# Design and Control of an Animatronic Aardvark

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHNICAL ENGINEERING
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE



By

**Stefan Hrabar**

January 2002

Supervised by

**Prof A. Sass**

I, Stefan Ellis De Nagy Koves Hrabar, hereby declare that the work presented in this thesis is my own.

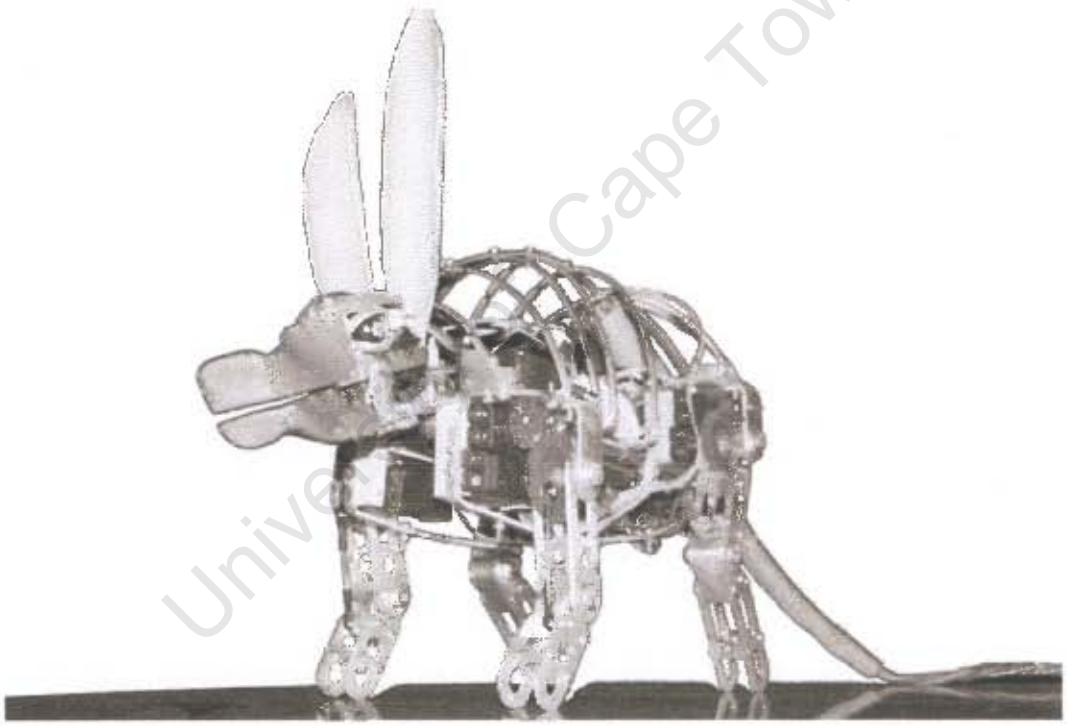
## Acknowledgements

I would like to thank Professor Andrew Sass for his guidance and support throughout the period of completing this thesis. His enthusiasm in my work was always an inspiration and I am greatly indebted to him for this. I would also like to thank Lynne and Phillip Richardson for involving me in the aardvark project, and for their support and motivation. To my parents and fiancé Catherine, for the interest they always showed in this project and the positive feedback I always received from them.

# Abstract

This report describes the design, construction and programming of an animatronic Aardvark that was built and successfully used in the filming of a wildlife documentary for National Geographic.

The animatron was required to walk, move its head, and have as many facial movements as possible. These requirements were met by using hobby servos to produce the movements, and control was achieved with a Motorola based micro controller (the Handy Board). The proportions of the animatron were based on those of a real aardvark, made to approximately ¼ scale. The final product met all the requirements and was filmed on location interacting with a real aardvark.



# Contents


| | Page |
|---|---|
| Abstract | III |
| List of illustrations | VI |
| 1 Introduction | 1 |
| 2 Discussion of final solution | 2 |
| 2.1 Overall Design Proportions | 2 |
| 2.2 Leg Design | 3 |
| 2.3 Head Design | 4 |
| 2.4 Control | 5 |
| 2.4.1 Handy Board | 5 |
| 2.4.2 Mini Serial Servo Commander | 7 |
| 2.4.3 Control Box | 7 |
| 2.4.4 Infra Red Control | 8 |
| 2.5 Power Supply | 8 |
| 2.6 Overall Layout | 9 |
| 2.7 Walking | 10 |
| 2.7.1 Programming the Walking Sequence | 10 |
| 3 Conceptual Design | 11 |
| 3.1 Selection of Microcontroller | 11 |
| 3.1.1 Handy Board | 11 |
| 3.1.2 PIC 16F84 Board | 12 |
| 3.1.3 OOPic Board | 13 |
| 3.1.4 Conclusion on Microcontroller Selection | 14 |
| 3.2 Development of Leg design | 14 |
| 3.2.1 Selection of Motor Type | 14 |
| 3.2.2 Geometry and Construction of Legs | 15 |
| 3.3 Development of Head and Facial Features | 23 |
| 3.3.1 Head Movement | 23 |
| 3.3.2 Development of the Eyes | 25 |
| 3.2.2.1 Eye Movements | 25 |
| 3.2.2.2 Construction of the Eyelids | 27 |
| 3.4 Development of the Mouth Mechanism | 28 |
| 3.5 Weight Shifting Mechanism | 30 |
| 3.6 Description of Infra Red Control | 31 |

3.7 Construction of the Control Box 32
3.7.1 Head and Facial Movements 33
3.7.2 Push Button Controls 35
3.7.3 Power Supply Interface 36
3.7.4 Handy Board Programming Interface 37
3.8 Walking 38
3.8.1 Dynamic vs. Static stability 38
3.8.2 Statically stable gate for a quadruped 39
3.8.3 Using the Weight Shifting Mechanism to assist the walking 40
3.8.4 Other uses of the weight shifting mechanism 40

4 The Filming of Ardie 41

5 Conclusions 45

6 Future Work 45

References 46

Appendix A: Author's background 47

Appendix B: Communication between Handy Board and SSC 48

Appendix C: Extract from Handy Board datasheet 49

Appendix D Mini Serial Servo Commander datasheet 52

Appendix E: Drawings for leg construction 54

Appendix F: Program listing for Handy Board library file. 63

Appendix G: Program listing for Expansion Board routines 69

Appendix H: Listing of Aardvark Control Program 70

Appendix I: Listing of program for controlling the SSC via the Handy Board 82

Appendix J: Code Listing for Application to Program the Walking Sequence 84

Appendix K: Calculation of Overall Dimensions 87

# List of Illustrations


| | | |
|---|---|---|
| Figure 1: | Photo Of Aardvark Used To Measure Proportions | 2 |
| Figure 2 | Side View Of The Front And Rear Legs Of The Animatron | 3 |
| Figure 3 | Oblique View Of Ardie's Head | 3 |
| Figure 4 | An Assembled Handy Board (Excluding Expansion Board) | 5 |
| Figure 5 | Diagram Of The Handy Board's Components | 6 |
| Figure 6 | Photo Of Handy Board Expansion Board | 6 |
| Figure 7 | Hands In Position On The Control Box | 7 |
| Figure 8 | Photo Of 10Ah Battery Pack | 8 |
| Figure 9 | Schematic Diagram Of Overall Layout Of Control Components And Actuators | 9 |
| Figure 10 | Screen Shot Of Program For Programming The Walking Sequence | 10 |
| Figure 11 | Photo Of PIC 16F84 Based Board | 12 |
| Figure 12 | Photo Of OOpic Board | 13 |
| Figure 13a | Photo Of Small Stepper Motor | 14 |
| Figure 13b | Photo Of Hobby Servo | 14 |
| Figure 14 | Illustration Of Motors Driving Leg Joints Directly | 16 |
| Figure 15a | Photo Of Leg Prototype Using Pulleys | 16 |
| Figure 15b | Photo Of Leg Prototype Using Pulleys | 16 |
| Figure 16a | Photo Of The Second Leg Pulley Prototype | 17 |
| Figure 16b | Photo Of The Second Leg Pulley Prototype | 17 |
| Figure 17 | Photo Of Leg Prototype Using Small Toothed Gears | 17 |
| Figure 18 | Illustration Of How Lower Joint Angle Changes As Hip Joint Angle Changes. | 18 |
| Figure 19a | Photo Of Leg Prototype With Large Toothed Gears | 19 |
| Figure 19b | Photo Of Final Leg Design With Large Toothed Gears | 19 |
| Figure 20 | Photo Of First Prototype With 4 Longer Legs Mounted On Balsawood. | 19 |
| Figure 21 | Photo Of Prototype With Shorter Legs | 20 |
| Figure 22 | Photo Of The CNC Machined Aluminium Leg Plates | 21 |
| Figure 23 | Rendered Knee Joint And Photo Of The Real Knee Joint | 21 |
| Figure 24 | Partly Assembled Knee Joint | 21 |
| Figure 25 | Close Up Photo Of Ball And Socket Joint | 22 |
| Figure 26 | Rendered View Of Leg And Photo Of Real Leg | 22 |
| Figure 27 | Photo Showing Neck Hinge For Head Pitch Movement | 23 |
| Figure 28 | Photo Showing Servo For Head Yaw Motion Attached To The Neck Hinge Plate | 24 |
| Figure 29 | Bottom View Of Head Showing Yaw Motion Servo In The Centre | 24 |
| Figure 30 | Photo Showing Swivel Bush Plate Attached To Add Support For The Head | 24 |
| Figure 31 | Photo Of Swivel Plate For Head Support. | 24 |
| Figure 32 | Illustration Of How Eyes Would Be Mounted In Relation To The Head | 26 |
| Figure 33 | Illustration Of Eyelid Movement Mechanism | 26 |
| Figure 34 | Photo Of Prototype Perspex Eyelids | 27 |
| Figure 35a | Photo Of Aluminium Eyelids | 28 |
| Figure 35b | Photo Of Eyeball And Mounting Screw | 28 |
| Figure 36 | Photo Of Helical Cam For Mouth Movement Mechanism | 28 |
| Figure 37 | Photo Of Lower Jaw | 29 |
| Figure 38 | Photo Of Jaw Hinge | 29 |
| Figure 39 | Photos Of Mouth In The Open And Closed Positions | 29 |
| Figure 40a | Photo Of Weight Shifting Mechanism With Weight Shifted Back | 30 |
| Figure 40b | Photo Of Weight Shifting Mechanism With Weight Shifted Forward | 30 |
| Figure 41 | Illustration Of Infra Red Control System | 31 |
| Figure 42 | Top View Of Control Box | 32 |
| Figure 43 | Front View Of Control Box | 32 |
| Figure 44 | Photo Of Inside Of Control Box | 33 |
| Figure 45 | Potentiometer Used To Control The Upper Eyelids | 34 |

| | | |
|---|---|---|
| Figure 46 | Hands In Position On The Control Box – Right Thumb On The Mouth Movement Lever | 34 |
| Figure 47 | Forefingers In Position On The Eyelid Control Levers | 35 |
| Figure 48 | Photo Of Handy Board Programming/Charging Unit | 36 |
| Figure 49 | Photo Of 6v Lead Acid Battery | 36 |
| Figure 50 | Photo Of Canon 4-Pin Power Connector | 36 |
| Figure 51 | Programming/Charger Unit Housed Inside The Control Box | 37 |
| Figure 52a | Photo Sequence Of Ardie Walking – Part 1 | 38 |
| Figure 52b | Photo Sequence Of Ardie Walking – Part 2 | 39 |
| Figure 52c | Photo Sequence Of Ardie Walking – Part 3 | 39 |
| Figure 52d | Photo Sequence Of Ardie Walking – Part 4 | 39 |
| Figure 53 | Photo Sequence Of Sitting Up | 40 |
| Figure 54 | Photo Of The Author With Ardie And Timmy | 43 |
| Figure 55 | Photo Of Filming Ardie And Timmy Together | 43 |
| Figure 56 | Photo Of Timmy Opening A Termite Mound | 44 |
| Figure 57 | Photo Of Timmy Feasting On Termites. | 44 |
| Figure 58 | Photo of Tina | 47 |
| Figure 59 | Rear leg in position producing maximum torque arm | 87 |

# 1. Introduction

Animatronics is the art of producing mechanical puppets that perform a pre-programmed sequence of movements or are controlled remotely. The first animatrons created were mechanical puppets used in amusement parks. Walt Disney's 'Imagineers' pioneered the field in the early 1960's. From there it spread to the film industry, and today this is where most animatronic creations are found. In the beginning, complex linkage mechanisms were used to give the animatrons motion. Today motion is achieved by a combination of motors, pneumatics and hydraulics, and computers are used to control the motion. The main difference between animatronics and robotics is that robots have sensors, and thus can react dynamically to their environment with the aid of artificial intelligence, whereas an animatron is essentially a puppet, that performs a pre-programmed sequence of motions.

The author has long since had an interest in animatronics and robotics (See Appendix A), and was presented with the opportunity of building an animatronic aardvark for a National Geographic wildlife film. The aardvark was built as part of his MSc in Mechanical Engineering at UCT. The animatron (Ardie) can walk, move its head, eyelids and mouth, and is controlled by a microcontroller. This report describes the animatron and how it was designed, built and programmed, as well as its performance while being filmed.

The main objectives of this report are as follows:

1. To describe the brief that was given for the robotic aardvark.
2. To describe the final product that was created.
3. To describe the various prototype stages involved in building the aardvark's components.
4. To explain how the animatron was programmed and controlled.
5. To comment on the effectiveness of the final solution and suggest further work to be done and improvements to be made to it.

Although the author researched many areas of robotics as part of his MSc, this report focuses primarily on the building of the robotic aardvark, and material related to that. Material that is not directly related to the aardvark appears in the appendices. Much of the work that went into this project was in the programming of the aardvark. Since the program listings may be difficult to understand for those without a programming background, a full explanation of the program is given as well as the code.

Most of the research for this project went into finding the most suitable way for controlling the animatron, how to program the controller, and how to interface the servos to the controller. It was found that the largest and most up-to-date resource for this information was the Internet, and hence the author used this extensively. In addition, various books on robotics, electronics and computer science where referred to.

This report will read as follows: First the brief given to the author by the Film Producer will be presented and then the final product will be described with an emphasis on how each of the requirements in the brief was actually met. A description of how each of the movements was achieved will be given – both physically and from a control point of view. Once the final solution has been presented, the design process of each component/feature will be given. An explanation of the various conceptual designs, the prototyping processes, and why the final solution for each feature was chosen, will be given. The order in which these are described will follow the order in which the author approached the problems – i.e. first develop the linkage system for a single leg, then find a control solution for it before duplicating the leg and developing the control for all 4 legs. The head, eyelid and mouth movements will then be discussed. A report on how the animatron performed while being filmed will be given in section 4, and then conclusions and possible future work will be discussed.

# 1a. Brief Given by Film Producer

The film producer had envisaged using the animatron in various scenes, and in order to make these scenes a success, the following points had been given in the initial brief:

1. The animatron should resemble an aardvark, but it did not have to look like a real one. It was more important for it to be a likable character than for it to be an accurate model of an aardvark.

2. The animatron should be able to walk. This was important, as the Film Producer wanted to have the animatron walk into a scene being filmed, then stand still and interact with other action happening in the scene before walking out of frame. This helps to create continuity in the film. It was also hoped to film the animatron walking alongside the real aardvark as it was searching for termites, and even to film it walking down into an aardvark burrow.

3. There should be sufficient facial and head movements to make the animatron look 'alive' and as though it was interacting with other action in the scene, such as on of the research scientists explaining about the equipment he/she was using.

4. The animatron should be robust, as while filming on location in the bush, it would be difficult to do any repairs, and this would result in costly downtime for the rest of the crew.

5. While filming on location, the routine would be to find the real aardvark in the evening using a radio-tracking device, and then follow it through the bush and film it searching for ants, putting the animatron into the scene whenever the terrain etc allowed it. Since this could go on for many hours until the aardvark went to sleep in the early morning, the power supply for the animatron would have to last for several hours.

6. A single person should be able to control the animatron. This was important since it would be difficult to have more than one person trying to coordinate all the movements while the animatron was interacting with other live action going on around it, and the budget did not allow for a second person.

7. Since the budget for the film was limited, it was hoped that the materials and construction cost for the animatron would be less than R7000.

8. Since filming was to begin in April 2001, the animatron would have to be completed by then, leaving the author 2 ½ months to complete the project.

# 2. Discussion of Final Solution

## 2.1 Overall Design Proportions

For the Animatron to resemble an aardvark, the author had to ensure that the underlying structure had the correct proportions. Since access to a live aardvark was not available during the construction period, photographs had to be used to approximate the proportions. A side profile photograph was used to obtain the proportions of the various leg joints, the distance between the shoulders, the overall height, the head dimensions etc. (see Figure 1) A suitable front profile photo could not be found though, and so the width had to be estimated. It was decided to err on the side of being too wide, and so the original prototype was very stocky. Ardie was later narrowed down (especially the front shoulders) when a better idea of the proportions where obtained. A fully-grown aardvark can measure almost 2m from head to tail, whereas Ardie measures 50cm, making him approximately a ¼ scale.

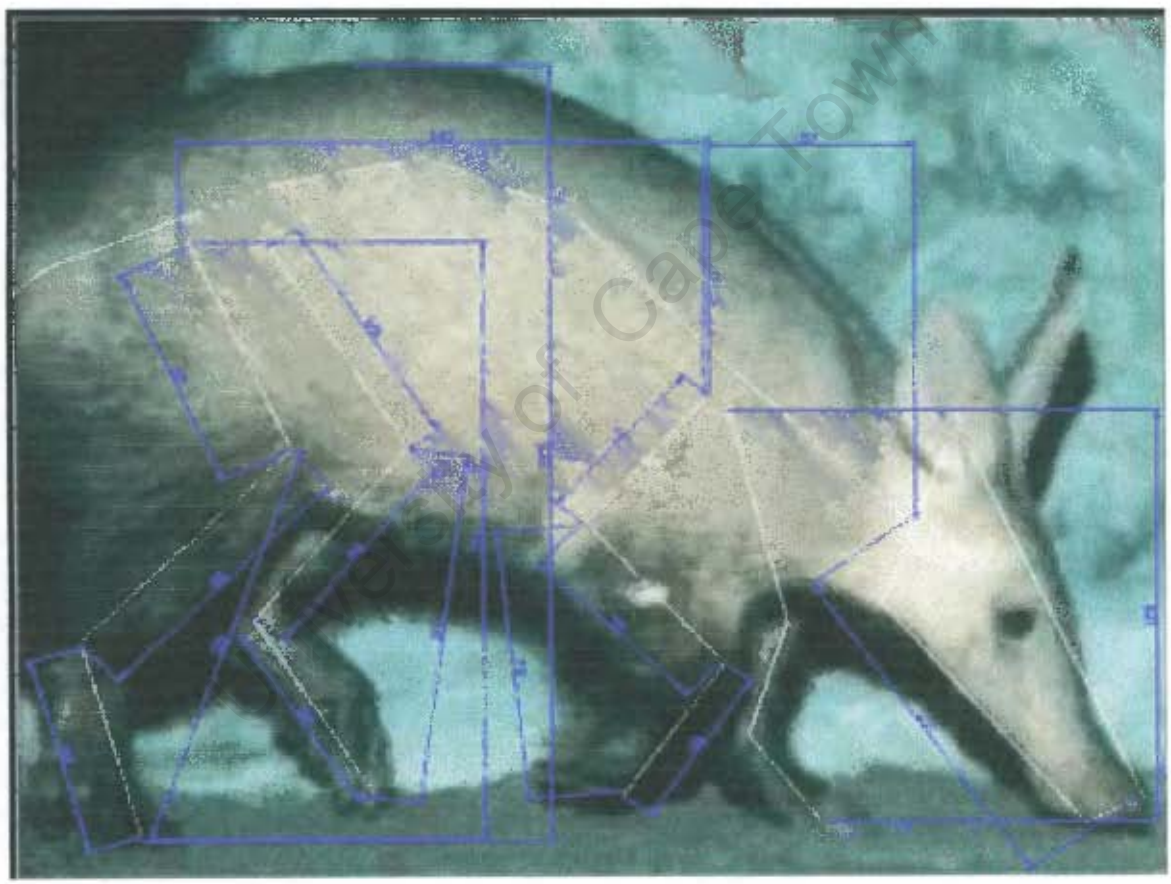


**Figure 1: Photograph used to obtain the anatomical proportions of an aardvark**

## 2.2 Leg Design

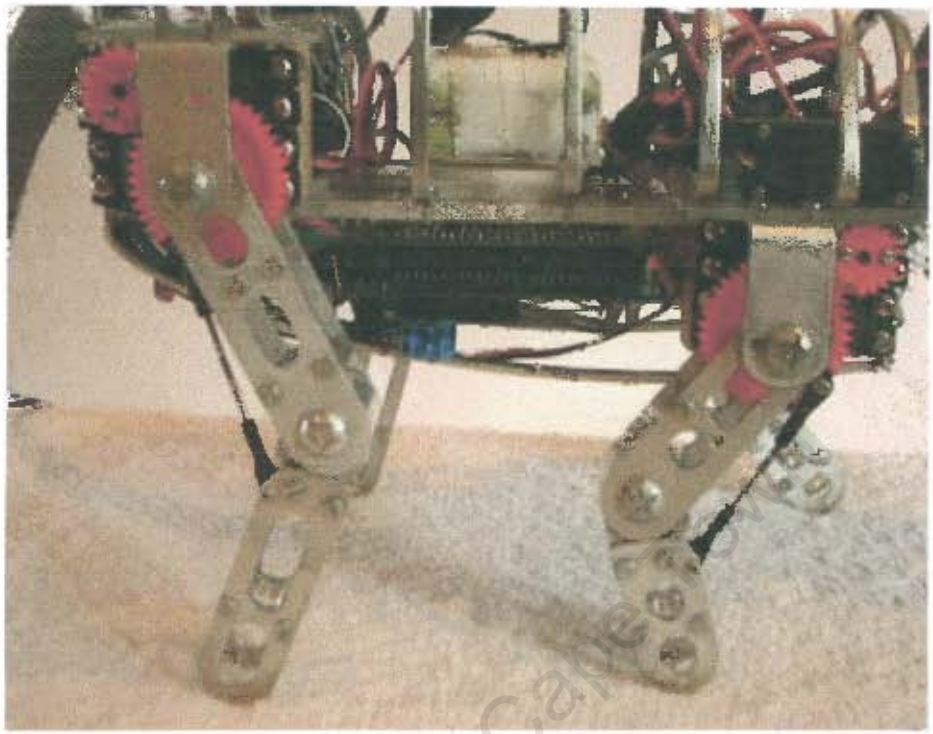


**Figure 2: Side view of the front and rear legs of the animatron**

Each leg has two degrees of freedom, and a servo is used to give motion to each of these degrees of freedom. Although this is sufficient to achieve a walking motion, it is not an accurate representation of how a real aardvark's leg works, as this has a third degree of freedom. The hip and knee joints are replicated on Ardie, however there is no ankle joint.

The upper leg motion is achieved by connecting the 'thigh bone' directly to the servo shaft, while the lower leg motion is achieved by gearing down the second servo's output, and then driving a push-pull rod coupled to the lower leg

As is the case with the real aardvark, the front and rear legs are not identical. Although the driving mechanisms for both pairs of legs are the same, the front legs are shorter, and have the knee bending in the opposite direction to the rear legs.

After building the prototype from Perspex, an aluminium version of the leg was designed and machined using a CNC milling machine.

## 2.3 Head Design

**Figure 3: Oblique view of Ardie's Head**

The head itself has 2 degrees of freedom, and in addition the upper and lower eyelids can be moved and the mouth opened. A total of 5 servos are used to achieve these movements in the following ways:

**Up/down**

This is achieved by a servo mounted on the body of the animatron, which drives the head via a push/pull rod. The head is attached to the body by a hinged joint, which allows for pitch movement. The push/pull rod is mounted at unequal distances from the servo and head hinge, producing a slight mechanical advantage in the linkage. This in effect gears down the servo, making more torque available to drive the head. The range of motion for the head is approximately 40 degrees, which allows Ardie to lower his nose to the ground while standing, and to look straight up ahead.

**Left/right**

The left/right motion is achieved by a mini servo mounted in the head. The output shaft is connected directly to the up/down hinge. The mini servo has a range of 60 degrees, and so the head can be moved to 30 degrees each side.

The two upper eyelids can be moved together as well as the two lower lids. A pair of mini servos in the head is used for these motions – one drives the upper lids, while the second drives the lower lids. The motion is achieved by a pulley system – mounted to the servo shafts are small pulleys, and attached to the pulleys are fine threads. The threads are fed thought a series of guides and then attached to the perimeter of the eyelids. As the servos rotate, the threads are wound onto the pulleys, and the eyelids are pulled open. The eyelids are spring loaded with small elastic bands, which pull them closed as the tension in the threads is released by reversing the servos. This allows the eyelids to be opened by varying degrees, and at different speeds, which helps to add character to the aardvark in the form of facial expressions.

**Mouth**

The mouth can be opened to any position between closed and the fully opened position (approximately 30mm wide). This is achieved by a mini servo inside the head, which drives a helical cam. The cam runs on a ridge in the lower jaw, and so rotating the servo drives the mouth open. The lower jaw is spring loaded with small elastic bands, and so returns to the closed position when the servo rotation is reversed.

**Aesthetics**

The head was made to resemble that of a real aardvark, but was intentionally given a robotic look as well. The snout was shaped from Perspex, while the eyelids were made by bending and shaping strips of aluminium. The eyeballs were made from plastic beads, and are held in place with self-tapping screws, that also serve as pupils. The ears were made from a washing-machine outlet hose.

## 2.4 Control

The animatron is controlled by a combination of a Motorola 16HC11 based microcontroller board and a PIC based Mini Serial Servo Commander, as well as a customized control box that is used to send signals to the microcontroller. Each of these is described in detail below.

### 2.4.1 Handy Board

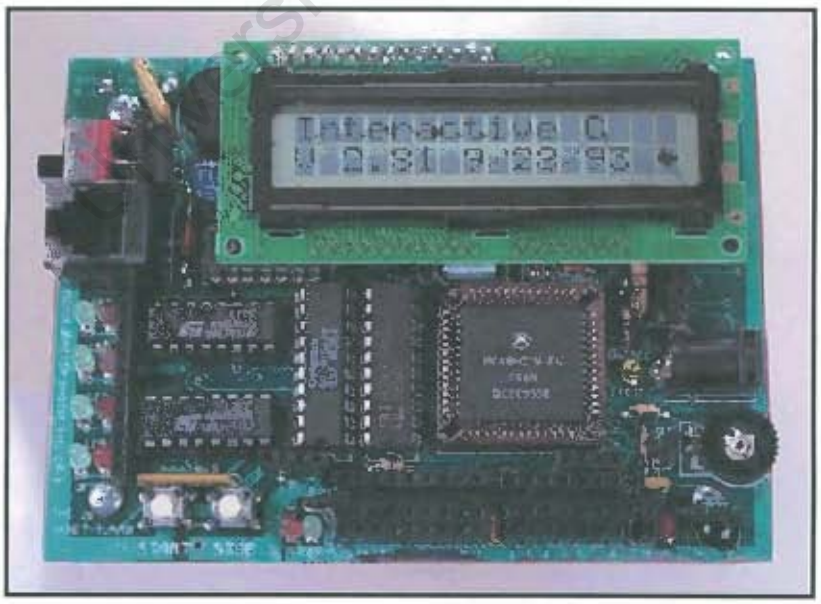


Figure 4: An assembled Handy Board (excluding expansion board)

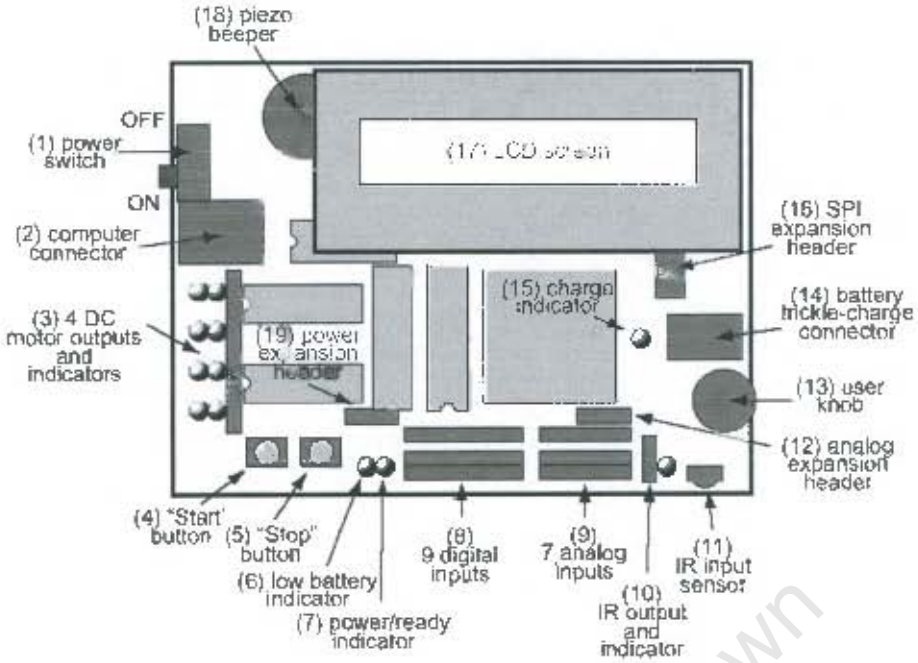



**Figure 5: Diagram of the Handy Board's components**

The Handy Board was developed by Fred Martin at MIT, and is based on the popular Motorola 16HC11 microcontroller. The board was developed specifically for small-scale robotic projects, and is widely used throughout the world in universities and for private robotic projects. Because of its wide use, it has a vast support and information resource on the Internet, with input from users around the world. The Handy Board has many features that make it ideal for robotic projects. In addition to the standard Handy Board, an expansion board is also available, which plugs into the Handy Board and provides additional input and output, as well as an interface to other devices such as hobby servos.

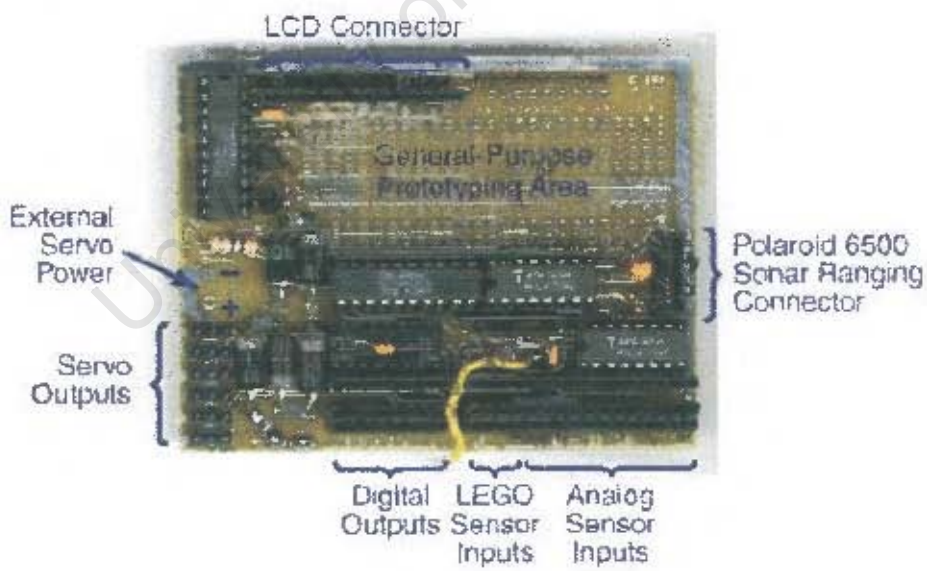



**Figure 6: Handy Board Expansion Board**

A Handy Board and expansion board were ordered in kit form from Acroname Robotics in Colorado USA, and assembled by the author. As the board was only capable of controlling 6 servos, a solution had to be found for controlling a further 8 servos. This was achieved with a Mini Serial Servo Commander, which is described below

## 2.4.2 Mini Serial Servo Commander (SSC)

The mini SSC was developed by Scott Edwards Electronics. It is a board based on the PIC 16C821 and is specifically designed to control 8 hobby servos from a computer serial port. The board receives commands via serial communication, and these instruct it where to position each of the servos it is controlling. The commands are in the form of a servo number (0-7) and a position (0-254). Once a command is received, the SSC drives the corresponding servo to the instructed position, and holds it there until a new command with a different servo position is recieved.

In order for the Handy Board to control all 14 servos, one of its digital output ports was used to communicate with the SSC. The port was used to send commands to the SSC via serial signals, in the same way that signals would be sent to the SSC from a computer serial port (refer to Appendix I for a detailed description of this). The Handy Board sends commands to the SSC to control the 8 leg servos, which are attached to the SSC, while the other servos for the head and weight shifting are controlled directly by the Handy Board.

## 2.4.3 Control Box



**Figure 7: Hands in position on the control box**

The control box is used to send signals to the Handy Board via a tether. The box is used to control the various head movements and has push buttons that are used to start or stop certain sequences or movements (such as walking, sitting up etc.). The control box allows the operator to control more than one head movement at the same time, and in real time, which makes the movements appear more realistic. Also, this allows the animatron to 'react' to things around it as they happen. For example, when filming Ardie and Timmy (the real aardvark) together, it was vital that Ardie could follow Timmy's movements with his head to make it appear as if he was actually watching him. This kind of control was not possible without the control box.

The serial communication and recharging circuitry for the Handy Board are also housed in the control box. The tether from the box to Ardie carries two separate power supplies – a 6v supply to power the servos, and a 12v supply to keep the Handy Board's onboard batteries charged.

## 2.4.4 Infra Red Control

Initially, it was decided that Ardie should be able to carry his own power supply, and be controlled without any tethers attached to him. To achieve this type of control, the Handy Board's infrared receiving circuitry was used. The board was programmed to receive and decode infrared signals sent from a standard Sony TV remote control. Each button on the control could be programmed to start or stop a sequence (such as walking), or move one of the servos a certain amount. This form of control was adequate for starting and stopping the walking sequence, or making Ardie perform pre-programmed sequences such as sitting up, but because each push of a button could only be used to move a servo incrementally, the IR control was not adequate for producing smooth and realistic head movements.

## 2.5 Power Supply

The animatron has the following power requirements:

Standard and high-torque servos – 6v
Mini servos – 4.8v
Handy Board – 9.6v
Mini SSC – 9v

**Figure 8: Two 6v 10Ah lead acid batteries used to power the servos**

The 6v supply for the standard and high-torque servos is supplied by a 10Ah lead acid battery pack, and is routed through the tether via the control box (shown in figure 8 above). This supply allows Ardie to be operated non-stop for a few hours, which was more than adequate for a night of filming. The 6v supply is also regulated down to 4.8v and used to supply the mini servos. Although these servos are connected to the Handy Board for control, they receive power via the Handy Board's external supply input. The Handy Board has a 1000mAh 9.6v Ni-Cad battery pack which powers the processor and memory, and this is carried onboard Ardie. The mini SSC has a small 9v battery to power it's logic circuitry, and this is also carried onboard Ardie. To recharge the Handy Board's Ni-Cad battery pack, current is supplied to it via the charging circuitry housed in the control box. The charger requires a 12v source, and this comes from either an AC adaptor, or the 12v lead acid battery pack (the two 6v batteries are connected in series for this).

## 2.6 Overall Layout

The figure below shows how the various control components, actuators, and power supplies are connected together.

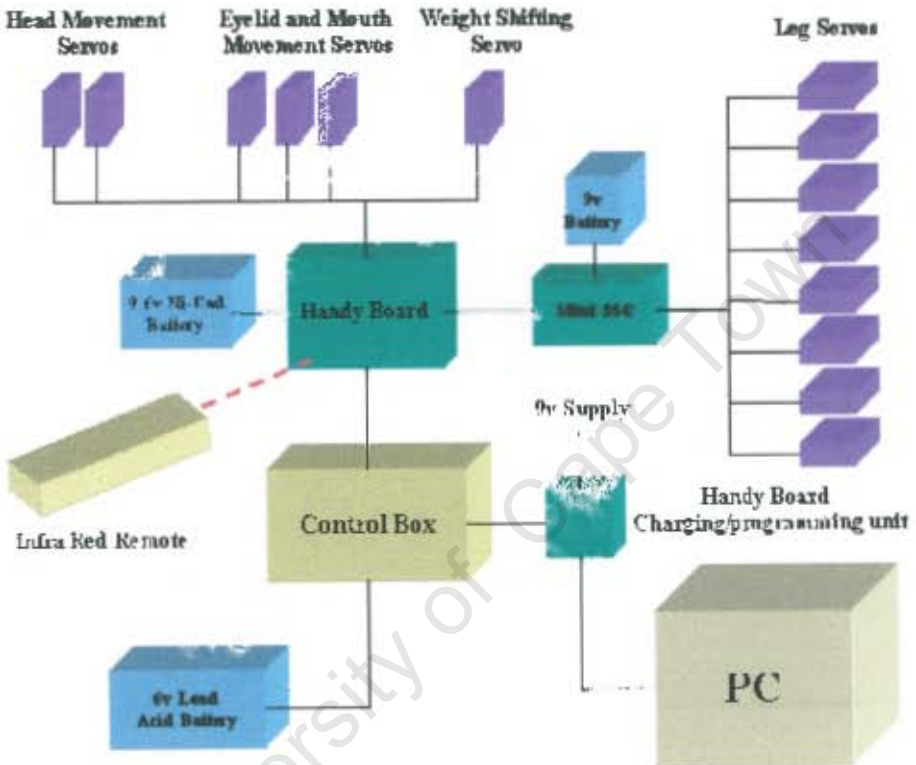



Figure 9: Schematic diagram of overall layout of control components and actuators

## 2.7 Walking

The walking motion is achieved by coordinating the movements of the 8 leg servos. The sequence for their movement is stored in the form of commands in the program that is downloaded to the Handy Board's memory. When the walking part of the program is run, the Handy Board sends signals to the SSC, which in turn drives the servos to the required positions. In order to determine the positions that the legs should be in at each stage of the walking sequence, a technique was developed for programming the walking sequence.

### 2.7.1 Programming the walking sequence

To program the walking sequence, the SSC is disconnected from the Handy Board, and connected to a PC's serial port. A Windows based application was developed that can be used to control the servos attached to the SSC by adjusting slider bars on the screen with a mouse. The sliders are placed in a layout that represents the legs of the aardvark, and each slider represents one of the eight leg servos. By adjusting the slider positions, the legs can be moved to a certain position, and the position can then be saved. The leg positions are saved in the form of a set of 8 numbers, each representing the position of one of the servos. These coordinates are written to a text file and stored on the hard drive. Once the positions are saved, the legs are moved to new positions, and these are saved again. In this way, one cycle of the walking sequence is 'animated', and the positions of the servos in each of the 'frames' of the animation is saved.

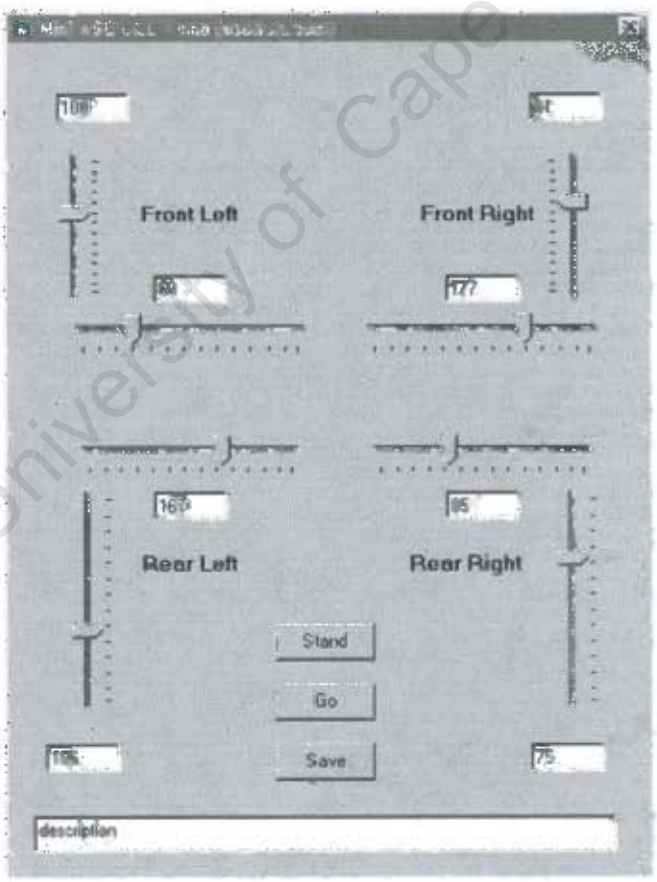



**Figure 10: Screen-shot of the application used to program the walking sequence**

# 3. Conceptual Design

When starting out on the aardvark project, the author decided to deal with what he considered would be the most difficult problems first, and then move onto the problems that he felt more confident about solving. It was realized that making the aardvark walk would be the toughest problem, so this was approached first. Before the mechanical aspect of the walking could be investigated, it was first necessary to decide how the robot would be controlled.

## 3.1 Selection of Microcontroller

Although the author had already gained experience with the Handy Board and a PIC 16F86 based board, it was decided to investigate any other microcontroller boards on the market that might be suitable. Below is a description of these two boards, as well as the OOPic board, and the suitability of each to the aardvark project is given.

### 3.1.1 Handy Board

As mentioned in section 2.4.1, The Handy Board was developed by Fred Martin at MIT, and is based on the Motorola 16HC11 microcontroller.

**The Handy Board (with Expansion Board) had the following advantages:**

- The author had already ordered and built one and had become familiar with using it.
- It has sufficient memory to hold large programs (32K RAM).
- The power supply to the memory is maintained even when the board is switched off, so the program is is not lost.
- The board is programmed using Interactive C, a scaled down version of C, which the author had experience in using.
- Interactive C is interactive – one can type a line of code on the PC, and it is executed immediately on the Handy Board. This is useful for testing and debugging parts of a program before downloading them to the Handy Board's memory.
- It has an LCD screen, which is useful for displaying messages as a program runs, making the debugging process easier.
- It has sufficient digital inputs to interface to the push buttons on the control box.
- There are sufficient analog inputs to interface to the head and eyelid controls on the control box.
- It can be used to control 6 hobby servos.
- An external power supply can be used for powering the servos, reducing the drain on the logic power supply.
- It has infrared receiving circuitry so IR can be used for control.
- There is a large support and resource base on the Internet.

**The Handy Board had the following Disadvantages:**

- It is relatively bulky.
- If the logic power supply batteries are run flat, the program is lost from memory and the board needs to be reprogrammed.

### 3.1.2 PIC 16F84 Based Board

Microchip manufactures the PIC brand of microcontrollers that are widely used in control projects. The complexity and processing power of the PICs vary from the very basic 16F5x family up to the 12Cxxx family. As part of the course in Digital Logic and Microprocessors that the author completed at UCT (MEC 350W), he became familiar with a microprocessor board based on the PIC 16F84, developed by Mr De Beer at Cape Technicon. He had used the board for basic control applications, so decided to investigate its suitability to the aardvark project.

**The board has the following advantages:**

- It is fairly compact.
- Its power requirements are low.
- The board was relatively inexpensive (R250), and the author already owned one.
- The author had experience with using the board and programming the PIC.
- There is technical support available for programming the PIC at UCT, as it is widely used by other students and in the MEC350W course.
- Since EEPROM is used, the program is not lost when power is removed from the board.

**The board has the following disadvantages:**

- It has limited memory, so would not be able to store a complex program.
- The programming has to be done in assembly code, which is more difficult than using a high level language such as C++ or Java.
- The board has a limited number of digital input and outputs.
- There is no analog input (A/D converter).
- Only 4 seven-segment-displays are used to display any information while the program is running.
- There is no built in interface for controlling servos, so one would need to write complex code to do this, and use the already limited output pins to drive the servos.



**Figure 11: PIC 16F84 based board, keypad and programming unit.**

### 3.1.3 The OOPic Board


**Figure 12: OOPic microcontroller board**

The OOPic is a Micro-controller board that was developed by Savage Innovations, and is different from other Micro-controller boards in that it acts as a Programmable Virtual Circuit, and not just a Programmable Micro-controller. OOPic is an acronym for Object-Oriented Programmable Integrated Circuit, and it is the use of the Object-Oriented concept that makes it different. This concept is used in programming languages such as C++ and Java. [16]

**The OOPic has the following advantages:**

- The board is reasonably priced ($45), and the author had ordered and received one from the United States.
- It is compact – just larger than a credit card.
- The programming software runs on the Microsoft Windows 9x operating systems and is available for download, free of charge.
- The OOPic starts running a program that is downloaded onto it as soon as the download has completed. After that, it does not matter if the OOPic's power is turned off, the OOPic will retain its program for a rated 40 years. When the power is turned back on, the OOPic will start running the program again.
- It has a 5-volt regulator built in so any battery, from 6 to 15 volts, can be connected to the power connector. Connections are also provided for a 5-volt power source.
- The board includes a small prototyping area where additional circuitry can be built up.
- It has sufficient input digital input and output, as well as analog input pins.
- Code is available for download from the Internet for interfacing the board with various peripherals, such as servos.

**The OOPic has the following Disadvantages:**

- Although the author had purchased an OOPic, he had not yet had the opportunity to experiment and become familiar with the board.
- It does not include any display.

### 3.1.4 Conclusion on Microcontroller Selection

Before starting the investigation into the various microcontroller boards, it was suspected that the Handy Board would be the most suitable of the three, but the author wanted to make sure. By the end of the investigation it was clear that the Handy Board was the most suitable choice. Although the PIC board was inexpensive and the author had a fair amount of experience with it, it fell way short of the requirements for controlling 14 servos and receiving input from the control box. The OOPic would have met most of these requirements, but as the author had not actually used the board before, he couldn't be sure of this. Learning to use the board would have been too time consuming. It was felt that the Handy Board would be able to handle the required amount of input and output, and even though it could only control 6 servos directly, it had been established that it could be used to communicate with a Mini SSC to control a further 8 servos. Being familiar with the board was a big advantage, as it meant the author could begin using it immediately. The only disadvantage of the Handy Board was it's size compared to the other two options, but once it had been established what size the animatron would be, it was realised that the Handy Board's size would not be a problem.

## 3.2 Development of the Leg Design

Once it had been established how the animatron was going to be controlled, work was begun on designing the legs. Before designing the mechanics of the legs, it was first necessary to establish how they would be powered, as this would determine certain limitations (such as space and available torque) in the design.

### 3.2.1 Selection of Motor Type

To achieve the walking motion, the motors would have to be able to rotate to a position accurately and hold that position. During the research that the author conducted in 2000 various motor types used in robotics were investigated, including stepper motors and servos. Both of these are used for precision positioning applications. From this research, as well as from previous experience with both types of motor, the author deduced that servos would be more suitable for the aardvark project. The reasons for this can be summed up in the following table which compares servo and stepper motors.

**Figure 13a: Stepper motor**

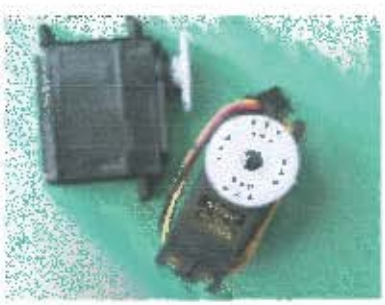


**Figure 13b: Standard hobby servos**

| Criteria | Stepper Motor | Servo |
|---|---|---|
| Size | Compact motors available, but these are very low torque motors | Very compact (40x40x20mm) |
| Weight | Steal construction, therefore heavy | Mostly plastic, therefore light (45g) |
| Torque | Relatively low torque | Motor is geared down to produce high torque for its size. (Standard =0. 3 Nm, High torque = 0.5Nm,) |
| Power requirements | 12V | 6V |
| Availability | Typically salvaged from old hardware, so difficult to find many of the same type. | Freely available in hobby shops |
| Price | Salvaged = free, otherwise from R500 | Standard = R120, High torque = R210 |
| Ease of control | Needs complex code written or a stepper motor driver IC | Controlled by Pulse Width Modulation – can be connected to SSC or Handy Board |
| Durability | Difficult to damage | Gears can be stripped or logic circuitry damaged |
| Precision | 3.6° (full step mode) | 0.7° (software dependant) |
| Range of motion | 360° | 180° |

**Table 1: Comparison between Servo and Stepper motors**

The high torque/weight ratio of hobby servos, their compact size, availability and ease of interfacing to the Handy Board or SSC made them ideal for this type of application. Also, the motion required of the motors for walking is less than 180°, so that possible limitation was not an issue.



## 3.2.2 Geometry and Construction of the Legs

An aardvark's leg has 3 major joints – the hip, knee and ankle joints. To replicate the movements of each of these joints accurately would require 3 motors per leg. Transmitting the power from the motors to the knee and ankle joint would require a complex linkage system (assuming the motors were to be mounted on the body of the aardvark, and not within the legs themselves). Before trying to solve a problem of this complexity, it was decided first to construct a leg with two degrees of freedom, and determine if this could produce a suitable walking motion. This would involve having a hip joint, and a knee joint, but no ankle joint.

The most straightforward way of doing this would have been to have a motor directly drive each of the joints (as illustrated in figure 14), but because of space limitations, it was not possible to mount a servo onto the leg at the knee to drive this joint. An alternative was to have both motors mounted at the hip end of the leg, and to use some way of transferring power from the knee motor to the knee joint.

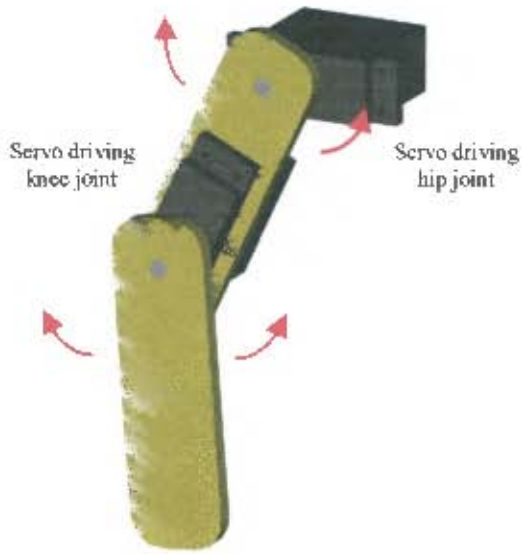



**Figure 14: Illustration of how the servos could be used to drive the joints directly**

Because of the limited time, budget and materials, it was not possible to simply design the ultimate aardvark leg and then build it. Instead, the author had to make do with the materials available to him, and follow a process of trial and error with various prototypes. First, lengths of balsa wood were used to experiment with the geometry and motion of the leg, and then a prototype was built using 3mm Perspex. Once the design had been finalized, the leg was rebuilt using aluminium.

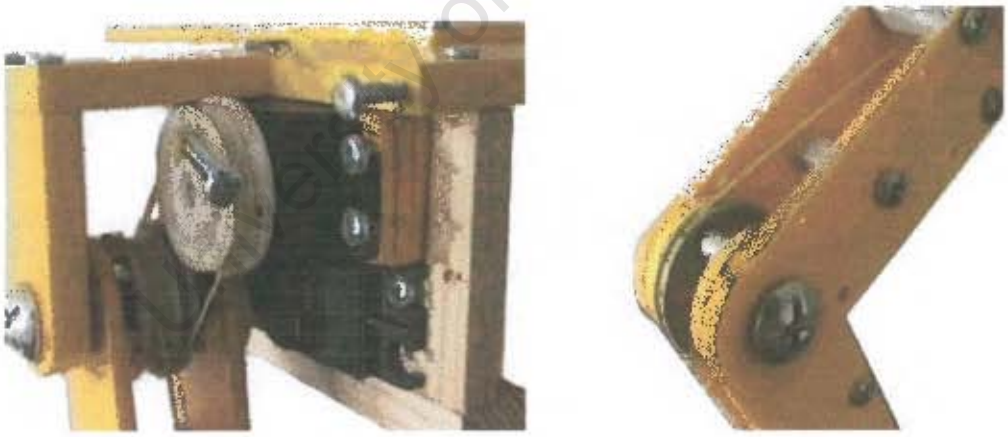


**Figure 15a and b: Photos of the Perspex leg prototype using pulleys and chord to drive the knee joint.**

The first leg prototype that was built had the hip joint directly driven by a servo, and used a system of pulleys and nylon chord for a second servo to drive the knee joint. (See figures 15a and b). Initially, a double pulley system was used – the first set transferred power from the servo to a second pulley mounted on the axis of the hip joint servo shaft. From here, the second set of pulleys transferred the power to the lower limb. The problem with this configuration was that the geometry caused the knee angle to change as the hip angle changed.

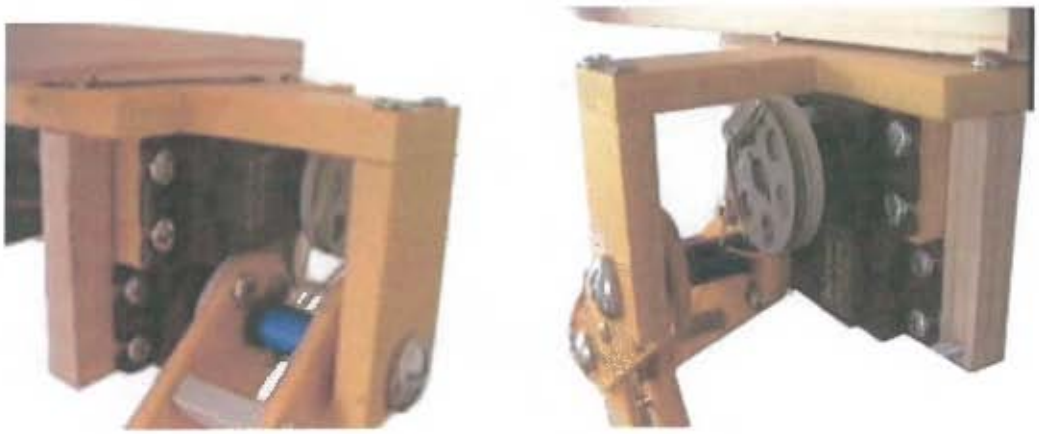


**Figure 16a and b: Photos of the second leg pulley prototype.**

To reduce this effect, the double pulley system was done away with. Instead the servo transferred power to the lower limb by only one pulley, and the chord was fed through a hole in the hip axis (shown in figures 16a and b above). This did reduce the effect considerably, but placed friction on the chord where it passed through the hole, making the joint stiff. Although the pulley system allowed for smooth motion, it was found to be fairly flimsy, and not enough torque was transmitted to the knee joint. Also, it was difficult to set up the tension of the chord consistently, and if all four legs were not tensioned to the same degree, they would react differently, making it difficult to coordinate them accurately.

It was realised that a rigid linkage mechanism would be needed to drive the knee joint, and that the mechanism would have to provide some mechanical advantage so as to increase the torque transmitted to the knee.

**Figure 17: Photo of prototype using small-toothed gears and push-pull rod to drive knee joint**

The next prototype was built using gears and a push-pull rod system to drive the knee joint. The first gear was mounted directly to the knee servo shaft, this drove a larger diameter gear that rotated on the axis of the hip joint servo shaft. A push/pull rod was then connected between the larger gear and the lower limb. The system did work, but it was found that the teeth on the plastic gears were too fine, and the gears tended to jump teeth when the leg was overloaded. Also, not enough mechanical advantage was provided by the system.

The gears were replaced with a pair that had larger teeth, and a larger gear ratio (shown in Figures 19a and b). This solved the problems that had occurred with the finer toothed gears. The only disadvantage of this system was that the knee joint angle changed as the hip joint angle changed. It was decided that this effect could be corrected in programming of the leg movement. Also, it was discovered that the effect actually helped the walking action on the rear legs, because as the hip joint moved back, the knee joint straightened out, resulting in a more linear path for the foot. (See Figure 18)


**Key**

Original leg position – 115° between upper and lower sections.

Final leg position – 145° between upper and lower sections.

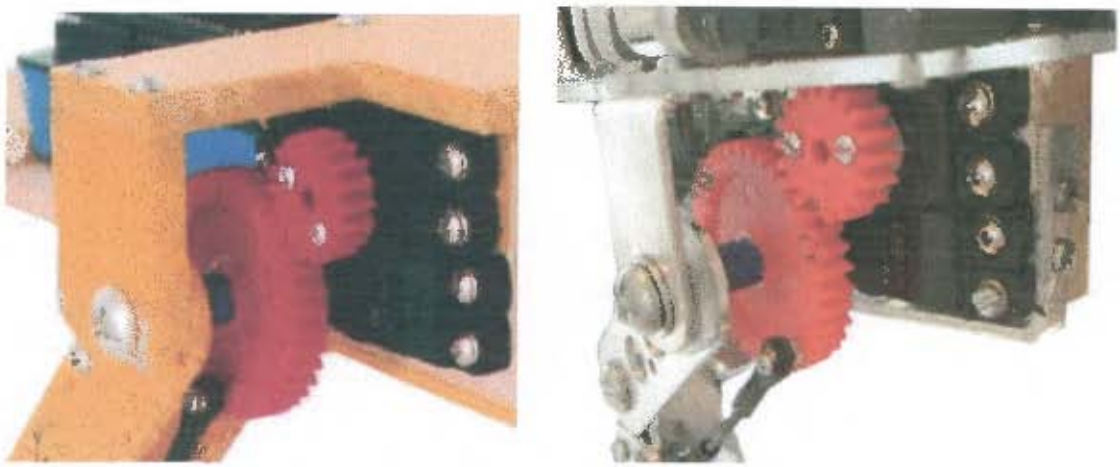
Position that lower section would end up in if the angle of the knee joint did not change as the hip angle changed.

Note:

The large red gear onto which the push-pull rod is attached, does not rotate as the lower servo rotates – it is free to rotate on the axis of this servo, and is only rotated by the upper servo (via the small gear).

As the lower servo rotates the upper section of the leg backwards, the push-pull rod causes the knee to open out. This results in the foot following a path that is almost parallel to the ground plane.

**Figure 18: Illustration of how knee joint changes as hip joint is changed**



**Figures 19a and b: Photos of large toothed gears used for leg in the prototype and final design.**

The front and rear legs of an aardvark differ in size, and in the way the knee bends. With the front legs, the knee bends backwards, while with the rear legs, the knee bends forwards, as it does for a human knee. This meant that two versions of the leg had to be built. The prototyping was based on the dimensions of the rear legs, and so to build the front legs some modifications to the original design had to be made. The same driving mechanism was used, but the lengths of the limbs were reduced accordingly, and the mounting of the leg was reversed.



**Figure 20: Photo of the original prototype with longer legs.**

Initially, standard hobby servos were used to drive the legs, and all four legs were made a certain size. The legs were temporarily mounted onto a strip of balsa wood, spaced out according to the scale of the aardvark that was been worked towards. (See figure 20) After supplying power to the legs and attempting to make the quadruped stand, it was realised that the servos would not be powerful enough for the application, as they could hardly support the weight of the animatron.

Alternative motor options were investigated, and it was discovered that high-torque versions of the servos were available. These have the same dimensions, but produce 0.5Nm of torque as opposed to 0.3Nm for the standard versions. To determine if these motors would produce sufficient torque to support the aardvark, the weight of the final product was estimated, and the torque that each motor would have to produce was calculated. The calculations showed that the motors would still be too weak. Increasing the torque of the servos would require using the next available size of servo, and these would be too bulky for the aardvark. Also, they were considerably more expensive (R600 as apposed to R210 for the small high-torque servos). It was decided to modify the legs until the standard sized high-torque servos could produce sufficient torque to support the animatron while walking. This involved reducing the length of the limbs, so that the moment produced by each about the hip joint would be reduced. Using the same weight approximation as before, a maximum length for the limbs was calculated. The solution that was arrived at meant that the animatron would be approximately ¼ scale of a live aardvark. This would be suitable for filming purposes, and so the length and motor modifications were made (shown in Figure 21), and the quadruped prototype was re-tested. This time it could support itself easily.

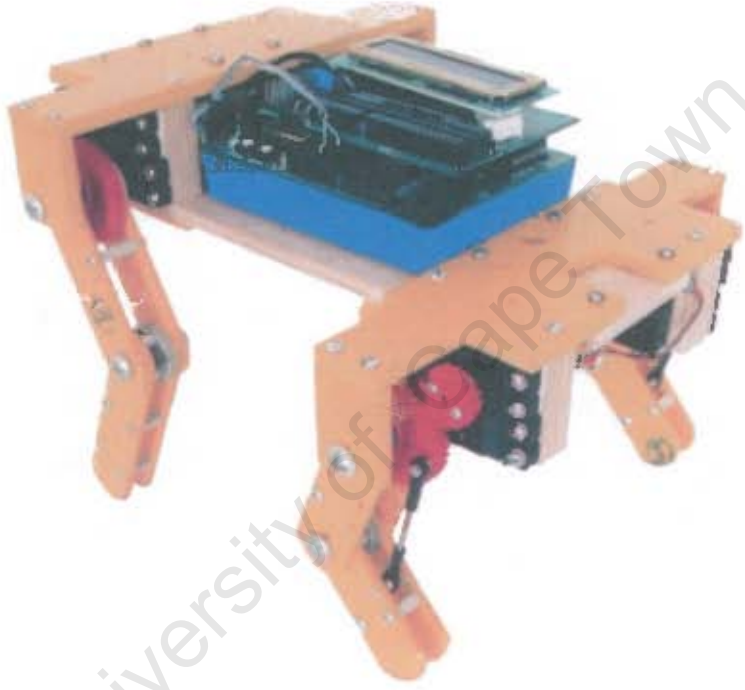


**Figure 21: Photo of prototype with shortened legs.**



Once all four legs had been attached to a makeshift body, it was time to determine if it would be possible to make the animatron walk with only two degrees of freedom per leg. After testing various gate options, one was discovered that worked with two degrees of freedom, and so it was not necessary to add the third degree of freedom.

After sufficient testing had been done with all four legs and the design was found to be satisfactory, it was decided to have them remade in aluminium. This decision was made primarily to increase the rigidity and durability of the legs, as the Perspex versions had shown signs of flexing while under load, and it was important not to have any failures while filming out on location in the bush. A 3D model of the leg was created s using AutoCAD, and from this drawings were produced for the key structural components (See Appendix E). Mr Martin Batho then used the drawings to program the CNC milling machine in the Mechanical Engineering workshop at UCT, and machined the parts. It was decide to keep the double plate structure supported by cross-pieces, as had been used in the Perspex prototype. This reduced the weight of the legs, while giving them the rigidity they required. The knee structure was re-designed using a solid aluminium block onto which the plates of the lower limbs could be fastened. The block was drilled out to reduce its weight. For the hinge pin in the knee joint, a length of Perspex tube was used, and a bolt was inserted through this to keep the tube in place and add rigidity to the joint. For the ends of the push-pull rod, ball and socket joints were used (shown in Figure 25). These were purchased from a hobby shop, and are commonly used in model Radio Control aeroplanes. The Figures below show parts of the manufacturing process for the legs, as well as comparisons between the 3D models of some of the components, and the actual components.

Figure 22: Photo of the CNC machined aluminium leg plates.

Figure 23: Rendered knee joint and photo of the real knee joint.

Figure 24: Leg with knee joint separated.

Figure 25: Close up of ball and socket joint.

Figure 26: Rendered view of leg and photo of real leg.

## 3.3 Development of Head and Facial Features

### 3.3.1 Head Movement

A real aardvark has three degrees of motion for its head – pitch, yaw and roll. To re-produce all three in the animatron would require three servos, which would be very bulky. It was decided that two degrees of freedom would provide sufficient movement to make the head motions look realistic. The up/down and left/right movements (pitch and yaw) were seen as being most important, and so it was decided to replicate these two.

After having used hobby servos for the legs, the author was sure that these would also be suitable for the head motion. The main decision that had to be made was where to mount the servos in relation to the head. They could either be mounted on the body, and transfer the motion to the head via a linkage system, or one servo could be mounted to the body (for pitch) motion, and the other mounted inside the head (for yaw) motion.

**Figure 27: Photo showing neck hinge for pitch movement of the head.**

First of all, the author experimented with various ways of having both servos mounted on the body, as this would reduce the bulkiness of the head. Transferring power to the head for the yaw motion in this way proved difficult – various linkage mechanisms, as well as flexible push-pull cables were tried, but none of these were found to be satisfactory. It was realised that one servo would have to be mounted inside the head, and so the first prototype was built this way. A hinge was created at the front of the body, onto which the head could be mounted. The hinge would allow the head to have pitch movement. The simplest way of achieving the pitch motion would have been for a servo to drive the neck hinge directly. The servo would have to be mounted very far forward on the body in this configuration, and would give the aardvark a bulky looking neck. Instead, it was decided to mount the servo further back, and transfer power from it to the neck with a push-pull rod (Shown in figure 27 above). The yaw servo was mounted vertically and up side down with its output shaft attached directly to the free end of the neck hinge (see figure 28). The head could then be attached to this servo. This configuration produced satisfactory head movements, allowing the head to be turned 40 degrees to the left and right of the forward position, and gave approximately 40 degrees of pitch movement.

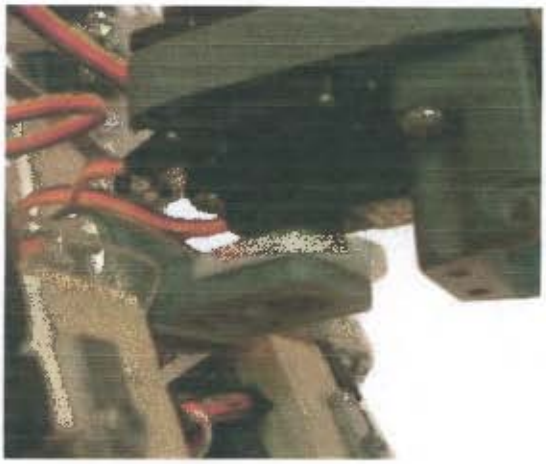


**Figure 28: Photo showing the yaw servo attached to the neck hinge.**

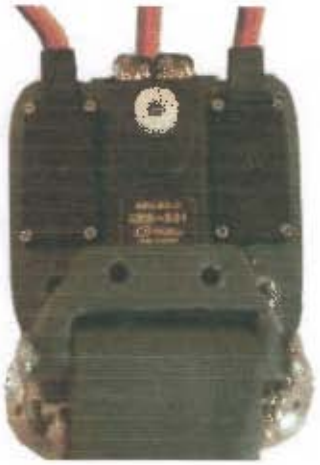


**Figure 29: bottom view of head showing the yaw servo in the centre.**

When the design of the eye movement mechanism was begun, it was realised that having a standard sized servo for the horizontal motion in the head took up too much space, and would have resulted in the head being too bulky. Various other hobby servo options were investigated, and it was discovered that mini servos were available. These were more expensive than the standard servos (R200 as apposed to R120), and could produce less torque, but would be suitable for the head application. The standard servo was replaced with a mini servo (JR Servo model NES 331), and this became the final design solution.

Figure 29 shows the servo in position, mounted between the two eyelid movement servos. The only problem that the author foresaw with this solution was that the shaft of the mini servo would support the full weight of the head. To prevent the servo from being damaged and to give extra support, a swivel bush plate was added. In figure 28, the plate is missing, while in figure 30 it has been fitted. Figure 31 shows the bush plate.

**Figure 30: Photo showing swivel bush plate attached to add support for the head.**

**Figure 31: Photo of swivel bush plate**

## 3.3.2 Development of the Eyes

### 3.2.2.1 Eye Movements

In order to give more life to the character, the Film Producer had asked the author to create some movement with the eyes. It was realized that because of the limited space available inside the head, it would only be possible to achieve two such movements (each movement would require an actuator). Either the eyeballs could be made to move up/down and left/right, or the eyelids could be given movement, to open and close the eyes. It was decided that the latter would be more effective for filming purposes, as eyelid movement would be more noticeable than eyeball movement.

For the first eyelid movement prototype, Nitonol™ shape-memory metal wire was used, as this would require minimal space, and consume little power. Although this material seemed perfectly suited to achieving the eyelid movement, a few problems were soon discovered with using it for this.

The first was in the way the wire regains its shape after the current has been removed – it requires a restoring force to pull it back to its original length. Small springs and lengths of piano wire were tried for this – the Nitonol would deform the spring or bend the piano wire as it contracted, and then be restored by the spring force when the current was removed from it and it cooled. The trick with this technique is applying the correct restoring force – too little and the Nitonol™ will not return to its original length, too much, and it will remain stretched when the current is removed. Its 'memory' will then be reset to this length, and the wire will not contract when it is heated again.

Another problem with Nitonol™ is in the way it has to be fastened to other structures. The fastener must not damage the wire, and excessive heat must not be applied (such as in soldering), as this will change the properties of the wire. These limitations made it difficult to attach the wire between the eyelids and the restoring springs. More importantly, since four eyelids were to be moved in this way, the wire would have to be attached in a consistent manner so that the movements of the eyelids would be the same.

The rate at which Nitonol™ returns to it's original length is dependant on the rate at which it cools, and thus on the ambient temperature as well as any other factors that could effect the cooling rate (such as air movement). This caused the motion of the prototype eyelids to be inconsistent. Also, under room temperature conditions, the wire would contract rapidly, but the restoration was rather slow and this did not produce realistic eyelid movements.

If Nitonol™ wire is heated above a certain temperature, it will contract fully, reaching its memory state. In order to make it contract to a lesser degree, its temperature has to be carefully regulated. This would involve regulating the amount of current passed through the wire, as well as the temperature of the surrounding air. The first of these would be possible to achieve (using Pulse Width Modulation for example), but would be difficult with the Handy Board. Without such control, the eyes would either be wide open, or closed, and achieving any in-between position would not be possible.

Given these idiosyncrasies of Nitonol™, as well as the fact that a reliable and easily maintainable system would be needed while filming on location, it was decided that Nitonol™ would not be suitable for the eyelid movement.

Instead, the possibility of using mini servos was investigated, and it was discovered that there were servos available that were even smaller than the one used for the head yaw movement (JR Servo model no NES 371). These servos were small enough to be mounted inside the head without making it too bulky. The next design challenge was to create an eyelid mechanism, and to transfer motion from the servos to the eyelids.

**Figure 32: Illustration of how the eyes were to be mounted in relation to the axis of the head.**

What made the second problem challenging was that the eyes would not be mounted square to the head (i.e. looking straight ahead, or out to the sides), but at an angle. This meant that a simple link between the left and right eyelids could not be used to make them move together. The other design decision that was faced was how the eyelids should be paired for movement. Since two servos could be fitted in the head, two separate motions could be achieved. Either one servo could be used for each eye, moving its upper and lower lids together, or it could be used to move the upper lids of both eyes, while the other servo moved the lower lids. The first configuration would allow the aardvark to wink with one eye, but it would be difficult to synchronize the motion of both eyes. The second would allow the top and bottom eyelids to be moved independently. Although this type of movement is not common in real eyes, it would help to give the effect of the eye looking up or down. Also, it would be easier to synchronize the motion of the left and right eyelids, and so this configuration was chosen.

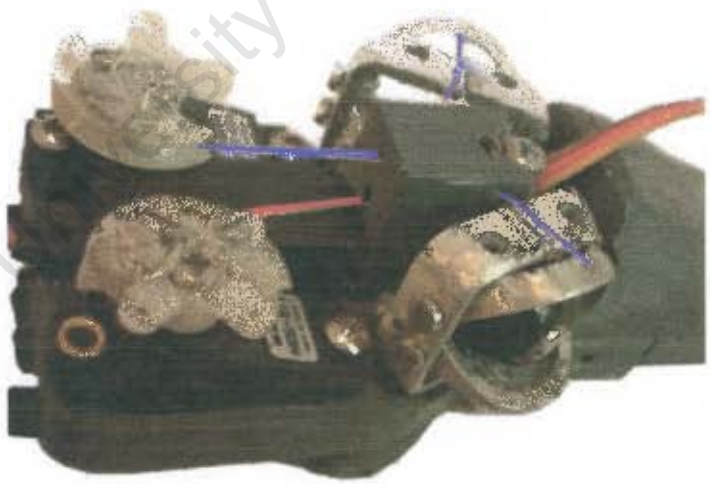


**Figure 33: Illustration showing how the thread was connected between the pulleys and the eyelids. (Thread for upper eyelids highlighted in blue, for lower eyelids in red)**

In order to translate the motion of the servos into eyelid motion, a linkage mechanism had to be devised. Given the limited space available in the head, and the fact that the eyes were not mounted square to the head, this turned out to be a challenging problem. Various linkage configurations and push/pull rod systems were tested, but none of these

provided satisfactory results. A different approach was then tried: instead of a rigid mechanism, a pulley and thread system was used. This immediately produced better results, as it gave more freedom as to where the servos could be mounted in relation to the eyelids, and meant that one servo could easily be used to control the left and right eyelids, without a complicated link between them.

For each servo, a pulley was mounted directly to its shaft, and two lengths of thread wound onto the pulley. The threads were then fed through guide holes – one leading to the right eye, and one to the left. Each tread was attached to the top of the respective eyelid. In this way, as the servos rotated, the threads would be wound onto the pulleys, and in turn pull the eyelids open. As the servos were rotated in the opposite direction, the tension in the thread would be released, and the spring mechanism in the eyes would pull the lids closed again. To ensure that the motion of the corresponding left and right lids were the same (i.e. they opened and closed to the same position) required that the tensions in the threads for each of the eyelids be adjusted until they were equal.

## Construction of The Eyelids

Once the mechanism for moving the eyelids had been finalized, focus was shifted to constructing the eyes and eyelids. Again, this was achieved by building a series of prototypes, with trial and error leading to satisfactory design.

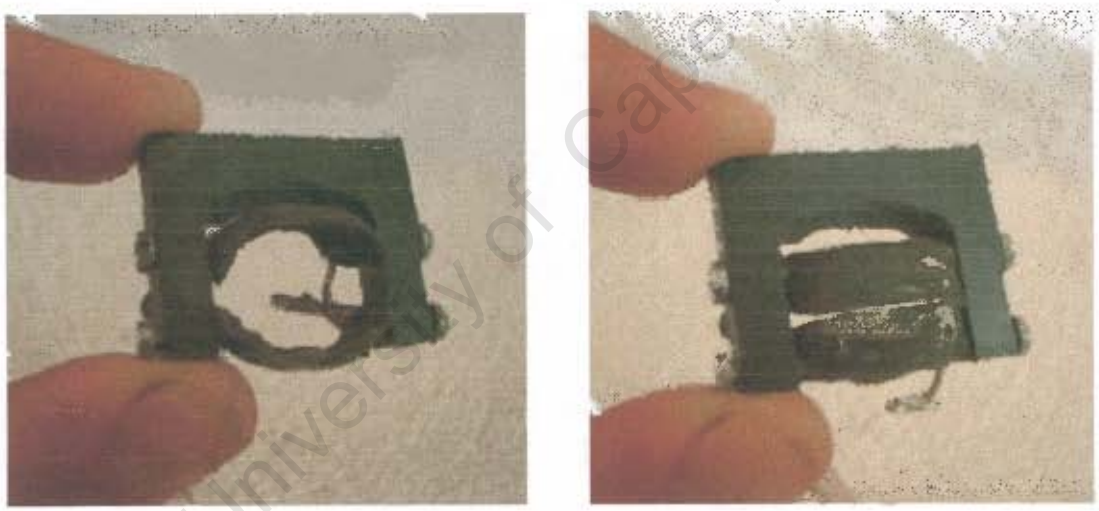


**Figure 34: First eyelid prototype made of Perspex.**

The first prototype was made from Perspex. For each eyelid, a Perspex block was shaped down using a hobby grinder. The eyelids needed to close over an eyeball, and so the block was hollowed out into a half-cup shape. Two such eyelids were then mounted in a Perspex frame so that they were free to swivel up and down. (See figure 34) Although this design did work to a degree, the motion it gave did not look very realistic – it looked more like a mouth opening and closing than an eye. Also, it was difficult to incorporate a spring mechanism that would pull the eyelids closed when the servo released the tension in the thread. Another problem was that when the eyelids were opened, the back of the cup shape of each would interfere with the eyeball mount. The prototype was also bulky, and would be difficult to make it fit into the available space in the head.

**Figure 35a: Eyelids and frame made from aluminium**

**Figure 35b: Eyeball and mounting screw**

For the next prototype, aluminium was used, as this made it possible to make an arched structure that could be used as the eyelid. Having just an arch, instead of a whole cup, meant that the eyelid would have plenty of clearance from the eyeball, and would not be obstructed by the eyeball mount as it was opened. Also, it was possible to extend the strip of aluminium beyond the pivot point in the eyelid to give a mounting point for the spring mechanism. A small aluminium frame was made to house the two eyelids and provide a way of mounting the eye to the head. The eyelids were attached to the frame by two small screws at the pivot points, which also acted as hinge-pins. A small elastic band was used for the spring mechanism to pull the eyelids closed. (See figures 35a and b). This design worked very well with the pulley and chord mechanism for opening the eyelids, and so was used in the final solution.

## 3.4 Development of the Mouth Mechanism

After the head and eye movements had been achieved, it was realized that there would be enough space in the head for another mini-servo, and this would be able to provide some movement for the mouth. It was very limited as to how the servo could be placed in the head though, as it had to fit in with the other three servos. The space in front of the yaw head movement servo was used to mount the mouth servo, which meant it was above the rear part of the mouth palate. The author did not want to have any visible signs of a linkage mechanism visible when the mouth was opened, so had to devise a system that was as compact as possible. Various linkage mechanisms that would translate the motion of the servo into mouth movement were visualized, but all of these would be too obtrusive. It was then decided to use a helical cam system, and a prototype of this was built to see if it would work.

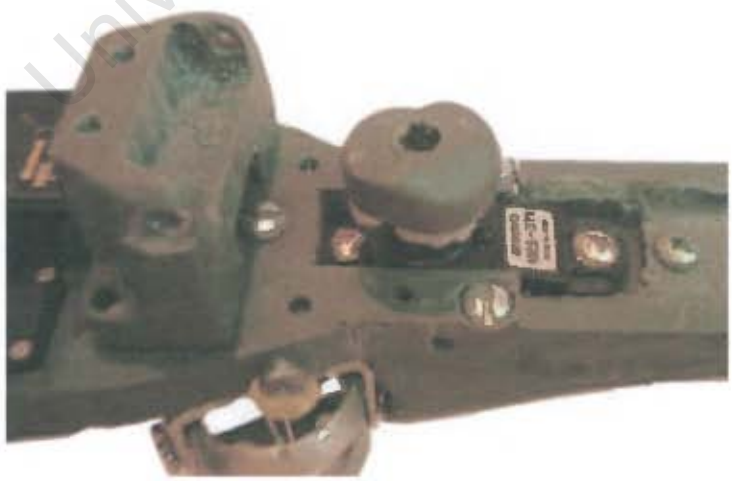


**Figure 36: Bottom view of mouth mechanism with lower jaw removed, showing helical cam attached to mini servo**

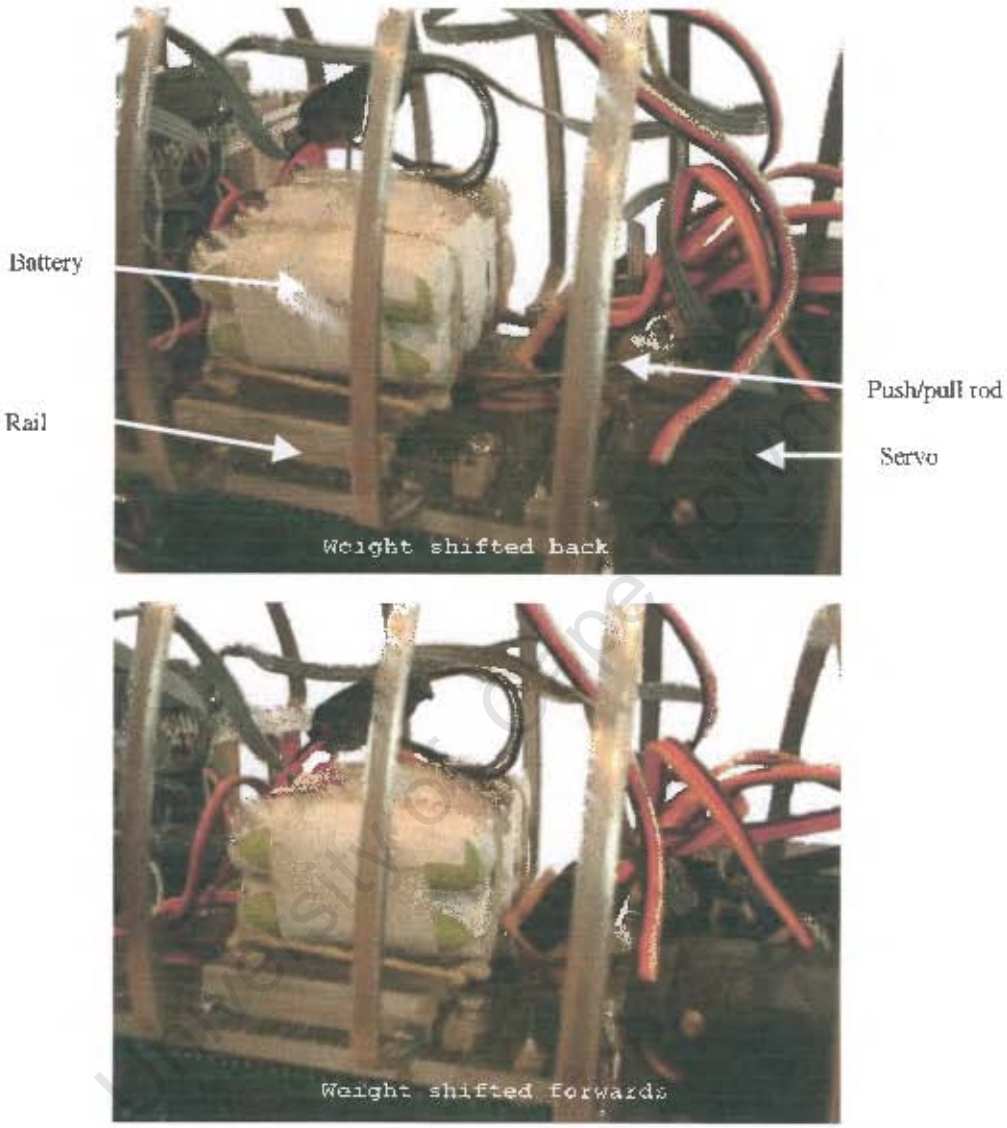

**Figure 37: Lower jaw**

**Figure 38: Lower jaw attached at hinge point. Elastic bands used to close mouth**

The servo was mounted as described above and inverted (See figure 36). A helical cam was shaped from Perspex and fitted to the shaft of the servo. The lower jaw was also shaped from Perspex, and included a ridge on its inner surface onto which the cam would run (See figure 37). The jaw was mounted to the head with a hinge. Rotating the cam would drive the mouth open, while two small elastics were attached between the lower jaw and the head to pull the jaw closed when the motion of the servo was reversed. (See figure 38). This design gave satisfactory results, and so was used in the final solution. The mouth could be opened sufficiently wide, although the movement was rather slow.

**Figures 39: Photos showing mouth in the closed and open positions**

## 3.5 Weight Shifting Mechanism




**Figures 40a and b: Weight shifting mechanism**

Once the first walking prototype of Ardie had been constructed and demonstrated to the Film Producer, one of her primary concerns was that it could not walk fast enough. This was not due to the speed at which the legs could move, but was in fact a result of the way in which the walking motion was achieved. The full weight of the aardvark had to be balanced on three legs while the forth was being brought forward, and then the weight shifted onto the next combination of three legs. This shifting of weight took some time to happen, and it was this that limited the walking speed. If the legs were made to move too fast, the weight would not have shifted to free the relevant leg, and it would make contact with the ground as it was brought through. In order to reduce the time taken for the weight shift, it was decided to devise a way of shifting a mass attached to the body. The mass could either be shifted from side to side, or back and forth. Space limitations on the aardvark made it more practical to use the latter option.

Since the overall weight of the aardvark was critical (the servos could only support a certain load), the author did not want to add additional weight for this mechanism, and so decided to use the Handy Board's Ni-Cad battery pack that

was already carried onboard. The battery pack was mounted onto a base that was able to slide on a set of parallel rods. The rods were secured to the main Perspex sheet that formed the 'backbone' of the aardvark. A standard servo was mounted at the front end of the body, and a push-pull rod attached between the servo and the sliding base. The servo would thus slide the weight back and forth in relation to the rest of the body. (See figures 40 a and b which show the weight shifted to the rear and front respectively) The success of this system for increasing the walking speed is discussed in section 3.8.3 on page 40.

## 3.6 Description of Infra Red Control

In order to control Ardie from a distance, without any visible tethers attached to him, a remote control system had to be developed. The system would have to make it possible to initiate any pre-programmed sequences, such as walking or sitting up, and if possible, also allow for real-time interactive movements of the head and eyes. Since the Handy Board includes an Infra Red receiver and demodulator, it was decided that this would be used for the remote control.

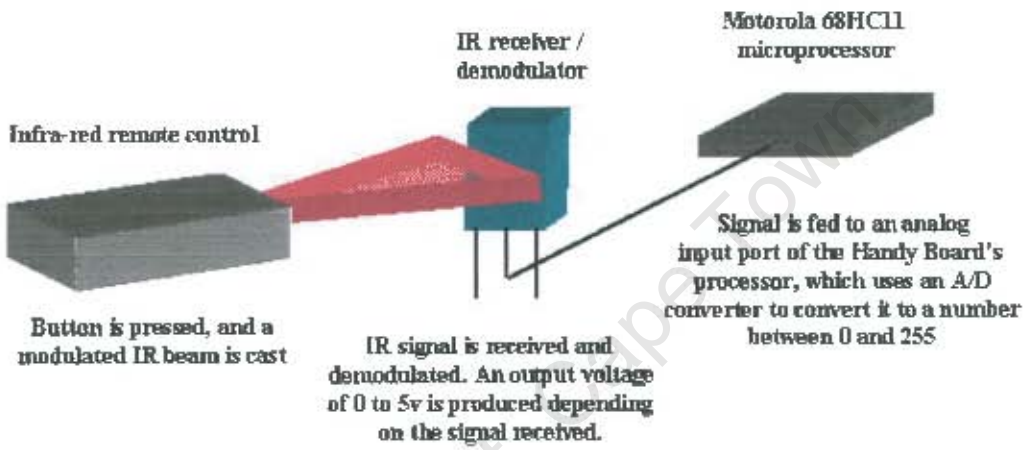



**Figure 41: Illustration of the Infrared control system**

The receiver circuit consists of a Sharp IR demodulator, which is interfaced to the Motorola microcontroller. The demodulator receives a modulated IR signal from the IR transmitter, and produces an output voltage that varies form 1 to 5 volts, depending on the signal received. This voltage is fed to an analog input pin on the Motorola microcontroller where an Analogue to Digital converter converts the signal to a digital number between 0 and 255. Thus any button on the IR transmitter produces a uniquely modulated signal that is translated to a number between 0 and 255 by the Motorola chip. The software library that provides commands to the programmer for making use of the translated IR input was developed by Fred Martin and Brian Silverman at MIT, and is available for download from the internet. The commands allow the programmer to write code which checks to see if the chip has received an IR signal, and if so, what the value of the translated signal was. Conditional branch statements can then be used to alter the running of the program depending on the IR signal received. For example, if an IR signal of 150 was received, initiate the walking sequence. In this way each button on the IR transmitter can be assigned to a certain function. (Refer to appendix H where the program listing is given and an explanation of what it does for a more detailed insight into how the commands where put to use.)

For the IR control, a Sony TV remote control was used with 24 buttons. Since controlling the head movements as well as starting and stopping all of the pre-programmed sequences required more than 24 buttons, some of the buttons were given more than one function, depending on what mode the aardvark was in. For example, pushing the on/off button would put the aardvark either into 'Head' or 'Walk' mode. In Head mode, the volume control buttons would be used to move the head up and down, while in Walk mode, these buttons would be used to change the walking speed.

Depending on the lighting conditions, this allowed the aardvark to be controlled from up to 8m away. The biggest problem was interference from other light sources, such as sunlight and fluorescent light. This made the remote control system unreliable in such conditions, and so the control box was built.

## 3.7 Construction of The Control Box

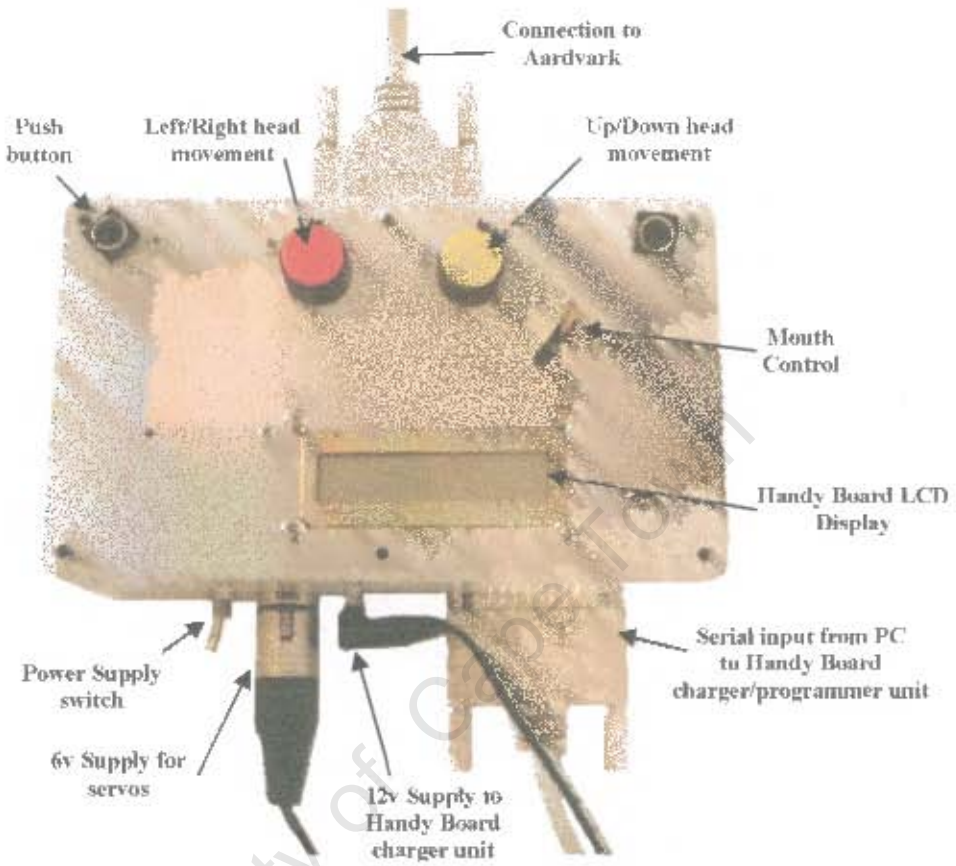



**Figure 42: Top view of the control box**

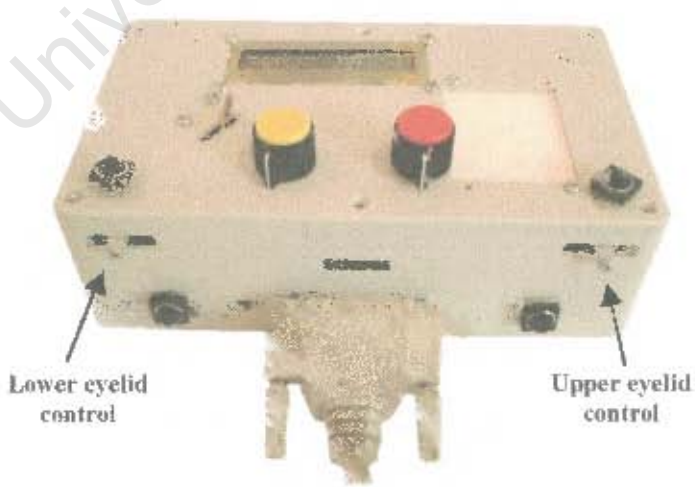



**Figure 43: Front view of the control box**

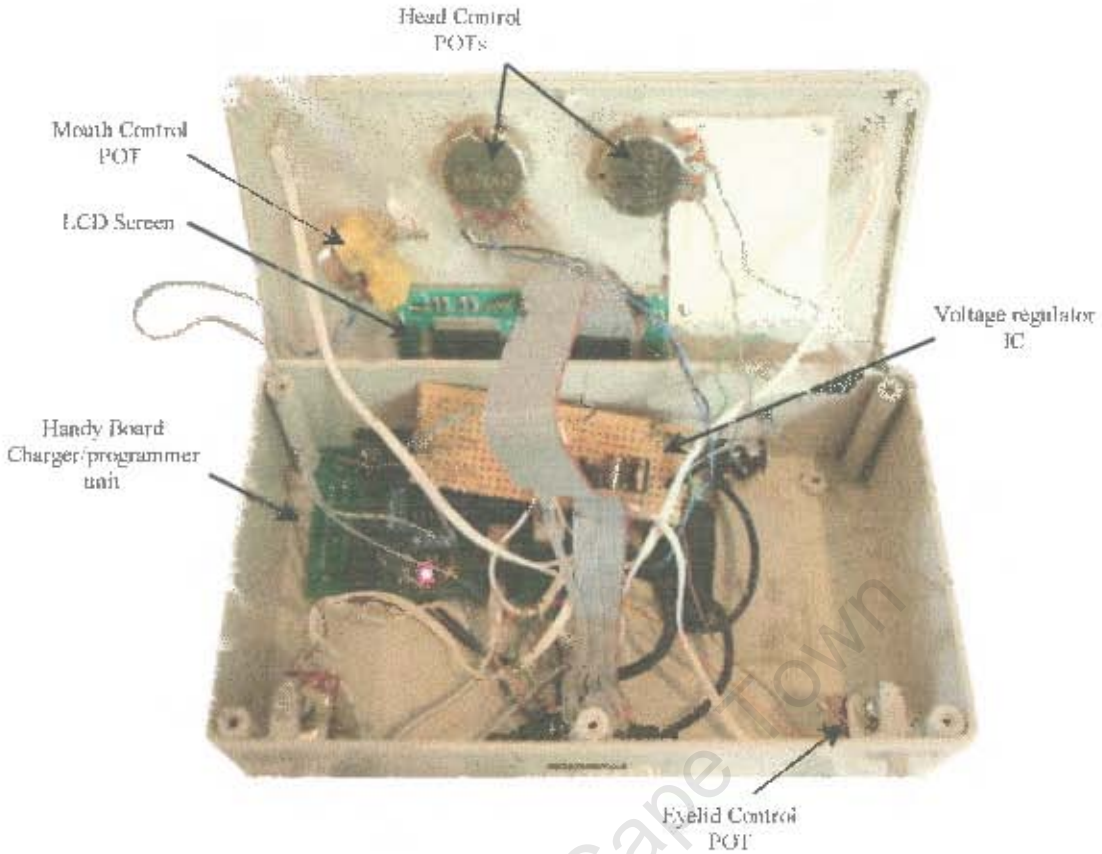



**Figure 44: Interior of the Control Box**

After having controlled the aardvark with the IR remote, it became clear that this would not give suitably realistic head movements, and that the IR was not always reliable due to interference from other light sources. Also, the author wanted to provide a neater method of supplying power to the aardvark (rather than the numerous wires running from the batteries to it), and also to integrate the charger/programmer interface for the Handy Board so that only one cable had to be connected to the aardvark for all these functions. The control box provided the solution to this problem, and the way in which it does so is described below:



## 3.7.1 Head and Facial Movements

A standard computer parallel cable was used to carry all the signals from the control box to the aardvark, as well as supply the necessary power. 25 pin male and female headers were mounted in the control box and on the aardvark respectively so that the cable could be disconnected from either if necessary.

Control of each of the head movements was achieved by connecting a potentiometer (pot) across one of the Handy Board's analog input ports, and then programming the Handy Board to position the corresponding servo according to the voltage measured at the port. Thus turning a knob on the control box would vary the voltage applied to the port, and the servo would move accordingly.

To make this system work correctly, it first had to be calibrated. This was done by measuring the full range of voltages that the pot would produce, and the range over which the servo would have to move. A ratio was then determined between these two, and this was used in the calculations that the program made continuously as it determined where the servo should be positioned.

To make it possible to control the pitch and yaw movements of the head at the same time using only one finger, a joystick was built. This was done by mounting one of the pots to the base of the control box, and then mounting the second pot to the shaft of the first one, at right angles. The first pot was used to control the pitch motion of the head, and the second one for its yaw motion. Once the system had been calibrated, the joystick did work, but the head tended to jitter continuously. This was due to noise in the signals from the potentiometers, as they were poor quality. Also, the joystick only moved the pots a small amount, causing only a small change in their resistance. This small change had to be used to move the servos over a relatively large range, so there was a large multiplication factor that was used in the software. This added to the jittering problem of the head. In order to alleviate this problem, the pots where replaced with better quality ones, and the joystick system abandoned for one using two knobs. This meant that a larger portion of the pot's range could be used, reducing the multiplication factor in the program. This combination provided better results and it eliminated the jitter.

**Figure 45: Potentiometer used to control the upper eyelids**

For the eyelid and mouth movements a similar system was used, but instead of using knobs to turn the pots, small levers where used instead. This was done to make it possible to control many movements simultaneously, as the levers could be controlled with one finger each, whereas the knobs required the thumb and forefinger.

The ergonomic layout of the levers and knobs on the box was also carefully considered. The levers for controlling the eyelids where placed in front where the forefingers would naturally fall while holding the box, while the mouth control lever was placed where the right thumb would naturally fall. Initially when the joystick was used, it was placed so that it could be moved by the left thumb, but the knobs which replaced it required that both hands be used to control the head movements.

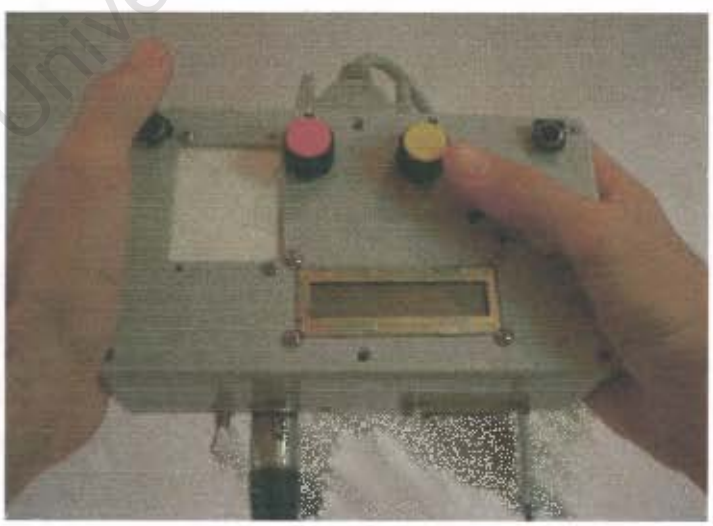


**Figure 46: Hands in position on the control box – right thumb on the mouth movement lever.**

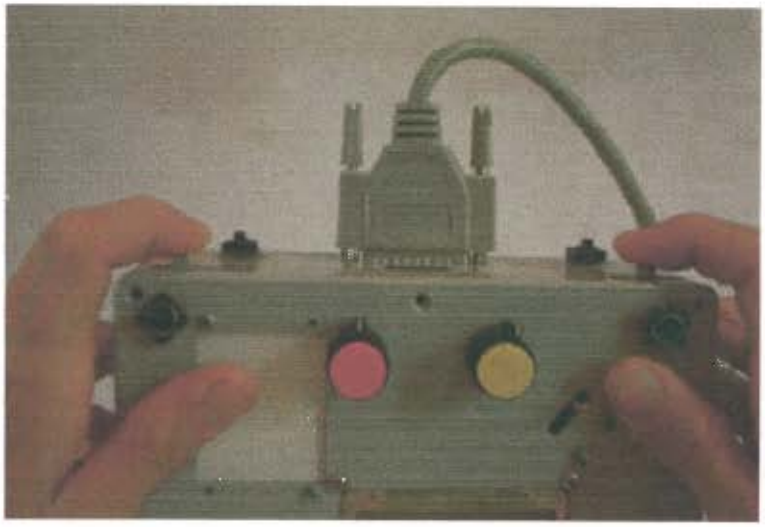


**Figure 47: Forefingers in position on the eyelid control levers**

## 3.7.2 Push Button Controls

To control the walking and other pre-programmed sequences, four push buttons where added to the control box. Each of these was interfaced to a digital input port of the Handy Board. In the program routine where the Handy Board checked the states of its analog input ports for the head movements, it was now also made to check the states of the digital input ports to which these push buttons where connected. Depending on the states of the input ports, the program would branch to carry out a set of instructions (for example, if push button A was pressed, a digital 1 would appear on input port 13, and this would cause the program to branch to the 'wink' routine. Hence pushing this button would make the aardvark wink)

Since four buttons were not enough to cover all the commands that need to be sent to the aardvark, various button combinations were also used, and the buttons had more than one function depending on what mode the aardvark was in. When first turned on, the aardvark would be in 'set-up' mode. In this mode, the buttons could be used to change the speed at which the aardvark would walk and perform the other pre-programmed routines. Then, once both the upper left and right buttons where pressed simultaneously, the aardvark would enter 'control' mode, where all the head movement controls would be activated, and pushing the buttons would initiate or stop the various routines. (for example, pushing the upper left button would start the walking routine, while pushing the upper right button would stop the walking) Again, various button combinations were used to increase the number of commands that could be issued. In addition to this technique, a delay technique was also used – i.e. pushing a button and releasing it would signal one command, while pushing the same button and holding it down for more than a second would signal a different command. This made it possible to control all the movements and sequences necessary for the filming, but meant that the various button combinations had to be memorized by the operator.

### 3.7.3 Power Supply Interface

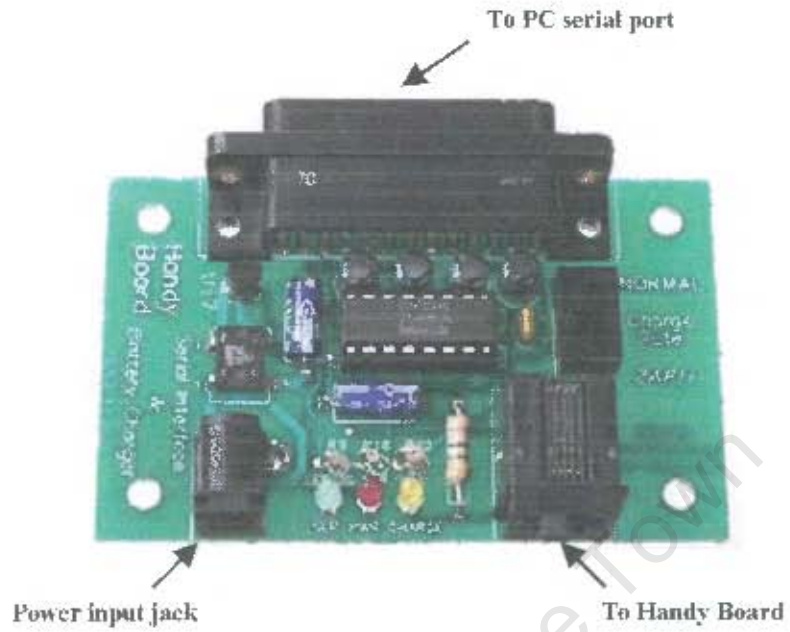



**Figure 48: Handy Board changer and programming interface**

Since the servos were not powerful enough to carry the necessary batteries for long-term continuous operation of the aardvark, an external 6v power supply had to be provided. The portable lights used for filming were powered by 6v lead acid battery packs, and these would be available on the film location, so it was decide to make use of them. The battery packs were already fitted with a standard male 4 pin Cannon connector, and so a female version of the connector was housed in the control box to allow the battery to be connected to it. Power from the connector was fed through a toggle switch, and then to the parallel cable output pins. This 6v supply was connected to the power input pins of the mini SSC in the aardvark, and thus used to power the 8 leg servos.

**Figure 49: Lead Acid battery used to power the servos**

**Figure 50: Canon 4 pin connector**

The servos that were controlled directly by the Handy Board were mainly mini servos, which require a 4.8v supply. The control box was used to house a voltage regulator IC that reduced the 6v supply down to 4.8v. This 4.8v supply was then fed to the external servo power supply pins of the Handy Board via the parallel interface cable.

To ensure that the Handy Board's Ni-Cad battery pack did not loose its charge while operating the aardvark for extended periods, it was necessary to supply the Handy Board with 12 volts while it was in use. This was important while doing the filming, as the Ni-Cad battery was used to power the Handy Board's RAM – if the power failed, the program stored in memory would be lost, and the board would have to be reprogrammed. This would require access to a computer, which would have disrupted the filming process. The Handy Board could either be powered directly or via its charging/programming unit. To use the latter option, the circuit board for the charging/programming unit was housed inside the control box. A separate power supply socket was inserted in the wall of the control box, and connected to the power input point of the charger unit. The power output of the unit was then fed to the Handy Board via the parallel cable. To charge the Handy Board's batteries while out filming, a 12v lead acid battery was connected to the socket that fed the charging unit. When a mains power supply was available, the batteries could be charged by using a 12v transformer, instead of the lead acid battery.

### 3.7.4 Handy Board Programming Interface

To download a program into the Handy Board's RAM, it is connected to the serial port of a computer. This is done via the programming unit, which converts the 13v RS232 signal from the computer to a 5v signal, which is sent to the Handy Board. Usually a standard telephone cable carries the signal from the programming unit to the Handy Board, with RJ11 plugs on each end (telephone plugs). The cable plugs into RJ11 jacks on the Handy Board and programming unit. The programming unit was housed in the control box in such a way that the 25 pin serial input socket was exposed so that it could still be connected to the serial port of a computer as usual. The serial output of the unit was then carried to the Handy Board by the parallel cable that also carried the control signals and power supply.

The Handy Board is supplied with an LCD screen that plugs into it. This is very useful for debugging a program, as it can display text while a program runs, giving the user feedback as to which part of the program is being executed. It was realised that when the Handy Board was mounted inside the aardvark, it would not always be easy to see the LCD screen, and so the screen was mounted in the control box instead. To connect the screen to the Handy Board, a length of ribbon cable was used. This was fitted with male headers at both ends that plug into sockets on the Handy Board and the control box. Since the screen needs 14 lines to interface it to the Handy Board, it was not possible to use the parallel cable that carries the control signals and power supply for this, as not enough lines were available. The screen is thus not always connected to the Handy Board, but is plugged in while debugging a program or when a visual output is required.

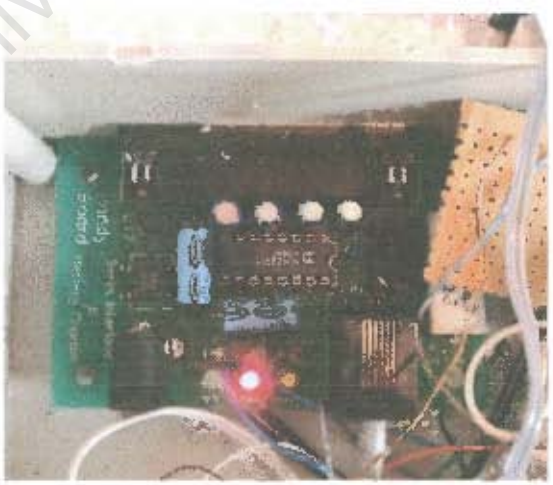


**Figure 46: Programming/charger unit housed inside the control box**

## 3.8 Walking

One of the main requirements of Ardie was that he should be able to walk, as this would help to bring him to life, and would provide continuity in the filming (i.e. being able to move into a shot and then out again).

From the outset, the legs had been designed and built to carry the weight of the aardvark, and to have the movement necessary to for walking, but it was not until all four legs had been mounted on the chassis that the author could experiment with actually making the aardvark walk, and in fact, determining if this would be possible at all.

### 3.8.1 Dynamic vs. Static Stability

Most robotic walkers have 6 or eight legs, which makes them statically stable while walking. This is because they can always have at least 3 legs on the ground at a time, forming a tripod on which to balance while the other legs are brought forward. Thus at any point in the gate, if the action was frozen, they would be stable. In the case of animals, both biped and quadruped, a dynamically stable gate is used. This means that they are stable while walking as long as they keep moving – if the action were frozen at any point in the stride, the animal would be unstable and would possibly fall over. This dynamic stability is achieved by 'falling into' the step and helps to give a smoother walking action.

To replicate dynamic stability artificially in a robot would require a gate that took into account the dynamic properties of the robot. To best achieve this, sensors (e.g. accelerometers) should be used to detect the motion of the robot, and this feedback used in calculating the next step. Since the budget available for building the aardvark would not cover the necessary processing power and sensors for such a system, it was not possible to develop a dynamically stable gate for the animatron.

### 3.8.2 Statically Stable Gate For a Quadruped

Because it was not possible to develop a dynamically stable gate, a compromise had to be reached, and this would mean developing a statically stable gate for the quadruped. Ardie would thus not be able to mimic the gate of a real aardvark, and the walking motion would look rather artificial, but this would be satisfactory for the film.

In order for Ardie to be statically stable while walking, he would need to have 3 legs on the ground at a time, with his centre of gravity over the triangle that those three legs formed. While balanced like this, the fourth leg could be brought forward. To test this system, a windows based application was developed using Visual Basic. This could be used to control the mini SSC to drive the leg servos directly (not via the Handy Board), and to animate the walking sequence. (See Appendix J for a listing of the program). Initial trials showed that it was possible to achieve a statically stable gate for Ardie, as illustrated in the sequences below:

**Figure 52a: walking sequence part one.** First, the legs are shifted so that most of the weight is over the front legs. The front left leg is then dipped slightly, and this frees up the back right leg. The back right leg is then tucked up and brought forward, before being straightened and placed on the ground just behind the front right foot.

**Figure 52b: Walking sequence part two.** The back right leg is then driven further and the other legs adjusted so that the weight is taken off the front right leg. This leg is then tucked up and brought through, before being stretched out to the front. The other legs are then adjusted so that the weight tips onto the extended front right leg, and this frees up the rear left leg.

**Figure 52c: Walking sequence part three.** The rear left is then tucked up and brought forward to be placed behind the front left foot. As was the case with the rear right leg earlier, the rear left is used to take the weight off the front left (the other legs are adjusted to help this too), and the front left is then free to be tucked up and brought forward.

**Figure 52d: Walking sequence part four.** As the front left is brought through and stretched out to the front, the other legs are adjusted so that the weight tips onto the front left leg, freeing up the rear right.

At this point, one cycle of the walking sequence has been completed, and the cycle is then repeated. The first cycle is slightly different to the rest of the cycles though, as it starts off with Ardie in the standing position, whereas the other cycles start with him in mid stride. The forward motion of the body is assisted by driving the three legs that are supporting the weight, backwards, as the fourth leg is brought forward.

### 3.8.3 Using the Weight Shifting Mechanism to Assist The Walking

Once the walking had been achieved, it was realised that Ardie would not be able to walk very fast, as time was needed for the weight to tip from on set of three legs to the next at each stage of the walking sequence. The tipping action was slow because the aardvark was almost balanced at the point where the tipping occurred. In order to speed up the tipping action, this balance needed to be upset. If the centre of gravity of the aardvark was brought further forward, it would tip forwards more quickly, but would then tip backwards slower, or perhaps not at all. In order to speed up the tipping action in both directions, the centre of gravity would have to be shifted. To achieve this, the onboard Ni-Cad battery pack was mounted on a sliding base that could be driven back and forth by a servo. The walking routine was then adjusted to include this new variable as follows: While the aardvark was balanced on three legs, the weight would be shifted so that it was over those three legs, to make the stance more stable. When it was necessary for the aardvark to tip onto the next group of three legs, the adjustable weight was shifted to put the aardvark off balance, thus aiding and speeding up the tipping action.

In practice, it was found that this system did work to a degree, but did not help to speed up the walking very much. The biggest problem was that the adjustable weight could not be moved fast enough by the servo, and so a delay in the walking was necessary each time the weight was to be moved. This made the walking sequence appear staggered and less realistic.

### 3.8.4 Other Uses Of The Weight Shifting Mechanism

The adjustable weight did prove be very useful in other manoeuvres though, such as the sitting up manoeuvre. For this the weight is shifted to the rear so that the front legs can both be freed. When returning to the standing position it is necessary to shift the weight to the front so that the rear legs can lift the rear end of the aardvark. Without the shiftable weight this manoeuvre would not be possible.

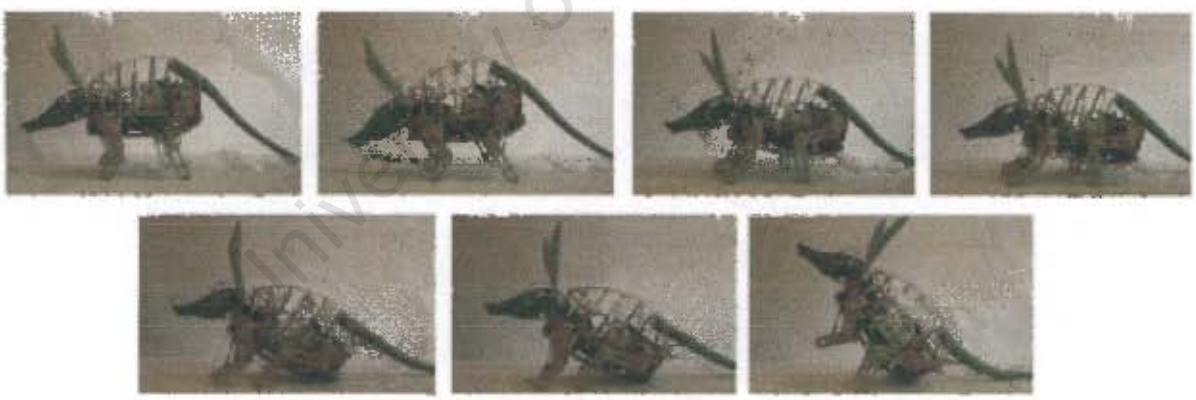


**Figure 53: Sequence showing Ardie sitting up.**

# 4 The Filming of Ardie

The footage of the real aardvark for "The Earthlings" was taken in the "Tussen Die Riviere" game reserve in the Free State, which lies in the triangle of land formed by the junction of the Caledon and Orange rivers. A male aardvark (Timmy) had been researched there on and off for 4 years, and had been habituated to the extent where one could walk alongside him and even pat and scratch him while he burrowed for termites. Timmy had a radio transmitter implanted in his abdomen, and a receiver was used to track him down each night for filming. When the author travelled up to the game reserve with Lynne and Phillip Richardson, most of the footage for the hour-long documentary on aardvarks and aardwolves had already been taken. The aim of this trip was to obtain footage for the 'making of' movie, which would show how "The Earthlings" was filmed. Ardie was to feature in this making of movie, and the idea was to see the making of the film from his perspective. In order for this concept to work, the Director hoped to get the following shots:

- Ardie watching the camera crew as they explained to him what they were doing and how their equipment worked – this would require him to walk into the frame, stop, move his head and eyes while the explanation was given, and then walk out of the frame.

- Ardie and Timmy together – Ardie watching Timmy as he burrowed for ants, Ardie 'shadowing' Timmy as he walked through the veld. This would require Ardie to interact with Timmy using head and eye movements, and to walk over rough terrain.

- Ardie going into and emerging out of an aardvark burrow. This would require him to walk up and down a steep sandy incline.

Until going out into the bush, Ardie had only been tested under 'lab' conditions – he had only walked on fairly flat and even surfaces. Ideally he should have been tested under realistic conditions before going off to do the filming, but time constraints on building him prevented this. The Director realised that most of the shots she required would be very demanding on Ardie, especially on his walking ability, and was prepared in advance to have to make compromises.

It did not take long to realise that compromises would have to be made for his walking ability, as he did not fair well at all in the rough terrain. Because he was only just balancing on three legs while the fourth was being brought forward, it did not take much to upset this balance. A small tuft of grass or unevenness in the ground was enough to trip him up. Also, he was unable to walk on a gradient (even very gentle), as this would upset the balance while on 3 legs.

In order to get the shots of Ardie walking into frame with one of the camera crew, we could prepare a level piece of ground before hand, and film as many takes as we needed to get the shot. When it came to filming with Timmy however, this was not possible as Timmy would be on the move most of the time. When Timmy did stay in one spot for a while we managed to position Ardie near him and then film the two of them together. It was interesting to see Timmy's reaction (of rather, lack there of) to Ardie. He had become so used to having people watching him, and camera equipment pointed at him, that he seemed totally unaware of any human presence whatsoever. He did give Ardie the odd sniff, but didn't show much surprise or interest.

For the shots of Ardie and Timmy walking together, it was realized that a plan would have to be made. Ardie could not walk fast enough to keep up with Timmy, even on clear, level ground. It was decided to mount Ardie on a skateboard and pull him alongside Timmy, while framing the shot so as to cut out the board. A set of mounts was made on a skateboard for this, but this still didn't solve the problem. Even with the skateboard, fairly level ground was needed, and in fact the right set of conditions never presented themselves and so the shot was never taken.

Filming Ardie going into and out of a burrow also required some cheating. For this, a small sled was made from a sheet of aluminium, onto which Ardie could be mounted. He was mounted so that his legs were free to move and he could still carry out the walking action. The sled was then dragged up or down the entrance to the burrow at a rate that matched the moving legs, and the shot was framed to exclude the sled. The final effect was fairly realistic, and hopefully would be convincing enough.

To film Ardie inside the burrow required quite a bit of manual labour. A vertical shaft had to be dug about 2m from the entrance so that it intersected the burrow as it levelled off. This required digging a hole almost 2m deep, which was

large enough for the cameraman and his equipment to fit in. It was very difficult to make the floor of the burrow level enough for Ardie to walk on, and so the aluminium sled was used again, dragging him into and out of the shot.

Because it had not been possible to film Ardie and Timmy walking together, Lynne decided to take some blue screen footage of Ardie, so that he could be superimposed onto a different background in the post-production stage if necessary. This involved hanging a blue sheet that ran onto the floor, and filming Ardie in front of it. Ardie was filmed walking from various angles and with different lighting positions to match the possible lighting conditions that the background onto which he would be superimposed might have.

What made it even more difficult for Ardie to walk was that he had to have a mock-up of a mini video camera attached to his head. The 'lipstick' camera is a 3CCD digital video camera approximately 40x30x20mm (plus lens) in size, and was used to film in tight spaces (such as inside the burrow) and for Ardie's POV shots (Point Of View). This would give the effect of seeing things from his perspective. The real camera was mounted to the end of an inverted monopod for filming, and not on Ardie, as the camera was very expensive and could have been damaged. Ardie was filmed with the camera mock-up camera on his head, to make it appear as if the POV shots had been taken by him. It was only decided to do this while out on location, and so the author did not have time to adjust his walking sequence for the new weight, and had to build the camera out there. The body of the mock-up as made from aluminium, and one of the real lenses was attached to it. This added weight to the front of Ardie, affecting his balance. In order to restore the balance, the adjustable weight was shifted to the rear, and an extra battery pack was also added to the rear. Not much more weight could not be added, as this would place strain on the servos, and so the balance was not fully restored.

Although Ardie didn't quite meet the requirements as far has his walking was concerned, when it came to the head and facial movements, he did very well. By using the control box, the author was able to produce realistic movements with the head and eyes, and could make him interact with what was going on around him. For example, when the film crew were demonstrating their equipment, or when Timmy was digging in front of him, Ardie was able to follow the action with his head, and blink his eyes from time to time. Also, the mouth movement made it appear as if he was talking to the crewmembers, asking how their equipment worked etc.

Another aspect of the Ardie that it had not been possible to test properly before going out on location was his durability. It was realised that once out on location, if anything went wrong it would be very difficult to repair, and could jeopardise the filming. While building Ardie, the author had this in mind all the time, and did try to make him as robust as possible. This is why it was decided to rebuild the legs using aluminium rather than Perspex, and why aluminium was used for the 'rib cage'. This design strategy seemed to have paid off, as no mechanical problems were encountered with the legs even though Ardie was subjected to some pretty rough conditions (and even managed to survive falling from a desk onto the floor before a demonstration at UCT).

One area where a few problems did arise was with the servos. Although high torque servos where used for the legs, these were put under much strain while walking, and were operating at close to their maximum rated torque. In the manoeuvre where Ardie sits up, the servos for the hind legs are put under even more strain, as they take the full weight of the robot. It was while doing this manoeuvre on uneven ground that the gears in on of the servos stripped, disabling the leg. Spare gears or a spare high torque servo were not available on location, so a complete repair could not be done. A spare standard servo was available though, and the author managed to use this to solve the problem. The damaged servo had been used to drive the upper part of the leg directly. The lower part is driven by a servo that is first geared down, so this servo is under less strain. The damaged servo was swapped for the one that had been driving the lower part of the leg, and the spare standard servo was then used to then drive the lower part. It was found that the standard servo did actually produce enough torque to perform this function, and the leg was back in action.

Another incident that occurred was with the mini servo that drove the head from side to side. At one point while filming Ardie and Timmy together, Timmy reversed into Ardie and knocked his head. This pushed his neck to the side, stripping the gears in the servo. Luckily this did not render the servo totally useless – some side-to-side movement was still possible, but to a limited degree. A spare set of gears was not available, but luckily this was on the last night of filming, and so most of the shots had been taken. The servo was easy to repair with a replacement gear when the author returned to Cape Town.

While on location, various other modifications and improvements had to be made as the filming progressed. Also, the role that the Director saw Ardie playing in the movie changed somewhat as she saw what his capabilities were. Although a fur coat had been made for him, Ardie looked more impressive with his coat off, and so most of the filming was done with him 'naked'. Also, because he was now meant to be seen as a robot, it did not matter if there was a visible tether running to him. This made it easy to control him with the control box and keep the battery connected permanently. Below are some photographs taken while filming on location:

Figure 54: The Author with Ardie and Timmy

Figure 55: Filming Ardie and Timmy together

Figure 56: Timmy opening a termite mound

Figure 57: Timmy feasting on termites.

# 5. Conclusions

This report has shown how an animatronic aardvark was built, programmed and controlled while being filmed for a National Geographic wildlife film. The animatron met the requirements that were set out by the Film Producer, by being able to walk, move its head, eyelids and mouth. Although a realistic dynamically stable gate could not be achieved for the animatron, a satisfactory compromise was reached by developing a statically stable gate. It was possible for a single operator to control the head, eyelid and mouth movements using the control box, to give the impression that the animatron was interacting with its environment. Although the animatron was limited to walking over smooth terrain, the Film Producer was able to film most of the required shots, and was very happy with the animatron's performance.

# 6. Future Work

Since the Film Producer is interested in using the animatron for future productions, additional work will be done to improve its performance. The walking sequence will be revised with the aim of making it more stable and allowing the animatron to walk on rougher terrain.

For research purposes, the author also intends to convert the animatron into an autonomous mobile robot, by adding sensors, and modifying the control program so that it can use the input from the sensors to make decisions for itself. As an example, infra red proximity sensors could be interfaced to the Handy Board, and these could be used to detect obstacles. The animatron could then be programmed to roam around in an environment with obstacles, and avoid these obstacles. For this to be possible, additional walking capabilities would also need to be added, such as turning and walking backwards.

Since most of the mechanics and electronics of the animatron are left exposed and visible, it serves as a good demonstration as to how mechanical and electrical engineering can be put to use. The animatron has been used for demonstration purposes on numerous occasions, both to school children and university students. The author intends to keep Ardie involved in such demonstrations in the hope that other students will become interested in the filed of animatronics and robotics.

# References


1. Charles Fraser, John Milne (1994) Integrated Electrical and Electronic Engineering for Mechanical Engineers (McGraw Hill)
2. John Iovine (1997)– Robots, Androids and Animatrons (McGraw Hill)
3. Gordon Mccomb (1987) – Robot Builder's Bonanza (McGraw Hill)
4. Karl Lunt (1999) Build Your Own Robot! (A K Peters)
5. James M. Conrad, Jonathan W. Mills (1998) Stiquito – Advanced Experiments With a Simple and Inexpensive Robot (IEEE Computer Society)
6. Design Engineers Guide to DC Stepping Motors – Superior Electric (1976)
7. Stepper Motor Applications Guide – Digiplan (1991)
8. Jeppe Mikkelsen (1998) A Machine Vision System Controlling a Lynxmotion Robot Along A Path – BSc thesis, Mech Eng UCT
9. Slo-Syn AC Synchronous, Gearmotors and DC Stepper Motors catalogue
10. Digiplan Electronic Motion Control Handbook – Digiplan
11. Fred G. Martin (2000), The Handy Board Technical Reference.
12. Fred G. Martin (1997), Assembling the Handy Board
13. Fred G. Martin (1996), The Interactive C Manual for the Handy Board

Online references:

14. http://63.249.196.254/pmDisneyAnimatronics.html
15. Handy Board website: http://handyboard.com/
16: OOpic website: http://www.oopic.com/

# Appendix A: The Author's Background

Robotics is a field that has interested the author since 1996 - originally in the basic form of being able to control mechanical devices from a computer, and to feed input from sensors into the computer. The 4th year thesis project for the author's BSc in Mechanical Engineering at UCT involved these two - an automated warehouse packaging system was developed and a working model was built to test the concept. His interest in the field grew, and after travelling Europe and working in London for two years, he decided that he would like to do an MSc in robotics. After researching which universities offered such a course, the author travelled to the United States to visit a few of the campuses. He was most impressed with the facilities and the research being conducted at the University of Southern California in Los Angeles, and decided that he would like to study there. In order to apply for the MSc at USC, it was necessary to have a background in computer science (the course is a MSc in Computer Science, specializing in Robotics), so the author returned to UCT and enrolled in first and second year computer science concurrently in 2000. He decided to make the most of his time at UCT, so looked into the possibility of doing a MSc in Mechanical Engineering. It was discovered that Professor Andrew Sass had become interested in the field of robotics, and was enthusiastic for the author to do an MSc under his supervision.

From the outset, the aim of the author's thesis was to research various aspects of robotics, and to create a resource of robotic related information for future students to use. It was also hoped to build a small autonomous mobile robot that could serve as a test-bed and working example of the various things that had been researched. During 2000, information was collected and an HTML based resource was developed. A small mobile robot was also built. The primary function of this robot was to test the capabilities of a microprocessor board that had been ordered from the United States (the Handy Board). The robot (Tina) had limited space for sensors and batteries, and so the author intended to build a bigger version that could carry all the sensors necessary to demonstrate what had been researched thus far.

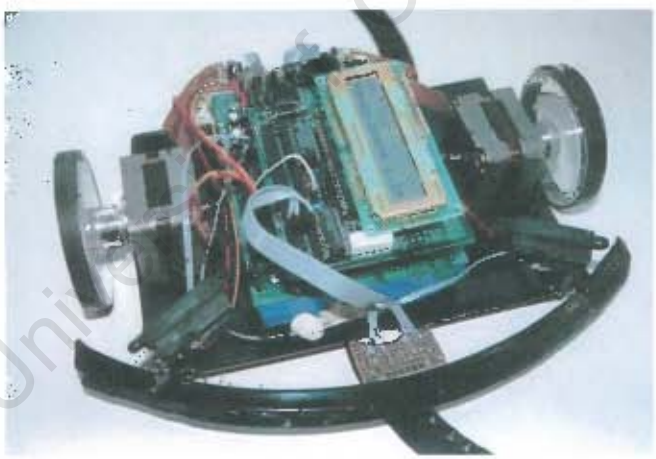


**Figure 58: Tina – the autonomous mobile robot**

Just before construction on the bigger version of Tina began (end of January 2001), the author was approached by Lynne Richardson of Africa Wildlife Films with an interesting proposal. Africa Wildlife Films is a Cape Town based company that produces wildlife films - they have produced films for the BBC and Discovery Channel, as well as National Geographic, for whom they were in the process of producing a film on aardvarks. To research and film the aardvark, various modern technologies were used, such as radio tracking devices, and Ground Penetrating Radar (to scan the aardvark burrows). The producer decided that there would be sufficient interesting material to put together a 20-minute "making of" movie, to explain how all these technologies were used. She decided it would be a good idea to have an animatronic aardvark character to tell the story of how the movie was made, and through who's eyes the viewer would see this being done. Professor Sass agreed that building the animatron would be a suitable exercise for a Masters Thesis, and so the author undertook this challenge instead of building the wheeled robot.

# Appendix B : Communication Between Handy Board and SSC

In order to control 14 servos, it was necessary to use a Mini Serial Servo Commander in conjunction with the Handy Board. The mini SSC is usually controlled from a PC's serial port. The SSC receives commands via serial communication, and these commands have the following format:

<Sync> <Servo number> <Position>

First the number 255 is sent to indicate that a command is being sent, then a number between 0 and 7 is sent to indicate which servo the command must be applied to, and finally, a number representing the position that this servo must be moved to is sent. This number ranges from 0 to 255. Once the command has been received, the SSC will move the servo into position, and then hold it there until a new position command is given. To move a number of servos at the same time, a number of commands in the format given above are sent sequentially, and the SSC carries them out as it receives them – the first servo movement does not have to be completed before the next is initiated.

When the Handy Board is used to communicate with the SSC, it sends commands in the form given above. The code for using the Handy Board's digital input pins as a serial communication port was written by Fred Martin and is hidden within a library file. Once this library has been loaded onto the Handy Board, the following lines of code are used to send a command to the SSC:

```
pa7i9600(255);
pa7i9600(servo);
pa7i9600(pos);
```

'pa7i9600()' is a function that sends whatever is given as a parameter to the SSC. 'servo' and 'pos' are variables used to give the servo number and position.

On the hardware side, PA7 of the handyboard (digital input 9) is connected to the serial input pin on the SSC, and the SSC ground pin is connected to the handyboard's ground. As well as an RJ11 jack, the SSC also has a pair of riser pins that can be used for serial communication. These pins were used to connect it to the handyboard.

# Appendix C: Extract from Handy Board datasheet

## 1 Specifications

The Handy Board features:

- 52–pin Motorola 6811 microprocessor with system clock at 2 MHz.
- 32K of battery-backed CMOS static RAM.
- Two L293D chips capable of driving four DC motors.
- 16 × 2 character LCD screen.
- Two user-programmable buttons, one knob, and piezo beeper.
- Powered header inputs for 7 analog sensors and 9 digital sensors.
- Internal 9.6v nicad battery with built-in recharging circuit.
- Hardware 38 kHz oscillator and drive transistor for IR output and on-board 38 kHz IR receiver.
- 8-pin powered connector to 6811 SPI circuit (1 Mbaud serial peripheral interface).
- Expansion bus with chip selects allows easy expansion using inexpensive digital I/O latches.
- Board size of 4.25 × 3.15 inches, designed for a commercial, high grade plastic enclosure which holds battery pack beneath the board.

# 2 Ports and Connectors

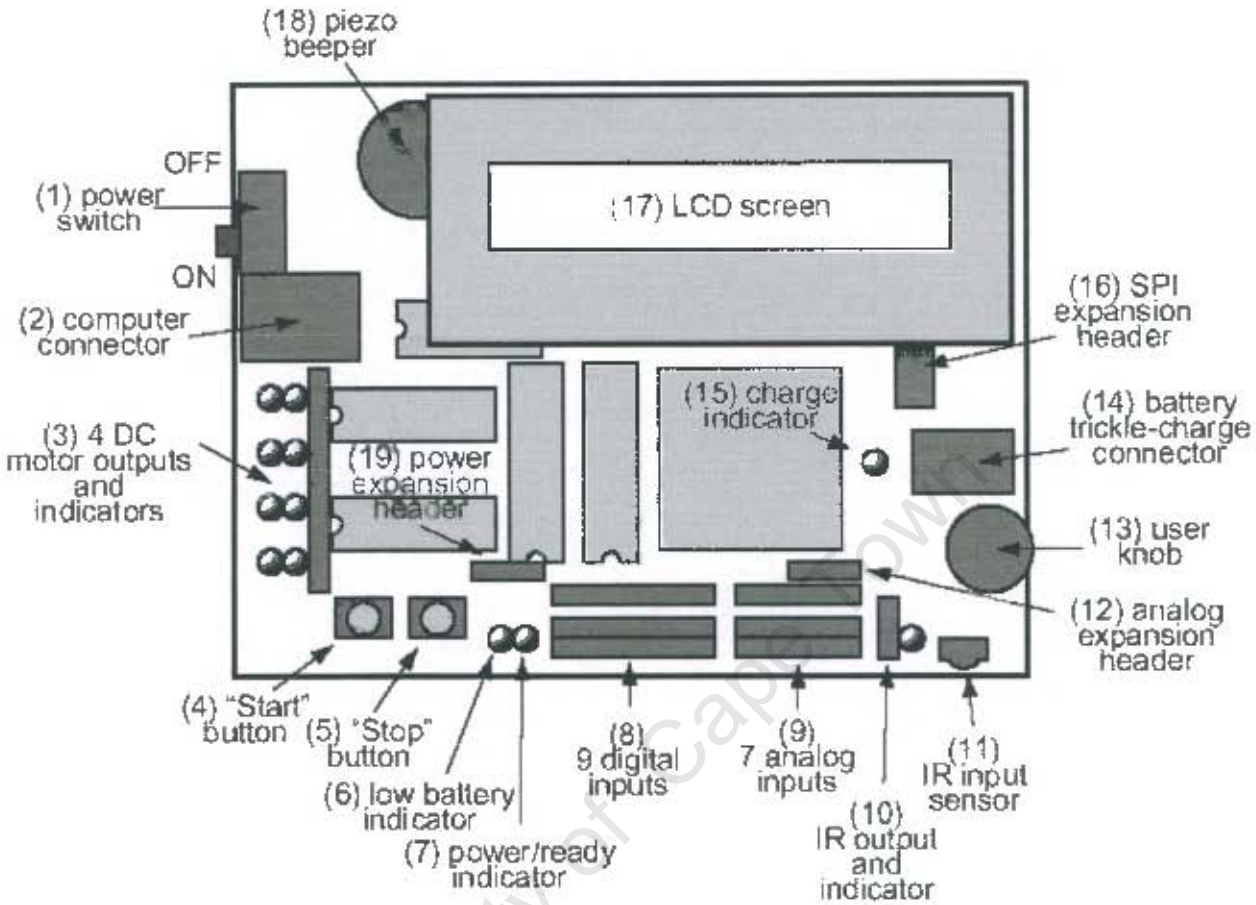



Figure 1: Labeled Handy Board Diagram

Figure 1, above, shows a labeled view of the Handy Board's ports, connectors, inputs, and outputs. In the following, each of these is briefly described.

1. **Power Switch.** The power switch is used to turn the Handy Board on and off. The Handy Board retains the contents of its memory even when the board is switched off.

2. **Computer Connector.** Via this RJ11 connector, the Handy Board attaches to a desktop computer (using the separate Interface/Charger Board).

3. **4 DC Motor Outputs and Indicators.** The Handy Board's four motor outputs are located at this single 12-pin connector. Each motor output consists of three pins; the motor connects to the outer two pins and the center pin is not used. Red and green LEDs indicate motor direction. From top to bottom, the motor outputs are numbered 0 to 3.

4. **Start Button.** The Start button is used to control the execution of Interactive C programs. Also, its state may be read under user program control.

5. **Stop Button.** The Stop button is used to put the Handy Board into a special bootstrap download mode. Also, its state may be read under user program control.

6. **Low Battery Indicator.** The red Low Battery LED lights for a brief interval each time the Handy Board is switched on. If this LED is on steadily, it indicates that the battery is low and that the CPU is halted.

7. **Power/Ready Indicator.** The green Power/Ready LED lights when the Handy Board is in normal operation, and flashes when the Handy Board is transmitting serial data. If the board is powered on and this LED is off, then the Handy Board is in special bootstrap mode.

8. **9 Digital Inputs.** The bank of digital input ports is here. From right to left, the digital inputs are numbered 7 to 15.

9. **7 Analog Inputs.** The bank of analog input ports is here. From right to left, the analog inputs are numbered 0 to 6.

10. **IR Output and Indicator.** The infrared output port is here. The red indicator LED lights when the output is enabled.

11. **IR Input Sensor.** The dark green-colored infrared sensor is here.

12. **Analog Expansion Header.** The analog expansion header is a 1×4 connector row located above analog inputs 0 to 3.

13. **User Knob.** The user knob is a trimmer potentiometer whose value can be read under user program control.

14. **Battery Trickle-Charge Connector.** The battery charge connector is a coaxial power jack to accept a 12 volt signal for trickle-charging the Handy Board's internal battery.

15. **Charge Indicator.** The yellow charge indicator LED lights when the Handy Board is charging via the coaxial power jack.

16. **SPI Expansion Header.** The SPI expansion header is a 2×4 pin jack that allows connection with the 6811's *serial peripheral interface* circuit. See the CPU and memory schematic diagram for a pin-out of this connector.

17. **LCD Screen.** The Handy Board is provided with a 16×2 LCD screen which can display data under user control.

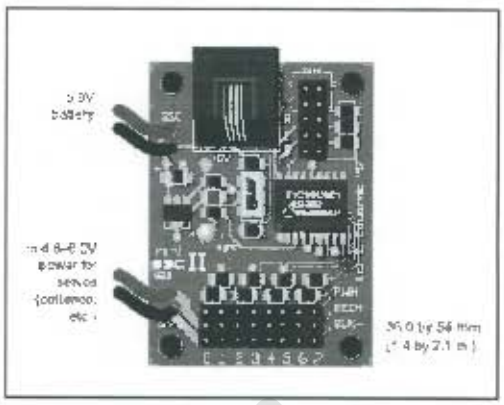
18. **Piezo Beeper.** The Handy Board has a simple piezo beeper for generating tones under user control.

19. **Power Expansion Header.** The power expansion header is a 1×4 pin jack that provides access to the unregulated motor power and ground signals.

## Appendix D: Mini Serial Servo Commander Datasheet

**Data Sheet**

# Serial Servo Controller (SSC) for R/C Servos



*Use a computer's serial port to control standard R/C servos for robotics, automation, and animatronics. Mini SSCs accept serial input at 2400 or 9600 bps, output eight rock-steady channels of servo-control signals.*

### Easy Motion Control

R/C servos are the positioning motors used in radio-control cars, boats, and planes. They are popular for small-scale robotics, automation, and special efforts because of their low cost, small size, and high precision.

Our Mini SSC II allows you to control up to eight servos through a computer's serial port using simple instructions at 2400 or 9600 bps. Program your computer to send three bytes—

<255 (sync)> <servo#> <position>

...and the Mini SSC moves that servo to the specified position, and keeps it there until told to change the position. Two Mini SSCs can share the same serial port to control up to 16 servos, or, you may special order Mini SSCs with higher servo numbers and control up to 255 servos. Contact us for details.

Most Mini SSC applications require you to write your own programs for the host computer you plan to use. The manuals include examples in BASIC for Stamp® microcontrollers and PCs running DOS. But you can use any combination of hardware and software that can drive an RS-232 serial port. If you don't want to program, visit our web page and follow the links to third-party software packages for Windows (`www.seetron.com/ssc.htm`).

### Neat Features

The Mini SSC II is an assembled and tested module that features a convenient phone-style jack for serial input, three-conductor servo connectors, power/sync LED to verify correct operation, and switchable range/resolution from 90° at 0.36° per unit or 180° at 0.72°/unit.

### Ordering Information

Assembled Mini SSC II (SSC-ASD2) ..... 44.00

PC (DB9) serial cable for Mini SSC II (SSC-CBL) ..... 6.00

Scott Edwards Electronics, Inc.
1939 S. Frontage Road, Suite F, Sierra Vista, AZ 85635 USA
ph: 520-459-1802 • fax: 520-459-0623 • www.seetron.com

*Table 1. Basic specifications*

| | |
|---|---|
| Power requirements (Mini SSC) | 7 to 15 Vdc @ 10mA |
| Power requirements (servo) | 4.8 to 6.0 Vdc (current varies) |
| Serial input connector | header posts (both); 6p4c phone jack (II only) |
| Serial input | RS-232, or inverted TTL/CMOS, 9600 or 2400, N81 |
| Operating temperature | 0° to 50°C (32° to 122°F) |
| Servo output connector | 3-pin header, 0.1"spacing: (PWM)(+V)(GND) |
| Pulse frequency | ~60 Hz |
| Pulsewidth range (normal) | 1.0 to 2.0 ms |
| Pulsewidth range (Mini SSC II, "R" jumper on) | 0.5 to 2.5 ms |
| Pulsewidth at startup (centered) | 1.5 ms |
| Pulsewidth resolution (normal) | 4 µs |
| Pulsewidth resolution (Mini SSC II, "R" jumper on) | 8 µs |
| Servo numbers ("I" jumper off) | 0—7 |
| Servo numbers ("I" jumper on) | 8—15 |

*Table 2. Program Examples in BASIC*

The following code examples move servo number 5 to position 200. For more detailed examples, see the user manuals at www.seetron.com.

**BASIC Stamp I**

```
' No jumpers at "I" or "B"
SEROUT 0,N2400,(255,5,200)
```

**BASIC Stamp II**

```
' No jumper at "I"; install jumper at "B"
SEROUT 0,$4054,[255,5,200]
```

**QBASIC/QuickBASIC on PC running DOS**

```
' No jumper at "I"; install jumper at "B"
OPEN "COM1:9600,N,8,1,CD0,CS0,DS0" FOR OUTPUT AS #1
PRINT #1, CHR$(255);CHR$(5);CHR$(200);
```

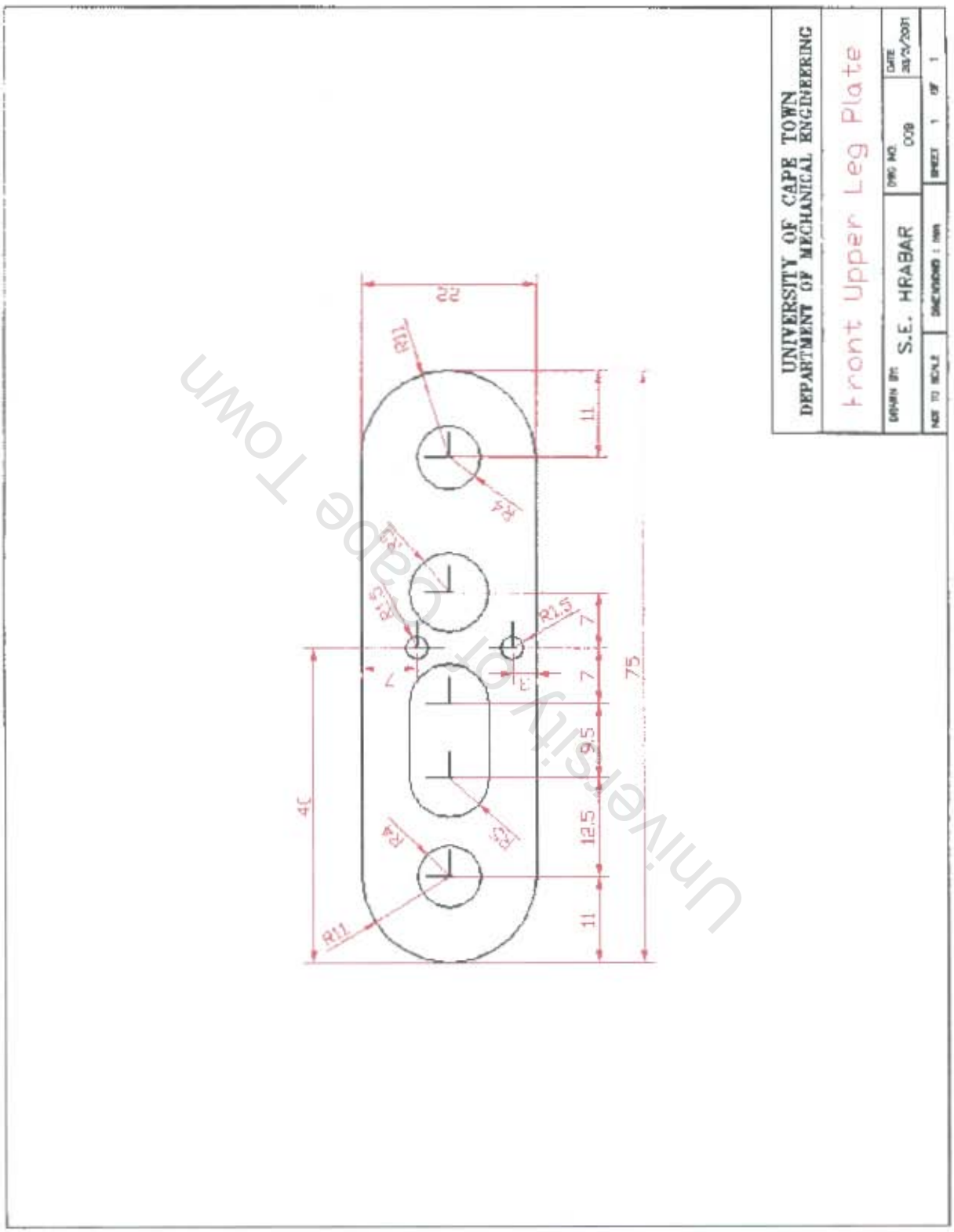
*Table 3. Sources for Servos, Robotic Kits and Related Items*

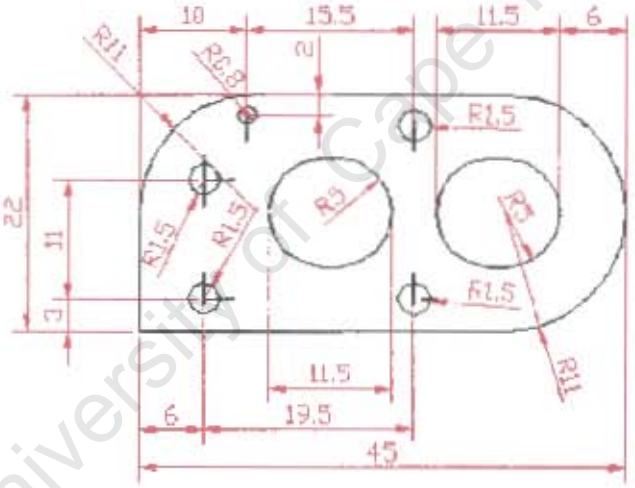
**Lynxmotion (robotic kits, servos)**
ph: 309-382-1816 • Net: www.lynxmotion.com

**Tower Hobbies (servos)**
ph: 1-800-637-4989 or 217-398-3636 • Net www.towerhobbies.com

**CK Design Tech (heavy-duty servos)**
ph: 805-522-3750 • E-mail: ckdsgn@aol.com

# Appendix E: Drawings for Leg Construction



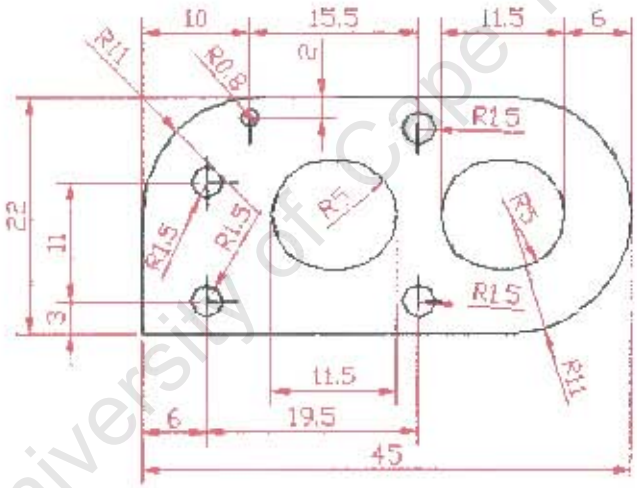





UNIVERSITY OF CAPE TOWN
DEPARTMENT OF MECHANICAL ENGINEERING

Front Lower Leg Plate

| DRAWN BY: S.E. HRABAR | DWG NO. 008 | DATE 29/3/2001 |
|---|---|---|
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 |

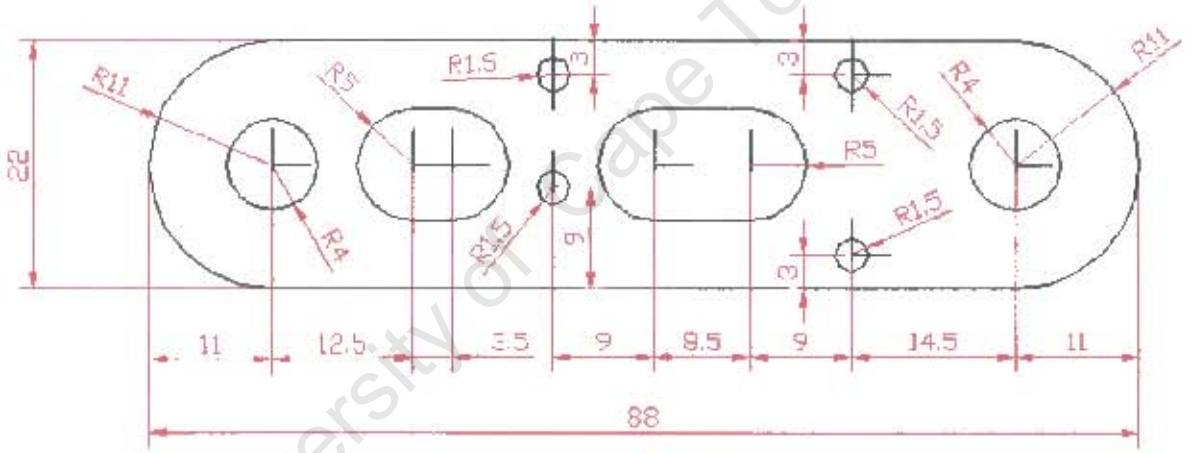





UNIVERSITY OF CAPE TOWN
DEPARTMENT OF MECHANICAL ENGINEERING

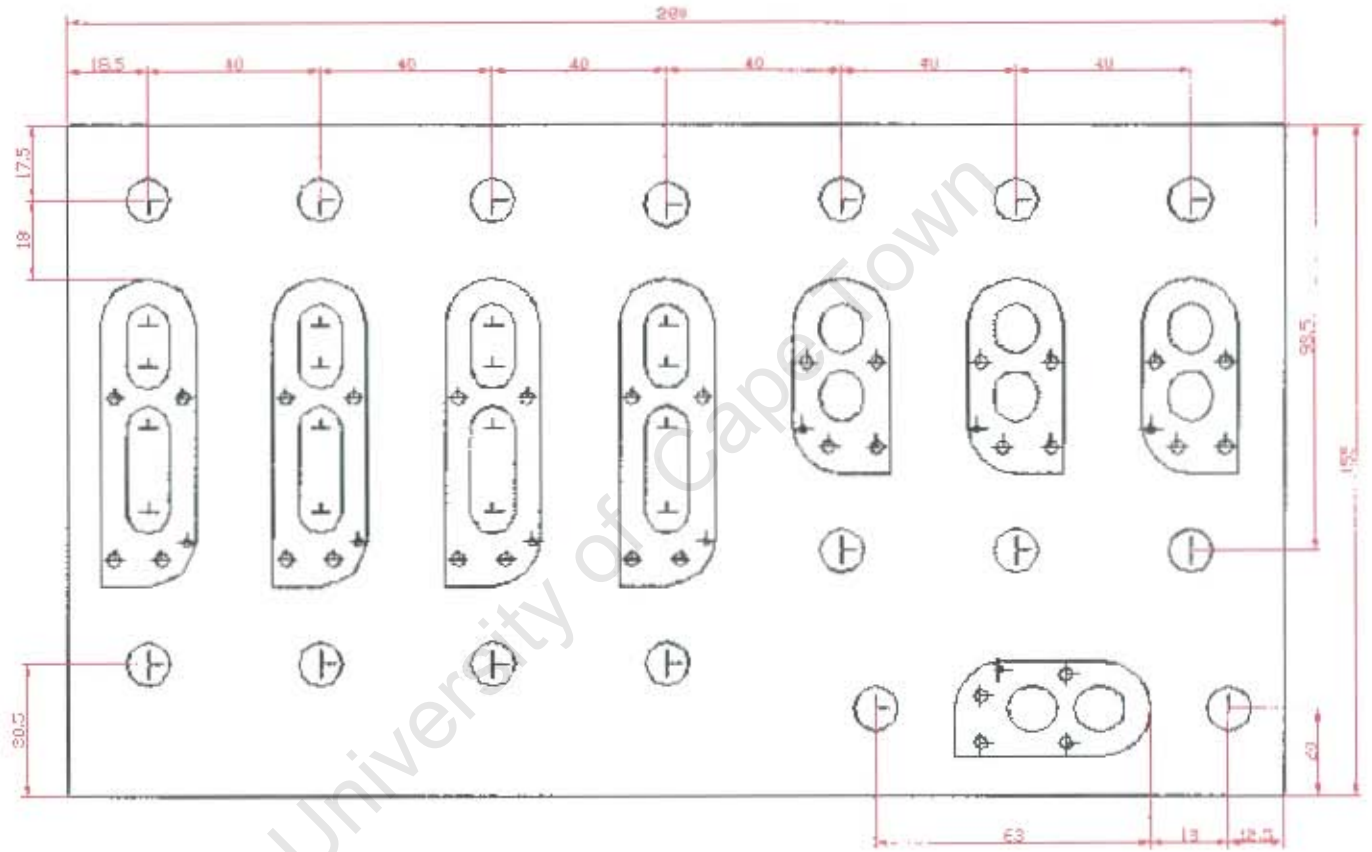
Front Lower Leg Plate

| DRAWN BY: S.E. HRABAR | DWG NO. 008 | DATE 23/3/2001 |
|---|---|---|
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 |



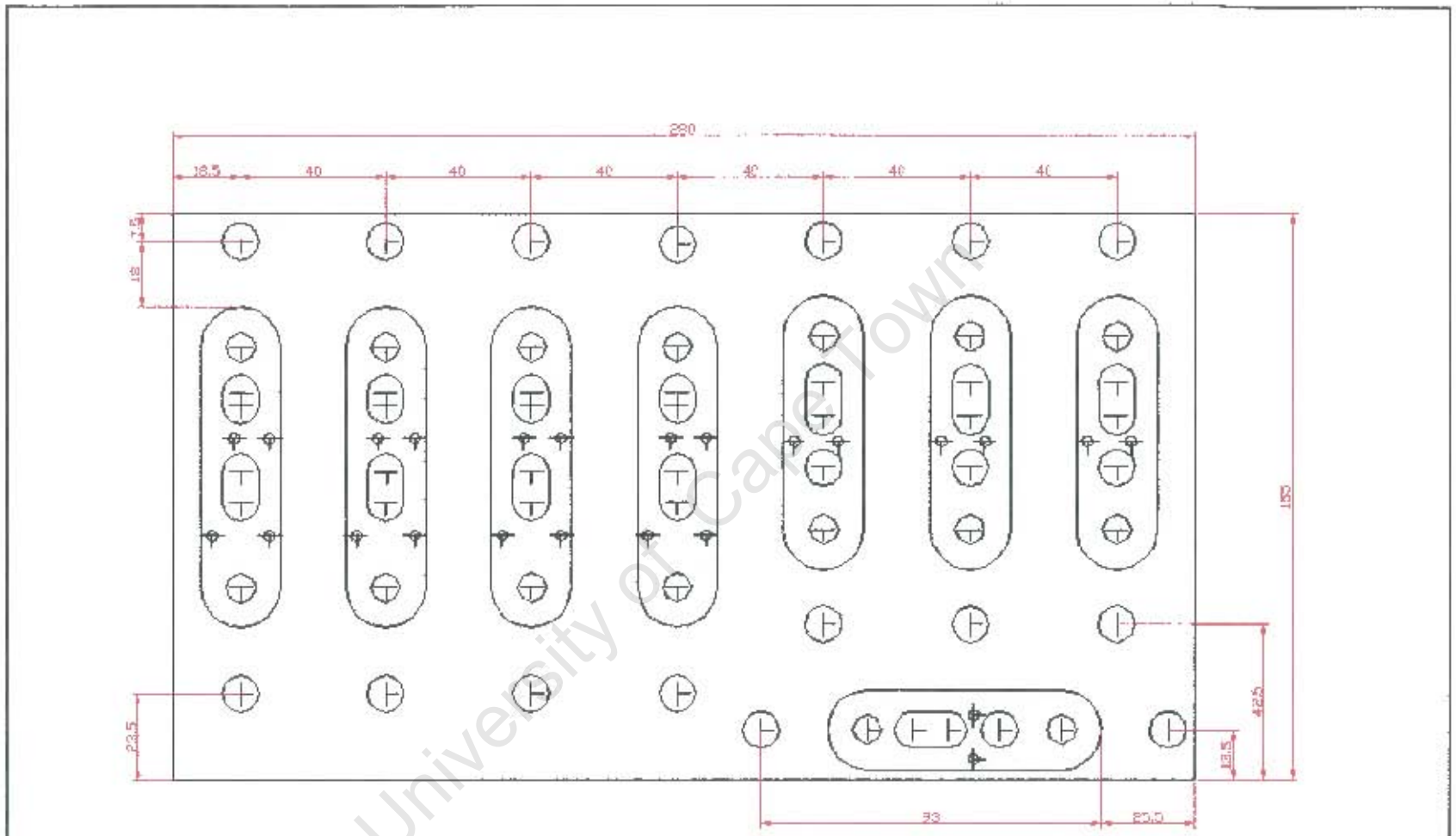



| UNIVERSITY OF CAPE TOWN<br>DEPARTMENT OF MECHANICAL ENGINEERING | | | |
|---|---|---|---|
| Upper Rear Leg Plate | | | |
| DRAWN BY: S.E. HRABAR | | DRG NO. 006 | DATE 20/3/2007 |
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 | |



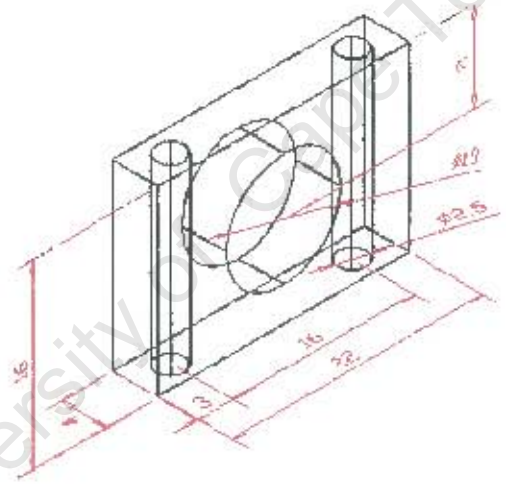



| UNIVERSITY OF CAPE TOWN<br>DEPARTMENT OF MECHANICAL ENGINEERING | | | |
|---|---|---|---|
| Profile For Machining Leg Plates | | | |
| DRAWN BY: S.E. HRABAR | | DWG NO. 005 | DATE 20/3/2001 |
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 | |



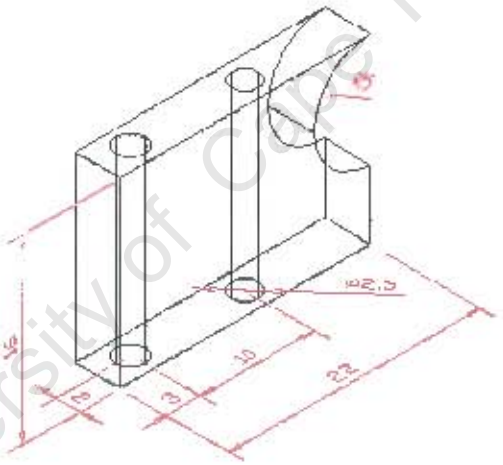



| UNIVERSITY OF CAPE TOWN<br>DEPARTMENT OF MECHANICAL ENGINEERING | | | |
|---|---|---|---|
| Profile for machining leg plates | | | |
| DRAWN BY: S.E. HRABAR | | DWG NO. 004 | DATE 20/3/2001 |
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 | |






| UNIVERSITY OF CAPE TOWN<br>DEPARTMENT OF MECHANICAL ENGINEERING | | | |
|---|---|---|---|
| Leg Cross Member | | | |
| DRAWN BY: S.E. HRABAR | | DWG NO. 003 | DATE: 20/3/2001 |
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 | |

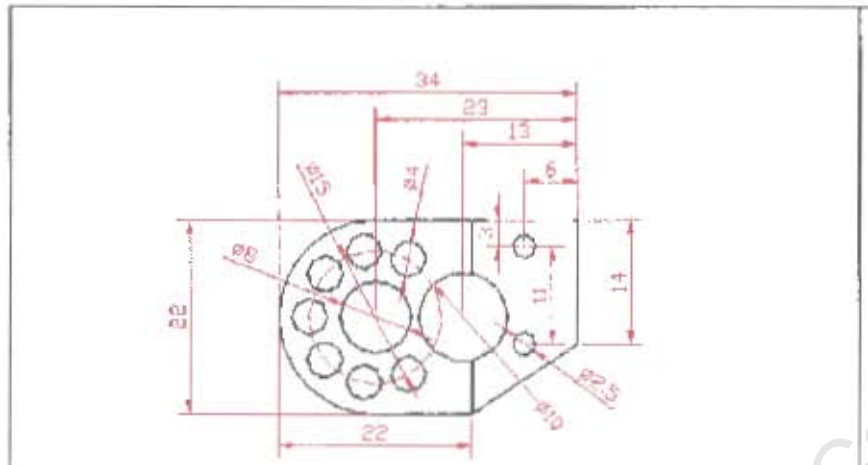


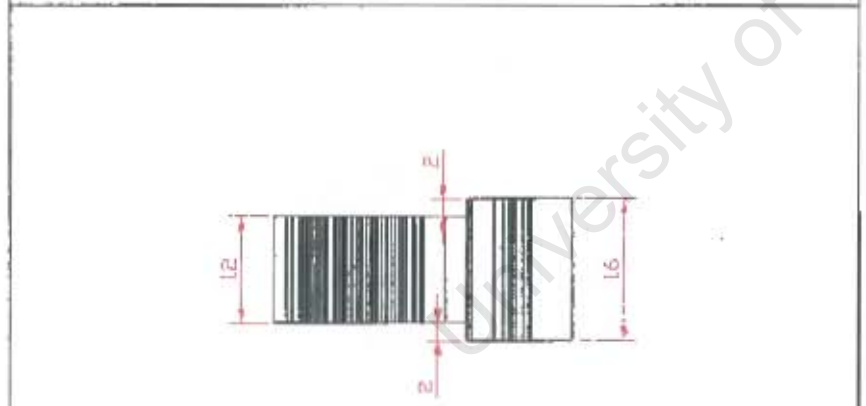


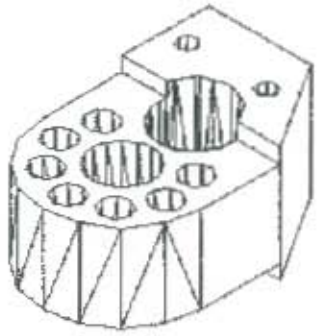

UNIVERSITY OF CAPE TOWN
DEPARTMENT OF MECHANICAL ENGINEERING

Leg Cross Member

| DRAWN BY: S.E. HRABAR | DWG NO. 002 | DATE 30/5/2001 |
|---|---|---|
| NOT TO SCALE | DIMENSIONS : mm | SHEET 1 OF 1 |



UNIVERSITY OF CAPE TOWN
DEPARTMENT OF MECHANICAL ENGINEERING

Aardvark Knee Joint

| DRAWN BY: S.E. HRABAR | DWG NO. 001 | DATE 28/3/2001 |
|---|---|---|
| SCALE 1 : 1.75 | DIMENSIONS mm | SHEET 1 OF 1 |

# Appendix F: Program Listing for Handy Board Library File.

The code listed below was written by Fred Martin, and contains the basic routines used by the Handy Board. This code is loaded into the Handy Board's memory along with the aardvark control code.

```
/*  for Handy Board v1.2 */
/*
  maintained by Fred Martin (fredm@media.mit.edu)

  VERSION HISTORY

  June 12, 1998              fredm
moved analog and digital to separate file, "hbsensor.c", to make room for substituting these functions with expbd functions.

  March 1, 1998              fredm
  re-wrote _raw_analog() in assembly

  May 13, 1995               fredm
  imported from lib_r22.c by Fred Martin and Randy Sargent added random() lib function
*/

/*******************/
/* TIME PRIMITIVES */
/*******************/

/************************************/
/* location of various time stuff: */
/* 0x14: time in milliseconds      */
/************************************/

void reset_system_time()
{
   pokeword(0x14, 0);
   pokeword(0x12, 0);
}
/*
   mseconds() is now a C primitive that returns type <long>
*/
/*  returns time since reset or reset_system_time in seconds */
float seconds()
{
   return ((float) mseconds()) / 1000.;
}
void sleep(float seconds)
{
   msleep((long)(int)(seconds * 1000.));
}
```

```
void msleep(long msec)
{
   long end_time= mseconds() + msec;

   while (1) {
         /* if the following test doesn't execute at least once a second,
            msleep may not halt */
         long done= mseconds()-end_time;
         if (done >= 0L && done <= 1000L) break;
   }
}

void beep()
{
   tone(500., .1);
}

/*  1/2 cycle delay in .5us goes in 0x26 and 0x27  */
void tone(float frequency, float length)
{
   set_beeper_pitch(frequency);
   beeper_on();
   sleep(length);
   beeper_off();
}

void beeper_on()
{
   bit_set(0x1020, 0b00000001);
   bit_set(0x1022, 0b00001000);
}

void beeper_off()
{
   bit_clear(0x1022, 0b00001000);
   bit_clear(0x1020, 0b00000001);
   bit_clear(0x1000, 0b00001000); /* turn power to spkr off */
}


void set_beeper_pitch(float frequency)
{
   pokeword(0x26, (int)(1E6 / frequency));
}
```

```
/*********************************************************************** */
/*                                                                      */
/* MOTOR PRIMITIVES                                                     */
/*                                                                      */
/*      fd(n)     sets motor n to full on in the green direction    */
/*      bk(n)     sets motor n to full on in the red direction      */
/*      motor(n, s)       sets motor n on at speed s;              */
/*                        s= 100 is full on green,                 */
/*                        s= -100 is full on red,                  */
/*                        s= 0 is off                              */
/*      off(n)            turns off motor n                        */
/*                                                                      */
/*      alloff()  turns off all motors                 */
/*      ao()              turns off all motors                   */
/*                                                                      */
/*                                                                      */
/*      motors are numbered 0 through 3.                         */
/*********************************************************************** */

/************************************************** */
/* location of various motor stuff for PWM routine */
/*                                  */
/* poke byte into "motor" to be output by PWM          */
/* motor:  0x0e                             */
/*     low bits of "motor" determine directions;  */
/*     high bits determine on/off state.            */
/*                                  */
/* internal speeds are bit masks                */
/*                                  */
/* speed0:  0x22                            */
/* speed1:  0x23                            */
/* speed2:  0x24                            */
/* speed3:  0x25                            */
/************************************************** */

int _motor_speed[]= {7, 7, 7, 7, 7, 7};          /* init to full on */int _speed_table[]= {
                        0b00000000, /* speed 0 */
                        0b00010001,
                        0b01001001,
                        0b01010101,
                        0b01010111,
                        0b01110111,
                        0b01111111,
                        0b11111111 /* speed 7 */
                       };

void fd(int motor)
{
   _set_motor(motor, 0, 7);
}

void bk(int motor)
{
   _set_motor(motor, 1, 7);
}
```

```
void off(int motor)
{
   bit_clear(0x0e, 1 << (4 + motor));
}

void alloff()
{
   poke(0x0e, 0b0000000);
}

void ao()
{
   alloff();
}

void motor(int m, int speed)
{
 if (speed>100) speed=100;
 if (speed<-100) speed=-100;
 if (speed >= 0)
   _set_motor(m, 0, (speed + 3) / 14);
 else
   _set_motor(m, 1, (-speed + 3) / 14);
 }

void _set_motor(int motor, int dir, int speed)
{
   bit_set(0x0e, 1 << (4 + motor));  /* turn motor on */
   if (dir)
         bit_set(0x0e, 1 << motor); /* set direction for backward */
   else
         bit_clear(0x0e, 1 << motor);          /* set dir for forward */
   _motor_speed[motor]= speed;
   _set_motor_speeds();
}

void _set_motor_speeds()
{
   int speed01= (_speed_table[_motor_speed[0]] << 8)
     + _speed_table[_motor_speed[1]];
   int speed23= (_speed_table[_motor_speed[2]] << 8)
     + _speed_table[_motor_speed[3]];

   pokeword(0x22, speed01);
   pokeword(0x24, speed23);
}
```

```
/**********************/
/*** SENSOR INPUTS ***/
/**********************/

int stop_button()
{
   return ! (0x40 & (peek(0x7fff)));
}

int start_button()
{
   return ! (0x80 & (peek(0x7fff)));
}

void start_press()
{
   while (!start_button());
   while (start_button());
   beep();
}

void stop_press()
{
   while (!stop_button());
   while (stop_button());
   beep();
}

int knob()
{
   return _raw_analog(7);
}

/*******************************/
/*** Multi-Tasking Support ***/
/*******************************/

/*  gives process that calls it 256 ticks (over 1/4 sec)
   more to run before being swapped out

   call repeatedly to hog processor indefinitely  */
void hog_processor()
{
   poke(0x0a, 0);
}

/*  defer is now a C primitive  */
```

```
/*********************************/
/*** System Interrupt Control ***/
/*********************************/
/*
   These functions allow you to turn on and off various features
   controlled by the system interrupt routines.  The more features
   you turn off, the faster your code will run.

   On reset, the features have the following state:
     pulse width modulation ON
     infrared decoding     ON
     LCD printing          ON
     quad shaft decoding   OFF
     IR tranmission              OFF

   This uses approx. 30% of total CPU time.

 Approximate benchmarks:

x   Feature                  % of CPU
    -------          --------
    PWM                        3
    IR decode                 11
    LCD printing (active)  8
    LCD printing (inactive) 1
    quad shaft decode          5
    IR transmission            1

*/
/*
  pulse width modulation control:
  if off, all motors run at full speed
  if on,  speed bytes are used to determine motor speed
*/
void system_pwm_on() {bit_set(0x39, 0b00000100);}
void system_pwm_off() {bit_clear(0x39, 0b00000100);}

/*
  printing to the LCD
  WARNING:  printf's will wedge once the print buffer becomes full
    if system printing is disabled  */
void system_print_on() {bit_set(0x39, 0b00000001);}
void system_print_off() {bit_clear(0x39, 0b00000001);}



/* random numbers from peeking at 2 MHZ system clock */
/* input from 2 to 32767 */
int random(int mod)
{
   return (peekword(0x100e) & 0x7fff) % mod;
}
```

# Appendix G: Program Listing for Expansion Board Routines

The code listed below was written by Fred Martin, and contains the routines for communicating with the Handy Board Expansion Board. This code is loaded into the Handy Board's memory along with the aardvark control code.

```
/* expsens.c
  maintained by Fred Martin (fredm@media.mit.edu)
  VERSION HISTORY:     June 12, 1998                fredm
*/

int analog(int port)
{
        if (port < 2) {
                printf("Port is in use  by expansion bd\n");
                beep();
                return -1;
        } else if (port < 7) {
        return _raw_analog(port);
   } else if (port < 16) {
        return 255 * !digital(port);
   } else if (port < 24) {
        return _exp_analog((port-16)<<8);
   } else if (port < 32) {
        return _exp_analog(((port-24)<<8)+1);
   } else {
        printf("Analog port out of range\n");
        beep();
        return -1;
   }
}
int digital(int port)
{
   if (port < 7) /* analogs */
        return analog(port) < 128;
   if (port == 7) /* TIC1 */
        return !(peek(0x1000) & 1);
   if (port == 8) /* TIC2 */
        return !(peek(0x1000) & 2);
   if (port == 9) /* PAI */
        return !(peek(0x1000) & 128);
   if (port < 16) /* normal bit of 7fff as gotten from digital chip */
    return !((peek(0x7fff) >> (port - 10)) & 1);
   if (port < 32) /* expbd analogs */
    return analog(port) < 128;
   else {
        printf("Digital port outof range\n");
        beep();
        return -1;
   }
}
```

# Appendix H Listing of Aardvark Control Program

The code listed below contains the routines used to control the aardvark. Note comments are colored blue.

```
walkSteps = int ir;
int headVert;
int headHori;
int ulids;
int blids;
int mouth;
int headVertNew;
int headHoriNew;
int headVertOld;
int headHoriOld;
int mouthOld;
int mouthNew;
int walkSteps = 100;

/*This is the part of the program that starts running when the Handy Board is switched on*/
void main()
{
        sony_init(1); /*activate the IR □eceiver – the program now monitors the state of the IR input */
        while(1)
        {
                int ir_now = ir_data(0);  /*set the ir_now variable to the value that is read at the IR port*/

                if(ir_now > 0) /*if the value of the IR port is greater than one, it means that an IR signal has been
                                received. Update the ir_now variable to this new signal*/
                        ir = ir_now;

                /*The program now branches to a subroutine depending on what the value of the IR signal was*/

                if(ir == 149)       /*branch to the setWalk routine if the IR signal was 149*/
                        setWalk();

                if(ir == 244) /*Set the walkSteps variable to 100 and branch to the walk routine if the IR signal was
                                244*/
                {
                        walkSteps = 100;
                        walk();
                }

                if(ir == 129)       /*branch to the stand routine if the IR signal was 129*/
                        stand();

                if(ir == 135)       /*branch to the sit routine if the IR signal was 135*/
                        sit();

                if(ir == 141)       /*branch to the walk routine if the IR signal was 141*/
                        walk();
```

Continued…

```
            if(ir == 165)       /*branch to the head routine if the IR signal was 149*/
                    head();

            /*As well as checking the state of the IR signal, the program also checks the states of the ports connected
              to the push-buttons on the control box. It will then branch to a routine if the correct combination of
              buttons being pressed is detected*/

            if(digital(12) == 1 && digital(14) == 1) /*If buttons 12 and 14 are pressed, jump to the controlBox
                                                       routine*/
                    controlBox();

            /*This is used to speed up the walking and sitting routines by reducing the number of steps that are
                inserted between the programmed waypoints in the routines*/

            if(digital(12) == 1 && digital(13) == 1)    /*If buttons 12 and 13 are pressed, increase the 'steps'
                                                        variable by 0.5 and display the variable on the screen. */
            {
                    steps += 0.5;
                    beep();
                    printf("\nSteps = %f", steps);
                    ir = 0;
            }

            /*This is used to slow down the walking and sitting routines by increasing the number of steps that are
                    inserted between the programmed waypoints in the routines*/

            if(digital(14) == 1 && digital(15) == 1)    /*If buttons 14 and 15 are pressed, increase the 'steps'
                                                        variable by 0.5 and display the variable on the screen. */
            {
                    steps -= 0.5;
                    beep();
                    printf("\nSteps = %f", steps);
                    ir = 0;
            }

    }
    sony_init(0);           /*deactivate the IR decoding routine so that the processor doesn't have to monitor the IR
                            port constantly*/
}


/*This routine makes the aardvark stand by driving the servos to their positions for standing*/
/*Refer to the code listing for 'ssc.c' for an explanation of how the moveTo function works*/
void stand()
{
    moveTo(130.0, 82.0, 120.0, 110.0, 120.0, 75.0, 122.0, 160.0 );
}
```

```
/*This routine makes the aardvark sit up by driving the servos to a series of pre-determined waypoints*/
  void sit()
{
        ir = 0;     /*reset the ir variable to 0, so that ir the IR control was used to initiate the 'sit' routine, the program will
                      not jump back to this routine from the 'main' routine continuously*/
        servo5 = 4402; /*Drive the adjustable weight to the rear*/
        moveTo(112.0, 82.0, 135.0, 125.0, 100.0, 75.0, 167.0, 160.0 );
        moveTo(77.0, 82.0, 160.0, 125.0, 100.0, 75.0, 167.0, 160.0 );
        moveTo(62.0, 82.0, 175.0, 125.0, 80.0, 75.0, 182.0, 160.0 );
        moveTo(62.0, 82.0, 175.0, 125.0, 80.0, 135.0, 182.0, 90.0 );
        moveTo(87.0, 62.0, 155.0, 169.0, 60.0, 160.0, 192.0, 65.0 );
        moveTo(132.0, 18.0, 120.0, 208.0, 60.0, 160.0, 192.0, 65.0 );
        moveTo(132.0, 18.0, 120.0, 208.0, 110.0, 55.0, 132.0, 180.0 );
        moveTo(132.0, 18.0, 120.0, 208.0, 120.0, 9.0, 152.0, 224.0 );
        moveTo(132.0, 18.0, 120.0, 208.0, 120.0, 0.0, 127.0, 235.0 );
        moveTo(180.0, 82.0, 90.0, 110.0,135.0, 0.0, 147.0, 235.0 );

}

/*Routine to make the aardvark stand up again after sitting*/
void standFromSit()
{

        ir = 0;
        moveTo(180.0, 82.0, 90.0, 125.0, 90.0, 0.0, 167.0, 235.0 );
        moveTo(180.0, 82.0, 90.0, 125.0, 70.0, 40.0, 177.0, 210.0 );
        servo5 = 2556; /* Drive the adjustable weight to the front*/
        moveTo(180.0, 82.0, 90.0, 125.0, 60.0, 40.0, 187.0, 210.0 );
        moveTo(180.0, 82.0, 75.0, 125.0, 90.0, 40.0, 162.0, 210.0 );
        moveTo(180.0, 82.0, 75.0, 125.0, 90.0, 40.0, 152.0, 165.0 );
        moveTo(150.0, 82.0, 100.0, 125.0, 90.0, 40.0, 152.0, 165.0 );
        moveTo(140.0, 82.0, 110.0, 125.0, 100.0, 70.0, 142.0, 145.0 );
        moveTo(130.0, 82.0, 120.0, 110.0, 120.0, 75.0, 122.0, 160.0 );

}

/*This is the routine that is run when the aardvark is controlled by the IR remote, and is put into 'head mode'*/
/*The state of the IR port is constantly monitored, and the program branches off to various subroutines depending on the
  value recieved at the IR port*/
/*These subroutines are used to control the movement of the head and facial features with the IR remote*/
void head()
{
        headVert = 2450;
        headHori = 2300;

        beep();
        sleep(0.2);
        ir = 0;
        init_expbd_servos(1); /*Initialize the servos connected to the Handy Board's expansion board*/

        while(ir != 165) /*Continue until a value of 165 is read on the IR port - this will take the robot out of 'head mode'
*/
```

Continued...

```
{
        int ir_now = ir_data(0);

        if(ir_now > 0)
                ir = ir_now;

        if(ir == 146) /*If the IR value read is 146, reduce the headVert variable by 60*/
        {
                headVert -= 60;
                ir = 0;
        }

        if(ir == 147) /*If the IR value read is 147, increase the headVert variable by 60*/
        {
                headVert += 60;
                ir = 0;
        }

        if(ir == 144) /*If the IR value read is 144, reduce the headHori variable by 60*/
        {
                headHori -= 60;
                ir = 0;
        }

        if(ir == 145)/*If the IR value read is 145, increase the headHori variable by 60*/
        {
                headHori += 60;
                ir = 0;
        }

        if(ir == 182)       /*If the IR value read is 182,branch to the knodding routine*/
                knod();

        if(ir == 252)       /*If the IR value read is 252,branch to the head shaking routine*/
                shakeHead();

        if(ir == 132)       /*If the IR value read is 132,branch to the 'lookUp' routine*/
                lookUp();

        if(ir == 137)       /*If the IR value read is 137,branch to the 'lookDown' routine*/
                lookDown();

        if(ir == 134)       /*If the IR value read is 134,branch to the 'lookLeft' routine*/
                lookLeft();

        if(ir == 136)       /*If the IR value read is 136,branch to the 'lookRight' routine*/
                lookRight();

        if(ir == 135)       /*If the IR value read is 182,branch to the 'lookCentre' routine*/
                lookCentre();

        if(ir == 150)
                ir = 0;
```

Continued...

```
            if(ir == 140)       /*If the IR value read is 140,branch to the blinking routine*/
                blink();

            if(ir == 128)       /*If the IR value read is 128,move the top eyelids up*/
                topUp();

            if(ir == 131)       /*If the IR value read is 131,move the bottom eyelids up*/
                botUp();

            if(ir == 133)       /*If the IR value read is 133,move the top eyelids down*/
                topDown();

            if(ir == 128)       /*If the IR value read is 128,move the bottom eyelids down*/
                botDown();

            if(ir == 149)       /*If the IR value read is 149,branch to the 'controlBox' routine*/
                controlBox();

            servo0 = headVert; /*Move the servo that controls the vertical head movement to the new value of
                                 'hadVert'*/
            servo1 = headHori; /*Move the servo that controls the horizontal head movement to the new value of
                                 'hadHori'*/

            printf("\n0 = %d 1 = %d", headVert, headHori); /*display the values of the 'headVert' and 'headHori'
                                                            variables*/
            msleep(10l); /*pause for 10ms */

        }
        init_expbd_servos(0); /*Deactivate the Handy Board servos*/
        ir = 0;
        beep();
}


/*Routine to make the head nod*/
void nod()
{
        int i;
        beep();

        for(i = 0; i < 3; i++)
        {
            for(headVert = 2090; headVert < 2960; headVert ++)
                servo0 = headVert;

            for(headVert = 2950; headVert > 2080; headVert --)
                servo0 = headVert;
        }

        ir = 0;
}
```

```
/*Routine to make the head shake from side to side*/
void shakeHead()
{
        int i;

        beep();

        for(i = 0; i < 3; i++)
        {
                for(headHori = 2980; headHori > 1720; headHori --)
                        servo1 = headHori;

                for(headHori = 1720; headHori < 2980; headHori ++)
                        servo1 = headHori;
        }


        ir = 0;
}

/*Routine to make the head look up*/
void lookUp()
{
        while(headVert >= 22114)
        {
                headVert -= 20;
                servo0 = headVert;
        }
        ir = 0;
}

/*Routine to make the head look down*/
void lookDown()
{
        while(headVert <= 4142)
        {
                headVert += 20;
                servo0 = headVert;
        }
        ir = 0;
}

/*Routine to make the head look left*/
void lookLeft()
{
                beep();

        while(headHori >= 1113)
        {
                headHori -= 20;
                servo1 = headHori;
        }
        ir = 0;
}
```

```
/*Routine to make the head look right*/
void lookRight()
{
                beep();

        while(headHori <= 29850)
        {
                headHori += 20;
                servo1 = headHori;
        }
        ir = 0;
}

/*Routine to move the head to the center*/
void lookCentre()
{
                headVert = 2450;
                headHori = 2300;

                servo0 = headVert;
                servo1 = headHori;

                ir = 0;

}

/*Routine to make the eyelids blink*/
void blink()
{

        servo2 = 1503;
        servo3 = 3544;

        sleep (0.1);

        servo2 = 2530;
        servo3 = 2361;

        sleep (0.2);

        servo2 = 1503;
        servo3 = 3544;



        ir = 0;
}

/*Routine to move the top eyelids up*/
void topUp()
{
        servo2 = 2816;
        ir = 0;

}
```

```
/*Routine to move the top eyelids down*/
void topDown()
{
        servo2 = 4155;
        ir = 0;

}

/*Routine to move the bottom eyelids up*/
void botUp()
{
        servo3 = 2231;
        ir = 0;

}

/*Routine to move the bottom eyelids down*/
void botDown()
{
        servo3 = 1243;
        ir = 0;

}

/*This is the routine that is run continuously when the aardvark is in 'Control Box' mode. The input ports that the control
  box  knobs and push-buttons are connected to are continuously monitored, and the program branches to various
  subroutines from this one, depending on the state of those ports*/

void controlBox()
{
        beep();
        ir = 0;
        sleep(0.2); /*give an audible response to confirm that 'Control Box' mode has been entered*/

        init_expbd_servos(1);       /*Initialize the Handy Board controlled servos*/

        servo5 = 3200;

        mouthOld = 1113 + 80*( - 92 + analog(20));

        while(ir != 149 && !(digital(13) == 1 && digital(15) == 1))
        {
                int irNow = ir_data(0);
                if(irNow > 0)
                        ir = irNow;

                /*These formulas calculate the positions that the servos for head movements must go to, depending on
                the values read from the knobs. The constants in the formulas were determined in the calibration
                process*/
                servo0 = 1867 + 7*analog(18);
                servo1 = 1282 + 8*(200 - analog(19));

                mouthNew = 4077 + 123*(102 - analog(20));;
```

Continued...

```
            if(abs(mouthNew - mouthOld) > 100)
            {
                mouthOld = mouthNew;
                servo4 = mouthNew;
            }

            /*These formulas calculate the positions that the serovs for eyelid movements must go to, depending on
            the values read from the eyelid pots. The constants in the formulas were determined in the calibration
            process*/
            servo2 = 2530 - 15*(116 - analog(16));
            servo3 = 2504 + 63*( - 103 + analog(17));

            if(digital(13) == 1 && digital(15) == 0)
                blink();


            /*If button 12 is pressed and then released, the walking sequence is started. If it is held down for more
            than a second, the tracking sequence is started*/
            if(digital(12) == 1)
            {
                beep();
                sleep(1.0);
                if(digital(12) == 0)
                    {
                        walk();
                        init_expbd_servos(1);
                    }
                else track();
            }

            /*If button 15 is pressed, the aardvark will sit up*/
            if(digital(15) == 1 && digital(14) == 0 )
            {
                sit();
                sleep(1.0);
            }

            /*If buttons 15 and 14 are pressed, the aardvark stands up again*/
            if(digital(15) == 1 && digital(14) == 1)
                standFromSit();

        }

        init_expbd_servos(0);
        beep();
        ir = 0;
}

/*routine to shift the adjustable weight to the front*/
void wf()
{
        init_expbd_servos(1);
        servo5 = 2556;
        msleep(1501);
        init_expbd_servos(0);
}
```

```
/*routine to shift the adjustable weight to the rear*/
void wb()
{
        init_expbd_servos(1);
        servo5 = 4402;
        msleep(150l);
        init_expbd_servos(0);
}


/*routine to shift the adjustable weight to the middlet*/
void wm()
{
        init_expbd_servos(1);
        servo5 = 2800;
        msleep(150l);
        init_expbd_servos(0);
}

/*routine to make the aardvark walk. */
/* This is done by using a series of 'moveTo' commands. The coordinates in these commands were obtained by using the
Visual Basic application for programming the walking sequence. Each of the eight numbers is a position reference to
which on of the leg servos must me moved.  After every couple of such moves, a routine is included to check if the 'Stop'
button has been pressed on the control box.*/

void walk()
{
        int takenSteps; /*integer to keep track of the number of steps taken*/
        float oldSteps = steps;
        steps = 4.5;
        wm(); /*move the adjustable weight to the center*/

        /* The first series of moves are to get the aardvark going from standstill. Once each leg has been brought forward
           once so that the aardvark is walking, the program enters a loop that repeats the walking sequence*/

        moveTo(130.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
        moveTo(130.0, 82.0, 135.0, 75.0, 100.0, 75.0, 127.0, 160.0 ); /* wf, free br */

        steps = oldSteps;

        moveTo(130.0, 82.0, 135.0, 75.0, 85.0, 21.0, 127.0, 160.0 ); /* lift & tuck br */
        moveTo(130.0, 82.0, 135.0, 75.0, 70.0, 91.0, 127.0, 160.0 ); /* bring br through */
        moveTo(130.0, 82.0, 135.0, 75.0, 70.0, 196.0, 127.0, 160.0 ); /* straighten br */
        moveTo(130.0, 82.0, 135.0, 75.0, 110.0, 231.0, 127.0, 160.0 ); /* br to ground */
        moveTo(153.0, 82.0, 135.0, 115.0, 115.0, 231.0, 127.0, 160.0 ); /* wb, free fr */
        moveTo(173.0, 147.0, 135.0, 115.0, 115.0, 231.0, 127.0, 160.0 ); /* lift & tuck fr */
        moveTo(153.0, 162.0, 135.0, 140.0, 125.0, 231.0, 127.0, 160.0 ); /* bring fr through */
        moveTo(138.0, 217.0, 135.0, 140.0, 125.0, 231.0, 127.0, 160.0 ); /* bring fr through */
        moveTo(74.0, 217.0, 135.0, 140.0, 125.0, 231.0, 127.0, 160.0 ); /* bring fr through */
        moveTo(64.0, 92.0, 135.0, 140.0, 125.0, 231.0, 127.0, 160.0 ); /* straighten fr */
        moveTo(64.0, 92.0, 120.0, 140.0, 160.0, 231.0, 117.0, 125.0 ); /* wf, fr to ground */
```

```
/*At this point, the program enters the loop which repeats the walking sequence until the stop button is pressed, or
the desired umber of steps have been taken*/
for(takenSteps = 0; takenSteps < walkSteps; takenSteps++)
{

        moveTo(64.0, 92.0, 120.0, 140.0, 155.0, 141.0, 117.0, 196.0 ); /* free bl */
        moveTo(64.0, 92.0, 110.0, 180.0, 155.0, 141.0, 192.0, 176.0 ); /* lift & tuck bl */
        moveTo(89.0, 127.0, 110.0, 180.0, 160.0, 136.0, 192.0, 46.0 ); /* straighten bl */
        moveTo(114.0, 127.0, 105.0, 225.0, 160.0, 121.0, 142.0, 16.0 ); /* bl to ground */

        /* Check to see if the stop buton has been pressed. If so, the legs are moved to the standing position and
           the walking loop is exited*/
        if(ir_data(0) == 148 || digital(14) == 1)
        {
                takenSteps = walkSteps;
                ir = 148;
                steps = 6.0;
                moveTo(112.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
                steps = oldSteps;
                break;
        }

        moveTo(114.0, 33.0, 85.0, 225.0, 160.0, 121.0, 122.0, 16.0 ); /* wb, free fl */
        moveTo(99.0, 33.0, 75.0, 225.0, 160.0, 121.0, 122.0, 16.0 ); /* wb, free fl */
        moveTo(99.0, 33.0, 75.0, 115.0, 160.0, 121.0, 122.0, 16.0 ); /* wb, free fl */
        moveTo(99.0, 33.0, 115.0, 44.0, 160.0, 121.0, 122.0, 16.0 ); /* lift & tuck fl */
        moveTo(99.0, 33.0, 175.0, 44.0, 160.0, 121.0, 122.0, 16.0 ); /* bring fl through */
        moveTo(114.0, 33.0, 175.0, 134.0, 160.0, 121.0, 122.0, 96.0 ); /* wf, tip to fl */

        /*Check for stop condition*/
        if(ir_data(0) == 148 || digital(14) == 1)
        {
                takenSteps = walkSteps;;
                ir = 148;
                steps = 6.0;
                moveTo(112.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
                steps = oldSteps;
                break;
        }

        moveTo(114.0, 32.0, 175.0, 134.0, 160.0, 121.0, 112.0, 96.0 ); /* free br */
        moveTo(124.0, 32.0, 160.0, 119.0, 95.0, 10.0, 112.0, 96.0 ); /* lift & tuck br */
        moveTo(134.0, 0.0, 140.0, 119.0, 65.0, 100.0, 102.0, 96.0 ); /* bring br through */
        moveTo(134.0, 0.0, 140.0, 119.0, 80.0, 195.0, 102.0, 96.0 ); /* straighten br */
        moveTo(134.0, 0.0, 140.0, 119.0, 125.0, 200.0, 102.0, 96.0 ); /* br to ground */
```

Continued...

```
            /*Check for stop condition*/
            if(ir_data(0) == 148 || digital(14) == 1)
            {
                takenSteps = walkSteps;;
                ir = 148;
                steps = 6.0;
                moveTo(112.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
                steps = oldSteps;
                break;
            }


            moveTo(159.0, 0.0, 140.0, 139.0, 130.0, 200.0, 102.0, 96.0 ); /* wb, free fr */
            moveTo(184.0, 80.0, 140.0, 139.0, 130.0, 200.0, 102.0, 96.0 ); /* lift & tuck fr */
            moveTo(124.0, 209.0, 140.0, 139.0, 130.0, 200.0, 102.0, 96.0 ); /* bring fr through */
            moveTo(54.0, 209.0, 140.0, 139.0, 130.0, 200.0, 102.0, 96.0 ); /* bring fr through */
            moveTo(54.0, 114.0, 140.0, 139.0, 130.0, 200.0, 102.0, 96.0 ); /* straighten fr */
            moveTo(54.0, 114.0, 120.0, 134.0, 135.0, 180.0, 102.0, 86.0 ); /* wf, tip - fr to ground */

            /*Check for stop condition*/
            if(ir_data(0) == 148 || digital(14) == 1)
            {
                takenSteps = walkSteps;;
                ir = 148;
                steps = 6.0;
                moveTo(112.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
                steps = oldSteps;
                break;
            }
        }

        steps = 6.0;
        moveTo(112.0, 82.0, 135.0, 125.0, 120.0, 75.0, 142.0, 160.0 ); /* stand */
        steps = oldSteps;

        ir = 0;
}
```

# Appendix I: Listing of Program for Controlling the SSC via the Handy Board

The code listed below contains the routines used to control the SSC from the Handy Board. Note comments are colored blue.

```
float serv0;
float serv1;
float serv2;
float serv3;
float serv4;
float serv5;
float serv6;
float serv7;

float steps = 4.5;

/*Routine that sends a command to the SSC in the form of a servo number and position. The pa7i9600 function is included
in the pa7i9600.icb file – a library file that contains commands for sending serial information from the Handy Board to a
mini SSC.*/
void ssc(int servo, int pos)
{
        pa7i9600(255);
        pa7i9600(servo);
        pa7i9600(pos);
}

/*Routine that moves each of the eight leg servos to a desired position. The move from the old positions to the new ones is
done in a certain number of steps. Since the servos are not necessarily all moved the same amount, the routine has to
calculate how much to move each servo during each step, so that the servos all finish their movements in the same number
of steps. By increasing or decreasing the number of steps, the motion is either slowed down or speeded up – this technique
is used to control the walking speed.*/

void moveTo(float s0, float s1, float s2, float s3, float s4, float s5, float s6, float s7)
{

        int c;

        /*First the difference between the current position and desired new position for each servo is calculated*/
        float diff0 = s0 - serv0;
        float diff1 = s1 - serv1;
        float diff2 = s2 - serv2;
        float diff3 = s3 - serv3;
        float diff4 = s4 - serv4;
        float diff5 = s5 - serv5;
        float diff6 = s6 - serv6;
        float diff7 = s7 - serv7;
```

```
	for(c = 0; c < (int)steps; c++) /*loop repeats step times*/
	{
		/*The move increment for each servo is calculated by dividing the total amount that the servo has to
		move, by the total number of steps*/
		serv0 += diff0/steps;
		serv1 += diff1/steps;
		serv2 += diff2/steps;
		serv3 += diff3/steps;
		serv4 += diff4/steps;
		serv5 += diff5/steps;
		serv6 += diff6/steps;
		serv7 += diff7/steps;

		/*Each servo is then moved to the new position*/
		ssc(0, (int)serv0);
		ssc(1, (int)serv1);
		ssc(2, (int)serv2);
		ssc(3, (int)serv3);
		ssc(4, (int)serv4);
		ssc(5, (int)serv5);
		ssc(6, (int)serv6);
		ssc(7, (int)serv7);
	}
}
```

# Appendix J: Code Listing for Application to Program the Walking Sequence

```
Private Declare Function SSC_OPEN& Lib "Ssc05.dll" (ByVal commPort&, ByVal baudRate&)
Private Declare Function SSC_CLOSE& Lib "Ssc05.dll" ()
Private Declare Function SSC_MOVE& Lib "Ssc05.dll" (ByVal servo&, ByVal pos&)

Private Sub Form_Load()     ' When form loads,
  Call SSC_OPEN(2, 9600)    ' open comm 5 for SSC at 9600 baud.
  Call Stand_Click
  Open "c:\aardvark\waypoints.txt" For Output As #1
End Sub

Private Sub Form_Unload(Cancel As Integer)
  Call SSC_CLOSE            ' When form unloads, close the SSC/comm port
  Close #1
End Sub

Private Sub Go_Click()
    Call SSC_MOVE(0, Slider0.Value)
    Call SSC_MOVE(1, Slider1.Value)
    Call SSC_MOVE(2, Slider2.Value)
    Call SSC_MOVE(3, Slider3.Value)
    Call SSC_MOVE(4, Slider4.Value)
    Call SSC_MOVE(5, Slider5.Value)
    Call SSC_MOVE(6, Slider6.Value)
    Call SSC_MOVE(7, Slider7.Value)
End Sub

Private Sub Save_Click()
    Print #1, "moveTo("; Text0.Text; ".0, "; Text1.Text; ".0, "; Text2.Text; ".0, "; Text3.Text; ".0, "; Text4.Text; ".0, ";
    Text5.Text; ".0, "; Text6.Text; ".0, "; Text7.Text; ".0 "; "); /* "; descrip.Text; " */"
End Sub

Private Sub Slider0_Click()
    Call SSC_MOVE(0, Slider0.Value)
    Text0.Text = Slider0.Value
End Sub

Private Sub Slider1_Click()
    Call SSC_MOVE(1, Slider1.Value)
    Text1.Text = Slider1.Value
End Sub

Private Sub Slider2_Click()
    Call SSC_MOVE(2, Slider2.Value)
    Text2.Text = Slider2.Value
End Sub

Private Sub Slider3_Click()
    Call SSC_MOVE(3, Slider3.Value)
    Text3.Text = Slider3.Value
End Sub
```

```
Private Sub Slider4_Click()
   Call SSC_MOVE(4, Slider4.Value)
   Text4.Text = Slider4.Value
End Sub

Private Sub Slider5_Click()
   Call SSC_MOVE(5, Slider5.Value)
   Text5.Text = Slider5.Value
End Sub

Private Sub Slider6_Click()
   Call SSC_MOVE(6, Slider6.Value)
   Text6.Text = Slider6.Value
End Sub

Private Sub Slider7_Click()
   Call SSC_MOVE(7, Slider7.Value)
   Text7.Text = Slider7.Value
End Sub

Private Sub Stand_Click()
   Slider0.Value = 177
   Slider1.Value = 81
   Slider2.Value = 60
   Slider3.Value = 100
   Slider4.Value = 85
   Slider5.Value = 75
   Slider6.Value = 167
   Slider7.Value = 165

   Text0.Text = Slider0.Value
   Text1.Text = Slider1.Value
   Text2.Text = Slider2.Value
   Text3.Text = Slider3.Value
   Text4.Text = Slider4.Value
   Text5.Text = Slider5.Value
   Text6.Text = Slider6.Value
   Text7.Text = Slider7.Value

   Call Go_Click
End Sub

Private Sub Text0_Change()
   Slider0.Value = Text0.Text
End Sub

Private Sub Text1_Change()
   Slider1.Value = Text1.Text
End Sub

Private Sub Text2_Change()
   Slider2.Value = Text2.Text
End Sub
```

```
Private Sub Text3_Change()
    Slider3.Value = Text3.Text
End Sub

Private Sub Text4_Change()
    Slider4.Value = Text4.Text
End Sub

Private Sub Text5_Change()
    Slider5.Value = Text5.Text
End Sub

Private Sub Text6_Change()
    Slider6.Value = Text6.Text
End Sub

Private Sub Text7_Change()
    Slider7.Value = Text7.Text
End Sub
```



--- The End ---

# Appendix K: Calculation of Overall Dimensions

Since the leg servos could produce a maximum torque of 0.5Nm, the dimensions of the legs had to be adjusted so that this maximum would not be exceeded while the animatron was walking. As the rear legs would be longer that the front ones, the hip joint servos for the rear legs would be subject to the greatest moment arm (the torque required by the knee joint servos would be less as the gear assembly helps to increase the torque they produce). For this reason, all calculations were based on the rear legs being in the position during the walking sequence that would produce the greatest moment at the hip joint servo. (As illustrated in figure 59).

Since this calculation had to be made while the animatron was still in the prototype Perspex phase, an estimation of the final weight had to be made. Based on the known weight of the servos, the Handy Board and the batteries, the total projected weight was estimated at 3kg. It was assumed that the weight would be distributed evenly among all four legs, so that each leg would have to carry 0.75kg.

To determine the maximum moment arm that could be supported, the following formula was used:

$D = T/F$

Where D = moment arm
T = torque produced by servo (0.5Nm)
F = force applied at moment arm ($F = M \times g = 0.75kg \times 9.8 = 7.35N$)

Inserting the above values into the equation produced a resulting moment arm of D = 0.068m. It was decided to add a small safety factor, and so a final moment arm of 55mm was chosen. Working backwards from this value, the dimensions of the upper and lower limbs where calculated, and then the overall dimensions for the rest of the animatron where scaled accordingly. This resulted in the animatron being approximately ¼ scale.

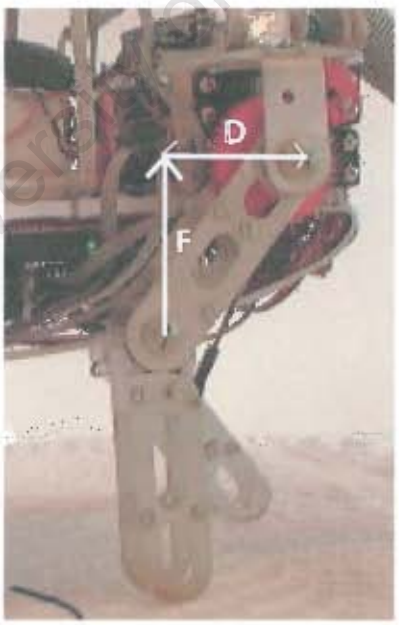



Figure 59. Rear leg in position producing maximum torque arm.