The Creation of Motion Planning Software for a Car-like Robot

Liam K. McAlpine

Master of Science in Engineering specialising in Mechanical Engineering
Research Master's by Dissertation
Department of Mechanical Engineering
Faculty of Engineering & the Built Environment
University of Cape Town

July 2023

Supervised by Arnold Pretorius

Plagiarism Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This dissertation has been submitted to the Turnitin module and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signature: Signed by candidate

Abstract

As the demand for robots and vehicles capable of autonomously navigating through obstacle-rich environments increases, so will the need for software capable of achieving such functionality.

This dissertation presents software capable of finding a solution to the motion planning problem for a car-like robot in an obstacle-rich, static environment and then controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions. The insights gained from this dissertation may be used to develop motion planning software for industrial robots and autonomous vehicles.

To achieve the objective of this dissertation, the use of planning, control, perception and localisation software is necessary. The planning software uses an online Rapidly-exploring Random Tree Star (RRT*) algorithm, which attempts to improve upon the solutions generated by the RRT* algorithm while the robot is tracking the prevailing best solution. The control software uses a trajectory tracking control method that, through a Lyapunov-like analysis using Barbălat's Lemma, is shown to provide global asymptotic stability. The perception software relays wheel encoder and onboard camera information to the localisation software, which uses the extended Kalman filter (EKF) to estimate the configuration and configuration covariance of the robot.

The software is based on the Robot Operating System (ROS) and is tested using a Duckiebot (DB21-M). Because the Duckiebot does not possess a rear-facing camera, the mathematical model used for the car-like robot is the Dubins car, where a compact set of closed-form equations that describe the set of Dubins paths is presented in this dissertation. These equations are derived by modelling the Dubins car as an underactuated system on the special Euclidean group in dimension 2 and then solving an associated set of inverse kinematics problems. A similar set of equations that describe the set of Reeds-Shepp paths is also presented in this dissertation.

To demonstrate the efficacy of the software, three tests are conducted. In the first, second and third tests, the environment contains no obstacles, one obstacle and three obstacles, respectively, where the obstacles are wooden stands with AprilTags attached, which are used to help localise the Duckiebot. The results of these tests show the software searching for and finding a solution to the motion planning problem in these environments and then controlling the Duckiebot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.

Contents

List of Figures

List of Tables

Abbreviations, Acronyms and Initialisms

ARM Advanced RISC (Reduced Instruction Set Computer) Machine. 18

CPU Central Processing Unit. 18

DC Direct Current. 17, 18

EKF extended Kalman filter. 3, 15, 49–51, 69

GIF Graphics Interchange Format. 53, 54, 58, 65

GPIO General Purpose Input/Output. 17

GPU Graphics Processing Unit. 18

IMU Inertial Measurement Unit. 15, 17

KF Kalman filter. 15

LED Light-Emitting Diode. 17

LPDDR Low-Power Double Data Rate. 18

Ltd. limited. 14, 16, 52, 53

PRM Probabilistic Roadmap. 14, 24

PWM Pulse-Width Modulation. 15, 17, 41, 43, 52

RAM Random-Access Memory. 18

RGB Red-Green-Blue. 17

ROS Robot Operating System. 3, 14, 52–54, 69

RRT Rapidly-exploring Random Tree. 6, 14, 24–27, 37, 65

RRT* Rapidly-exploring Random Tree Star. 3, 6, 14, 27–31, 37, 65, 69

UKF unscented Kalman filter. 15

Nomenclature

$(x_1, \ldots, x_n)$ An n-tuple

(x, y) An open interval or a 2-tuple

$(x, y]$ A left-open and right-closed interval

$[x, y)$ A left-closed and right-open interval

$[x, y]$ A closed interval

$\emptyset$ The empty set

$\{x\}$ A set containing x, or a reference frame

$\{x_1, \ldots, x_n\}$ A set of n elements

$\{x_1, \ldots, x_n\}_{<y}$ A set of n elements less than y

$\{x_1, \ldots, x_n\}_{>y}$ A set of n elements greater than y

$\{x_1, \ldots, x_n\}_{\leq y}$ A set of n elements less than or equal to y

$\{x_1, \ldots, x_n\}_{\geq y}$ A set of n elements greater than or equal to y

$\{x_1, \ldots, x_n\}_{\neq y}$ A set of n elements not equal to y

$\mathbb{N}$ The set of natural numbers, using the convention that $\mathbb{N}$ does not include 0

$\mathbb{R}$ The set of real numbers

$\mathbb{R}^n$ The set of n-tuples of real numbers equipped with the dot product

$\mathbb{R}^{n \times m}$ The set of $n \times m$ matrices with real entries

$\mathbb{S}^n$ The n-dimensional sphere

$\mathbb{Z}$ The set of integers

$SE(n)$ The special Euclidean group in dimension n

$\mathfrak{se}(n)$ The Lie algebra of $SE(n)$

$\mathcal{N}(\mu, \Sigma)$ The multivariate Gaussian distribution with mean μ and covariance Σ

$\mu(x)$ The Lebesgue measure of x

$\text{card}(x)$ The cardinality of x

$\text{cl}(x)$ The closure of x

$\det(x)$ The determinant of x

$\exp(x)$ The exponential of x

$\log(x)$ The logarithm of x

$\operatorname{sgn}(x)$ The sign of x

Δx A change in x

$\dot{x}$ The derivative of x with respect to time

$\hat{x}$ The unit vector along the x-axis or the estimated value of x

$\tilde{x}$ The homogeneous form of x

x^+ The predicted value of x

$x^\sharp$ The observed value of x

x^T The transpose of x

$|x|$ The absolute value of x

$\|x\|$ The Euclidean norm of x

$x \simeq y$ x is naturally equivalent to y, with respect to a given context

$x \sim y$ x has the distribution of y

$x \in y$ x is an element of y

$x \subset y$ x is a subset of y

$x : y \to z$ x is a map from y to z

$x \mapsto y$ x maps to y

$x \leftarrow y$ y is assigned to x

$x \bmod y$ x modulo y

$x \cup y$ The union of x and y

$x \setminus y$ The relative complement of y with respect to x

$x \times y$ The Cartesian product of x and y

$x \wedge y$ The logical conjunction of x and y

Chapter 1

Introduction

Modern robotic systems typically consist of a robot equipped with sensors and actuators, utilising perception, localisation, planning and control software. Perception software attempts to use a robot's sensors to take measurements of its environment. Localisation software attempts to use perception information to estimate the configuration of a robot relative to its environment. Planning software attempts to use localisation information to plan a trajectory through a robot's environment. Control software attempts to use a robot's actuators in such a way that the robot achieves a specific goal. Figure 1.1 shows a typical robotic system.

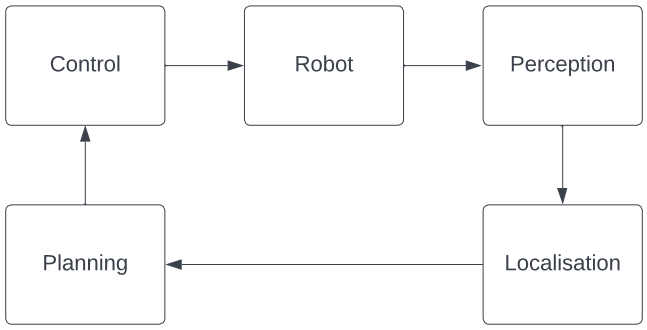


Figure 1.1: A typical robotic system, where rounded rectangles represent subsystems and arrows represent the flow of information.

1.1 Objective

The objective of this dissertation is to present software capable of finding a solution to the motion planning problem for a car-like robot, shown in Figure 1.2, in an obstacle-rich, static environment and then controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions. Figure 1.3 shows one of the obstacle-rich, static environments used for testing in Chapter 8. Subfigure 1.3a shows the real-world environment, and Subfigure 1.3b shows the simulated environment, where the robot at its desired and estimated configuration is shown as a blue and red rectangle, respectively, which are currently overlapping, the goal region is shown as a cyan disc, the search region is shown as a white rectangle, and the obstacles are shown as grey rectangles within white rounded rectangles.

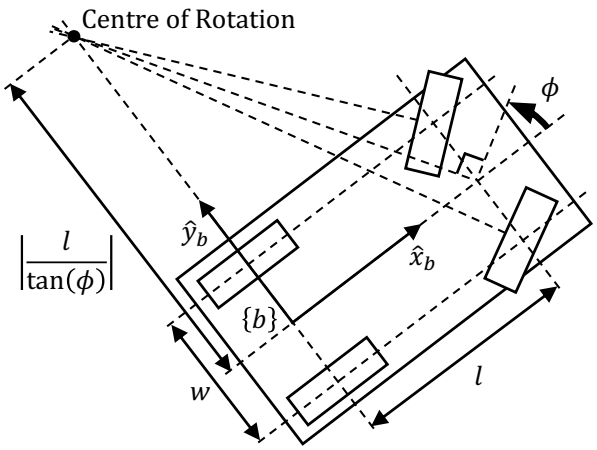
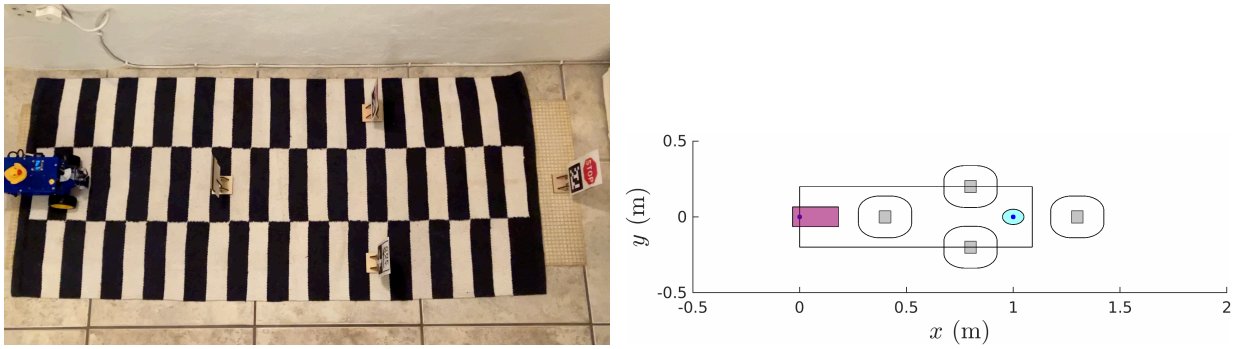


Figure 1.2: A car-like robot with wheelbase l, track w, body frame $\{b\}$ and steering angle ϕ with respect to $\{b\}$.



(a) (b)

Figure 1.3: An obstacle-rich, static environment. (a) Real-world environment. (b) Simulated environment.

1.2 Scope

The main focus of this dissertation is online, single-query motion planning software for a car-like robot in an obstacle-rich, static environment. However, the use of control, perception and localisation software is necessary for the real-world robot to closely track the solutions provided by the motion planning software. Therefore, the control, perception and localisation software necessary to achieve the objective of this dissertation is also examined. Chapter 2 focuses on the robot, Chapters 3, 4, 5 and 6 focus on planning, control, perception and localisation, respectively, Chapter 7 focuses on the software and Chapter 8 focuses on testing.

1.3 Assumptions

For this dissertation, several assumptions are made. All reference frames are stationary and right-handed. The robot operates in a plane. The robot's environment is static. A reliable map of the robot's environment is provided to it. The radii of the robot's wheels are the same. A motor and wheel encoder exists for each of the robot's rear wheels. The number of ticks per revolution for these wheel encoders is the same. There exists a front-facing onboard camera.

Note that, for this dissertation, all angles are assumed to be in radians unless stated otherwise, and, to achieve the objective of this dissertation, a kinematic model of the robot will suffice, as explained in Section 2.3.

1.4 Literature Review

The following subsections present the literature relevant to this dissertation.

1.4.1 The Robot

The robot used for testing the software presented in this dissertation is a Duckiebot (DB21-M), which was developed by Duckieworks Ltd., as part of the Duckietown project [2], presented in [3]. Information on the Duckiebot's specifications is given in Chapter 2, where more information is available in [4]. Information on robotics, planning, control, perception and localisation in the context of Duckietown is available at [5]. While this information was useful as an introduction to these fields, more in-depth information was required for this dissertation. Previously conducted research in the context of Duckietown is available at [6], such as alternative navigation methods using convolutional neural networks [7], vision transformers [8] and deep learning with the corrective advice communicated by humans [9], and an automatic wheel and camera calibration method for monocular differential-drive robots [10].

Note that the Duckiebot does not possess a rear-facing camera, and it is not assumed that the Duckiebot is always very confident in the estimate of its configuration relative to its environment. Therefore, for this dissertation, reversing is completely avoided. Therefore, the mathematical model used for the car-like robot is the Dubins car, which is a car that cannot reverse, as opposed to the Reeds-Shepp car, which is a car that can reverse. Information on the Dubins and Reeds-Shepp car is provided in Chapter 2, where more information is available in [11] and [12].

The software presented in this dissertation is based on the Robot Operating System (ROS) as this simplified the integration of the software with third-party software that is also based on ROS. The structure of the software is expanded upon in Chapter 7. Information on ROS is available at [13].

1.4.2 Planning

The motion planning algorithm used by the software is the online Rapidly-exploring Random Tree Star (RRT*) algorithm presented in [14], which attempts to improve upon the solutions generated by the RRT* algorithm while the robot is tracking the prevailing best solution, where RRT*, presented in [15], is an asymptotically-optimal variation of the Rapidly-exploring Random Tree (RRT) algorithm, presented in [16], which is a single-query sampling-based motion planning algorithm. If a multiple-query sampling-based motion planning algorithm is required, the Probabilistic Roadmap (PRM) algorithm, presented in [17], or one of its variations is recommended. Information on the online RRT* algorithm is provided in Chapter 3. Information on path planning, motion planning, nonholonomic systems, nonholonomic motion planning, motion planning for mobile robots and motion planning algorithms is available in [11], [12], [18], [19], [20], [21], [22], [23], [24] and [25]. A method for extending RRT* to systems with kinodynamic constraints is presented in [26], and a method for extending RRT* to systems with complex differential and geometric constraints with high-dimensional state spaces is available in [27]. When checking for collisions, the software uses the van der Corpus sequence. Information on collision detection and the van der Corpus sequence is available in [11] and [28].

The software uses the set of Dubins paths, presented in [29], to generate solutions, as opposed to the set of Reeds-Shepp paths, presented in [30] as a set of 48 paths and later presented in [31] as a set of 46 paths. Information on Dubins and Reeds-Shepp paths is provided in Section 3.6 and Appendix A, respectively. More information on Dubins and Reeds-Shepp paths is available in [11], [12] and [24]. To find the shortest Dubins path, given that a certain condition is met, the software uses the method presented in [32], which also provides equations that may be used to generate the set of Dubins paths. However, a more compact set of equations is presented in Section 3.7. Following the approach taken in [33], these equations are derived by modelling the Dubins car as an underactuated system on the special Euclidean group in dimension 2 and then solving an associated set of inverse kinematics problems. A similar set of equations that describe the set of Reeds-Shepp paths is presented in Appendix B. Information on the mathematics used to derive these sets of equations is available in [11], [12], [18] and [19].

1.4.3 Control

The control method used by the software is based on the unicycle-like robot control method presented in [34], where the robot's linear velocity is controlled such that the position error is reduced, and the robot's angular velocity is controlled such that the cross-track and orientation errors are reduced. A Lyapunov-like analysis using Barbălat's Lemma shows that this control method provides global asymptotic stability. Information on this control method is provided in Section 4.1. Information on Lyapunov-like analysis using Barbălat's Lemma is available in [35]. To control the Duckiebot, which is a differential-drive robot, the control variables are converted into PWM signals for the Duckiebot's motorised wheels. Information on this conversion is provided in Sections 4.2 and 7.2. Information on differential-drive models is available in [11], [12] and [36]. Information on feedback control for the car-like robot is available in [37].

Another control method is presented in [14], where the robot's velocity is controlled using proportional-integral control, and the robot's steering angle is controlled such that the cross-track and orientation errors are reduced. Yet another control method is presented in [36], where the robot's velocity and steering angle are controlled using proportional-integral and proportional control, respectively. Information on proportional-integral-derivative control and its subsets is available in [12].

1.4.4 Perception

To achieve the objective of this dissertation, the use of wheel encoders for odometry and a front-facing onboard camera for landmark observations will suffice, as the information they provide is sufficient for reliable localisation, although this could be improved by using additional sensor information, such as the gyroscope, accelerometer and magnetometer information from the Duckiebot's IMU. The mathematical model used for the camera is the discrete pixel central projection model. Information on wheel encoder and camera information is provided in Chapter 5, where more information is available in [36].

1.4.5 Localisation

For localisation, the software fuses odometric information, inferred using information from the robot's wheel encoders, and landmark observation information, inferred using information from the images captured by the robot's camera, together using the extended Kalman filter (EKF), which is a nonlinear variation of the Kalman filter (KF), presented in [38], as this will extract the best qualities of each type of sensor information if the various noises involved are accurately modelled. Information on this localisation method is provided in Chapter 6. Information on sensor fusion is available in [36] and [39]. Unfortunately, the EKF is limited due to its first-order linearisation. Therefore, the unscented Kalman filter (UKF), presented in [40], is recommended for highly nonlinear systems. While not covered in this dissertation, as it is assumed that a reliable map of the robot's environment is provided to it, information on simultaneous localisation and mapping is available in [36] and [41].

AprilTags, presented in [42], are used to estimate the range and bearing of landmarks in the robot's environment. If multiple AprilTags are identified in an image, the information from the one with the lowest object-space collinearity error, generated by the orthogonal iteration algorithm presented in [43], is used. Information on AprilTags is provided in Section 6.2.

1.5 Research Contributions

This dissertation presents:

- Software capable of finding a solution to the motion planning problem for a car-like robot in an obstacle-rich, static environment and then controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.
- A compact set of closed-form equations that describe the set of Dubins and Reeds-Shepp paths.

Chapter 2

The Robot

The robot used for testing the software presented in this dissertation is a Duckiebot (DB21-M), shown in Figure 2.1, which was developed by Duckieworks Ltd., as part of the Duckietown project [2]. This robot was chosen because it can model the behaviour of a car-like robot and is equipped with a front-facing camera, wheel encoders and an onboard computer. It is also accompanied by a rich codebase developed by Duckieworks Ltd., which is available at [44]. However, the software presented in this dissertation is not limited to the Duckiebot (DB21-M) and can be adapted for other platforms. Note that this may affect the performance of the software.

Figure 2.1: The Duckiebot (DB21-M). Source: [1].

2.1 Specifications

The following specifications for the Duckiebot are available in [4].

For power, the Duckiebot is equipped with a custom-made power bank known as the Duckiebattery. For computation, the Duckiebot is equipped with an NVIDIA Jetson Nano developer kit. For sensing, the Duckiebot is equipped with two Hall effect sensor wheel encoders (DZ08003134-01), a front-facing camera (IMX-219-160), a front-facing time-of-flight sensor (VL53L0X), an IMU (MPU-9250), which includes a gyroscope, accelerometer and magnetometer, and various sensors for battery diagnostics. For actuation, the Duckiebot is equipped with two DC motors (DG01D 48:1), each actuating a front wheel, and four addressable RGB LEDs, mounted on the Duckiebot's front and back bumpers. The DC motors receive PWM signals from the Duckietown Hut, which is a shield connected to the Jetson Nano's GPIO pins. The locations of these components on the Duckiebot and their connections to the Duckietown Hut are shown in Figures 2.2 and 2.3, respectively. More detailed specifications are shown in Table 2.1.

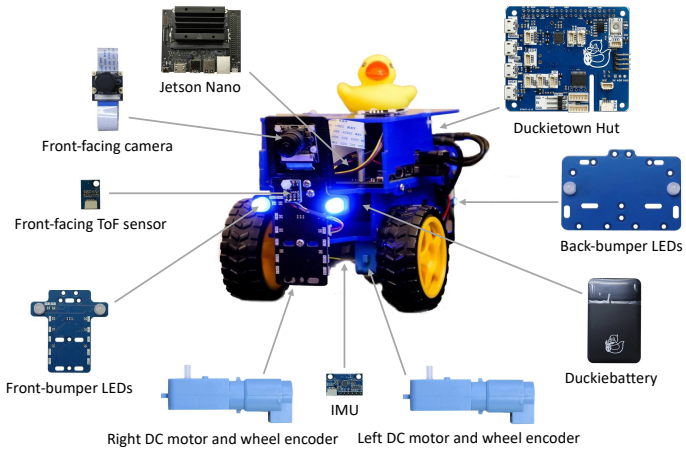


Figure 2.2: The locations of the Duckiebot's components. Source: Adapted from [1].

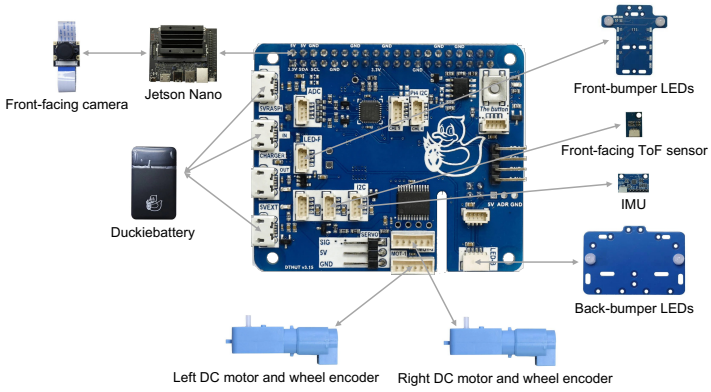


Figure 2.3: The connections of the Duckietown Hut's components. Source: Adapted from [1].

Component	Specifications
Wheel encoders	135 ticks per revolution
Camera	8 MP, 160° field of view, 2.35 aperture, 3.15 mm focal length and less than 14.3% distortion
Time-of-flight sensor	∼0.05 m to ∼1.2 m range in default mode and ∼ 2 m in "long-range" mode
DC motors	90 ± 10 rpm no-load speed and 0.078 N.m allowable maximum torque
CPU	1.43 GHz quad-core ARMv8-A A57
GPU	128-core NVIDIA Maxwell
RAM	2 GB 64-bit 1.6 GHz LPDDR4 25.6 GB.s^{-1}
Battery	10 A.h at 3.7 V capacity, 5 V at up to 2 A charging and 5 V at up to 4.5 A combined output

Table 2.1: Duckiebot specifications.

2.2 Simulating the Behaviour of a Car-like Robot

The Duckiebot is a differential-drive robot, where two wheels are driven by independent DC motors, and a third passive omnidirectional wheel provides equilibrium. However, one can simulate the behaviour of a car-like robot by treating the Duckiebot's motorised wheels as one would the rear wheels of the car-like robot, as shown in Figure 2.4.

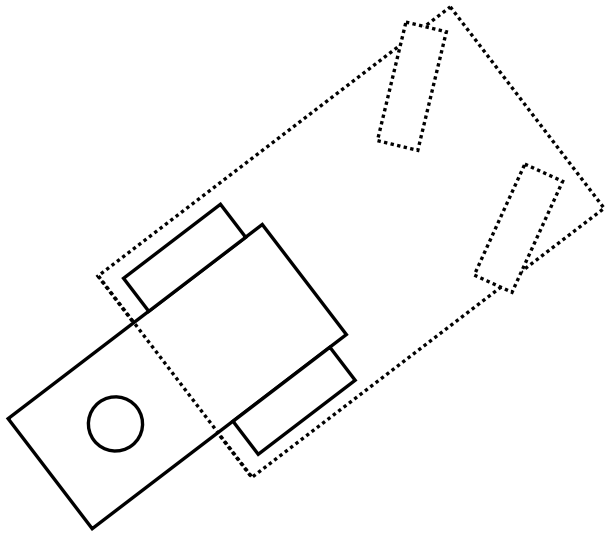


Figure 2.4: The Duckiebot simulating the behaviour of a car-like robot, where solid lines represent the Duckiebot and short dashed lines represent the car-like robot.

Because the Duckiebot is simulating the behaviour of a car-like robot, the distance between the front of the simulated robot and its rear axle should be further than or equal to the distance between the front of the Duckiebot and its axle. However, the width of the car-like robot should be set to the width of the Duckiebot to avoid unknowingly colliding with any obstacles. If reversing is allowed, the distance between the back of the car-like robot and its rear axle should be further than or equal to the distance between the back of the Duckiebot and its axle to avoid unknowingly colliding with any obstacles.

2.3 Choosing a Mathematical Model

The information in this section is derived from [11] and [12].

It is often the case that a dynamic model is required to accurately describe the behaviour of a real-world robot. Unfortunately, because dynamic equations are of second-order, they are often more difficult to analyse than kinematic models, which are only of first-order. However, if there exist motor controllers capable of stabilising the angular velocities of the robot's wheels, given that the desired angular velocities of the robot's wheels and the motor loads may vary continuously, or if the speed of the robot is kept small enough, such that friction quickly overcomes the momentum of the robot in the absence of controls, then, to achieve the objective of this dissertation, a kinematic model of the robot will suffice.

For this dissertation, the maximum speed of the robot is limited such that, as observed through testing, friction quickly overcomes the momentum of the robot in the absence of controls. Therefore, a kinematic model of the robot will suffice.

Note that the Duckiebot does not possess a rear-facing camera. Therefore, if the Duckiebot is located within a dynamic environment, then reversing should be completely avoided unless an external source is providing the Duckiebot with a constantly updating map of its environment and the Duckiebot is always very confident in the estimate of its configuration relative to its environment. However, if the Duckiebot's environment is static, then reversing does not need to be completely avoided if the Duckiebot possesses a reliable map of its environment and is always very confident in the estimate of its configuration relative to its environment.

For this dissertation, it is assumed that the Duckiebot's environment is static and that the Duckiebot is provided with a reliable map of its environment. However, it is not assumed that the Duckiebot is always very confident in the estimate of its configuration relative to its environment. Therefore, for this dissertation, reversing is completely avoided. Therefore, the mathematical model chosen is the Dubins car, which is a car that cannot reverse, as opposed to the Reeds-Shepp car, which is a car that can reverse.

2.4 The Kinematic Model

The information in this section is derived from [11] and [12].

Let $\{s\}$ denote the spatial frame and $\{b\}$ denote the body frame of the robot, where the origin of $\{b\}$ is located at the centre of the rear axle, with its x-axis pointing towards the centre of the front axle and its y-axis pointing towards the centre of the left rear wheel, as shown in Figure 2.5.

Let $\mathbb{R}$ denote the set of real numbers, $\mathbb{R}^n$ denote the set of n-tuples of real numbers equipped with the dot product, and $\mathbb{S}^n$ denote the n-dimensional sphere.

For this dissertation, the configuration space of the robot, which is the space of all configurations of the robot, will be a smooth manifold referred to as the configuration manifold. Therefore, let $Q = \mathbb{R}^2 \times \mathbb{S}^1$ denote the configuration manifold of the robot and $q \in Q$ denote its configuration, where in coordinates (x, y, θ),

$$q = (x, y, \theta), \tag{2.1}$$

where $(x, y) \in \mathbb{R}^2$ denotes the position of the origin of $\{b\}$ with respect to the origin of $\{s\}$ and $\theta \in \mathbb{S}^1$ denotes the orientation of $\{b\}$ with respect to $\{s\}$.

Let $\Delta t \in \mathbb{R}_{>0}$ denote a positive duration over which the motion of the robot is analysed. The smaller Δt is, the more the path of the robot approximates a straight line along the x-axis of $\{b\}$, which, in the limit as Δt tends to zero, implies that

$$\frac{dy}{dx} = \tan(\theta). \tag{2.2}$$

As stated in [11], this can be written as the Pfaffian constraint

$$-\dot{x}\sin(\theta) + \dot{y}\cos(\theta) = 0, \tag{2.3}$$

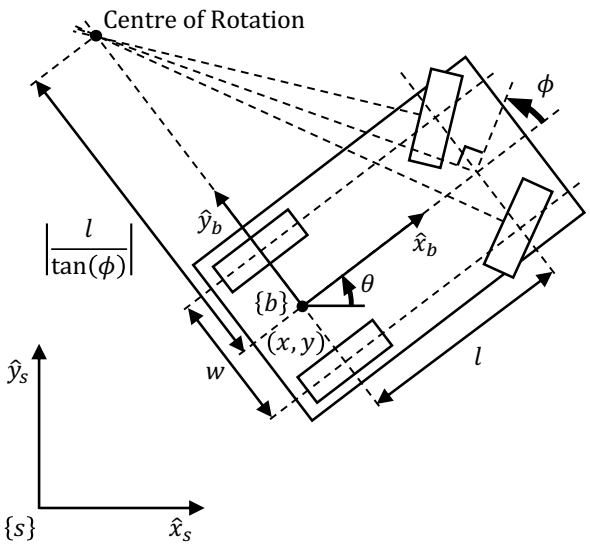


Figure 2.5: A car-like robot with position (x, y) and orientation θ with respect to the spatial frame $\{s\}$.

which is nonintegrable and is, therefore, a nonholonomic constraint, meaning that it reduces the dimension of the robot's feasible velocities but does not reduce the dimension of its reachable configuration manifold. The constraint is satisfied if

$$\dot{x} = v\cos(\theta), \tag{2.4}$$

$$\dot{y} = v\sin(\theta), \tag{2.5}$$

where $v \in \mathbb{R}$ denotes the velocity of the robot along the x-axis of $\{b\}$. For a real-world robot being described by a kinematic model, let $v \in [v_{\min}, v_{\max}]$, where $v_{\min} \in \mathbb{R}_{<0}$ and $v_{\max} \in \mathbb{R}_{>0}$ denote the minimum and maximum allowable velocities of the robot, respectively. However, to satisfy the Dubins car model, let $v \in [0, v_{\max}]$.

Let $\phi \in [-\phi_{\max}, \phi_{\max}]$ denote the steering angle of the robot's front wheels, where $\phi_{\max} \in \left(0, \frac{\pi}{2}\right)$ denotes the maximum steering angle. Therefore, let

$$\omega = \begin{cases} \frac{v}{r} & \text{if } \phi \neq 0, \\ 0 & \text{if } \phi = 0 \end{cases} \tag{2.6}$$

denote the angular velocity of the robot, where $l \in \mathbb{R}_{>0}$ denotes the wheelbase of the robot and, through inspection using Figure 2.5,

$$r = \frac{l}{\tan(\phi)}, \tag{2.7}$$

where $|r|$ is the turning radius of the robot.

Let

$$s = \int_0^\tau v dt \tag{2.8}$$

denote the robot's path length, where $\tau \in \mathbb{R}_{\geq 0}$ denotes a point in time. Therefore, through inspection,

$$d\theta = \frac{ds\tan(\phi)}{l}. \tag{2.9}$$

Therefore,

$$\dot{\theta} = \frac{v \tan(\phi)}{l} \tag{2.10}$$

$$= \begin{cases} \frac{v}{r} & \text{if } \phi \neq 0, \\ 0 & \text{if } \phi = 0 \end{cases} \tag{2.11}$$

$$= \omega. \tag{2.12}$$

2.5 Summary

To summarise this chapter:

- The robot used for testing the software presented in this dissertation is a Duckiebot (DB21-M).
- While the Duckiebot is a differential-drive robot, one can simulate the behaviour of a car-like robot by treating the Duckiebot's motorised wheels as one would the rear wheels of the car-like robot.
- For this dissertation, the maximum speed of the robot is limited such that, as observed through testing, friction quickly overcomes the momentum of the robot in the absence of controls. Therefore, a kinematic model of the robot will suffice.
- The Duckiebot does not possess a rear-facing camera and it is not assumed that the Duckiebot is always very confident in the estimate of its configuration relative to its environment. Therefore, for this dissertation, reversing is completely avoided. Therefore, the mathematical model chosen is the Dubins car, which is a car that cannot reverse.

Chapter 3

Planning

In robotics, planning involves the use of localisation information to plan a path or trajectory through a robot's environment. As explained in [12], planning often refers to either motion planning, which is the problem of finding a collision-free motion from an initial state to a goal state while satisfying motion and control constraints, or path planning, which is the purely geometric problem of finding a collision-free path from an initial configuration to a goal configuration without concern for time, dynamics, motion constraints or control constraints.

Given that this dissertation is using a kinematic model to describe the behaviour of a car-like robot, the focus of this chapter is kinematic motion planning, where the state of the robot is defined by its configuration, as opposed to dynamic motion planning, where the state of the robot is defined by its configuration and velocity.

The subsequent section describes the motion planning problem and what a computer requires to solve it. First, one needs to define the obstacle and obstacle-free regions of the robot's configuration manifold. Therefore, given a configuration manifold Q, let Q_{obs} denote its obstacle region, such that $Q \setminus Q_{\text{obs}}$ is open, and $Q_{\text{free}} = \text{cl}(Q \setminus Q_{\text{obs}})$ denote its obstacle-free region.

3.1 The Motion Planning Problem

The information in this section is derived from [12].

The motion planning problem for a robot being described by the Dubins car model can be stated as the following:

Given an initial configuration $q_{\text{initial}} \in Q_{\text{free}}$ *and a goal configuration* $q_{\text{goal}} \in Q_{\text{free}}$*, if possible, find a time* $\tau \in \mathbb{R}_{\geq 0}$ *and a control* $u : [0, \tau] \to \{u_v, u_\phi\}$ *such that the controlled trajectory* (γ, u) *with* $\gamma(0) = q_{\text{initial}}$ *has the property that* $\gamma(\tau) = q_{\text{goal}}$*, where* $\gamma : [0, \tau] \to Q_{\text{free}}$ *denotes the trajectory of the robot, represented in coordinates* (x, y, θ) *by* $[0, \tau] \ni t \mapsto \big(x(t), y(t), \theta(t)\big)$*, and* $u_v \in [0, v_{\max}]$ *and* $u_\phi \in [-\phi_{\max}, \phi_{\max}]$ *denote the control variables, defined by*

$$\dot{x} = u_v \cos(\theta), \tag{3.1}$$

$$\dot{y} = u_v \sin(\theta), \tag{3.2}$$

$$\dot{\theta} = \frac{u_v \tan(u_\phi)}{l}. \tag{3.3}$$

One could also consider a goal region $Q_{\text{goal}} \subset Q_{\text{free}}$ as opposed to q_{goal}, where the condition $\gamma(\tau) = q_{\text{goal}}$ is changed to $\gamma(\tau) \in Q_{\text{goal}}$, if any configuration in Q_{goal} would suffice.

The control for this problem is currently open-loop, where no feedback is used to correct for error, and will therefore be unreliable for controlling a real-world robot, as any difference between the modelled behaviour

and the actual behaviour of the robot will be unaccounted for, which may lead to failure. A more reliable strategy is to incorporate closed-loop control, where feedback is used to correct for error, and track a desired trajectory $\gamma_d : [0, \tau] \to Q_{\text{free}}$. This strategy is outlined in Chapter 4.

Multiple-query motion planning involves the construction of a data structure that accurately represents the obstacle-free region of a static environment, such that this data structure can then be used to solve multiple motion planning problems within that environment. Single-query motion planning solves each new motion planning problem from scratch. The focus of this dissertation is single-query motion planning.

Typically, for a computer to solve a problem, a set of instructions needs to be provided to it in the form of code. This set of instructions is known as an algorithm. Therefore, for a computer to solve the motion planning problem, a motion planning algorithm needs to be provided to it in the form of code.

3.2 Motion Planning Algorithms

The information in this section is derived from [12].

A motion planning algorithm is said to be complete if it is guaranteed to find a solution in a finite period if one exists and report failure if one does not exist. It is said to be resolution complete if it is guaranteed to find a solution in a finite period if one exists at the resolution of a grid representation of the obstacle-free region. Finally, it is said to be probabilistically complete if the probability of finding a solution, if one exists, approaches 100% as time approaches infinity.

An online algorithm attempts to improve upon the solutions it generates while the robot tracks the prevailing best solution, whereas an offline algorithm does not. Therefore, to achieve the objective of this dissertation, as stated in Section 1.1, the use of an online motion planning algorithm is required.

Motion planning algorithms tend to utilise at least one of the common types of motion planning methods, including complete, grid, sampling, virtual potential field and nonlinear optimisation methods. The descriptions provided in [12] for each method are given below.

3.2.1 Complete Methods

Complete methods use an exact representation of the obstacle-free region to solve the motion planning problem. These methods ensure completeness. However, exact representations of obstacle-free regions can be mathematically and/or computationally prohibitive to derive for complex or high-degree-of-freedom motion planning problems.

3.2.2 Grid Methods

Grid methods discretise the obstacle-free region into a grid and search that grid for a solution to the motion planning problem. These methods tend to be easy to implement and may be able to return optimal solutions. However, the memory required to store grids of a fixed resolution and the time it takes to search that grid grow exponentially with the dimension of the configuration space.

3.2.3 Sampling Methods

Sampling methods use a random or deterministic function to sample from the configuration space, evaluate whether the sample is in the obstacle-free region, determine the nearest previous sample located in the obstacle-free region and try to move towards or connect to the new sample from the previous sample, building up a graph of feasible motions. These methods tend to be easy to implement, tend to be probabilistically complete, and can often solve high-degree-of-freedom motion planning problems with relative ease. However, they tend to be satisficing, as opposed to optimal, and it is often difficult to analyse their computational complexity.

3.2.4 Virtual Methods

Virtual potential field methods apply forces to a robot such that it is repelled by obstacles and attracted towards the goal. These methods tend to be relatively easy to implement, even for high-degree-of-freedom motion planning problems, and relatively fast to evaluate, allowing for online planning. However, the robot may get stuck in configurations where the potential function is at a local minimum.

3.2.5 Nonlinear Optimisation Methods

Nonlinear optimisation methods convert the motion planning problem into a nonlinear optimisation problem. This is done by representing the path or controls as a finite number of design parameters and solving for the design parameters that minimise a cost function while satisfying constraints on the controls, obstacles and goal. These methods tend to produce near-optimal solutions. However, they require an initial guess of the solution, which can lead to the optimisation process getting stuck far away from a feasible solution. This is because the objective function and feasible solution space tend to be nonconvex.

3.2.6 The Chosen Method

For this dissertation, a sampling method was chosen because it would not require an exact representation of the obstacle-free region, it would be relatively easy to implement, it would have relatively low memory requirements, and it would avoid the local minima problem. The two main types of sampling-based motion planning algorithms are the Rapidly-exploring Random Tree (RRT) algorithm, which uses a tree graph representation of feasible motions for single-query motion planning, and the Probabilistic Roadmap (PRM) algorithm, which uses a roadmap graph of feasible motions for multiple-query motion planning. Therefore, because the focus of this dissertation is single-query motion planning, the RRT algorithm was chosen.

3.3 The RRT Algorithm

Rapidly-exploring Random Tree (RRT), presented in [16], is a single-query, probabilistically complete, sampling-based motion planning algorithm. The algorithm begins by initialising a directed, weighted tree graph at the initial configuration. The algorithm then randomly samples from an almost-uniform distribution over Q_{free}, slightly biased towards the goal. The configuration that is a certain distance along the collision-free motion from the vertex that is nearest to the sampled configuration becomes the new vertex. Then, the edge from the nearest vertex to the new vertex is added to the tree graph. This process is repeated until a certain condition is met. Unfortunately, this algorithm is asymptotically non-optimal, as the cost of the returned solution converges almost surely to a non-optimal value.

The functions used in the RRT algorithm are defined below.

Let $Q = \mathbb{R}^n$, where orientation is accounted for in the motion along the edges of the tree graph, and $q_{\text{initial}} \in Q_{\text{free}}$ denote an initial configuration. For a car-like robot, let $n = 2$ and enlarge Q_{obs} such that collision-free paths in Q_{free} correspond to collision-free paths in the robot's workspace.

Let `random_free` : $(Q_{\text{free}}) \mapsto q$ denote a function that takes the obstacle-free region Q_{free} and returns a random configuration $q \in Q_{\text{free}}$ from a given distribution, with a slight bias towards the goal.

Let `nearest` : $(V, q) \mapsto v$ denote a function that takes a set of vertices $V \subset Q$ and a configuration $q \in Q$, and returns the vertex $v \in V$ that is nearest to q in terms of a given distance function.

Let `dist` : $(q_1, q_2) \mapsto d$ denote a function that takes two configurations $q_1, q_2 \in Q$ and returns the distance $d \in \mathbb{R}_{\geq 0}$ between them in terms of a given metric. For the software presented in this dissertation, the Euclidean metric is used.

Let `steer` : $(q_1, q_2) \mapsto q_3$ denote a function that takes two configurations $q_1, q_2 \in Q$ and returns a configuration $q_3 \in Q$ such that $\texttt{dist}(q_2, q_3)$ is minimised and $\texttt{dist}(q_1, q_3) \leq r_{\max}$, for a given $r_{\max} \in \mathbb{R}_{>0}$.

Let `collision_free` : (q_1, q_2) denote a Boolean function that takes two configurations $q_1, q_2 \in Q$ and returns `true` if the path between q_1 and q_2 lies in Q_{free}, and `false` otherwise. For the software presented in this dissertation, the van der Corpus sequence is used to sample the path, where the samples are used to check for collisions.

The RRT algorithm is shown in Algorithm 1.

Algorithm 1: Rapidly-exploring Random Tree (RRT)

$V \leftarrow \{q_{\text{initial}}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, k$ **do**
  $q_{\text{rand}} \leftarrow$ `random_free`(Q_{free});
  $q_{\text{nearest}} \leftarrow$ `nearest`(V, q_{rand});
  $q_{\text{new}} \leftarrow$ `steer`$(q_{\text{nearest}}, q_{\text{rand}})$;
  if `collision_free`$(q_{\text{nearest}}, q_{\text{new}})$ **then**
    $V \leftarrow V \cup \{q_{\text{new}}\}$;
    $E \leftarrow E \cup \big\{(q_{\text{nearest}}, q_{\text{new}})\big\}$;
  end
end
return $G = (V, E)$;

Figure 3.1 shows a visualisation of the RRT algorithm, where the robot is shown as a blue rectangle, the goal region is shown as a cyan disc, the search region is shown as a white rectangle, and the tree graph is shown in black, with the path to the goal in blue.

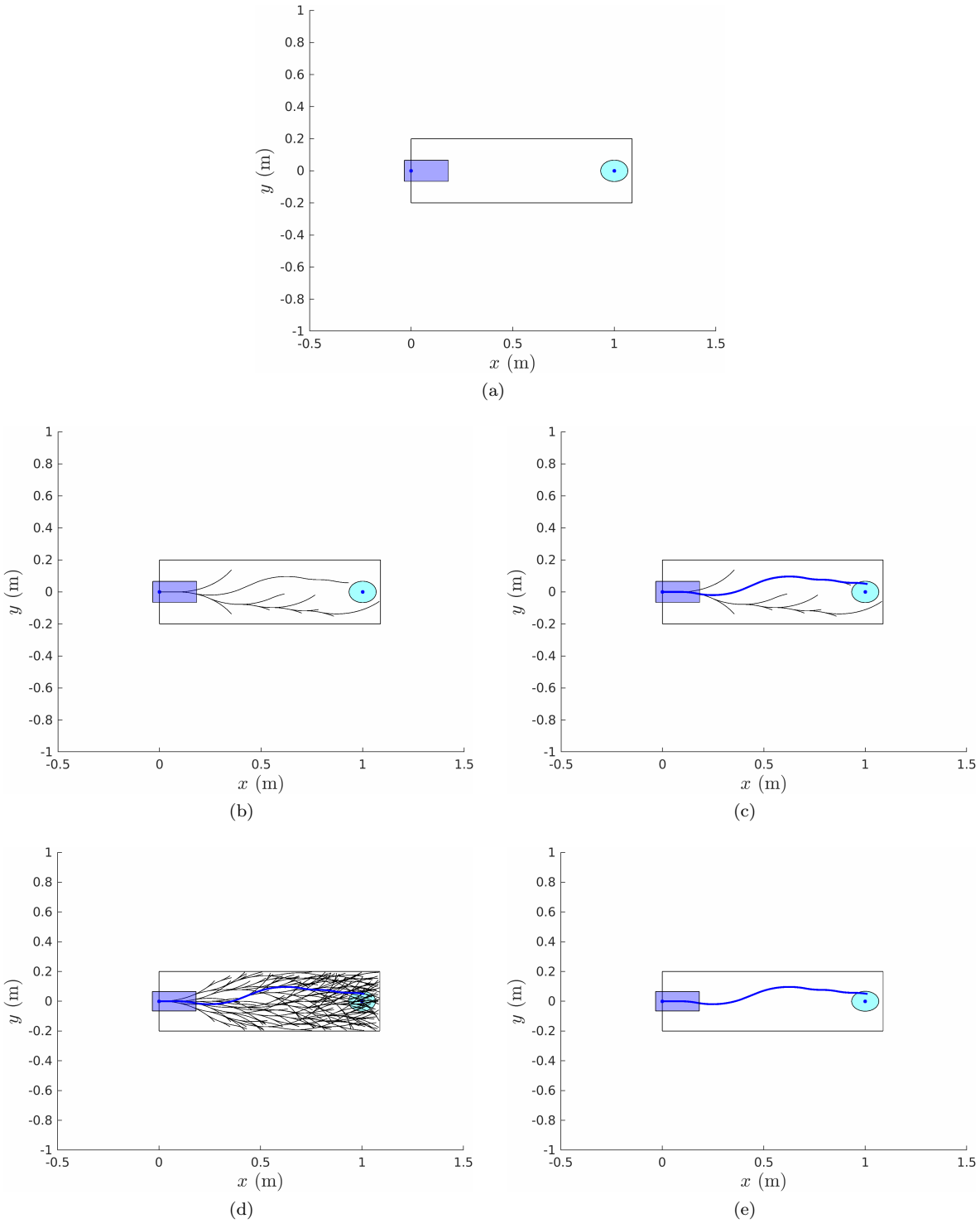


Figure 3.1: A visualisation of the RRT algorithm, where $\phi_{\max} = 25°$ and $l = 0.15$ m. (a) Setup. (b) Searching for a solution. (c) A solution has been found. (d) Search ended at $\text{card}(V) = 500$. (e) The final path to the goal.

3.4 The RRT* Algorithm

Rapidly-exploring Random Tree Star (RRT*), presented in [15], is an asymptotically-optimal variation of the RRT algorithm. Before adding an edge to the tree graph, the vertex that is nearest to the sampled configuration is labelled the minimum-cost vertex. The algorithm then tests all vertices within a given vicinity of the new vertex to find a vertex that provides a lower-cost collision-free path between the initial configuration and the new vertex, if an edge from the test vertex to the new vertex were to be added to the tree graph as opposed to an edge from the minimum-cost vertex to the new vertex. If found, this vertex becomes the new minimum-cost vertex and the test continues until every vertex within the given vicinity has been tested. An edge from the minimum-cost vertex to the new vertex is then added to the tree graph. The algorithm then tests the same vertices with the given vicinity of the new vertex to see if the new vertex provides a lower-cost collision-free path between the initial configuration and the test vertex, in which case the parent vertex of the test vertex is changed to the new vertex. Therefore, by continually rewiring the tree graph, such that the cost of the path between the initial configuration and each of the tree graph's vertices is minimised, the cost of the solution returned by this algorithm converges almost surely to an optimal value. Therefore, this algorithm is asymptotically optimal. Unfortunately, it is still an offline algorithm in its current form.

The functions present in the RRT* algorithm but not the RRT algorithm are defined below.

Let $\mathbb{Z}$ denote the set of integers.

Let

$$\gamma_{\text{RRT}^*} = 2\left(\left(1+\frac{1}{n}\right)\frac{\mu(Q_{\text{free}})}{\zeta_n}\right)^{\frac{1}{n}}, \tag{3.4}$$

where $n \in \mathbb{Z}_{\geq 2}$, $\mu(Q_{\text{free}})$ denotes the Lebesgue measure of Q_{free} and ζ_n denotes the volume of the unit ball in $\mathbb{R}^n$. Therefore, for a car-like robot,

$$\gamma_{\text{RRT}^*} = \left(\frac{6\mu(Q_{\text{free}})}{\pi}\right)^{\frac{1}{2}}. \tag{3.5}$$

Let `near` $: (V, q, r) \mapsto V'$ denote a function that takes a set of vertices $V \subset Q$, a configuration $q \in Q$ and a positive real number $r \in \mathbb{R}_{>0}$, and returns the vertices $V' \subset V$ that are contained in a ball of radius r centred at q.

Let `cost` $: (G, v) \mapsto c$ denote a function that takes a tree graph $G = (V, E)$, where $V \subset Q$ denotes a set of vertices and E denotes a set of edges, and a vertex $v \in V$, and returns the cost $c \in \mathbb{R}_{\geq 0}$ of the unique path from q_{initial} to v through G.

Let `path_cost` $: (q_1, q_2) \mapsto c$ denote a function that takes two configurations $q_1, q_2 \in Q$ and returns the cost $c \in \mathbb{R}_{\geq 0}$ of the shortest path from q_1 to q_2. For a robot being described by the Dubins and Reeds-Shepp car models, this path will be a Dubins and Reeds-Shepp path, respectively. The problem of finding this path for a robot being described by the Dubins and Reeds-Shepp car models is considered in Section 3.6 and Appendix A, respectively.

Let `parent` $: (G, v_1) \mapsto v_2$ denote a function that takes a tree graph $G = (V, E)$, where $V \subset Q$ denotes a set of vertices and E denotes a set of edges, and a vertex $v_1 \in V$, and returns the unique vertex $v_2 \in V$ such that $(v_1, v_2) \in E$.

The RRT* algorithm is shown in Algorithm 2. Note that $r_{\max}$ appears in the `steer` function.

Algorithm 2: Rapidly-exploring Random Tree Star (RRT*)

$V \leftarrow \{q_{\text{initial}}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, k$ **do**
  $q_{\text{rand}} \leftarrow \texttt{random\_free}(Q_{\text{free}})$;
  $q_{\text{nearest}} \leftarrow \texttt{nearest}(V, q_{\text{rand}})$;
  $q_{\text{new}} \leftarrow \texttt{steer}(q_{\text{nearest}}, q_{\text{rand}})$;
  if $\texttt{collision\_free}(q_{\text{nearest}}, q_{\text{new}})$ **then**
    $N \leftarrow \text{card}(V)$;
    $r \leftarrow \min\left(\left\{\gamma_{\text{RRT}^*}\left(\frac{\log(N)}{N}\right)^{\frac{1}{n}}, r_{\max}\right\}\right)$;
    $V_{\text{near}} \leftarrow \texttt{near}(V, q_{\text{new}}, r)$;
    $V \leftarrow V \cup \{q_{\text{new}}\}$;
    $q_{\min} \leftarrow q_{\text{nearest}}$;
    $c_{\min} \leftarrow \texttt{cost}(G, q_{\text{nearest}}) + \texttt{path\_cost}(q_{\text{nearest}}, q_{\text{new}})$;
    foreach $q_{\text{near}} \in V_{\text{near}}$ **do**
      $c \leftarrow \texttt{cost}(G, q_{\text{near}}) + \texttt{path\_cost}(q_{\text{near}}, q_{\text{new}})$;
      if $\texttt{collision\_free}(q_{\text{near}}, q_{\text{new}}) \wedge (c < c_{\min})$ **then**
        $q_{\min} \leftarrow q_{\text{near}}$;
        $c_{\min} \leftarrow c$;
      end
    end
    $E \leftarrow E \cup \{(q_{\min}, q_{\text{new}})\}$;
    foreach $q_{\text{near}} \in V_{\text{near}}$ **do**
      $c \leftarrow \texttt{cost}(G, q_{\text{new}}) + \texttt{path\_cost}(q_{\text{new}}, q_{\text{near}})$;
      if $\texttt{collision\_free}(q_{\text{new}}, q_{\text{near}}) \wedge \left(c < \texttt{cost}(G, q_{\text{near}})\right)$ **then**
        $q_{\text{par}} \leftarrow \texttt{parent}(G, q_{\text{near}})$;
      end
      $E \leftarrow \left(E \setminus \{(q_{\text{par}}, q_{\text{near}})\}\right) \cup \{(q_{\text{new}}, q_{\text{near}})\}$;
    end
  end
end
return $G = (V, E)$;

Figure 3.2 shows a visualisation of the RRT* algorithm, similar to Figure 3.1.

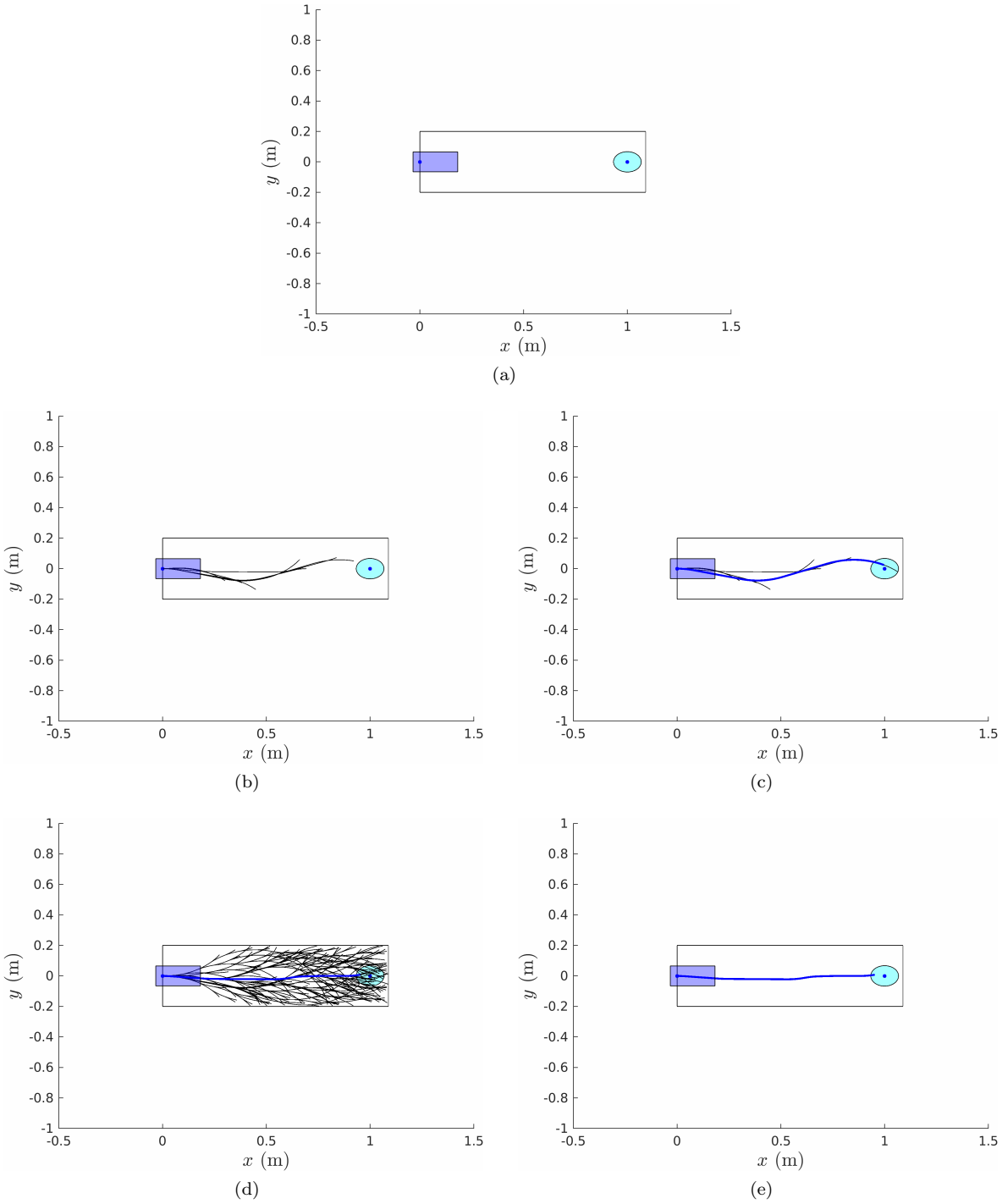


Figure 3.2: A visualisation of the RRT* algorithm, where $\phi_{\max} = 25°$ and $l = 0.15$ m. (a) Setup. (b) Searching for a solution. (c) A solution has been found. (d) Search ended at $\text{card}(V) = 500$. (e) The final path to the goal.

3.5 The Online RRT* Algorithm

The motion planning algorithm used by the software presented in this dissertation is the online RRT* algorithm presented in [14]. Once begun, the algorithm begins an initial planning phase, where the RRT* algorithm runs until the robot must begin to move towards the goal. The algorithm then begins an iterative planning phase, where, given a trajectory $\gamma : [0, \tau] \to Q_{\text{free}}$ generated by the RRT* algorithm, the robot executes an initial portion of the trajectory, known as the committed trajectory $\gamma : [0, \tau_{\text{com}}] \to Q_{\text{free}}$, until a committed time $\tau_{\text{com}} \in \mathbb{R}_{\geq 0}$. During the execution of the committed trajectory, the algorithm deletes each of its branches and declares a new tree graph root at the end of the committed trajectory $\gamma(\tau_{\text{com}})$, shielding the committed trajectory from any further modifications. This phase is repeated until the robot reaches the goal. The algorithm also uses the `branch_and_bound` function to build the tree graph more efficiently.

The functions present in the online RRT* algorithm but not the RRT* algorithm are defined below.

Let `cost_to_go` : $(Q_{\text{goal}}, q) \mapsto c$ denote a function that takes a goal region $Q_{\text{goal}} \subset Q_{\text{free}}$ and a configuration $q \in Q$, and returns the cost $c \in \mathbb{R}_{\geq 0}$ of the path from q to Q_{goal}.

Let `branch_and_bound` : $(G, Q_{\text{goal}}, v, v_{\text{min}}) \mapsto G$ denote a function that takes a tree graph $G = (V, E)$, where $V \subset Q$ denotes a set of vertices and E denotes a set of edges, a goal region $Q_{\text{goal}} \subset Q_{\text{free}}$, a vertex $v \in V$ and the vertex v_{min} that lies in the goal region and has the lowest-cost trajectory that reaches Q_{goal} through G, deletes the set of vertices for which $\texttt{cost}(v) + \texttt{cost\_to\_go}(Q_{\text{goal}}, v) \geq \texttt{cost}(v_{\text{min}})$, and returns G.

Figure 3.3 shows a visualisation of the online RRT* algorithm, similar to Figures 3.1 and 3.2.

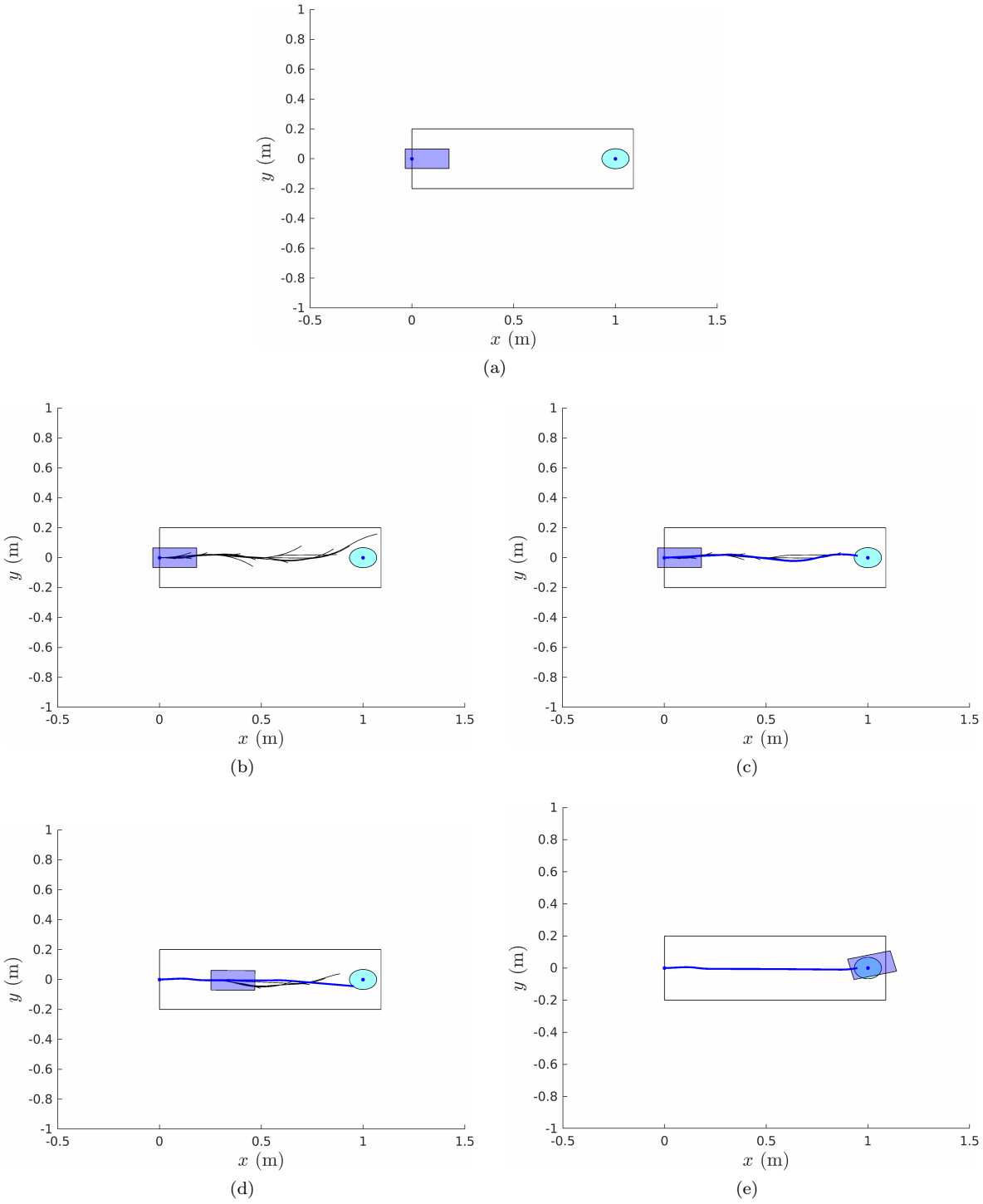


Figure 3.3: A visualisation of the online RRT* algorithm, where $\phi_{\max} = 25°$ and $l = 0.15$ m. (a) Setup. (b) Searching for a solution. (c) A solution has been found. (d) Searching for a better solution while tracking the prevailing best solution. (e) The robot has reached the goal.

3.6 Dubins Paths

The information in this section is derived from [11], [12] and [32].

Given configurations $q_1, q_2 \in Q_{\text{free}}$, where, in coordinates (x, y, θ),

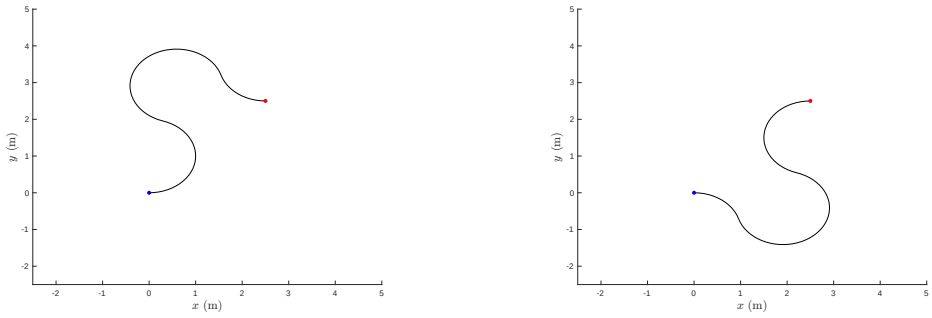
$$q_1 = (x_1, y_1, \theta_1), \tag{3.6}$$
$$q_2 = (x_2, y_2, \theta_2), \tag{3.7}$$

the shortest path that can be traversed by the Dubins car between q_1 and q_2, without colliding with obstacles, belongs to a set of paths known as Dubins paths, shown in Table 3.1, where L and R denote motion primitives corresponding to left and right forward motions with a minimum turning radius, respectively, S denotes a motion primitive corresponding to a straight forward motion and C denotes an L or R motion primitive.

Path family	Path types
CCC	LRL, RLR
CSC	LSL, LSR, RSL, RSR

Table 3.1: Dubins path families and types.

Figures 3.4 and 3.5 show the set of Dubins paths, where $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ are shown in blue and red, respectively.



(a) (b)

Figure 3.4: CCC Dubins paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) LRL. (b) RLR.

Let

$$\alpha = (\theta_1 - \gamma) \bmod 2\pi, \tag{3.8}$$
$$\beta = (\theta_2 - \gamma) \bmod 2\pi, \tag{3.9}$$

where

$$\gamma = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right). \tag{3.10}$$

While the shortest Dubins path between q_1 and q_2 can be found by comparing the length of each Dubins path that exists between them, it was shown in [32] that if

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} > \sqrt{4 - \Big(\big|\cos(\alpha)\big| + \big|\cos(\beta)\big|\Big)^2} + \big|\sin(\alpha)\big| + \big|\sin(\beta)\big|, \tag{3.11}$$

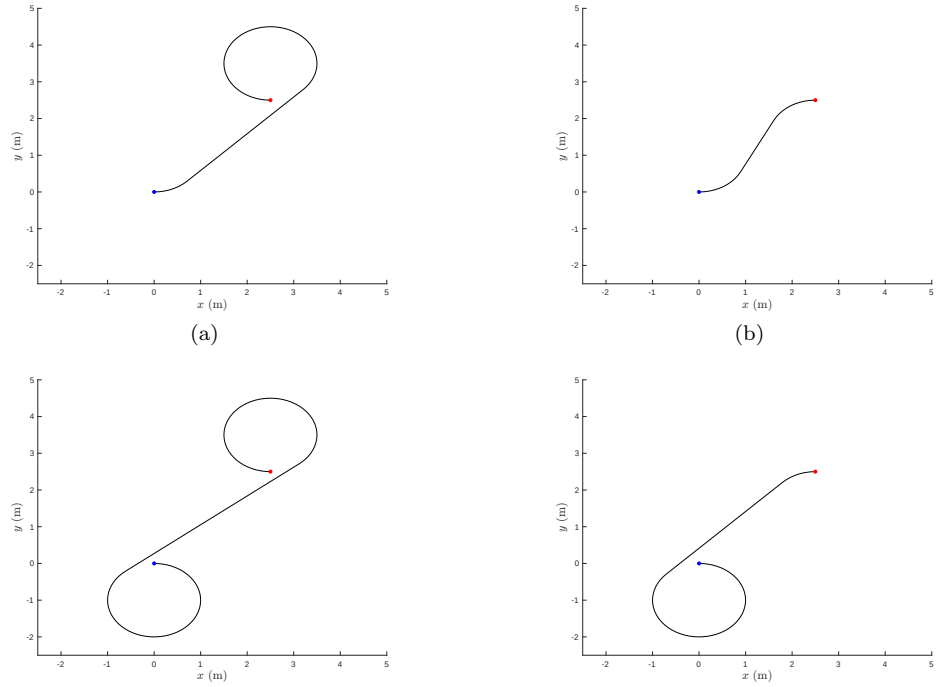


Figure 3.5: CSC Dubins paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) LSL. (b) LSR. (c) RSL. (d) RSR.

then the shortest Dubins path would belong to the set of CSC Dubins paths.

It was also shown that the shortest Dubins path could be found using Table 3.2, where

$$S_{12} = p_{RSR} - p_{RSL} - 2(q_{RSL} - \pi r_{\min}), \tag{3.12}$$
$$S_{13} = t_{RSR} - \pi r_{\min}, \tag{3.13}$$
$$S_{14}^1 = t_{RSR} - \pi r_{\min}, \tag{3.14}$$
$$S_{14}^2 = q_{RSR} - \pi r_{\min}, \tag{3.15}$$
$$S_{21} = p_{LSL} - p_{RSL} - 2(t_{RSL} - \pi r_{\min}), \tag{3.16}$$
$$S_{22}^1 = p_{LSL} - p_{RSL} - 2(t_{RSL} - \pi r_{\min}), \tag{3.17}$$
$$S_{22}^2 = p_{RSR} - p_{RSL} - 2(q_{RSL} - \pi r_{\min}), \tag{3.18}$$
$$S_{24} = q_{RSR} - \pi r_{\min}, \tag{3.19}$$
$$S_{31} = q_{LSL} - \pi r_{\min}, \tag{3.20}$$
$$S_{33}^1 = p_{RSR} - p_{LSR} - 2(t_{LSR} - \pi r_{\min}), \tag{3.21}$$
$$S_{33}^2 = p_{LSL} - p_{LSR} - 2(q_{LSR} - \pi r_{\min}), \tag{3.22}$$
$$S_{34} = p_{RSR} - p_{LSR} - 2(t_{LSR} - \pi r_{\min}), \tag{3.23}$$
$$S_{41}^1 = t_{LSL} - \pi r_{\min}, \tag{3.24}$$
$$S_{41}^2 = q_{LSL} - \pi r_{\min}, \tag{3.25}$$
$$S_{42} = t_{LSL} - \pi r_{\min}, \tag{3.26}$$
$$S_{43} = p_{LSL} - p_{LSR} - 2(q_{LSR} - \pi r_{\min}), \tag{3.27}$$

where

$$r_{\min} = \frac{l}{\tan(\phi_{\max})} \tag{3.28}$$

denotes the minimum turning radius of the Dubins car, $t_{LSL}, t_{LSR}, t_{RSL}, t_{RSR} \in \mathbb{R}_{\geq 0}$ denote the path lengths of the first motion primitive for LSL, LSR, RSL and RSR Dubins paths, respectively, $p_{LSL}, p_{LSR}, p_{RSL}, p_{RSR} \in \mathbb{R}_{\geq 0}$ denote the path lengths of the second motion primitive for LSL, LSR, RSL and RSR Dubins paths, respectively, and $q_{LSL}, q_{LSR}, q_{RSL}, q_{RSR} \in \mathbb{R}_{\geq 0}$ denote the path lengths of the third motion primitive for LSL, LSR, RSL and RSR Dubins paths, respectively.

	$0 \leq \beta \leq \frac{\pi}{2}$	$\frac{\pi}{2} \leq \beta \leq \pi$	$\pi \leq \beta \leq \frac{3\pi}{2}$	$\frac{3\pi}{2} \leq \beta < 2\pi$ or $\beta = 0$
$0 \leq \alpha \leq \frac{\pi}{2}$	RSL	if $S_{12} < 0$: RSR else if $S_{12} > 0$: RSL	if $S_{13} < 0$: RSR else if $S_{13} > 0$: LSR	if $S_{14}^1 > 0$: LSR else if $S_{14}^2 > 0$: RSL else: RSR
$\frac{\pi}{2} \leq \alpha \leq \pi$	if $S_{21} < 0$: LSL else if $S_{21} > 0$: RSL	if $\alpha > \beta$ and $S_{22}^1 < 0$: LSL else if $\alpha > \beta$ and $S_{22}^1 > 0$: RSL else if $\alpha < \beta$ and $S_{22}^2 < 0$: RSR else if $\alpha < \beta$ and $S_{22}^2 > 0$: RSL	RSR	if $S_{24} < 0$: RSR else if $S_{24} > 0$: RSL
$\pi \leq \alpha \leq \frac{3\pi}{2}$	if $S_{31} < 0$: LSL else if $S_{31} > 0$: LSR	LSL	if $\alpha < \beta$ and $S_{33}^1 < 0$: RSR else if $\alpha < \beta$ and $S_{33}^1 > 0$: LSR else if $\alpha > \beta$ and $S_{33}^2 < 0$: LSL else if $\alpha > \beta$ and $S_{33}^2 > 0$: LSR	if $S_{34} < 0$: RSR else if $S_{34} > 0$: LSR
$\frac{3\pi}{2} \leq \alpha < 2\pi$ or $\alpha = 0$	if $S_{41}^1 > 0$: RSL else if $S_{41}^2 > 0$: LSR else: LSL	if $S_{42} < 0$: LSL else if $S_{42} > 0$: RSL	if $S_{43} < 0$: LSL else if $S_{43} > 0$: LSR	LSR

Table 3.2: Decision table for finding the shortest Dubins path given that (3.11) is true.

Equations for the path lengths of the motion primitives for each Dubins path type are provided in [32]. However, a more compact set of equations that describe the set of Dubins paths is presented in the subsequent section.

3.7 Generating Dubins Paths

Following the approach taken in [33], this section presents a compact set of closed-form equations that describe the set of Dubins paths. They are compact in that they describe the set of Dubins path families as opposed to the set of Dubins path types. The information in this section is derived from [19] and [11].

Let $SE(n)$ denote the special Euclidean group in dimension n and $\mathfrak{se}(n)$ denote the Lie algebra of $SE(n)$.

For $i \in \{1, 2, 3, 4\}$, let $q_i = (x_i, y_i, \theta_i) \in \mathbb{R}^2 \times \mathbb{S}^1$ denote the configuration of a Dubins car such that, for $i \in \{1, 2, 3\}$,

$$g_{i+1} = g_i \exp(X_i \Delta t_i), \tag{3.29}$$

where, noting that $\mathbb{R}^2 \times \mathbb{S}^1 \simeq SE(2)$,

$$g_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & x_i \\ \sin(\theta_i) & \cos(\theta_i) & y_i \\ 0 & 0 & 1 \end{bmatrix} \in SE(2) \tag{3.30}$$

denotes the matrix representation of the configuration of the Dubins car,

$$X_i = \begin{bmatrix} 0 & -\omega_i & v_i \\ \omega_i & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathfrak{se}(2) \tag{3.31}$$

denotes the matrix representation of the planar body twist of the Dubins car between g_i and g_{i+1}, and $\Delta t_i \in \mathbb{R}_{\geq 0}$ denotes the duration of the motion from g_i to g_{i+1}, where $v_i \in \mathbb{R}_{>0}$ and

$$\omega_i = \begin{cases} \frac{v_i}{r_i} & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0, \end{cases} \tag{3.32}$$

where $\phi_i \in [-\phi_{\max}, \phi_{\max}]$ and

$$r_i = \frac{l}{\tan(\phi_i)}. \tag{3.33}$$

Therefore,

$$g_4 = g_1 \prod_{i=1}^{3} \exp(X_i \Delta t_i). \tag{3.34}$$

Note that

$$\exp(X_i \Delta t_i) = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) & \Delta x_i \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) & \Delta y_i \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.35}$$

where

$$\Delta x_i = \begin{cases} r_i \sin(\Delta\theta_i) & \text{if } \phi_i \neq 0, \\ v_i \Delta t_i & \text{if } \phi_i = 0, \end{cases} \tag{3.36}$$

$$\Delta y_i = \begin{cases} r_i\big(1 - \cos(\Delta\theta_i)\big) & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0, \end{cases} \tag{3.37}$$

$$\Delta\theta_i = \begin{cases} \frac{v_i \Delta t_i}{r_i} & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0 \end{cases} \tag{3.38}$$

denote the changes along the x, y and θ directions of $\{b\}$, respectively. Therefore, for $i \in \{1, 2, 3\}$,

$$\Delta t_i = \begin{cases} \frac{\Delta\theta_i r_i}{v_i} & \text{if } \phi_i \neq 0, \\ \frac{\Delta x_i}{v_i} & \text{if } \phi_i = 0, \end{cases} \tag{3.39}$$

where Δx_i and $\Delta\theta_i$ may be determined using (3.34). Given that g_1 and g_4 are known, it will be beneficial to repose (3.34) as

$$g_1^{-1} g_4 = \prod_{i=1}^{3} \exp(X_i \Delta t_i), \tag{3.40}$$

where

$$g_1^{-1} g_4 = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & \Delta x \\ \sin(\Delta\theta) & \cos(\Delta\theta) & \Delta y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.41}$$

where

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 \\ -\sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_4 - x_1 \\ y_4 - y_1 \\ \theta_4 - \theta_1 \end{bmatrix}. \tag{3.42}$$

Necessary conditions for Dubins paths to exist between g_1 and g_4 are shown in Tables 3.3 and 3.4.

Let

$$R_1 = \begin{bmatrix} \cos(\Delta\theta_1) & -\sin(\Delta\theta_1) \\ \sin(\Delta\theta_1) & \cos(\Delta\theta_1) \end{bmatrix}. \tag{3.43}$$

Path family	$\lvert\Delta\theta_1\rvert$	$\lvert\Delta\theta_2\rvert$	$\lvert\Delta\theta_3\rvert$
CCC	$[0, 2\pi)$	$(\pi, 2\pi)$	$[0, 2\pi)$
CSC	$[0, 2\pi)$	0	$[0, 2\pi)$

Table 3.3: Dubins path turning circle angles.

Path family	ϕ_1	ϕ_2	ϕ_3
CCC	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$
CSC	$\{-\phi_{\max}, \phi_{\max}\}$	0	$\{-\phi_{\max}, \phi_{\max}\}$

Table 3.4: Dubins path steering angles.

For CCC Dubins paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = r_1\left(\begin{bmatrix}\sin(\Delta\theta)\\ 1-\cos(\Delta\theta)\end{bmatrix} - 2R_1\begin{bmatrix}\sin(\Delta\theta_2)\\ 1-\cos(\Delta\theta_2)\end{bmatrix}\right), \tag{3.44}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3. \tag{3.45}$$

Therefore, to generate CCC Dubins paths between g_1 and g_4, for $i \in \{1, 2, 3\}$, let

$$\Delta\theta_i = \operatorname{sgn}(\phi_i)\big(\operatorname{sgn}(\phi_i)\Delta\theta_i' \bmod 2\pi\big), \tag{3.46}$$

where

$$\Delta\theta_1' = 2\arctan\left(\frac{\sqrt{\alpha^2+\beta^2}-\alpha}{\beta}\right) - \frac{\Delta\theta_2'}{2}, \tag{3.47}$$

$$\Delta\theta_2' = \operatorname{sgn}(\phi_1)\arccos\left(1 - \frac{\alpha^2+\beta^2}{2}\right) \bmod 2\pi, \tag{3.48}$$

$$\Delta\theta_3' = \Delta\theta - \Delta\theta_1' - \Delta\theta_2', \tag{3.49}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix}\sin(\Delta\theta)\\ 1-\cos(\Delta\theta)\end{bmatrix} - \frac{1}{r_1}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right). \tag{3.50}$$

For CSC Dubins paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = R_1\begin{bmatrix}\Delta x_2\\ r_3 - r_1\end{bmatrix} + \begin{bmatrix}r_3\sin(\Delta\theta)\\ r_1 - r_3\cos(\Delta\theta)\end{bmatrix}, \tag{3.51}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_3. \tag{3.52}$$

Therefore, to generate CSC Dubins paths between g_1 and g_4 using (3.39), let

$$\Delta x_2 = \sqrt{\alpha^2+\beta^2-(r_3-r_1)^2} \tag{3.53}$$

and, for $i \in \{1, 3\}$,

$$\Delta\theta_i = \operatorname{sgn}(\phi_i)\big(\operatorname{sgn}(\phi_i)\Delta\theta_i' \bmod 2\pi\big), \tag{3.54}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} - \begin{bmatrix}r_3\sin(\Delta\theta)\\ r_1 - r_3\cos(\Delta\theta)\end{bmatrix}, \tag{3.55}$$

$$\Delta\theta_1' = 2\arctan\left(\frac{\Delta x_2 - \alpha}{\beta + r_3 - r_1}\right), \tag{3.56}$$

$$\Delta\theta_3' = \Delta\theta - \Delta\theta_1'. \tag{3.57}$$

Note that

$$\Delta t_i = \begin{cases} \frac{\Delta\theta_i' r_i \bmod 2\pi|r_i|}{v_i} & \text{if } \phi_i \neq 0, \\ \frac{\Delta x_i}{v_i} & \text{if } \phi_i = 0 \end{cases} \tag{3.58}$$

may be used instead of (3.39).

These equations were used to generate the paths presented in Figures 3.4 and 3.5, demonstrating their efficacy. For this dissertation, let $v_i = v_{\max}$, for $i \in \{1, 2, 3\}$, so that the time required for the robot to reach its goal is minimised.

3.8 Summary

To summarise this chapter:

- In robotics, planning involves the use of localisation information to plan a path or trajectory through a robot's environment. Planning often refers to either motion planning or path planning. Given that this dissertation is using a kinematic model to describe the behaviour of a car-like robot, the focus of this chapter is kinematic motion planning.
- For a computer to solve the motion planning problem, a motion planning algorithm needs to be provided to it in the form of code. For this dissertation, a sampling method was chosen because it would not require an exact representation of the obstacle-free region, it would be relatively easy to implement, it would have relatively low memory requirements, and it would avoid the local minima problem. The motion planning algorithm used by the software presented in this dissertation is an online RRT* algorithm.
- The online RRT* algorithm attempts to improve upon the solutions generated by the RRT* algorithm while the robot is tracking the prevailing best solution, where RRT* is an asymptotically-optimal variation of the RRT algorithm, which is a single-query sampling-based motion planning algorithm.
- In the `path_cost` function of the online RRT* algorithm, the shortest path between two configurations needs to be found. For a robot being described by the Dubins and Reeds-Shepp car models, this path will be a Dubins and Reeds-Shepp path, respectively.
- To generate Dubins paths, a compact set of closed-form equations that describe the set of Dubins paths is presented in this dissertation.

Chapter 4

Control

In robotics, control involves the use of a robot's actuators in such a way that the robot achieves a certain goal. For this dissertation, the goal is for the robot to closely track a desired trajectory.

Let $\{d\}$ denote the desired frame of the robot, as shown in Figure 4.1.

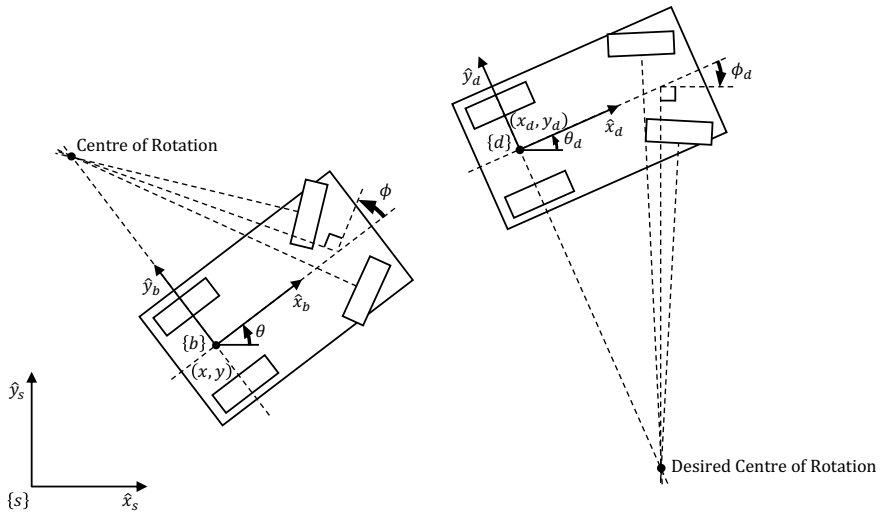


Figure 4.1: A car-like robot in its current configuration, represented with solid lines, and in its desired configuration, represented with short dashed lines.

Let (γ_d, u_d) denote the desired controlled trajectory generated by the motion planning algorithm, where $\gamma_d : [0, \tau] \to Q$ denotes the desired trajectory, represented in coordinates (x_d, y_d, θ_d) by $[0, \tau] \ni t \mapsto \big(x_d(t), y_d(t), \theta_d(t)\big)$, and $u_d : [0, \tau] \to \big\{(u_d)_v, (u_d)_\phi\big\}$ denotes the desired control, where $(u_d)_v = (v_d)_{\max}$ and $(u_d)_\phi \in \{-(\phi_d)_{\max}, 0, (\phi_d)_{\max}\}$ denote the desired control variables, defined by

$$\dot{x}_d = (u_d)_v \cos(\theta_d), \tag{4.1}$$

$$\dot{y}_d = (u_d)_v \sin(\theta_d), \tag{4.2}$$

$$\dot{\theta}_d = \frac{(u_d)_v \tan\big((u_d)_\phi\big)}{l}, \tag{4.3}$$

where $(v_d)_{\max} \in \mathbb{R}_{>0}$ and $(\phi_d)_{\max} \in \big(0, \frac{\pi}{2}\big)$ denote the maximum desired velocity and steering angle, respectively. Note that $(u_d)_v = (v_d)_{\max}$ so that the time required for the robot to reach its goal is minimised and $(u_d)_\phi \in \{-(\phi_d)_{\max}, 0, (\phi_d)_{\max}\}$ because γ_d is composed of Dubins paths.

Let

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos(\theta_d) & \sin(\theta_d) & 0 \\ -\sin(\theta_d) & \cos(\theta_d) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{bmatrix} \tag{4.4}$$

denote the configuration error of the robot in $\{d\}$. Therefore,

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \dot{\theta}_d \begin{bmatrix} y_e \\ -x_e \\ \theta_e \end{bmatrix} + \begin{bmatrix} \cos(\theta_d) & \sin(\theta_d) & 0 \\ -\sin(\theta_d) & \cos(\theta_d) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} - \dot{x}_d \\ \dot{y} - \dot{y}_d \\ \dot{\theta} - \dot{\theta}_d \end{bmatrix}. \tag{4.5}$$

Therefore, to closely track γ_d, a control that renders the origin of the system (4.5) globally asymptotically stable is desired.

4.1 Car-like Robot Control Method

The information in this section is derived from [12], [34] and [35].

The following control method is based on the unicycle-like robot control method presented in [34], which, through a Lyapunov-like analysis using Barbălat's Lemma, is shown to provide global asymptotic stability. Barbălat's Lemma, as stated in [35], is given in the following theorem.

Theorem 4.1.1 (Barbălat's Lemma) *If the differentiable function $f(t)$ has a finite limit as $t \to \infty$, and if $\dot{f}(t)$ is uniformly continuous, then $\dot{f}(t) \to 0$ as $t \to \infty$.*

Let

$$z_1 = x_e, \tag{4.6}$$
$$z_2 = y_e, \tag{4.7}$$
$$z_3 = \tan(\theta_e), \tag{4.8}$$
$$u_1 = u_v \cos(\theta_e) - (u_d)_v, \tag{4.9}$$
$$u_2 = \frac{\dot{\theta}_e}{\cos^2(\theta_e)}, \tag{4.10}$$
$$u_\omega = \dot{\theta} \tag{4.11}$$
$$= \frac{u_v \tan(u_\phi)}{l}, \tag{4.12}$$

where $\theta_e \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. Therefore,

$$\dot{z}_1 = \dot{\theta}_d z_2 + u_1, \tag{4.13}$$
$$\dot{z}_2 = -\dot{\theta}_d z_1 + (u_d)_v z_3 + u_1 z_3, \tag{4.14}$$
$$\dot{z}_3 = u_2. \tag{4.15}$$

Let

$$V = \frac{(z_1)^2 + (z_2)^2 + \frac{(z_3)^2}{k_2}}{2} \tag{4.16}$$

denote a candidate Lyapunov function, where $k_2 \in \mathbb{R}_{>0}$. Given that, with reference to (4.13)–(4.15),

$$\dot{V} = u_1(z_1 + z_2 z_3) + z_3\left((u_d)_v z_2 + \frac{u_2}{k_2}\right), \tag{4.17}$$

let

$$u_1 = -k_1\big|(u_d)_v\big|(z_1 + z_2 z_3), \tag{4.18}$$

$$u_2 = -k_2(u_d)_v z_2 - k_3\big|(u_d)_v\big|z_3, \tag{4.19}$$

where $k_1, k_3 \in \mathbb{R}_{>0}$, so that

$$\dot{V} = -k_1\big|(u_d)_v\big|(z_1 + z_2 z_3)^2 - \frac{k_3\big|(u_d)_v\big|(z_3)^2}{k_2}, \tag{4.20}$$

which renders V a monotonically decreasing function and

$$\lim_{t\to\infty} V = V_{\lim}, \tag{4.21}$$

where $V_{\lim} \in \mathbb{R}_{\geq 0}$. Note that, because $(u_d)_v$ is continuous and $\dot{(u_d)}_v$ is bounded, $\big|(u_d)_v\big|$ is uniformly continuous, rendering $\dot{V}$ uniformly continuous. Therefore, by Theorem 4.1.1,

$$\lim_{t\to\infty} \dot{V} = 0, \tag{4.22}$$

which, with reference to (4.20), implies that

$$\lim_{t\to\infty} (u_d)_v(z_1 + z_2 z_3) = 0, \tag{4.23}$$

$$\lim_{t\to\infty} (u_d)_v z_3 = 0, \tag{4.24}$$

which implies that

$$\lim_{t\to\infty} (u_d)_v z_1 = 0. \tag{4.25}$$

Note that, with reference to (4.15) and (4.19),

$$\frac{d\big((u_d)_v\big)^2 z_3}{dt} = 2(u_d)_v\dot{(u_d)}_v z_3 - k_2\big((u_d)_v\big)^3 z_2 - k_3\big|(u_d)_v\big|\big((u_d)_v\big)^2 z_3, \tag{4.26}$$

which is uniformly continuous. Therefore, by Theorem 4.1.1,

$$\lim_{t\to\infty} (u_d)_v z_2 = 0. \tag{4.27}$$

Therefore, with reference to (4.24), (4.25) and (4.27), if

$$\lim_{t\to\infty} (u_d)_v \neq 0, \tag{4.28}$$

then (4.18) and (4.19) would render the origin of the system (4.13)–(4.15) globally asymptotically stable. Noting that, for this dissertation, $(u_d)_v = (v_d)_{\max}$, it can be seen that this condition has been met. Therefore, by substituting (4.6)–(4.11) into (4.18) and (4.19), noting that $\theta_e \in \big(-\frac{\pi}{2}, \frac{\pi}{2}\big)$, it can be seen that

$$u_v = \frac{(u_d)_v - k_1\big|(u_d)_v\big|\big(x_e + y_e\tan(\theta_e)\big)}{\cos(\theta_e)}, \tag{4.29}$$

$$u_\omega = \dot{\theta}_d - \Big(k_2(u_d)_v y_e + k_3\big|(u_d)_v\big|\tan(\theta_e)\Big)\cos^2(\theta_e) \tag{4.30}$$

render the origin of the system (4.5) globally asymptotically stable.

The following descriptions of (4.29) and (4.30) were given in [12]. In reference to (4.29),

$$-\frac{k_1\big|(u_d)_v\big|\big(x_e + y_e\tan(\theta_e)\big)}{\cos(\theta_e)} \tag{4.31}$$

attempts to reduce x_e and y_e by speeding up or slowing down the robot. In reference to (4.30),

$$-k_2(u_d)_v y_e \cos^2(\theta_e) \tag{4.32}$$

attempts to reduce y_e in the future by turning the robot such that the x-axis of $\{b\}$ points towards the origin of $\{d\}$ and

$$-k_3\big|(u_d)_v\big| \tan(\theta_e) \cos^2(\theta_e) \tag{4.33}$$

attempts to reduce θ_e by turning the robot.

Note that, for practical purposes, bounds may need to be placed on u_v and u_ω. Therefore, let

$$(u_v)_b = \max\Big(\min\big(u_v, v_{\max}\big), v_{\min}\Big), \tag{4.34}$$

$$(u_\omega)_b = \max\Big(\min\big(u_\omega, \omega_{\max}\big), -\omega_{\max}\Big) \tag{4.35}$$

denote bounded versions of u_v and u_ω, respectively, where

$$\omega_{\max} = \frac{\big|(u_v)_b\big| \tan(\phi_{\max})}{l}. \tag{4.36}$$

However, to satisfy the Dubins car model, let

$$(u_v)_b = \max\Big(\min\big(u_v, v_{\max}\big), 0\Big). \tag{4.37}$$

Note that (4.37) may lead to overshoot that the robot cannot recover from, depending on, for example, the curvature of the desired trajectory, the minimum turning radius of the robot and how much Q_{obs} was enlarged by.

4.2 Duckiebot Control Method

The information in this section is derived from [12].

To control the Duckiebot using the above control method, one could convert u_v and u_ω into control variables for the Duckiebot's motorised wheels and then convert these control variables into PWM signals, which may be sent from the Duckietown Hut to each of the Duckiebot's motors.

Let $\varphi_l, \varphi_r \in \mathbb{S}^1$ denote the rolling angles of the Duckiebot's left and right motorised wheels, respectively. Therefore, by inspection,

$$\dot{\varphi}_l R = u_v - \frac{w u_\omega}{2}, \tag{4.38}$$

$$\dot{\varphi}_r R = u_v + \frac{w u_\omega}{2}, \tag{4.39}$$

where $R \in \mathbb{R}_{>0}$ denotes the radius of the Duckiebot's motorised wheels and $w \in \mathbb{R}_{>0}$ denotes the track of the Duckiebot. Therefore,

$$\dot{\varphi}_l = \frac{2u_v - u_\omega w}{2R}, \tag{4.40}$$

$$\dot{\varphi}_r = \frac{2u_v + u_\omega w}{2R}. \tag{4.41}$$

Let

$$u_l = k_l \dot{\varphi}_l, \tag{4.42}$$

$$u_r = k_r \dot{\varphi}_r \tag{4.43}$$

denote the control variables for the Duckiebot's left and right motorised wheels, respectively, where $k_l, k_r \in \mathbb{R}_{>0}$ denote gains for the left and right motorised wheels, respectively, which can be tuned to reduce any unwanted difference between the output of the two motors when sent the same signal. Therefore, let

$$(u_l)_b = \frac{k_l\big(2(u_v)_b - (u_\omega)_b w\big)}{2R}, \tag{4.44}$$

$$(u_r)_b = \frac{k_r\big(2(u_v)_b + (u_\omega)_b w\big)}{2R} \tag{4.45}$$

denote the bounded versions of u_l and u_r, respectively. Figure 4.2 shows the control method used in this dissertation.

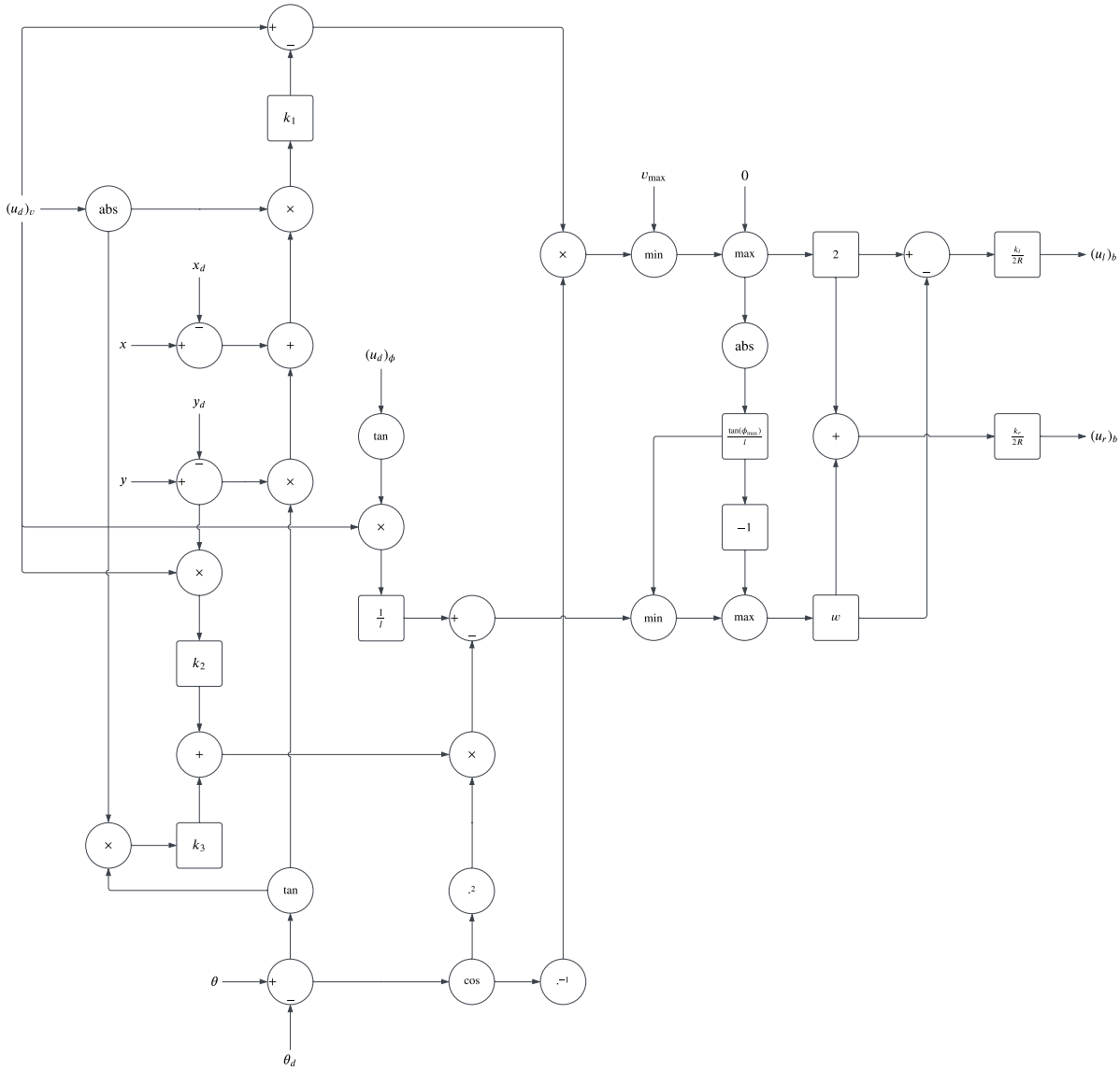


Figure 4.2: The control method used in this dissertation, where rounded rectangles represent multiplication with the value inside of them, circles represent all other operations and arrows represent the flow of information.

4.3 Summary

To summarise this chapter:

- In robotics, control involves the use of a robot's actuators in such a way that the robot achieves a certain goal. For this dissertation, the goal is for the robot to closely track a desired trajectory.
- The chosen car-like robot control method is based on a unicycle-like robot control method, which, through a Lyapunov-like analysis using Barbalat's Lemma, is shown to provide global asymptotic stability.
- To control the Duckiebot, the control variables of the car-like robot control method are converted into control variables for the Duckiebot's motorised wheels. These control variables are then converted into PWM signals, which may be sent from the Duckietown Hut to each of the Duckiebot's motors.

Chapter 5

Perception

In robotics, perception involves the use of a robot's sensors to take measurements of its environment. As explained in [36], proprioceptive sensors, such as a wheel encoder, gyroscope and battery sensor, measure the state of a robot, whereas exteroceptive sensors, such as a camera, magnetometer and time-of-flight sensor, measure the state of the world with respect to the robot. Note that an accelerometer is both a proprioceptive sensor, as it can measure the acceleration due to the motion of the robot, and an exteroceptive sensor, as it can measure the acceleration due to gravity. To achieve the objective of this dissertation, the use of wheel encoders for odometry and a front-facing onboard camera for landmark observations will suffice, as the information they provide is sufficient for reliable localisation, although this could be improved by using additional sensor information.

5.1 Wheel Encoders

Let $n \in \mathbb{Z}$ denote the nett number of ticks detected by one of the robot's wheel encoders, which is the number of ticks detected going in one direction minus the number of ticks detected going in the other direction, over a period. Therefore, let

$$\Delta\varphi \approx \frac{2\pi n}{N} \tag{5.1}$$

denote the change in the rolling angle of the wheel whose motion is being measured over that period, where $N \in \mathbb{R}_{>0}$ denotes the number of ticks per revolution of the wheel encoder. Note that the approximation in (5.1) is due to the discretisation of the rolling angle.

5.2 Camera

The information in this section is derived from [36].

For this dissertation, the discrete pixel central projection model, shown in Figure 5.1, is used as a model of the robot's camera.

Let $\{c\}$ denote the camera frame, $f \in \mathbb{R}_{>0}$ denote the focal length of the camera's lens, $P \in \mathbb{R}^3$ denote the position of a point in the world relative to $\{c\}$ and $p \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ denote the projection of P onto the image plane whose principal point $(u_0, v_0) \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ is located a distance f along the z-axis of $\{c\}$ such that

$$\tilde{p} \approx KX\tilde{P}, \tag{5.2}$$

where $\tilde{p} \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \times \{1\}$ denotes the homogeneous form of p,

$$K = \begin{bmatrix} \frac{f}{w_u} & s & u_0 \\ 0 & \frac{f}{w_v} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

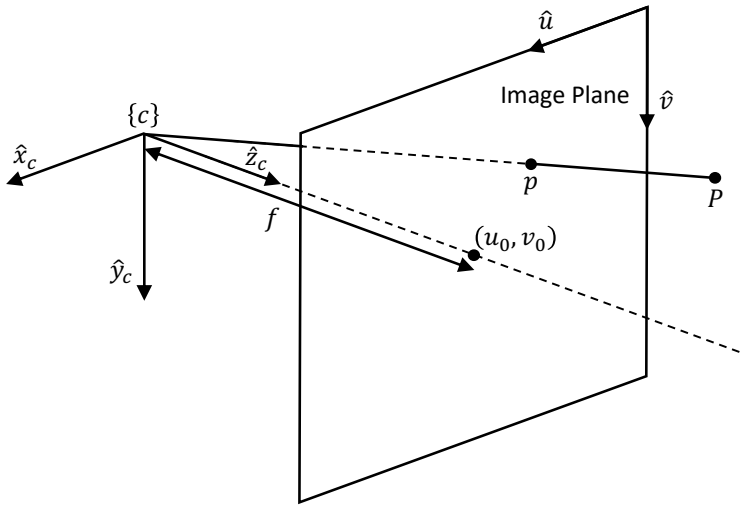


Figure 5.1: The discrete pixel central projection model with camera frame $\{c\}$, focal length f, world point P, image point p and principal point (u_0, v_0).

denotes the camera parameter matrix,

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.4}$$

denotes the projection matrix and $\tilde{P} \in \mathbb{R}^3 \times \{1\}$ denotes the homogeneous form of P, where $w_u, w_v \in \mathbb{R}_{>0}$ denote the lengths of one of the camera's pixels along the u- and v-axes, respectively, and $s \in \mathbb{R}$ denotes the camera's skew coefficient.

Note that homogeneous transformations may be used to represent $\tilde{P}$ in $\{s\}$ or $\{b\}$ and that the approximation in (5.2) is due to distortions created by lens imperfections, with geometric distortions often being the most problematic for robotic applications. Geometric distortions include radial distortions, which cause image points to be displaced along radial lines from the principal point, and tangential distortions, which occur at right angles to these radial lines but are generally less significant than radial distortions. Therefore, let

$$p_d = p + \delta_r + \delta_t \tag{5.5}$$

denote the distorted image point, where

$$\delta_r \approx \sum_{i=1}^{3} k_i r^{2i} p \tag{5.6}$$

denotes the radial displacement and

$$\delta_t \approx \begin{bmatrix} 2t_1 p_u p_v + t_2\big(r^2 + 2(p_u)^2\big) \\ t_1\big(r^2 + 2(p_v)^2\big) + 2t_2 p_u p_v \end{bmatrix} \tag{5.7}$$

denotes the tangential displacement, where $k_1, k_2, k_3 \in \mathbb{R}$ denote the radial distortion coefficients, $t_1, t_2 \in \mathbb{R}$ denote the tangential distortion coefficients and

$$r = \big\| p - (u_0, v_0) \big\|. \tag{5.8}$$

Note that these distortion coefficients may be used to undistort a distorted image captured by the robot's camera.

5.3 Summary

To summarise this chapter:

- In robotics, perception involves the use of a robot's sensors to take measurements of its environment.
- To achieve the objective of this dissertation, the use of wheel encoders for odometry and a front-facing onboard camera for landmark observations will suffice, as the information they provide is sufficient for reliable localisation. For this dissertation, the discrete pixel central projection model is used as a model of the robot's camera.

Chapter 6

Localisation

In robotics, localisation involves the use of perception information to estimate the configuration of a robot relative to its environment. For this dissertation, the perception information includes wheel encoder and camera information.

Let $\mathbb{N}$ denote the set of natural numbers, using the convention that $\mathbb{N}$ does not include 0, and $\mathbb{R}^{n\times m}$ denote the set of $n \times m$ matrices with real entries.

Let

$$\hat{q}_i = \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{\theta}_i \end{bmatrix} \tag{6.1}$$

and $\hat{P}_i \in \mathbb{R}^{3\times 3}$ denote the estimated configuration and configuration covariance of the robot at time step i, respectively, where $i \in \mathbb{N}$ and, for this dissertation,

$$\hat{P}_1 = \begin{bmatrix} \big((\sigma_x)_1\big)^2 & 0 & 0 \\ 0 & \big((\sigma_y)_1\big)^2 & 0 \\ 0 & 0 & \big((\sigma_\theta)_1\big)^2 \end{bmatrix}, \tag{6.2}$$

where $(\sigma_x)_1, (\sigma_y)_1, (\sigma_\theta)_1 \in \mathbb{R}_{\geq 0}$ denote the configuration noise standard deviations associated with the x, y and θ directions of $\{s\}$ at time step 1, respectively.

Let

$$(U_{xy})_i = \sqrt{\det\left(\big(\hat{P}_i\big)_{xy}\right)} \tag{6.3}$$

denote a scalar representation of the uncertainty about the position of the robot at time step i, where $\big(\hat{P}_i\big)_{xy} \in \mathbb{R}^{2\times 2}$ denotes the estimated position covariance of the robot at time step i, which is the top left 2×2 submatrix of $\hat{P}_i$. Note that error ellipses may also be used to represent the uncertainty about the position of the robot. Therefore, let

$$(p_\mu)_i = \sqrt{s_c\big(\hat{P}_i\big)_{xy}} \begin{bmatrix} \cos(\mu) \\ \sin(\mu) \end{bmatrix} + \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \end{bmatrix} \tag{6.4}$$

denote the point at $\mu \in \mathbb{S}^1$ on the error ellipse associated with the confidence value $c \in [0, 1]$ at time step i, where $s_c \in \mathbb{R}_{\geq 0}$ denotes the inverse cumulative distribution function of the χ^2-distribution with 2 degrees of freedom evaluated at c.

6.1 Odometry

The information in this section is derived from [36].

In robotics, odometry is the process of estimating the configuration of a robot using information about its motion. For a car-like robot, this information can be inferred from the motion of its wheels.

Let

$$(\Delta x_l)_i = R(\Delta\varphi_l)_i, \tag{6.5}$$
$$(\Delta x_r)_i = R(\Delta\varphi_r)_i \tag{6.6}$$

denote the changes along the x direction of $\{b\}$ for the robot's left and right rear wheels between time steps i and $i+1$, respectively, where $R \in \mathbb{R}_{>0}$ denotes the radius of the robot's rear wheels and $(\Delta\varphi_l)_i, (\Delta\varphi_r)_i \in \mathbb{S}^1$ denote the changes in the rolling angles of the robot's left and right rear wheels between time steps i and $i+1$, respectively. Therefore, with reference to (4.38) and (4.39), let

$$(\Delta x)_i = \frac{(\Delta x_l)_i + (\Delta x_r)_i}{2}, \tag{6.7}$$
$$(\Delta\theta)_i = \frac{(\Delta x_r)_i - (\Delta x_l)_i}{w} \tag{6.8}$$

denote the changes along the x and θ directions of $\{b\}$ between time steps i and $i+1$, respectively, where $w \in \mathbb{R}_{>0}$ denotes the track of the robot. Let

$$\upsilon_i = \begin{bmatrix} (\upsilon_x)_i \\ (\upsilon_\theta)_i \end{bmatrix} \sim \mathcal{N}\Big(0, \hat{V}\Big) \tag{6.9}$$

denote the odometry noise between time steps i and $i+1$, due to wheel slippage and rolling angle discretisation, for example, which, for simplicity, is modelled as a zero-mean multivariate Gaussian process, where $(\upsilon_x)_i, (\upsilon_\theta)_i \in \mathbb{R}$ denote the odometry noises associated with the x and θ directions of $\{b\}$ between time steps i and $i+1$, respectively, and $\hat{V} \in \mathbb{R}^{2\times 2}$ denotes the estimated odometry noise covariance, where, for this dissertation,

$$\hat{V} = \begin{bmatrix} (\varsigma_x)^2 & 0 \\ 0 & (\varsigma_\theta)^2 \end{bmatrix}, \tag{6.10}$$

where $\varsigma_x, \varsigma_\theta \in \mathbb{R}_{\geq 0}$ denote the odometry noise standard deviations associated with the x and θ directions of $\{b\}$, respectively. Therefore, let

$$\Delta\hat{q}_i = \begin{bmatrix} \big((\Delta x)_i + (\upsilon_x)_i\big)\cos\big(\hat{\theta}_i\big) \\ \big((\Delta x)_i + (\upsilon_x)_i\big)\sin\big(\hat{\theta}_i\big) \\ (\Delta\theta)_i + (\upsilon_\theta)_i \end{bmatrix} \tag{6.11}$$

denote the change in the estimated configuration of the robot between time steps i and $i+1$.

Let

$$\hat{q}^+_{i+1} = \hat{q}_i + \Delta\hat{q}_i \tag{6.12}$$
$$= \big(\hat{x}^+_{i+1}, \hat{y}^+_{i+1}, \hat{\theta}^+_{i+1}\big), \tag{6.13}$$
$$\hat{P}^+_{i+1} = (F_q)_i \hat{P}_i \big((F_q)_i\big)^T + (F_\upsilon)_i \hat{V} \big((F_\upsilon)_i\big)^T \tag{6.14}$$

denote the predicted estimates of the configuration and configuration covariance of the robot at time step

$i+1$, respectively, where

$$(F_q)_i = \left.\frac{\partial \hat{q}_{i+1}^+}{\partial \hat{q}_i}\right|_{\upsilon_i=0} \tag{6.15}$$

$$= \begin{bmatrix} 1 & 0 & -(\Delta x)_i \sin\left(\hat{\theta}_i\right) \\ 0 & 1 & (\Delta x)_i \cos\left(\hat{\theta}_i\right) \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.16}$$

$$(F_\upsilon)_i = \left.\frac{\partial \hat{q}_{i+1}^+}{\partial \upsilon_i}\right|_{\upsilon_i=0} \tag{6.17}$$

$$= \begin{bmatrix} \cos\left(\hat{\theta}_i\right) & 0 \\ \sin\left(\hat{\theta}_i\right) & 0 \\ 0 & 1 \end{bmatrix}. \tag{6.18}$$

Note that a Gaussian distribution of angles on $\mathbb{S}^1$ is impossible. Therefore, $(\upsilon_\theta)_i$ is incorrectly modelled. As stated in [36], a more appropriate, Gaussian-like distribution of angles on $\mathbb{S}^1$ is the von Mises distribution. However, the smaller σ_θ is, the smaller the error that results from the incorrect modelling of $(\upsilon_\theta)_i$.

Unfortunately, odometric estimation errors tend to accumulate over time. Therefore, the probability that odometric estimation errors will prevent the robot from achieving its objective increases as the time it takes the robot to achieve its objective increases. Therefore, it may be necessary to supplement odometric information with sensor information that does not accumulate errors over time, such as landmark observation information.

6.2 Landmark Observations

The information in this section is derived from [36], [43] and [42].

Let

$$z_{i+1}^\sharp = \begin{bmatrix} \rho_{i+1}^\sharp \\ \beta_{i+1}^\sharp \end{bmatrix} \tag{6.19}$$

denote a landmark observation at time step $i+1$, where $\rho_{i+1}^\sharp \in \mathbb{R}_{>0}$ and $\beta_{i+1}^\sharp \in \mathbb{S}^1$ denote the observed range and bearing of the landmark relative to $\{b\}$ at time step $i+1$, respectively.

For this dissertation, AprilTags are used to estimate the range and bearing of landmarks in the robot's environment. As explained in [42], AprilTags are visual fiducials that can be identified in an image. Moreover, with a small information payload of around 12 bits, AprilTags can be identified in low-resolution images, images with uneven lighting and images where the AprilTag has a skewed orientation relative to $\{c\}$. If the edge lengths and centroid location of an AprilTag are known, the estimated range and bearing of that AprilTag may be determined. Therefore, by attaching AprilTags to landmarks in the robot's environment, the estimated range and bearing of those landmarks can be determined. Suppose multiple AprilTags are identified in an image. The information from the one with the lowest object-space collinearity error, generated by the orthogonal iteration algorithm presented in [43], is used. Figure 6.1 shows an AprilTag from the 36h11 tag family.

While this type of sensor information does not accumulate over time, it often has greater uncertainty and a lower update frequency than odometric information. Therefore, the solution is to fuse these different types of sensor information using an estimation framework, such as the extended Kalman filter (EKF), as this will extract the best qualities of each type of sensor information if the various noises involved are accurately modelled.

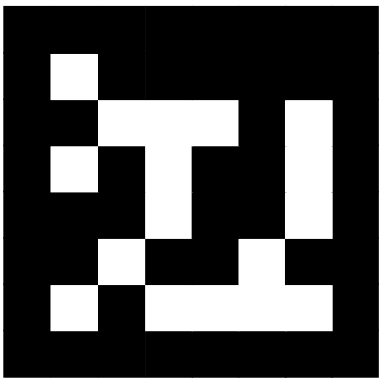


Figure 6.1: An AprilTag from the 36h11 tag family.

6.3 The Extended Kalman Filter

The information in this section is derived from [36].

The extended Kalman filter (EKF) consists of a prediction and update phase. For this dissertation, the prediction phase uses odometric information to predict the estimated new configuration and configuration covariance of the robot, and the update phase uses landmark observation information to correct these predictions, where, for a given landmark observation, the innovation, which is the difference between the landmark observation and the predicted landmark observation, is mapped into a correction for the estimated new configuration of the robot by the Kalman gain, which is also used in the update of the new configuration covariance of the robot.

For an observed landmark, let

$$\hat{\rho}_{i+1}^{+} = \sqrt{\left(x_l - \hat{x}_{i+1}^{+}\right)^2 + \left(y_l - \hat{y}_{i+1}^{+}\right)^2}, \tag{6.20}$$

$$\hat{\beta}_{i+1}^{+} = \arctan\left(\frac{y_l - \hat{y}_{i+1}^{+}}{x_l - \hat{x}_{i+1}^{+}}\right) \tag{6.21}$$

denote the predicted estimates of the range and bearing of the landmark relative to $\{b\}$ at time step $i+1$, respectively, where $(x_l, y_l) \in \mathbb{R}^2$ denotes the position of the landmark relative to $\{s\}$. Let

$$\psi_{i+1} = \begin{bmatrix} (\psi_\rho)_{i+1} \\ (\psi_\beta)_{i+1} \end{bmatrix} \sim \mathcal{N}\left(0, \hat{W}\right) \tag{6.22}$$

denote the landmark detector noise at time step $i+1$, due to geometric distortions, for example, which, for simplicity, is modelled as a zero-mean multivariate Gaussian process, where $(\psi_\rho)_{i+1}, (\psi_\beta)_{i+1} \in \mathbb{R}$ denote the landmark detector noises associated with ρ and β at time step $i+1$, respectively, and $\hat{W} \in \mathbb{R}^{2\times 2}$ denotes the estimated landmark detector noise covariance, where, for this dissertation,

$$\hat{W} = \begin{bmatrix} (\varsigma_\rho)^2 & 0 \\ 0 & (\varsigma_\beta)^2 \end{bmatrix}, \tag{6.23}$$

where $\varsigma_\rho, \varsigma_\beta \in \mathbb{R}_{\geq 0}$ denote the landmark detector noise standard deviations associated with ρ and β, respectively. Therefore, let

$$\hat{z}_{i+1}^{+} = \begin{bmatrix} \hat{\rho}_{i+1}^{+} + (\psi_\rho)_{i+1} \\ \hat{\beta}_{i+1}^{+} + (\psi_\beta)_{i+1} \end{bmatrix} \tag{6.24}$$

denote the predicted estimate of the landmark observation at time step $i+1$.

Let

$$\hat{q}_{i+1} = \hat{q}_{i+1}^{+} + K_{i+1}\nu_{i+1}, \tag{6.25}$$

$$\hat{P}_{i+1} = \big(I - K_{i+1}(H_q)_{i+1}\big)\hat{P}_{i+1}^{+} \tag{6.26}$$

denote the estimated configuration and configuration covariance of the robot at time step $i+1$, respectively, where

$$\nu_{i+1} = z_{i+1}^{\sharp} - \hat{z}_{i+1}^{+}, \tag{6.27}$$

$$K_{i+1} = \hat{P}_{i+1}^{+}\big((H_q)_{i+1}\big)^T\Big(\big(H_q)_{i+1}\hat{P}_{i+1}^{+}\big((H_q)_{i+1}\big)^T + \hat{W}\Big)^{-1} \tag{6.28}$$

denote the innovation and Kalman gain at time step $i+1$, respectively,

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.29}$$

$$(H_q)_{i+1} = \left.\frac{\partial \hat{z}_{i+1}^{+}}{\partial \hat{q}_{i+1}^{+}}\right|_{\psi_{i+1}=0} \tag{6.30}$$

$$= \begin{bmatrix} -\frac{x_l - \hat{x}_{i+1}^{+}}{\hat{\rho}_{i+1}^{+}} & -\frac{y_l - \hat{y}_{i+1}^{+}}{\hat{\rho}_{i+1}^{+}} & 0 \\ \frac{y_l - \hat{y}_{i+1}^{+}}{\big(\hat{\rho}_{i+1}^{+}\big)^2} & -\frac{x_l - \hat{x}_{i+1}^{+}}{\big(\hat{\rho}_{i+1}^{+}\big)^2} & -1 \end{bmatrix}. \tag{6.31}$$

Note that the estimated configuration covariance of the robot must be positive-definite. However, due to accumulated numerical errors, it may become antisymmetric if updated too many times. Therefore, if necessary, let

$$\hat{P}_{i+1} = \big(I - K_{i+1}(H_q)_{i+1}\big)\hat{P}_{i+1}^{+}\big(I - K_{i+1}(H_q)_{i+1}\big)^T + K_{i+1}\hat{V}(K_{i+1})^T, \tag{6.32}$$

which will maintain a positive-definite structure but is more computationally expensive.

6.4 Summary

To summarise this chapter:

- In robotics, localisation involves the use of perception information to estimate the configuration of a robot relative to its environment. For this dissertation, the perception information includes wheel encoder and camera information.
- In robotics, odometry is the process of estimating the configuration of a robot using information about its motion. For a car-like robot, this information can be inferred from the motion of its wheels.
- Unfortunately, odometric estimation errors tend to accumulate over time. Therefore, it may be necessary to supplement odometric information with sensor information that does not accumulate errors over time, such as landmark observation information. For this dissertation, AprilTags are used to estimate the range and bearing of landmarks in the robot's environment.
- While landmark observation information does not accumulate over time, it often has greater uncertainty and a lower update frequency than odometric information. Therefore, the EKF, which consists of a prediction and update phase, is used to fuse these different types of sensor information. For this dissertation, the prediction phase uses odometric information to predict the estimated new configuration and configuration covariance of the robot, and the update phase uses landmark observation information to correct these predictions.

Chapter 7

Software

The software presented in this dissertation includes planning, control and localisation software written in C++ and initialisation and animation software written in MATLAB, where the planning, control and localisation software is based on the contents of Chapters 3, 4 and 6, respectively. The software is available at [45], and a guide for using the software is available at [46]. Note that third-party perception software based on the contents of Chapter 5 was used for testing.

7.1 The Robot Operating System

The information in this section is derived from [47] and [48].

The software is based on the Robot Operating System (ROS), specifically ROS Noetic Ninjemys, which, as explained at [48], is an open-source, meta-operating system for robotic systems whose primary goal is to support code reuse in robotics research and development. As explained at [47], the Computation Graph is the peer-to-peer network of ROS processes, known as nodes, that process data together. Nodes communicate with each other by passing data structures known as messages, which are comprised of typed fields. The name used to identify the content of a message is known as the topic. The structure of the software presented in this dissertation can be understood using the Computation Graph.

7.2 The Computation Graph

Figure 7.1 shows the section of the Computation Graph relevant to this dissertation.

Note that "matlab global node", "planning node", "control node", "odometry node", "apriltag detector node" and "extended kalman filter node", whose code is available at [49], [50], [51], [52], [53] and [54], respectively, were developed for this dissertation. The remaining nodes were developed by Duckieworks Ltd., where the code for "camera node", "front centre time-of-flight sensor node" and "wheels node" is available at [55], [56] and [57], respectively, and the code for "left wheel encoder node" and "right wheel encoder node" is available at [58].

Using information from a parameter file, which is available at [59], "matlab global node" will initialise "planning node", "control node", "odometry node", "apriltag detector node" and "extended kalman filter node". If at least one controlled trajectory has been found, "planning node" will send the prevailing best controlled trajectory to "control node", which in turn will send information about the desired configuration of the robot to "planning node" in the form of an index location along this trajectory. Using this information, "planning node" will alert "control node" as to when planning is complete. Using the estimated configuration of the robot sent by "extended kalman filter node", "control node" will send wheel commands to "wheels node", which will convert wheel commands into PWM signals and then send those signals to the Duckiebot's motors. "control node" will also send wheel commands to stop the Duckiebot if the range sent by "front centre time-of-flight sensor node" is less than a minimum allowable range. Note that the relevant code in [60], which

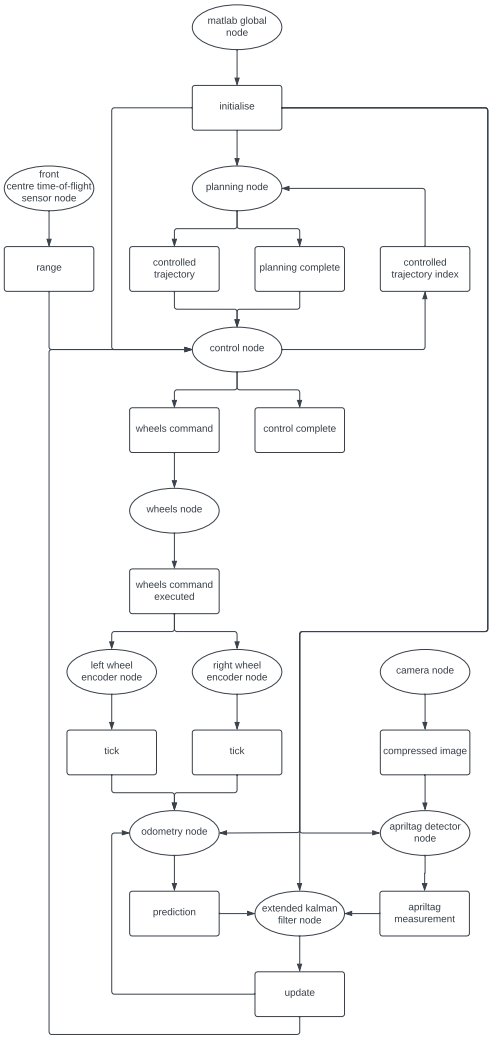


Figure 7.1: The section of the Computation Graph relevant to this dissertation, where ovals represent nodes, rounded rectangles represent topics and arrows represent the flow of information.

was also developed by Duckieworks Ltd., has been incorporated into "control node". Using the compressed image information sent by "camera node", "apriltag detector node" will send AprilTag measurements to "extended kalman filter node". Using the wheel encoder information sent by "left wheel encoder node" and "right wheel encoder node", which use the information sent by "wheels node", "odometry node" will send the predicted estimates of the configuration and configuration covariance of the robot to "extended kalman filter node", which in turn will send the estimated configuration and configuration covariance of the robot to "odometry node".

7.3 Animation

A MATLAB program, whose code is available at [61], was developed to visualise the tests presented in Chapter 8. This program can create a GIF of a test, once it has concluded, using information from the parameters file and the test's bag file, if one was created, where, as explained at [47], bags are a format for saving and playing back ROS message data.

7.4 Summary

To summarise this chapter:

- The software presented in this dissertation includes planning, control and localisation software written in C++ and initialisation and animation software written in MATLAB, where third-party perception software was used for testing.
- The software is based on ROS, which is an open-source, meta-operating system for robotic systems.
- The animation software can create a GIF of a test, once it has concluded, using information from a parameters file and the test's bag file, if one was created.

Chapter 8

Testing

To demonstrate the efficacy of the software presented in this dissertation, three tests were conducted. Each test tasked the software with finding a solution to the motion planning problem for a car-like robot in a static environment and then controlling the Duckiebot such that it closely tracks the prevailing best solution while the software continually searches for better solutions. In the first, second and third tests, the environment contained no obstacles, one obstacle and three obstacles, respectively. Note that every node other than "matlab global node" was run on the Duckiebot's onboard computer.

8.1 Setup

The following Subsections present the setup used during testing.

8.1.1 The Robot

As shown in Figure 8.1, the length of the robot's wheelbase was set to 0.15 m, its length was set to the length of its wheelbase plus the diameter of the Duckiebot's motorised wheels, which is equal to 0.066 m, and its width was set to the length of the Duckiebot's track, which is equal to 0.106 m, plus the width of the Duckiebot's motorised wheels, which is equal to 0.025 m.

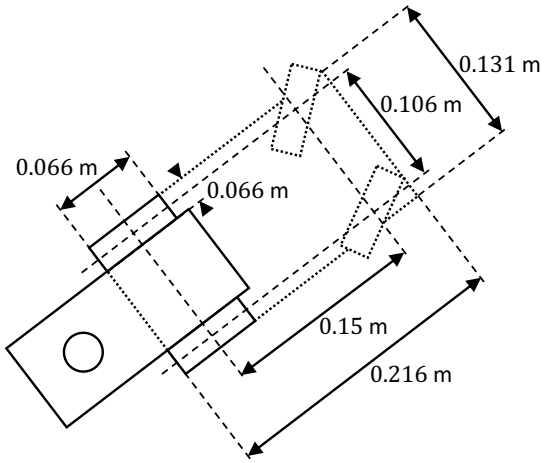


Figure 8.1: The setup for the robot.

8.1.2 Obstacles

The obstacles used for the three tests were wooden stands with AprilTags from the 36h11 tag family attached, similar to Figure 8.2. These AprilTags were given the name "Obstacle tag". Another stand with an AprilTag attached was used for localisation near the goal region. This AprilTag was given the name "Goal tag". The obstacles had a maximum horizontal length and width of 0.08 m and 0.054 m, respectively, and the AprilTags had an edge length of 0.065 m.

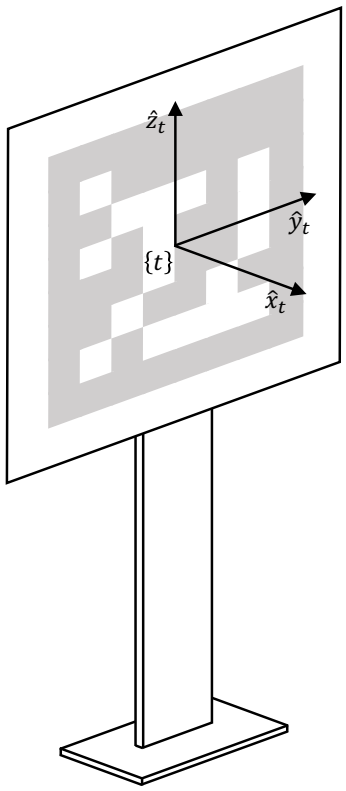
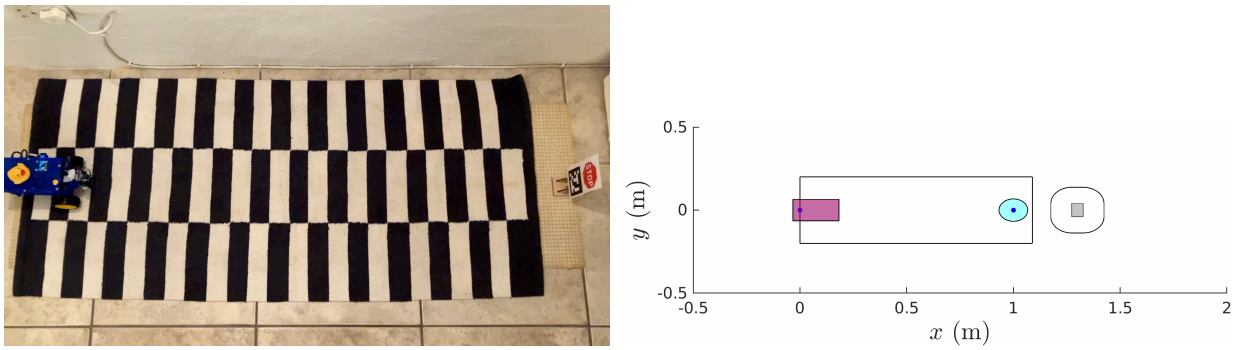


Figure 8.2: An AprilTag from the 36h11 tag family attached to a stand with reference frame $\{t\}$.

8.1.3 Map

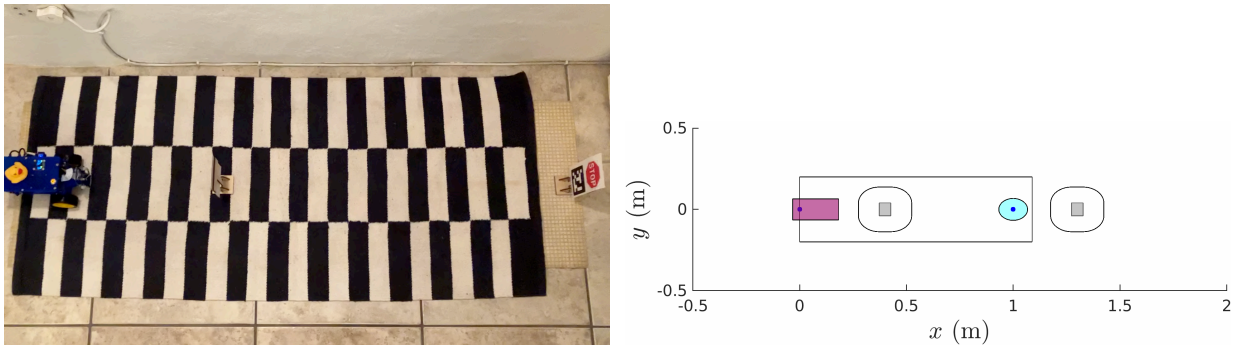
The initial configuration of the robot was set to $(0\text{ m}, 0\text{ m}, 0\text{ rad})$. The ranges for the search region along the x and y directions of $\{s\}$ were set to $[-0.2\text{ m}, 1.2\text{ m}]$ and $[-0.2\text{ m}, 0.2\text{ m}]$, respectively. A circular goal region was chosen for simplicity. The position of the centre of the goal region was set to $(1\text{ m}, 0\text{ m})$. The MATLAB program sets the radius of the goal region to be the minimum value between the maximum allowable radius and the distance between the centre of the goal region and the nearest point on an obstacle or the boundary of the search region. The maximum allowable radius of the goal region was set to 0.1 m. For each test, the stand with "Goal tag" attached was placed with a configuration of $(1.3\text{ m}, 0\text{ m}, \pi\text{ rad})$. For the second test, an obstacle was placed with a configuration of $(0.4\text{ m}, 0\text{ m}, \pi\text{ rad})$. For the third test, another two obstacles were placed with configurations $(0.8\text{ m}, 0.2\text{ m}, \pi\text{ rad})$ and $(0.8\text{ m}, -0.2\text{ m}, \pi\text{ rad})$, respectively.

Figures 8.3–8.5 show the setups for the first, second and third tests, respectively. Subfigures 8.3a–8.5a show the real-world setups and Subfigures 8.3b–8.5b shows the simulated setups in Figures 8.3b–8.5b, where the robot at its desired and estimated configuration is shown as a blue and red rectangle, respectively, which are currently overlapping, the goal region is shown as a cyan disc, the search region is shown as a white rectangle, and the stands, to which the goal and obstacle tags are attached, are shown as grey rectangles within white rounded rectangles.



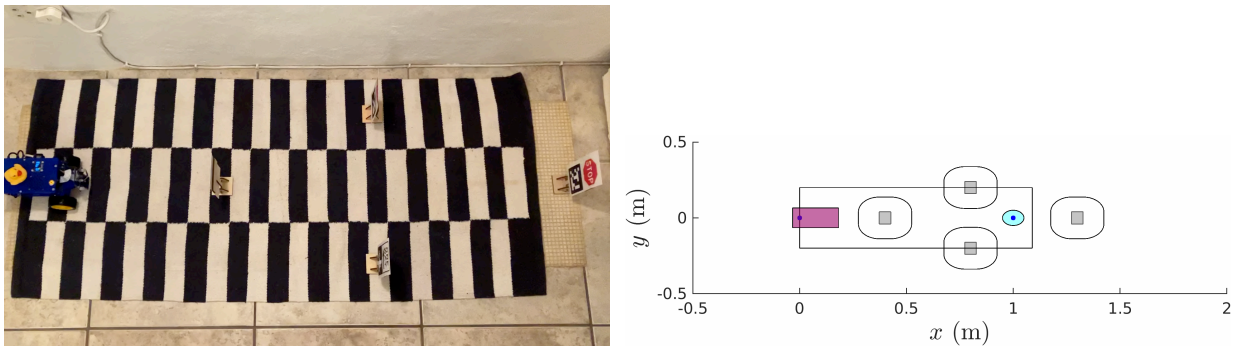
(a) (b)

Figure 8.3: The setup for the first test. (a) Real-world environment. (b) Simulated environment.



(a) (b)

Figure 8.4: The setup for the second test. (a) Real-world environment. (b) Simulated environment.



(a) (b)

Figure 8.5: The setup for the third test. (a) Real-world environment. (b) Simulated environment.

8.1.4 Planning

The following values refer to those presented in Chapter 3. For the `random_free` function, the centre of the goal region would be returned every 10 samples. For the `steer` function, $r_{\max}$ was set to 0.1 m. For the `collision_free` function, the number of samples taken for any given path was set to 8. Q_{obs} was enlarged by 0.75 times the width of the robot. The time taken to travel along the path from q_{nearest} to q_{new} was divided into 5 equal time steps. τ_{com} was set to equal the next time step of the uncommitted trajectory.

8.1.5 Control

The following values refer to those presented in Chapter 4. $(v_d)_{\max}$ and $v_{\max}$ were set to 0.05 m.s^{-1} and 0.5 m.s^{-1}, respectively, as these values allowed "control node" to quickly reduce the configuration error of the robot while not preventing friction from quickly overcoming the momentum of the robot in the absence of controls. $(\phi_d)_{\max}$ and $\phi_{\max}$ were set to 25° and 75°, respectively, as these values allowed "control node" to quickly reduce the configuration error of the robot while not preventing solutions to the motion planning problem from being found in obstacle-rich environments. k_1, k_2, k_3 were set to 150, 500 and 500, respectively, as these values allowed "control node" to provide satisfactory corrective action while limiting overshoot, which tended to increase as these values were further increased. k_l and k_r were set to 1, as any unwanted difference between the output of the two motors when sent the same signal did not prevent the robot from closely tracking the desired trajectory. The minimum allowable range that may be sent by "front centre time-of-flight sensor node", below which "control node" will send wheel commands to stop the robot, was set to 0.1 m. The `motor_constant` parameter used by "control node" was set to the value of the `k` parameter in [60], which is equal 27.

8.1.6 Localisation

The following values refer to those presented in Chapter 6 and were chosen through experimentation, iterating on the values used in [36]. $(\sigma_x)_1$, $(\sigma_y)_1$ and $(\sigma_\theta)_1$ were set to 0.005 m, 0.005 m and 0.001°, respectively, as the uncertainty about the initial configuration of the robot is very low. ς_x, ς_θ, ς_ρ and ς_β were set to 0.01 m, 0.5°, 0.1 m and 5°, respectively, as these values allowed "extended kalman filter node" to provide satisfactory corrective action. Keeping ς_x and ς_θ fixed, further increasing ς_ρ and ς_β would result in "extended kalman filter node" providing too little corrective action, limiting the benefit of using "extended kalman filter node", while further decreasing ς_ρ and ς_β would result in "extended kalman filter node" providing too much corrective action, possibly leading to failure.

8.2 Results

Figures 8.6, 8.8 and 8.10 show the results of the first, second and third tests, respectively, where the tree graph is shown in black, the path to the goal is shown in blue, the history of the estimated configuration of the robot is shown in red, and the error ellipse associated with a confidence value of 0.99 is shown as a green ellipse. Each subfigure is comprised of a visualisation of the test above an image received through the robot's camera. If an AprilTag was identified in an image, its name, estimated range in metres and estimated bearing in degrees were shown in that image. The GIFs from which Figures 8.6, 8.8 and 8.10 were derived are available at [62], [63] and [64], respectively. Figures 8.7, 8.9 and 8.11 show the results of the first, second and third tests, respectively, from a top-down perspective. The videos from which Figures 8.7, 8.9 and 8.11 were derived are available at [65], [66] and [67], respectively.

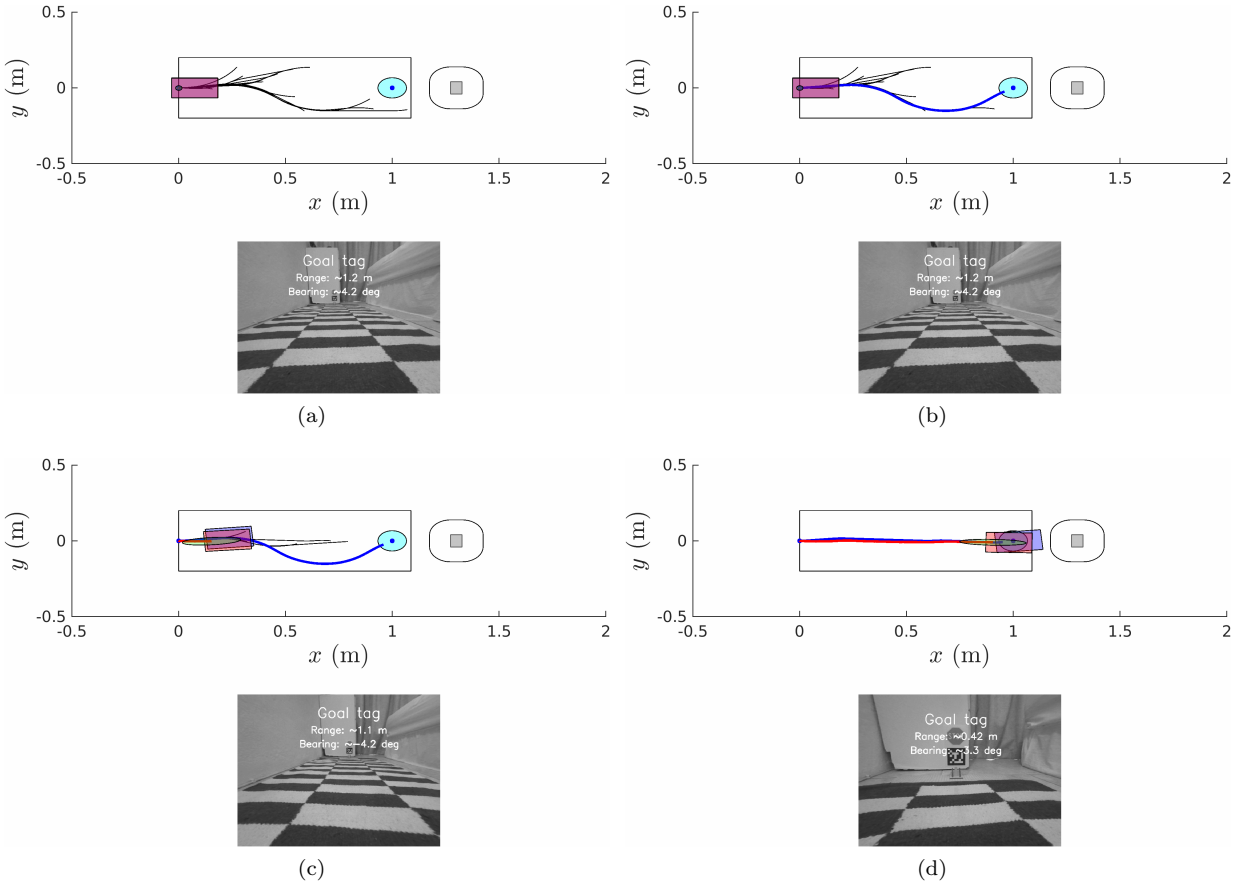
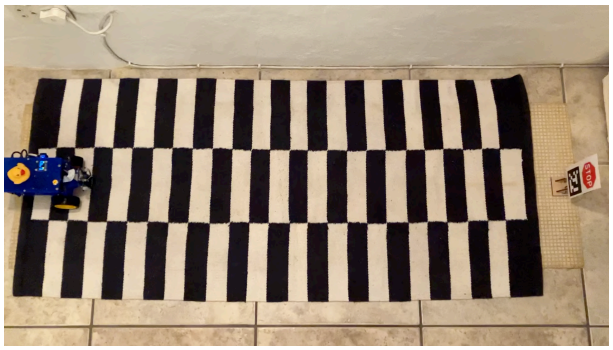
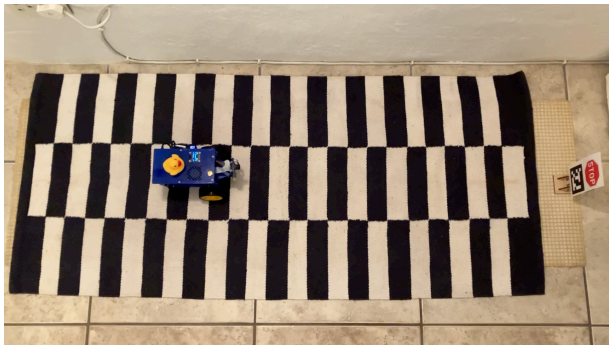


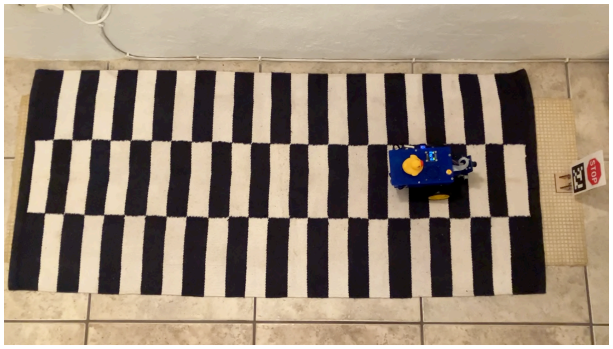
Figure 8.6: The result of the first test. (a) Searching for a solution. (b) A solution has been found. (c) Controlling the robot while searching for better solutions. (d) The robot has reached the goal.



(a)



(b)



(c)

Figure 8.7: The result of the first test from a top-down perspective. (a) Searching for and finding a solution. (b) Controlling the robot while searching for better solutions. (c) The robot has reached the goal.

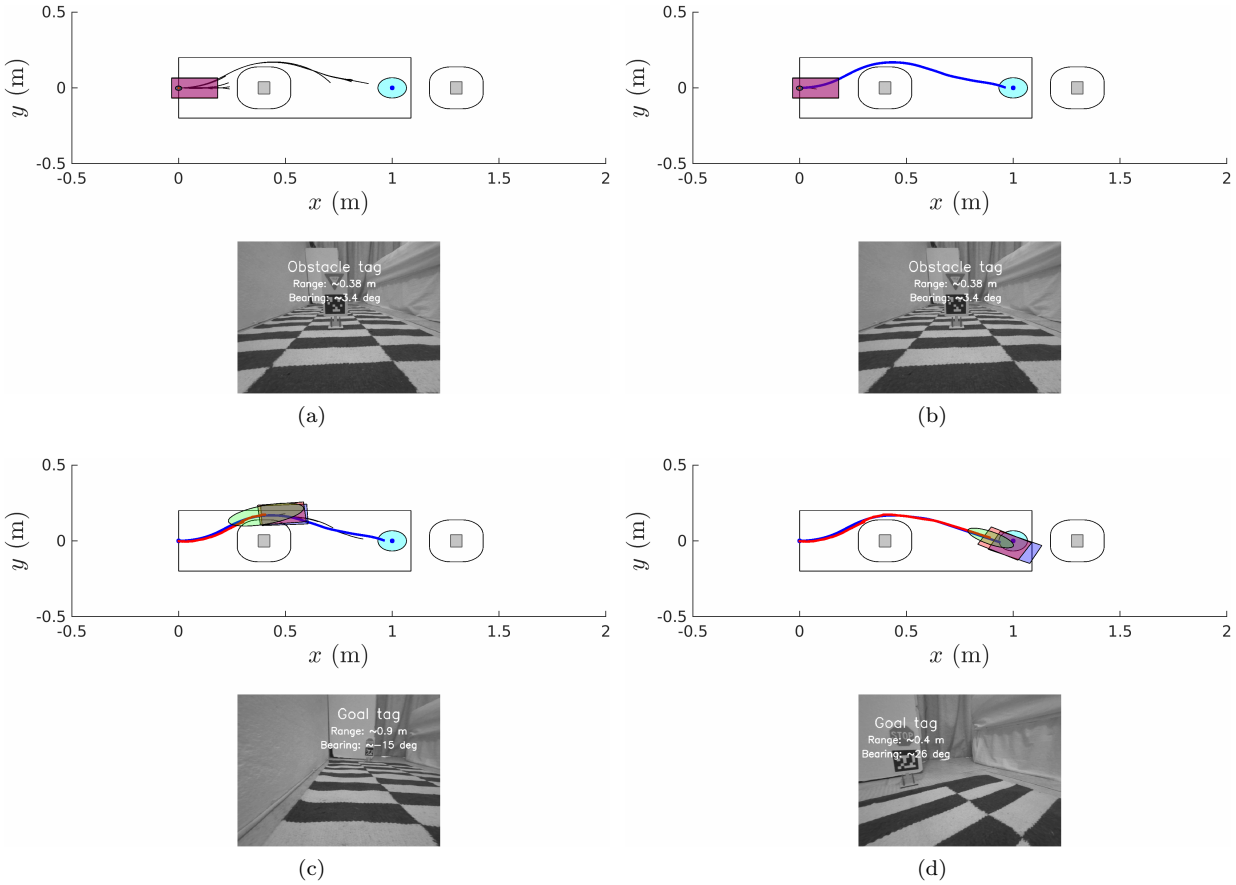
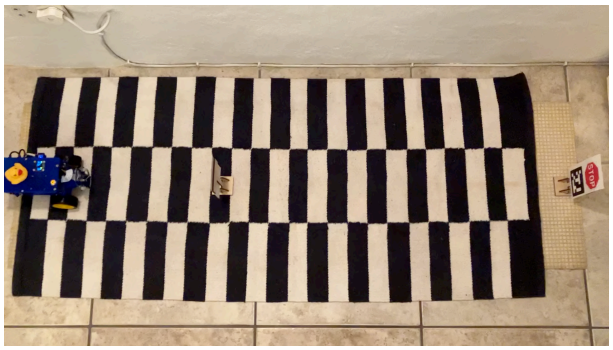
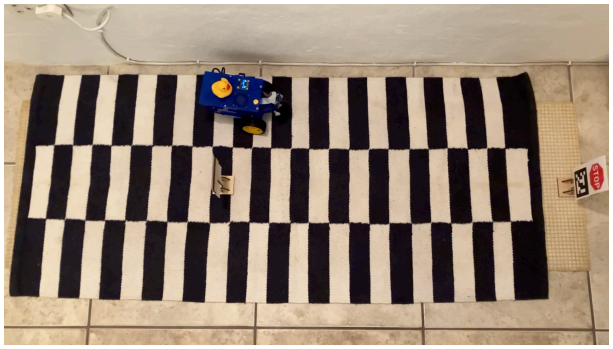


Figure 8.8: The result of the second test. (a) Searching for a solution. (b) A solution has been found. (c) Controlling the robot while searching for better solutions. (d) The robot has reached the goal.



(a)



(b)

(c)

Figure 8.9: The result of the second test from a top-down perspective. (a) Searching for and finding a solution. (b) Controlling the robot while searching for better solutions. (c) The robot has reached the goal.

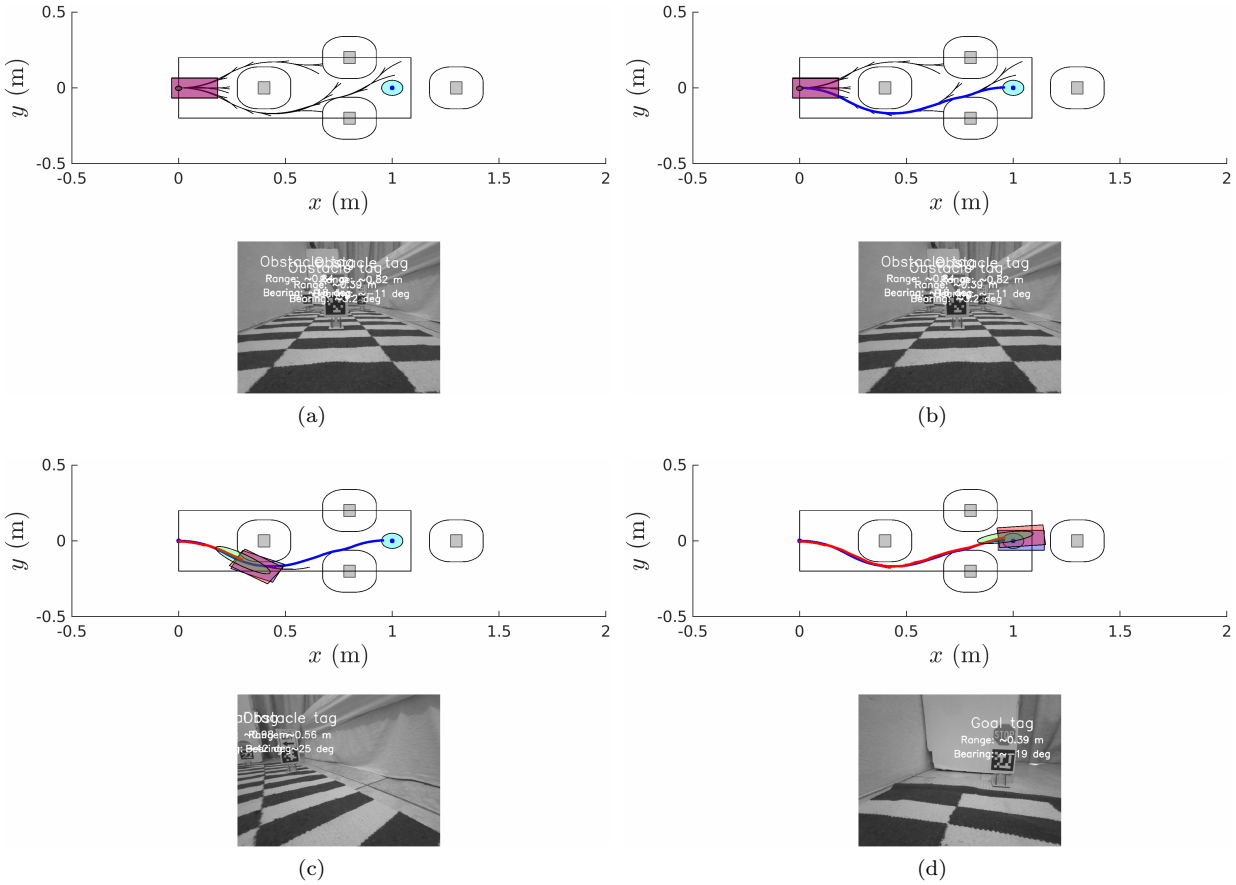
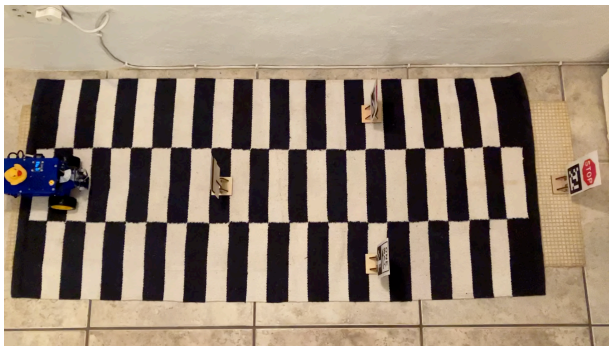
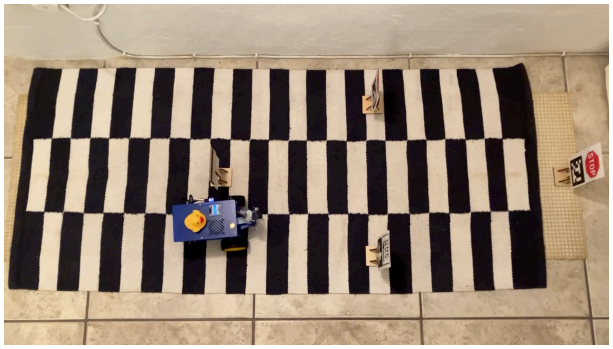


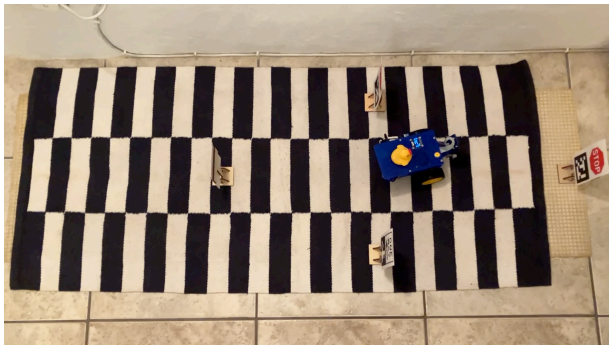
Figure 8.10: The result of the third test. (a) Searching for a solution. (b) A solution has been found. (c) Controlling the robot while searching for better solutions. (d) The robot has reached the goal.



(a)



(b)



(c)

Figure 8.11: The result of the third test from a top-down perspective. (a) Searching for and finding a solution. (b) Controlling the robot while searching for better solutions. (c) The robot has reached the goal.

8.3 Analysis

The results of the first, second and third tests show the software searching for and finding a solution to the motion planning problem for a car-like robot in a static environment containing no obstacles, one obstacle and three obstacles, respectively. They also show the software controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions. Therefore, the objective of this dissertation, as stated in Section 1.1, has been achieved.

Note that

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} \tag{8.1}$$

denotes the configuration error of the robot in the desired frame $\{d\}$, U_{xy} denotes a scalar representation of the uncertainty about the position of the robot, $(u_d)_v$ and $(u_d)_\phi$ denote the desired control variables associated with v and ϕ, respectively, and $(u_v)_b$ and $(u_\omega)_b$ denote the bounded control variables associated with v and ω, respectively. Figures 8.12, 8.13 and 8.14 show the x_e, y_e, θ_e, U_{xy}, $(u_d)_v$, $(u_d)_\phi$, $(u_v)_b$ and $(u_\omega)_b$ histories for the first, second and third tests, respectively.

As can be seen from Subfigures 8.12a, 8.13a and 8.14a, overshoot was common due to either the limitations of the kinematic model or the update frequency of "control node" being too low. However, it was small enough for the robot to be able to continue tracking the desired trajectory.

As can be seen from Subfigures 8.12d, 8.13d and 8.14d, U_{xy} increases until an AprilTag is identified, at which point it decreases, producing the peaks seen in the subfigures. This same behaviour can be seen in the GIFs of the tests, where the error ellipses increase in size until an AprilTag is identified, at which point they decrease. Note that, as seen in the GIFs of the tests, the software could not identify every visible AprilTag.

The benefit of using RRT*, as opposed to RRT, can be seen in Subfigures 8.12f, 8.13f and 8.14f, as $(u_d)_\phi$ did not change very often.

When comparing Subfigures 8.12e, 8.13e and 8.14e to Subfigures 8.12g, 8.13g and 8.14g, respectively, it can be seen that $(u_v)_b$ was on average much greater than $(u_d)_v$, implying that the vast majority of the corrective action was due to feedback control. As can be seen from Subfigures 8.12g, 8.13g and 8.14g, $(u_v)_b$ rarely reached $v_{\max}$, which was set to 0.5 m.s^{-1}, and was therefore rarely saturated.

8.4 Summary

To summarise this chapter:

- To demonstrate the efficacy of the software presented in this dissertation, three tests were conducted. Each test tasked the software with finding a solution to the motion planning problem for a car-like robot in a static environment and then controlling the Duckiebot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.
- In the first, second and third tests, the environment contained no obstacles, one obstacle and three obstacles, respectively.
- The results of the first, second and third tests show the software searching for and finding a solution to the motion planning problem for a car-like robot in a static environment containing no obstacles, one obstacle and three obstacles, respectively. The results also show the software controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.
- It can therefore be concluded that the objective of this dissertation has been achieved.

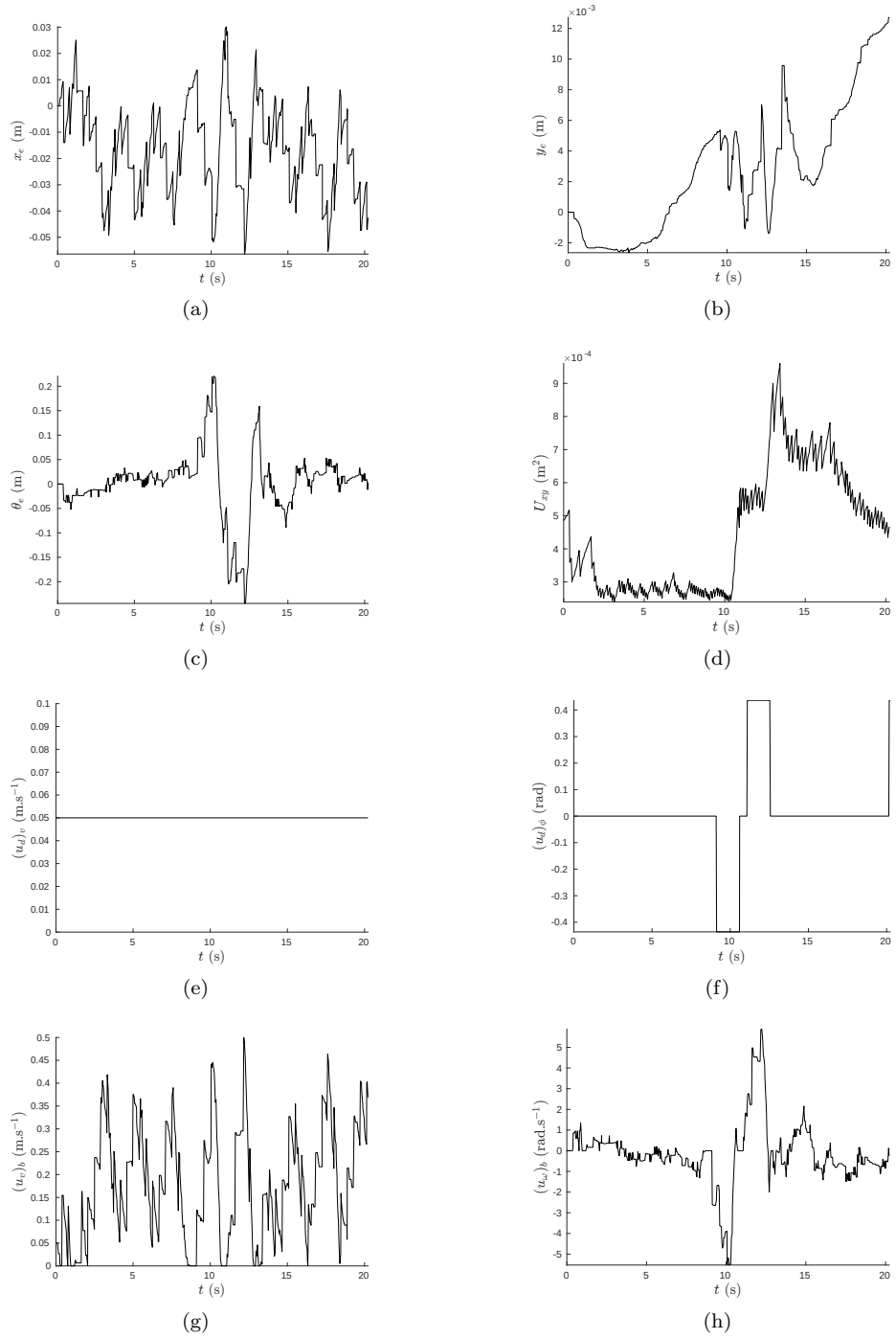


Figure 8.12: The information histories for the first test. (a) x_e history. (b) y_e history. (c) θ_e history. (d) U_{xy} history. (e) $(u_d)_v$ history. (f) $(u_d)_\phi$ history. (g) $(u_v)_b$ history. (h) $(u_\omega)_b$ history.

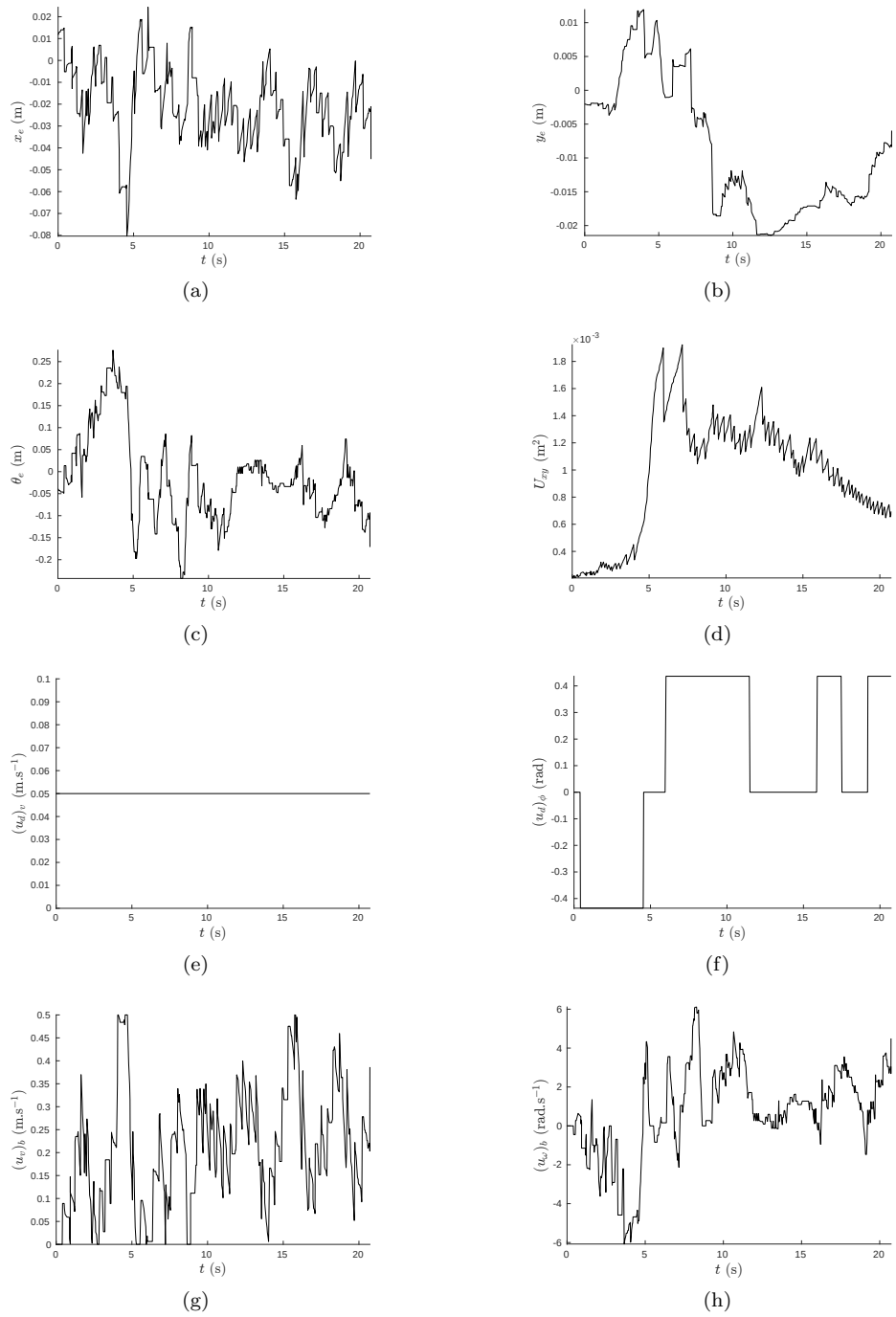


Figure 8.13: The information histories for the second test. (a) x_e history. (b) y_e history. (c) θ_e history. (d) U_{xy} history. (e) $(u_d)_v$ history. (f) $(u_d)_\phi$ history. (g) $(u_v)_b$ history. (h) $(u_\omega)_b$ history.

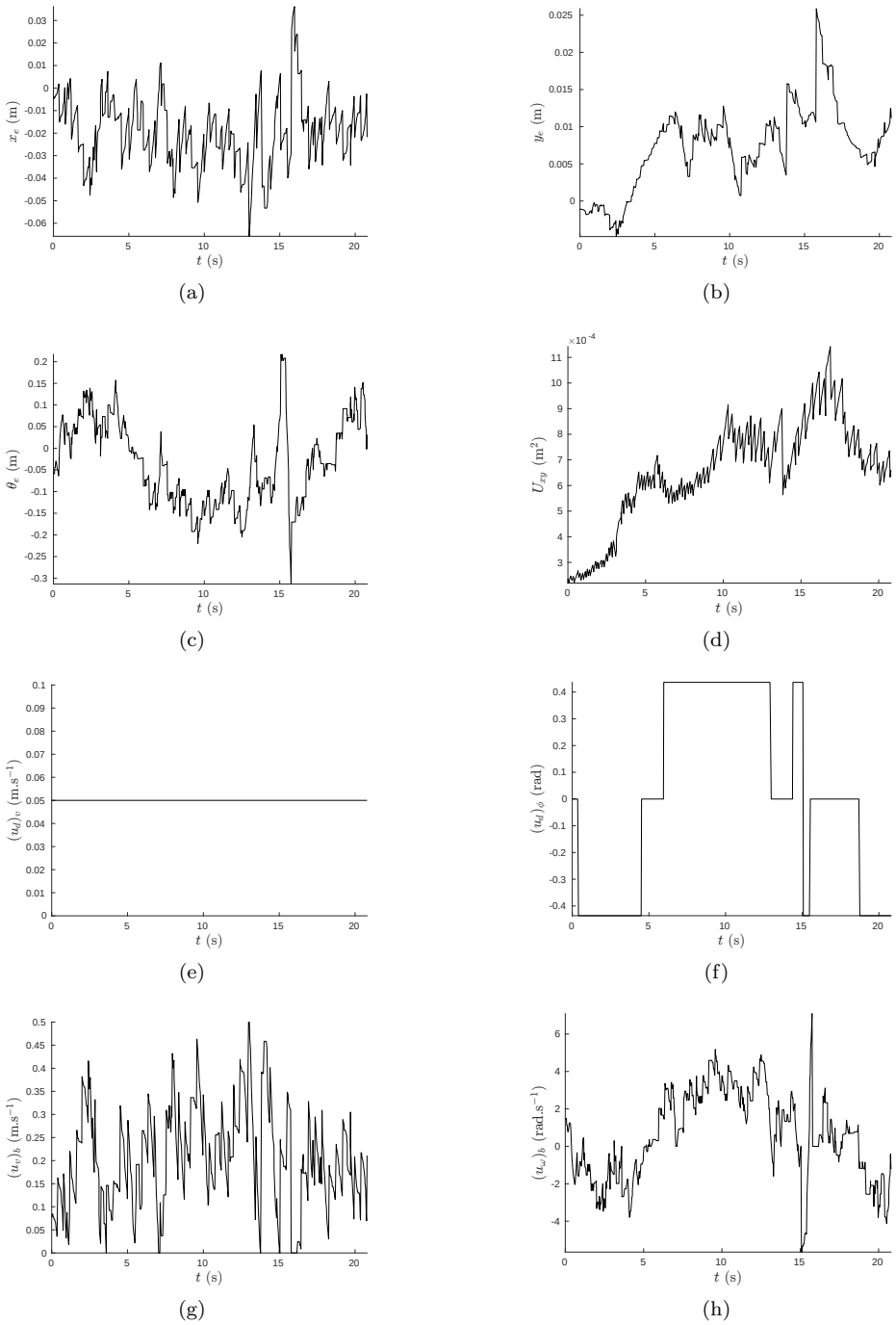


Figure 8.14: The information histories for the third test. (a) x_e history. (b) y_e history. (c) θ_e history. (d) U_{xy} history. (e) $(u_d)_v$ history. (f) $(u_d)_\phi$ history. (g) $(u_v)_b$ history. (h) $(u_\omega)_b$ history.

Chapter 9

Conclusion

The objective of this dissertation was to present software capable of finding a solution to the motion planning problem for a car-like robot in an obstacle-rich, static environment and controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.

To achieve the objective of this dissertation, the use of planning, control, perception and localisation software was necessary. The planning software made use of an online RRT* algorithm, the control software made use of a trajectory tracking control method that, through a Lyapunov-like analysis using Barbălat's Lemma, was shown to provide global asymptotic stability, the perception software made use of wheel encoder and camera information, and the localisation software made use of the EKF to fuse odometric and landmark observation information.

The software was based on ROS and tested using a Duckiebot (DB21-M). Because the Duckiebot does not possess a rear-facing camera, the mathematical model used for the car-like robot was the Dubins car, where a compact set of closed-form equations that describe the set of Dubins paths was presented in this dissertation. These equations were derived by modelling the Dubins car as an underactuated system on $SE(2)$ and then solving an associated set of inverse kinematics problems.

To demonstrate the efficacy of the software, three tests were conducted. In the first, second and third tests, the environment contained no obstacles, one obstacle and three obstacles, respectively, where the obstacles were wooden stands with AprilTags attached, which are used to help localise the Duckiebot. These tests showed the software searching for and finding a solution to the motion planning problem in these environments and then controlling the Duckiebot such that it closely tracks the prevailing best solution while the software continually searches for better solutions. Therefore, the objective of this dissertation has been achieved.

9.1 Research Contributions

This dissertation presented:

- Software capable of finding a solution to the motion planning problem for a car-like robot in an obstacle-rich, static environment and then controlling a real-world robot such that it closely tracks the prevailing best solution while the software continually searches for better solutions.
- A compact set of closed-form equations that describe the set of Dubins and Reeds-Shepp paths.

The insights gained from this dissertation may be used to develop motion planning software for industrial robots and autonomous vehicles.

9.2 Further Research

If the insights gained from this dissertation are to be used to develop motion planning software for industrial robots and autonomous vehicles, it may be useful to explore the following avenues of research:

- Planning in dynamic environments.
- Allowing the robot to reverse.
- Increasing $v_{\max}$ to a point where a dynamic model of the robot is required.
- Testing on different robotic platforms and in different environments to assess reliability.
- Using a more robust control method to improve reliability.
- Using simultaneous localisation and mapping.

Bibliography

[1] Duckietown: Store. https://get.duckietown.com/ [Accessed: 16 July 2023].

[2] Duckietown. https://www.duckietown.org/ [Accessed: 16 July 2023].

[3] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. Fan Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S. -Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H. -C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi. Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1497–1504, 2017.

[4] Duckietown: Duckiebot (DB21-M) datasheet. https://drive.google.com/file/d/1e_OQaWcVt0ix1i0kD1DZ_3miPz4djBUt/view [Accessed: 16 July 2023].

[5] Duckietown: Educational resources. https://www.duckietown.org/instructors/classes/educational-resources [Accessed: 16 July 2023].

[6] Duckietown: Research papers. https://www.duckietown.org/about/papers [Accessed: 16 July 2023].

[7] P. Almàsi, R. Moni, and B. Gyires-Tòth. Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[8] S. -R. Miguel, S. Morin, and L. Paull. Monocular Robot Navigation with Self-Supervised Pretrained Vision Transformers. In *2022 19th Conference on Robots and Vision (CRV)*, pages 197–204, 2022.

[9] R. Pérez-Dattari, C. Celemin, J. Ruiz-del-Solar, and J. Kober. Interactive Learning with Corrective Feedback for Policies based on Deep Neural Networks. In J. Xiao and T. Kröger, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics (ISER 2018)*, pages 353–363. Springer, Cham, 2020.

[10] K. Chaika, A. Filatov, A. Filatov, and K. Krinkin. Automatic Wheels and Camera Calibration for Monocular and Differential Mobile Robots. *Applied Sciences*, 11(13), 2021.

[11] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[12] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[13] Robot Operating System. https://www.ros.org/ [Accessed: 11 July 2023].

[14] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime Motion Planning using the RRT*. In *IEEE International Conference on Robotics and Automation*, pages 1478–1483, 2011.

[15] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[16] S. M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Department of Computer Science, Iowa State University, 1998.

[17] L. E. Kavraki, P. Švestka, J. -C. Latombe, and M. H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[18] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Florida, 1994.

[19] F. Bullo and A. D. Lewis. *Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems*. Springer Science+Business Media, New York, 2005.

[20] L. E. Kavraki and S. M. LaValle. Motion Planning. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics, First Edition*, pages 109–131. Springer-Verlag, 2008.

[21] J. Minguez, F. Lamiraux, and J. -P. Laumond. Motion Planning and Obstacle Avoidance. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics, First Edition*, pages 827–852. Springer-Verlag, 2008.

[22] J. -P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in Nonholonomic Motion Planning for Mobile Robots. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 1–54. Springer-Verlag, 1998.

[23] A. Bellaïche, F. Jean, and J. -J. Risler. Geometry of Nonholonomic System. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 55–92. Springer-Verlag, 1998.

[24] P. Souères and J. -D. Boissonnat. Optimal Trajectories for Nonholonomic Mobile Robots. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 93–170. Springer-Verlag, 1998.

[25] P. Švestka and M. H. Overmars. Probabilistic Path Planning. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 255–304. Springer-Verlag, 1998.

[26] S. Karaman and E. Frazzoli. Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods. In *49th IEEE Conference on Decision and Control (CDC)*, pages 7681–7687, 2010.

[27] J. h. Jeon, S. Karaman, and E. Frazzoli. Anytime Computation of Time-Optimal Off-Road Vehicle Maneuvers using the RRT*. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 3276–3282, 2011.

[28] P. Jiménez, F. Thomas, and C. Torras. Collision Detection Algorithms for Motion Planning. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 305–339. Springer-Verlag, 1998.

[29] L. E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[30] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. In *Pacific Journal of Mathematics*, volume 145, pages 367–393, 1990.

[31] H. Sussman and G. Tang. Shortest paths for the Reeds-Shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYCON-91-10, Department of Mathematics, Rutgers University, Piscataway, New Jersey, 1991.

[32] A. M. Shkel and V. Lumelsky. Classification of the Dubins set. *Robotics and Autonomous Systems*, 34(4):179–202, 2001.

[33] S. Martínez, J. Cortés, and F. Bullo. A catalog of inverse-kinematics planners for underactuated systems on matrix Lie groups. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 625–630, 2003.

[34] P. Morin and C. Samson. Motion Control of Wheeled Mobile Robots. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics, First Edition*, pages 799–826. Springer-Verlag, 2008.

[35] J. -J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, New Jersey, 1991.

[36] P. Corke. *Robotics, Vision and Control Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition*. Cham : Springer International Publishing : Imprint: Springe, second edition, 2017.

[37] A. De Luca, G. Oriolo, and C. Samson. Feedback Control of a Nonholonomic Car-Like Robot. In J. -P. Laumond, editor, *Robot Motion Planning and Control*, volume 229, pages 171–254. Springer-Verlag, 1998.

[38] R. E. Kálmán. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME, Journal of Basic Engineering*, 82(1):35–45, 1960.

[39] H. Durrant-Whyte and T. C. Henderson. Multisensor Data Fusion. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics, First Edition*, pages 585–610. Springer-Verlag, 2008.

[40] S. J. Julier and J. K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[41] S. Thrun and J. J. Leonard. Simultaneous Localization and Mapping. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics, First Edition*, pages 871–889. Springer-Verlag, 2008.

[42] E. Olsen. AprilTag: A robust and flexible visual fiducial system. In *IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[43] C. -P. Lu, G. D. Hager, and E. Mjolsness. Fast and Globally Convergent Pose Estimation from Video Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622, 2000.

[44] GitHub: Duckietown. `https://github.com/duckietown` [Accessed: 3 July 2023].

[45] GitHub: Motion planning software for a car-like robot. `https://github.com/MCLLIA002/motion_planner` [Accessed: 10 October 2023].

[46] GitHub: README.md. `https://github.com/MCLLIA002/motion_planner/blob/master/README.md` [Accessed: 10 October 2023].

[47] Robot Operating System: Concepts. `http://wiki.ros.org/ROS/Concepts` [Accessed: 11 July 2023].

[48] Robot Operating System: Introduction. `http://wiki.ros.org/ROS/Introduction` [Accessed: 11 July 2023].

[49] GitHub: initialise.m. `https://github.com/MCLLIA002/motion_planner/blob/master/initialise.m` [Accessed: 10 October 2023].

[50] GitHub: planning_node.cpp. `https://github.com/MCLLIA002/motion_planner/blob/master/packages/motion_planner/src/planning_node.cpp` [Accessed: 10 October 2023].

[51] GitHub: control_node.cpp. `https://github.com/MCLLIA002/motion_planner/blob/master/packages/motion_planner/src/control_node.cpp` [Accessed: 10 October 2023].

[52] GitHub: odometry_node.cpp. `https://github.com/MCLLIA002/motion_planner/blob/master/packages/motion_planner/src/odometry_node.cpp` [Accessed: 10 October 2023].

[53] GitHub: apriltag_detector_node.cpp. `https://github.com/MCLLIA002/motion_planner/blob/master/packages/motion_planner/src/apriltag_detector_node.cpp` [Accessed: 10 October 2023].

[54] GitHub: extended_kalman_filter_node.cpp. `https://github.com/MCLLIA002/motion_planner/blob/master/packages/motion_planner/src/extended_kalman_filter_node.cpp` [Accessed: 10 October 2023].

[55] GitHub: jetson_nano_camera_node.py. https://github.com/duckietown/dt-duckiebot-interface/blob/daffy/packages/camera_driver/src/jetson_nano_camera_node.py [Accessed: 3 July 2023].

[56] GitHub: tof_node.py. https://github.com/duckietown/dt-duckiebot-interface/blob/daffy/packages/tof_driver/src/tof_node.py [Accessed: 3 July 2023].

[57] GitHub: wheels_driver_node.py. https://github.com/duckietown/dt-duckiebot-interface/blob/daffy/packages/wheels_driver/src/wheels_driver_node.py [Accessed: 3 July 2023].

[58] GitHub: wheel_encoder_node.py. https://github.com/duckietown/dt-duckiebot-interface/blob/daffy/packages/wheel_encoder/src/wheel_encoder_node.py [Accessed: 3 July 2023].

[59] GitHub: parameters.m. https://github.com/MCLLIA002/motion_planner/blob/master/parameters.m [Accessed: 10 October 2023].

[60] GitHub: kinematics_node.py. https://github.com/duckietown/dt-car-interface/blob/daffy/packages/dagu_car/src/kinematics_node.py [Accessed: 3 July 2023].

[61] GitHub: animate.m. https://github.com/MCLLIA002/motion_planner/blob/master/animate.m [Accessed: 10 October 2023].

[62] GitHub: first_test.gif. https://github.com/MCLLIA002/motion_planner/blob/master/first_test.gif [Accessed: 10 October 2023].

[63] GitHub: second_test.gif. https://github.com/MCLLIA002/motion_planner/blob/master/second_test.gif [Accessed: 10 October 2023].

[64] GitHub: third_test.gif. https://github.com/MCLLIA002/motion_planner/blob/master/third_test.gif [Accessed: 10 October 2023].

[65] GitHub: first_test.MOV. https://github.com/MCLLIA002/motion_planner/blob/master/first_test.MOV [Accessed: 10 October 2023].

[66] GitHub: second_test.MOV. https://github.com/MCLLIA002/motion_planner/blob/master/second_test.MOV [Accessed: 10 October 2023].

[67] GitHub: third_test.MOV. https://github.com/MCLLIA002/motion_planner/blob/master/third_test.MOV [Accessed: 10 October 2023].

Appendix A

Reeds-Shepp Paths

The information in this appendix is derived from [11], [12] and [31].

Given configurations $q_1, q_2 \in Q_{\text{free}}$, where, in coordinates (x, y, θ),

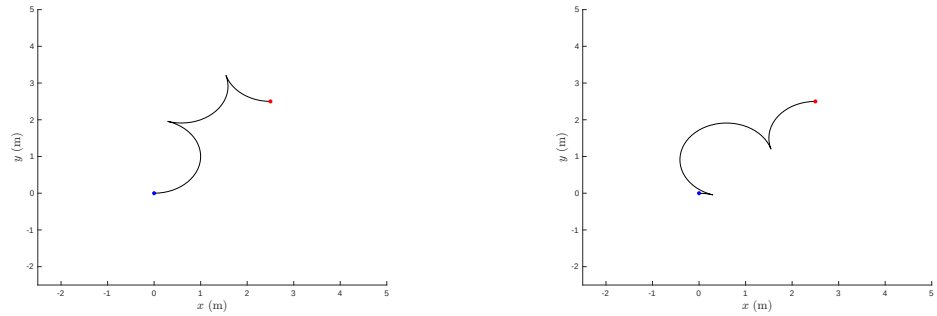
$$q_1 = (x_1, y_1, \theta_1), \tag{A.1}$$
$$q_2 = (x_2, y_2, \theta_2), \tag{A.2}$$

the shortest path that can be traversed by the Reeds-Shepp car between q_1 and q_2, without colliding with obstacles, belongs to a set of paths known as Reeds-Shepp paths, shown in Table A.1, where L and R denote motion primitives corresponding to left and right motions with a minimum turning radius, respectively, S denotes a motion primitive corresponding to a straight motion, C denotes an L or R motion primitive, $|$ denotes a change from a forward motion to a reverse motion or vice versa and the $+$ and $-$ superscripts denote the forward and reverse directions of motion, respectively. Note that, as shown in [31], $L^-R^+L^-$ and $R^-L^+R^-$ do not belong to the set of Reeds-Shepp paths.

Path family	Path types
$C\|C\|C$	$L^+R^-L^+$, $R^+L^-R^+$
$CC\|C$	$L^+R^+L^-$, $L^-R^-L^+$, $R^+L^+R^-$, $R^-L^-R^+$
$C\|CC$	$L^+R^-L^-$, $L^-R^+L^+$, $R^+L^-R^-$, $R^-L^+R^+$
CSC	$L^+S^+L^+$, $L^+S^+R^+$, $L^-S^-L^-$, $L^-S^-R^-$, $R^+S^+L^+$, $R^+S^+R^+$, $R^-S^-L^-$, $R^-S^-R^-$
$CC\|CC$	$L^+R^+L^-R^-$, $L^-R^-L^+R^+$, $R^+L^+R^-L^-$, $R^-L^-R^+L^+$
$C\|CC\|C$	$L^+R^-L^-R^+$, $L^-R^+L^+R^-$, $R^+L^-R^-L^+$, $R^-L^+R^+L^-$
$C\|CSC$	$L^+R^-S^-L^-$, $L^+R^-S^-R^-$, $L^-R^+S^+L^+$, $L^-R^+S^+R^+$, $R^+L^-S^-L^-$, $R^+L^-S^-R^-$, $R^-L^+S^+L^+$, $R^-L^+S^+R^+$
$CSC\|C$	$L^+S^+L^+R^-$, $L^+S^+R^+L^-$, $L^-S^-L^-R^+$, $L^-S^-R^-L^+$, $R^+S^+L^+R^-$, $R^+S^+R^+L^-$, $R^-S^-L^-R^+$, $R^-S^-R^-L^+$
$C\|CSC\|C$	$L^+R^-S^-L^-R^+$, $L^-R^+S^+L^+R^-$, $R^+L^-S^-R^-L^+$, $R^-L^+S^+R^+L^-$

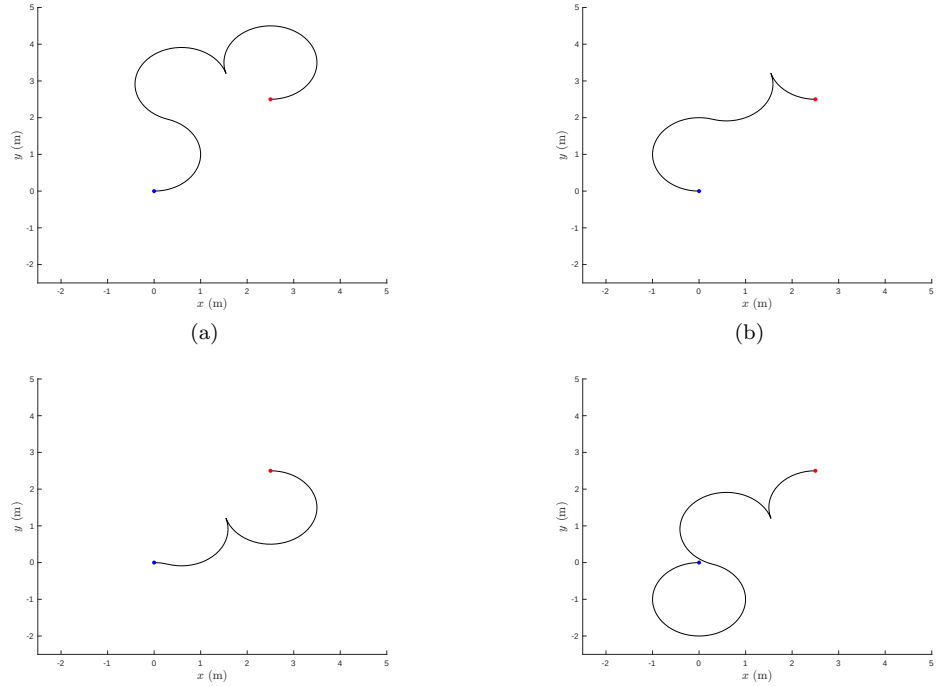
Table A.1: Reeds-Shepp path families and types.

Figures A.1–A.9 show the set of Reeds-Shepp paths, where $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ are shown in blue and red, respectively.



(a) (b)

Figure A.1: $C|C|C$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^-L^+$. (b) $R^+L^-R^+$.



(a) (b)

(c) (d)

Figure A.2: $CC|C$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^+L^-$. (b) $L^-R^-L^+$. (c) $R^+L^+R^-$. (d) $R^-L^-R^+$.

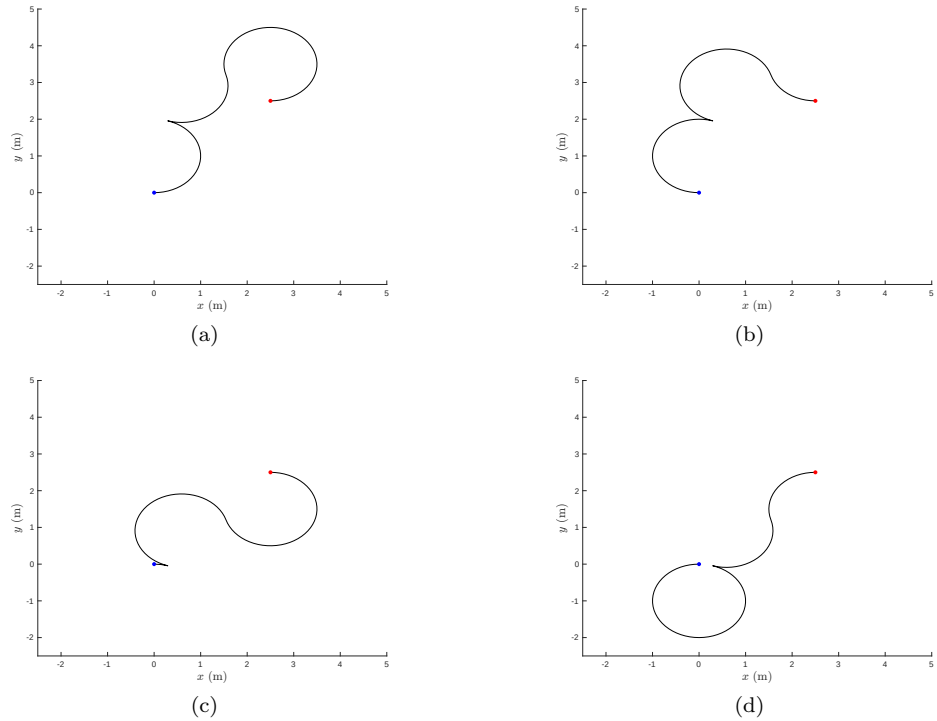


Figure A.3: $C|CC$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^-L^-$. (b) $L^-R^+L^+$. (c) $R^+L^-R^-$. (d) $R^-L^+R^+$.

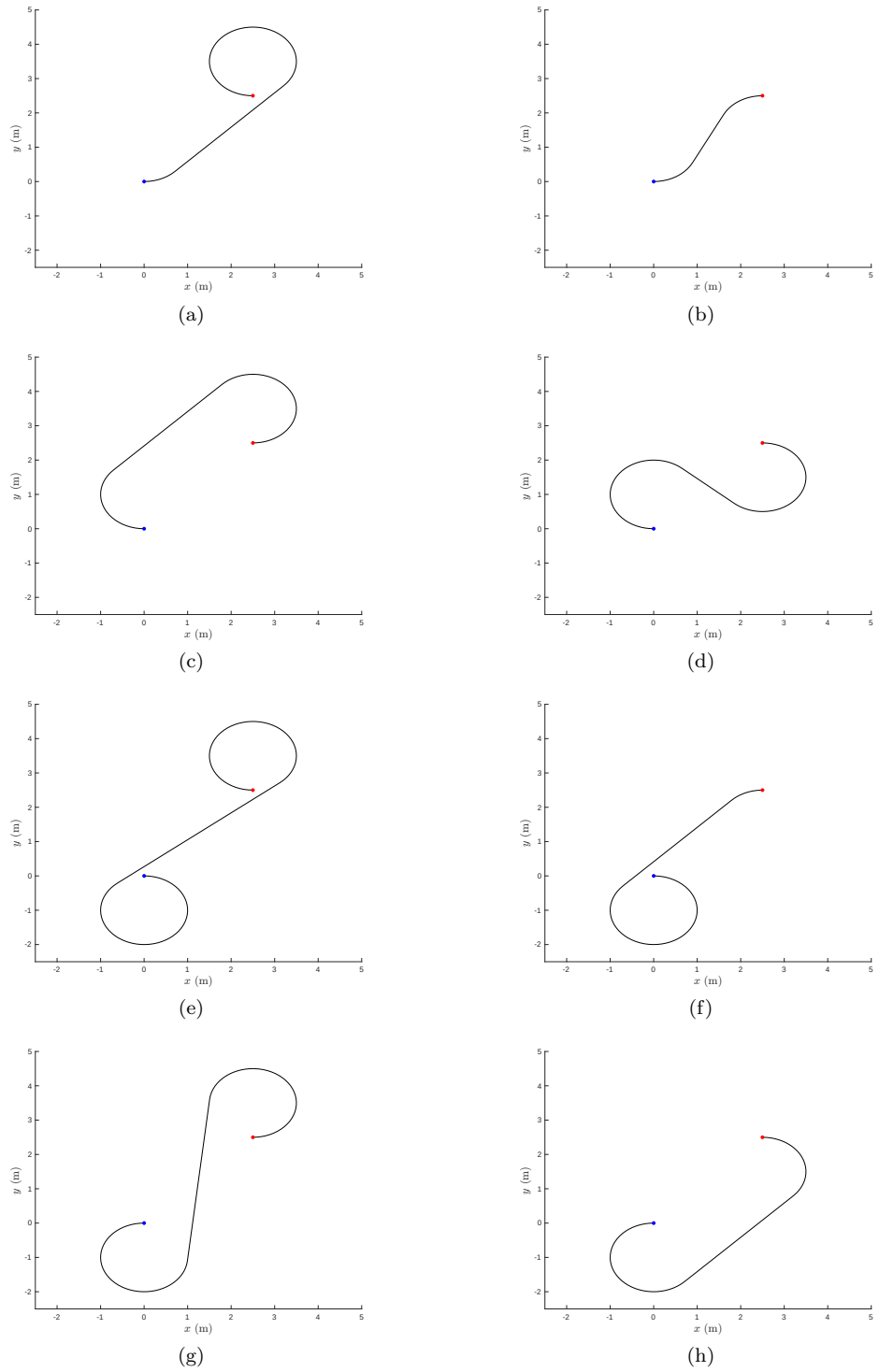


Figure A.4: CSC Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+S^+L^+$. (b) $L^+S^+R^+$. (c) $L^-S^-L^-$. (d) $L^-S^-R^-$. (e) $R^+S^+L^+$. (f) $R^+S^+R^+$. (g) $R^-S^-L^-$. (h) $R^-S^-R^-$.

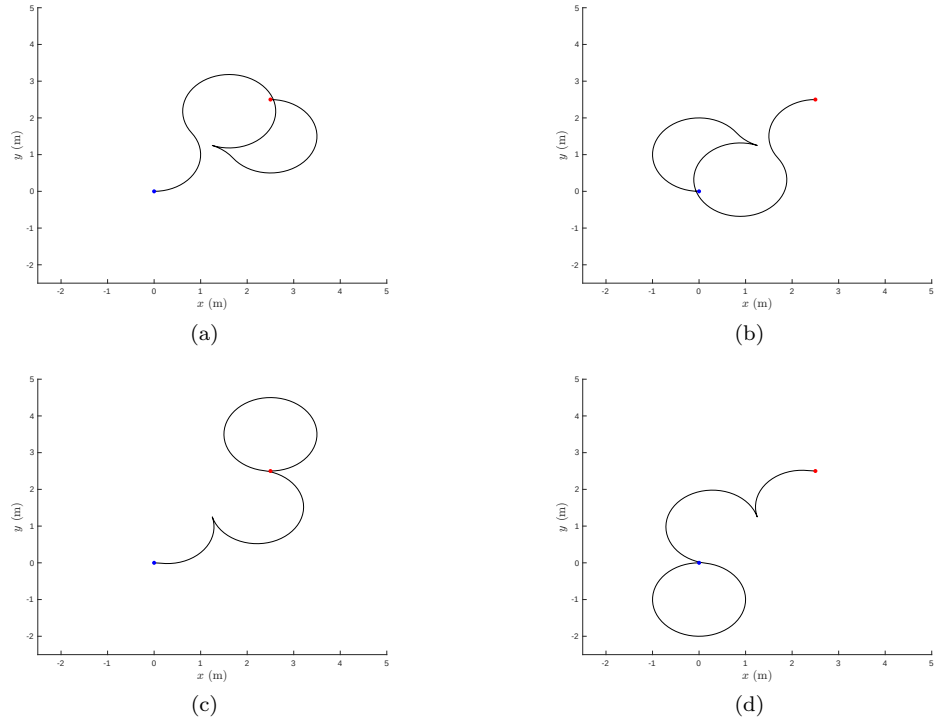


Figure A.5: $CC|CC$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^+L^-R^-$. (b) $L^-R^-L^+R^+$. (c) $R^+L^+R^-L^-$. (d) $R^-L^-R^+L^+$.

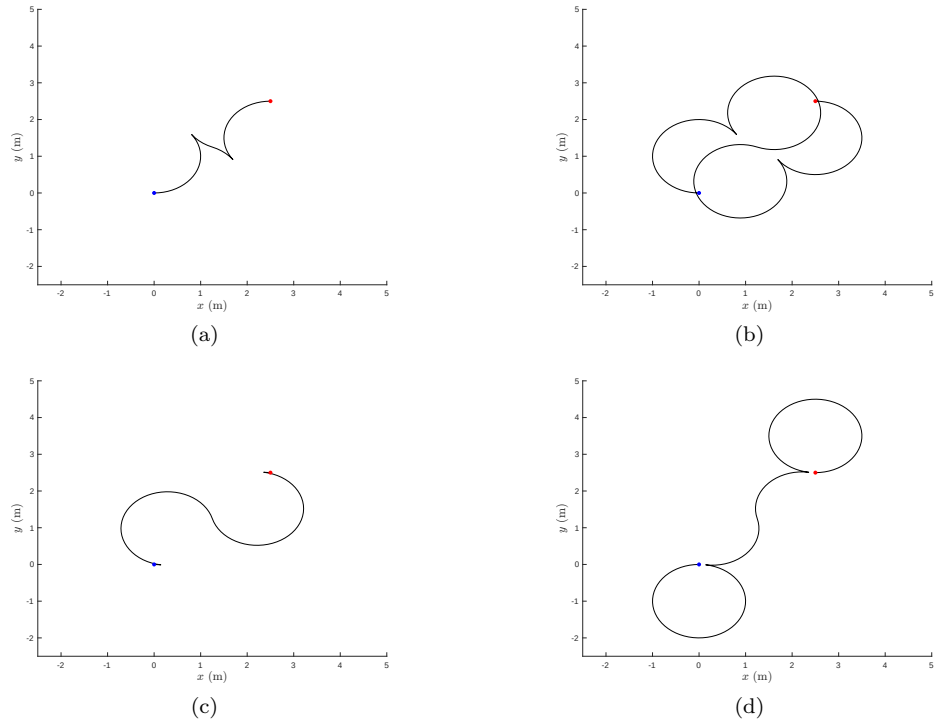


Figure A.6: $C|CC|C$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^-L^-R^+$. (b) $L^-R^+L^+R^-$. (c) $R^+L^-R^-L^+$. (d) $R^-L^+R^+L^-$.

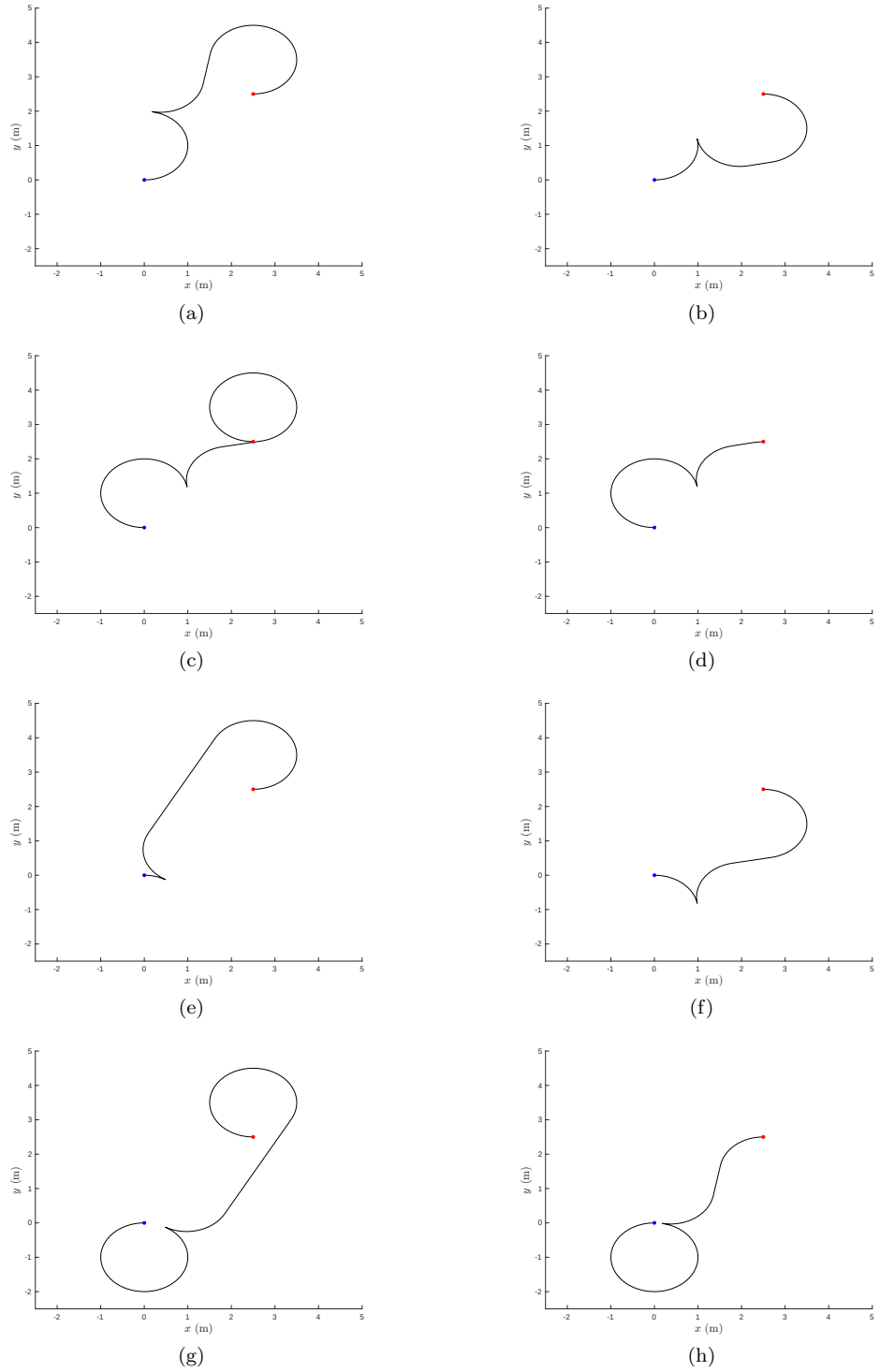


Figure A.7: $C|CSC$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+R^-S^-L^-$. (b) $L^+R^-S^-R^-$. (c) $L^-R^+S^+L^+$. (d) $L^-R^+S^+R^+$. (e) $R^+L^-S^-L^-$. (f) $R^+L^-S^-L^-$. (g) $R^-L^+S^+L^+$. (h) $R^-L^+S^+R^+$.

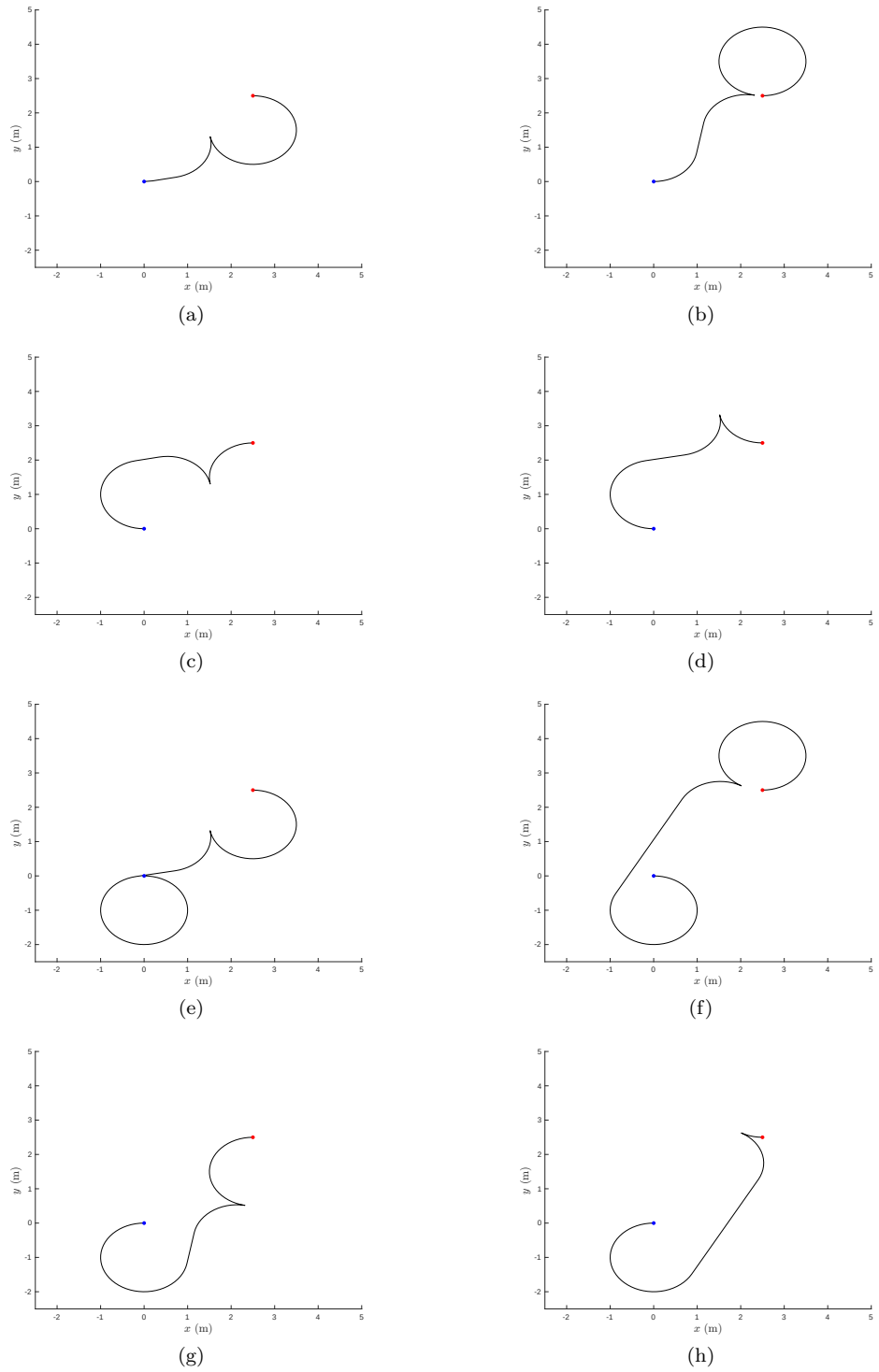


Figure A.8: $CSC|C$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1$ m. (a) $L^+S^+L^+R^-$. (b) $L^+S^+R^+L^-$. (c) $L^-S^-L^-R^+$. (d) $L^-S^-R^-L^+$. (e) $R^+S^+L^+R^-$. (f) $R^+S^+R^+L^-$. (g) $R^-S^-L^-R^+$. (h) $R^-S^-R^-L^+$.

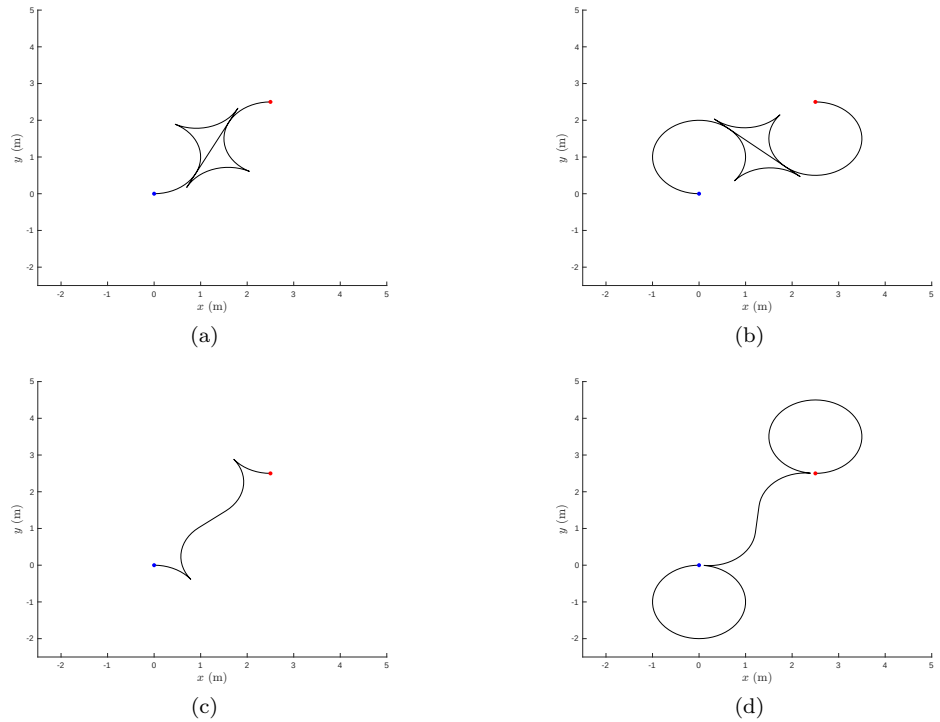


Figure A.9: $C|CSC|C$ Reeds-Shepp paths from $q_1 = (0\text{ m}, 0\text{ m}, 0\text{ rad})$ to $q_2 = (2.5\text{ m}, 2.5\text{ m}, 0\text{ rad})$, where $\phi_{\max} = 45°$ and $l = 1\text{ m}$. (a) $L^+R^-S^-L^-R^+$. (b) $L^-R^+S^+L^+R^-$. (c) $R^+L^-S^-R^-L^+$. (d) $R^-L^+S^+R^+L^-$.

Appendix B

Generating Reeds-Shepp Paths

Following the approach taken in [33], this appendix presents a compact set of closed-form equations that describe the set of Reeds-Shepp paths. They are compact in that they describe the set of Reeds-Shepp path families as opposed to the set of Reeds-Shepp path types. The information in this appendix is derived from [19] and [11].

Let $n \in \mathbb{N}$ denote the number of motion primitives in a given path.

For $i \in \mathbb{N}_{\leq n+1}$, let $q_i = (x_i, y_i, \theta_i) \in \mathbb{R}^2 \times \mathbb{S}^1$ denote the configuration of a Reeds-Shepp car such that, for $i \in \mathbb{N}_{\leq n}$,

$$g_{i+1} = g_i \exp(X_i \Delta t_i), \tag{B.1}$$

where, noting that $\mathbb{R}^2 \times \mathbb{S}^1 \simeq SE(2)$,

$$g_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & x_i \\ \sin(\theta_i) & \cos(\theta_i) & y_i \\ 0 & 0 & 1 \end{bmatrix} \in SE(2) \tag{B.2}$$

denotes the matrix representation of the configuration of the Reeds-Shepp car,

$$X_i = \begin{bmatrix} 0 & -\omega_i & v_i \\ \omega_i & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathfrak{se}(2) \tag{B.3}$$

denotes the matrix representation of the planar body twist of the Reeds-Shepp car between g_i and g_{i+1}, and $\Delta t_i \in \mathbb{R}_{\geq 0}$ denotes the duration of the motion from g_i to g_{i+1}, where $v_i \in \mathbb{R}_{\neq 0}$ and

$$\omega_i = \begin{cases} \frac{v_i}{r_i} & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0, \end{cases} \tag{B.4}$$

where $\phi_i \in [-\phi_{\max}, \phi_{\max}]$ and

$$r_i = \frac{l}{\tan(\phi_i)}. \tag{B.5}$$

Therefore,

$$g_{n+1} = g_1 \prod_{i=1}^{n} \exp(X_i \Delta t_i). \tag{B.6}$$

Note that

$$\exp(X_i \Delta t_i) = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) & \Delta x_i \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) & \Delta y_i \\ 0 & 0 & 1 \end{bmatrix}, \tag{B.7}$$

where

$$\Delta x_i = \begin{cases} r_i \sin(\Delta\theta_i) & \text{if } \phi_i \neq 0, \\ v_i \Delta t_i & \text{if } \phi_i = 0, \end{cases} \tag{B.8}$$

$$\Delta y_i = \begin{cases} r_i\big(1 - \cos(\Delta\theta_i)\big) & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0, \end{cases} \tag{B.9}$$

$$\Delta\theta_i = \begin{cases} \frac{v_i \Delta t_i}{r_i} & \text{if } \phi_i \neq 0, \\ 0 & \text{if } \phi_i = 0 \end{cases} \tag{B.10}$$

denote the changes along the x, y and θ directions of $\{b\}$, respectively. Therefore, for $i \in \mathbb{N}_{\leq n}$,

$$\Delta t_i = \begin{cases} \frac{\Delta\theta_i r_i}{v_i} & \text{if } \phi_i \neq 0, \\ \frac{\Delta x_i}{v_i} & \text{if } \phi_i = 0, \end{cases} \tag{B.11}$$

where Δx_i and $\Delta\theta_i$ may be determined using (B.6). Given that g_1 and g_{n+1} are known, it will be beneficial to repose (B.6) as

$$g_1^{-1} g_{n+1} = \prod_{i=1}^{n} \exp(X_i \Delta t_i), \tag{B.12}$$

where

$$g_1^{-1} g_{n+1} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & \Delta x \\ \sin(\Delta\theta) & \cos(\Delta\theta) & \Delta y \\ 0 & 0 & 1 \end{bmatrix}, \tag{B.13}$$

where

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 \\ -\sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n+1} - x_1 \\ y_{n+1} - y_1 \\ \theta_{n+1} - \theta_1 \end{bmatrix}. \tag{B.14}$$

Necessary conditions for Reeds-Shepp paths to exist between g_1 and g_{n+1} are shown in Tables B.1–B.3.

Path family	$\lvert\Delta\theta_1\rvert$	$\lvert\Delta\theta_2\rvert$	$\lvert\Delta\theta_3\rvert$	$\lvert\Delta\theta_4\rvert$	$\lvert\Delta\theta_5\rvert$
$C\lvert C\lvert C$	$[0, \pi]$	$[0, \pi]$	$[0, \pi]$	–	–
$CC\lvert C$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	$\left[0, \frac{\pi}{2}\right]$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	–	–
$C\lvert CC$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	$\left[0, \frac{\pi}{2}\right]$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	–	–
CSC	$\left[0, \frac{\pi}{2}\right]$	0	$\left[0, \frac{\pi}{2}\right]$	–	–
$CC\lvert CC$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	$\left[0, \frac{\pi}{2}\right]$	$\lvert\Delta\theta_2\rvert$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	–
$C\lvert CC\lvert C$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	$\left[0, \frac{\pi}{2}\right]$	$\lvert\Delta\theta_2\rvert$	$\left[0, \lvert\Delta\theta_2\rvert\right]$	–
$C\lvert CSC$	$\left[0, \frac{\pi}{2}\right]$	$\frac{\pi}{2}$	0	$\left[0, \frac{\pi}{2}\right]$	–
$CSC\lvert C$	$\left[0, \frac{\pi}{2}\right]$	0	$\frac{\pi}{2}$	$\left[0, \frac{\pi}{2}\right]$	–
$C\lvert CSC\lvert C$	$\left[0, \frac{\pi}{2}\right]$	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$	$\left[0, \frac{\pi}{2}\right]$

Table B.1: Reeds-Shepp path turning circle angles.

Let

$$R_1 = \begin{bmatrix} \cos(\Delta\theta_1) & -\sin(\Delta\theta_1) \\ \sin(\Delta\theta_1) & \cos(\Delta\theta_1) \end{bmatrix}. \tag{B.15}$$

Path family	v_1	v_2	v_3	v_4	v_5
$C\|C\|C$	$\mathbb{R}_{>0}$	$-v_1$	$-v_2$	–	–
$CC\|C$	$\mathbb{R}_{\neq 0}$	v_1	$-v_2$	–	–
$C\|CC$	$\mathbb{R}_{\neq 0}$	$-v_1$	v_2	–	–
CSC	$\mathbb{R}_{\neq 0}$	v_1	v_2	–	–
$CC\|CC$	$\mathbb{R}_{\neq 0}$	v_1	$-v_2$	v_3	–
$C\|CC\|C$	$\mathbb{R}_{\neq 0}$	$-v_1$	v_2	$-v_3$	–
$C\|CSC$	$\mathbb{R}_{\neq 0}$	$-v_1$	v_2	v_3	–
$CSC\|C$	$\mathbb{R}_{\neq 0}$	v_1	v_2	$-v_3$	–
$C\|CSC\|C$	$\mathbb{R}_{\neq 0}$	$-v_1$	v_2	v_3	$-v_4$

Table B.2: Reeds-Shepp path velocities.

Path family	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
$C\|C\|C$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$	–	–
$CC\|C$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$	–	–
$C\|CC$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$	–	–
CSC	$\{-\phi_{\max}, \phi_{\max}\}$	0	$\{-\phi_{\max}, \phi_{\max}\}$	–	–
$CC\|CC$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$	$-\phi_3$	–
$C\|CC\|C$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	$-\phi_2$	$-\phi_3$	–
$C\|CSC$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	0	$\{-\phi_{\max}, \phi_{\max}\}$	–
$CSC\|C$	$\{-\phi_{\max}, \phi_{\max}\}$	0	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_3$	–
$C\|CSC\|C$	$\{-\phi_{\max}, \phi_{\max}\}$	$-\phi_1$	0	$-\phi_2$	$-\phi_4$

Table B.3: Reeds-Shepp path steering angles.

For $C|C|C$, $CC|C$ and $C|CC$ Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = r_1\left(\begin{bmatrix}\sin(\Delta\theta)\\ 1-\cos(\Delta\theta)\end{bmatrix} - 2R_1\begin{bmatrix}\sin(\Delta\theta_2)\\ 1-\cos(\Delta\theta_2)\end{bmatrix}\right), \tag{B.16}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3. \tag{B.17}$$

Therefore, to generate $C|C|C$, $CC|C$ and $C|CC$ Reeds-Shepp paths between g_1 and g_4 using (B.11), for $i \in \{1,2,3\}$, let

$$\Delta\theta_i = \operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\big(\operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\Delta\theta_i' \bmod 2\pi\big), \tag{B.18}$$

where

$$\Delta\theta_1' = 2\arctan\left(\frac{\sqrt{\alpha^2+\beta^2}-\alpha}{\beta}\right) - \frac{\Delta\theta_2'}{2}, \tag{B.19}$$

$$\Delta\theta_2' = \arccos\left(1-\frac{\alpha^2+\beta^2}{2}\right), \tag{B.20}$$

$$\Delta\theta_3' = \Delta\theta - \Delta\theta_1' - \Delta\theta_2', \tag{B.21}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix}\sin(\Delta\theta)\\ 1-\cos(\Delta\theta)\end{bmatrix} - \frac{1}{r_1}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right). \tag{B.22}$$

For CSC Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = R_1 \begin{bmatrix}\Delta x_2\\ r_3 - r_1\end{bmatrix} + \begin{bmatrix} r_3 \sin(\Delta\theta)\\ r_1 - r_3\cos(\Delta\theta)\end{bmatrix}, \tag{B.23}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_3. \tag{B.24}$$

Therefore, to generate CSC Reeds-Shepp paths between g_1 and g_4 using (B.11), let

$$\Delta x_2 = \operatorname{sgn}(v_2)\sqrt{\alpha^2 + \beta^2 - (r_3 - r_1)^2} \tag{B.25}$$

and, for $i \in \{1, 3\}$,

$$\Delta\theta_i = \operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\big(\operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\Delta\theta_i' \bmod 2\pi\big), \tag{B.26}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} - \begin{bmatrix} r_3 \sin(\Delta\theta)\\ r_1 - r_3\cos(\Delta\theta)\end{bmatrix}, \tag{B.27}$$

$$\Delta\theta_1' = 2\arctan\left(\frac{\Delta x_2 - \alpha}{\beta + r_3 - r_1}\right), \tag{B.28}$$

$$\Delta\theta_3' = \Delta\theta - \Delta\theta_1'. \tag{B.29}$$

For $CC|CC$ and $C|CC|C$ Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = r_1\left(\begin{bmatrix}-\sin(\Delta\theta)\\ 1 + \cos(\Delta\theta)\end{bmatrix} - 2R_1\begin{bmatrix}\sin(\Delta\theta_2)\\ 2 - \cos(\Delta\theta_2)\end{bmatrix}\right), \tag{B.30}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_4. \tag{B.31}$$

Therefore, to generate $CC|CC$ and $C|CC|C$ Reeds-Shepp paths between g_1 and g_5 using (B.11), for $i \in \{1, 2, 4\}$, let

$$\Delta\theta_i = \operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\big(\operatorname{sgn}(v_i)\operatorname{sgn}(\phi_i)\Delta\theta_i' \bmod 2\pi\big), \tag{B.32}$$

where

$$\Delta\theta_1' = \operatorname{sgn}(\phi_1)\arccos\left(\frac{\alpha\sqrt{16 - (\alpha^2 + \beta^2 - 5)^2} + \beta(\alpha^2 + \beta^2 + 3)}{4(\alpha^2 + \beta^2)}\right), \tag{B.33}$$

$$\Delta\theta_2' = \arccos\left(\frac{5 - \alpha^2 - \beta^2}{4}\right), \tag{B.34}$$

$$\Delta\theta_4' = \Delta\theta - \Delta\theta_1', \tag{B.35}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{2}\left(\begin{bmatrix}-\sin(\Delta\theta)\\ 1 + \cos(\Delta\theta)\end{bmatrix} - \frac{1}{r_1}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right). \tag{B.36}$$

For $C|CSC$ Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = R_1\begin{bmatrix}-(r_1 + r_4)\sin(\Delta\theta_2)\\ \Delta x_3\sin(\Delta\theta_2) - 2r_1\end{bmatrix} + \begin{bmatrix} r_4\sin(\Delta\theta)\\ r_1 - r_4\cos(\Delta\theta)\end{bmatrix}, \tag{B.37}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_2 + \Delta\theta_4. \tag{B.38}$$

Therefore, to generate $C|CSC$ Reeds-Shepp paths between g_1 and g_5 using (B.11), let

$$\Delta x_3 = \operatorname{sgn}(v_3)\sqrt{\alpha^2 + \beta^2 - (r_1 + r_4)^2} + 2r_1\sin\left(\Delta\theta_2'\right) \tag{B.39}$$

and, for $i \in \{1, 2, 4\}$,

$$\Delta\theta_i = \text{sgn}(v_i)\text{sgn}(\phi_i)\big(\text{sgn}(v_i)\text{sgn}(\phi_i)\Delta\theta'_i \bmod 2\pi\big), \tag{B.40}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix} r_4\sin(\Delta\theta)\\ r_1 - r_4\cos(\Delta\theta)\end{bmatrix} - \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}, \tag{B.41}$$

$$\Delta\theta'_1 = 2\arctan\left(\frac{\Delta x_3 + (\beta - 2r_1)\sin\left(\Delta\theta'_2\right)}{\alpha\sin\left(\Delta\theta'_2\right) + r_1 + r_4}\right), \tag{B.42}$$

$$\Delta\theta'_2 = \text{sgn}(v_2)\text{sgn}(\phi_2)\left(\frac{\pi}{2}\right), \tag{B.43}$$

$$\Delta\theta'_4 = \Delta\theta - \Delta\theta'_1 - \Delta\theta'_2. \tag{B.44}$$

For $CSC|C$ Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = R_1\begin{bmatrix}\Delta x_2 + 2r_3\sin(\Delta\theta_3)\\ r_3 - r_1\end{bmatrix} + \begin{bmatrix}-r_3\sin(\Delta\theta)\\ r_1 + r_3\cos(\Delta\theta)\end{bmatrix}, \tag{B.45}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_3 + \Delta\theta_4. \tag{B.46}$$

Therefore, to generate $CSC|C$ Reeds-Shepp paths between g_1 and g_5 using (B.11), let

$$\Delta x_2 = \text{sgn}(v_2)\sqrt{\alpha^2 + \beta^2 - (r_3 - r_1)^2} - 2r_3\sin\left(\Delta\theta'_3\right) \tag{B.47}$$

and, for $i \in \{1, 3, 4\}$,

$$\Delta\theta_i = \text{sgn}(v_i)\text{sgn}(\phi_i)\big(\text{sgn}(v_i)\text{sgn}(\phi_i)\Delta\theta'_i \bmod 2\pi\big), \tag{B.48}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} - \begin{bmatrix}-r_3\sin(\Delta\theta)\\ r_1 + r_3\cos(\Delta\theta)\end{bmatrix}, \tag{B.49}$$

$$\Delta\theta'_1 = 2\arctan\left(\frac{\Delta x_2 + 2r_3\sin\left(\Delta\theta'_3\right) - \alpha}{\beta + r_3 - r_1}\right), \tag{B.50}$$

$$\Delta\theta'_3 = \text{sgn}(v_3)\text{sgn}(\phi_3)\left(\frac{\pi}{2}\right), \tag{B.51}$$

$$\Delta\theta'_4 = \Delta\theta - \Delta\theta'_1 - \Delta\theta'_3. \tag{B.52}$$

For $C|CSC|C$ Reeds-Shepp paths,

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = R_1\begin{bmatrix}-(r_1 + r_4)\sin(\Delta\theta_2)\\ \big(\Delta x_3 + 2r_4\sin(\Delta\theta_4)\big)\sin(\Delta\theta_2) - 2r_1\end{bmatrix} + \begin{bmatrix}-r_4\sin(\Delta\theta)\\ r_1 + r_4\cos(\Delta\theta)\end{bmatrix}, \tag{B.53}$$

$$\Delta\theta = \Delta\theta_1 + \Delta\theta_2 + \Delta\theta_4 + \Delta\theta_5. \tag{B.54}$$

Therefore, to generate $C|CSC|C$ Reeds-Shepp paths between g_1 and g_6 using (B.11), let

$$\Delta x_3 = \text{sgn}(v_3)\sqrt{\alpha^2 + \beta^2 - (r_1 + r_4)^2} + 2\Big(r_1\sin\left(\Delta\theta'_2\right) - r_4\sin\left(\Delta\theta'_4\right)\Big) \tag{B.55}$$

and, for $i \in \{1, 2, 4, 5\}$,

$$\Delta\theta_i = \text{sgn}(v_i)\text{sgn}(\phi_i)\big(\text{sgn}(v_i)\text{sgn}(\phi_i)\Delta\theta'_i \bmod 2\pi\big), \tag{B.56}$$

where

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}-r_4\sin(\Delta\theta)\\ r_1 + r_4\cos(\Delta\theta)\end{bmatrix} - \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}, \tag{B.57}$$

$$\Delta\theta_1' = 2\arctan\left(\frac{\Delta x_3 + (\beta - 2r_1)\sin\left(\Delta\theta_2'\right) + 2r_4\sin\left(\Delta\theta_4'\right)}{\alpha\sin\left(\Delta\theta_2'\right) + r_1 + r_4}\right), \tag{B.58}$$

$$\Delta\theta_2' = \operatorname{sgn}(v_2)\operatorname{sgn}(\phi_2)\left(\frac{\pi}{2}\right), \tag{B.59}$$

$$\Delta\theta_4' = \operatorname{sgn}(v_4)\operatorname{sgn}(\phi_4)\left(\frac{\pi}{2}\right), \tag{B.60}$$

$$\Delta\theta_5' = \Delta\theta - \Delta\theta_1' - \Delta\theta_2' - \Delta\theta_4'. \tag{B.61}$$

Note that

$$\Delta t_i = \begin{cases}\frac{\Delta\theta_i' r_i}{v_i} \bmod \frac{2\pi|r_i|}{|v_i|} & \text{if } \phi_i \neq 0,\\ \frac{\Delta x_i}{v_i} & \text{if } \phi_i = 0\end{cases} \tag{B.62}$$

may be used instead of (B.11).

These equations were used to generate the paths presented in Figures A.1–A.9, demonstrating their efficacy.