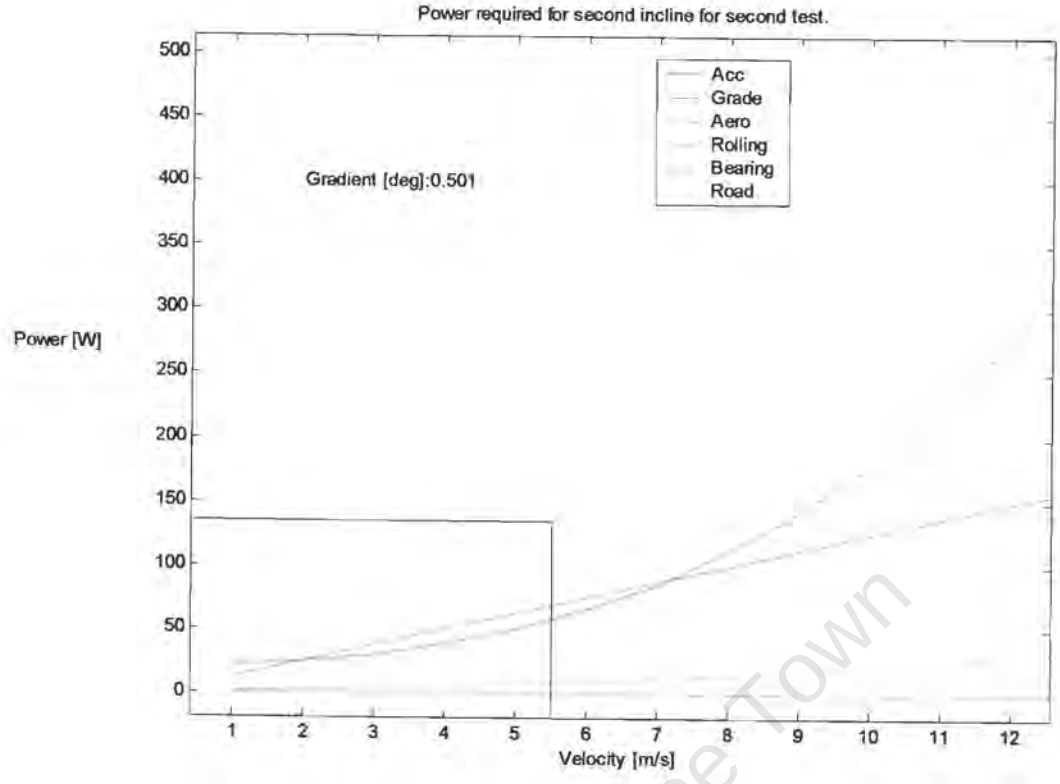



**Figure 42 Power required on second incline of second test**

For 125 W of power the prototype can now maintain about 5.5 $m.s^{-1}$ or 20 kph. These results are consistent with the recorded speeds of the prototype on the second incline between 2.5 and 5 km on the route. The estimated CdA is reasonable and the estimated cyclist output is reasonable.

From these results, one can conclude that the EMS did not function correctly on either of the test runs. It may have provided a marginal level of assistance, but nothing greater than 25 W, allowing for some uncertainty in the current measurement.

### 10.3.2 Simulations of Test Route

Two simulations are presented below, the first is a look at the performance of the prototype had the wind not been present on the test runs, and the second looks at the expected performance had the EMS been functional for the second test run.

#### a. Test route with no wind and no assistance

The results are shown in figure 43. The mean velocity is 5.4029 $m.s^{-1}$ or 19.45 kph. The time to complete the route would be 2647 seconds or about 45 minutes. The gradual decline in the velocity corresponds to the gradual decline in the cyclist's power output.

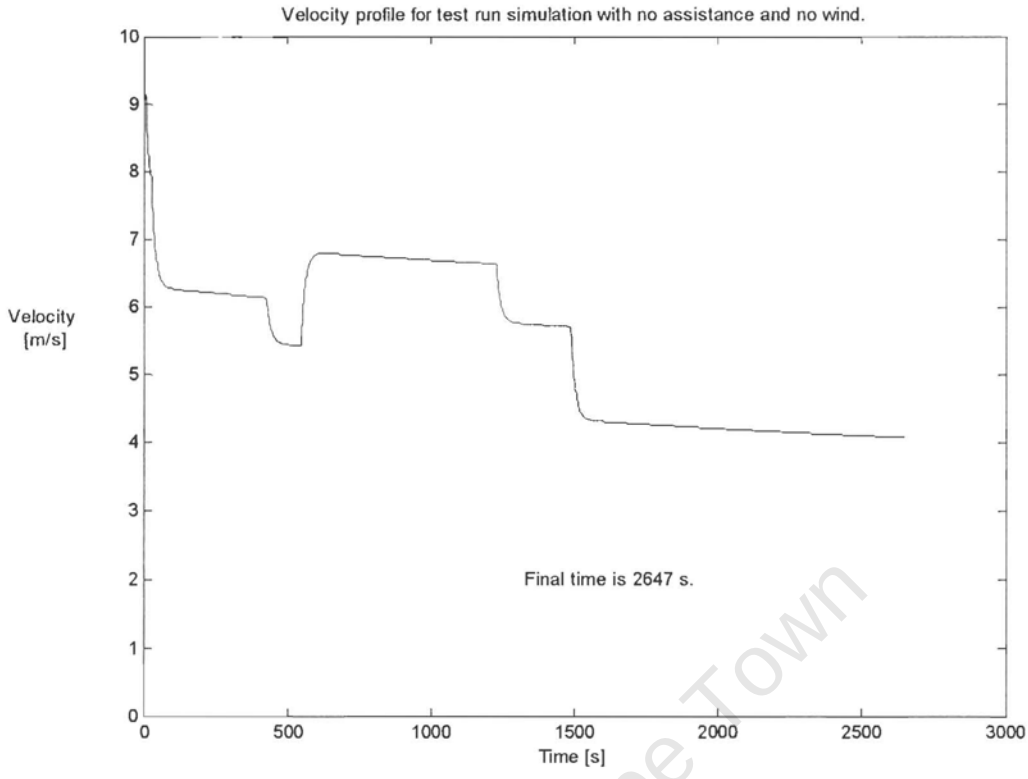



**Figure 43 Velocity trajectory for test route simulation with no assistance and no wind.**

### b. Test route with assistance and wind

The results for the simulation of the EMS reveal a flaw in the system. The battery utilisation is 298%. This confirms that the SS2 solver is not adequate for the task. As illustrated in the figure below, the mean velocity is 9.4532 $m.s^{-1}$ or 34 kph.

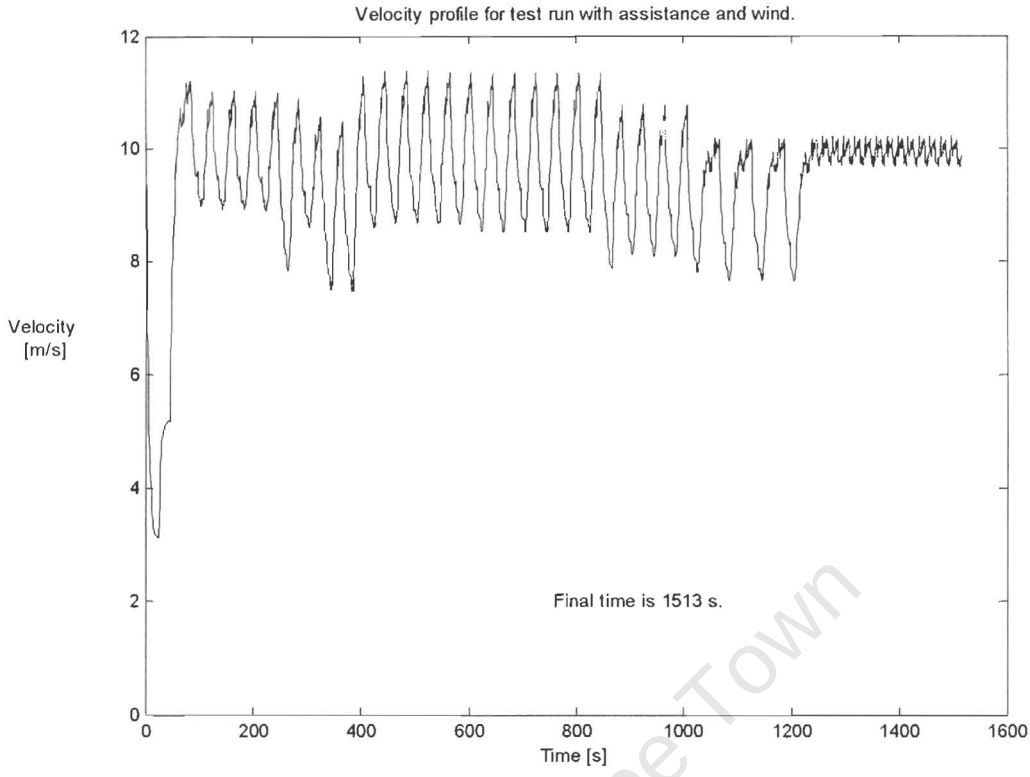



**Figure 44 Velocity trajectory for test route simulation with assistance and wind.**

The route is completed in 1513 seconds or about 25 minutes. The velocity seems to oscillate throughout the simulation, looking at the battery power trajectory in figure 45 shows that the power is in fact oscillating.

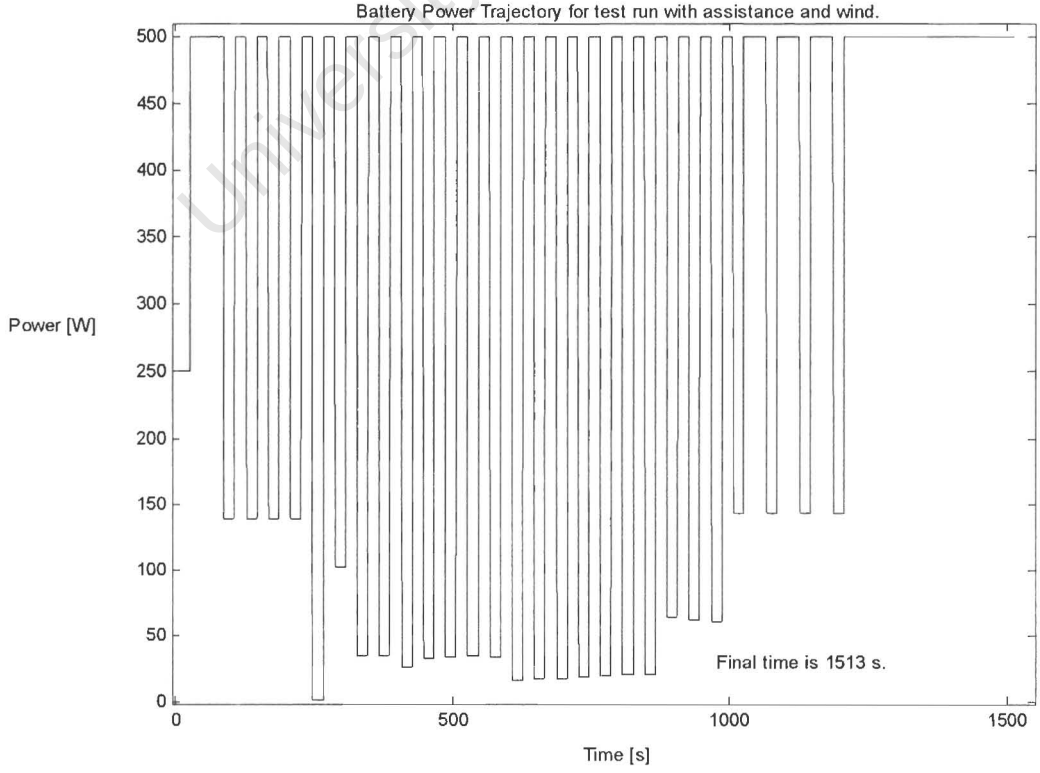



**Figure 45 Power trajectory for test route with assistance and wind.**

This oscillation indicates an additional problem in the current implementation of the EMS.

Figures 46 & 47 are two photographs of the Solar esCape prototype. The white tab on top of the canopy is the wind speed measurement fence.

## *10.4 Conclusions drawn from Field Testing*

The EMS suffered from one confirmed electronic failure and at least one suspected fault or failure. The first occurred during the first test run, while the second appears to have prevented the EMS from functioning correctly in the second test run.

The simulations suggest that the EMS did not function as expected during the test runs. Hence, the design objectives have not been achieved.



**Figure 46 Solar esCape Prototype**

Figure 47 Prototype with canopy removed

# 11 Recommendations for Future Work

Drawing from the insights gained during the project and the results of the field tests the following recommendations are made for the continued development of the Solar esCape EMS. They are listed in descending priority.

1. The electronic systems are rebuilt or redesigned to accommodate the vibration levels in their operating environment. Some forensic analysis of the existing electronics will provide a good starting point.
2. The TSPC is retained in the EMS.
3. A rule-based solver is developed for the EMS.
4. A robust non-linear method is developed to solve the MTP – a thorough literature review of MTPs should give some examples to study.
5. A DSP with an integrated development environment is used – many of the implementation problems would have been easier to spot and overcome using a debugger.
6. The second design objective, i.e. the tactical component, is incorporated in the EMS when the above recommendations have been implemented.

# 12 Conclusion

The conclusions reached in this dissertation are grouped into three areas pertinent to the development of Solar esCape.

## *12.1 The use of optimal control theory*

The use of optimal control theory gave a number of insights into the behaviour of the cycle. In particular, it showed that the battery energy must be depleted as the solar cycle reaches the end of the stage and that the use of regeneration does not offer any advantage.

The optimal trajectory for a given set of conditions has still not been found. This is largely a numerical method issue. The MTP is non-linear and gradient methods are not suitable for minimising the problem. A robust numerical method that can handle non-linear problems is needed. As such, the question of the best approach to controlling the velocity of a solar cycle is unanswered.

The constant velocity approach appears to be the closest to optimal when all environmental inputs are known.

The alternative approaches to designing a controller for the energy management system need to be revisited.

## *12.2 The performance of the TSPC*

The TSPC is able to compensate for the uncertainty inherent in a noisy input, in this case the prevailing wind.

## *12.3 The performance of the prototype EMS*

The empirical results showed that the EMS suffered from a failure in the velocity measurement electronics. An additional electronic failure or fault is suspected.

The field test simulations suggest that the EMS would not perform as expected had it functioned.

The design objective, to use the available energy to complete the day's stage in the minimum time has not been reached.

## *12.4 Summary*

The question of the best approach to control the velocity of a solar cycle has not been answered. This prevented the design objective for the EMS from being reached.

The prototype EMS suffered from at least one electronics failure when tested.

Therefore, the objective of this dissertation has only been partly met.

# References


[1] Storey, J.W.V, A.E.T. Schinckel and C.R. Kyle, Solar Racing Cars, 1993 World Solar Challenge. Canberra: Australian Government Publishing Service, 1994.

[2] Roche, D.M., J.W.V. Storey, A.E.T. Schinckel, C.P. Humpris and M.R. Guelden, Speed of Light, 1996 World Solar Challenge. Sydney: Photovoltaics Special Research Centre, 1997.

[3] Roche *et* al, p217.

[4] Tamai, G, The Leading Edge, Aerodynamic Design of Ultra-streamlined Land Vehicles. Cambridge MA: Robert Bentley Inc., 1999.

[5] Pudney, P., Optimal energy management for solar-powered cars, PhD thesis in Applied Science, University of South Australia, 2000. <http://scg.levels.unisa.edu.au/Pudney/thesis/>

[6] Storey *et al*, p191

[7] Putney, p.6

[8] Hemer, Ray. Technical Chief and Chief Scrutineer for the WSCC 1999.

[9] Storey *et al*, p190 – 191

[10] Subscribe via wsc.egroups.yahoogroups.com

[11] Zorpette, G., "Sun Kings Cross the Outback", IEEE Spectrum, February 2002, p40-46

[12] Bryson, A.E. and Y. Ho, Applied Optimal Control, Optimization, estimation and control, Waltham, Massachusetts: Blaisdell Publishing Company, 1969.

[13] This is a well-established field of study. Two works referred to during the course of this project are those by Faires & Burden and Roberts & Shipman.

[14] Bryson and Ho. p225. Chapter 7 deals with range of methods.

[15] Schwartz, A.L., Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems, PhD thesis in Electrical Engineering and Computer Sciences, University of California at Berkley, 1996.

[16] Optimization Toolbox, User's Guide, Ver. 2.0, The MathWorks Inc, 2000.

[17] Wolfe, M.A., Numerical Methods for unconstrained Optimisation: An Introduction, New York: Van Nostrand Reinhold, 1978. p.131.

[18] Kelley, C.T. "Iterative Methods for Optimization: MATLAB Codes", <http://www4.ncsu.edu/~ctk/matlab_darts.html>, 1999. `nelder.m`

[19] Schwartz, p75.

[20] Wolfe, p.85.

[21] Kelley, `gradproj.m`

[22] Malinowski, K., "Repetitive Optimization for Predictive Control", Computational Optimal Control. International Series of Numerical Mathematics, Vol. 115. Basel: Birkhauser Verlag, 1992, p163-175.

[23] Pudney p57.

[24] Kelley, `gradproj.m`.

[25] Wolfe, p.80.

[26] Lagarias, J.A., M.H. Wright and P.E. Wright, "Convergence propreties of the Nelder-Mead Simplex Method in low dimensions", SIAM Journal of Optimization, Vol. 9, No. 1., 1998, p.112-147.

[27] Malinowski, K., "Repetitive Optimization for Predictive Control", Computational Optimal Control. International Series of Numerical Mathematics, Vol. 115. Basel: Birkhauser Verlag, 1992, p163-175.

[28] Schwartz, p76

[29] TMS320F243, TMS320F241 DSP CONTROLLERS, Datasheet SPRS064C, Texas Instruments, 2000.

[30] Fedigan, S.J. and C.P. Cole, A Variable-Speed Sensorless Drive System for Switched Reluctance Motors, Application Report SPRA600, Texas Instruments, 1999.

[31] 433MHz Single Chip RF Transceiver, nRF401, Nordic VLSI ASA, 1999. <http://www.nvlsi.no>

[32] Mohan, N., T.M. Undeland and W. P. Robbins, Power Electronics, Converters, Applications and Design, 5th ed., New York: John Wiley & Sons Inc., 1995. See Ch 28 p696 - 729.

[33] Mohan *et al* p699.

[34] South African Weather Service, <http://www.weathersa.co.za>

# Bibliography


Alter, D., Using the Capture Units for Low Speed Velocity Estimation on a TMS320C240, Application brief SPRA363, Texas Instruments, 1997.

Andersen, J.A. and E. Rosenfeld, Talking Nets, An Oral History of Neural Networks, Cambridge, Massachusetts: The MIT Press, 1998.

Beber, D., DSP Controlled Three Phase to Single Phase Uninterruptible Power Supply, MSc Dissertation in Electrical Engineering, University of Cape Town, 2001.

Bryson, A.E. and Y. Ho, Applied Optimal Control, Optimization, estimation and control, Waltham, Massachusetts: Blaisdell Publishing Company, 1969.

DiRenzo, M.T., Switched Reluctance Motor Control – Basic Operation and Example Using the TMS320F240, Application Report SPRA420A, Texas Instruments, 2000.

Faires, J.D., R.L. Burden, Numerical Methods. Boston: PWS Publishing Company, 1993.

Hariri, A. and O.P. Malik, "A Fuzzy Logic Based Power System Stabilizer with Learning Ability", IEEE Transactions on Energy Conversion, Vol. 11, No. 4, December 1996, p.721-727.

Horowitz, P. and W. Hill, The Art of Electronics, 2nd ed., Cambridge, England: Cambridge University Press,1989.

HV Floating MOS-Gate Driver ICs, Application Note AN978, International Rectifier, Undated.

Instrument Control Toolbox, User's Guide, Ver. 1.0, The MathWorks Inc, 2000.

IR2110/IR2113 High and Low Side Driver, Datasheet PD60147-P, International Rectifier, Undated.

IRFP250N, Power MOSFET, Datasheet PD94008, International Rectifier, 2000.

Jain, A.K., J. Mao and K.M. Mohiuddin, "Artificial Neutral Networks: A Tutorial", Computer, March 1996, p.31-44.

Johnson, R. A., Miller and Freund's Probability and Statistics for Engineers, 5th ed., Englewood Cliffs: Prentice-Hall, 1994.

Kelley, C.T. "Iterative Methods for Optimization: MATLAB Codes", <http://www4.ncsu.edu/~ctk/matlab_darts.html>, 1999.

Kuc, T-Y, S-M, Baek and K.Park, "Adaptive learning controller for autonomous mobile robots", IEE Proceedings on Control Theory Applications, Vol. 148, No.1, January 2001, p.49-54.

Lagarias, J.A., M.H. Wright and P.E. Wright, "Convergence propreties of the Nelder-Mead Simplex Method in low dimensions", SIAM Journal of Optimization, Vol. 9, No. 1., 1998, p.112-147.

Linkens, D.A. and H.O.Nyongesa, "Genetic algorithms for fuzzy control, Part 1: Offline system development and application", IEE Proceedings on Control Theory Applications, Vol.142, No.3, May 1995, p.161-176.

Linkens, D.A. and H.O.Nyongesa, "Genetic algorithms for fuzzy control, Part 2: Online system development and application", IEE Proceedings on Control Theory Applications, Vol.142, No.3, May 1995, p.177-185.

Malinowski, K., "Repetitive Optimization for Predictive Control", Computational Optimal Control. International Series of Numerical Mathematics, Vol. 115. Basel: Birkhauser Verlag, 1992, p163-175.

McNeil, P. and P. Freiberger, Fuzzy Logic, The Revolutionary Computer Technology That is Changing our World, New York: Touchstone, 1993.

Milliken, W.F., and D.L. Milliken, Race Car Vehicle Dynamics, Warrendale: Society of Automotive Engineers Inc., 1995.

Mohan, N., T.M. Undeland and W. P. Robbins, Power Electronics, Converters, Applications and Design, 5th ed., New York: John Wiley & Sons Inc., 1995.

Optimization Toolbox, User's Guide, Ver. 2.0, The MathWorks Inc, 2000.

Perlovsky, L.P. "Conundrum of Combinatorial Complexity", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 6, June 1998, p666-670.

Pudney, P., Optimal energy management for solar-powered cars, PhD thesis in Applied Science, University of South Australia, 2000.

<http://scg.levels.unisa.edu.au/Pudney/thesis/>

Roche, D.M., J.W.V. Storey, A.E.T. Schinckel, C.P. Humpris and M.R. Guelden, Speed of Light, 1996 World Solar Challenge. Sydney: Photovoltaics Special Research Centre, 1997.

Roberts, S.M., J.S. Shipman, Two-Point Boundary Value Problems: Shooting Methods. Modern Analytic and Computational Methods in Science and Mathematics. Number 31, New York: American Elsevier, 1972.

Schildt, H., The Complete Reference C, 3rd ed., Berkley: Osborne, 1995.

Schwartz, A.L., Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems, PhD thesis in Electrical Engineering and Computer Sciences, University of California at Berkley, 1996.

Storey, J.W.V, A.E.T. Schinckel and C.R. Kyle, Solar Racing Cars, 1993 World Solar Challenge. Canberra: Australian Government Publishing Service, 1994.

Tamai, G, The Leading Edge, Aerodynamic Design of Ultra-streamlined Land Vehicles. Cambridge MA: Robert Bentley Inc., 1999.

TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide, Literature Number SPRU024E, Texas Instruments, 1999.

TMS320F243, TMS320F241 DSP CONTROLLERS, Datasheet SPRS064C, Texas Instruments, 2000.

TMS320F243/F241/C242 DSP Controllers Reference Guide, System and Peripherals, Literature Number SPRU276C, Texas Instruments, 2000.

Watt, A., and F. Policarpo, 3D Games, Real-time Rendering and Software Technology, Vol. 1, ACM SIGGRAPH Series. Harlow, England: Pearson Education, 2001.

Wolfe, M.A., Numerical Methods for unconstrained Optimisation: An Introduction, New York: Van Nostrand Reinhold, 1978.

Zorpette, G., "Sun Kings Cross the Outback", IEEE Spectrum, February 2002, p.40-46.

# Appendices

# Appendix A: 2001 WSCC Technical Regulations

The latest regulations and most recent information about the World Solar Challenge and the World Solar Cycle Challenge can be found on the official website: www.wsc.org.au.

## A.1 Regulations

***Regulations***

**of the**

## 2001 World Solar Cycle Challenge

***Chapter 1 - Administration and Conduct***

**(v.060201)**



### 1. Event Title

**1.1** The name: The World Solar Cycle Challenge® is the "Correct Title" of the Event.

**1.2** If a "Naming Rights" sponsor is signed, the "Correct Title" of the event may become the "(sponsorship name) World Solar Cycle Challenge".

**1.3** Entrants shall use the "Correct Title" in all references to The Event.

### 2. Organisation and Status

**2.1** The event is recognised by the International Solarcar Federation (ISF).

**2.2** The event will be conducted by these regulations and any further regulations that may be issued.

### 3. Entrants & Eligibility

**3.1** The Entrant is the legal entity completing the Participation Agreement.

**3.2** Eligible Vehicles are power assisted pedal cycles (as defined by the Motor Vehicles Act 1966) and are compliant with the descriptions contained in the Technical Regulations section of this document.

**3.3** A team may comprise up to 15 people.

**3.4** All team members must be registered and sign in at scrutineering.

### 4. Organiser and Promoter

**4.1**
**Australian Major Events**
**World Solar Challenge**
GPO Box 1972
Adelaide 5001
AUSTRALIA
**Telephone**: 8-8463 4690

**Facsimile:** 8-8305 0175
**E-Mail:** wsc@saugov.sa.gov.au

4.2 **Organising Committee:**

Chris Selwood, Belinda Dewhirst, Damon Cramp, Brian Scholz, Ray Hemer and representatives of Bicycle SA

Senior Officials:

**4.4.1** Event Director - Chris Selwood

**4.4.2** Event Manager - to be advised

**4.4.3** Clerk of the Course - to be advised

4.4 **4.4.4** Assistant Clerk of the Course - to be advised

**4.4.5** Secretary of the Meeting - To be advised

**4.4.6** Chief Medical Officer - Dr Bill Boyd

**4.4.7** Chief Scrutineer - To be advised

**4.4.8** Chief Timekeeper - To be advised

Date and Venue of the Event

**4.5.1** The event will commence with scrutineering on 17 November 2001 and conclude following the prize-giving on 25 November 2001.

**4.5.2** The event is to commence in Alice Springs, NT and conclude in Adelaide, SA.

4.5 **4.5.3** Pre-event scrutineering commences - 17 November 2001.

**4.5.4** Stability and brake test - 17 November 2001.

**4.5.5** Pre event and daily briefings as notified in further regulations.

**4.5.6** Start of on-road component - 19 November 2001.

**4.5.7** Presentation of Awards - 25 November 2001.

Entries

**4.6.1** Applications for participation may be made by Entrants with eligible vehicles which are described in regulation 3.2

**4.6.2** Applications are open with the issue of these regulations.

4.6 **4.6.3** Applications must be made on the approved form and signed by the Entrant.

**4.6.4** Every Entrant must complete a participation agreement.

**4.6.5** The field will be limited to twenty-five (25) teams

**4.6.6** Entries received after the twenty-five (25) team maximum may be put on a reserve list.

**4.6.7** 25% of entry fee will be invoiced on acceptance of entry.

**4.6.8** Insurance fees to be paid by 31 July 2001 (Invoiced on 1 July 2001). IV

Remainder of entry fee to be paid by 31 July 2001 (Invoiced on 1 July

**4.6.9** Remainder of entry fee to be paid by 31 July 2001 (Invoiced on 1 July 2001).

**4.6.10** Entries close 31 July 2001.

**4.6.11** Vehicle and battery data sheets must be lodged by 31 July 2001.

**4.6.12** Any team not supplying the required information by the due date will attract a penalty of $220.

**4.6.13** The Organisers reserve the right to reject any entry; such refusal is not subject to protest.

Change of Entry Details

**4.7**

**4.7.1** An Entrant may change the specification of the vehicle or drivers up to the scheduled start time of scrutineering. No guarantee is given that changes will appear in printed lists.

**4.7.2** Once a vehicle has passed scrutineering, no changes to specification will be permitted.

**4.7.3** The Organisers reserve the right to determine the class of any vehicle.

Entry Fee

**4.8**

**4.8.1** The entry fee for all classes is AU$ 550 (including GST)
Australian entries registered for GST will receive a tax invoice
Fees are payable by **31 July 2001**.
Payments are to be made to Australian Major Events, WORLD SOLAR CHALLENGE.

**4.8.2** 25% will be invoiced on acceptance of nomination (this is non-refundable).

**4.8.3** Remaining 75% will be invoiced on 1 July 2001 (payable by 31 July 2001).

**4.8.4** Insurance fees as detailed in Article 5 will be invoiced 1 July 2001.

Financial

**4.9**

**4.9.1** The South Australian Tourism Commission will issue invoices on behalf of the World Solar Challenge.

**4.9.2** All invoices are payable by the end of the month of issue.

Definition of Categories

**4.10**

**4.10.1** The definitions of type of entries are as defined the technical regulations

Amendments to the Regulations

**4.11**

**4.11.1** These regulations may be amended by the issue of Further Regulations.

Interpretation of Regulations

**4.12**

**4.12.1** The Stewards of The Event are the only authority empowered to make a decision on the interpretation of these regulations.

Route

4.13

**4.13.1** The Event shall be conducted on public roads between Alice Springs, Northern Territory and Adelaide, South Australia. A minimum distance of 1400 km

**4.13.2** The organisers reserve the right to cancel the event or any stage thereof for any reason outside of its control.

Authority to be on Public Roads

4.14

**4.14.1** Only vehicles successful in scrutineering will be become eligible to compete.

**4.14.2** All vehicles associated with the event shall comply with the relevant road traffic laws.

**4.14.3** Any vehicle associated with the event shall be driven in a careful and lawful manner at all times.

Riders

4.15

**4.15.1** Teams will contain a minimum of two and a maximum of six riders

**4.15.2** All riders will be required to sign on in Alice Springs and unless injured or sick, compete in every stage.

Drivers and Entrants' Qualifications

4.16

**4.16.2** All riders shall have a minimum experience of 3 hours riding the solar cycle on public highways.

**4.16.3** Only the nominated riders may ride during competition hours.

**4.16.4** Any participant under the age of 18 shall be duly vouched for (see notes).

Administrative Checking

4.17

**4.17.1** Place: Red Centre Resort, Alice Springs

**4.17.2** Date and Time: Saturday 17 November 2001 from 0800 hours.

Scrutineering

Scrutineering will take place at the Red Centre Resort (Alice Springs) and such other place and time as the Clerk of the Course may direct.

**4.19.2** Saturday 17 November 2001: 0800 to 1800 hours.

4.19

**4.19.3** Specific scrutineering times for each team will be made by draw and results posted on the Official Noticeboard at the Red Centre Resort at the time stated in 4.19.2

**4.19.4** All competing vehicles are required to be presented at the set time from November 17 in Alice Springs in a READY TO START condition. Scrutineers will examine the vehicles to ensure they conform with these regulations.

**4.19.5** If a vehicle arrives late for scrutineering, the Team will incur a fine of $50.00 and may be demoted to reserve status.

**4.19.6** Any vehicle failing scrutineering may be re-presented at the discretion

of the Chief Scrutineer.

**4.19.7** No vehicle will be allowed to start the event until it has passed scrutineering, and the relevant briefing has been attended.

**4.19.8** Speed, stability and brake testing forms part of the scrutineering process

**4.19.9** The Primary Support Vehicle (detailed in reg. 6.4.3) shall be presented for inspection at Alice Springs on Saturday 17 November 2001.

Unfair Practice

**4.19.10** Exclusion will occur if, during the Event or at scrutineering, the Organisers discover that an entrant or crew has deliberately violated these Regulations to gain unfair advantage over other entries, or departed from the spirit of the competition.

Grid Positions

**4.20**

**4.20.1** Grid position will be determined by performance at the speed trials.

Pre-Event Briefings

**4.21**

**4.21.1** The pre-event briefing will be prior to the event start on 17 November 2001 at the Red Centre Resort, or such other place as may be announced in further regulations. The Team Manager, Safety officer and all riders must attend all briefings.

**4.21.2** Daily stage briefings will be held 30 minutes prior to each stage start.

**4.21.3** A register must be signed confirming attendance at the briefings. Non-compliance will incur a time penalty.

Compulsory Signs

**4.22**

**4.22.1** The organisers shall supply signs that carry event and sponsor logos.

**4.22.2** These signs must be displayed on the solar cycle at all times and in such a way that they are clearly visible to a person standing near the cycle.

**4.22.3** Unbroken spaces of at least 200mm x 200mm on each side of the solar cycle to be provided for this purpose.

**4.22.4** Unbroken spaces of at least 200mm (height) x 500mm (width) on both front doors of the support vehicle to be provided for this purpose.

Competition Numbers

**4.23**

**4.23.1** Cycles will be allocated numbers upon receipt of entry.

**4.23.2** Special requests may be submitted. Allocation is at the sole discretion of the Organisers.

**4.23.3** Competition numbers must be in an area of at least 100mm x 100mm and be clearly displayed on each side of the solar cycle.

**4.23.4** Competition numbers shall be in contrasting colours to their background and acceptable in every way to the Chief Scrutineer.

**4.23.5** Competing vehicles must carry the national flag of the country of entry, fixed adjacent to the windscreen. Minimum size 70mm x 40mm

4.23.5 Competing vehicles must carry the national flag of the country of entry, fixed adjacent to the windscreen. Minimum size 70mm x 40mm

Advertising Signs

4.24 4.24.1 Advertising signs are permitted on any vehicle. Signs and lettering must not be of an offensive nature.

Overnight Accommodation

4.25 Any cost incurred for overnight accommodation at the start, finish and any stage, is the responsibility of the Entrant.

## 5. Insurance Cover

All Entrants will be covered by the Organisers Public Liability policy.

5.1.1 In addition to this, Entrants are required to carry the following minimum cover:

5.1 All team members (as defined in clause 3.4) will be covered by personal accident and ambulance cover. The arrangements for this will be advised in Further Regulations.

5.1.2 **NB** The additional insurance covers as described in 5.1.2 are compulsory for all entrants. The Organisers will arrange this cover. Payment of any insurance premium will be the responsibility of the entrant, and will be invoiced on or after 1 July 2001.

Comprehensive Vehicle Insurance

5.3 5.3.1 Comprehensive vehicle insurance is the responsibility of the entrant.

## 6. Conduct of the Event

The Start Line: Red Centre Resort, Alice Springs: 0900, Monday 19 November 2001

6.1 6.1.2 The Organisers reserve the right to change the time and/or place of the start, subject to the approval of the Stewards.

The Course

6.2 6.2.1 The Event shall be conducted on the Stuart Highway between Alice Springs and Adelaide. The Event will be a staged with a set finish location every day. All competitors will travel from Alice Springs to Adelaide over seven days covering a total distance of 1434 km. Entrants will travel between the hours of 08:00 and 18:00 (6:00pm). If they have not completed the set distance on any given day, the team will be instructed by the Sweep Official to transport their cycle to the finish point

***The event itinerary is as follows:***

| | | | |
|---|---|---|---|
| **Saturday 17/11** | Registration and Scrutineering at Alice Springs | | |
| **Monday 19/11** | DAY 1 | Alice Springs to Erldunda | 199 km |
| **Tuesday 20/11** | DAY 2 | Erldunda to Marla | 252 km |

| | | | |
|---|---|---|---|
| **Tuesday 20/11** | DAY 2 | Erldunda to Marla | 252 km |
| **Wednesday 21/11** | DAY 3 | Marla to Coober Pedy | 233 km |
| **Thursday 22/11** | DAY 4 | Coober Pedy to Glendambo | 253 km |
| **Friday 23/11** | DAY 5 | Glendambo to Port Augusta | 289 km |
| **Saturday 24/11** | DAY 6 | Port Augusta to Quorn | 116 km |
| **Sunday 25/11** | DAY 7 | Balaklava to Adelaide | 92 km |
| | | TOTAL DISTANCE | 1434 km |

STARTING ORDER

**6.2.3** The Speed, Stability and Brake Test will determine the starting order on Day 1. The fastest cycle will take the first position on the starting grid. The remaining teams will be ranked according to their Speed, Stability and Brake Test results. The starting order on Days 2-7 will be based on the previous day's results.

Control Points

**6.3.1** Control Points will be established at intervals along the route.

**6.3.2** Control Points (static and mobile) may or may not be disclosed.

**6.3.3** Any vehicle associated with the event may be required to stop at control points. Time will be credited. No repair work may be carried out.

Support Vehicles

**6.4.1** Each team must have a minimum of one support vehicle.

**6.4.2** A maximum of two vehicles may accompany each entrant at any one time (1 in front and 1 to rear).

**6.4.3** At any given time, the rear support vehicle shall be designated the primary support vehicle, and shall have communications in accordance

**6.4.4** A vehicle complying with regulation 6.4.3 must be placed behind the solar cycle at any time it is on the public road.

**6.4.5** The support vehicles shall carry a flashing amber beacon (compliant with Australian Standards) on a prominent high point of the vehicle, in a

**6.4.6** Whilst travelling on the open road, all vehicles associated with the Entrant, with the exception of those identified in 6.4.2 shall keep a

**6.4.7** A warning sign not less than 900mm x 300mm with lettering in contrasting colour to the background, and proportional to the overall

**6.4.8** All team vehicles must carry a sign, visible from the rear, stating the name of the Entrant.

**6.4.9** Vehicles equipped with CB radio (qv 6.14.2) shall carry a sign, visible from the rear, advising the selected CB channel number.

**6.4.9** Vehicles equipped with CB radio (qv 6.14.2) shall carry a sign, visible from the rear, advising the selected CB channel number.

**6.4.10** A primary support vehicle (qv 6.4.3) shall not be a bus, truck, or large campervan, or tow a box trailer such that the ability of road users approaching from the rear would have their vision obscured.

**6.4.11** When stopped, all wheels of all vehicles must be off the highway. Wherever possible a minimum 1m from the edge of the tarmac surface should be observed.

**6.4.12** The Entrant shall provide the organiser with a list of all vehicles associated with the team.

**6.4.13** The Entrant shall be responsible for the actions of all crew and vehicles associated with the team.

**6.4.14** Support and team vehicles may be subject to scrutineering at any time during the event to ensure compliance with these regulations.

Timing

**6.5.1** Timing is under the control of the Official Timekeeper

**6.5.2** The official start time each day is 0800 (0900 Day 1)

**6.5.3** Official finish time each day is 1800. All solar cycles must have ceased running by 1800 (6pm) each day. A Sweep crew will travel at the rear of the field to facilitate this. A time penalty will be imposed on teams that do not reach the Overnight Control before the daily finish time.

**6.5**

**6.5.6** A vehicle starting before its official start time will be subject to penalty (as detailed in 8.3.13).

LATE TIME LIMIT:

**6.5.7** Any cycle transported into the Overnight Control at the end of the day will be given the total time for the day, plus 2 minutes for each kilometre not covered, plus any penalty for arriving after the close of the day. The Entrant may continue in the next day, starting at the rear of the field. The distance covered on the previous day will determine the starting order.

**6.6** Conduct on road

**6.6.1** CONVOYS: Entrants and support vehicles may not drive in convoys (qv. 6.4.6) and must allow easy overtaking.

**6.6.2** CATTLE GRIDS: Stock control devices may not be covered, and should be crossed with caution. Solar Cycles may not be carried by hand over grids.

**6.6.3** OVERTAKING: Any cycle or support vehicle being overtaken must allow the overtaking solar cycle or World Solar Challenge solar car and support vehicle to pass without incident. The team being overtaken must allow sufficient room for passing by keeping to the left and giving way until both the overtaking solar vehicle and its rear support vehicle has merged successfully. Any team vehicle that is observed not to facilitate overtaking will incur a penalty for the team.

Any event vehicle being overtaken MUST give way. This includes support vehicles.

**6.6.4** RETIREMENTS: Any Entrant wishing to retire must give written notice to the Clerk of the Course of the intention to retire. If the retirement decision is taken during a daily stage, the Entrant must transport the solar cycle to the end of the stage.

Vehicle Movement

**6.8.1** Competing Vehicles may only move under their own power between their start time and finish time, unless abnormal circumstances prevail (see notes).

**6.8.2** Competing vehicles may be pushed on and off the highway

**6.8.3** Push starting the vehicle is not allowed.

**6.8.4** Regenerative power systems must not be on when hand pushing or being towed under circumstances allowed in 6.8.5.

**6.8** **6.8.5** The vehicle may not be towed or carried forward by another vehicle, unless it is disabled.

**6.8.6** Slip streaming is not allowed (driving closer than 60m). Whilst travelling on the open road a solar cycle may be no closer than 60m to the vehicle in front unless overtaking.

**6.8.7** Pressure-wave pushing is not allowed (driving closer than 10m). The rear support vehicle is not allowed within 10m of the solar cycle whilst travelling on the open road.

**6.8.8** Driving tests may be carried out by any qualified crew member (qv 4.6.1) from sunrise until 0800 and from 1800 until sunset.

Intoxicating Substances

**6.11** **6.11.1** The consumption or taking of intoxicating items is strictly forbidden. Australian civil law applies to drugs and to driving under the influence of alcohol. Drivers, team members and officials are to maintain a 0% blood/alcohol level whilst engaged in any duties associated with the event.

**6.12** Safety

**6.12.1** The Entrant is responsible for the road-worthiness of its solar vehicle, and will be required to sign a declaration of the vehicle's integrity and suitability for the event.

**6.12.2** The scrutineering process will determine whether the solar cycle complies with the Regulations.

**6.12.3** No warranty or representation, whether expressed or implied, is made in relation to the mechanical and/or systems roadworthiness of the entrant vehicle in providing this compliance with the event Regulations as specified above.

**6.12.4** All solar cycles and support vehicles are operated and driven at the Entrant's own risk.

**6.12.5** All solar cycles and support vehicles must be maintained in a safe, road-worthy condition and be operated safely and within the law at all times.

**6.12.6** A team may be disqualified from the Event at any time if the team, in the opinion of the Stewards, is operating its solar cycles, or support vehicles, in an unsafe manner.

**6.12.7** The cycle must be stable at all race speeds. Class C cycles should be stable at rest. Wind or buffeting by passing traffic must not change the direction of the cycle's travel at any time. All entrants must undergo a general stability test. The purpose of this test is to determine the general stability of the vehicle and is not a thorough analysis of the vehicle's stability. Additional riders may have to undergo the stability test if requested by an official. The team will be notified at the time of scrutineering.

**6.12.8** All riders must start each ride with at least 500ml of drinking water for every 30 minutes spent on the solar cycle (ie 1 litre per hour). This should be consumed during the ride to avoid dehydration.

**6.12.9** All riders must wear shatterproof glasses, goggles or a helmet visor if the solar cycle does not have a windscreen that provides suitable protection

**6.12.10** No sandals or thongs will be allowed. Class B/C vehicles must have substantial protection below the feet to prevent the feet from falling onto the roadway in the event of dislodging from the pedals.

**6.12.11** All riders must wear an approved bicycle helmet while riding the solar cycle

Safety Officer

**6.13.1** Each team must provide a Safety Officer.

**6.13** **6.13.2** Each team must provide suitable and appropriate safety equipment, including (but not limited to): first aid kit; safety glasses and gloves for handling batteries; hazard warning cones; red warning flag; fire extinguishers; safety vests which shall be used on all appropriate occasions

Communications (see notes)

**6.14.1** Every solar cycle shall have means of communication with the primary support vehicle. Entrants in Non-aerodynamic classes may use a loudspeaker

**6.14** **6.14.2** The primary support vehicle (and other support vehicles) of each team must have 40 channel UHF CB radio facilities compliant with Australian Standards.

**6.14.3** The chosen channel number shall be displayed on the rear of the vehicle as detailed in 6.4.9.

6.14.3 The chosen channel number shall be displayed on the rear of the vehicle as detailed in 6.4.9.

## 7. Observers

Appointment of Observers

7.1 7.1.1 An Observer may be appointed by the organisers to travel with each competing team.

7.1.2 Observers will be considered as Judges of fact.

7.1.3 Observers may be changed between Entrants throughout the Event.

## 8. Penalties and Protests

8.1 Entrants committing the following offences shall be subject to a minimum 10 minute - maximum 60 minute penalty.

8.2 Time penalties shall be served on the day of issue and prior to crossing the finish line.

8.3 8.3.1 Misrepresentation.

8.3.2 Failure to comply with any provision of these regulations.

8.3.3 Slip streaming, hand pushing or pressure wave pushing.

8.3.4 Failure to observe a request by Police or Officials.

8.3.5 Obstructing an overtaking vehicle.

8.3.6 Wilful damage or interference to property.

8.3.7 Failure to follow the route instructions.

8.3.8 Failure to report damage to stock.

8.3.9 Consumption of intoxicants.

8.3.10 Failure to stop at a control.

8.3.11 Running without warning signs and lights.

8.3.12 Exceeding any posted speed limit.

8.3.13 Starting prior to due time of departure (time prior to due time being added to penalty).

Failing to get off the highway when stopped.

8.3.14 **NOTE:** In the case of a serious or repeated breach of any of the offences outlined above, the organisers may, at their sole discretion, exclude any competitor from the event.

Exclusion from the Event

8.4 Entrants committing the following offences shall be liable to exclusion as determined by the Stewards:

8.4.1 Replacement of battery without permission.

8.4.3 Charging of batteries from any source other than the relevant scrutineered array during Event Time.

8.4.4 Failing the stability test.

8.4.6 Carrying or towing a competing vehicle.

8.4.7 Wilful disregard of the regulations and the spirit of the event.

8.4.8 Running without rear support vehicle as described in regulation 6.4.2.

8.5 Protests and appeals

8.5.1 A protest must be lodged in writing and handed directly to the Clerk of Course or his delegate, being any listed official.

8.5.2 A Protest fee of $55 must accompany any protest.

8.5.3 The decision of the Stewards is final, and binding on all parties.

## 9. Determination of Winners - Finishers

9.1 The winner will be the first solar cycle of each class to have completed the course in accordance with these regulations.

9.1.1 To be classed as a finisher, a vehicle must have completed the entire course within the time allowed.

9.2 Provisional Result

9.2.1 The progress of solar cycles may be publicised during the event. These results may not include all penalties and, therefore, will not be accurate and final

9.3 Final Result

9.3.1 Final results will be published once passed by the Stewards.

## 10. Winners and Finishers

10.1 The winning team will make itself available for a press conference if so required.

10.2 Finishing vehicles shall be made available for public exhibition up to the time of the prize giving ceremony.

## 11. Prizes and Awards

11.1 1st, 2nd and 3rd in each class

11.2 Certificates to all finishers

11.3 Other trophies and prizes to be announced in the Further Regulations.

## 12. Advertising and Publicity of Results

12.1 All advertising, sales promotion and publicity material produced by or in connection with the entrants or their sponsors, concerning or referring to, the Event, shall refer prominently to the Event by the correct title as defined in Regulation 1, and all entrants shall, by entering the Event, specifically agree to

abide by this Regulation.

12.2 Photographs, drawings, vehicle specifications and background information for the program must be provided by 2 July 2001.

12.3 By entering the Event, teams agree to the free use of their names and photographs, and those of their vehicles and equipment in any publicity material that may be issued by the Organisers and the Event's principal sponsors and its associated companies.

12.4 All teams will provide the organiser with a precis of their achievements, which may be published in a post-event report.

## A.2 Notes to Regulations

### World Solar Cycle Challenge 2001

#### *Notes*

#### *to be read in conjunction with the regulations*

**Note C1.**

Any person under the age of 18 must have a letter consenting to their participation signed by a legal parent or guardian.

**Note C2. Communications.**

The Australian communications regulations may be found at http://www.aca.gov.au

**Note C3. "...unless abnormal circumstances prevail"**

It is the responsibility of the Entrant to operate their cycle safely at all times. The Entrant may take whatever action they consider appropriate to any given situation and report the action taken in the vehicle log.

**Note C4.**

***The requirements of the Australian Design Rules (ADR4/01) for Motor Vehicles:***

Seat belts are designed to bear upon the bony structure of the body, and should be worn across the chest, shoulders and low across the front of the pelvis; wearing the lap section of the belt across the abdominal area must be avoided. Seat belts should be adjusted as firmly as possible, consistent with comfort, to provide the protection for which they have been designed. A slack belt will greatly reduce the protection afforded to the wearer.

## A.3 Technical Regulations

# World Solar Cycle Challenge 2001

## *Technical Regulations*

**(v.060201)**

## A VEHICLE SPECIFICATIONS

### (i) CYCLE SIZE:

The maximum size of the solar cycle in motion is 3.5 metres long by 1.6 metres wide by 1.6 metres high. The minimum height is 1 metre. 3 wheel vehicles must have 2 wheels configured in the style of a transverse axle. The minimum track is 0.75 metre. The minimum wheelbase is 1 metre.

### (ii)BRAKING:

There must be 2 brake systems independent of each other. The brake systems must be constructed in a way that no single-point failure compromises both systems. The braking power of each and every braking system shall be symmetrical about the vehicle's longitudinal centre line. The cycle must be able to stop within 22 m from a speed of 30 km/h on a dry bitumen surface. This will be tested at the time of the Speed, Brake and Stability test.

### (iii) VISIBILITY:

The rider must have a clear view of the road without significantly changing his/her position. Minimum eye height is 700 mm. Riders must have adequate 180-degree vision of the road to the front of the vehicle. A mirror or rear electronic view system is required to give a clear view to the rear of the cycle.

### (iv) RIDING POSITION:

The riding position shall not compromise machine controllability or safety, nor shall the riding position place the rider in a potentially hazardous position in the event of a collision. For these reasons a prone riding position is not allowed. The Class C vehicles should take into account protection for the rider in the event of a collision.

### (v) SAFETY BELTS:

Safety belts with a minimum four (4) point attachment must be worn in all 3 and 4 wheeled cycles. Attachment points must be designed in accordance with sound engineering practice. Entrants should note the requirements of the Australian Design rules concerning the wearing of belts.

### (vi) EXTREMITY MARKINGS:

All cycles must have a fluorescent / iridescent strip or LED lights attached to the front and rear extremities of the cycle.

### (vii) MINIMUM HEIGHT:

The minimum height for a solar cycle is 1 metre. A fin may be erected to achieve this height. If a fin is employed, it must be fluorescent or carry LED lights. Power for the LED lights may be supplied by batteries isolated from the main battery system.

### (viii) ELECTRICAL:

The driver must be protected from electrical shock hazards. High Voltage warning signs must be fitted if using in excess of 32 volts. The driver must have a means of electrically isolating the battery and the solar panel

### (ix) FAIRING:

A front cowl is permitted on cycles entered under either Class A or B but this must be located between the front of the vehicle and the rear of the front tyre. No other structure with the sole purpose of streamlining is allowed in Classes A and B. This includes rear wheel spats.

### (x) EGRESS:

The rider must be able to be extricate themselves from the vehicle within 15 seconds. If the extrication involves permanent damage to the vehicle then the Team Manager will need to satisfy and declare the method to the Scrutineer at the time of Scrutineering.

### (xi) STEERING:

The steering effort must be forward of the centre of mass (i.e. no rear-wheel steering) The type of steering mechanism is free, provided the rider is afforded continuous positive control without the need for regular adjustment. Simple rope systems are not permitted. Steering linkages shall operate freely from full left to full right lock without binding or fouling.

### (xii) EQUIPMENT:

Vehicles must carry all equipment as fitted at the time of scrutineering at all times during the course of the event.

### (xiii) WARNING DEVICE:

Cycles must be fitted with a bell or horn.

---

## B. Power

Only a combination of solar radiation and instant human power may be used for propulsion.

Solar collectors must not exceed 1.6 square metres in total. This area may be carried on board either the solar cycle or a combination of support vehicle and solar cycle.

(The array must be constructed in a way that allows easy measurement. If there are greater than two separate array areas, the largest cell will be measured and multiplied by the total number of cells to determine the total area)

The solar cycle must carry a minimum of 0.15 square metres measured horizontally.

Solar collectors carried on board the support vehicle may be used to recharge spare battery packs during the event.

No other source of charging is permitted during event time.

Mirrors, lenses or any optical device used to increase cell output are not permitted. Any cell cooling may only be achieved using coolant at ambient temperature.

**NOTE:** All competitors may start each day with fully charged batteries. This allows teams to recharge by mains or generator outside of Event time. Batteries will NOT be

collected at the conclusion of each day. The organisers make no provision for, and give no guarantee that power will be available at any given point on the route.

### B.1 BATTERIES:

All propulsion batteries must be rechargeable. The battery type options and maximum weight allowances are as follows:

| BATTERY TYPE | CLASS A | CLASS B | CLASS C1 | CLASS C2 |
|---|---|---|---|---|
| Lead Acid (Pb/Acid) | 24 kg | 24 kg | 24 kg | 24 kg |
| Nickel Cadmium (Ni/Cd) | 19.2 kg | 19.2 kg | 19.2 kg | |
| Nickel Zinc (Ni/Zn) | 14.5 kg | 14.5 kg | 14.5 kg | |
| Lithium Ion | 6.8 kg | 6.8 kg | 6.8 kg | |
| Nickel Metal Hydride (NiMH) | 13.7 kg | 13.7 kg | 13.7 kg | |

Any battery not listed will have its theoretical energy density determined by the CSIRO battery laboratory. Entrants desiring to use an electrochemistry not listed are encouraged to contact the organisers as soon as possible.

Batteries may be broken down into a maximum of three (3) battery packs. The total weight of the battery packs must not exceed the maximum weight allowance. ALL battery packs will be scrutineered.

Only one battery pack is to be carried on the solar cycle at any given time.

All entrants may commence each day with a full charge

Any unauthorised battery replacement will incur a time penalty or exclusion from the Event.

Small instruments may be powered by primary cells. A two-way radio with an integral rechargeable battery may also be used. Any other rechargeable battery will be considered part of the total battery mass allowed.

Teams must submit a chemical incident contingency plan relevant to the battery chemistry employed and include a statement of intent with regard to handling or disposal of cells, batteries or component materials. This should include all cells used in ancillary equipment used by the team as well as that in the competing vehicle.

Energy storage devices other than batteries may be used but at the beginning of each day's stage the total of stored energy must be less than a nominal 950watt hours (at a 20hour rate)

### B.2 MOTORS

**This event is for solar power assisted cycles. Power assisted cycles are defined by law in the Motor Vehicle Regulations as detailed below.**

SOUTH AUSTRALIA

## MOTOR VEHICLES REGULATIONS 1996

*These regulations are reprinted ~~pursuant~~ to the Subordinate ~~Legislation~~ Act 1978 ~~and~~ incorporate all amendments in force as at 22 April 1999.*

**Exemption from registration and insurance for power-assisted pedal cycles**

**9B** A power-assisted pedal cycle may be driven on roads without registration or

insurance.

**"power-assisted pedal cycle"** means a pedal cycle that has one or more auxiliary propulsion motors with a combined power output not exceeding 200 watts;

B.2.1

Solar Cycles shall have one or more auxiliary propulsion motors with a combined power output not exceeding 200 watts;

B3. CLASSES:

**NOTE:** Only single seat cycles will be allowed to compete

Class A. Non-aerodynamic, commercially available standard bicycles (maximum 2 wheels - not recumbent)

Class B. Non-aerodynamic Recumbent or Experimental Cycle

Class C1. Aerodynamic Experimental Vehicle - Open class (minimum 3 wheels)

Class C2. Aerodynamic Experimental Vehicle with lead-acid batteries (minimum 3 wheels)

Restricted to Lead Acid batteries and Production solar cells. Solar cells and batteries must be commercially available to all competitors and approved by the WSC.

Teams must demonstrate that the cells being used in the array were purchased from a standard production lot.

Additional class awards may apply provided that a minimum of three entries can be so grouped. The organisers reserve the right to determine the class of any entrant.

B4 DEFINITION OF CATEGORIES:

**PRIVATE ENTRY / SCHOOL:** Entered in the name of an individual or educational institution.

**TERTIARY INSTITUTIONS:** Entered in the name of a tertiary institution

**COMMERCIAL ORGANISATIONS:** This entry is in the name of a company or manufacturer.

# Appendix B:A Note on Low Speed Aerodynamics

This appendix discusses the aerodynamics of solar cycles in more detail. It offers a number of comparisons and a more detailed look at Solar esCape.

## B.1 The Drag Coefficient and Drag Area

The term drag area, $C_d A$, refers to the product of the drag coefficient and the area relative to which it was measured. The drag area is a very useful metric to compare the aerodynamic efficiency of a diverse range of vehicles. In automotive convention, the drag coefficient is measured relative to the frontal area of the vehicle. The frontal area is the area of the vehicle projected forward, sometimes called the front profile area. CAR, December 2000, p115 gives the BMW 330i a frontal area of 2.46 square metres. Assuming a drag coefficient of 0.30 with respect to the frontal area, the drag area of the BMW is 0.738 square metres. To maintain a constant speed of 75 kph would require 14.75 kW of power to overcome the aerodynamic load.

A number of heuristics governing the aerodynamics have developed since the inception of solar racing in 1988, they are presented here briefly and farther discussion is presented in Roche *et al* and Tamai.

In order of importance:

1. The airflow must be attached over every surface of the vehicle.
2. Wetted area must be minimised.
3. Laminar flow over the wetted area must be maximised.
4. The surface finish must be as smooth and consistent as the budget allows.
5. The vehicle must produce zero lift.
6. The frontal area must be minimised.
7. All appendages, i.e. wheel spats, canopies, must be smoothly integrated into the body.
8. Ventilation drag must be minimised.

One of the implications of the above heuristics is that it is advantageous to sacrifice solar panel area. The wetted area, i.e. the outside surface area, of Solar esCape is about 10 $m^2$, however, the plan area, i.e. the area projected to the ground, is about 1.8 $m^2$. The top surface of the cycle is also highly curved. While it is conceivable to construct a highly curved panel, the projected area sunwards is about 1 $m^2$ at best. This is confirmed by Reflex – Southern Alliance, their 1999 WSCC cycle fitted the minimum panel area into a transparent section on the top of their canopy.

The good $C_d A$ value of the Zero To Darwin Project' solar cycle was achieved using a track of 500 mm. Subsequent regulations limit the track to a minimum of 750 mm and set a minimum height for the cyclists eyes. These changes make it highly unlikely that a solar cycle will have such a low $C_d A$ in the near future.

## B.2 Solar esCape

The design procedure used for the aerodynamics followed that described by Tamai. Initially, a wide range of shapes and configurations was considered. After careful analysis, a torpedo-with-wings configuration was selected. This configuration allows

a high degree of freedom in selecting the track of the rear wheels, an important parameter in the lateral stability of the solar cycle.

For the purposes of simulation, the aerodynamic configuration was modelled in a parametric manner to allow the aerodynamics to be tuned within the constraints of the cycle as a whole. This model is listed in `aero.m` in Appendix D.

The prototype is expected to have a drag area, $C_dA$, of 0.15165 square metres, with a wetted area of about 10 square metres. This is comparable to that of the middle range solar cars. WestCape, in contrast, had a $C_dA$ of 0.244.

# Appendix C: Bryson and Ho's Gradient Method

## C.1 Introduction

The method described here is a first order gradient method. A number of other methods are suggested in the text, but this one appeared to be the simplest to implement, and keeping our particular problem statement in mind, the most suited to the type of MTP under investigation.

The method was expected to be relatively insensitive to the initial conditions, that is, it would converge to a solution from inaccurate first guesses. Indeed first order gradient methods are typically used in this fashion to provide starting values for second order methods.

## C.2 General Procedure

This procedure follows that listed on p. 225 of Bryson and Ho.

Given the problem statement in the form:

*Differential equations* :

$$\dot{x} = f(x,u,t),$$

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial x}\right)^T \lambda - \left(\frac{\partial L}{\partial x}\right)^T,$$

*where* $u(t)$ *satisfies*

$$\frac{\partial H}{\partial u} = \left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T = 0 \qquad (7.54)$$

*with split boundary conditions* :

$x(t_0)$ *given,*

$$\lambda(t_f) = \left(\frac{\partial \varphi}{\partial x}\right)^T$$

Step A: Estimate a final time, $t_f$, and a control trajectory, $u(t)$.

Step B: Integrate the system equations, $\dot{x} = f(x,u,t)$, forward from the initial conditions, record $x(t), \psi[x(t_f),t_f], \left[\frac{d\phi}{dt} + L\right]_{t_f}$, *and* $\left.\frac{d\psi}{dt}\right|_{t_f}$

Step C: Backward integrate the following approximations to the influence equations:

$$\dot{p} = -(f_x)^T p - (L_x)^T, \quad p(t_f) = \phi_x\big|_{t_f}$$

$$\dot{R} = -(f_x)^T R, \qquad R(t_f) = \psi_x\big|_{t_f}$$

Step D: Evaluate the following integrals:

$$I_{\psi\psi} = \int_{t_0}^{t_f} R^T f_u W^{-1} (f_u)^T R dt$$

$$I_{\psi J} = (I_{J\psi})^T = \int_{t_0}^{t_f} (p^T f_u + L_u) W^{-1} (f_u)^T R dt$$

$$I_{JJ} = \int_{t_0}^{t_f} (p^T f_u + L_u) W^{-1} \left[ (f_u)^T p + (L_u)^T \right] R dt$$

Where W is an arbitrary positive definite weighting matrix.

For example: $W = \varepsilon \frac{\partial^2 H}{\partial u^2}$, $0 < \varepsilon \leq 1$, provided the result is positive definite.

Step E: Choose $d\psi$ such that $\psi[x(t_f)] \to 0, e.g. d\psi = -\varepsilon\psi[x(t_f)], 0 < \varepsilon \leq 1.$

Determine:

$$\nu = -\left[ I_{\psi\psi} + \frac{1}{b} \frac{d\psi}{dt} \left( \frac{d\psi}{dt} \right)^T \right]^{-1} \left[ d\psi + I_{\psi J} + \frac{1}{b} \left( \frac{d\varphi}{dt} + L \right) \left( \frac{d\psi}{d} \right)^T \right]$$

Where b is a positive weighting factor.

Step F: Calculate the corrections to the estimates and add to the estimates.

$$\delta u(t) = -[W(t)]^{-1} \left[ \frac{dL}{du} + (p + R\nu)^T f_u \right]^T$$

$$dt_f = -\frac{1}{b} \left( \frac{d\varphi}{dt} + \nu^T \frac{d\psi}{dt} + L \right)_{t_f}$$

Repeat the process until the terminal conditions are met to the required accuracy:

$$\psi[x(t_f)] = 0$$

$$\left[ \frac{d\varphi}{dt} + \nu^T \frac{d\psi}{dt} + L \right]_{t_f} = 0$$

$$I_{JJ} - I_{J\psi} I_{\psi\psi}^{-1} I_{\psi J} = 0$$

## C.3 Procedure applied to MTP

A number of simplifications arise when applying the method to the MTP. These are due to the analytical results of $\varphi = 0$ *and* $L = 1$.

The changes are:

Step C: $\dot{p} = -(f_x)^T p, \quad p(t_f) = 0$

Step D:

$$I_{\psi J} = (I_{J\psi})^T = \int_{t_0}^{t_f} (p^T f_u) W^{-1} (f_u)^T R dt$$

$$I_{JJ} = \int_{t_0}^{t_f} (p^T f_u) W^{-1} \left[ (f_u)^T p \right] R dt$$

Step E:
$$\nu = -\left[ I_{\psi\psi} + \frac{1}{b}\frac{d\psi}{dt}\left(\frac{d\psi}{dt}\right)^T \right]^{-1} \left[ d\psi + I_{\psi J} + \frac{1}{b}\left(\frac{d\psi}{d}\right)^T \right]$$

$$\delta u(t) = -[W(t)]^{-1}\left[(p + R\nu)^T f_u\right]^T$$

Step F:
$$dt_f = -\frac{1}{b}\left( \nu^T \frac{d\psi}{dt} + 1 \right)_{t_f}$$

One of the termination conditions also changes:

$$\left[ \nu^T \frac{d\psi}{dt} + 1 \right]_{t_f} = 0$$

## C.4 Application to Solar Cycle

By performing some analytical work prior to implementing the numerical method, one can reduce the computational workload, and reduce potential errors. From the problem statement in section 8, we can refine the method as follows:

Step B: The initial conditions are zero for the states and unknown for the Lagrange multipliers.

$$\psi(t_f) = \begin{pmatrix} Capacity - E_{batt}(t_f) \\ Dist - s(t_f) \\ v(t_f) \end{pmatrix}$$

$$\left.\frac{d\psi}{dt}\right|_{t_f} = \begin{pmatrix} -P_{batt}(t_f) \\ -v(t_f) \\ \dot{v}(t_f) \end{pmatrix}$$

Step C: The DE's reduce to:

$$\dot{p}_v = -(0 \quad 1 \quad f_{3v})p, \quad p(t_f) = 0$$

$$\dot{R}_v = -(0 \quad 1 \quad f_{3v})R, \qquad R(t_f) = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$$

$$where\ f_{3v} = -\frac{1}{mv^2}\left(P_{batt} + P_{sol} + P_{cyc} - \left(Dv^3 + Rv^2 + Gv\right)\right) + \frac{1}{mv}\left(3Dv^2 + 2Rv + G\right)$$

Step D: $f_u = \begin{pmatrix} 1 \\ 0 \\ \frac{1}{mv} \end{pmatrix}$, *and* $f_{uu} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ Hence the Hamiltonian is not positive, and we cannot use it as the basis for the weighting matrix. Since we only have one control, $P_{batt}$, the weighting matrix reduces to a scalar.

Let $W = \delta t$, our sampling interval.

Step E: Let $\varepsilon = 0.5, b = 0.5$.

# Appendix D:Simulation Programs

## D.1 Description of models

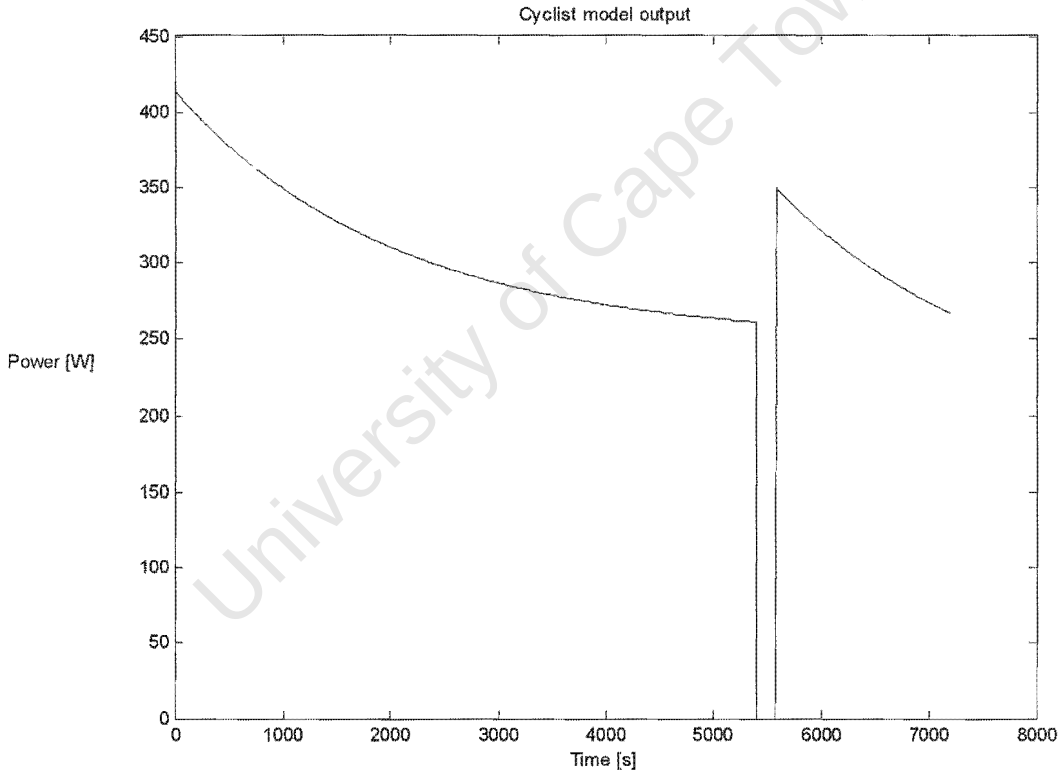
### D.1.1 Cyclist Model

The cyclist model is a decaying exponential curve, as follows:

$$P_{cyc} = P_{transient}\, e^{-Rate(t-180)} + P_{steady}$$

Each cyclist is measured on a time trial basis. This allows one to determine his steady state power output $P_{steady}$, transient power output $P_{tranisient}$ and decay rate $Rate$. Thus, an average endurance cyclist will start pedalling at 300 W and decay to 200 W over two hours. While a sprinter will start with 450 W and decay to 175 W over two hours.

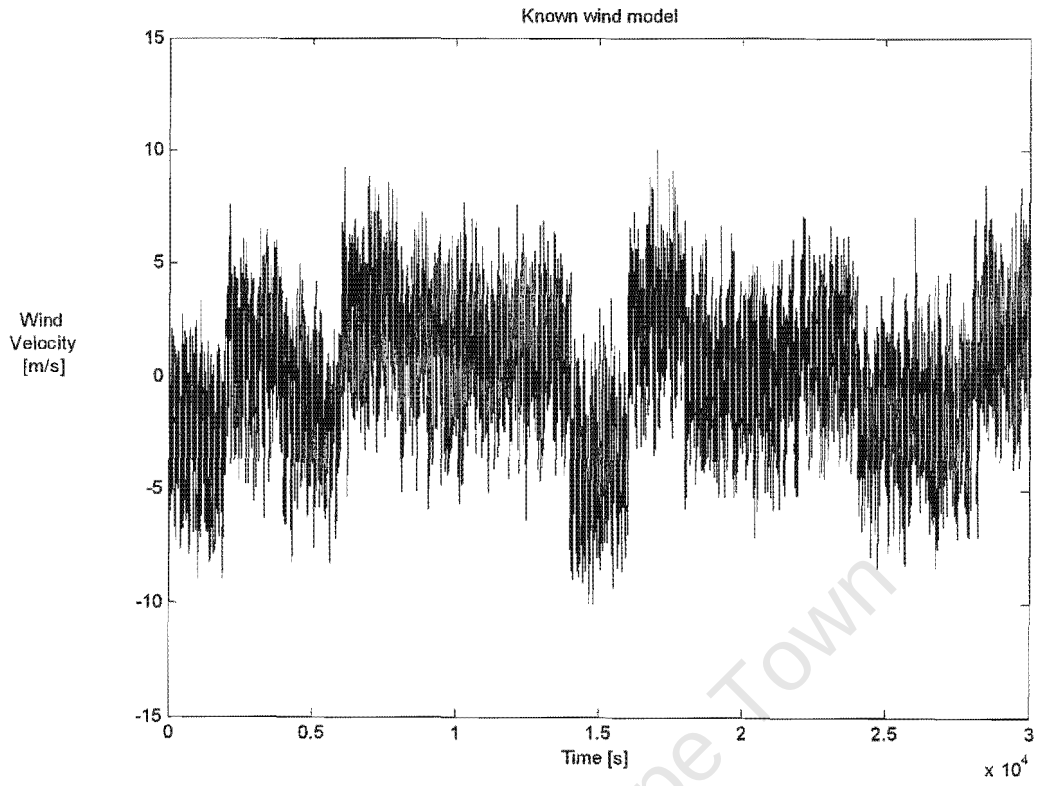
The figure below illustrates the model implemented in `cyclist.m`, which includes a cyclist change after 1.5 hours. The model starts with the best cyclist, a sprinter, and swaps to a good cyclist.



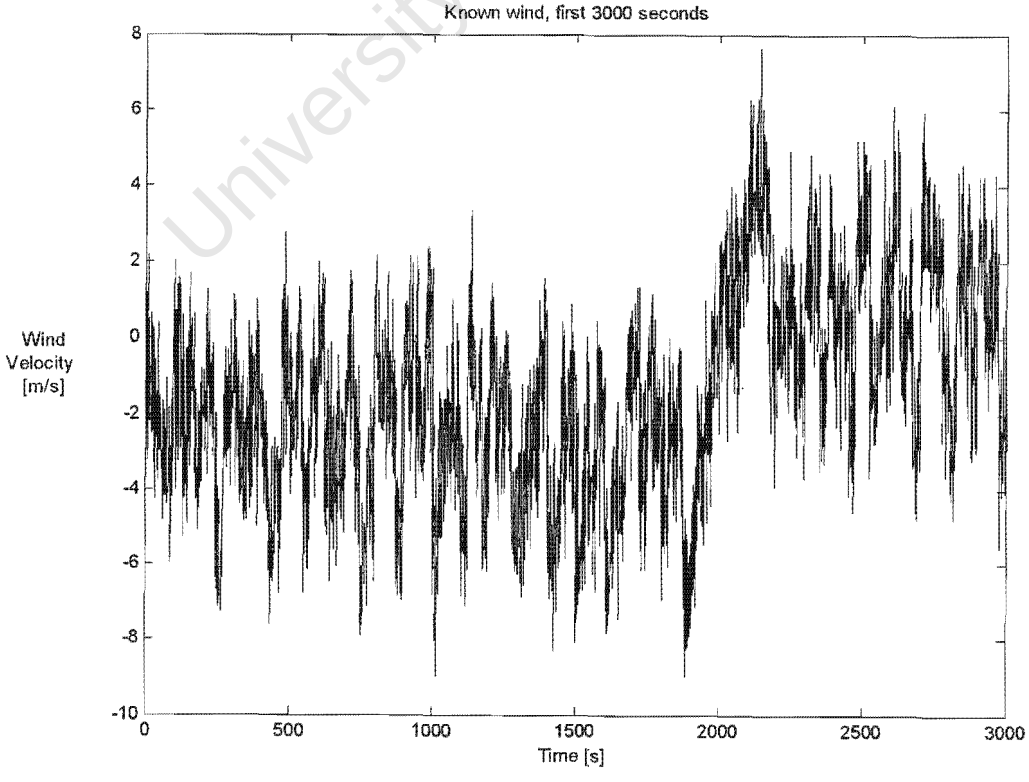



### D.1.2 Wind Model

The wind model is the sum of three potential wind sources. They are small-scale effects, e.g. turbulence from passing traffic, large-scale effects due to terrain and meteorological wind effects. Each model is a generated using a gaussian random number generator scaled according to the velocity and time periods dictated by the model. For example, the turbulence model is scaled to a quarter of the maximum wind velocity, say 5 kph of 20 kph and changes every second, while the meteorological model is scaled to half of the maximum wind velocity and changes after about 30 minutes. The output is very noisy as shown in the figure below.




The figure covers 30000 seconds, or 8.33 hours, the worst-case perfomance for any given WSCC stage. To figure below illustrates the portion of the wind model used in the simulations of section 8.



For the first 2000 seconds the wind is a tailwind, before swapping to a headwind. The magnitude of the velocity is less than that experienced on the test runs, so the model needs some further work to be an accurate model of the wind experienced. The captured by the wind speed measurement circuitry data should be used as a basis for the improved model.

## D.2 Structure of Simulations

The development of the simulations took a meandering and evolutionary path. As such, the files are littered with comments referring to obsolete functions and comments that are obsolete themselves. Frequently, a development in one area was cross-pollinated with another area to improve that areas performance. A case in particular is the solvers used for the MTP. They are the same used for the TSPC, but some derivatives cannot be used in the TSPC.

The overall structure is to use a single script to call a particular solver to operate on a selected route with a given set of conditions. Most of this functionality has been implemented using indices. The select of the solvers is still performed manually in the TSPC.

### D.2.1 Minimum Time Problem Simulation

The file structure is given in a nested format as per the function calling.

- steady.m
  - protomass.m
  - aero.m
  - TSPC_cycle.m
    - protomass.m
    - aero.m
    - solar.m
    - cyclist.m
    - wind.m
    - cycle_DE.m
    - cycle_DE_wind.m
  - TSPC_cycle_cP.m
    - protomass.m
    - aero.m
    - solar.m
    - cyclist.m
    - wind.m
    - cycle_DE.m
    - cycle_DE_wind.m
  - TSPC_solver_PD2.m
    - gradproj.m
      - cycle_PD2.m
        - protomass_opt.m
        - aero.m
        - cyclist_opt.m
        - cycle_DE_wind.m
      - cycle_PD2_grad.m

- cycle_PD2.m
  - protomass_opt.m
  - aero.m
  - cyclist_opt.m
  - cycle_DE_wind.m
- NM_cycle_veri.m
  - cycle_DE_wind_opt.m

- TSPC_solver_NM.m
  - nelderPP.m
    - cycle_NM.m
      - cycle_DE_wind.m
  - NM_cycle_veri.m
    - cycle_DE_wind_opt.m

### D.2.2 Two Stage Predictive Controller Simulation

- tester_converg.m
  - wind.m
  - TSPC_cycle_opt.m
    - protomass.m
    - aero.m
    - cyclist.m
    - cycle_DE.m
    - cycle_DE_wind.m
  - TSPC_SM_opt.m
    - TSPC_solver_opt.m
      - protomass.m
      - aero.m
      - cyclist.m
      - cycle_DE.m
      - cycle_DE_wind.m
    - TSPC_solver_PD2.m
      - gradproj.m
        - cycle_PD2.m
          - protomass_opt.m
          - aero.m
          - cyclist_opt.m
          - cycle_DE_wind.m
        - cycle_PD2_grad.m
          - cycle_PD2.m

- protomass_opt.m
- aero.m
- cyclist_opt.m
- cycle_DE_wind.m

- NM_cycle_veri.m
  - cycle_DE_wind_opt.m

- TSPC_solver_NM.m
  - nelderPP.m
    - cycle_NM.m
      - cycle_DE_wind.m
  - NM_cycle_veri.m
    - cycle_DE_wind_opt.m
- TSPC_solver_opt2.m
  - cycle_opt.m
    - cyclist_opt.m
    - cycle_DE_wind.m
- LMSerror.m

# D.3 MPT and TSPC MATLAB Code Listing

## D.3.1 steady.m

```
% This is the latest constant power and constant velocity simulation.
% It is developed to use the TSPC_cycle function, and is considerably
quicker than the old versions
% based on the Energy Balance code.

clear;
tic;

% Program control flags:
selectroute = 8;    % 1 for day one, 'til day 7 of WSCC, 8 for short route,
9 for field trial route.
dt = 1;                     % time step in seconds.
Simulation = 8;     % Constant power (1) or constant velocity (2)
simulation. (3) for second constant power , (4) for 2nd constant velocity
sim.
                    % 5 - Constant accerelation.
                    % 6 - Constant velocity with Regeneration.
                    % 7 - Projected Gradient -> very slow ~5hr per
iteration!
                    % 8 - Nelder-Mead

showplot = 1;       % Set for plotting.
Wind = 1;

% Variuos other parameters:
Shift_time = 1.5; % hours.
Shift = Shift_time*3600;
```

```
% Physical parameters:
g = 9.81;
Capacity = 750*3600;    % Reduce capacity in proportion to stage length and
forecast horizon length.


switch selectroute      % default to shortroute
case 1
    load routedata;
    ddist = dist1;
    ggrad = grad1;
    tspan_init = 20000;   % Starting traj time span in seconds.
    LL = 1;
    tol = 0.001*Capacity;
case 2
    load routedata;
    ddist = dist2;
    ggrad = grad2;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 3
    load routedata;
    ddist = dist3;
    ggrad = grad3;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 4
    load routedata;
    ddist = dist4;
    ggrad = grad4;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 5
    load routedata;
    ddist = dist5;
    ggrad = grad5;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 6
    load routedata;
    ddist = dist6;
    ggrad = grad6;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 7
    load routedata;
    ddist = dist7;
    ggrad = grad7;
    tspan_init = 30000;   % Starting traj time span in seconds.
case 9
    load fielddata;
    ddist = dist;
    ggrad = grad;
    tspan_init = 10000;   % Starting traj time span in seconds.
otherwise
    load shortroute;
    ddist = srd;
    ggrad = srg;
```

```
    Capacity = Capacity/4;
    tspan_init = 4000;   % Starting traj time span in seconds.
    LL = 1;
    tol = 0.001*Capacity;
end


% Load cycle parameters:
Prototype;
Diameter = D;   % From prototype.m, rename as D is used for Drag.
[cg,M,cg_h] = protomass;
aero_In = [L,D,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters listed
in Trike.m
Out = aero(aero_In);
CdA = real(Out(1)); % starting value - cycle's actual value.
CdA_prev = CdA;


% Initial power traj to get the cycle rolling.

%Pbatt_max = 500;    % Watts.
Pbatt_max = 20000;  % In the case of no power limit.
%Pbatt = random('norm',450,150,tf_limit/dt,1); % Use a normal distibution
about 15, with a spread of 1? Plot looks acceptable.
%Pbatt = 0.5*Pbatt_max*ones(tspan_init,1);  % half power initial traj.
Pbatt = 0.1*Pbatt_max*ones(tspan_init,1);  % half power initial traj.
route = zeros(length(ddist),2);
for i = 1:1:length(ddist),
    route(i,1) = ddist(i);
    route(i,2) = ggrad(i);
end


switch Simulation
case 1
    % Constant power simulation.
    imbalance = 5000;
    Pbatt_hist = [Pbatt];
    conv_hist = [Capacity];
while abs(imbalance) > tol,     % Scale according to capacity.

    % Evaluate power level:
    [s_c,v_c,Ebatt_c] =
TSPC_cycle(route,dt,Shift_time,0,0,0,Pbatt,Capacity,Wind);    % I.C.s are
zero for t,v,s.
    % Correct the power level:
    imbalance = max(Ebatt_c) - Capacity;
    % Control law
    %d_P = - K*a + Kv*v - LL*(Ebatt(round(t_end/dt) - t0) - Capacity)/(t_end
- t0) + 0*grad;
    d_P = - LL*imbalance/length(Pbatt);
    d_P = d_P';
    %figure;
    %plot(d_P);
```

```
    Pbatt = Pbatt + d_P;
    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
        elseif Pbatt(i) < 0,
            Pbatt(i) = 0;
        end
%        if v(i) > 33, Pbatt(i) = 0; end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

        sprintf('Current imbalance is %0.5g',imbalance)
%        sprintf('Current end time is %0.5g',t_end)
    % update historical data.
    Pbatt_hist = [Pbatt_hist,Pbatt];
    conv_hist = [conv_hist,imbalance];


end
% End of constant power sim

case 2
    % Constant velocity simulation - with steady energy constaint and power
constraint. Need to extend to self seeking.
    %v_ref = 20;
    imbalance = 5000;
    Pbatt_hist = [Pbatt];
    conv_hist = [Capacity];
    stdvel = 10;
    stdvel_hist = [stdvel];
while ((abs(imbalance) > tol)|(stdvel > 1)),%0.5)),     % Scale according to
capacity.

    % Evaluate power level:
    [s_c,v_c,Ebatt_c] = TSPC_cycle(route,dt,Shift_time,0,0,0,Pbatt,Wind);
% I.C.s are zero for t,v,s.

    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Determine the mean and std dev of velocity:
```

```
    vel = v_c(1:1:t);
    meanvel = mean(vel);
    stdvel = std(vel);
    % Correct the power level:
    imbalance = max(Ebatt_c) - Capacity;
    % Control law
    % For short route & no wind:
%    K = 12;
%    LL = 1.05*LL;
    % for short route and wind.
    K = 12;
    %LL = 1.01*LL;
    LL = LL + 0.01;
    % Long routes:
%    K = 3;
%    LL = 0.65*LL;
%    LL = 0;
    %d_P = - K*a + Kv*v - LL*(Ebatt(round(t_end/dt) - t0) - Capacity)/(t_end
- t0) + 0*grad;
    d_P = -K*stdvel*(vel - meanvel) - LL*imbalance/length(vel);
%    d_P = K*(v_ref - vel) - LL*imbalance/length(vel);
    d_P = d_P';
    %figure;
    %plot(d_P);

    for i = 1:1:length(Pbatt),
        if i <= length(d_P),
            PP(i) = d_P(i);
        else
            PP(i) = 0;
        end
    end
    Pbatt = Pbatt + PP';
    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
        elseif Pbatt(i) < 0,
            Pbatt(i) = 0;
        end
%        if v(i) > 33, Pbatt(i) = 0; end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

        sprintf('Current imbalance is %0.5g',imbalance)
```

```
        sprintf('Current stdvel is %0.5g',stdvel)
%        sprintf('Current end time is %0.5g',t_end)
    % update historical data.
    Pbatt_hist = [Pbatt_hist,Pbatt];
    conv_hist = [conv_hist,imbalance];
    stdvel_hist = [stdvel_hist,stdvel];


end
% End of constant velocity simulation.

case 3
    % 2nd constant power simulation. No energy constraint.
    % Constrain Pbatt to reasonable bounds.
    Pbatt_max = 500;    % Watts.
    %Pbatt = random('norm',450,150,tf_limit/dt,1); % Use a normal
distibution about 15, with a spread of 1? Plot looks acceptable.
    Pbatt = Pbatt_max*ones(tspan_init,1);  % half power initial traj.

    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Pbatt(i) < 0,
            Pbatt(i) = 0;
        end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

    % Evaluate power level:
    [s_c,v_c,Ebatt_c] =
TSPC_cycle_cP(route,dt,Shift_time,0,0,0,Pbatt,Capacity,Wind);    % I.C.s are
zero for t,v,s.

    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Determine the mean and std dev of velocity:
    vel = v_c(1:1:t);
    meanvel = mean(vel);
    stdvel = std(vel);
    % Correct the power level:
    imbalance = max(Ebatt_c) - Capacity;


        sprintf('Current imbalance is %0.5g',imbalance)
```

```
        sprintf('Current stdvel is %0.5g',stdvel)

    % End of 2nd constant power sim.

case 4
        % Constant velocity simulation - with or without power constraint.
        stdvel = 10;
        v_ref = 20;
%        Pbatt_max = 20000; % For without power constraint.
        Pbatt_max = 500;    % Power constraint.
while (stdvel > 0.51),     % Scale according to capacity.

    % Evaluate power level:
    [s_c,v_c,Ebatt_c] =
TSPC_cycle_cP(route,dt,Shift_time,0,0,0,Pbatt,Capacity,Wind);    % I.C.s are
zero for t,v,s.

    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Determine the mean and std dev of velocity:
    vel = v_c(1:1:t);
    meanvel = mean(vel);
    stdvel = std(vel);
    % Correct the power level:
%    imbalance = max(Ebatt_c) - Capacity;
    % Control law
    % For short route:
%    K = 18; % For wind
    K = 18;
    %d_P = - K*a + Kv*v - LL*(Ebatt(round(t_end/dt) - t0) - Capacity)/(t_end
- t0) + 0*grad;
    %d_P = - K*a
    %d_P = -K*stdvel*(vel - meanvel);
    d_P = K*(v_ref - vel);
    d_P = d_P';
    %figure;
    %plot(d_P);

    for i = 1:1:length(Pbatt),
        if i <= length(d_P),
            PP(i) = d_P(i);
        else
            PP(i) = 0;
        end
    end
    Pbatt = Pbatt + PP';
    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Ebatt_c(i) >= Capacity,    % Energy constraint.
```

```
            Pbatt(i) = 0;
        end
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
        elseif Pbatt(i) < 0,
            Pbatt(i) = 0;
        end
%        if v(i) > 33, Pbatt(i) = 0; end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

%        sprintf('Current imbalance is %0.5g',imbalance)
        sprintf('Current stdvel is %0.5g',stdvel)
%        sprintf('Current end time is %0.5g',t_end)


end
% End of 2nd constant velocity simulation.

case 5
    % Constant acceleration simulation - with steady energy constaint and
power constraint. Need to extend to self seeking.
    %v_ref = 20;
    imbalance = 5000;
    Pbatt_hist = [Pbatt];
    conv_hist = [Capacity];
    stdacc = 10;
    stdvel = 10;
    stdacc_hist = [stdacc];
while ((abs(imbalance) > tol)|(stdacc > 0.11)|(stdvel > 1.5)),%0.5)),     %
Scale according to capacity.

    % Evaluate power level:
    [s_c,v_c,Ebatt_c] = TSPC_cycle(route,dt,Shift_time,0,0,0,Pbatt,Wind);
% I.C.s are zero for t,v,s.

    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Correct the power level:
    %a = 1./(M*v_c).*(Pbatt' + P_sol + P_cyc - (D*v_c.^3 + Resist*v_c.^2 +
grad.*v_c));
    a = zeros(1,t/dt);
    a(1) = 5;
```

```
    for i = 2:1:length(a),
        a(i) = (v_c(i) - v_c(i-1))/dt;  % Approx.
        t = i*dt;
        if a(i) > 5, a(i) = 5; end
        if a(i) < -5, a(i) = -5; end
        %if i > tf/dt, a(i) = 0; end
         % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           a(i) = 0;
       end
    end
    % Determine the mean and std dev of acceleration:
    acc = a(1:1:t);
    acc1 = a(4:1:t);
    meanacc = mean(acc);
    stdacc = std(acc1);
    vel = v_c(1:1:t);
    meanvel = mean(vel);
    stdvel = std(vel);


    imbalance = max(Ebatt_c) - Capacity;
    % Control law
    % for short route and wind.
    if Pbatt_max < 1000,
        if Wind > 0,
            Ka = 1;
            Kv = 1.5;
            LL = LL;
        else
             Ka = 50;
             Kv = 1;
             LL = 1;
%            LL = 1.06*LL;
        end
    else
%        Ka = 50;
        Ka = 150;
        % Kv = 2;
        Kv = 8;
        LL = 1.03*LL;
%        LL = 1.09*LL;
%        LL = 1.5;

    end
    d_P = -Ka*stdacc*(acc - meanacc) - Kv*stdvel*(vel - meanvel) -
LL*imbalance/length(acc);
    d_P = d_P';
   % figure;
   % plot(d_P);
```

```
    for i = 1:1:length(Pbatt),
        if i <= length(d_P),
            PP(i) = d_P(i);
        else
            PP(i) = 0;
        end
    end
    Pbatt = Pbatt + PP';
    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
            % To simulate regen, remove the following two lines.
            %        elseif Pbatt(i) < 0,
           % Pbatt(i) = 0;
        end
%         if v(i) > 33, Pbatt(i) = 0; end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

        disp(sprintf('Current imbalance is %0.5g',imbalance));
        disp(sprintf('Current stdacc is %0.5g',stdacc));
        disp(sprintf('Current stdvel is %0.5g',stdvel));
%         sprintf('Current end time is %0.5g',t_end)
    % update historical data.
    Pbatt_hist = [Pbatt_hist,Pbatt];
    conv_hist = [conv_hist,imbalance];
    stdacc_hist = [stdacc_hist,stdacc];


end
% End of constant acceleration simulation.

case 6
    % Constant velocity simulation - as case 2, with regen.
    %v_ref = 20;
    imbalance = 5000;
    Pbatt_hist = [Pbatt];
    conv_hist = [Capacity];
    stdvel = 10;
    stdvel_hist = [stdvel];
while ((abs(imbalance) > tol)|(stdvel > 0.75)),     % Scale according to
capacity.
```

```
    % Evaluate power level:
    [s_c,v_c,Ebatt_c] = TSPC_cycle(route,dt,Shift_time,0,0,0,Pbatt,Wind);
% I.C.s are zero for t,v,s.

    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Determine the mean and std dev of velocity:
    vel = v_c(1:1:t);
    meanvel = mean(vel);
    stdvel = std(vel);
    % Correct the power level:
    imbalance = max(Ebatt_c) - Capacity;
    % Control law
    % For short route & no wind:
%    K = 12;
%    LL = 1.05*LL;
    % for short route and wind.
    K = 8;
    %LL = 1.01*LL;
    LL = LL;
    % Long routes:
%    K = 3;
%    LL = 0.65*LL;
%    LL = 0;
    %d_P = - K*a + Kv*v - LL*(Ebatt(round(t_end/dt) - t0) - Capacity)/(t_end
- t0) + 0*grad;
    d_P = -K*stdvel*(vel - meanvel) - LL*imbalance/length(vel);
%    d_P = K*(v_ref - vel) - LL*imbalance/length(vel);
    d_P = d_P';
    %figure;
    %plot(d_P);

    for i = 1:1:length(Pbatt),
        if i <= length(d_P),
            PP(i) = d_P(i);
        else
            PP(i) = 0;
        end
    end
    Pbatt = Pbatt + PP';
    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = i*dt;
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
            % Allow regen.
            %        elseif Pbatt(i) < 0,
            %Pbatt(i) = 0;

        end
```

```
        if Pbatt(i) < -Pbatt_max,
            Pbatt(i) = -Pbatt_max;
        end
%        if v(i) > 33, Pbatt(i) = 0; end
        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           Pbatt(i) = 0;
        end

    end

        sprintf('Current imbalance is %0.5g',imbalance)
        sprintf('Current stdvel is %0.5g',stdvel)
%        sprintf('Current end time is %0.5g',t_end)
    % update historical data.
    Pbatt_hist = [Pbatt_hist,Pbatt];
    conv_hist = [conv_hist,imbalance];
    stdvel_hist = [stdvel_hist,stdvel];


end
% End of constant velocity simulation - with regen.
case 7

    if Wind == 1,
        load winddata;
    elseif Wind == 2,
        v_wind = wind(tf_limit);
        %Wind = v_wind;
    else
        v_wind = 0;
    end


    t = 0;
    v0 = 0;
    s0= 0;
    horizon = max(ddist);
    disp('Projected gradient solver.');
    %[s_out,v_out,Ebatt_c,t,Pbatt] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_inc,horiz
on);
    %[s_out,v_out,Ebatt_c,t,Pbatt] =
TSPC_solver_PD2(2,route,dt,t,v0,s0,Capacity,v_wind,horizon);
    % Split into four stages:
    stage = round(length(route)/4);
    [s1_out,v1_out,Ebatt1_c,t1,Pbatt1] =
TSPC_solver_PD2(2,route(1:stage,:),dt,t,v0,s0,Capacity/4,v_wind,horizon);
    [s2_out,v2_out,Ebatt2_c,t2,Pbatt2] =
TSPC_solver_PD2(2,route(stage:2*stage,:),dt,t1,v1_out(t1),s1_out(t1),Capacit
y/4,v_wind,horizon);
```

```
    [s3_out,v3_out,Ebatt3_c,t3,Pbatt3] =
TSPC_solver_PD2(2,route(stage*2:stage*3,:),dt,t2,v2_out(t2),s2_out(t2),Capac
ity/4,v_wind,horizon);
    [s4_out,v4_out,Ebatt4_c,t4,Pbatt4] =
TSPC_solver_PD2(2,route(stage*3:stage*4,:),dt,t3,v3_out(t3),s1_out(t3),Capac
ity/4,v_wind,horizon);
    s_out = [s1_out,s2_out,s3_out,s4_out];
    v_out = [v1_out,v2_out,v3_out,v4_out];
    Ebatt_out = [Ebatt1_out,Ebatt2_out,Ebatt3_out,Ebatt4_out];
    Pbatt_out = [Pbatt1_out,Pbatt2_out,Pbatt3_out,Pbatt4_out];
    %end

case 8
    if Wind == 1,
        load winddata;
    elseif Wind == 2,
        v_wind = wind(tf_limit);
        %Wind = v_wind;
    else
        v_wind = 0;
    end


    t = 0;
    v0 = 0;
    s0= 0;
    horizon = max(ddist);

    disp('Nelder-Mead solver.');
%    [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_inc,horizo
n);
    [s_out,v_out,Ebatt_c,t,Pbatt] =
TSPC_solver_NM(2,route,dt,t,v0,s0,Capacity,v_wind,horizon);

    %end

otherwise
    disp(sprintf('Whoa, Sheila! We have a problem!'));
end

% Add a plotting section.

if showplot == 1,
% Plot velocity and power.


switch Simulation,
case 1
    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
```

```
% Determine the mean and std dev of velocity:
vel = v_c(1:1:t);
figure;
plot(vel);
title('Velocity profile for constant power controller.');
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
    text(500,2,'Day6');
case 7
    text(500,2,'Day7');
case 8
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
ylabel('Velocity [m/s]');
text(t/2,2,sprintf('Final time is %0.5g s.',t));

figure;
plot(Pbatt);
title('Battery Power Trajectory for constant power controller.');
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
    text(500,2,'Day6');
case 7
    text(500,2,'Day7');
case 8
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
```

```
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

case 2
    t = dt*length(vel);
    figure;
    plot(vel);
    title('Velocity profile for Constant Velocity Controller.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

    figure;
    plot(Pbatt);
    title('Battery Power Trajectory for Constant Velocity Controller.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
```

```
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));


case 3
    % Determine final time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
    % Determine the mean and std dev of velocity:
    vel = v_c(1:1:t);
    figure;
    plot(vel);
    title('Velocity profile for constant power controller2.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

    figure;
    plot(Pbatt);
    title('Battery Power Trajectory for constant power controller2.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
```

```
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

       figure;
    plot(Ebatt_c);
    title('Battery Energy Profile for constant power controller2.');
    switch selectroute,
    case 1
        text(500,2000,'Day1');
    case 2
        text(500,2000,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2000,'Short route');
    case 9
        text(500,2000,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Energy [J]');
    text(t/2,2000,sprintf('Final time is %0.5g s.',t));

case 4
    t = dt*length(vel);
    figure;
    plot(vel);
    title('Velocity profile for Constant Velocity Controller - without
energy constraint.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
```

```
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

    figure;
    plot(Pbatt);
    title('Battery Power Trajectory for Constant Velocity Controller -
without energy constraint.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

case 5
    t = dt*length(vel);
    figure;
    plot(vel);
    title('Velocity profile for Constant Acceleration Controller.');
```

```
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
    text(500,2,'Day6');
case 7
    text(500,2,'Day7');
case 8
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
ylabel('Velocity [m/s]');
text(t/2,2,sprintf('Final time is %0.5g s.',t));

figure;
plot(acc);
title('Acceleration profile for Constant Acceleration Controller.');
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
    text(500,2,'Day6');
case 7
    text(500,2,'Day7');
case 8
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
ylabel('Acceleration [m/s/s]');
text(t/2,2,sprintf('Final time is %0.5g s.',t));


figure;
```

```
    plot(Pbatt);
    title('Battery Power Trajectory for Constant Acceleration.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

case 6
    t = dt*length(vel);
    figure;
    plot(vel);
    title('Velocity profile for Constant Velocity Controller - with
Regen.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
```

```
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

    figure;
    plot(Pbatt);
    title('Battery Power Trajectory for Constant Velocity Controller - with
Regen.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

case 7
%    t = dt*length(vel);
    figure;
        vel = v_out;
    plot(v_out);
    title('Velocity profile for ProjGrad.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
```

```
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
ylabel('Velocity [m/s]');
text(t/2,2,sprintf('Final time is %0.5g s.',t));

figure;
plot(Pbatt);
title('Battery Power Trajectory for ProjGrad.');
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
    text(500,2,'Day6');
case 7
    text(500,2,'Day7');
case 8
    text(500,2,'Short route');
case 9
    text(500,2,'Field test');
end
xlabel('Time [s]');
ylabel('Power [W]');
text(t/2,2,sprintf('Final time is %0.5g s.',t));

case 8
t = dt*length(v_out);
figure;
vel = v_out;
plot(v_out);
title('Velocity profile for NM.');
switch selectroute,
case 1
    text(500,2,'Day1');
case 2
    text(500,2,'Day2');
case 3
    text(500,2,'Day3');
case 4
    text(500,2,'Day4');
case 5
    text(500,2,'Day5');
case 6
```

```
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

    figure;
    plot(Pbatt);
    title('Battery Power Trajectory for NM.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));


otherwise
    disp('Oops!');
end

disp(sprintf('Mean velocity %0.5g',mean(vel)));
disp(sprintf('Final time is %0.5g s.',t));
end % of plotting

toc;
```

## D.3.2 protomass.m

```
% Function, called by stab02.m
```

```
% And route simulations.
% Cleaned up for Solartrike3. 31-10-99.
% Modified for a single front wheel and two rear wheels.
% Initially guestimated.
% Using prototype frame dim.


% Estimate of mass and C.G.
% All masses is kg, position relative to front axle, positive toward rear
axle.

function [CG,T_mass,CG_h] = protomass();

Prototype;

% Components masses, positions(in portion of wheelbase) and heights (in
portion of wheel radius).
WR = D_wheel/2;
% Frame
m_frame = 10.8;
p_frame = wheelbase;      % Est.
h_frame = D_wheel;

% Battery    Swapped to NiMH/SLA
m_batt = 24;%13.7;
p_batt = 1.1*wheelbase;
h_batt = h_frame - 0.05;

% Motor
m_mot = 3;
p_mot = 0;
h_mot = WR + 0.05;

% Controller
m_cont = 1;
p_cont = 0.05;
h_cont = h_frame;


% Front Wheel
m_fw = 0.5;
p_fw = 0*wheelbase;
h_fw = WR;

% Rear Wheels
m_rw = 2*0.5;
p_rw = wheelbase;
h_rw = WR;

% Cyclist
m_cyc = 80;  % Rael: 80, Gordon: 60.
p_cyc = 1.5; % Est.
h_cyc = h_frame-0.2;
```

```
% Body
m_body = 6;  % Est.
p_body = wheelbase*0.5;    % Est.
h_body = h_frame+0.1;

% Ballast !?!!
m_ball = 0;
p_ball = 0;
h_ball = 0;

% Total mass
T_mass = m_body + m_frame + m_cyc + m_batt + m_mot + m_cont + m_fw + m_rw +
m_ball;

% C.G.
CG =
(m_body*p_body+m_frame*p_frame+m_cyc*p_cyc+m_batt*p_batt+m_mot*p_mot+m_cont*
p_cont+m_fw*p_fw+m_rw*p_rw+m_ball*p_ball)./T_mass;

% For height of CG:
CG_h =
(m_body*h_body+m_frame*h_frame+m_cyc*h_cyc+m_batt*h_batt+m_mot*h_mot+m_cont*
h_cont+m_fw*h_fw+m_rw*h_rw+m_ball*h_ball)./T_mass;

%out = [CG,T_mass];
```

## D.3.3 aero.m

```
% Aerodynamic synthesis of trike, based on Tamai's work p220+. Esp. Table
5.1.2
% Imports various parameters of vehicle to obtain the Cd_wet, CdA_wet, etc.


function Out = aero(In);

% This analysis uses A_wet, which can be calculated from CAD packages, but
A_plan will need to be used later'

% Input parameters:
L = In(1);    % Vehicle length.
D = In(2);    % Vehicle diameter.
x = In(3);    % Transition point.
V = In(4);    % Freestream velocity.

if nargin == 8,
   A_wet = In(5);   % Wetted area of body from CAD.
   A_wet_wing = In(6);     % Wetted area of wings and wheels from CAD.
   L_wing = In(7);  % Chord of wing.
   t_c = In(8);            % Thickness ratio of wing.
elseif nargin == 6,
   A_wet = In(5);
   A_wet_wing = In(6);
```

```
   L_wing = 1;
      t_c = 0.09;
else
   A_wet = 15;
   A_wet_wing = 10;
   L_wing = 1;
   t_c = 0.09;
end

% Assume that the transition point fo rthe wing&wheels is 10% less than that
of the body:
x_wing = 0.9*x;

% Physical parameters:
nu = 1.46e-5;% Air viscosity, at what conditions??



% <<<<<<<<< Flat Plate >>>>>>>>>>>

% Tamai has modeled his flat plate data on smooth turbulent flow eqns, i.e.:
% For body:
Re_x = V*(x*L)/nu;  % Where x is an imposed transistion point.
if Re_x <= 0, Re_x = 1; end;   % Prevent singularities @ V=0.


Cf_flat = 0.031/(Re_x)^(1/7);    % This is a bit of an approximation, as it
essentially assumes that the flow is turbulent.

% For wings:
Re_x = V*(x_wing*L_wing)/nu;
if Re_x <= 0, Re_x = 1; end;   % Prevent singularities @ V=0.

Cf_flat_wing = 0.031/(Re_x)^(1/7);

% <<<<<<<< Cd_wet >>>>>>>>>>>>>

% For Cd_wet of torpedo:
Cd_wet = Cf_flat*(1 + 1.5*(D/L)^1.5 + 7*(D/L)^3);
% For TFT:
Cd_TFT = Cf_flat*(1 + 1.5*(D/L)^1.5 + 19*(D/L)^6);
% For wings:
A_plan_wing = L_wing*2 + L_wing*2*1;    % Wing and 2 wheels.
Cd_wet_wing = 2*A_plan_wing/A_wet_wing*Cf_flat_wing*(1 + 2*t_c + 60*t_c^4);


% <<<<<<<<<<<<< CdA_wet >>>>>>>>>>>>>

CdA_wet = Cd_wet*A_wet;
CdA_wet_wing = Cd_wet_wing*A_wet_wing;
TFT = Cd_TFT*A_wet;
% Hence:
CdA_clean = CdA_wet + CdA_wet_wing;
```

```
%<<<<<<<<<<< Add influence of other items >>>>>>>>>>>>>

% Wheels, spats, etc:
CdA_front_wheel = 6e-3;   %Swept full fairing (LE & TE), sealed (even for
steering).
CdA_rear_wheels = 2*0.01; % Full fairing, sealed. Already partly included on
wing, but completely included here to be conservative.

% Ventilation:
% Currently assumed nil.
% Envisaged as a nose inlet directing air to cool cyclist's head and
shoulders and PE and gennie/motor. Whole system pressurised.

CdA_other = CdA_front_wheel + CdA_rear_wheels;

%<<<<< Total >>>>>>>

CdA = CdA_clean + CdA_other;




Out = [CdA;CdA_wet;CdA_wet_wing;CdA_other;TFT];




% Solar Viscocity Ratio:
%SVR = A_panel_plan/(CdA_wet);
```

## D.3.4 TSPC_cycle.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period.
% Includes random wind effect.

% Calls: cycle_DE.m, cyclist.m, solar.m, aero.m, protomass.m, prototype.m
% Called by: Steady.m, TSPC.m

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
function [s_out,v_out,Ebatt_out] =
TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt,Wind);

if nargin == 7,
    Wind = 0;
end

Shift = Shift_time*3600;

ddist = route(:,1);
```

```
ggrad = route(:,2);

% Physical parameters:
g = 9.81;

% Load cycle parameters:
Prototype;
Diameter = D;   % From prototype.m, rename as D is used for Drag.
[cg,M,cg_h] = protomass;
aero_In = [L,D,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters listed
in Trike.m
Out = aero(aero_In);
CdA = real(Out(1)); % Used as initial value.


% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
s(1) = s0;
%s = s0*ones(1,length(Pbatt));
v = zeros(1,length(Pbatt));
v(1) = v0;
%v = v0*ones(1,length(Pbatt));
grad = s;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.
Sol_array = 0.14*1;

%Dist = max(ddist);
%Capacity = 750*3600;    % Reduce capacity in proportion to stage length and
forecast horizon length.
%Capacity = hprop * Capacity;

% Solar parameters:
theta_init  = 70/180*pi;  % Degrees from normal.
theta_dot = theta_init/(3*3600); % Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
%for i = 1:1:tf_limit/dt,
for i = 1:1:length(Pbatt)*dt,
    %theta = theta_init - theta_dot*i*dt;
    P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;

       % Include stopping to change cyclists and a 2 minute pitstop.
    Shift =Shift_time*3600;      % Cyclists shift in hour multiples. Later
to be left to agent.
```

```
    tt = t0 + i*dt;
    P_cyc(i) = cyclist([tt,Shift_time]);

end
if Wind == 1,
%    v_wind = wind(length(Pbatt)*dt);
load winddata;
elseif length(Wind)>1,
    v_wind = Wind;
end



% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
%Dist = (maxdist - mindist);
%if maxdist == mindist,
    Dist = maxdist;
    %end
    % Int system eqn.
    i = 1;
    %while max(s)/1000 <= Dist, %for k = 1:1:Dist,
%    for i = 2:1:length(Pbatt),
while (max(s)/1000 <= Dist) & (i <= length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
%         if jj <= length(ddist),
         if jj < length(ddist),
 %            mindist = min(ddist);
%             maxdist = max(ddist);
%             if mindist == maxdist,
%                jj = 1;
%             else
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%                i = i
                end
                %            end
                %        else
%            jj = length(ddist);
%            disp('route out of range.')

        end
        G = M*g*sin(ggrad(jj)*pi/180);
        grad(i-1) = G;

        %t =(i-1)*dt;    % Elapsed time.
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle.
```

```
        %if length(CdAvector) == 1,
            aero_In = [L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];
       % All parameters listed in Trike.m
            Out = aero(aero_In);
            CdA = real(Out(1));
            D = 0.5*1.2*CdA;
            %else
            %D = 0.5*1.2*CdAvector(i);
            %        end

        % Solve system DEs:
        % simple direct integration approach:
        if Wind == 0,
            Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
            dydt = cycle_DE(t,y0,Para);
        else
            Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            if length(Wind) == 1,
                y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(i-1)];
            else
                y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(t/dt)];
            end
            dydt = cycle_DE_wind(t,y0,Para);
        end

        dEbatt = dydt(1);
        ds = dydt(2);
        dv = dydt(3);
        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
        s(i) = s(i-1) + ds*dt;
%        disp(i)
%        disp(s(i))

        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
           v(i) = 0;
           dv = 0;
        else
            V = v(i-1) + dv*dt;
            if V < 0, V = 0; end
            v(i) = V;

        end
       % i = i + 1;
        t_end = t;
        v_tf = V;
```

```
    end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(2:length(v));
s_out = s(2:length(s));
Ebatt_out = Ebatt(2:length(Ebatt));

% Setup output values.
```

## D.3.5 solar.m

```
% Estimated Daily insolation profile in Out back.

% Initial thumb suck:

function [insolation] = solar(In);

t = In(1);    % Parameter passing by putting all parameters (scalar!) into a
vector.

% Race starts at 09h00, I.C.s:
theta_init  = 70/180*pi;  % Radians from normal. Get used to swapping
between radians and degrees.
% Travels 60 deg in 3 hours in set increments:

theta_dot = theta_init/(3*3600); % Rad per second


theta = theta_init - theta_dot*t;

sun = 1000*cos(theta);
if sun <= 0,
    sun = 0;
end
insolation = sun;
```

## D.3.6 cyclist.m

```
% Cyclist function.
% maps out an exponential curve of the cyclist's steady state output.

function out = cyclist(in);

t = in(1);
ST = in(2);
Shift = ST*3600;
rider = 1;
```

```
if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) & (t<=(2*Shift+180))) |
((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) & (t<=(4*Shift+180))) |
((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) & (t<=(6*Shift+180))) |
((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) & (t<=(8*Shift+180))), %
no o/p.
   out = 0;
else
   if t - 8*Shift > 0,
      t = t - 8*Shift;
      rider = 9;
   elseif t - 7*Shift > 0,
      t = t - 7*Shift;
      rider = 8;
   elseif t - 6*Shift > 0,
      t = t - 6*Shift;
      rider = 7;
   elseif t - 5*Shift > 0,
      t = t - 5*Shift;
      rider = 6;
   elseif t - 4*Shift > 0,
      t = t - 4*Shift;
      rider = 5;
   elseif t - 3*Shift > 0,
      t = t - 3*Shift;
      rider = 4;
   elseif t - 2*Shift > 0,
      t = t - 2*Shift;
      rider = 3;
   elseif t - Shift > 0,
      t = t-Shift;
      rider = 2;
  else
      rider = 1;

   end;
   %t = t


   if (rider == 1)|(rider == 4)|(rider == 9),    % Peter
        out = 150*exp(-0.5e-3.*(t-180)) + 250; % Model valid for Peter.
   elseif (rider == 2)|(rider == 5),    % Andrew
        out = 150*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Andrew.
   elseif (rider == 3)|(rider == 7),    % Mark
        out = 100*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Mark or
Roelie.
    elseif (rider == 6)|(rider == 8),  % Gordon
        out = 100*exp(-0.5e-3.*(t-180)) + 100; % Model valid for Gordon.
    end



   %out = 100*exp(-0.5e-3.*t) + 100; % Model valid for Gordon.
   %out = 100*exp(-0.5e-3.*t) + 250; % Model valid for Peter.
```

```
    %out = 150*exp(-0.5e-3.*t) + 200; % Model valid for Andrew.

end;


%out = 100*exp(-0.5e-3.*t) + 100;
```

## D.3.7 wind.m

```
% Wind function.
% Called by: TSPC_cycle.m,TSPC_solver.m, Steadt.m, Mine.m,
Mine_predictive.m, opt vel traj soln.m, etc.

function v_wind = wind(in);

t = in(1);

v_wind_max = 5;

v_wind_turb = random('norm',0,v_wind_max/4,t,1); % Small scale effects -
turbulence from trees, passing traffic.

v_wind_terrain = random('norm',0,v_wind_max/3,ceil(t/25),1); % Small scale
effects - turbulence from hills, etc.

v_wind_met = random('norm',0,v_wind_max/2,ceil(t/2000),1); % Meteorological
wind effect.

v_wind = zeros(1,t);
for i = 1:1:t,

%    j = 1;
%    while j*1 <= i,
%        j = j + 1;
%    end
    k = 1;
    while k*25 <= i,
        k = k + 1;
    end
    l = 1;
    while l*2000 <= i,
        l = l + 1;
    end
%    if j > t/5, j = t/5; end
    if k > t/25, k = ceil(t/25); end
    if l > t/2000, l = ceil(t/2000); end
    v_wind(i) = v_wind_turb(i) + v_wind_terrain(k) + v_wind_met(l);
end
```

## D.3.8 cycle_DE.m

```
% cycle_DE.m This file contains the system DEs for the cycle.
% for use in ODEslver in calcengineB2.m
% And in own solver, Mine.m
% Also called by: Steady.m, TSPC_cycle.m, TSPC_cycle_cP.m

function dydt = cycle_DE(t,y,p);  %p1,p2,p3,p4,p5,p6,p7);

E_batt = y(1);
s = y(2);
v = y(3);
P_batt = p(1);
P_sol = p(2);
P_cyc = p(3);
D = p(4);
R = p(5);
G = p(6);
M = p(7);

% Prevent v from hitting zero.
if v == 0,
    v = 0.001;
end

y_1 = P_batt;

y_2 = v;

y_3 = 1/(M*v)*(P_batt + P_sol + P_cyc - (D*v^3 + R*v^2 + G*v));

% Limit acceleration to +- g.
if y_3 > 9.81,
    y_3 = 9.81;
elseif y_3 < -9.81,
    y_3 = -9.81;
end

dydt = [y_1;y_2;y_3];
```

## D.3.9 cycle_DE_wind.m

```
% cycle_DE.m This file contains the system DEs for the cycle.
% for use in ODEslver in calcengineB2.m
% And in own solver, Mine.m
% Also called by: Steady.m, TSPC_cycle.m, TSPC_cycle_cP.m

function dydt = cycle_DE_wind(t,y,p);  %p1,p2,p3,p4,p5,p6,p7);

E_batt = y(1);
s = y(2);
v = y(3);
v_wind = y(4);
```

```
P_batt = p(1);
P_sol = p(2);
P_cyc = p(3);
D = p(4);
R = p(5);
G = p(6);
M = p(7);

v_apparent = v + v_wind;

% Prevent v and v_apparent from hitting zero. Prevents singularities in the
y_3, acceleration term.
if v_apparent <= 0,
    v_apparent = 0.001;
end
if v <= 0,
    v = 0.001;
end



y_1 = P_batt;

y_2 = v;

y_3 = 1/(M*v)*(P_batt + P_sol + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));

% Limit acceleration to +- g.
if y_3 > 9.81,
    y_3 = 9.81;
elseif y_3 < -9.81,
    y_3 = -9.81;
end

dydt = [y_1;y_2;y_3];
```



### D.3.10 TSPC_cycle_cP.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period.
% Includes random wind effect.
% Altered to include flat battery for constant power and constant velocity
simulations.

% Calls: cycle_DE.m, cyclist.m, solar.m, aero.m, protomass.m, prototype.m
% Called by: Steady.m

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
function [s,v,Ebatt] =
TSPC_cycle_cP(route,dt,Shift_time,t0,v0,s0,Pbatt,Capacity,Wind);
```

```
if nargin == 7,
    Capacity = 750*3600;
    Wind = 0;
elseif nargin == 8,
    Wind = 0;
end

Shift = Shift_time*3600;

ddist = route(:,1);
ggrad = route(:,2);

% Physical parameters:
g = 9.81;

% Load cycle parameters:
Prototype;
Diameter = D;   % From prototype.m, rename as D is used for Drag.
[cg,M,cg_h] = protomass;
aero_In = [L,D,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters listed
in Trike.m
Out = aero(aero_In);
CdA = real(Out(1)); % Used as initial value.


% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
%s = zeros(1,length(Pbatt));
%s(1) = s0;
s = s0*ones(1,length(Pbatt));
%v = 0.01*ones(1,length(Pbatt));
%v(1) = v0;
v = v0*ones(1,length(Pbatt));
grad = s;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.
Sol_array = 0.14*1;

%Dist = max(ddist);
%Capacity = 750*3600;    % Reduce capacity in proportion to stage length and
forecast horizon length.
%Capacity = hprop * Capacity;

% Solar parameters:
theta_init  = 70/180*pi;  % Degrees from normal.
theta_dot = theta_init/(3*3600); % Deg per second
```

```
% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
%for i = 1:1:tf_limit/dt,
for i = 1:1:length(Pbatt)*dt,
    %theta = theta_init - theta_dot*i*dt;
    P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;

       % Include stopping to change cyclists and a 2 minute pitstop.
    Shift =Shift_time*3600;      % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist([tt,Shift_time]);
end
if Wind == 1,
%    v_wind = wind(length(Pbatt)*dt);
    load winddata;
end


% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
%Dist = (maxdist - mindist);
%if maxdist == mindist,
    Dist = maxdist;
    %end
    % Int system eqn.
    i = 1;
    %while max(s)/1000 <= Dist, %for k = 1:1:Dist,
%    for i = 2:1:length(Pbatt),
while (max(s)/1000 <= Dist) & (i <= length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
%         if jj <= length(ddist),
         if jj < length(ddist),
 %            mindist = min(ddist);
%             maxdist = max(ddist);
%             if mindist == maxdist,
%                 jj = 1;
%             else
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                  jj = jj + 1;
%               i = i
                end
                %            end
                %        else
%            jj = length(ddist);
%            disp('route out of range.')

        end
        G = M*g*sin(ggrad(jj)*pi/180);
```

```
        grad(i-1) = G;

        %t =(i-1)*dt;    % Elapsed time.
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle.

%        if length(CdAvector) == 1,
            aero_In = [L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];
      % All parameters listed in Trike.m
            Out = aero(aero_In);
            CdA = real(Out(1));
            D = 0.5*1.2*CdA;
            %       else
           % D = 0.5*1.2*CdAvector(i);
            %        end

        % Solve system DEs:
        % simple direct integration approach:
        if Wind == 0,
            if Ebatt(i-1) < Capacity,
                Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            else
                Para = [0,P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            end
            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
            dydt = cycle_DE(t,y0,Para);
        else
            if Ebatt(i-1) < Capacity,
                Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            else
                Para = [0,P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            end
            y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(i-1)];
            dydt = cycle_DE_wind(t,y0,Para);
        end

        dEbatt = dydt(1);
        ds = dydt(2);
        dv = dydt(3);
        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
        s(i) = s(i-1) + ds*dt;
%        disp(i)
%        disp(s(i))

        % Include pitstops.
        if ((t > Shift) & (t<=(Shift+180))) | ((t>2*Shift) &
(t<=(2*Shift+180))) | ((t>3*Shift) & (t<=(3*Shift+180))) | ((t>4*Shift) &
(t<=(4*Shift+180))) | ((t>5*Shift) & (t<=(5*Shift+180))) | ((t>6*Shift) &
(t<=(6*Shift+180))) | ((t>7*Shift) & (t<=(7*Shift+180))) | ((t>8*Shift) &
(t<=(8*Shift+180))), % no o/p.
            v(i) = 0;
            dv = 0;
```

```
        else
            V = v(i-1) + dv*dt;
            if V < 0, V = 0; end
            v(i) = V;

        end
       % i = i + 1;
        t_end = t;
        v_tf = V;

    end

% end of main iterative loop.


% Setup output values.
```

## D.3.11 TSPC_solver_PD2.m

```
% This is a wrapper for calling the Kelley's Gradient projjection solver for
the solar cycle problem.
% Developed from TSPC_solver_NM.m

% Optimised for test run.
% No solar power, can ignore pitstops.

% Calls: nelderPP.m, NM_cycle_veri.m
% Called by: TSPC_SM_opt.m

function [s_out,v_out,Ebatt_c,t,Pbatt] =
TSPC_solver_PD2(sl,route,dt,t0,v0,s0,Capacity,Wind,horizon);   % hprop is th
proportion of the horizon length comapred to the stage length, used from
reducing the Capacity used in horizon.

% Function control:
interim = 1;    % Show interim results if set.

%tf_limit = 1500;    % Initial guess, should be adequate.
ddist = route(:,1);
ggrad = route(:,2);

tf_limit = round(horizon*1000/11);  % Time to cover distance at 11m/s -
average speed of UCT cycle.

Pbatt_max = 500;

% Step 1:
% Estimate initial trajectory:

routelength = max(ddist)*1000;
```

```
%tf_limit = round(routelength/11);    % Worst-case, no power, and slow....
% Set initial values to P:
% Initial Power trajectories:
P0 = 0.5*Pbatt_max*rand(tf_limit,1);

% Set PD2 parameters:
tol = 0.001;
maxit = 100;
up = Pbatt_max*ones(size(P0));
low = zeros(size(P0));
% Above are default.
maxit = 1;  % To get things going.
tol = 1;

% Call PD2:
[Pbatt,histout,costdata] =
gradproj(P0,up,low,tol,maxit,Wind,ddist,ggrad,tf_limit,0,Capacity);
%[Pbatt,histout,costdata] =
projbfgs(P0,up,low,tol,maxit,Wind,ddist,ggrad,tf_limit,0,Capacity);



% Verify NM result and compute output traj.:

[Vel_mean,t,s_out,v_out,Ebatt_c] =
NM_cycle_veri(Pbatt,ddist,ggrad,Wind,Capacity,0);

disp(sprintf('time: %0.3d',t));

% Initialize recorded values: including all state variables
% State variables:
%Ebatt = zeros(1,length(Pbatt));
%s_out = zeros(1,length(Pbatt));
%v = v0*ones(1,length(Pbatt));

%%for i = 1:1:length(s_c);
%    s_out(i) = s0 + s_c(i);
%end
```

## D.3.12 gradproj.m

```
%function [x,histout,costdata] = gradproj(x0,f,up,low,tol,maxit)
function
[x,histout,costdata]=gradproj(x0,up,low,tol,maxit,v_wind,ddist,ggrad,tf_limi
t,pplot,Capacity)
%
% C. T. Kelley, June 11, 1998
%
% This code comes with no guarantee or warranty of any kind.
%
% function [x,histout,costdata] = gradproj(x0,f,up,low,tol,maxit)
```

```
%
% gradient projection with Armijo rule, simple linesearch
%
%
% Input: x0 = initial iterate
%        f = objective function,
%            the calling sequence for f should be
%            [fout,gout]=f(x) where fout=f(x) is a scalar
%              and gout = grad f(x) is a COLUMN vector
%        up = vector of upper bounds
%        low = vector of lower bounds
%        tol = termination criterion norm(grad) < tol
%              optional, default = 1.d-6
%        maxit = maximum iterations (optional) default = 1000
%
% Output: x = solution
%         histout = iteration history
%             Each row of histout is
%   [norm(grad), f, number of step length reductions, iteration count,
%            relative size of active set]
%         costdata = [num f, num grad, num hess] (for steep, num hess=0)
%
%
xc=x0; ndim=length(up); kku=zeros(ndim,1); kkl=zeros(ndim,1);
for i=1:ndim
        kku(i)=up(i); kkl(i)=low(i);
      if kkl(i) > kku(i)
        error(' lower bound exceeds upper bound')
        end
end
alp=1.d-4;

%
% put initial iterate in feasible set
%
if norm(xc - kk_proj(xc,kku,kkl)) > 0
      disp(' initial iterate not feasibile ');
      xc=kk_proj(xc,kku,kkl);
end

if length(v_wind) == 1,
    v_wind = v_wind*zeros(1,tf_limit);
end

itc=1;
fc = cycle_PD2(xc,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
gc = cycle_PD2_grad(xc,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,fc);
numf=1; numg=1; numh=0;
ithist=zeros(maxit,5);
xt=kk_proj(xc - gc,kku,kkl);
pgc=xc - kk_proj(xc - gc,kku,kkl);
ia=0;
for i=1:ndim,
```

```
    if(xc(i)==kku(i) | xc(i)==kkl(i)),
        ia=ia+1;
    end;
end;
ithist(1,5)=ia/ndim;
ithist(1,1)=norm(pgc); ithist(1,2) = fc; ithist(1,4)=itc-1; ithist(1,3)=0;
while(norm(pgc) > tol & itc <= maxit)
        lambda=1;
        xt=kk_proj(xc-lambda*gc,kku,kkl);
        ft = cycle_PD2(xt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
        numf=numf+1;
      iarm=0; itc=itc+1;
        pl=xc - xt; fgoal=fc-(pl'*pl)*(alp/lambda);
%
%       simple line search
%
        q0=fc; qp0=-gc'*gc; qc=ft;
        term = 0;   % Armijo loop termination.

      while(ft > fgoal)|term
                lambda=lambda*.1;
            iarm=iarm+1;
            xt=kk_proj(xc-lambda*gc,kku,kkl);
               pl=xc-xt;
            ft = cycle_PD2(xt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
        numf = numf+1;
            if(iarm > 10)
            disp(' Armijo error in gradient projection')
%           histout=ithist(1:itc,:); costdata=[numf, numg, numh];
%           return
           % Run with the error.
           term = 1;
        end
        fgoal=fc-(pl'*pl)*(alp/lambda);
      end
      xc=xt;
    fc = cycle_PD2(xc,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
    gc = cycle_PD2_grad(xc,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,fc);
    numf=numf+1; numg=numg+1;
        pgc=xc-kk_proj(xc-gc,kku,kkl);
      ithist(itc,1)=norm(pgc); ithist(itc,2) = fc;
      ithist(itc,4)=itc-1; ithist(itc,3)=iarm;
ia=0;
for i=1:ndim,
    if(xc(i)==kku(i) | xc(i)==kkl(i)),
        ia=ia+1;
    end
end
ithist(itc,5)=ia/ndim;
end
x=xc;
histout=ithist(1:itc,:);
costdata=[numf, numg, numh];
```

```
%
% projection onto active set
%
function px = kk_proj(x,kku,kkl)
ndim=length(x);
px=zeros(ndim,1);
px=min(kku,x);
px=max(kkl,px);
```

### D.3.13 cycle_PD2.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Derived from cycle_PD.m for use in NM and TSPC with NM.
% Reverted for PD2.

% Input: Pbatt only.
% Output: Vel_ave only!! - scalar.
% Power and battery energy constrains handled in calling function.

% Calls: cycle_PD.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: ????

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
%function [s_out,v_out,Ebatt_out] =
TSPC_cycle_opt(route,dt,t0,v0,s0,Pbatt,Wind);
%function [Vel_ave,t] =
cycle_PD(PP,v_wind,ddist,ggrad,m,uscale,P2,tf_limit,pplot);   % For PD
algor.
function t = cycle_PD2(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
% For PD2 algor.
    % ->> Must return the gradient!! as a column vector. - seperate
function.

    dt = 1;


if pplot == 1,
    figure;
    plot(Pbatt);
end
%pause;


%load winddata;
% v_wind = Wind(t);
```

```
% Physical parameters:
g = 9.81;

% Load cycle parameters:
% Physical parameters:
track = 1.0; % Track in metres. Aero analysis assumes that this is constant.
wheelbase = 2;
L = 3; % Vehicle length.
Diameter = 0.66;    % Vehicle diameter.
x = 0.6;      % Transition point on body.
A_wet = 5.01;% Wetted area of body from CAD.
L_wing = 0.56;      % Chord of wing.
A_wet_wing = 1.8*track*L_wing + 2*.96;
t_c = 0.21;         % Thickness ratio of wing.
% Rolling parameters:
D_wheel = 0.707;    % Wheel diameter m.
Tor_b = 0.005;              % Bearing torque drag. Nm.
% Tyres, ass: Michelin
C0 = 0.005;         % Rolling resistance coeff. Check in Storey et al.
C1 = 0.005;
% Electrical parameters:
nu_motor = 0.9; % Later incorporate the whole curve.
nu_panels = 0.14; % Ditto. Also for temp.
nu_controller = 0.95;
% Panel parameters:
A_panels = 4*0.21;  % Reusing n UCT panels.

[cg,M,cg_h] = protomass_opt;
aero_In = [L,Diameter,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters
listed in Trike.m
Out = aero(aero_In);
CdA = real(Out(1)); % Used as initial value.

% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = zeros(1,length(Pbatt));
grad = s;
t0 = 0;


% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.
%Sol_array = 0.14*1;

% Solar parameters:
%theta_init  = 70/180*pi; % Degrees from normal.
```

```
%theta_dot = theta_init/(3*3600);% Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
for i = 1:1:length(Pbatt)*dt,
    %P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;
    P_sol(i) = 0;
       % Include stopping to change cyclists and a 2 minute pitstop - can
ignore.
%    Shift =Shift_time*3600;     % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);

end

% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
    Dist = maxdist;
    % Int system eqn.
    i = 1;
while (max(s)/1000 <= Dist) & (i < length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
         if jj < length(ddist),
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%               i = i
               end
        end
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consideer 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle.

            aero_In = [L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];
      % All parameters listed in Trike.m
            Out = aero(aero_In);
            CdA = real(Out(1));
            D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!

%        if Wind == 0,
%            Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
%            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
%            dydt = cycle_DE(t,y0,Para);
           %        elseif  Wind == 1,
           Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
```

```
%             if length(Wind) == 1,
                 y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(i-1)];
                  dydt = cycle_DE_wind(t,y0,Para);
                  %           else
%                  y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(t/dt)];
                 %            end
             % dydt = cycle_DE_wind(t,y0,Para);
             %      end



         dEbatt = dydt(1);
         ds = dydt(2);
         dv = dydt(3);
         % System variables update:
         Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
         s(i) = s(i-1) + ds*dt;
         V = v(i-1) + dv*dt;
         if V < 0, V = 0; end
         v(i) = V;

    % Could add an interval constraint on the batt energy here. - this
should work, as better perform results with moderate power usage.
   if Ebatt(t) > Capacity,
       Pbatt(t+1) = 0;
   end


    end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(1:t);
if pplot == 1,
    figure;
    plot(v_out);
end
% Setup output values - this is the value to be minimised.
%Vel_ave = 50 - mean(v_out); % Hence the smallet, the higher the average,
the quicker the cycle, the shorter the time, etc.....
% Above for fmincon.m, which does not solve the problem.

Vel_ave = mean(v_out);

% Cheat a bit in calculating final time, to overcome discretisation:
t = Dist*1000/Vel_ave;


%disp(sprintf('Mean velocity: %3d', Vel_ave));
%disp(sprintf('Time of stage: %3d', t));
```

## D.3.14 protomass_opt.m

```
% Function, called by stab02.m
% And route simulations.
% Cleaned up for Solartrike3. 31-10-99.
% Modified for a single front wheel and two rear wheels.
% Initially guestimated.
% Using prototype frame dim.


% Estimate of mass and C.G.
% All masses is kg, position relative to front axle, positive toward rear
axle.

function [CG,T_mass,CG_h] = protomass_opt();

% Load cycle parameters:
% Physical parameters:
track = 1.0; % Track in metres. Aero analysis assumes that this is constant.
wheelbase = 2;
L = 3; % Vehicle length.
Diameter = 0.66;    % Vehicle diameter.
x = 0.6;      % Transition point on body.
A_wet = 5.01;% Wetted area of body from CAD.
L_wing = 0.56;      % Chord of wing.
A_wet_wing = 1.8*track*L_wing + 2*.96;
t_c = 0.21;         % Thickness ratio of wing.
% Rolling parameters:
D_wheel = 0.707;    % Wheel diameter m.
Tor_b = 0.005;              % Bearing torque drag. Nm.
% Tyres, ass: Michelin
C0 = 0.005;         % Rolling resistance coeff. Check in Storey et al.
C1 = 0.005;
% Electrical parameters:
nu_motor = 0.9; % Later incorporate the whole curve.
nu_panels = 0.14; % Ditto. Also for temp.
nu_controller = 0.95;
% Panel parameters:
A_panels = 4*0.21;  % Reusing n UCT panels.

% Components masses, positions(in portion of wheelbase) and heights (in
portion of wheel radius).
WR = D_wheel/2;
% Frame
m_frame = 10.8;
p_frame = wheelbase;       % Est.
h_frame = D_wheel;

% Battery    Swapped to NiMH/SLA
m_batt = 24;%13.7;
p_batt = 1.1*wheelbase;
h_batt = h_frame - 0.05;
```

```
% Motor
m_mot = 3;
p_mot = 0;
h_mot = WR + 0.05;

% Controller
m_cont = 1;
p_cont = 0.05;
h_cont = h_frame;


% Front Wheel
m_fw = 0.5;
p_fw = 0*wheelbase;
h_fw = WR;

% Rear Wheels
m_rw = 2*0.5;
p_rw = wheelbase;
h_rw = WR;

% Cyclist
m_cyc = 80;  % Rael: 80, Gordon: 60.
p_cyc = 1.5; % Est.
h_cyc = h_frame-0.2;

% Body
m_body = 6;  % Est.
p_body = wheelbase*0.5;    % Est.
h_body = h_frame+0.1;

% Ballast !?!!
m_ball = 0;
p_ball = 0;
h_ball = 0;

% Total mass
T_mass = m_body + m_frame + m_cyc + m_batt + m_mot + m_cont + m_fw + m_rw +
m_ball;

% C.G.
CG =
(m_body*p_body+m_frame*p_frame+m_cyc*p_cyc+m_batt*p_batt+m_mot*p_mot+m_cont*
p_cont+m_fw*p_fw+m_rw*p_rw+m_ball*p_ball)./T_mass;

% For height of CG:
CG_h =
(m_body*h_body+m_frame*h_frame+m_cyc*h_cyc+m_batt*h_batt+m_mot*h_mot+m_cont*
h_cont+m_fw*h_fw+m_rw*h_rw+m_ball*h_ball)./T_mass;

%out = [CG,T_mass];
```

## D.3.15 cyclist_opt.m

```
% Cyclist function.
% maps out an exponential curve of the cyclist's steady state output.
% Optimised for test run - can ignore pitstops.

function out = cyclist_opt(t);

%t = in(1);
%ST = in(2);
%Shift = ST*3600;
rider = 2;      % Currently planning to have Andrew cycling.



   if (rider == 1),    % Peter
        out = 150*exp(-0.5e-3.*(t-180)) + 250; % Model valid for Peter.
   elseif (rider == 2),    % Andrew
        out = 150*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Andrew.
   elseif (rider == 3),    % Mark
        out = 100*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Mark or
Roelie.
    elseif (rider == 6),  % Gordon
        out = 100*exp(-0.5e-3.*(t-180)) + 100; % Model valid for Gordon.
    end
```

## D.3.16 cycle_PD2_grad.m

```
% Calculates the columnar gradient vector, given a power trajectory.
% Called by gradproj.m



function [grad] =
cycle_PD2_grad(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,ft);
% For PD2 algor.
    % ->> Must return the gradient!

%grad = zeros(1,length(Pbatt);
grad = zeros(length(Pbatt),1);

Fk = ft;
%rFk = round(ft);
%Pbatt = Pbatt(1:rFk + 20);
gk = grad;%zeros(rFk + 20,1);
for i = 1:1:length(Pbatt),%rFk + 20,
    Pbatt_i = Pbatt;
    Pbatt_i(i) = Pbatt_i(i)*1.1;
    Fki = cycle_PD2(Pbatt_i,v_wind,ddist,ggrad,tf_limit,pplot,Capacity);
    if Pbatt_i(i) == 0,
```

```
        gk(i) = 0;
    else
        gk(i) = (Fki - Fk)/(Pbatt_i(i) - Pbatt(i));
    end
%    disp(sprintf('Progress: %3d',i));
end

grad = gk;
```

## D.3.17 NM_cycle_veri.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Used to verify the results given by tester_NM.m and nelderPP.m


% Calls: cycle_DE_wind.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: tester_NM.m

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
function [Vel_ave,t,s_out,v_out,Ebatt_out] =
NM_cycle_veri(Pbatt,ddist,ggrad,Wind,Capacity,display);



% Physical parameters:
g = 9.81;
dt = 1;

% Load cycle parameters:
% Physical parameters:

M = 75 + 70;    % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.


% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = zeros(1,length(Pbatt));
```

```
grad = s;
t0 = 0;


% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;    % Improve - use correct eqn.
%Sol_array = 0.14*1;

% Solar parameters:
%theta_init  = 70/180*pi; % Degrees from normal.
%theta_dot = theta_init/(3*3600);% Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;    % hours.
for i = 1:1:length(Pbatt)*dt,
    %P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;
    P_sol(i) = 0;
       % Include stopping to change cyclists and a 2 minute pitstop - can
ignore.
%    Shift =Shift_time*3600;     % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);

end

% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
%    Dist = maxdist;
Dist = maxdist-mindist;
    % Int system eqn.
    i = 1;
while (max(s)/1000 <= Dist) & (i < length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
         if jj < length(ddist),
                 while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                    jj = jj + 1;
%                 i = i
                end
        end
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consideer 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle, This
is to make provision for the variation of Cd with
```

```
%             aero_In =
[L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];     % All parameters
listed in Trike.m
%             Out = aero(aero_In);
%             CdA = real(Out(1));
              D = 0.5*1.2*CdA;
          % Solve system DEs:
          % simple direct integration approach:   NO SOLAR POWER!
%          if Wind == 0,
%              Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
%              y0 = [Ebatt(i-1),s(i-1),v(i-1)];
%              dydt = cycle_DE(t,y0,Para);
%          else
              Para = [Pbatt(i-1),P_cyc(i-1),D,Resist,G,M];
              if length(Wind) == 1,
                  y0 = [Ebatt(i-1),s(i-1),v(i-1),Wind];   % Mean wind velocity
given.
                  dydt = cycle_DE_wind_opt(t,y0,Para);
              else
                  y0 = [Ebatt(i-1),s(i-1),v(i-1),Wind(i-1)];
                  %           end
              dydt = cycle_DE_wind_opt(t,y0,Para);
              end

          dEbatt = dydt(1);
          ds = dydt(2);
          dv = dydt(3);
          % System variables update:
          Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
          s(i) = s(i-1) + ds*dt;
          V = v(i-1) + dv*dt;
          if V < 0, V = 0; end
          v(i) = V;

      % Could add an interval constraint on the batt energy here.
       if Ebatt(t) > Capacity,
         Pbatt(t+1) = 0;
       end


      end

% end of main iterative loop.

tf = t;
% Setup output values.
%v_out = v(2:length(v));
v_out = v(1:tf);
%s_out = s(2:length(s));
s_out = s(1:tf);
%Ebatt_out = Ebatt(2:length(Ebatt));
Ebatt_out = Ebatt(1:tf);
Vel_ave = mean(v_out);
```

```
t = Dist*1000/Vel_ave;

if display == 1,

% Plot results:

   figure;
    plot(v_out);
    title('Velocity profile for Nelder-Mead solution.');
        text(500,2,'Test Route');

    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

%    figure;
%    plot(s,v);
%    plot(s,v,ddist*1000,ele/5,ddist*1000,ggrad*5)
%    title('Velocity vs distance plot.');
 %   legend('vel','Ele','Gradx5');


    % Insert IC into vel:
%    PB = zeros(1,t+1);
    PB = Pbatt(1:tf);%Pbatt(1:t/dt);

    figure;
    plot(PB);
    title('Battery Power Trajectory for Nelder-Mead solution.');
        text(500,2,'Test Route');
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));


disp(sprintf('Mean velocity %0.5g',Vel_ave));

% Calculate residual energy/ energy imbalance and compare to power integral:

Imbalance = max(Ebatt) - Capacity;
disp(sprintf('Imbalance in battery energy, (-ve is un-utilised energy)
%0.5g',Imbalance));
disp(sprintf('In percentage terms %0.3g%%',Imbalance/Capacity*100));
Pint = sum(PB)*dt;
disp(sprintf('Power integral %0.5g',Pint));
PImbalance = Pint - Capacity;
disp(sprintf('Imbalance in battery energy, according to power integral
%0.5g',PImbalance));
disp(sprintf('In percentage terms %0.3g%%',PImbalance/Capacity*100));
end
```

### D.3.18 cycle_DE_wind_opt.m

```
% cycle_DE.m This file contains the system DEs for the cycle.
% for use in ODEsIver in calcengineB2.m
% And in own solver, Mine.m
% Also called by: Steady.m, TSPC_cycle.m, TSPC_cycle_cP.m
% Removed solar power.

function dydt = cycle_DE_wind_opt(t,y,p);  %p1,p2,p3,p4,p5,p6,p7);

E_batt = y(1);
s = y(2);
v = y(3);
v_wind = y(4);
P_batt = p(1);
%P_sol = p(2);
P_cyc = p(2);
D = p(3);
R = p(4);
G = p(5);
M = p(6);

v_apparent = v + v_wind;

% Prevent v and v_apparent from hitting zero. Prevents singularities in the
y_3, acceleration term.
if v_apparent <= 0,
    v_apparent = 0.001;
end
if v <= 0,
    v = 0.001;
end



y_1 = P_batt;

y_2 = v;

%y_3 = 1/(M*v)*(P_batt + P_sol + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));
y_3 = 1/(M*v)*(P_batt + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));

% Limit acceleration to +- g.
if y_3 > 9.81,
    y_3 = 9.81;
elseif y_3 < -9.81,
    y_3 = -9.81;
end

dydt = [y_1;y_2;y_3];
```

## D.3.19 TSPC_solver_NM.m

```
% This is a wrapper for calling the Nelder-Mead solver for the solar cycle
problem.


% Optimised for test run.
% No solar power, can ignore pitstops.
% Select number of segments on lines 30-33.

% Calls: nelderPP.m, NM_cycle_veri.m
% Called by: TSPC_SM_opt.m

function [s_out,v_out,Ebatt_c,t,Pbatt] =
TSPC_solver_NM(sl,route,dt,t0,v0,s0,Capacity,Wind,horizon);   % hprop is th
proportion of the horizon length comapred to the stage length, used from
reducing the Capacity used in horizon.

% Function control:
interim = 0;     % Show interim results if set.
Psingle = 0;     % Use a single power level instead of a spline. - does not
work.

%tf_limit = 1500;     % Initial guess, should be adequate.
ddist = route(:,1);
ggrad = route(:,2);

tf_limit = round(horizon*1000/11);  % Time to cover distance at 11m/s -
average spee of UCT cycle.

%load winddata;
%v_wind = v_wind(1:tf_limit);

Pbatt_max = 500;

% Step 1:
% Estimate initial trajectory:
% Setup Bspline.
% Control point array:
M = 4;%10;%4;% Number of ctrl points, 3 is min. range for TSPC

%M = 40;%40;%100;%400;     % Number of ctrl points, 3 is min. range for MTP
solution.

P = ones(M+1,1);
P2 = P;
% 1st row is vel. 2nd row is dist.

% Curve of m-2 segments Q3,...,Qm or m+1 ctrl pts P0,P1,...,Pm  m>3.
% Hence Q3 is single segment curve, Q4 is second segment of 2 segment curve.
routelength = max(ddist)*1000;
%tf_limit = round(routelength/11);     % Worst-case, no power, and slow....
u_scale = tf_limit./(M+1);
% Set initial values to P:
for k = 1:1:(M+1),
```

```
   P(k) = 4;%100;
   P2(k) = u_scale*k;
end
% Initial Power trajectories:
% Offset set in cyclePD.m
for j = 1:1:M+2,
    P0(:,j) = 15*rand(size(P));    % Works for TSPC short route.
 %   P0(:,j) = 10*rand(size(P));    % Works for TSPC test route.
%    P0(:,j) = 5*rand(size(P));    % Works MPT
%P0(:,j) = [j:1:M+1]' - (M+1)/2;
%    P0(:,j) = j*ones(size(P));      % Does not work.
%    P0(:,j) = 5*ones(size(P));      % does not work!!!!!!!!!!!!!!!
%    P0(:,j) = 5*rand(size(P));      % Attempt to get "flatter" power
curves.No effect.

end

% Single power point:
if Psingle == 1,
    P = [0;Pbatt_max/2];
    P0 = P;
end

% For M = 3:

%P0 = [ 1.1 0 -1 -2 -3;
%       2 0.5 0 -1 -2;
%       3 2 1 6 -1;
%       4 3.1 2 1 4.6];


% Set NM parameters:
tol = 0.001;
maxit = 100;
budget = 50*length(P0);
% Above are default.
maxit = 50;
budget = 20*length(P0);

% Call NM:
[P,lhist,histout,simpdata] =
nelderPP(P0,tol,maxit,budget,Wind,ddist,ggrad,M,u_scale,P2,tf_limit,0,Capaci
ty);



% Flesh out spline.
Pbatt = zeros(1,tf_limit);
Pdisp = P(:,1); % First vertex is best result in simplex.
no_seg = M-2;
if Psingle == 0,
for j = 1:1:tf_limit,
    if no_seg > 1,
```

```
        u = (j - u_scale*no_seg)/100;  % index;
    else
        u = j/100;
    end
    for k = 0:1:3,
          h = Pdisp(no_seg+k,:);
     PPP(k+1,:) = h';
    end
    UU = [u^3 u^2 u 1];
    B = [ -1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];
    P_u = UU/6*B*PPP;
    if P_u(1,1) > 0,
        Pbatt(j) = P_u(1,1);
    else
        Pbatt(j) = 0;
    end
    if P_u(1,1)  > Pbatt_max,
        Pbatt(j) = Pbatt_max;
    end


end
else
    Pbatt = ones(1,tf_limit)*P;
end

% Verify NM result and compute output traj.:

[Vel_mean,t,s_out,v_out,Ebatt_c] =
NM_cycle_veri(Pbatt,ddist,ggrad,Wind,Capacity,0);

% Initialize recorded values: including all state variables
% State variables:
%Ebatt = zeros(1,length(Pbatt));
%s_out = zeros(1,length(Pbatt));
%v = v0*ones(1,length(Pbatt));

%%for i = 1:1:length(s_c);
%    s_out(i) = s0 + s_c(i);
%end
```

## D.3.20 nelderPP.m

```
%function
[x,lhist,histout,simpdata]=nelderPP(x0,f,tol,maxit,budget,varargin)
function
[x,lhist,histout,simpdata]=nelderPP(x0,tol,maxit,budget,v_wind,ddist,ggrad,m
m,uscale,P2,tf_limit,pplot,Capacity)
%
% Nelder-Mead optimizer, No tie-breaking rule other than MATLAB's sort
%
```

```
% C. T. Kelley, December 12, 1996
%
% Extended to include parameter passing.    GAW 9-7-02
% Cop-out: insert objective function directly.
%
% This code comes with no guarantee or warranty of any kind.
%
% function [x,lhist,histout,simpdata] =
nelder(x0,f,tol,maxit,budget,varargin)
%
% inputs:
%      vertices of initial simplex = x0 (n x n+1 matrix)
%          The code will order the vertices for you and no benefit is
%          accrued if you do it yourself.
%
%       objective function = f
%
%       termination tolerance = tol
%       maximum number of iterations = maxit (default = 100)
%           As of today, dist = | best value - worst value | < tol
%           or when maxit iterations have been taken
%       budget = max f evals (default=50*number of variables)
%                The iteration will terminate after the iteration that
%                exhausts the budget
%
%
% outputs:
%      final simplex = x (n x n+1) matrix
%
%       number of iterations before termination = itout (optional)
%       iteration histor = histout itout x 5
%         histout = iteration history, updated after each nonlinear
iteration
%                 = lhist x 5 array, the rows are
%                   [fcount, fval, norm(grad), dist, diam]
%                   fcount = cumulative function evals
%                   fval = current best function value
%                   norm(grad) = current simplex grad norm
%                   dist = difference between worst and best values
%                   diam = max oriented length
%       simpdata = data for simplex gradient restart
%              = [norm(grad), cond(v), bar f]
%
% initialize counters
%
lhist=0; fcount=0;
%
% set debug=1 to print out iteration stats
%
debug=0;
%
% Set the N-M parameters
%
```

```
rho=1; chi=2; gamma=.5; sigma=.5;
dsize=size(x0); n=dsize(1);

% set the paramters for stagnation detection/fixup
% setting oshrink=0 gives vanilla Nelder-Mead
%
oshrink=1; restartmax=3; restarts=0;
%
%
% Order the vertices for the first time
%
x=x0; [n,m]=size(x); histout=zeros(maxit*3,5); simpdata=zeros(maxit,3);
itout=0; orth=0;
xtmp=zeros(n,n+1); z=zeros(n,n); delf=zeros(n,1);
for j=1:n+1;
%    fv(j)=feval(f,x(:,j));

fv(j)=cycle_NM(x(:,j),v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacit
y);
end;
fcount=fcount+n+1;

[fs,is]=sort(fv); xtmp=x(:,is); x=xtmp; fv=fs;
itc=0; dist=fv(n+1)-fv(1);
diam=zeros(n,1);
for j=2:n+1
   v(:,j-1)=-x(:,1)+x(:,j);
   delf(j-1)=fv(j)-fv(1);
   diam(j-1)=norm(v(:,j-1));
end
sgrad=v'\delf;
alpha=1.d-4*max(diam)/norm(sgrad);
lhist=lhist+1;
histout(lhist,:)=[fcount, fv(1), norm(sgrad,inf), 0, max(diam)];
%
% main N-M loop
%
while(itc < maxit & dist > tol & restarts < restartmax & fcount <= budget)
    fbc=sum(fv)/(n+1);
    xbc=sum(x')'/(n+1);
    sgrad=v'\delf;
    simpdata(itc+1,1)=norm(sgrad);
    simpdata(itc+1,2)=cond(v);
    simpdata(itc+1,3)=fbc;
    if(det(v) == 0)
        disp('simplex collapse')
        break
    end
    happy=0; itc=itc+1; itout=itc;
%
% reflect
%
    y=x(:,1:n);
```

```
    xbart = sum(y')/n;  % centriod of better vertices
    xbar=xbart';
    xr=(1 + rho)*xbar - rho*x(:,n+1);
%    fr=feval(f,xr);
    fr=cycle_NM(xr,v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacity);
    fcount=fcount+1;
    if(fr >= fv(1) & fr < fv(n)) happy = 1; xn=xr; fn=fr; end;
%    if(happy==1) disp(' reflect '); end
%
% expand
%
    if(happy == 0 & fr < fv(1))
        xe = (1 + rho*chi)*xbar - rho*chi*x(:,n+1);
%        fe=feval(f,xe);

fe=cycle_NM(xe,v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacity);
        fcount=fcount+1;
        if(fe < fr) xn=xe;  fn=fe; happy=1; end
        if(fe >=fr) xn=xr;  fn=fr; happy=1; end
%        if(happy==1) disp(' expand '); end
    end
%
% contract
%
   if(happy == 0 & fr >= fv(n) & fr < fv(n+1))
%
% outside contraction
%
       xc=(1 + rho*gamma)*xbar - rho*gamma*x(:,n+1);
%       fc=feval(f,xc);

fc=cycle_NM(xc,v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacity);
       fcount=fcount+1;
       if(fc <= fr) xn=xc; fn=fc; happy=1; end;
%       if(happy==1) disp(' outside '); end;
   end
%
% inside contraction
%
   if(happy == 0 & fr >= fv(n+1))
       xc=(1 - gamma)*xbar+gamma*x(:,n+1);
%       fc=feval(f,xc);

fc=cycle_NM(xc,v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacity);
       fcount=fcount+1;
       if(fc < fv(n+1)) happy=1; xn=xc; fn=fc; end;
%   if(happy==1) disp(' inside '); end;
   end
%
%  test for sufficient decrease,
%  do an oriented shrink if necessary
%
   if(happy==1 & oshrink==1)
```

```
        xt=x; xt(:,n+1)=xn; ft=fv; ft(n+1)=fn;
%        xt=x; xt(:,n+1)=xn; ft=fv; ft(n+1)=feval(f,xn); fcount=fcount+1;
        fbt=sum(ft)/(n+1); delfb=fbt-fbc; armtst=alpha*norm(sgrad)^2;
        if(delfb > -armtst/n)
            restarts=restarts+1;
            orth=1; diams=min(diam);
            sx=.5+sign(sgrad); sx=sign(sx);
if debug==1
            [itc, delfb, armtst]
end
            happy=0;
            for j=2:n+1; x(:,j)=x(:,1);
            x(j-1,j)=x(j-1,j)-diams*sx(j-1); end;
        end
    end
%
%  if you have accepted a new point, nuke the old point and
%  resort
%
    if(happy==1)
        x(:,n+1)=xn; fv(n+1)=fn;
%        x(:,n+1)=xn; fv(n+1)=feval(f,xn); fcount=fcount+1;
        [fs,is]=sort(fv); xtmp=x(:,is); x=xtmp; fv=fs;
    end
%
% You're in trouble now! Shrink or restart.
%
    if(restarts >= restartmax) disp(' stagnation in Nelder-Mead'); end;
    if(happy == 0 & restarts < restartmax)
        if(orth ~=1) disp(' shrink '); end;
        if(orth ==1)
        if debug == 1 disp(' restart '); end
        orth=0; end;
        for j=2:n+1;
            x(:,j)=x(:,1)+sigma*(x(:,j)-x(:,1));
%            fv(j)=feval(f,x(:,j));

fv(j)=cycle_NM(x(:,j),v_wind,ddist,ggrad,mm,uscale,P2,tf_limit,pplot,Capacit
y);
        end
        fcount=fcount+n;
        [fs,is]=sort(fv); xtmp=x(:,is); x=xtmp; fv=fs;
    end
%
%  compute the diameter of the new simplex and the iteration data
%
    for j=2:n+1
        v(:,j-1)=-x(:,1)+x(:,j);
        delf(j-1)=fv(j)-fv(1);
        diam(j-1)=norm(v(:,j-1));
    end
    dist=fv(n+1)-fv(1);
    lhist=lhist+1;
```

```
   sgrad=v'\delf;
   histout(lhist,:)=[fcount, fv(1), norm(sgrad,inf), dist, max(diam)];
end
```

## D.3.21 cycle_NM.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Derived from cycle_PD.m for use in NM and TSPC with NM.

% Input: Pbatt only.
% Output: Vel_ave only!! - scalar.
% Power and battery energy constrains handled in calling function.

% Calls: cycle_PD.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: ????

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
%function [s_out,v_out,Ebatt_out] =
TSPC_cycle_opt(route,dt,t0,v0,s0,Pbatt,Wind);
%function [Vel_ave,t] =
cycle_PD(PP,v_wind,ddist,ggrad,m,uscale,P2,tf_limit,pplot);   % For PD
algor.
function t =
cycle_NM(PP,v_wind,ddist,ggrad,m,uscale,P2,tf_limit,pplot,Capacity);
% For NM algor.


%Capacity = 750*3600/4;  % Est of NiCd pack.
Pbatt_max = 500;
%Capacity = 750*3600/16;  % Est of NiCd pack.
    dt = 1;

if length(PP) > 2,
% Reconstruct spline from control points:
P = zeros(2,length(PP));
P(1,:) = PP';
% Add offset to work around lb problem.
%P = P - 0.5;
P = P - 5;
P(2,:) = P2';
% Create fine power vector from spline:
no_seg = m-2;
%PBatt = zeros(1,length(tf_limit));
Pbatt = zeros(1,tf_limit);
%index = 1;
```

```
%for j = 1:1:length(tf_limit/dt),
for j = 1:dt:tf_limit,
    if no_seg > 1,
        u = (j - uscale*no_seg)/100;    % index;
    else
        u = j/100;
    end
    for k = 0:1:3,
        h = P(:,no_seg+k);
        PPP(k+1,:) = h';
    end
    UU = [u^3 u^2 u 1];
    B = [ -1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];
    P_u = UU/6*B*PPP;

    if P_u(1,1) > Pbatt_max,
        Pbatt(j) = Pbatt_max;
    elseif P_u(1,1)  > 0,
        Pbatt(j) = P_u(1,1);
    else
        Pbatt(j) = 0;
    end

end
if pplot == 1,
    figure;
    plot(Pbatt);
end
%pause;
else
    dP = PP(2) - PP(1);
    dT = tf_limit;
    dPdt = dP/dT;
    Pbatt(1) = PP(1);
    for i = 2:1:tf_limit,
        Pbatt(i) = PP(1) + i*dPdt;
    end
%
end

%load winddata;
% v_wind = Wind(t);

% Physical parameters:
g = 9.81;
M = 75 + 70;     % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
```

```
v = zeros(1,length(Pbatt));
grad = s;
t0 = 0;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.

for i = 1:1:length(Pbatt)*dt,
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);
end

% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
    Dist = maxdist-mindist;
    % Int system eqn.
    i = 1;
while (max(s)/1000 <= Dist) & (i < length(Pbatt)-1),
        i = i + 1;
%            if i >= length(Pbatt),
%                disp('sdfg');
%           end
         % For grad...
         jj = 1;
         if jj < length(ddist),
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%               i = i
               end
        end
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consider 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % cycle parameters
            D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!
            Para = [Pbatt(i-1),P_cyc(i-1),D,Resist,G,M];
                y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind];
            dydt = cycle_DE_wind_opt(t,y0,Para);

        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dydt(1)*dt;
        s(i) = s(i-1) + dydt(2)*dt;
        V = v(i-1) + dydt(3)*dt;
        if V < 0, V = 0; end
        v(i) = V;

    % Could add an interval constraint on the batt energy here. - this
should work, as better perform results with moderate power usage.
```

```
    if (Ebatt(t) > Capacity),
        Pbatt(t+1) = 0;
    end


     end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(1:t);

Vel_ave = mean(v_out);

% Cheat a bit in calculating final time, to overcome discretisation:
t = Dist*1000/Vel_ave;
```

## D.3.22 NM_cycle_veri.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Used to verify the results given by tester_NM.m and nelderPP.m


% Calls: cycle_DE_wind.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: tester_NM.m

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
function [Vel_ave,t,s_out,v_out,Ebatt_out] =
NM_cycle_veri(Pbatt,ddist,ggrad,Wind,Capacity,display);



% Physical parameters:
g = 9.81;
dt = 1;

% Load cycle parameters:
% Physical parameters:

M = 75 + 70;     % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.
```

```
% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = zeros(1,length(Pbatt));
grad = s;
t0 = 0;


% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.
%Sol_array = 0.14*1;

% Solar parameters:
%theta_init  = 70/180*pi; % Degrees from normal.
%theta_dot = theta_init/(3*3600);% Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
for i = 1:1:length(Pbatt)*dt,
    %P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;
    P_sol(i) = 0;
       % Include stopping to change cyclists and a 2 minute pitstop - can
ignore.
%    Shift =Shift_time*3600;     % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);

end

% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
%    Dist = maxdist;
Dist = maxdist-mindist;
    % Int system eqn.
    i = 1;
while (max(s)/1000 <= Dist) & (i < length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
         if jj < length(ddist),
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%               i = i
               end
        end
```

```
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consideer 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle, This
is to make provision for the variation of Cd with


%            aero_In =
[L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];     % All parameters
listed in Trike.m
%            Out = aero(aero_In);
%            CdA = real(Out(1));
           D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!
%        if Wind == 0,
%            Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
%            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
%            dydt = cycle_DE(t,y0,Para);
%        else
           Para = [Pbatt(i-1),P_cyc(i-1),D,Resist,G,M];
           if length(Wind) == 1,
               y0 = [Ebatt(i-1),s(i-1),v(i-1),Wind];   % Mean wind velocity
given.
               dydt = cycle_DE_wind_opt(t,y0,Para);
           else
               y0 = [Ebatt(i-1),s(i-1),v(i-1),Wind(i-1)];
               %           end
           dydt = cycle_DE_wind_opt(t,y0,Para);
           end

        dEbatt = dydt(1);
        ds = dydt(2);
        dv = dydt(3);
        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
        s(i) = s(i-1) + ds*dt;
        V = v(i-1) + dv*dt;
        if V < 0, V = 0; end
        v(i) = V;

    % Could add an interval constraint on the batt energy here.
     if Ebatt(t) > Capacity,
       Pbatt(t+1) = 0;
     end


    end

% end of main iterative loop.

tf = t;
% Setup output values.
```

```
%v_out = v(2:length(v));
v_out = v(1:tf);
%s_out = s(2:length(s));
s_out = s(1:tf);
%Ebatt_out = Ebatt(2:length(Ebatt));
Ebatt_out = Ebatt(1:tf);
Vel_ave = mean(v_out);
t = Dist*1000/Vel_ave;

if display == 1,

% Plot results:

   figure;
    plot(v_out);
    title('Velocity profile for Nelder-Mead solution.');
        text(500,2,'Test Route');

    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

%    figure;
%    plot(s,v);
%    plot(s,v,ddist*1000,ele/5,ddist*1000,ggrad*5)
%    title('Velocity vs distance plot.');
 %   legend('vel','Ele','Gradx5');


    % Insert IC into vel:
%    PB = zeros(1,t+1);
    PB = Pbatt(1:tf);%Pbatt(1:t/dt);

    figure;
    plot(PB);
    title('Battery Power Trajectory for Nelder-Mead solution.');
        text(500,2,'Test Route');
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));


disp(sprintf('Mean velocity %0.5g',Vel_ave));

% Calculate residual energy/ energy imbalance and compare to power integral:

Imbalance = max(Ebatt) - Capacity;
disp(sprintf('Imbalance in battery energy, (-ve is un-utilised energy)
%0.5g',Imbalance));
disp(sprintf('In percentage terms %0.3g%%',Imbalance/Capacity*100));
Pint = sum(PB)*dt;
disp(sprintf('Power integral %0.5g',Pint));
PImbalance = Pint - Capacity;
```

```
disp(sprintf('Imbalance in battery energy, according to power integral
%0.5g',PImbalance));
disp(sprintf('In percentage terms %0.3g%%',PImbalance/Capacity*100));
end
```

## D.3.23 cycle_DE_wind_opt.m

```
% cycle_DE.m This file contains the system DEs for the cycle.
% for use in ODEslver in calcengineB2.m
% And in own solver, Mine.m
% Also called by: Steady.m, TSPC_cycle.m, TSPC_cycle_cP.m
% Removed solar power.

function dydt = cycle_DE_wind_opt(t,y,p);  %p1,p2,p3,p4,p5,p6,p7);

E_batt = y(1);
s = y(2);
v = y(3);
v_wind = y(4);
P_batt = p(1);
%P_sol = p(2);
P_cyc = p(2);
D = p(3);
R = p(4);
G = p(5);
M = p(6);

v_apparent = v + v_wind;

% Prevent v and v_apparent from hitting zero. Prevents singularities in the
y_3, acceleration term.
if v_apparent <= 0,
    v_apparent = 0.001;
end
if v <= 0,
    v = 0.001;
end



y_1 = P_batt;

y_2 = v;

%y_3 = 1/(M*v)*(P_batt + P_sol + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));
y_3 = 1/(M*v)*(P_batt + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));

% Limit acceleration to +- g.
if y_3 > 9.81,
    y_3 = 9.81;
```

```
elseif y_3 < -9.81,
    y_3 = -9.81;
end

dydt = [y_1;y_2;y_3];
```

### D.3.24 tester_converg.m

```
% This file is used to initialise and test the system.
% It calls TSPC_SM_opt in a cyclic manner.
% This is a dummy scheduler-type approach.

% This is taken from the above to focus on the convergence of the solver,
and avoid he kickouts.

% TSPC_SM init code.
% This is expected to evolve into the final code for the laptop. Derived
from laptop_init.m

clear;
tic;
selectroute = 9;%9;
showplot = 1;
Wind = 2;   % Wind handling needs to change to use wind speed readings.

Capacity = 750*3600;    % Reduce capacity in proportion to stage length and
forecast horizon length.
% For short route:
%Capacity = Capacity/4;

dt = 1;             % time step in seconds - code does not operate on
anything other than dt = 1.
%Shift_time = 1.5; % hours.
valid_period = 100;

switch selectroute      % default to shortroute
case 1
%    load fielddata;
    load testroutedata;
    ddist = dist;
    ggrad = grad;
    Capacity = Capacity/4;
case 9
%    load fielddata;
    load testroutedata2;
    ddist = dist1;
    ggrad = grad1;
       Capacity = Capacity/4;
%    tspan_init = 10000;   % Starting traj time span in seconds.
otherwise
    load shortroute;
    ddist = srd;
    ggrad = srg;
    Capacity = Capacity/4;
```

```
  %  tspan_init = 4000;   % Starting traj time span in seconds.
%    LL = 1;

end

% Initial power traj to get the cycle rolling.
tf_limit = 11000;   % 20kph for 56 km.
tspan_init = 200;   % Starting traj time span in seconds.
Pbatt_max = 500;    % Watts.
%Pbatt = random('norm',450,150,tf_limit/dt,1); % Use a normal distibution
about 15, with a spread of 1? Plot looks acceptable.
Pbatt = 0.5*Pbatt_max*ones(tspan_init,1);  % half power initial traj.
P0 = 0.5*Pbatt_max*ones(tf_limit,1);
route = zeros(length(ddist),2);
for i = 1:1:length(ddist),
    route(i,1) = ddist(i);
    route(i,2) = ggrad(i);
end
if  Wind == 0,
    v_wind = zeros(tf_limit,1);
elseif Wind == 1,
    load winddata;
elseif Wind == 2,
    v_wind = wind(tf_limit);
    save windata2 v_wind;
%    Wind = v_wind;
elseif Wind == 3,   % Measure wind...
    % ???
end
%    [s_c,v_c,Ebatt_c] =
TSPC_cycle(route,dt,Shift_time,0,0,0,Pbatt,Capacity,Wind);
    [s_c,v_c,Ebatt_c] = TSPC_cycle_opt(route,dt,0,0,0,Pbatt,Wind);

    % Determine TSPC start time:
    t = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t = t + dt; end
    end
% Update time series:
s = s_c;
v = v_c;
Ebatt = Ebatt_c;
%P = Pbatt;

% Start looping the rest of the way.
Dist = max(ddist);
%i = 2;
% Main loop:

while max(s)/1000 <= Dist, %for k = 1:1:Dist,

meanwind = mean(v_wind(t-valid_period:t));    % Calculate meanwind over most
recent validation period.
```

```
%[P,route_rem] =
TSPC_SM_opt(Capacity,ggrad,ddist,tspan_init,tf_limit,t,s,v,Ebatt,Pbatt,meanw
ind);
%plot(v)
%pause
[P,route_rem] =
TSPC_SM_opt(Capacity,ggrad,ddist,tspan_init,tf_limit,t,s,v,Ebatt,P0,meanwind
);

%disp('tic');

% Update time series - from measurements:
% This is the input point for the cycles measurement data.
% Truncate power traj to limit time of sim.:
%plot(P);
%pause;
%P = P(1:20);   % Too short for NM solver, never mind the best.
if length(P) > valid_period,
    P = P(1:valid_period);
end

[s_c,v_c,Ebatt_c] = TSPC_cycle_opt(route_rem,dt,t,v(t),s(t),P,Wind);
%[s_c,v_c,Ebatt_c] = TSPC_cycle_opt(route_rem,dt,t-1,v(t-1),s(t-1),P,Wind);

% Determine final time:
    t_c = dt;
    for i = 2:1:length(s_c),
        if s_c(i) > 0, t_c = t_c + dt; end
    end

% Use wind model.
if Wind >= 1,
    Wind = 1;
else
    Wind = 0;
end

 t0 = t;
 EE = Ebatt(t);
 for i = 1:1:t_c,
%     t = t + dt:dt:t_c,
    ti = t0 + i;

     Ebatt(ti) = Ebatt_c(i) + EE;
     s(ti) = s_c(i);
     v(ti) = v_c(i);
%     s(ti-1) = s_c(i);
%     v(ti-1) = v_c(i);
Pbatt(ti) = P(i);

 end
t = t0 +t_c;

%            for ii = t/dt:1:(length(P) + t/dt - 1),
```

```
%                Pbatt(ii) = P(ii - t/dt + 1);
%            end


end

% End of main loop - include incremental saves.

 % Add a plotting section.

if showplot == 1,
% Plot velocity and power.


    % Determine final time:
    t = dt;
    for i = 2:1:length(s),
        if s(i) > 0, t = t + dt; end
    end
    % Insert IC into vel:
    vel = zeros(1,t+1);
    vel(2:t+1) = v(1:t);

    figure;
    plot(vel);
    title('Velocity profile for TSPC solution.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Velocity [m/s]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));

%    figure;
%    plot(s,v);
%    plot(s,v,dist*1000,ele/5,dist*1000,grad*5)
```

```
%    title('Velocity vs distance plot.');
%    legend('vel','Ele','Gradx5');


    % Insert IC into vel:
%    PB = zeros(1,t+1);
    PB = Pbatt(1:t/dt);

    figure;
    plot(PB);
    title('Battery Power Trajectory for TSPC solution.');
    switch selectroute,
    case 1
        text(500,2,'Day1');
    case 2
        text(500,2,'Day2');
    case 3
        text(500,2,'Day3');
    case 4
        text(500,2,'Day4');
    case 5
        text(500,2,'Day5');
    case 6
        text(500,2,'Day6');
    case 7
        text(500,2,'Day7');
    case 8
        text(500,2,'Short route');
    case 9
        text(500,2,'Field test');
    end
    xlabel('Time [s]');
    ylabel('Power [W]');
    text(t/2,2,sprintf('Final time is %0.5g s.',t));


disp(sprintf('Mean velocity %0.5g',mean(vel)));
disp(sprintf('Final time is %0.5g s.',t));
% Calculate residual energy/ energy imbalance and compare to power integral:

Imbalance = max(Ebatt) - Capacity;
disp(sprintf('Imbalance in battery energy, (-ve is un-utilised energy)
%0.5g',Imbalance));
disp(sprintf('In percentage terms %0.3g%%',Imbalance/Capacity*100));
Pint = sum(PB)*dt;
disp(sprintf('Power integral %0.5g',Pint));
PImbalance = Pint - Capacity;
disp(sprintf('Imbalance in battery energy, according to power integral
%0.5g',PImbalance));
disp(sprintf('In percentage terms %0.3g%%',PImbalance/Capacity*100));

% For mean wind velocity on route:
mean_v_wind = mean(v_wind(1:1:t));
```

```
disp(sprintf('Mean wind velocity over route: %0.5g',mean_v_wind));

end % of plotting
toc;
```

## D.3.25 TSPC_cycle_opt.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.

% Calls: cycle_DE.m, cyclist_opt.m, aero.m, protomass.m, prototype.m
% Called by: Steady.m, TSPC_SM_opt.m

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
function [s_out,v_out,Ebatt_out] =
TSPC_cycle_opt(route,dt,t0,v0,s0,Pbatt,Wind);

%if nargin == 7,
%    Wind = 0;
%end
%if t0 > 1500,
%    disp('halt');
%end
%Shift = Shift_time*3600;    % Can ignore pitstops.

ddist = route(:,1);
ggrad = route(:,2);

% Physical parameters:
g = 9.81;

% Load cycle parameters:
% Physical parameters:
track = 1.0; % Track in metres. Aero analysis assumes that this is constant.
wheelbase = 2;
L = 3; % Vehicle length.
Diameter = 0.66;    % Vehicle diameter.
x = 0.6;     % Transition point on body.
A_wet = 5.01;% Wetted area of body from CAD.
L_wing = 0.56;      % Chord of wing.
A_wet_wing = 1.8*track*L_wing + 2*.96;
t_c = 0.21;         % Thickness ratio of wing.
% Rolling parameters:
D_wheel = 0.707;    % Wheel diameter m.
Tor_b = 0.005;              % Bearing torque drag. Nm.
% Tyres, ass: Michelin
```

```
C0 = 0.005;         % Rolling resistance coeff. Check in Storey et al.
C1 = 0.005;
% Electrical parameters:
nu_motor = 0.9; % Later incorporate the whole curve.
nu_panels = 0.14; % Ditto. Also for temp.
nu_controller = 0.95;
% Panel parameters:
A_panels = 4*0.21;  % Reusing n UCT panels.

[cg,M,cg_h] = protomass_opt;
aero_In = [L,Diameter,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters
listed in Trike.m
Out = aero(aero_In);
CdA = real(Out(1)); % Used as initial value.

% Step 1:
% Estimate initial trajectory:

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
s(1) = s0;
v = zeros(1,length(Pbatt));
v(1) = v0;
grad = s;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.
Sol_array = 0.14*1;

% Solar parameters:
theta_init  = 70/180*pi;  % Degrees from normal.
theta_dot = theta_init/(3*3600); % Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
for i = 1:1:length(Pbatt)*dt,
    P_sol(i) = 0;%Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;

       % Include stopping to change cyclists and a 2 minute pitstop - can
ignore.
%    Shift =Shift_time*3600;     % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);

end
if Wind == 1,
%    v_wind = wind(length(Pbatt)*dt);
load winddata;
elseif Wind == 2,
```

```
    load windata2;
elseif length(Wind)>1,
    v_wind = Wind;
end



% Main iterative loop:
mindist = min(ddist);
maxdist = max(ddist);
      Dist = maxdist-max(s)/1000;%mindist;
    % Int system eqn.
    i = 1;
while ((max(s)-s0)/1000 <= Dist) & (i <= length(Pbatt)),
        i = i + 1;

         % For grad...
         jj = 1;
         if jj < length(ddist),
                 while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                    jj = jj + 1;
%                i = i
                end
        end
        G = M*g*sin(ggrad(jj)*pi/180);
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        v_apparent = v(i-1);    % The apparent wind seen by the cycle.

            aero_In = [L,Diameter,x,v_apparent,A_wet,A_wet_wing,L_wing,t_c];
      % All parameters listed in Trike.m
            Out = aero(aero_In);
            CdA = real(Out(1));
            D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!
        if Wind == 0,
            Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
            dydt = cycle_DE(t,y0,Para);
        elseif  (Wind == 1)|(Wind == 2),
            Para = [Pbatt(i-1),0,P_cyc(i-1),D,Resist,G,M];
%            if length(Wind) == 1,
               y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(i-1)];
                dydt = cycle_DE_wind(t,y0,Para);

            else
               y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(t/dt)];
               %            end
            dydt = cycle_DE_wind(t,y0,Para);
        end
```

```
        dEbatt = dydt(1);
        ds = dydt(2);
        dv = dydt(3);
        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
        s(i) = s(i-1) + ds*dt;
%        disp(i)
%        disp(s(i))


            V = v(i-1) + dv*dt;
            if V < 0, V = 0; end
            v(i) = V;


        t_end = t;
        v_tf = V;

    end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(2:length(v));
s_out = s(2:length(s));
Ebatt_out = Ebatt(2:length(Ebatt));

% Setup output values.
```

## D.3.26 TSPC_SM_opt.m

```
% Implementation code.
% This is the third attempt at writing the TSPC.
% This is a proper shooting method approach, based on the Secant method, see
FB p??.

% Basic steps:
% 1. Measure states - external.
% 2. Generate forecasts.
% 3. Generate optimal trajectories.
% 4. Validate trajectories.
% 5. Repeat.

% This code will need to be modified to run from instrumentation data.
% Also optimised for speed in laptop on test run - NO solar power!!
% Calls optimised functions: TSPC_solver_opt.m, TSPC_cycle_opt.m,
% Tuned for MEX-file compilation, as are TSPC_solver_opt.m, TSPC_cycle_opt.m
and dependant functions.
% Changed to Nelder-Mead solver.
% Changed to second simple solver.
```

```
% Initialisation functions moved to laptop_init.m, terminiation functions
moved to laptop_term.m
% Converted to function.
% Setup for wind measurements, not estimation of wind. -Oh?
% Select solver on lines 139 -160.

function [P,route_rem] =
TSPC_SM_opt(Capacity,ggrad,ddist,tspan_init,tf_limit,t,s_c,v_c,Ebatt_c,Pbatt
,meanwind);

dt = 1; % Retain as code not operable on anything else.

% Physical parameters:
g = 9.81;
M = 75 + 70;     % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.


tol = 0.001*Capacity;
Pbatt_max = 500;     % Watts.

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,tf_limit);
s = zeros(1,tf_limit);
v = 0.01*ones(1,tf_limit);
%for i = 1:dt:length(Pbatt),
%for i = t-length(Pbatt):dt:t,
for i = 1:1:length(s_c),
    Ebatt(i) = Ebatt_c(i);
    s(i) = s_c(i);
    v(i) = v_c(i);
end

%t = tspan_init;

%Dist = max(ddist);
i = 2;
% Main loop - moved externally:
%while max(s)/1000 <= Dist, %for k = 1:1:Dist,

% Step 1: Measurements - handled outside for states, wind
      % x: E,s,v
      % For grad... prediction.
         j = 1;
         while ddist(j) <= (s(i-1)./1000),
                   j = j + 1;
             end
        G = M*g*sin(ggrad(j)*pi/180);
        grad(i-1) = G;

        % Wind:
```

```
        % Cycle parameters:
       % aero_In = [L,Diameter,x,v(i-1),A_wet,A_wet_wing,L_wing,t_c];   %
All parameters listed in Trike.m
       % Out = aero(aero_In);
       % CdA = real(Out(1)); % actual value with no wind. Use in
measurements of wind uncertainty - later.


% Step 2: Forecasts:
       wind_inc = meanwind + meanwind*0.1;
       wind_dec = meanwind - 0.11*meanwind;
       wind_same = meanwind;

% Step 3: Solve traj.
       disp('Step 3')
        horizon = 2;    % <--------------- Forecast horizon setting!! in km.
---------------------->
%        horizon = 10;    % 2 is too small, gives null returns in solver. 5
seems ok, expect at end.
       % Find current position:
       j = 1;
    while ddist(j) <= (s(t/dt-1)./1000),
          j = j + 1;
       end
       % Find end point:
       k = j;
          while (ddist(k) <= (s(t/dt-1)./1000 + horizon)) & (k <
length(ddist)),
              k = k + 1;
          end

       % Setup forecast route:
        if k > j,
            route = zeros(k-j,2);
            if k == j + 1,
                route(1,1) = ddist(j);
                route(2,1) = ddist(k);
                route(1,2) = ggrad(j);
                route(2,2) = ggrad(k);
            else
               for ii = 1:1:k-j,
                route(ii,1) = ddist(ii+j);
                route(ii,2) = ggrad(ii+j);
               end
           end
           segment_dist = max(route(:,1)) - min(route(:,1));
       else
           route = zeros(2,2);
           route(2,1) = ddist(k);
           route(1,1) = s(t)/1000;
           route(2,2) = ggrad(k);
           route(1,2) = ggrad(j-1);
           segment_dist = max(route(:,1)) - min(route(:,1));
       end
```

```
       if segment_dist == 0,      % For the case when only the final value of
ddist is available.
           segment_dist = max(route(:,1)) - max(s)/1000;
       end
        segment_capacity = Capacity*segment_dist/max(ddist);
        horizon_capacity = segment_capacity;
%       disp('horz. dist:');
%       disp(segment_dist);
%       disp('capacity:');
%       disp(Capacity);
%       disp('seg. Cap.');
%       disp(segment_capacity);
%        pause;

       % Solve over forecast horizon:
%          if t > 460,
%        v(t-1)
%        t
%        plot(v)
%        pause;
% end
       disp('inc forecast.')
%         [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-1),horizon_capacity,wind_inc);
%         [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_inc);
%         [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_inc,horizo
n);
%         [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_inc,horiz
on);
        [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_inc,hori
zon,Pbatt(t/dt));
%        disp('hor. cap: ');
%        disp(horizon_capacity);
        %pause;  % No horizon_capacity!!
        disp('dec forecast.')
%         [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-1),horizon_capacity,wind_dec);
%         [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_dec);
%         [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_dec,horizo
n);%
%         [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_dec,horiz
on);
        [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_dec,hori
zon,Pbatt(t/dt));
       disp('static forecast.')
```

```
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-
1),horizon_capacity,wind_same);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_same);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_same,horiz
on);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_same,hori
zon);
        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_same,hor
izon,Pbatt(t/dt));



       % Setup remaining route:
       route_rem = zeros(length(ddist)-k,2);
       for ii = 1:1:(length(ddist) - k + 1),
           route_rem(ii,1) = ddist(ii + k - 1);
           route_rem(ii,2) = ggrad(ii + k - 1);
       end
       rem_dist = max(ddist) - route_rem(1,1);
       rem_prop = rem_dist/max(ddist);

% Step 4: Validation & Traj output
    disp('Step 4');
    % Wait for sufficient time to pass....

    % Setup recorded Vel from past validation period. -> need to include
pitstops.
       valid_period = 100;   % seconds
%       valid_period = 60;   % seconds
    im = length(v_inc);
    dm = length(v_dec);
    sm = length(v_same);
    small = min([im,sm,dm]);
    if small < valid_period,
        valid_period = min([im,sm,dm]);
    end

%          v_rec = v(t - valid_period:1:t-1);      % Recorded velocity for
last validation period.
          v_rec = v_c(t - valid_period:1:t-1);      % Recorded velocity for
last validation period.

          v_inc_rec = v_inc(1:1:valid_period);

          v_dec_rec = v_dec(1:1:valid_period);

          v_same_rec = v_same(1:1:valid_period);

          traj = LMSerror(v_rec,v_inc_rec,v_dec_rec,v_same_rec);
           if traj == 1,
```

```
                Ptraj = Pbatt_inc;
                disp(sprintf('wind increasing, t: %5d',t));
            elseif traj == 2,
                Ptraj = Pbatt_dec;
                disp(sprintf('wind decreasing, t: %5d',t));
            else traj == 3,         % Default.
                Ptraj = Pbatt_same;
                disp(sprintf('wind static, t: %5d',t));
            end
            disp(sprintf('distance: %5d',s(t)));

        %       for ii = t/dt:1:(length(Ptraj) + t/dt - 1),
        %           Pbatt(ii) = Ptraj(ii - t/dt + 1);
        %       end

% Step 5: Update cycle model and setup measurements -external.
% Step 5: Output power trajectory.
disp('Step 5');

P = Ptraj(1:1:valid_period);


%end
```

## D.3.27 TSPC_solver_opt.m

```
% This is a purposely derived solver for the solar cycle problem.
% Problem statement:
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% It starts by integrating the system eqns, from s(0) to s(tf) to determine
tf.
% It then checks the energy balance and adjusts the controls accordingly.
% This is repeated until the energy balance is within a certain tolerance.
% The influence egn are then backward integrated.
% This enables one to check the closeness of fit of the optimality
condition.

% Converted to Steepest descent. cf FB 357


% Optimised for test run.
% No sloar power, can ignore pitstops.

% Calls: cycle_DE.m, cyclist_opt.m, aero.m, protomass_opt.m, prototype.m
% Called by: TSPC_SM_opt.m

function [s,v,Ebatt,t,Pbatt] =
TSPC_solver_opt(sl,route,dt,t0,v0,s0,hCap,Wind);   % hprop is th proportion
of the horizon length comapred to the stage length, used from reducing the
Capacity used in horizon.
```

```
% Function control:
interim = 1;    % Show interim results if set.

%Shift = Shift_time*3600;

ddist = route(:,1);
ggrad = route(:,2);
% Physical parameters:
g = 9.81;

% Load cycle parameters: - look at inlining.
% Physical parameters:
track = 1.0; % Track in metres. Aero analysis assumes that this is constant.
wheelbase = 2;
L = 3; % Vehicle length.
Diameter = 0.66;    % Vehicle diameter.
x = 0.6;     % Transition point on body.
A_wet = 5.01;% Wetted area of body from CAD.
L_wing = 0.56;      % Chord of wing.
A_wet_wing = 1.8*track*L_wing + 2*.96;
t_c = 0.21;         % Thickness ratio of wing.
% Rolling parameters:
D_wheel = 0.707;    % Wheel diameter m.
Tor_b = 0.005;              % Bearing torque drag. Nm.
% Tyres, ass: Michelin
C0 = 0.005;         % Rolling resistance coeff. Check in Storey et al.
C1 = 0.005;
% Electrical parameters:
nu_motor = 0.9; % Later incorporate the whole curve.
nu_panels = 0.14; % Ditto. Also for temp.
nu_controller = 0.95;
% Panel parameters:
A_panels = 4*0.21;  % Reusing n UCT panels.

[cg,M,cg_h] = protomass_opt;
aero_In = [L,Diameter,x,0,A_wet,A_wet_wing,L_wing,t_c];    % All parameters
listed in Trike.m
Out = aero(aero_In);
CdA = real(Out(1));

% Step 1:
% Estimate initial trajectory:
if sl == 1,
    tf_limit = 11000;
    KO_val = 20;
elseif sl == 2,                 % Default case for test run....
%    tf_limit = 1500;
%    tf_limit = 2500;
%    tf_limit = 700;    Does not work - too short
    tf_limit = 1000;
    KO_val = 20;%7;             % Attempting to elim. spikes.
else
```

```
    tf_limit = 7000;
    KO_val = 15;
end
Pbatt_max = 500;    % Watts.
Pbatt = 0.5*Pbatt_max*ones(tf_limit/dt,1);  % half power initial traj.

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = v0*ones(1,length(Pbatt));
    a = zeros(size(v));
grad = s;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.

%Sol_array = 0.14*1;
Sol_array = 0;  % no panels.

Dist = max(ddist) - min(ddist);
if Dist == 0,     % For the case when only the final value of ddist is
available.
           Dist = max(ddist) - s0/1000;
end
Capacity = hCap;

% Solar parameters:
theta_init  = 70/180*pi;  % Degrees from normal.
theta_dot = theta_init/(3*3600); % Deg per second

% For P_sol, P_cyc:
Shift_time = 1.5;   % hours.
for i = 1:1:tf_limit/dt,
%    P_sol(i) = Sol_array*solar(t0 + i*dt);%Psol*sin(theta)*theta_dot;
    P_sol(i) = 0;

       % Include stopping to change cyclists and a 2 minute pitstop.
    Shift =Shift_time*3600;      % Cyclists shift in hour multiples. Later
to be left to agent.
    tt = t0 + i*dt;
    P_cyc(i) = cyclist([tt,Shift_time]);    % Try save and load to improve
speed, or pass the parameter.

end

if length(Wind) == 1,
%    if Wind == 1.00,
%        load winddata;
%    elseif Wind == 2.00,
%        load winddata2;
%    else
```

```
        v_wind = Wind*ones(length(Pbatt));
        %   end
else
    v_wind = Wind;
end


%imbalance = 50000;  % to start loop.
imbalance = 500;  % to start loop.
stdacc = 1;
stdvel = 1;
% Main iterative loop:
Kickout = 0;    % Termination flag to terminate oscillatory and out-of-
tolerance solutions.

%while ~(~((abs(imbalance) > 0.001*Capacity)|(stdacc > 0.1)|(stdvel >
1.75))&(Kickout >= KO_val))&(((abs(imbalance) > 0.001*Capacity)|(stdacc >
0.1)|(stdvel > 1.75))&~(Kickout >= KO_val)),
%while ~(~((abs(imbalance) > 0.01*Capacity)|(stdacc > 0.70)|(stdvel >
2.1000))&(Kickout >= KO_val))&(((abs(imbalance) > 0.01*Capacity)|(stdacc >
0.70)|(stdvel > 2.1000))&~(Kickout >= KO_val)),
while ~(~((abs(imbalance) > 0.01*Capacity)|(stdacc > 0.20)|(stdvel >
2.1000))&(Kickout >= KO_val))&(((abs(imbalance) > 0.01*Capacity)|(stdacc >
0.20)|(stdvel > 2.1000))&~(Kickout >= KO_val)),
    % Initialize recorded values: including all state variables
    % State variables:
    Ebatt = zeros(1,length(Pbatt));
    s = zeros(1,length(Pbatt));
    v = v0*ones(1,length(Pbatt));

    grad = s;

    % Step 2a:
    % Int system eqn.
    i = 2;
    while max(s)/1000 <= Dist, %for k = 1:1:Dist,
         % For grad...
         j = 2;
         if size(ddist) > 1,
            while (ddist(j-1) - min(ddist)) <= (s(i-1)./1000),
                  j = j + 1;
               end
        end

%        G = M*g*sin(ggrad(j)*pi/180);
        G = M*g*sin(ggrad(j-1)*pi/180);   % To compensate for 'look-ahead'
in grad.
        grad(i-1) = G;

        t = t0 + (i-1)*dt;    % Elapsed time.
        % Cycle parameters:
        aero_In = [L,Diameter,x,v(i-1),A_wet,A_wet_wing,L_wing,t_c];    %
All parameters listed in prototype.m
        Out = aero(aero_In);
        CdA = real(Out(1));
```

```
        D = 0.5*1.2*CdA;

        % Solve system DEs:
        % simple direct integration approach:

        if Wind == 0,
            Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            y0 = [Ebatt(i-1),s(i-1),v(i-1)];
            dydt = cycle_DE(t,y0,Para);
        else
            Para = [Pbatt(i-1),P_sol(i-1),P_cyc(i-1),D,Resist,G,M];
            y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind(i-1)];
            dydt = cycle_DE_wind(t,y0,Para);
        end
        dEbatt = dydt(1);
        ds = dydt(2);
        dv = dydt(3);
        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dEbatt*dt;
        s(i) = s(i-1) + ds*dt;
            V = v(i-1) + dv*dt;
            if V < 0, V = 0.01; end
            v(i) = V;
        % Other variables:
        a(i) = dv;

        i = i + 1;
        v_tf = V;

    end

    % Step 3a:

    % Error term:
    % Determine final time:
    t = dt;
    for i = 2:1:length(s),
        if s(i) > 0, t = t + dt; end
    end

    % Expand into for loop to remove zero levels.
%    a = zeros(size(v));

 %   for i = 1:1:length(a),
 %       a(i) = 1./(M*v(i)).*(Pbatt(i) + P_sol(i) + P_cyc(i) - (D*v(i).^3 +
Resist*v(i).^2 + grad(i).*v(i)));
 %   end
    for i = 1:1:length(a),
       t = i*dt;
        if a(i) > 5, a(i) = 5; end
        if a(i) < -5, a(i) = -5; end
    end
```

```
    imbalance = max(Ebatt) - Capacity;
   % Determine the mean and std dev of acceleration:
    acc = a(1:1:t/dt);
    acc1 = a(2:1:t/dt-1);   % Remove spikes at the ends.
    meanacc = mean(acc1);
    stdacc = std(acc1);
    vel = v(1:1:t/dt);
    meanvel = mean(vel);
    stdvel = std(vel);

%    if t0 > 460,
%        v0
%    figure;
%    plot(v);
%    figure;
%    plot(a);
%    figure;
%    plot(acc1);
%    pause;
%end


   %if sl == 2, % For horizon.
%        if (stdacc < 0.1)&(stdvel < 1),
%            Ka = 0;
%            Kv = 0;
%            LL = 2;     %8;
%        elseif (stdacc < 0.1)&(stdvel < 2),
%            Ka = 0;
%            Kv = 3;
 %           LL = 5;     %8;

            %        else
%            Ka = 2;
%            Kv = 2;
%            LL = 1;
%       else
            Ka = 2;         %13;     %%13;    %14    % 11
            Kv = 4;     %1.2507;   %%1.25;  %1.27   %1.15
            LL = 2;       %100;     %%95;    %95    %55
            %       end

        %    end

      d_P = -Ka*stdacc*(acc - meanacc) - Kv*stdvel*(vel - meanvel) -
LL*imbalance/(t/dt);%length(acc);
d_P = d_P';


    % Step 4:
    for i = 1:1:length(Pbatt),
        if i <= length(d_P),
            PP(i) = d_P(i);
```

```
        else
            PP(i) = 0;
        end
    end
    Pbatt = Pbatt + PP';


    % Constrain Pbatt to reasonable bounds.
    for i = 1:1:length(Pbatt),
        t = t0 + i*dt;
        if Pbatt(i) > Pbatt_max,
            Pbatt(i) = Pbatt_max;
        elseif Pbatt(i) < 0,
            Pbatt(i) = 0;
        end
%         if v(i) > 33, Pbatt(i) = 0; end

    end



Kickout = Kickout + 1;

if interim == 1,
        disp(sprintf('Current imbalance is %0.5g',imbalance));
         disp(sprintf('Current stdacc is %0.5g',stdacc));
        disp(sprintf('Current stdvel is %0.5g',stdvel));
        %if Kickout >=KO_val, disp('Kickout'); end
    end
            if Kickout >=KO_val, disp('Kickout'); end

end
% end of main iterative loop.


% Setup output values.
```

## D.3.28 TSPC_solver_opt2.m

```
% This is a purposely derived solver for the solar cycle problem.
% Problem statement:
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% It starts by integrating the system eqns, from s(0) to s(tf) to determine
tf.
% It then checks the energy balance and adjusts the controls accordingly.
% This is repeated until the energy balance is within a certain tolerance.
% The influence egn are then backward integrated.
% This enables one to check the closeness of fit of the optimality
condition.
```

```
% Optimised for test run.
% No solar power, can ignore pitstops.

% Calls: cycle_opt.m
% Called by: TSPC_SM_opt.m

function [s_0,v_0,Ebatt_0,t_0,Pbatt_0] =
TSPC_solver_opt2(sl,route,dt,t0,v0,s0,hCap,v_wind,horizon,Pbatt0);   % hprop
is th proportion of the horizon length comapred to the stage length, used
from reducing the Capacity used in horizon.

% Function control:
interim = 0;    % Show interim results if set.
improve = 1;
pplot = 0;

% Solver parameters:
acc_tol = 0.01;
vel_tol = 0.15;
E_tol = 10;

%Shift = Shift_time*3600;
Capacity = hCap;
ddist = route(:,1);
ggrad = route(:,2);
routelength = horizon;%max(ddist)*1000-s0;

% Step 1:
% Calc initial trajectory:
if sl == 1,
    tf_limit = 11000;
    KO_val = 20;
elseif sl == 2,                 % Default case for test run....
    tf_limit = round(routelength*1000/11);
    KO_val = 20;%7;             % Attempting to elim. spikes.
else
    tf_limit = 7000;
    KO_val = 15;
end
Pbatt_max = 500;    % Watts.
%Pbatt_0 = zeros(tf_limit/dt,1);  % zero power initial traj.
Pbatt_0 = Pbatt0*ones(tf_limit/dt,1);


[s_0,v_0,Ebatt_0,t_0] =
cycle_opt(Pbatt_0,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);

% Step 2:
% Calc a_0, etc.
% Start of main iterative loop, though should be single step.
tick = 0;
while improve,
    improve = 0;
    Pbatt = Pbatt_0;
```

```
  %  rt_0 = floor(t_0-t0);
   a_0 = diff(v_0);
%    stdacc = std(a_0);
%    meanacc = mean(a_0);
    a_0 = [0,a_0,0,0];
    % smooth accel curve:        --->> Does not help.

if length(v_0) > 20,
    meanvel = mean(v_0(1:20));
    stdvel = std(v_0(1:20));
%    a_0 = diff(v_0(1:20));
    stdacc = std(a_0(1:20));
else
    meanvel = mean(v_0);
    stdvel = std(v_0);
    a_0 = diff(v_0);
    stdacc = std(a_0);
end

for i = 1:1:length(v_0),%(Pbatt),%rt_0,

    if meanvel > 18,
       Pbatt(i) = 0;
   elseif v_0(i) < 12,
       if meanvel < 11,%stdvel < 0.5, %> 0.5,
            Pbatt(i) = Pbatt_max;
        else
%           Pbatt(i) = 250;
             Pbatt(i) = 0.5*stdvel*Pbatt_max;
         end
     else
        Pbatt(i) = Pbatt(i) - 7*(meanvel - 20);
end

if Pbatt(i) > Pbatt_max,
        Pbatt(i) = Pbatt_max;
    elseif Pbatt(i) < 0,
        Pbatt(i) = 0;
    end

end

% Step 3:
% Check and adjust the energy balance:
    balance = sum(Pbatt) - hCap;
 if (balance > 0)&(meanvel > 10),      % Drop power level.
        drop = -balance/length(v_0);
        if drop < -1,
            for i = 1:1:length(v_0),
                if Pbatt(i) + drop > 0,
                    Pbatt(i) = Pbatt(i) + drop;
                else
                    Pbatt(i) = 0;
```

```
                end
                if Pbatt(i) > Pbatt_max,
                    Pbatt(i) = Pbatt_max;
                elseif Pbatt(i) < 0,
                    Pbatt(i) = 0;
                end

            end
        end
    end


% Step 4:
% Verify and repeat if necc.

[s,v,Ebatt,t] =
cycle_opt(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);
%if (tick > 2)|((stdvel < vel_tol)&(stdacc < acc_tol)),
if (tick == 1),
    improve = 1;
    v_0 = v;
    s_0 = s;
    Ebatt_0 = Ebatt;
    Pbatt_0 = Pbatt;
else
    % Update;
    v_0 = v;
    s_0 = s;
    Ebatt_0 = Ebatt;
    t_0 = t;
    Pbatt_0 = Pbatt;
end

if pplot == 1,
    figure;
    plot(Pbatt);
    figure;
    plot(v_0);
    figure;
    plot(a_0);
    disp(sprintf('Current imbalance is %0.5g',max(Ebatt) - hCap));
    pause;
end


if interim == 1,
        disp(sprintf('Current imbalance is %0.5g',imbalance));
%         disp(sprintf('Current stdacc is %0.5g',stdacc));
%        disp(sprintf('Current stdvel is %0.5g',stdvel));
    end
end
tick = tick + 1;
% end of main iterative loop.
```

## D.3.29 cycle_opt.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Derived from cycle_PD.m for use in NM and TSPC with NM.
% Refined farther for TSPC_solver_opt vesion 2.....

% Input: Pbatt only.
% Output: Vel_ave only!! - scalar.
% Power and battery energy constrains handled in calling function.

% Calls: cycle_PD.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: ????

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
%function [s_out,v_out,Ebatt_out] =
TSPC_cycle_opt(route,dt,t0,v0,s0,Pbatt,Wind);
%function [Vel_ave,t] =
cycle_PD(PP,v_wind,ddist,ggrad,m,uscale,P2,tf_limit,pplot);
function [s_out,v_out,Ebatt_out,t] =
cycle_opt(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);



%Pbatt_max = 500;
dt = 1;

if pplot == 2,
    figure;
    plot(Pbatt);
    pause;
end



%load winddata;
% v_wind = Wind(t);

% Physical parameters:
g = 9.81;
M = 75 + 70;    % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.
```

```
% Initialize recorded values: including all state variables
% State variables:
Pbatt = Pbatt(1:tf_limit);
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = zeros(1,length(Pbatt));
% Preface initial conditions:
v = [v0,v];
s = [s0,s];
grad = s;
%t0 = 0;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.

for i = 1:1:length(Pbatt)*dt,
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);
end

% Main iterative loop:
mindist = s0/1000;%min(ddist);
maxdist = max(ddist);
    Dist = maxdist - mindist;%-max(s)/1000;%mindist;
    % Int system eqn.
    i = 1;
while ((max(s)-s0)/1000 <= Dist) & (i < length(Pbatt)-1),
        i = i + 1;
%            if i >= length(Pbatt),
%                disp('sdfg');
%           end
         % For grad...
         jj = 1;
         if jj < length(ddist),
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%                i = i
                end
        end
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consider 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % cycle parameters
            D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!
            Para = [Pbatt(i-1),P_cyc(i-1),D,Resist,G,M];
                y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind];
            dydt = cycle_DE_wind_opt(t,y0,Para);

        % System variables update:
```

```
        Ebatt(i) = Ebatt(i-1) + dydt(1)*dt;
        s(i) = s(i-1) + dydt(2)*dt;
        V = v(i-1) + dydt(3)*dt;
        if V < 0, V = 0; end
        v(i) = V;

    % Could add an interval constraint on the batt energy here. - this
should work, as better perform results with moderate power usage.
%   if (Ebatt(t-t0) > Capacity),
%       Pbatt(t-t0+1) = 0;
%   end


    end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(2:t-t0+3);
s_out = s(2:t-t0+3);
Ebatt_out = Ebatt(1:t-t0);

Vel_ave = mean(v_out);

% Cheat a bit in calculating final time, to overcome discretisation:
t = max(s_out)/Vel_ave;
```

## D.3.30 LMSerror.m

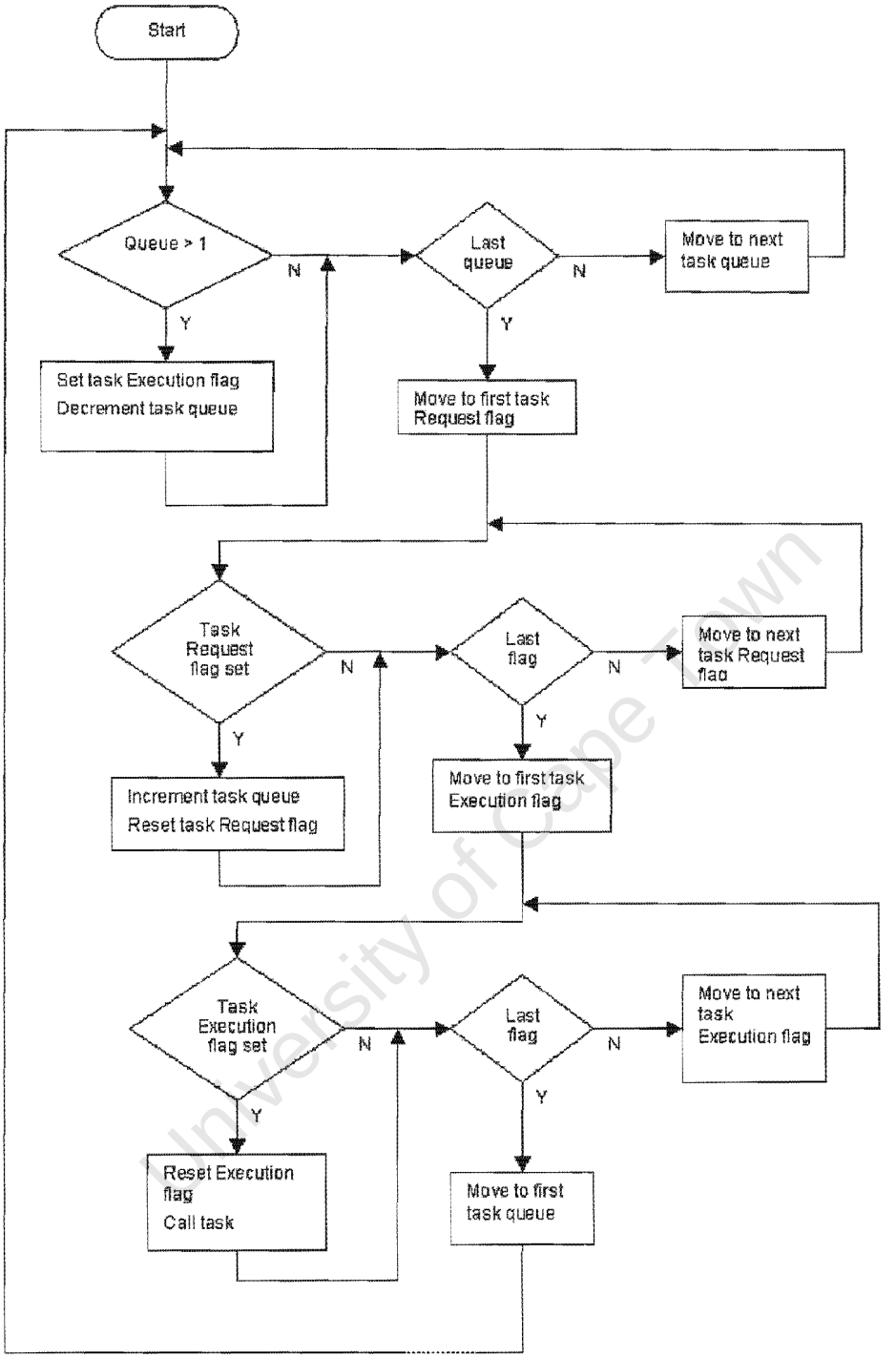
```
% This function computes tej least mean squared error between the forecast
traj. anfd the past recorded traj.
% it is called by TSPC.m

function out = LMSerror(t,f1,f2,f3)

% Test forecast 1:
E1 = LMS(t,f1);
% Test forecast 1:
E2 = LMS(t,f2);
% Test forecast 1:
E3 = LMS(t,f3);

if E1 > E2,
    if E1 > E3,
        out = 1;
    else
        out = 3;
    end
else
    if E2 > E3
```

```
        out = 2;
    else
        out = 3;
    end
end
```

```
function Err = LMS(a,b)
Err = 0;
for i = 1:1:length(a),
    e = a(i) - b(i);
    Err = Err + e^2;
end
```

# Appendix E:EMS Laptop MATLAB Code Listing

## E.1 Scheduler flowchart



## E.2 trial2.m

```
% First attempt at integration of TSPC and datalogger.
% Works.
% Developed from trial1.m - for first test run.
% Completely integrated procedure.


laptop_init;

% Let the cycle run:
%[s_c,v_c,Ebatt_c] = TSPC_cycle_opt(route,dt,0,0,0,Pbatt,Wind);

[s_d,vel_d,Ebatt_d,meanwind_d,t_d] = datalogger4(comport,logno);

% Prepare to run TSPC:
t = dt;
%for i = 2:1:length(s_c),
for i = 2:1:length(s_d),
        if s_d(i) > 0, t = t + dt; end
end
% Update time series:
%s = [0 0 0 0 0 s_c];
s = [0 0 0 0 0 s_d];
%v = [0 0 0 0 0 v_c];
v = [0 0 0 0 0 v_d];
%Ebatt = [0 0 0 0 0 Ebatt_c];
Ebatt = [0 0 0 0 0 Ebatt_d];
%P = Pbatt;
sd = 0;
vd = 0;
Ebd = 0;
mwd = 0;
td = 0;
%[s_d,vel_d,Ebatt_d,meanwind_d,t_d]

% Start looping the rest of the way.
Dist = max(ddist);

%    meanwind = mean(v_wind(1:20));    % Calculate meanwind over most recent
validation period.
meanwind = meanwind_d;
% <<<<<<<<<<<<<<< Main loop >>>>>>>>>>>>>>>>>>>>>>

while max(s)/1000 <= Dist, %for k = 1:1:Dist,
    tic;
    logno = logno + 1;

    [P,route_rem] =
TSPC_SS2_opt(Capacity,ggrad,ddist,tspan_init,tf_limit,t,s,v,Ebatt,Pbatt,mean
wind);
```

```
    % Update time series - from measurements:
    % This is the input point for the cycles measurement data.
    if length(P) > 20,
        P = P(1:20);
    end
    meanP = mean(P);

    % Issue power command to cycle:
    fprintf(comport,'>s%03.0f',meanP);

   % [s_c,v_c,Ebatt_c] = TSPC_cycle_opt(route_rem,dt,t,v(t),s(t),P,Wind);

    [s_d,vel_d,Ebatt_d,meanwind_d,t_d] = datalogger4(comport,logno);

    % Determine final time:
    %t_c = dt;
    t_d = dt;
%    for i = 2:1:length(s_c),
    for i = 2:1:length(s_d),
%        if s_c(i) > 0, t_c = t_c + dt; end
        if s_d(i) > 0, t_d = t_d + dt; end
    end

        meanwind = meanwind_d;    % Calculate meanwind over most recent
validation period.

     t0 = t;
     EE = Ebatt(t);
%     for i = 1:1:t_c,
     for i = 1:1:t_d,
        ti = t0 + i;
%         Ebatt(ti) = Ebatt_c(i) + EE;
         Ebatt(ti) = Ebatt_d(i) + EE;
%         s(ti) = s_c(i);
         s(ti) = s_d(i);
%         v(ti) = v_c(i);
         v(ti) = v_d(i);
        Pbatt(ti) = P(i);
    end
%    t = t0 + t_c;
    t = t0 + t_d;
    % Append datalogs:
    sd = [sd s_d];
    vd = [vd vel_d];
    Ebd = [Ebd Ebatt_d];
    mwd = [mwd meanwind_d];
    td = td + t_d;

    toc;
end
```

```
% <<<<<<<<<<<<<<<<< End of main loop - include incremental saves.
>>>>>>>>>>>>>>>>>>>>>>>>>

% Send turnoff command:
t1 = clock;
fprintf(comport,'>sc');
disp('Turn-off instruction sent.');
while etime(clock,t1) < 15, end % To verify turn-off instruction.
%fprintf(comport,'>sc');
%disp('Turn-off instruction sent, again.');

% Terminate serial connection
laptop_term;

% Save workspace to workspace.mat:
save workspace;
```

## E.3 laptop_init.m

```
% This file is used to initialise the system for the test run.
% To be expanded later.

% <<<<<<<<<<<<<<<<<<<< TSPC_SM init code. >>>>>>>>>>>>>>>>>>>>>>

selectroute = 8;%9;
showplot = 0;
Wind = 1;   % Wind handling needs to change to use wind speed readings.

Capacity = 750*3600;    % Reduce capacity in proportion to stage length and
forecast horizon length.
% For short route:
%Capacity = Capacity/4;

dt = 1;             % time step in seconds - code does not operate on
anything other than dt = 1.
%Shift_time = 1.5; % hours.
%valid_period = 100;
valid_period = 20;
logno = 0;

switch selectroute      % default to shortroute
case 1
%    load fielddata;
    load testroutedata;
    ddist = dist;
    ggrad = grad;
    Capacity = Capacity/4;
case 9
%    load fielddata;
    load testroutedata2;
    ddist = dist1;
    ggrad = grad1;
```

```
       Capacity = Capacity/4;
%    tspan_init = 10000;   % Starting traj time span in seconds.
otherwise
    load shortroute;
    ddist = srd;
    ggrad = srg;
    Capacity = Capacity/4;
  %  tspan_init = 4000;   % Starting traj time span in seconds.
%    LL = 1;

end

% Initial power traj to get the cycle rolling.
tf_limit = 11000;   % 20kph for 56 km.
tspan_init = 20;   % Starting traj time span in seconds.
Pbatt_max = 500;    % Watts.
%Pbatt = random('norm',450,150,tf_limit/dt,1); % Use a normal distibution
about 15, with a spread of 1? Plot looks acceptable.
Pbatt = 0.5*Pbatt_max*ones(tspan_init,1);  % half power initial traj.
P0 = 0.5*Pbatt_max*ones(tspan_init,1);
route = zeros(length(ddist),2);
for i = 1:1:length(ddist),
    route(i,1) = ddist(i);
    route(i,2) = ggrad(i);
end
if Wind == 1,
    load winddata;
elseif Wind == 2,
    v_wind = wind(tf_limit);
    Wind = v_wind;
elseif Wind == 3,   % Measure wind...
    % ???
end

% <<<<<<<<<<<< Datalogger2.m init code, from datalogger2.m: >>>>>>>>>>>>>


comport = serial('COM1');
%set(comport, 'Terminator', 'CR')  % Terminiator on sent commands.
set(comport, 'BaudRate', 19200, 'StopBits', 1, 'DataBits', 8)
%set(comport, 'Terminator', 'CR', 'Parity', 'odd')
set(comport, 'Parity', 'odd')
set(comport, 'FlowControl', 'none')
comport   % To check port values, etc.

fopen(comport);   % Open device.
get(comport,'Status')    % Should return "open".

% Asynchronous:
% Continuos:
set(comport, 'ReadAsyncMode', 'continuous')

get(comport, 'BytesAvailable')    % Check is data is in RX buffer.
```

```
% Xfer data with FREAD, FSCANF, etc.

% Initialise:
packet_log = 0;
p_log = 0;
ABlog = 0;
CDlog = 0;
CAlog = 0;
CBlog = 0;
EFlog = 0;
i = 0;
j = 0;
bytesQued = 0;%get(comport, 'BytesAvailable');
command = 10;
toggle = 2; % Must start with high to issue first command. Else "run-
through".
tic;

% Turn on scheduler, use a delay to allow DSP to boot, and me to turn it
on.... :
t0 = clock;
while etime(clock,t0) < 15, end
fprintf(comport,'>st');
disp('Turn-on instruction sent.');
fprintf(comport,'>s%03.0f',command);
t0 = clock;
while etime(clock,t0) < 15, end % To verify turn-on instruction.
%fprintf(comport,'>st');
%disp('Turn-on instruction sent, again.');

%while i == 0,

% Write a command:
%command = command + 1;      % Ramp
% Random command:
%command = round(1000*0.6*rand(1))
% Stepped input:          % Step
%if j == 40,
%    command = 50;
    %end
%if j == 70,
%    command = 150;
    %end
command = 25;
%if command >= 600, command = 1; end
%fprintf(comport,'>s%03.0f',command);    % Fixed point 3 digits
%fprintf(comport,'>s%04.0x',command);     % Hex 4 digits = 2 char
% Send command every second cycle, i.e. every second.
%if toggle == 2,
    fprintf(comport,'>s%03.0f',command);
    %end
```

## E.4 datalogger4.m

```
% Integration code for serial datalogger.
% MATLAB needs to be restarted if an error in opening the COM port is found.
% Developed from datalogger2.m
% Uses a ring buffer to catch and detect packets.
% Added random command generator to check Icontroller and powerController.
% Added a step to measure step response of system.

% Based on code in instrschool.
% Developed from datalogger3.m
% For first test run.
% Completely integrated procedure.
% Note change in logged variable assignments.

function [s,vel,Ebatt,meanwind,delta_t] = datalogger4(s,logno);

% Initialise:
packet_log = 0;
p_log = 0;
ABlog = 0;
CDlog = 0;
CAlog = 0;
CBlog = 0;
EFlog = 0;
i = 0;
j = 0;
bytesQued = 0;%get(s, 'BytesAvailable');
%speed = 0;
%toggle = 2; % Must start with high to issue first command. Else "run-
through".

% Start timing for distance calc.
t0 = clock;

while i == 0,


%while bytesQued < 24,    % Increased packet size.
%while bytesQued < 50,    % Should contain two packets.
while bytesQued < 50,       % To test serial buffer size, run to 2k. 500 OK.
600 not -> 512 byte buffer.
    bytesQued = get(s, 'BytesAvailable');
end

% Read serial buffer into ring buffer, and search for packet header.
start = 0;
h_check = 0;
packetheader = [2;88;80];       % MATLAB is not checking the whole vector -
fault in NOT.
loop = 1;
start = fread(s,25,'uint8');  % load ring buffer.
```

```
while loop,
%    for f = 1:1:24,
        h_check = start(1:3);
        if (h_check == packetheader)&(loop == 1),
            loop = 0;
            data = start(4:25);
        else
            loop = 1;
%            oldstart = start(4:24);
%            start = [oldstart;fread(s,3,'int8')];
            oldstart = start(2:25);
            start = [oldstart;fread(s,1,'uint8')];
        end
        %   end
end


%    disp(sprintf('Data set:'))  %2d',data))
%    disp(data)
    j = j + 1;
    if j == 20;%200,     % Loop termination flag.
        i = 1;
    end

    % Format and integrate data into logs:
    [err,packet,AB,CD,CA,CB,EF] = dataformat(data)
    if err == 1,
        disp('Packet error, rejected packet.');
    else
        % Check for wrapping:
        if packet_log > packet,
            packet = packet_log + 1;    % Needs improvement.
        end
        % Check for packet sequence:    % Ignore missed packets.
            ABlog = [ABlog,AB];
            CDlog = [CDlog,CD];
            CAlog = [CAlog,CA];
            CBlog = [CBlog,CB];
            EFlog = [EFlog,EF];
            packet_log = packet;
            p_log = [p_log,packet];

    end



end

% Process logs:
% Add code to strip extra zeros & Fill some o/p variables:
%vel = CDlog;
%iFB = CAlog;
```

```
%V = CBlog;
%wind = EFlog;
%grad = ABlog;
% Calculate distance traveled:
delta_t = etime(clock,t0);
meanvel = mean(vel);
s = meanvel/delta_t;
Ebatt = sum(iFB.*V);        % Assumes that vectors are the same length....
Make it so....
meanwind = mean(wind) - meanvel;

% Save logs and processed results:

% Use functional form for str parameters: SAVE('filename','var1','var2')

save(sprintf('file%03.0f',logno),'vel','iFB','V','wind','grad','s','delta_t'
);
```

## E.5 dataformat.m

```
% Data formatting function.
% Takes the data matrix supplied by the datalogging script or function
% and pulls out the packet number and other variables.
% With the packet marker beeing moved to the header check, the data chunk
has lost its first value.
% Hence all indices must be reduced by one.

function [err,packet,AB,CD,CA,CB,EF] = dataformat(data)

% Check that the packet is formatted correctly:
err = 0;
%if ((data(1) ~=
80)|(data(3)~=65)|(data(4)~=66)|(data(7)~=67)|(data(8)~=68)),  - moved to
header test.
%if ((data(2)~=65)|(data(3)~=66)|(data(6)~=67)|(data(7)~=68)),  - bigger
packet.
if
((data(2)~=65)|(data(3)~=66)|(data(6)~=67)|(data(7)~=68)|(data(10)~=67)|(dat
a(11)~=65)|(data(14)~=67)|(data(15)~=66)),
    err = 1;
end

%packet = data(2);
packet = data(1);
%AB = data(5)*10 + data(6); % Ass: values are char.
%CD = data(9)*10 + data(10);
% Join high and low bytes again:
%AB = data(5)*2^8 + data(6); %AB = data(5)*100 + data(6);
%CD = data(9)*2^8 + data(10); %CD = data(9)*100 + data(10);
%EF = data(13);

AB = data(4)*2^8 + data(5); %AB = data(5)*100 + data(6);
```

```
CD = data(8)*2^8 + data(9); %CD = data(9)*100 + data(10);
% Temp for checking proc_cmd:
%CD = data(9);
CA = data(12)*2^8 + data(13); %CD = data(9)*100 + data(10);
CB = data(16)*2^8 + data(17); %CD = data(9)*100 + data(10);
EF = data(20)*2^8 + data(21);

%if AB > 1500, AB = 1500; end;
%if CA > 1500, CA = 1500; end;
%if CB > 1500, CB = 1500; end;
%if CD > 1500, CD = 1500; end;
%if EF > 1500, EF = 1500; end;

if AB > 1500, AB = 0; end;
if CA > 1500, CA = 0; end;
if CB > 1500, CB = 0; end;
if CD > 1500, CD = 0; end;
if EF > 1500, EF = 0; end;
```

## E.6 TSPC_SS2_opt.m

```
% Implementation code.
% This is the third attempt at writing the TSPC.
% This is a proper shooting method approach, based on the Secant method, see
FB p??.

% Basic steps:
% 1. Measure states - external.
% 2. Generate forecasts.
% 3. Generate optimal trajectories.
% 4. Validate trajectories.
% 5. Repeat.

% This code will need to be modified to run from instrumentation data.
% Also optimised for speed in laptop on test run - NO solar power!!
% Calls optimised functions: TSPC_solver_opt.m, TSPC_cycle_opt.m,
% Tuned for MEX-file compilation, as are TSPC_solver_opt.m, TSPC_cycle_opt.m
and dependant functions.
% Changed to Nelder-Mead solver.
% Changed to second simple solver.

% Initialisation functions moved to laptop_init.m, terminiation functions
moved to laptop_term.m
% Converted to function.
% Setup for wind measurements, not estimation of wind.

function [P,route_rem] =
TSPC_SS2_opt(Capacity,ggrad,ddist,tspan_init,tf_limit,t,s_c,v_c,Ebatt_c,Pbat
t,meanwind);

dt = 1; % Retain as code not operable on anything else.
```

```
% Physical parameters:
g = 9.81;
M = 75 + 70;    % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.


tol = 0.001*Capacity;
Pbatt_max = 500;    % Watts.

% Initialize recorded values: including all state variables
% State variables:
Ebatt = zeros(1,tf_limit);
s = zeros(1,tf_limit);
v = 0.01*ones(1,tf_limit);
%for i = 1:dt:length(Pbatt),
%for i = t-length(Pbatt):dt:t,
for i = 1:1:length(s_c),
    Ebatt(i) = Ebatt_c(i);
    s(i) = s_c(i);
    v(i) = v_c(i);
end

%t = tspan_init;

%Dist = max(ddist);
i = 2;
% Main loop - moved externally:
%while max(s)/1000 <= Dist, %for k = 1:1:Dist,

% Step 1: Measurements - handled outside for states, wind
      % x: E,s,v
      % For grad... prediction.
         j = 1;
         while ddist(j) <= (s(i-1)./1000),
               j = j + 1;
            end
        G = M*g*sin(ggrad(j)*pi/180);
        grad(i-1) = G;

        % Wind:
        % Cycle parameters:
       % aero_In = [L,Diameter,x,v(i-1),A_wet,A_wet_wing,L_wing,t_c];   %
All parameters listed in Trike.m
       % Out = aero(aero_In);
       % CdA = real(Out(1)); % actual value with no wind. Use in
measurements of wind uncertainty - later.


% Step 2: Forecasts:
       wind_inc = meanwind + meanwind*0.1;
       wind_dec = meanwind - 0.11*meanwind;
       wind_same = meanwind;
```

```
% Step 3: Solve traj.
       disp('Step 3')
        horizon = 2;    % <--------------- Forecast horizon setting!! in km.
---------------------->
%        horizon = 10;    % 2 is too small, gives null returns in solver. 5
seems ok, expect at end.
       % Find current position:
       j = 1;
    while ddist(j) <= (s(t/dt-1)./1000),
            j = j + 1;
       end
       if ddist(j) > (s(t/dt-1)./1000),
           j = j - 1;
       end
       % Find end point:
       k = j;
           while (ddist(k) <= (s(t/dt-1)./1000 + horizon)) & (k <
length(ddist)),
               k = k + 1;
           end

       % Setup forecast route:
        if k > j,
            route = zeros(k-j+1,2);
            if k == j + 1,
                route(1,1) = ddist(j);
                route(2,1) = ddist(k);
                route(1,2) = ggrad(j);
                route(2,2) = ggrad(k);
            else
               for ii = 0:1:k-j,
                route(ii+1,1) = ddist(ii+j);
                route(ii+1,2) = ggrad(ii+j);
               end
           end
           segment_dist = max(route(:,1)) - min(route(:,1));
       else
           route = zeros(2,2);
           route(2,1) = ddist(k);
           route(1,1) = s(t)/1000;
           route(2,2) = ggrad(k);
           route(1,2) = ggrad(j-1);
           segment_dist = max(route(:,1)) - min(route(:,1));
       end


       if segment_dist == 0,     % For the case when only the final value of
ddist is available.
           segment_dist = max(route(:,1)) - max(s)/1000;
       end
        segment_capacity = Capacity*segment_dist/max(ddist);
%       disp('horz. dist:');
%       disp(segment_dist);
```

```
%       disp('capacity:');
%       disp(Capacity);
%       disp('seg. Cap.');
%       disp(segment_capacity);
%        pause;

      % Solve over forecast horizon:
%         if t > 460,
%        v(t-1)
%        t
%        plot(v)
%        pause;
% end
      disp('inc forecast.')
%        [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-1),horizon_capacity,wind_inc);
%        [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_inc);
%        [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_inc,horizo
n);
%        [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_inc,horiz
on);
       [a1,v_inc,a3,t_end_inc,Pbatt_inc] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_inc,hori
zon,Pbatt(t/dt));
%        disp('hor. cap: ');
%        disp(horizon_capacity);
       %pause;  % No horizon_capacity!!
       disp('dec forecast.')
%        [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-1),horizon_capacity,wind_dec);
%        [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_dec);
%        [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_dec,horizo
n);
%        [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_dec,horiz
on);
       [a1,v_dec,a3,t_end_dec,Pbatt_dec] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_dec,hori
zon,Pbatt(t/dt));
      disp('static forecast.')
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt(2,route,dt,t,v(t/dt-1),s(t/dt-
1),horizon_capacity,wind_same);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_same);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_NM(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_same,horiz
on);
%        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_PD2(2,route,dt,t,v(t/dt),s(t/dt),horizon_capacity,wind_same,hori
zon);
```

```
        [a1,v_same,a3,t_end_same,Pbatt_same] =
TSPC_solver_opt2(2,route,dt,t,v(t/dt),s(t/dt),segment_capacity,wind_same,hor
izon,Pbatt(t/dt));


if t > 2900,
    disp('halt');
end

       % Setup remaining route:
       route_rem = zeros(length(ddist)-j,2);
       for ii = 1:1:(length(ddist) - j),
           route_rem(ii,1) = ddist(ii + j-1);
           route_rem(ii,2) = ggrad(ii + j-1);
       end
%       rem_dist = max(ddist) - route_rem(1,1);
    rem_dist = max(ddist) - s(t)/1000;
       rem_prop = rem_dist/max(ddist);
       if rem_dist < horizon,
           route_rem = zeros(length(ddist(j:length(ddist))),2);
           route_rem(:,1) = ddist(j:length(ddist));
           route_rem(:,2) = ggrad(j:length(ddist));
       end

% Step 4: Validation & Traj output
    disp('Step 4');
    % Wait for sufficient time to pass....

    % Setup recorded Vel from past validation period. -> need to include
pitstops.
%       valid_period = 100;   % seconds
       valid_period = 20;   % seconds
%       valid_period = 60;   % seconds
    im = length(v_inc);
    dm = length(v_dec);
    sm = length(v_same);
    small = min([im,sm,dm]);
    if small < valid_period,
        valid_period = min([im,sm,dm]);
    end

%           v_rec = v(t - valid_period:1:t-1);      % Recorded velocity for
last validation period.
    if t < valid_period,
        v_rec = zeros(size(valid_period));
    else
           v_rec = v_c(t - (valid_period-1):1:t-1);      % Recorded velocity
for last validation period.
       end


           v_inc_rec = v_inc(1:1:valid_period);

           v_dec_rec = v_dec(1:1:valid_period);
```

```
            v_same_rec = v_same(1:1:valid_period);

            traj = LMSerror(v_rec,v_inc_rec,v_dec_rec,v_same_rec);
             if traj == 1,
                 Ptraj = Pbatt_inc;
                 disp(sprintf('wind increasing, t: %5d',t));
             elseif traj == 2,
                 Ptraj = Pbatt_dec;
                 disp(sprintf('wind decreasing, t: %5d',t));
             else traj == 3,          % Default.
                 Ptraj = Pbatt_same;
                 disp(sprintf('wind static, t: %5d',t));
             end
             disp(sprintf('distance: %5d',s(t)));

      %       for ii = t/dt:1:(length(Ptraj) + t/dt - 1),
      %           Pbatt(ii) = Ptraj(ii - t/dt + 1);
      %      end

% Step 5: Update cycle model and setup measurements -external.
% Step 5: Output power trajectory.
disp('Step 5');

P = Ptraj(1:1:valid_period);


%end
```

## E.7 TSPC_solver_opt2.m

```
% This is a purposely derived solver for the solar cycle problem.
% Problem statement:
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% It starts by integrating the system eqns, from s(0) to s(tf) to determine
tf.
% It then checks the energy balance and adjusts the controls accordingly.
% This is repeated until the energy balance is within a certain tolerance.
% The influence egn are then backward integrated.
% This enables one to check the closeness of fit of the optimality
condition.

% Optimised for test run.
% No solar power, can ignore pitstops.

% Calls: cycle_opt.m
% Called by: TSPC_SM_opt.m

function [s_0,v_0,Ebatt_0,t_0,Pbatt_0] =
TSPC_solver_opt2(sl,route,dt,t0,v0,s0,hCap,v_wind,horizon,Pbatt0);   % hprop
is th proportion of the horizon length comapred to the stage length, used
from reducing the Capacity used in horizon.
```

```
% Function control:
interim = 0;    % Show interim results if set.
improve = 1;
pplot = 0;

% Solver parameters:
acc_tol = 0.01;
vel_tol = 0.15;
E_tol = 10;

%Shift = Shift_time*3600;
Capacity = hCap;
ddist = route(:,1);
ggrad = route(:,2);
routelength = horizon;%max(ddist)*1000-s0;

% Step 1:
% Calc initial trajectory:
if sl == 1,
    tf_limit = 11000;
    KO_val = 20;
elseif sl == 2,                 % Default case for test run....
    tf_limit = round(routelength*1000/11);
    KO_val = 20;%7;             % Attempting to elim. spikes.
else
    tf_limit = 7000;
    KO_val = 15;
end
Pbatt_max = 500;    % Watts.
%Pbatt_0 = zeros(tf_limit/dt,1);  % zero power initial traj.
Pbatt_0 = Pbatt0*ones(tf_limit/dt,1);


[s_0,v_0,Ebatt_0,t_0] =
cycle_opt(Pbatt_0,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);

% Step 2:
% Calc a_0, etc.
% Start of main iterative loop, though should be single step.
tick = 0;
while improve,
    improve = 0;
    Pbatt = Pbatt_0;
  %  rt_0 = floor(t_0-t0);
   a_0 = diff(v_0);
%    stdacc = std(a_0);
%    meanacc = mean(a_0);
    a_0 = [0,a_0,0,0];
    % smooth accel curve:       --->> Does not help.

if length(v_0) > 20,
    meanvel = mean(v_0(1:20));
```

```
    stdvel = std(v_0(1:20));
%    a_0 = diff(v_0(1:20));
    stdacc = std(a_0(1:20));
else
    meanvel = mean(v_0);
    stdvel = std(v_0);
    a_0 = diff(v_0);
    stdacc = std(a_0);
end

for i = 1:1:length(v_0),%(Pbatt),%rt_0,

    if meanvel > 18,
       Pbatt(i) = 0;
   elseif v_0(i) < 12,
       if meanvel < 11,%stdvel < 0.5, %> 0.5,
            Pbatt(i) = Pbatt_max;
        else
%           Pbatt(i) = 250;
            Pbatt(i) = 0.5*stdvel*Pbatt_max;
         end
     else
        Pbatt(i) = Pbatt(i) - 7*(meanvel - 20);
end

if Pbatt(i) > Pbatt_max,
        Pbatt(i) = Pbatt_max;
    elseif Pbatt(i) < 0,
        Pbatt(i) = 0;
    end

end

% Step 3:
% Check and adjust the energy balance:
    balance = sum(Pbatt) - hCap;
 if (balance > 0)&(meanvel > 10),     % Drop power level.
        drop = -balance/length(v_0);
        if drop < -1,
            for i = 1:1:length(v_0),
                if Pbatt(i) + drop > 0,
                    Pbatt(i) = Pbatt(i) + drop;
                else
                    Pbatt(i) = 0;
                end
                if Pbatt(i) > Pbatt_max,
                    Pbatt(i) = Pbatt_max;
                elseif Pbatt(i) < 0,
                    Pbatt(i) = 0;
                end

            end
        end
```

```
    end


% Step 4:
% Verify and repeat if necc.

[s,v,Ebatt,t] =
cycle_opt(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);
%if (tick > 2)|((stdvel < vel_tol)&(stdacc < acc_tol)),
if (tick == 1),
    improve = 1;
    v_0 = v;
    s_0 = s;
    Ebatt_0 = Ebatt;
    Pbatt_0 = Pbatt;
else
    % Update;
    v_0 = v;
    s_0 = s;
    Ebatt_0 = Ebatt;
    t_0 = t;
    Pbatt_0 = Pbatt;
end

if pplot == 1,
    figure;
    plot(Pbatt);
    figure;
    plot(v_0);
    figure;
    plot(a_0);
    disp(sprintf('Current imbalance is %0.5g',max(Ebatt) - hCap));
    pause;
end


if interim == 1,
        disp(sprintf('Current imbalance is %0.5g',imbalance));
%         disp(sprintf('Current stdacc is %0.5g',stdacc));
%        disp(sprintf('Current stdvel is %0.5g',stdvel));
    end
end
tick = tick + 1;
% end of main iterative loop.
```

## E.8 cycle_opt.m

```
% This is a model of the solar cycle.
% States: E_batt, s, v
% Controls: P_batt, P_sol, P_cyc.
% Given a power trajectory the model returns the state variables for that
period - optimised for test run.
% Includes random wind effect.
% Need to add wind measurement.
% No solar power.
% Derived from cycle_PD.m for use in NM and TSPC with NM.
% Refined farther for TSPC_solver_opt vesion 2.....

% Input: Pbatt only.
% Output: Vel_ave only!! - scalar.
% Power and battery energy constrains handled in calling function.

% Calls: cycle_PD.m, cyclist_opt.m, aero.m, protomass_opt.m
% Called by: ????

%function [s,v,Ebatt] = TSPC_cycle(route,dt,Shift_time,t0,v0,s0,Pbatt);
%function [s_out,v_out,Ebatt_out] =
TSPC_cycle_opt(route,dt,t0,v0,s0,Pbatt,Wind);
%function [Vel_ave,t] =
cycle_PD(PP,v_wind,ddist,ggrad,m,uscale,P2,tf_limit,pplot);
function [s_out,v_out,Ebatt_out,t] =
cycle_opt(Pbatt,v_wind,ddist,ggrad,tf_limit,pplot,Capacity,v0,t0,s0);


%Pbatt_max = 500;
dt = 1;

if pplot == 2,
    figure;
    plot(Pbatt);
    pause;
end


%load winddata;
% v_wind = Wind(t);

% Physical parameters:
g = 9.81;
M = 75 + 70;    % Measured mass of cycle and est. of cyclist.
CdA = 0.85; % Mean value from aero.m over 15 to 25 m/s speed range. Replace
with empirical value.

% Initialize recorded values: including all state variables
% State variables:
Pbatt = Pbatt(1:tf_limit);
Ebatt = zeros(1,length(Pbatt));
s = zeros(1,length(Pbatt));
v = zeros(1,length(Pbatt));
% Preface initial conditions:
v = [v0,v];
```

```
s = [s0,s];
grad = s;
%t0 = 0;

% System parameters:
D = 0.5*1.2*CdA;
Resist = 0.005*M;   % Improve - use correct eqn.

for i = 1:1:length(Pbatt)*dt,
    tt = t0 + i*dt;
    P_cyc(i) = cyclist_opt(tt);
end

% Main iterative loop:
mindist = s0/1000;%min(ddist);
maxdist = max(ddist);
    Dist = maxdist - mindist;%-max(s)/1000;%mindist;
    % Int system eqn.
    i = 1;
while ((max(s)-s0)/1000 <= Dist) & (i < length(Pbatt)-1),
        i = i + 1;
%            if i >= length(Pbatt),
%                disp('sdfg');
%           end
         % For grad...
         jj = 1;
         if jj < length(ddist),
                while (ddist(jj)-mindist) <= (s(i-1)./1000 - mindist),
                   jj = jj + 1;
%               i = i
               end
        end
        G = M*g*sin(ggrad(jj)*pi/180);  % Need to consider 'look-ahead'
problem.
        grad(i-1) = G;
        t = t0 + (i-1)*dt;    % Elapsed time.
        % cycle parameters
            D = 0.5*1.2*CdA;
        % Solve system DEs:
        % simple direct integration approach:   NO SOLAR POWER!
            Para = [Pbatt(i-1),P_cyc(i-1),D,Resist,G,M];
                y0 = [Ebatt(i-1),s(i-1),v(i-1),v_wind];
            dydt = cycle_DE_wind_opt(t,y0,Para);

        % System variables update:
        Ebatt(i) = Ebatt(i-1) + dydt(1)*dt;
        s(i) = s(i-1) + dydt(2)*dt;
        V = v(i-1) + dydt(3)*dt;
        if V < 0, V = 0; end
        v(i) = V;

    % Could add an interval constraint on the batt energy here. - this
should work, as better perform results with moderate power usage.
```

```
%   if (Ebatt(t-t0) > Capacity),
%       Pbatt(t-t0+1) = 0;
%   end


    end

% end of main iterative loop.


% Remove I.C's from s,v,Ebatt arrays:
v_out = v(2:t-t0+3);
s_out = s(2:t-t0+3);
Ebatt_out = Ebatt(1:t-t0);

Vel_ave = mean(v_out);

% Cheat a bit in calculating final time, to overcome discretisation:
t = max(s_out)/Vel_ave;
```

## E.9 cycle_DE_wind_opt.m

```
% cycle_DE.m This file contains the system DEs for the cycle.
% for use in ODEsIver in calcengineB2.m
% And in own solver, Mine.m
% Also called by: Steady.m, TSPC_cycle.m, TSPC_cycle_cP.m
% Removed solar power.

function dydt = cycle_DE_wind_opt(t,y,p);  %p1,p2,p3,p4,p5,p6,p7);

E_batt = y(1);
s = y(2);
v = y(3);
v_wind = y(4);
P_batt = p(1);
%P_sol = p(2);
P_cyc = p(2);
D = p(3);
R = p(4);
G = p(5);
M = p(6);

v_apparent = v + v_wind;

% Prevent v and v_apparent from hitting zero. Prevents singularities in the
y_3, acceleration term.
if v_apparent <= 0,
    v_apparent = 0.001;
end
if v <= 0,
    v = 0.001;
end
```

```
y_1 = P_batt;

y_2 = v;

%y_3 = 1/(M*v)*(P_batt + P_sol + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));
y_3 = 1/(M*v)*(P_batt + P_cyc - (D*v_apparent^3 + R*v^2 + G*v));

% Limit acceleration to +- g.
if y_3 > 9.81,
    y_3 = 9.81;
elseif y_3 < -9.81,
    y_3 = -9.81;
end

dydt = [y_1;y_2;y_3];
```

## E.10 cyclist_opt.m

```
% Cyclist function.
% maps out an exponential curve of the cyclist's steady state output.
% Optimised for test run - can ignore pitstops.

function out = cyclist_opt(t);

%t = in(1);
%ST = in(2);
%Shift = ST*3600;
rider = 2;      % Currently planning to have Andrew cycling.



   if (rider == 1),    % Peter
        out = 150*exp(-0.5e-3.*(t-180)) + 250; % Model valid for Peter.
   elseif (rider == 2),    % Andrew
        out = 150*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Andrew.
   elseif (rider == 3),    % Mark
        out = 100*exp(-0.5e-3.*(t-180)) + 200; % Model valid for Mark or
Roelie.
    elseif (rider == 6),  % Gordon
        out = 100*exp(-0.5e-3.*(t-180)) + 100; % Model valid for Gordon.
    end
```

## E.11 LMSerror.m

```
% This function computes tej least mean squared error between the forecast
traj. anfd the past recorded traj.
```

```
% it is called by TSPC.m

function out = LMSerror(t,f1,f2,f3)

% Test forecast 1:
E1 = LMS(t,f1);
% Test forecast 1:
E2 = LMS(t,f2);
% Test forecast 1:
E3 = LMS(t,f3);

if E1 > E2,
    if E1 > E3,
        out = 1;
    else
        out = 3;
    end
else
    if E2 > E3
        out = 2;
    else
        out = 3;
    end
end




function Err = LMS(a,b)
Err = 0;
for i = 1:1:length(a),
    e = a(i) - b(i);
    Err = Err + e^2;
end
```



## E.12 laptop_term.m

```
% This is the clean-up and termination file for the mobile system, i.e. the
test run system.


% Code to close and clean up serial comms session:
% stop async:
stopasync(comport);
% clean up:
fclose(comport);
delete(comport);
clear comport
```

# Appendix F: C code listing for TI F243 DSP

## F.1 C243.h

```
/************************************************************
*   TMS320x240 Test Bed Code                                 *
*   Texas Instruments, Inc.                                  *
*   Copyright (c) 1996 Texas Instruments Inc.                *
*   11/05/96   Version 1.0                                   *
*   Jeff Crankshaw                                           *
*     Creator:     As above                  *
*     Modifier:    Gordon Webber             *
*                                            *
*     Revisions/modifications:               *
*     1. Altered header, added comments indicating  *
*           sections of code requiring changes.    *
*           05/02/02.                        *
*     2. Trashed some EVM def. and ext. mem. inf. bits. 19/02/02.
**
*                                                          *
*   TMS320C240 Peripheral Register Addresses        *
*     -> check if valid for C243                    *
*                                                          *
************************************************************/
#ifndef c240_h
#define c240_h

#include "typedefs.h"


/*-----------------------------------------------*/
/* Definitions specific to MLT Dev I/O space  */
/*-----------------------------------------------*/



/*-------------------------------------------------------*/
/* Definitions of CPU core registers                     */
/*-------------------------------------------------------*/
#define IMR_REG      (( PORT )0x0004 )
#define IFR_REG      (( PORT )0x0006 )
#define PIVR        (( PORT ) 0x701e )  /* Peripheral Int Vector Reg. Use to
check Int 1 vector.

/*-------------------------------------------------------*/
/* System Module Registers                               */
/*-------------------------------------------------------*/
#define SYSCR       (( PORT )0x07018)      /* System Module Control Register
*/ /* scsr */
#define SYSSR       (( PORT )0x0701A)      /* System Module Status Register
*/
#define SYSIVR            (( PORT )0x0701E)      /* System Interrupt Vector
Register *//* read peripheral ivr.     */
```

```
#define XINT1_CR          (( PORT )0x07070)      /* Int1 (type A) Control
reg */
#define NMI_CR            (( PORT )0x07072)      /* Non maskable Int (type
A) Control reg */
#define XINT2_CR    (( PORT )0x07078)      /* Int2 (type C) Control reg */
#define XINT3_CR    (( PORT )0x0707A)      /* Int3 (type C) Control reg */
#define PDPINT_CR         (( PORT )0x0742C)      /* Power Drive Protection
Int cntl reg */

/*-------------------------------------------------------*/
/* System Interrupt Vector Register - Address offsets */
/*-------------------------------------------------------*/
#define PHANTOM_INT_VECTOR    0x00
#define NMI_INT_VECTOR        0x02
#define XINT1_INT_VECTOR      0x01
#define XINT2_INT_VECTOR      0x11
#define XINT3_INT_VECTOR      0x1f
#define SPI_INT_VECTOR        0x05
#define SCI_RX_INT_VECTOR     0x06
#define SCI_TX_INT_VECTOR     0x07
#define RTI_INT_VECTOR        0x10
#define PDP_INT_VECTOR        0x20
#define EV_CMP1_INT_VECTOR    0x21
#define EV_CMP2_INT_VECTOR    0x22
#define EV_CMP3_INT_VECTOR    0x23
#define EV_SCMP1_INT_VECTOR   0x24
#define EV_SCMP2_INT_VECTOR   0x25
#define EV_SCMP3_INT_VECTOR   0x26
#define EV_T1PER_INT_VECTOR   0x27
#define EV_T1CMP_INT_VECTOR   0x28
#define EV_T1UF_INT_VECTOR    0x29
#define EV_T1OF_INT_VECTOR    0x2a
#define EV_T2PER_INT_VECTOR   0x2b
#define EV_T2CMP_INT_VECTOR   0x2c
#define EV_T2UF_INT_VECTOR    0x2d
#define EV_T2OF_INT_VECTOR    0x2e
#define EV_T3PER_INT_VECTOR   0x2f
#define EV_T3CMP_INT_VECTOR   0x30
#define EV_T3UF_INT_VECTOR    0x31
#define EV_T3OF_INT_VECTOR    0x32
#define EV_CAP1_INT_VECTOR    0x33
#define EV_CAP2_INT_VECTOR    0x34
#define EV_CAP3_INT_VECTOR    0x35
#define EV_CAP4_INT_VECTOR    0x36
#define AC2_INT_VECTOR        0x04

/*-------------------------------------------------------*/
/* Digital I/O  Registers                                */
/*-------------------------------------------------------*/
#define OCRA        (( PORT )0x07090)   /* Output Control Reg A */
#define OCRB        (( PORT )0x07092)   /* Output Control Reg B */
#define PADATDIR          (( PORT )0x07098)   /* I/O port A Data & Direction
reg. */
```

```
#define PBDATDIR          (( PORT )0x0709A)   /* I/O port B Data & Direction
reg. */
#define PCDATDIR          (( PORT )0x0709C)   /* I/O port C Data & Direction
reg. */
#define PDDATDIR          (( PORT )0x0709E)   /* I/O port D Data & Direction
reg. */

/*-------------------------------------------------------------------*/
/* Watch-Dog(WD) / Real Time Int(RTI) / Phase Lock Loop(PLL) Registers */
/*-------------------------------------------------------------------*/
#define RTICNTR           (( PORT )0x07021)       /* RTI Counter reg */
#define WDTCNTR           (( PORT )0x07023)   /* WD Counter reg  */
#define WDTKEY            (( PORT )0x07025)       /* WD Key reg */
#define RTICR             (( PORT )0x07027)       /* RTI Control reg */
#define WDCR        (( PORT )0x07029)       /* WD Control reg */
#define CKCR0             (( PORT )0x0702B)       /* PLL control reg 1 */
#define CKCR1             (( PORT )0x0702D)         /* PLL control reg 2 */

/*-----------------------------------------------------------*/
/* Analog-to-Digital Converter(ADC) registers          */
/*-----------------------------------------------------------*/
#define ADCTRL1           (( PORT )0x07032)         /* ADC Control & Status
reg */
#define ADCTRL2           (( PORT )0x07034)       /* ADC Configuration reg
*/
#define ADCFIFO1    (( PORT )0x07036)         /* ADC Channel 1 Result Data
*/
#define ADCFIFO2          (( PORT )0x07038)         /* ADC Channel 2 Result
Data */

/*-----------------------------------------------------------*/
/* Serial Peripheral Interface (SPI) Registers         */
/*-----------------------------------------------------------*/
#define SPICCR            (( PORT )0x07040)       /* SPI Config Control Reg
*/
#define SPICTL            (( PORT )0x07041)       /* SPI Operation Control
Reg */
#define SPISTS            (( PORT )0x07042)       /* SPI Status Reg */
#define SPIBRR            (( PORT )0x07044)       /* SPI Baud rate control
reg */
#define SPIEMU            (( PORT )0x07046)       /* SPI Emulation buffer
reg */
#define SPIRXBUF    (( PORT )0x07047)       /* SPI Serial Input buffer reg
*/
#define SPITXBUF    (( PORT )0x07048)       /* SPI Serial output buffer reg
*/
#define SPIDAT            (( PORT )0x07049)       /* SPI Serial Data reg */
#define SPIPC1            (( PORT )0x0704D)       /* SPI Port control reg1
*/
#define SPIPC2            (( PORT )0x0704E)       /* SPI Port control reg2
*/
#define SPIPRI            (( PORT )0x0704F)       /* SPI Priority control
reg */

/*-----------------------------------------------------------*/
/* Serial Communications Interface (SCI) Registers         */
```

```
/*---------------------------------------------------------*/
#define SCICCR              (( PORT )0x07050)       /* SCI Comms Control Reg
*/
#define SCICTL1             (( PORT )0x07051)       /* SCI Control Reg 1 */
#define SCIHBAUD    (( PORT )0x07052)       /* SCI Baud rate control */
#define SCILBAUD    (( PORT )0x07053)       /* SCI Baud rate control */
#define SCICTL2             (( PORT )0x07054)       /* SCI Control Reg 2 */
#define SCIRXST             (( PORT )0x07055)       /* SCI Receive status reg
*/
#define SCIRXEMU    (( PORT )0x07056)       /* SCI EMU data buffer */
#define SCIRXBUF    (( PORT )0x07057)       /* SCI Receive data buffer */
#define SCITXBUF    (( PORT )0x07059)       /* SCI Transmit data buffer */
#define SCIPC1              (( PORT )0x0705D)       /* SCI Port control reg1
*/
#define SCIPC2              (( PORT )0x0705E)       /* SCI Port control reg2
*/
#define SCIPRI              (( PORT )0x0705F)       /* SCI Priority control
reg */

/*---------------------------------------------------------*/
/* Event Manager (EV) Registers                            */
/*---------------------------------------------------------*/
#define GPTCON              (( PORT )0x07400)         /* General Timer
Controls */
#define T1CNT               (( PORT )0x07401)       /* T1 Counter Register */
#define T1CMP               (( PORT )0x07402)       /* T1 Compare Register */
#define T1PER               (( PORT )0x07403)       /* T1 Period Register */
#define T1CON               (( PORT )0x07404)       /* T1 Control Register */
#define T2CNT               (( PORT )0x07405)       /* T2 Counter Register */
#define T2CMP               (( PORT )0x07406)       /* T2 Compare Register */
#define T2PER               (( PORT )0x07407)       /* T2 Period Register */
#define T2CON               (( PORT )0x07408)       /* T2 Control Register */
#define T3CNT               (( PORT )0x07409)       /* T3 Counter Register */
#define T3CMP               (( PORT )0x0740a)       /* T3 Compare Register */
#define T3PER               (( PORT )0x0740b)       /* T3 Period Register */
#define T3CON               (( PORT )0x0740c)       /* T3 Control Register */
#define COMCON              (( PORT )0x07411)       /* Compare Unit Control */
#define ACTR                (( PORT )0x07413)       /* Full Compare Unit
Output Action Control */
#define SACTR               (( PORT )0x07414)       /* Simple Comp Unit Output
Action Control */
#define DBTCON              (( PORT )0x07415)       /* Dead Band Timer Control
*/
#define CMPR1               (( PORT )0x07417)       /* Full Compare Channel 1
Threshold */
#define CMPR2               (( PORT )0x07418)       /* Full Compare Channel 2
Threshold */
#define CMPR3               (( PORT )0x07419)       /* Full Compare Channel 3
Threshold */
#define SCMPR1              (( PORT )0x0741a)       /* Simple Comp Channel 1
Threshold */
#define SCMPR2              (( PORT )0x0741b)       /* Simple Comp Channel 2
Threshold */
#define SCMPR3              (( PORT )0x0741c)       /* Simple Comp Channel 3
Threshold */
#define CAPCON              (( PORT )0x07420)       /* Capture Unit Control */
```

```
#define CAPFIFO             (( PORT )0x07422)       /* FIFO1-4 Status Register
*/
#define FIFO1               (( PORT )0x07423)       /* Capture Channel 1 FIFO
Top */
#define FIFO2               (( PORT )0x07424)       /* Capture Channel 2 FIFO
Top */
#define FIFO3               (( PORT )0x07425)       /* Capture Channel 3 FIFO
Top */
#define FIFO4               (( PORT )0x07426)       /* Capture Channel 4 FIFO
Top */
#define IMRA                (( PORT )0x0742c)       /* Group A Interrupt Mask
Register */
#define IMRB                (( PORT )0x0742d)       /* Group B Interrupt Mask
Register */
#define IMRC                (( PORT )0x0742e)       /* Group C Interrupt Mask
Register */
#define IFRA                (( PORT )0x0742f)       /* Group A Interrupt Flag
Register */
#define IFRB                (( PORT )0x07430)       /* Group B Interrupt Flag
Register */
#define IFRC                (( PORT )0x07431)       /* Group C Interrupt Flag
Register */
#define IVRA                (( PORT )0x07432)       /* Group A Int. Vector
Offset Register */
#define IVRB                (( PORT )0x07433)       /* Group B Int. Vector
Offset Register */
#define IVRC                (( PORT )0x07434)       /* Group C Int. Vector
Offset Register */

#endif
```

## F.2 Scaffold.h

```
/*********************************************************/
/*     Scaffold.h This header file lists the core    */
/*     function prototypes used in scaffold.c        */
/*     Created: 19-2-2     Gordon Webber             */
/*                                              */
/*********************************************************/

/*---------------------------------------------------------------*/
/*     PROTOTYPE DEFINITIONS                                     */
/*---------------------------------------------------------------*/

void main(void);
void ToggleLED(int);
void disable_interrupts(void);
void DACout(int);
void scheduler(void);
/*void initialise(void)*/
void init(void);
void enable_int(void);
void send_packet(void);
```

```
void proc_cmd (acycle_struct *acycle);
void sci_init (void);
/*int Runflag_test(acycle_struct *acycle)
/*void CCtest(acycle_struct, int);
void VCtest(acycle_struct);*/
```

## F.3 Scaffold.c

```
/*****************************************************/
/*       General Purpose Timer testprogram.c         */
/*         (Sample file for walkthrough )            */
/*****************************************************/

/*     Debugging check list, recheck with new cct:
1.     Timer Int     -      working
2.     PWM o/p               -      working
3.     Init_cycle    -      working
4.     Imeas         -      working
5.     Vmeas         -      working
6.     Icontroller   -      working
7.     serialDAQ     -      working
8.     VelMeas               -      working
9.     Imeas calib.  -      working
10.    Vmeas calib.  -      working
11.    Cmd Proc      -      working
12.    scheduler     -      working
13.    Pcontroller   -      working
14.    GradMeas      -
15.    Grad calib.   -
16.    WindMeas      -
17.    Wind calib.   -

*/

#include     "c243.h"
#include     "constant.h"
#include     "cycle.h"
#include     "scaffold.h"
#include     "input.h"

/*-----------------------------------------------*/
/*     Program timing                            */
/*-----------------------------------------------*/
int    count, slice, cvelcount, pcount;
/*-----------------------------------------------*/
/*     Task control                        */
/*-----------------------------------------------*/
int    Req_currentController;                 /* Flag to request current
Controller execution.*/
int    Exec_currentController;                /* Flag to execute current
Controller.*/
```

```
int    Req_CycleVelocityMeasurement;
int    Exec_CycleVelocityMeasurement;
int    Req_powerController;                    /* Flag to request current
Controller execution.*/
int    Exec_powerController;
int    Req_DAQ;
int    Exec_DAQ;

int    standing_req[7];          /* array of outstanding execution
requests.*/

/*     Misc.  */

int intcount, turn_on_flag;
acycle_structproto;

int count;
int t0;

int swfr;
int delay,i;

int LEDtick,LEDvalue;


interrupt void Test1(void)
       {
              /*     For high priority RX int. */
              if (*SCIRXST & 0x040){
                     proc_cmd(&proto);

                     }
       }

interrupt void GPT1_underflow(void)
       {

              /* Functions to be performed per slice:*/
              currentMeas(&proto);
              /*proc_cmd(&proto); /*    This may not work due to too long a
ISR period.   */
              /* What about processing commands here?? - as a polled
operation    */
              /* Increment counters.     */
              count++;                    /* increment count        */
              slice++;                    /* increment slice        */
              cvelcount++;        /* increment cvelcount           */
              pcount++;
              /*     Set task execution request flags and reset counters.*/
              if (count == 5){    /* request current controller execution. */
                     Req_currentController = 1;/*     ~2kHz  for 10kHz cpu
tick.*/
                     count = 0;                /* reset mvelcount*/
              }
```

```
            if (slice == DELAY_1S){   /* request DAQ execution. */
                  Req_DAQ = 1;              /*     ~1Hz   */
                  slice = 0;                /* reset mvelcount*/
            }
            if (cvelcount == 600){                  /* request cycle
velocity measurement. 25Hz*/
            /*if (cvelcount == 400){        /* For 10kHz cpu tick.     */
                  Req_CycleVelocityMeasurement = 1;
                  cvelcount = 0;                    /*     ~25Hz -> ~16.7
Hz     */
/*                LEDvalue = 0x0010;
                  ToggleLED(LEDvalue);*/
            }
            if (pcount == 20){  /* request current controller execution. */
                  Req_powerController = 1;  /*     ~500Hz */
                  pcount = 0;               /* reset mvelcount*/
            }

            /*LEDvalue = 0x0001;*/
/*          LEDvalue = 0x00ff;
            ToggleLED(LEDvalue);
            /*IFRA |= 0x0200;   /* Reset int flags - check that this covers
the relevant ones....     */
            *IFRA = 0x0f00;     /* Clear all T1 int.       */
      }

interrupt void Test3(void)/* T2 int ISR. - -works    */
      {
                  *IFRB = 0x0fff;
      }

interrupt void Test4(void)/* CAP ISR - works  -> No longer used.  */
      {
                  *IFRC = 0x0ff;
      }

interrupt void Test5(void)
      {
      }

interrupt void XINT2(void)
      {
      }

void ToggleLED(int LEDs)
      {
            /**OCRB = 0x00bc;*/
            *OCRB |= 0x01bc;
            if (LEDtick == 1) {
                  *PDDATDIR = 0xff00;
                  *PCDATDIR = 0xff00;
                  LEDtick = 0;
                  /*asm("      SETC   XF");*/
```

```
		}
		else if (LEDtick == 0){
			*PDDATDIR |= LEDs;
			*PCDATDIR |= LEDs;
			LEDtick = 1;
			/*asm("	CLRC	XF");*/
		}

	}

void main(void)
	{
		int out;

		init();
		init_cycle(&proto);
		sci_init();
		asm("  clrc INTM");
		/*CCtest(&proto, 12);*/
		/*switch_mux(0,3+8); /*	Ch1 = line2 = Imeas, Ch2 = line3 =
Vmeas. */
		/*switch_mux(0,2); /*	Ch1 = line0 = Imeas, Ch2 = line2 =
Vmeas. */
		/*switch_mux(0,1);*/
		/**CMPR1 = 0x15f;   /* Starting value;  */
	for(;;)
	{
		/*	PWM Test. - works   */
		/*	Current meas and controller tests.	*/
/*		CCtest(&proto, 12);	/* Current controller test.	*/
/*		currentController(&proto);
		/* serialDAQ test code. */
		/*	Packet sending test */
/*		prep(&proto);
		send_packet();
		/* Call scheduler.  */
		/*if (Runflag_test(&proto)){*/
			scheduler();
			/*}*/
	}

}

void init(void)
{
		/* Setup PWM */
		/**ACTR = 0x02;	/* PWM1 only. - active high	*/
		*ACTR = 0x02a;	/* Try active low.  */
		/**OCRA = 0x0ff;	/*0x0ffb;*/
		*OCRA = 0x0ffb;
		*DBTCON = 0x00;	/* No Deadband	*/

		/**COMCON = 0xCA00; /* Setup & load above.	*/
```

```
                *COMCON = 0x0200;
                *COMCON = 0x8200;   /* Ready to run.    */
/*              *CMPR1 = 0x00f;     /* Initial value.   */
                *CMPR1 = 0xffff;
                *PDDATDIR = 0xff00;
                *PADATDIR = 0xff00;
                *PCDATDIR = 0xff00;

                /* Setup interrupts.       */
                *IMR_REG = 0x03;    /* 0x02 for GPTint only, 0x0a to include
CAP. */
                                    /* 0x03 to include high priority RX int.
        */
                *IFR_REG = 0xFFFF;
                /*IMRA = 0x0101;    /* Works      */
                *IMRA = 0x0100;

                /* Setup timers     */
                /* Fro T2:    */
                /*IMRB = 0x004;     /* Attempt for T2 int.      */
                *IMRB = 0x000;      /* Not req.  */

                swfr = CPU_INT_PERIOD;
                /*swfr = 5024;*/
                *T1PER = swfr;  /* f = 15kHz */
                *T1CON = 0x01002;
                *T1CON = 0x01042;
                /*swfr = PWMPERIOD/4;*/
                *T2PER = swfr;
                *T2CON = 0x01002;
                *T2CON = 0x01042;
                *GPTCON = 0x024A;   /*0x0240;     /* T2 ADC start.    */


                /* Set up ADC.      */
                *ADCTRL1 = 0x1810;  /*0x0800;     /* Ch 0. */
                *ADCTRL2 = 0x0400;  /* EVSOC en. */

                /* Initialise variables. */
                LEDtick = 0;
                count = 0;
                slice = 0;
                cvelcount= 0;
                pcount = 0;
                turn_on_flag = 0;
                for (i = 0; i>7; i++){
                        standing_req[i] = 0;
                        }
                /*turn_on_flag = 1; /* Test on/off toggle later. */

                *IMRC = 0x0000;
}
```

```
/*void CCtest(acycle_struct *acycle, int ADres)
{
	/*acycle->iDes[0] = 512;  /*1024;*/
/*	acycle->iDes[0] = 330;	/* Range from 307 to 717. */

/*}*/

int Runflag_test(acycle_struct *acycle)
{
	if (acycle->turn_on_flag = 1){
		return 1;
	}
	else {
		return 0;
	}
}

/*	Scheduling routine. */
void scheduler()
{
/*while (1){
	/* Determine highest priority standing request, and prepare for
execution.	*/

	if (standing_req[0]>0){
		Exec_currentController = 1;	/* Execute current controller
*/
		standing_req[0]--;
	}
	else if (standing_req[1]>0){
		Exec_CycleVelocityMeasurement = 1;	/* Execute Cycle
Velocity Measurement */
		standing_req[1]--;
/*		LEDvalue = 0x0020;
		ToggleLED(LEDvalue);*/
	}
	if (standing_req[2]>0){
		Exec_powerController = 1; /* Execute power controller */
		standing_req[2]--;
	}
	else if (standing_req[3]>0){
		Exec_DAQ = 1;	/* Execute Command processor */
		standing_req[3]--;
	}
	/* Log current requests.  */
	if (Req_currentController == 1){
		Req_currentController = 0;/* Reset and add to outstanding. */
		standing_req[0]++;
	}
	if (Req_CycleVelocityMeasurement == 1){
		Req_CycleVelocityMeasurement = 0;/* Reset and add to
outstanding. */
		standing_req[1]++;
	}
```

```
        if (Req_powerController == 1){
                Req_powerController = 0;  /* Reset and add to outstanding. */
                standing_req[2]++;
        }
        if (Req_DAQ == 1){
                Req_DAQ = 0; /* Reset and add to outstanding. */
                standing_req[3]++;
        }

/*      Only one of the tasks listed below will be executed per loop.     */

        if (Exec_currentController == 1) {        /*  current control task
                */
                currentController(&proto);
                Exec_currentController = 0;
           }
        if (Exec_CycleVelocityMeasurement == 1) {     /*  Cycle Velocity
measurement task    */
                VelocityMeas(&proto);
/*              LEDvalue = 0x0040;
                ToggleLED(LEDvalue);*/
                Exec_CycleVelocityMeasurement = 0;
           }

        if (Exec_powerController == 1) {    /*  power control task        */
                powerController(&proto);
                Exec_powerController = 0;
/*              LEDvalue = 0x00ff;
                ToggleLED(LEDvalue);*/
           }
        if (Exec_DAQ == 1) {        /*  Perform DAQ operation.*/
                /*proc_cmd (&SRM);*/
                prep(&proto);
                send_packet();
                Exec_DAQ = 0;
           }

/*      } /* end infinite loop */
} /* end subroutine */
```

## F.4 SCI.C

```
/*      This routine dumps a data packet onto the serial port. > Taken from
SCI dump.c    */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++*/
/*                                                                        */
/*      File:               SCI.C                                         */
/*      Target Processor:   TMS320F243                                    */
/*      Compiler Version:   6.6                                           */
/*      Assembler Version:  1.96                                          */
/*      Created:            10/31/97                                      */
```

```
/*     Modified:           05/02/02                                    */
/*     Creator:            TI Staff                                    */
/*     Modifier:           Gordon Webber                               */
/*                                                                     */
/*     Revisions/modifications:                                        */
/*           1. Altered header, added comments indicating sections of code
*/
/*                 requiring changes. 05/02/02.
      */
/**/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++*/
/*     This file contains the routines for initializing and using the SCI.
      */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++*/

/*-----------------------------------------------------------------------
----*/
/*     INCLUDE FILES                                                   */
/*-----------------------------------------------------------------------
----*/
/*#include "srm.h"*/
#include "c243.h"
#include "constant.h"
#include     "cycle.h"
/*#include "globals.h"*/

extern int turn_on_flag;
/* Data array:      Only the LSB is transmitted.*/
int datatable[25] = {
      0x02,  /* STX */
      0x58,  /* Should be X, added in increase size of header.    */
      0x0050,       /* Packet number marker. */
      0x0001,       /* Packet number. Need to think about packet numbering
and data values.*/
      0x41,  /* A */
      0x42,  /* B */
      0x31,
      0x32,
      0x43,  /* C */
      0x44,  /* D */
      0x33,
      0x34,
      0x43,  /* C */
      0x41,  /* A */
      0x35,
      0x36,
      0x43,  /* C */
      0x42,  /* B */
      0x37,
      0x38,
      0x45,  /* E */
      0x46,  /* F */
```

```
        0x30,
        0x31,
        0x03}; /* ETX */
/*char *databyte;   /* pointer to datatable - byte size.*/
int *databyte;
/*int LEDtick,LEDvalue;*/
/*-----------------------------------------------*/
/*     Command processing variables              */
/*-----------------------------------------------*/
int    cmd_state,digit_cnt;
unsigned cmd_num;   /*     In attempt to unwrap commands.   */

/*-------------------------------------------------------------------------
----*/
/*     SCI PORT INITIALIZATION                                             */
/*-------------------------------------------------------------------------
----*/
/*     Initializes the SCI port for polled RS-232 communications at a    */
/*     speed of 19.2 baud, with 7 data bits, odd parity, and 1 stop bit.*/
/*-------------------------------------------------------------------------
----*/
void sci_init()
{

/*-----------------------------------------------------------------------*/
/*     Setup SCI for 7 data bits, 1 stop bit, odd parity, idle mode,     */
/*     .                                 */
/*-----------------------------------------------------------------------*/
  *SCICCR = 0x27; /* -->>> 8 bit!!      */

/*-----------------------------------------------------------------------*/
/*     Reset the SCI port, enable the internal clock, the enable  */
/*     the transmit and receive channels.                        */
/*-----------------------------------------------------------------------*/
 *SCICTL1=0x03;
 /**SCICTL2=0x00;*/
 *SCICTL2=0x02;     /* Set RX interrupt.      */

/*--------------------------------------*/
/*     Operate the SCI at 19,200 baud   */
/*--------------------------------------*/
*SCIHBAUD=0x00;
*SCILBAUD=0x81;

/*------------------------------------------------------*/
/*     Select the SCI txd and rxd pin functions      */
/*------------------------------------------------------*/
/**OCRA=*OCRA|0x03;*/
/**OCRA = 0xfb;*/   /*     Handled in init().  */

/*--------------------------------------*/
/*     Take the SCI out of reset */
/*--------------------------------------*/
```

```
 *SCICTL1=0x23;

 /*    Init all variables. */
 cmd_state = digit_cnt = cmd_num = 0;

}



/*-------------------------------------------------------------------*/
/*     SCI PUTCHAR ROUTINE                                           */
/*-------------------------------------------------------------------*/
/*     Wait for transmit ready to be asserted and place     */
/*     character in transmit buffer.                                 */
/*-------------------------------------------------------------------*/
void sci_put(char c)
{
while (!(*SCICTL2&0x80))
       ;
*SCITXBUF=c;
}

/*-------------------------------------------------------------*/
/*     SCI GETCHAR ROUTINE                                     */
/*-------------------------------------------------------------*/
/*     Return character if receive buffer non-empty; */
/*     otherwise, return a NULL.                 */
/*-------------------------------------------------------------*/
char sci_get()
{
char c;
if (*SCIRXST&0x40) {
       c=*SCIRXBUF; /* This assumes that the buffer is a single character. -
> which it is!!!! 8bit    */
       }
else {
       c=0;
       }
return c;
}

/*-------------------------------------------------------------------------------
------------*/
/*     PACKET SENDING PROCEDURE. - need to add packet prep.
       */
/*-------------------------------------------------------------------------------
------------*/

/*void main(void)*/
void send_packet(void)
{
       int i,k;
       int *start;
       /*sci_init();*/
```

```
	start = datatable;

	/**PBDATDIR = 0xff00;*/
	*PBDATDIR = 0xff10; /* Transmit enable. */
		for (i = 0; i<25; i++)	/* Limits the number of characters
transmitted. - Must be the same size as the data table!!*/
		{
			sci_put(datatable[i]);
		}
		*PBDATDIR = 0xff00; /* Transmit disable. */
		/*datatable[2]++;	/* Increment packet number.*/
		datatable[3]++;	/* Increment packet number. - position has
changed!!	*/
}


/*-------------------------------------------------------------------------
------------*/
/*	COMMAND PROCESSOR                                                    */
/*-------------------------------------------------------------------------
------------*/
/*	The command processor has the following five commands in its command
set:	*/
/*                                                                       */
/*	Command:	Description:                                         */
/*	-------		------------
	*/
/*	>t		Turn on command.  Begins motor startup procedure.
	*/
/*	>sxxxx		Sets a new target speed from 150 to 4500 RPM.
	*/
/*	>c		Cutoff the drive system.                             */
/*                                                                       */
/*	All of these commands must be followed by a carriage return.  Serial
commands	*/
/*	are only processed when the ramp controller is in its wait state.
Like the	*/
/*	ramp controller, the command processor is implemented as a software
state	*/
/*	machine.                                                             */
/*-------------------------------------------------------------------------
------------*/
void proc_cmd(acycle_struct *acycle)
{
/*unsigned char c;*/
unsigned int c;
/*unsigned int digit;*/

/*-------------------------------------------------------------*/
/*	Check if a command is in the receive buffer.  If not,*/
/*	exit routine; otherwise process it.                  */
/*-------------------------------------------------------------*/

/* This needs further work!!	*/
/*cmd_num = sci_get();*/
```

```
c = sci_get();
if (c>=0) {
	switch (cmd_state) {
		case STATE_LEADIN:
			/*-------------------------------*/
			/* Process leadin character      */
			/*-------------------------------*/
			if (c=='>') {
				cmd_state=STATE_CMD;
				}
			break;
		case STATE_CMD:
			/*-------------------------------*/
			/* Process command character     */
			/*-------------------------------*/
		/*	cmd=c;*/
			if (c=='s') {
				cmd_state=STATE_SPEED_NUM;
				digit_cnt = 3;	/*4; - 3 for power. */
				cmd_num=0;   /* chnaged to test cmd_num sending.
	*/
				}
			else if (c=='c') {
				cmd_state=STATE_CUTOFF_TERM;
				}
			else if (c=='t') {
				cmd_state=STATE_TURN_ON_TERM;
				}
			else {
				cmd_state=STATE_LEADIN;
				}
			break;
		case STATE_SPEED_NUM:
			/*-------------------------------*/
			/* Get 4 digits in target speed  -> 3 digits in target
power. */
			/*-------------------------------*/

			/* Power command sent in 10W increments to work around
wrapping problem -> 2 didgits in cmd.  */
			/*digit=(int)(c-'0');*/
/*			digit=(int)(c-48);
			if ((digit>=0) && (digit<=9)) {
				/*cmd_num = (10*cmd_num) + digit*10;*/
/*				cmd_num = 10*cmd_num + digit;
				digit_cnt--;
				if (digit_cnt <= 0) {
					cmd_state=STATE_LEADIN;
					/*acycle->power_des = cmd_num*10;/* Scale
up. */
					/*acycle->iDes[0] = cmd_num*10;*/
/*					acycle->iDes[0] = cmd_num;
					/*acycle->power_des = 100;/* Test
powerController.    */
```

```
/*                          }
                            acycle->iDes[0] = cmd_num;
                            }
                      else {
                            cmd_state=STATE_LEADIN;
                      }*/
                      cmd_num = (10*cmd_num) + (c - 48);/* + digit;*/
                      digit_cnt--;
               /*     acycle->iDes[0] = cmd_num;*/
                      /*acycle->iDes[0] = 650;  /*     To check Icontroller
function.     */
                      if (digit_cnt > 0){
                            cmd_state=STATE_SPEED_NUM;
                      }
                      else {
                            cmd_state=STATE_LEADIN;
                            /*acycle->iDes[0] = cmd_num;*/
                            acycle->power_des = cmd_num;
                      }
                      break;
               case STATE_CUTOFF_TERM:
                      /*----------------------*/
                      /* Cutoff the motor */
                      /*----------------------*/
/*                    asm(" SETC   INTM");*/
                      acycle->turn_on_flag = 0;
                      /*cutoff_motor();*/
                      cmd_state=STATE_LEADIN;
                      break;
               case STATE_TURN_ON_TERM:
                      /*-------------------------------------------------------
*/
                      /* Tell main() that a turn on command has been received
       */
                      /*-------------------------------------------------------
*/
                      acycle->turn_on_flag = 1;
                      cmd_state=STATE_LEADIN;
                      break;
               default:
                      cmd_state=STATE_LEADIN;
                      break;
               } /* end switch */
       } /* end if */
}


/*-------------------------------------------------------------------------
------------*/
/*     PACKET PREP                                                      */
/*-------------------------------------------------------------------------
------------*/

void prep(acycle_struct *acycle) /*     Packet Preparation. */
```

```
{
	/*	CD - Vel	*/
	datatable[10] = acycle->Vcycle_ave>>8 & 0x0ff;
	datatable[11] = acycle->Vcycle_ave;
/*	datatable[10] = cmd_num>>8 & 0x0ff;	/*	To allow debugging. */
/*	datatable[11] = cmd_num;*/
	/*datatable[10] = acycle->V_ave>>8 & 0x0ff;	/*	To allow
debugging.	*/
	/*datatable[11] = acycle->V_ave;*/
	/*	AB - Grad	*/
	datatable[6] = acycle->Gradient_ave>>8 & 0x0ff;
	datatable[7] = acycle->Gradient_ave;
/*	datatable[6] = acycle->dutyRatio[0]>>8 & 0x0ff;
	datatable[7] = acycle->dutyRatio[0];*/
	/*datatable[6] = acycle->power_des>>8 & 0x0ff;
	datatable[7] = acycle->power_des & 0x0ff;*/
	/*	CA - iFB	*/
	/*datatable[14] = acycle->Pcycle_ave>>8 & 0x0ff;
	datatable[15] = acycle->Pcycle_ave;*/
/*	datatable[14] = acycle->iDes[0]>>8 & 0x0ff;
	datatable[15] = acycle->iDes[0];*/
	datatable[14] = acycle->iFB[0]>>8 & 0x0ff;
	datatable[15] = acycle->iFB[0];
	/*	CB - Vmeas	*/
	datatable[18] = acycle->V_ave>>8 & 0x0ff;
	datatable[19] = acycle->V_ave;
	/*datatable[18] = acycle->iFB[0]>>8 & 0x0ff;
	datatable[19] = acycle->iFB[0];*/
	/*	EF - Wmeas	*/
	/*datatable[22] = acycle->V_ave>>8 & 0x0ff;
	datatable[23] = acycle->V_ave;*/
	datatable[22] = acycle->Wind_ave>>8 & 0x0ff;
	datatable[23] = acycle->Wind_ave;

}
```

## F.5 Cycle.h

```
/* This code is handles all the solar cycle related functions. Similar to
SRM.c  */
/* Includes the definition of the cycle data structure.	*/

typedef struct  {
	int	a2d_chan[NUMBER_OF_PHASES];
	long	integral_power_error;
	long	integral_current_error;
	unsigned int iDes[NUMBER_OF_PHASES];
/*	unsigned int imax;*/
	unsigned int iFB[NUMBER_OF_PHASES];
	unsigned int dutyRatio[NUMBER_OF_PHASES];
	unsigned int dutyMax;
	int	power_des;
	int	Active;
```

```
        char   turn_on_flag;
        /* Add solar cycle variables and states:*/
        int    Pcycle_prev;        /* Previous power reading,*/
                                   /* used to calc dp/dt to add the*/
                                   /* the average power.*/
        int    Pcycle_ave;         /* Average power.   */
        /*int  Gradient_prev;*/
        unsigned int Gradient_ave;       /* These values are measured and
passed to an IIR filter, o/p stored here. */
        unsigned int Vcycle_ave;         /*     "      */
        unsigned int V_ave;              /*     "      */
        unsigned int Wind_ave;           /*     "      */
        /*int  Tilt_prev;
        long   Tilt_ave;*/
/*      long   BattE;              /* V*I*dt added to est. battery energy
used.*/
/*      long   Capacity;           /* Battery energy capacity.*/
/*      long   Dist_trav;          /* Distance traveled. Increment with
v*dt.*/
/*      long   Dist_left;          /* Distance left to travel. Decrement with
v*dt.*/
} acycle_struct;


/*-------------------------------------------------------------------*/
/*     PROTOTYPE DEFINITIONS                                         */
/*-------------------------------------------------------------------*/
/* This list is incomplete.      */
void init_cycle(acycle_struct *acycle);
void velocityController(acycle_struct *acycle);
void currentController(acycle_struct *acycle);
/*long VelocityMeas(acycle_struct *acycle);*/
/*void VelocityMeas(acycle_struct *acycle, int);*/
/*void VelocityMeas(acycle_struct *acycle);
/*void GradientMeas(acycle_struct *acycle, int);
/*void switch_mux(int adc1, int adc2);*/
void switch_mux(int set);
void start_background();
void cutoff_motor ();
/*void currentMeas(acycle_struct *acycle); */

void sci_put (char c);
char sci_get ();
/*unsigned read_a2d(int);*/
```

## F.6 Cycle.c

```
/* This code is handles all the solar cycle related functions. Similar to
SRM.c  */
/* Includes the definition of the cycle data structure.    */


#include     "constant.h"
#include     "globals.h"
```

```
#include     "c243.h"
#include     "cycle.h"
#include     "math.h"

/*-----------------------------------------------------------------*/
/*     Cycle velocity measurement function.                        */
/*     Changed to analogue approach. Included a couple of          */
/*     other DAQ variables.                                        */
/*-----------------------------------------------------------------*/

/*void VelocityMeas(acycle_struct *acycle)
{
/*     int alphac;
       int alphag;*/
/*     unsigned Velmeas, Wmeas, Grad;
       Velmeas = 0;
       Wmeas = 0;    /*     Wmeas still to be implemented.   */
/*     Grad = 0;
/*     alphac= 0.3; /* IIR coeff. */    /* ===> Est. Coef!!!!!*/
/*     alphag = 0.9;/* IIR coeff. */    /* ===> Est. Coef!!!!!*/

       /* ADC alternative. */
/*     switch_mux(1);
       Grad = read_a2d(1);
       Velmeas = (*ADCFIFO2 >> 6) & 0x03ff;
/*     acycle->Vcycle_ave = alphac*acycle->Vcycle_ave + (1-alphac)*Velmeas;*/
/*     acycle->Vcycle_ave = Velmeas;
       /*acycle->Wind_ave = alphac*acycle->Wind_ave + (1-alphac)*Wmeas;*/
/*     acycle->Gradient_ave = alphag*acycle->Gradient_ave + (1-
alphag)*Grad;/**Gg;*/
/*     acycle->Gradient_ave = Grad;
}
*/

/*-----------------------------------------------------------------------
----*/
/*     POWER CONTROL LOOP ALGORITHM           -> Modify to PID          */
/*-----------------------------------------------------------------------
----*/
/*     The algorithm implements a PI compensator for the power          */
/*     control of the motor.  The PI filter limits the integrator       */
/*     to prevent windup.                                               */
/*-----------------------------------------------------------------------
----*/
void powerController(acycle_struct *acycle)
{
       int power_error, integral_error, active, ctrl_out, diff_error;
       int Ki,Kp,Kd;
       active = acycle->Active;

       /*     Filter coefficients:      */
       Ki = 0;/*0.95;            /*     Usually a bit better with ~0.95. */
       Kp = 1;             /*     Open loop gain needs to be large.*/
       Kd = 0;/*1;
```

```
	/*	Calculate error signal	*/
	power_error = acycle->power_des - acycle->Pcycle_ave;

	/*	Integrate error		*/
/*	acycle->integral_power_error = acycle->integral_power_error +
(long)power_error;
	/* Check integrator saturation limit	*/
/*	if (acycle->integral_power_error > INTEGRAL_LIMIT) {
		acycle->integral_power_error = INTEGRAL_LIMIT;
	}
	if (acycle->integral_power_error < -INTEGRAL_LIMIT) {
		acycle->integral_power_error = -INTEGRAL_LIMIT;
	}
	/*integral_error = (int) ((Ki*acycle->integral_power_error) >> 17);*/
  /*	integral_error = (int) ((Ki*acycle->integral_power_error) >> 16);

	/*	Differentiate error */
/*	diff_error = Kd*(power_error - acycle->Pcycle_prev);
/*	acycle->Pcycle_prev = power_error;

	/* PID filter*/
	ctrl_out = Kp*(power_error);/* + integral_error + diff_error);*/
	if (ctrl_out > I_MAX) {		/*	Constrain controller output.
	*/
		ctrl_out = I_MAX;
	}
	else if (ctrl_out < 0){
		ctrl_out = 0;
	}
	/*acycle->iDes[active] = ctrl_out;*/
	acycle->iDes[0] = ctrl_out;	/*	---> Temp. disabled to allow
tuning of Icontroller.	*/
}   /* end  powerController */


/*-------------------------------------------------------------*/
/*	Current measurement function - now battery measurement.	*/
/*-------------------------------------------------------------*/
/*void currentMeas(acycle_struct *acycle)
{
	int alphac,alphav;
	unsigned Imeas,phase;
	unsigned Velmeas, Vmeas, Wmeas, Grad;
	/*alphac= 0.5;	/*0.2; - IIR temp removed.*/
	/*alphav= 0.5;	/* IIR coeff. */
/*	alphac= 0.2; /* tuning */
/*	alphav= 0.75;/* tuning */
/*	phase = 0;
	Imeas = 0;
	Vmeas = 0;
	switch_mux(2);
	/*Vmeas = read_a2d(1);*/
```

```
/*     while (((*ADCTRL1)&0x80)==1);
       Vmeas = (*ADCFIFO1 >> 6) & 0x03ff;
       Imeas = (*ADCFIFO2 >> 6) & 0x03ff;
/*     acycle->V_ave = alphav*acycle->V_ave + (1-alphav)*Vmeas;
       acycle->iFB[0] = alphac*acycle->iFB[0] + (1-alphac)*Imeas;*/
/*     acycle->V_ave = Vmeas;
       acycle->iFB[0] = Imeas;

       /*     Measure other values here for debugging.      */
/*     switch_mux(1);      /*     This is needed to force compiler to
implement ADC conv above. */
             /*     If this line is absent, the ADC returns daft, 'floatng'
values.      */
/*     Grad = read_a2d(1);
/*     Velmeas = (*ADCFIFO2 >> 6) & 0x03ff;
       acycle->Vcycle_ave = Velmeas;
       acycle->Gradient_ave = Grad;/**Gg;*/
/*
}
*/
/*-------------------------------------------------------------------------
----*/
/*     CURRENT CONTROL LOOP ALGORITHM   -> Modify                          */
/*-------------------------------------------------------------------------
----*/
/*     Employs a proportional control law to realize the current requests
that */
/*     arrive from the power loop. Added I term.                           */
/*-------------------------------------------------------------------------
----*/
void currentController(acycle_struct *acycle) {

       /*int phase;*/
       int ierr, integral_error, ctrl_out, diff_error, pave;
       int Ki,Kp,Kd;
       /*     Filter coefficients:      */
       Ki = 0;/*.95;       /*     Usually a bit better with ~0.95. */
       /*     Winds up too quickly.     */
       Kp = 1;             /*     Open loop gain needs to be large.*/
       /*phase = 0; */

       /* Calculate error signal              */
       ierr = acycle->iDes[0] - acycle->iFB[0];
       /*     Place limit on size on error.    */
       if (ierr > PWMPERIOD) {
             ierr = PWMPERIOD;
       }

       /*     Integrate error           */
  /*   acycle->integral_current_error = acycle->integral_current_error +
(long)ierr;
       /* Check integrator saturation limit   */
/*     if (acycle->integral_current_error > INTEGRAL_LIMIT) {
             acycle->integral_current_error = INTEGRAL_LIMIT;
```

```
	}
	if (acycle->integral_current_error < -INTEGRAL_LIMIT) {
		acycle->integral_current_error = -INTEGRAL_LIMIT;
	}
	/*integral_error = (int) ((Ki*acycle->integral_current_error) >>
17);*/
  /*	integral_error = (int)((Ki*acycle->integral_current_error) >> 16);


	/* PI filter */
	ctrl_out = Kp*(ierr);/* + integral_error);/* + diff_error);*/

	/*	Output limit.*/
	if (ctrl_out > PWMPERIOD){
		ctrl_out = PWMPERIOD;
	}
	else if (ctrl_out < 0) {
		ctrl_out = 0;
	}
	/*	Constrained Output signal */
	acycle->dutyRatio[0] = PWMPERIOD - ctrl_out;  /*	Limited by PWM
period ~1332 cycles.	*/
	*CMPR1 = acycle->dutyRatio[0];   /* As inverse is true.	*/

	/*	Calculate average power for power controller - done here for
better resolution.  */
	/*acycle->Pcycle_ave = (int)((acycle->iFB[phase]*acycle-
>V_ave)/IV_SCALE);*/

	/*acycle->Pcycle_ave = (int)((acycle->iFB[0]*acycle-
>V_ave)/IV_SCALE);*/
/*	pave = (int)((acycle->iFB[0])*(acycle->V_ave)/IV_SCALE);
	/* Maybe: */
	pave = (int)(acycle->iFB[0]/I_SCALE*acycle->V_ave/V_SCALE);/* works!!
*/
	if (pave > 1000){
		acycle->Pcycle_ave = 1000;
	}
	else if (pave < 0){
		acycle->Pcycle_ave = 0;
	}
	else {
		acycle->Pcycle_ave = pave;
	}
	/*	Above works!!*/

/*	LEDvalue = 0x00ff;
	ToggleLED(LEDvalue);*/

		/*	Should work if comp. load not too heavy.	*/

} /* end  currentController */
```

```
/* ADC routines.    */

/*-------------------------------------------------------------------------*/
/*     SWITCH A/D INPUT CHANNEL                                            */
/*-------------------------------------------------------------------------*/
/*     Each A/D converter unit has an 8:1 input multiplexer which */
/*     must be selected to the desired channel, prior to          */
/*     sampling.  The channel is selected by manipulating bits          */
/*     of the ADCTRL1 control register.                           */
/*                                                                */
/*     inputs:       adc1 = desired input channel for A/D #1      */
/*                          range: 0-7                            */
/*                   adc2 = desired input channel for A/D #2      */
/*                          range:  8-15                          */
/*     outputs:      none                                         */
/*                                                                */
/*-------------------------------------------------------------------------*/
/*void switch_mux(int adc1, int adc2)*/
void switch_mux(int set)  /*     Not switching properly - force to toggle
between sets.*/
{

      /*unsigned int ctrl_word;*/
/*    unsigned ctrl_word;

/*    ctrl_word = 0x2c00; /* WRONG! */              /* mask channel select
bits */
/*    ctrl_word = 0x1800;
      ctrl_word = ctrl_word | (adc1 << 4);          /* set ADC1 channel
bits */
/*    ctrl_word = ctrl_word | ((adc2-8) << 1);      /* set ADC2 channel
bits */
/*    *ADCTRL1 = ctrl_word;
      /**ADCTRL2 = 0x0400;       /*     For EVM start conv. */
      if (set == 1){
            *ADCTRL1 = 0x3802;  /*0x3802;    /* line0,line1      */
            /*    0x1802;*/
      }
      else if (set == 2){
            *ADCTRL1 = 0x3826;  /*0x3826;    /*     line2,line3  */
                  /*     0x1826;*/
      }
      else if (set == 3){
            *ADCTRL1 = 0x3858;  /*0x3858;    /*     line5,line4  */
                  /*     0x1826;*/
      }
      else {              /*     Default - no conversion.  */
            *ADCTRL1 = 0x1802;
            }

      *ADCTRL2 = 0x000;


}
```

```
/*-------------------------------------------------------------------------*/
/*     READ A/D FIFO REGISTER                                              */
/*-------------------------------------------------------------------------*/
/* This routine is used to read the sampled A/D data from the              */
/*     appropriate FIFO.  The 10-bit A/D data is stored in the             */
/*     FIFO in bits 15-6.  A right shift of 6, limits the data             */
/*     to the range 0-1023.                                                */
/*                                                                         */
/*     inputs:             a2d_chan = which FIFO to read                   */
/*                         range: 1-2                                      */
/*     outputs:      inval = A/D data                                      */
/*                         range: 0-1023                                   */
/*                               0 VDC = 0 bits                            */
/*                               5 VDC = 1023 bits                         */
/*     Changed to return both FIFO buffer values, to speed the code        */
/*     and simplify writing.                                               */
/*-------------------------------------------------------------------------*/
/*unsigned read_a2d(int a2d_chan)
/*unsigned read_a2d(void)*/
/*{
      unsigned inval;

      /* Wait for EOC flag to clear - should have cleared by the time this
is called.   */
/*    while (((*ADCTRL1)&0x80)==1);
      if (a2d_chan == 1) {
         inval = (*ADCFIFO1 >> 6) & 0x03ff;
      }
      else if (a2d_chan == 2) {
         inval = (*ADCFIFO2 >> 6) & 0x03ff;
      }
      return inval;
}*/

/*-------------------------------------------------------------------------
----*/
/*     CYCLE ALGORITHM INITIALIZATION   -> Modify, including the struct. */
/*-------------------------------------------------------------------------
----*/
/*     Initializes the variables in the master cycle data structure.
      */
/*-------------------------------------------------------------------------
----*/
void init_cycle(acycle_struct *acycle)
{
   int i;

   /* Define mux positions for current feedback of each phase     */
   acycle->a2d_chan[0] = 0;      /* phase A on pin ADCIN2 */
   acycle->a2d_chan[1] = 1;      /* phase B on pin ADCIN3 */
   acycle->a2d_chan[2] = 2;      /* phase C on pin ADCIN4 */
```

```
   /* Specify initial conditions */
   for (i=0; i < NUMBER_OF_PHASES; i++) {
       acycle->iDes[i] = 0;
       acycle->iFB[i] = 0;
       }

   /* Initialize the power loop  */
   acycle->power_des = 0;
   acycle->integral_power_error = 0;/*INTEGRAL_INIT;       /* windup
integrator to start under load */
   acycle->Vcycle_ave = 0;
       acycle->Pcycle_ave = 0;

   /* Initialize the other variables    */
   acycle->Gradient_ave = 0;
   acycle->V_ave = 0;
   acycle->Wind_ave = 0;
}

/*-------------------------------------------------------------------------
----*/
/*     MOTOR CUTOFF ROUTINE                                                */
/*-------------------------------------------------------------------------
----*/
/*     Sets all the compare registers to 0, turns off the lowside
transistors, */
/*     zeroes out the desired current, turns on all LEDs to indicate
shutdown, */
/*     and spinlocks the processor.                                        */
/*-------------------------------------------------------------------------
----*/

void cutoff_motor (acycle_struct *acycle)
{

/* Set PWM duty cycles to zero */
*CMPR1=PWMPERIOD;          /* Inverse true!!!  */
*CMPR2=0x0;
*CMPR3=0x0;

/* Zero out the desired current */
acycle->iDes[0]=0;
acycle->iDes[1]=0;
acycle->iDes[2]=0;

/* Zero the desired power */
acycle->power_des = 0;

/* Spinlock */
/*while (1){}/* Add a restart routine, reset in meantime.*/
/*while (!turn_on_flag){};
/*     ;*/
}
```

## F.7 Input.h

```
/*-------------------------------------------------------------------*/
/*     PROTOTYPE DEFINITIONS                                         */
/*-------------------------------------------------------------------*/
/* This list is incomplete.      */

void VelocityMeas(acycle_struct *acycle);
void currentMeas(acycle_struct *acycle);
unsigned read_a2d(int);
```

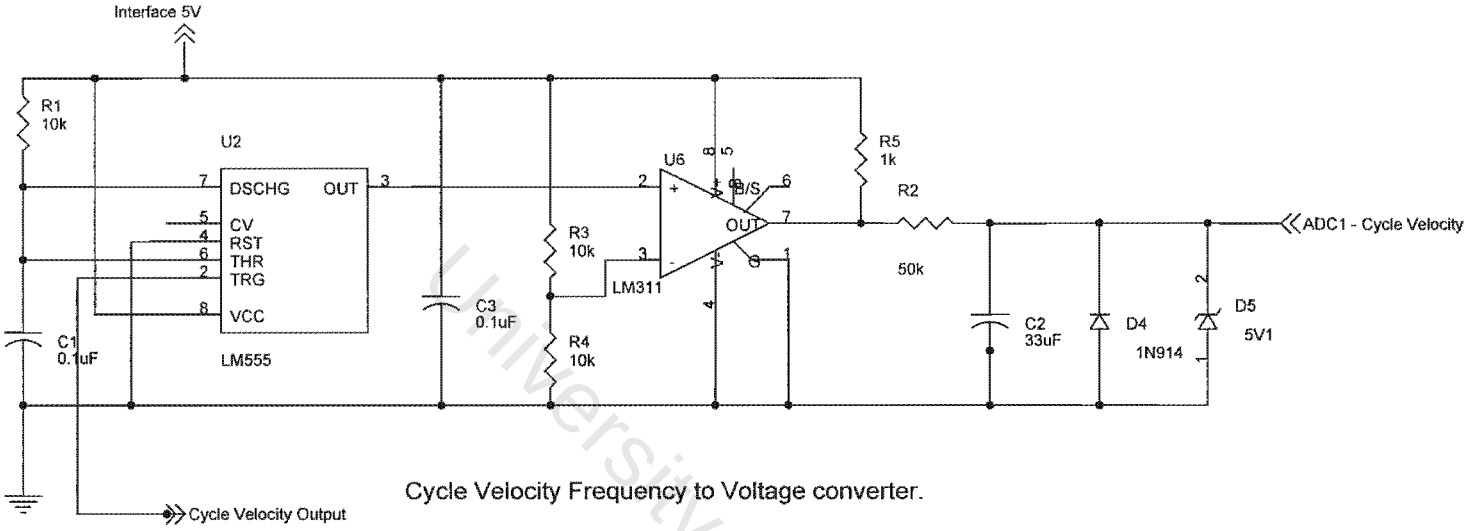
## F.8 Input.c

```
/*********************************************************/
/*     This file handles the ADC inputs.       */
/*     Based on old code in cycle.c.                 */
/*     Placed here in attempt to improve optimisation.      */
/*********************************************************/

#include     "constant.h"
#include     "globals.h"
#include     "c243.h"

#include     "cycle.h"
#include     "input.h"

/*-------------------------------------------------------------------*/
/*     Cycle velocity measurement function.                          */
/*     Changed to analogue approach. Included a couple of   */
/*     other DAQ variables.                                          */
/*-------------------------------------------------------------------*/

void VelocityMeas(acycle_struct *acycle)
{

       unsigned Velmeas, Wmeas, Grad, dummy;
       Velmeas = 0;
       Wmeas = 0;   /*     Wmeas still to be implemented - done.   */
       Grad = 0;
       dummy = 0;

       /* ADC alternative. */
       /*switch_mux(1);*/
       *ADCTRL1 = 0x3802;  /* line0,line1      */
       *ADCTRL2 = 0x000;
       Grad = read_a2d(1);
       /*Velmeas = read_a2d(2);  */
       /*while (((*ADCTRL1)&0x80)==1);
       Grad = (*ADCFIFO1 >> 6) & 0x03ff;*/
       Velmeas = (*ADCFIFO2 >> 6) & 0x03ff;
       /* For Wmeas:*/
       /*switch_mux(3);*/
       *ADCTRL1 = 0x3858;  /*     line5,line4   */
```

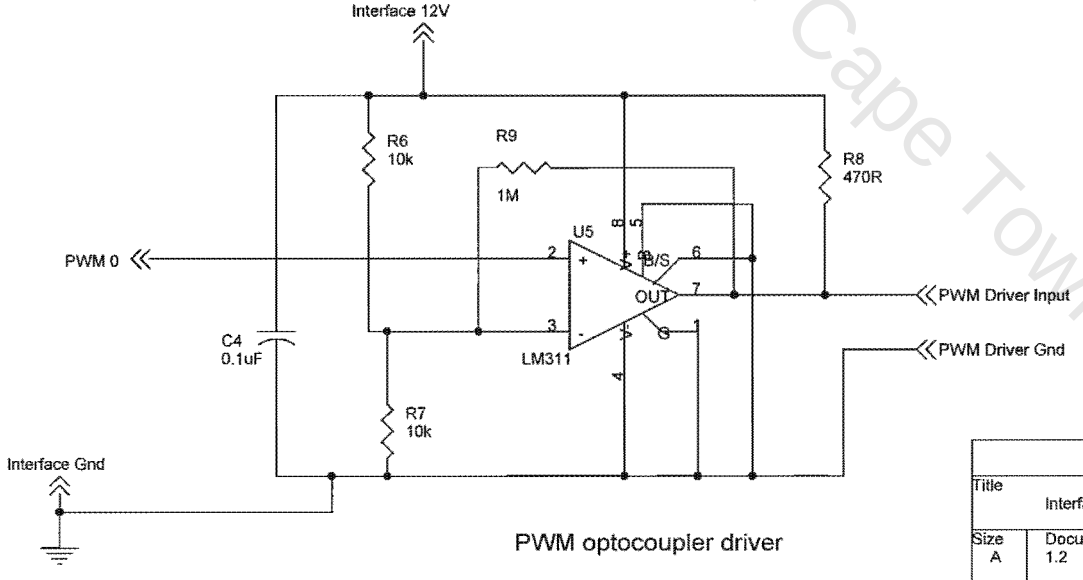
```
        *ADCTRL2 = 0x000;
        Wmeas = read_a2d(1);
        dummy = read_a2d(2);
        /*while (((*ADCTRL1)&0x80)==1);
        Wmeas = (*ADCFIFO1 >> 6) & 0x03ff;
        dummy = (*ADCFIFO2 >> 6) & 0x03ff;*/
        /* Update record.   */
        acycle->Vcycle_ave = Velmeas;
        acycle->Gradient_ave = Grad;
        acycle->Wind_ave = Wmeas;
        switch_mux(0);
        /**ADCTRL1 = 0x3802;      /* line0,line1      */
        /**ADCTRL2 = 0x000;*/
}

/*-------------------------------------------------------------------*/
/*     Current measurement function - now battery measurement.    */
/*-------------------------------------------------------------------*/
void currentMeas(acycle_struct *acycle)
{
        int alphac,alphav;
        unsigned Imeas,phase;
        unsigned Velmeas, Vmeas, Wmeas, Grad;

        phase = 0;
        Imeas = 0;
        Vmeas = 0;
        switch_mux(2);
        /**ADCTRL1 = 0x3826;      /*     line2,line3  */
        /**ADCTRL2 = 0x000;*/

        /*while (((*ADCTRL1)&0x80)==1);
        Vmeas = (*ADCFIFO1 >> 6) & 0x03ff;
        Imeas = (*ADCFIFO2 >> 6) & 0x03ff;*/
        Vmeas = read_a2d(1);
        Imeas = read_a2d(2);

        acycle->V_ave = Vmeas;
        acycle->iFB[0] = Imeas;

        /*     Measure other values here for debugging.      */
        switch_mux(0);      /*     This is needed to force compiler to
implement ADC conv above. */
                /*     If this line is absent, the ADC returns daft, 'floatng'
values.      */
        /**ADCTRL1 = 0x3802;      /* line0,line1      */
        /**ADCTRL2 = 0x000;*/


}

/*-------------------------------------------------------------------------*/
/*     READ A/D FIFO REGISTER                                              */
```

```
/*-------------------------------------------------------------------------*/
/* This routine is used to read the sampled A/D data from the              */
/*     appropriate FIFO.  The 10-bit A/D data is stored in the             */
/*     FIFO in bits 15-6.  A right shift of 6, limits the data             */
/*     to the range 0-1023.                                          */
/*                                                                 */
/*     inputs:             a2d_chan = which FIFO to read               */
/*                         range: 1-2                              */
/*     outputs:      inval = A/D data                              */
/*                         range: 0-1023                           */
/*                               0 VDC = 0 bits                    */
/*                               5 VDC = 1023 bits                 */
/*     Changed to return both FIFO buffer values, to speed the code    */
/*     and simplify writing.                                         */
/*-------------------------------------------------------------------------*/
unsigned read_a2d(int a2d_chan)
/*unsigned read_a2d(void)*/
{
      unsigned inval;

      /* Wait for EOC flag to clear - should have cleared by the time this
is called.   */
      while (((*ADCTRL1)&0x80)==1);
      if (a2d_chan == 1) {
         inval = (*ADCFIFO1 >> 6) & 0x03ff;
      }
      else if (a2d_chan == 2) {
         inval = (*ADCFIFO2 >> 6) & 0x03ff;
      }
      return inval;
}
```

# Appendix G: Electronic Circuit Diagrams

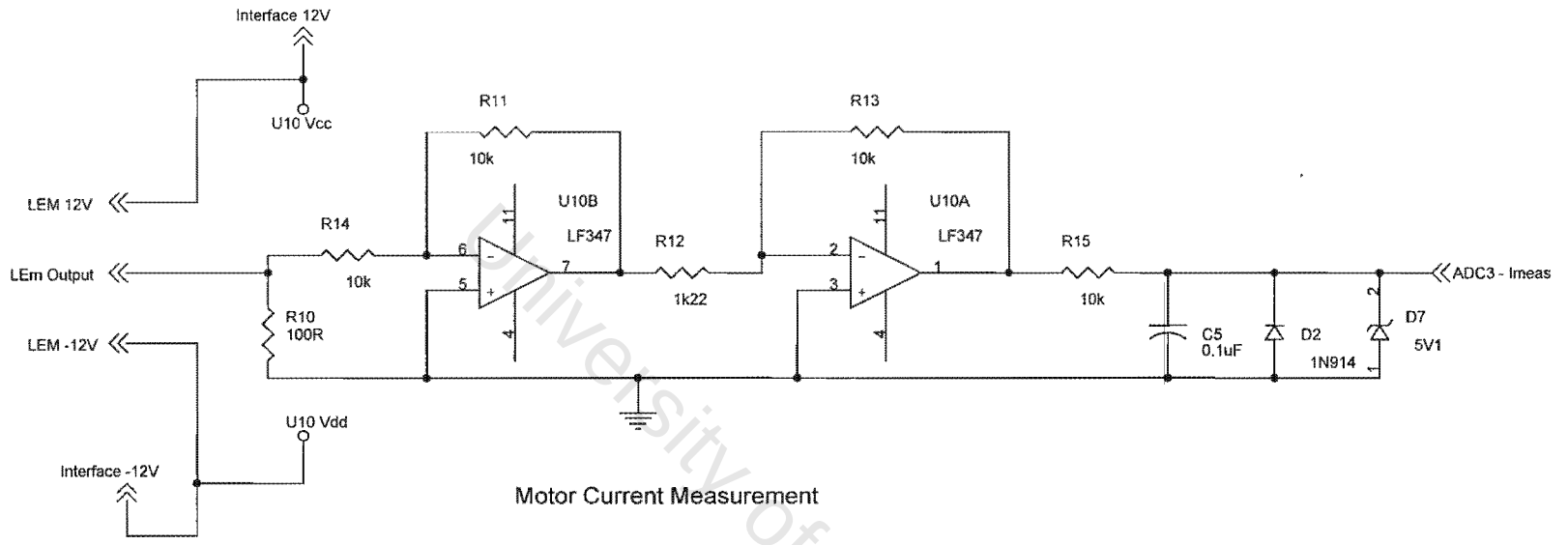



Cycle Velocity Frequency to Voltage converter.

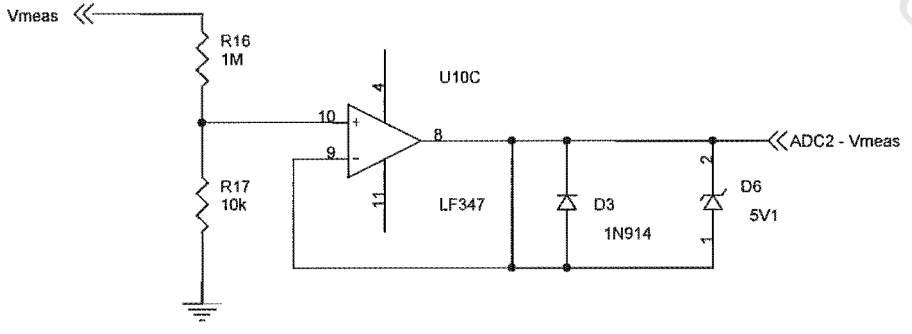



PWM optocoupler driver

| | | |
|---|---|---|
| Title: Interface Board, Part A | | |
| Size: A | Document Number: 1.2 | Rev: 1.0 |
| Date: Thursday, August 22, 2002 | Sheet 1 of 1 | |

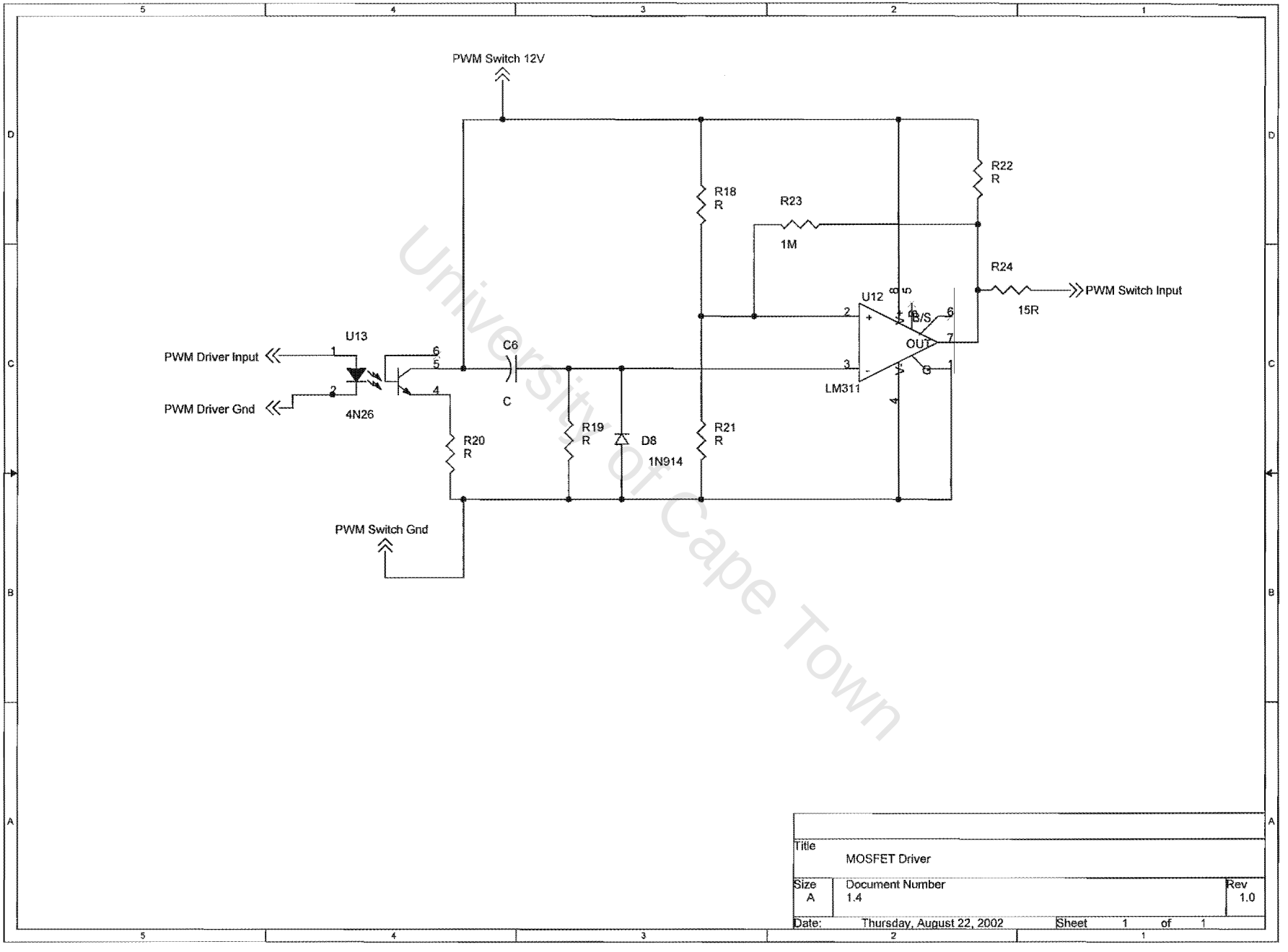


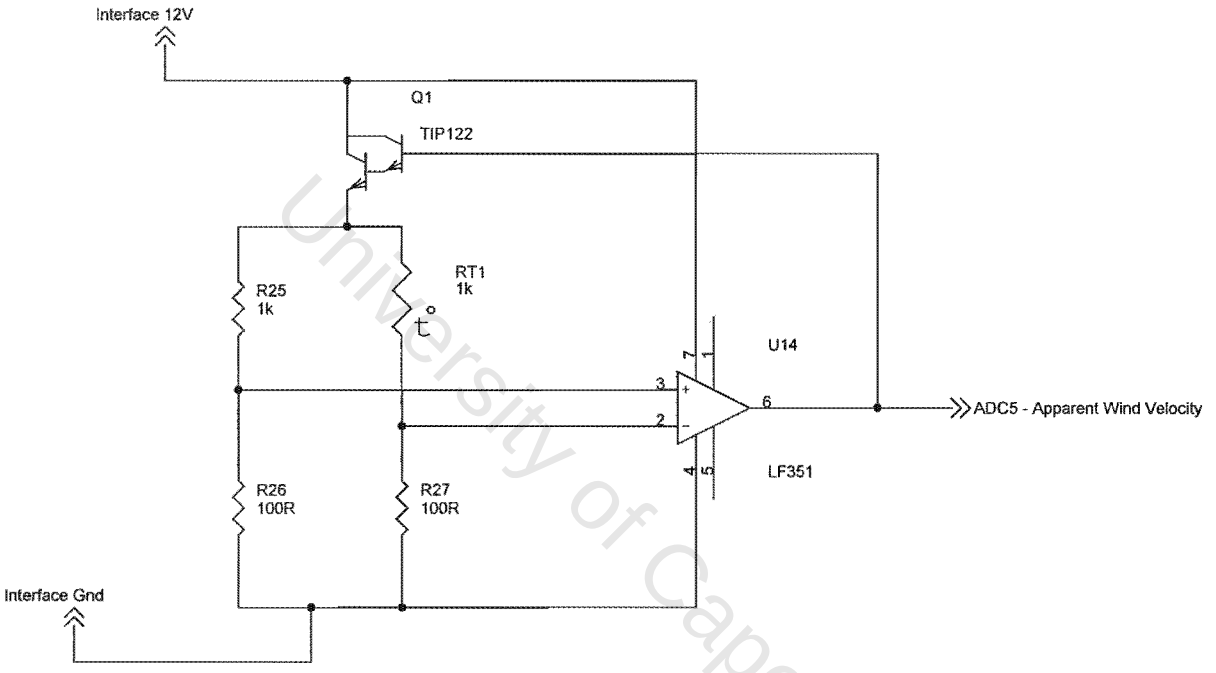

Motor Current Measurement




Battery Voltage Measurement

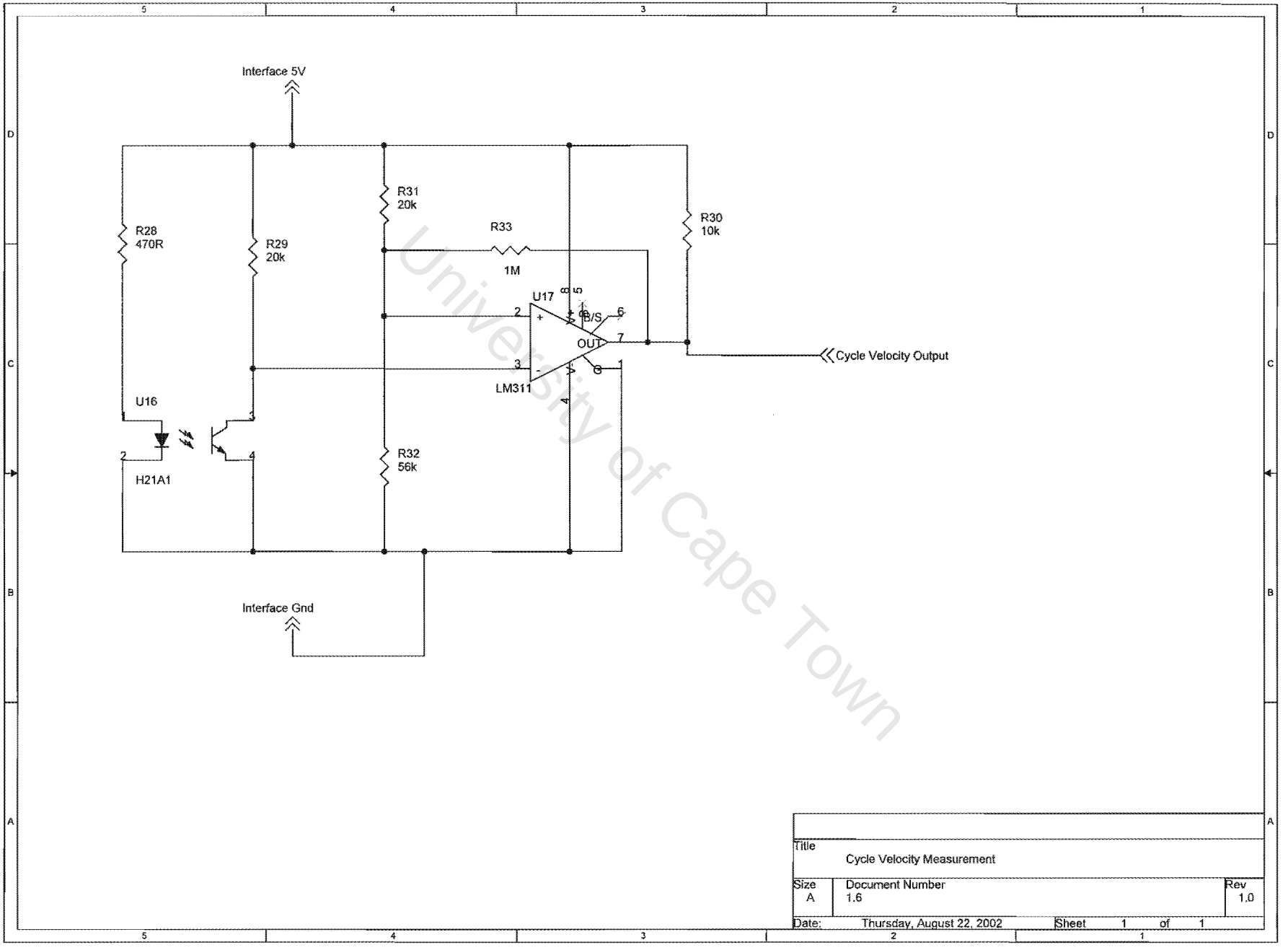
| | | |
|---|---|---|
| Title: Interface Board Part B | | |
| Size: A | Document Number: 1.3 | Rev: 1.0 |
| Date: Thursday, August 22, 2002 | Sheet 1 of 1 | |








| | | |
|---|---|---|
| Title: Anemometer | | |
| Size: A | Document Number: 1.5 | Rev: 1.0 |
| Date: Thursday, August 22, 2002 | Sheet 1 of 1 | |






| | | |
|---|---|---|
| Title: Cycle Velocity Measurement | | |
| Size: A | Document Number: 1.6 | Rev: 1.0 |
| Date: Thursday, August 22, 2002 | Sheet 1 of 1 | |