Automation and Navigation of a Terrestrial Vehicle

by

Wynand Visser

Thesis presented in partial fulfilment of the requirements
for the degree of Master of Science in Engineering at
Stellenbosch University

Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, 7602, Matieland, South Africa.

Supervisor: Prof. T. Jones

March 2012

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2012

Summary

This thesis presents the design and implementation of an autonomous navigational system and the automation of a practical demonstrator vehicle. It validates the proposed navigation architecture using simple functional navigational modules on the said vehicle.

The proposed navigation architecture is a hierarchical structure, with a mission planner at the top, followed by the route planner, the path planner and a vehicle controller with the vehicle hardware at the base. A vehicle state estimator and mapping module runs in parallel to provide feedback data.

The controls of an all terrain vehicle are electrically actuated and equipped with feedback sensors to form a complete drive-by-wire solution. A steering controller and velocity control state machine are designed and implemented on an existing on-board controller that includes a six degrees-of-freedom kinematic state estimator.

A lidar scanner detects obstacles. The lidar data is mapped in real time to a local three-dimensional occupancy grid using a Bayesian update process. Each lidar beam is projected within the occupancy grid and the occupancy state of affected cells is updated. A lidar simulation environment is created to test the mapping module before practical implementation. For planning purposes, the three-dimensional occupancy grid is converted to a two-dimensional drivability map.

The path planner is an adapted rapidly exploring random tree (RRT) planner, that assumes Dubins car kinematics for the vehicle. The path planner optimises a cost function based on path length and a risk factor that is derived from the drivability map.

A simple mission planner that accepts user-defined waypoints as objectives is implemented. Practical tests verified the potential of the navigational structure implemented in this thesis.

Opsomming

In hierdie tesis word die ontwerp en implementering van 'n outonome navigasiestelsel weergegee, asook die outomatisering van 'n praktiese demonstrasievoertuig. Dit regverdig die voorgestelde navigasie-argitektuur op die bogenoemde voertuig deur gebruik te maak van eenvoudige, funksionele navigasie-modules.

Die voorgestelde navigasie-argitektuur is 'n hiërargiese struktuur, met die missie-beplanner aan die bo-punt, gevolg deur die roetebeplanner, die padbeplanner en voertuigbeheerder, met die voertuighardeware as basisvlak. 'n Voertuigtoestandsafskatter en karteringsmodule loop in parallel om terugvoer te voorsien.

Die kontroles van 'n vierwiel-motorfiets is elektries geaktueer en met terugvoersensors toegerus om volledig rekenaarbeheerd te wees. 'n Stuur-beheerder en 'n snelheid-toestandmasjien is ontwerp en geïmplementeer op 'n bestaande aanboordverwerker wat 'n kinematiese toestandsafskatter in ses grade van vryheid insluit.

'n Lidar-skandeerder registreer hindernisse. Die lidar-data word in reële tyd na 'n lokale drie-dimensionele besettingsrooster geprojekteer deur middel van 'n Bayesiese opdateringsproses. Elke lidar-straal word in die besettingsrooster geprojekteer en die besettingstoestand van betrokke selle word opdateer. 'n Lidar-simulasie-omgewing is geskep om die karteringsmodule te toets voor dit geïmplementeer word. Die drie-dimensionele besettingsrooster word na 'n twee-dimensionele rybaarheidskaart verwerk vir beplanningsdoeleindes.

Die padbeplanner is 'n aangepaste spoedig-ontdekkende-lukrake-boom en neem Dubinskar kinematika vir die voertuig aan. Die padbeplanner optimeer 'n koste-funksie, gebaseer op padlengte en 'n risiko-faktor, wat vanaf die rybaarheidskaart verkry word.

'n Eenvoudige missie-beplanner, wat via-punte as doelstellings neem, is geïmplementeer. Praktiese toetsritte verifieer die potensiaal van die navigasiestruktuur, soos hier beskryf.

Acknowledgements

I would like to extend my gratitude to everybody who contributed towards my efforts during the course of this project. I would like thank the following people for their particular contributions:

- My study leader, Prof. Thomas Jones, for the encouragement and freedom to make this project my own.
- Chris Jaquet for your valued input in every aspect of this project and especially during the write-up.
- Wessel Croukamp en Lincoln Saunders by SED, vir die besondere netjiese vervaardiging van masjieneerde parte vir die aktueerders.
- Vir Dr. Corné van Daalen se leiding en insig met betrekking tot autonome navigasie en padbeplanning.
- Vir almal in die laboratorium wie se insette en belangstelling bygedra het en in die besonder vir AM en Lionel se hulp gedurende toetsritte.
- My opregte waardering aan al my vriende wie se woorde van bemoeding oor 'n koppie koffie die werk soveel makliker laat voel het.
- My gesin en geliefdes wie se volgehoue ondersteuning my inspireer om my passie uit te leef.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

2D	Two-Dimensional
3D	Three-Dimensional
ADC	Analogue-to-Digital Converter
ATV	All Terrain Vehicle
CAN	Controller Area Network
CVT	Continuously Variable Transmission
DC	Direct Current
DGPS	Differential Global Positioning System
DPLL	Digital Phase Locked Loop
GPS	Global Positioning System
IMU	Inertial Measurement Unit
Lidar	Light Detection And Ranging
LPM	Local Planning Method
OBC	On-board Controller
OG	Occupancy Grid
PTU	Pan/Tilt Unit
PWM	Pulse-Width Modulation

Radar	Radio Detection And Ranging
RPM	Revolutions per Minute
RRT	Rapidly exploring Random Tree
SLAM	Simultaneous Localisation And Mapping
Sonar	Sound Navigation And Ranging

Superscripts

S	Sensor
V	Vehicle

Greek Symbols

Θ	Pitch
Φ	Roll
Ψ	Yaw
α	Occupancy grid resolution
Δ	Change in
$\Delta()$	Range distribution
$\Gamma()$	Angular distribution
η	Efficiency
ρ	Range from sensor
σ	Standard deviation
τ	Torque or path

Lower Case Symbols

a	General object dimension
h	General height dimension

l	Motor constant
l	Moment arm
q	Configuration
r	Range measurement
x	Cross-track error

Subscripts

0	Axis origin
B	Vehicle body axis
i	logical integer index relating to x
I	Inertial Axis
j	logical integer index relating to y
k	logical integer index relating to z
n	general integer index
M	Map Axis
S	Sensor Axis
rec	Recognition
xy	Horisontal dimension
z	Vertical dimension

Upper Case Symbols

C	Cell in occupancy grid
$\mathcal{C}$	Configuration space
EMP	Empty state
F	Force

I	Current
$\mathcal{M}$	Map
$\mathbf{O}$	Orientation
OCC	Occupied state
$O()$	"Big Oh" notation / Order of complexity
$\mathbf{P}$	Position
R	Risk variable
T	Time period
$\mathbf{T}$	Rotation matrix
U	Forward velocity
$\mathbf{V}$	Velocity
$\mathcal{V}$	Vehicle
$\mathcal{W}$	World

Chapter 1

Introduction

1.1 Motivation

Autonomous self navigating vehicles are a dream that have been pursued for a number of decades. They require the amalgamation of a variety of disciplines: mechanics, control systems, sensing and signal processing to name but a few - each of which faces its own challenges with additional complexity when they interact.

Apart from the academic challenge (or the appeal to science fiction authors), autonomous vehicles potentially have many advantages over vehicles that rely wholly, or in part, on human control:

- Increased safety with faster reaction times and no limit on concentration span, especially for dull or repetitive tasks.
- Incurring economic savings by optimising multiple, and often underlying, cost factors.
- Operation in areas not feasible for human presence, such as disaster relief, rescue missions, war zones or exploration in areas that are dangerous.
- The predictable nature of a machine versus an employee.

The Autonomous Navigation (AutoNav) research group, formed at Stellenbosch University, has the aim to research and develop techniques required to create systems capable of such autonomous navigation.

1.2 Overview of Navigation

As a first attempt at self navigation, it is important to define the architecture of such a system. Since navigation is traditionally a human task, initial systems will tend to mimic the assumed human thought process and this is reflected in our choice of architecture. The architecture defines the information flow from goal to execution. Some concepts described are intentionally vague to prevent loss of generality of descriptions. This overview will introduce the most notable terms in the context of this thesis.

The purpose of travelling in a vehicle is usually to reach some goal, which we defined as the objectives of the mission. Achieving objectives is implemented in the **mission planner**. The nature of the objectives to be reached can greatly influence the mission. Mission planning is the top level of self navigation and its validity is best understood by listing some examples of objectives:

- Transporting cargo (which may include passengers) from origin to destination.
- Patrolling an area continuously.
- Exploring or mapping an unknown area.
- Following, finding or evading another entity.

The output of the mission planner is locations or manoeuvres in space, commonly referred to as waypoints, which are desirable in the fulfilment of the mission objectives. The constraints on these waypoints, such as order, precision or time at which they must be achieved, are largely determined by the nature of the mission. Changes to the objectives implies replanning the mission.

Once these waypoints are defined, a **route planner** determines the connections between waypoints, in accordance with the constraints imposed by the mission planner. In general it cannot be assumed that all (or any) information is known in advance about the operating environment. In other words, a complete and static map cannot be assumed. The level of detail in the planned route should therefore match the level of confidence and detail in the map.

Path planning is the detailed planning step. It outputs the exact trajectory the vehicle should follow. The path should be effective immediately and be complete up to some defined horizon. No further refinement should be necessary before passing this on to the vehicle controller. The path planner should therefore consider all vehicle, dynamic and mission constraints. The planning horizon reflects the practical distance (along space, time or some other metric) over which the map detail is considered complete.

Path planning investigates the direct interaction between the vehicle and its operating environment. It evaluates possible conflict between the environment and the vehicle. If a path involves probable conflict, a **conflict resolution** process finds an alternative path with an acceptably low probability of conflict.

Execution of a path is handled by a vehicle **controller**. A typical controller consist of a measurement system and a feedback control algorithm for actuators. Various sensors provide input to the state **estimator**, which determines vehicle state (position, orientation, related velocities and possibly other states). Feedback control ensures stable and accurate tracking of paths and manoeuvres.

All levels of planning require some representation of the operating environment, provided by a **mapping** module. Information may be known before the mission commences, be gathered as it progresses along the mission, or both. Mapping requires smart environmental sensors, integration with the pose estimator and a convenient output to all levels of planning.

This architecture is summarised in Figure 1.1. In a specific implementation of a mission, it may well be possible that a certain module or level may be redundant. Or that the complexity of interaction between adjacent modules becomes so interwoven, that it would be best to integrate them. It is important to note that some adaptation of this architecture will be required for specific cases.

1.3 Project Scope

The aim of this project is to implement the structure as set out in Section 1.2. Implementation includes a practical demonstrator on a suitable vehicle. The scope of a complete system far exceeds the capabilities of a single researcher, therefore this project

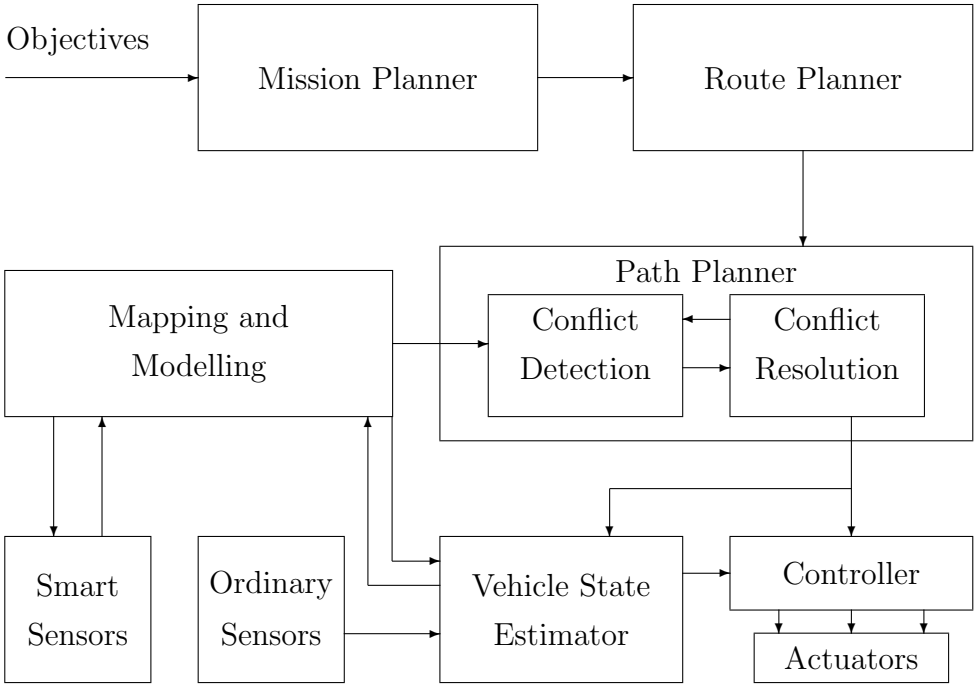


Figure 1.1: Overview of General Navigation Architecture[1]

will focus on installing simple, but functional, modules, which may be expanded by future researchers. This section describes the constraints of this thesis, with short motivations for constraining the project.

1.3.1 Operating Environment

The first constraint is a decision between indoor or outdoor operation. Indoor environments are generally limited in space. Furthermore, indoor environments are typically well structured, i.e. rooms and hallways is normally made up of flat surfaces, joined at perfect right angles (walls, floors, ceilings) with a near-perfect driving surface. The few exceptions include furniture and staircases. Outdoor environments can be chosen to be arbitrarily complex, which is convenient for incrementally more challenging testing. A secondary aspect is the availability of Global Positioning System (GPS) position measurements when operating outdoors.

The decision was made to use an outdoor terrestrial (ground based) vehicle to maintain generality. This limits the operating space to the two-dimensional projection of the driving surface onto a plane. The demonstration will be confined to a localised area within which the earth's curvature may be neglected.

[1]Derived from the architecture presented by Corné van Daalen internally to the AutoNav group.

Furthermore, the terrain will be sufficiently level such that the ground may be assumed drivable for the class of vehicle chosen i.e. no rugged or slippery terrain. Obstacle dimensions should be clearly distinguishable from the level surface and, by implication, a collision will be deemed catastrophic and unacceptable.

The final environmental constraint is to define the environment to be static. That is, no obstacles shall enter into, be removed from, or be allowed to move or change shape within the environment whilst the vehicle is operating.

1.3.2 Mission Planner

It is not feasible to try and implement every type and variation of imaginable missions. A representative mission is to simply drive from an initial location to a destination location while avoiding collisions. By extension this enables navigation through multiple waypoints and therefore more complex missions. Missions with changing waypoints could be accommodated by completely replanning on every change.

1.3.3 Route planner

The route planner will not be loaded with any prior information about the environment. This implies the maximum possible uncertainty and therefore no detailed route planning can be carried out. The route planner will revert to connecting the waypoints with straight lines and refer all planning tasks to the path planner.

1.3.4 Path Planner

To demonstrate practical obstacle avoidance, a functional path planner will be investigated and implemented. It should make provision for an incomplete map being populated as the mission progresses, but not for dynamic objects in the map.

As paths are calculated continuously, it is imperative that the path planner should execute in real time. In addition, it should demonstrate the possibility of implementing optimality within the solution.

The specific vehicle determines the possible modes of locomotion with its associated kinodynamic constraints. These constraints are inherited by the path planner. It follows that the path planner is very specific to vehicle class and therefore cannot be a general solution. The most common vehicle is a four wheeled car with limited turning circle and is the model to which the path planner shall be tailored.

1.3.5 Vehicle Controller

The vehicle controller executes the paths generated by the path planner. As described in Section 1.3.4, the path planner is specific to a vehicle class and as such, the controller cannot escape the same fate. To facilitate software reuse, the vehicle controller is split into inner-loop actuator controllers and high-level path-tracking controllers. In this manner, an upgrade of either the vehicle or path planner requires only partial rework.

1.3.6 Vehicle State Estimator

It will not be required to implement a Simultaneous Localisation and Mapping (SLAM) system. SLAM is the process whereby the mapping and state estimation is combined, i.e. the map consists of landmarks previously captured by sensors and the vehicle position is inversely determined from recognising the known landmarks. SLAM is the subject of parallel research within the AutoNav group and will not be covered here.

Localisation will be achieved via an available kinematic state estimator, which combines an inertial measurement unit (IMU) and high precision differential GPS (DGPS) system. The combined accuracy of the DGPS and update rate of the IMU dead reckoning is known to be sufficient for control applications.

1.3.7 Mapping

To demonstrate self navigation, it is required that the vehicle be able to detect and react to conflict within its surroundings. The system should be be able to reliably detect obstacles as described in Section 1.3.1 to avoid collision.

The known sensory systems are compared and the most suitable solution is implemented. It requires determining a feasible representation of the mapped environment and processing it for use with the planning algorithms. Since the sensor system will require feedback from the vehicle estimator, it is classified as a smart sensor.

1.4 Summary of Scope

The outputs of this project are:

- An actuated vehicle, fit for autonomous self navigation
- An architectural framework for navigation
- A path planning algorithm for the specific vehicle
- A controller able to follow generated paths
- A smart sensor for obstacle detection

1.5 Contributions

This thesis contributed the following items to the autonomous navigation research effort at Stellenbosch University:

- Validation of the proposed architectural framework
- A fully actuated drive-by-wire vehicle with an actuator controller and sensor board
- A three-dimensional occupancy grid mapping software module that can map lidar data in real time, accompanied by a lidar simulation environment for design verification
- An adapted RRT path planner, encapsulated in a networked interface for distributed computing, that can be extended with little effort for various vehicle types
- A vehicle steering controller and velocity controller state machine, implemented on the on-board controller

- Simple mission planner that orchestrates data transfer between modules

1.6 Overview

The thesis starts with this introduction, which gives an overview of autonomous navigation and the project scope. In Chapter 2 the design of the demonstration vehicle is discussed. Vehicle selection, electrical actuation, feedback sensors, the control electronics and safety systems are covered. The first part of Chapter 3 introduces common sensors that are used to detect and map the vehicle environment and motivates the use of a scanning lidar for this project. The second part investigates the characteristics of a scanning lidar and discusses sensor mounting options for the best mapping results. Chapter 4 covers the mapping of a three-dimensional environment using lidar. First common representations used to map environments are considered and occupancy grids are chosen (based on the works of [1, 2]) and expanded to three dimensions. The following two sections detail the implementation and simulation of a real-time three-dimensional mapping process for lidar data.

Chapter 5 develops a path planner for real-time navigation. General planning concepts used to represent the planning problem (mainly from [3]) are introduced and application-specific elements are dicussed. A sampling-based path planner is developed by adapting the rapidly exploring dense tree algorithm to the specific planning problem, focusing on efficient real-time execution (based on work by [3–6]). The path planner is then tested in simulation. In Chapter 6 the vehicle controller is designed that should execute the paths from the path planner (including work from [7]). The required state estimator sensors are also noted. Chapter 7 describes the integration of the developed modules, the procedure used for testing and the test results obtained from four practical tests.

The conclusion provided in Chapter 8 reflects on the achievements of this thesis as an overall supplement to the summaries of the individual chapters. Recommendations for immediate improvements and future work are also provided.

Chapter 2

Vehicle Design

To support and verify the research of the AutoNav group, a physical test vehicle is required. This vehicle will be used to demonstrate the results of this project. It may also be used as a test bed for future research. With the navigational architecture in place, it should be simple to replace any module with a revised or experimental version.

This chapter starts with considerations taken into account during the choice of vehicle. Next a description of the methodology used to electrically actuate the chosen vehicle is provided, as are the sensors added to monitor the vehicle. It concludes with a description of the vehicle-specific controller board that was designed.

2.1 Selection

In order for the vehicle to be a proper research tool, it should not be limiting in capabilities. For the selection of vehicle, one should consider the following:

- Actuation should be simple for reliable and robust operation
- The constraints of the vehicle will become the constraints of the mission, route and path planners, such as:
  - Velocity and acceleration
  - Turning circle
  - Modes of locomotion e.g. forward and reverse capabilities

  - Physical robustness over unpredictable terrain

- Budgetary constraints

For this purposes, an all terrain vehicle (ATV), colloquially known as a "quad bike", was chosen. The chosen model is a Kymco MXU 150 utility vehicle. Its key features include:

- 150cc Petrol which engine delivers adequate driving power
- Automatic continuously variable transmission (CVT) drivetrain which is easily actuated
- Front and rear utility racks which provides convenient mounting space
- Alternator and battery system which provides convenient electrical power
- Mechanical robustness

Figure 2.1: Picture of the ATV chosen for the demonstration vehicle

2.2 Servo Actuation

Actuation forces are delivered by geared DC electric motors with suitable gear ratios and linkages to the stock vehicle mechanics. For safety and convenience, all controls are still accessible via the original levers. This makes it possible for a test driver to override the controller outputs, either to set up a test or to take over in the event of a failure. The controls that are actuated are: steering, brakes, gear lever and throttle.

2.2.1 Actuator Selection

Electric DC motors are used to drive vehicle controls. Using electric actuators is convenient as the vehicle already runs a 12 V power circuit and needs no additional systems, which would be the case with hydraulic or pneumatic actuators. The electric power is sourced from the stock alternator, supplemented by a 12 Volt lead-acid car battery to buffer peak demand.

To select the DC motors, the required action forces and motions were characterised and translated into electric specifications. The actuation system should be able to apply the peak force required to move a control and deliver enough power to complete the motion in the required time.

Linear forces were measured by attaching a force scale to the control and steadily increasing the force until the control completed the action. The peak force required was recorded. In order to measure torque, a crank lever was used and the applied force measured with the force scale. The range of motion is measured and the total energy required to complete the action is computed using equations 2.2.1 and 2.2.2.

$$\text{Linear: } E = \Delta s \cdot F \tag{2.2.1}$$

$$\text{Rotary: } E = \Delta\theta \cdot \tau, \tag{2.2.2}$$

with the assumption that the force/torque is constant over the motion and $\Delta s/\Delta\theta$ is the range of motion. Using the peak value of force, the calculated energy represents an upper limit. For motions where the required force changes significantly, the energy may be found by integrating force over displacement. The measured values are shown in Table 2.1.

It is important to note that measured forces may be dependent on the operating conditions. Optimal designs can be made for the characteristic conditions, but designs should also be analysed for worst-case conditions to quantify the amount of performance degradation. If the degradation is unacceptable, the design should be adapted. As an example, the steering force depends on the friction between the wheels and the driving surface and whether or not the wheels are rotating. In this case the characteristic force was measured while slowly driving the vehicle over typical tarmac, but the design was verified to ensure that the motor can produce the torque required to steer when the vehicle is stationary.

The time specifications for actuation motion were chosen by the designer to match the purpose of control and determine the minimum output power requirements of the actuators. Start-up time was neglected, as this is small with respect to the total time. The chosen timing is shown in Table 2.1. An efficiency factor, η, is included to indicate the losses incurred by gearing and linkages between the motors and the respective controls. The motor output power is thus given by:

$$P_o = \frac{1}{\eta} \cdot \frac{E}{\Delta t} \tag{2.2.3}$$

Table 2.1: Actuator forces and time specifications

Control [Unit]	Force F[N]	Range Δs[m]	Energy E[J]	Time Δt[s]	Efficiency η	Output Power P_o[W]
Steering	85	0.65	55	2	0.75	37
Brake Cable	50-500	0.02	5.5	0.35	0.75	16
Gear Lever	160	2×0.025	2×4	2×1	1.0[a]	4
Throttle	5-25	0.025	0.4	0.2	1.0[b]	2

[a]Unity efficiency assumed for non time-critical control.
[b]Unity efficiency assumed for connection with negligible friction.

The DC motor model should be chosen according to output power specifications and convenient form factor. As the gearbox ratio influences the efficiency, this may be an iterative process. The required gear ratio (n : 1) to match the motor to the control can be inferred from the motor output torque τ_o, the gearbox efficiency η and the required torque τ_c(or force F_c and moment arm l_c) of the applicable control:

$$\mathrm{n} = \frac{1}{\eta} \cdot \frac{\tau_o}{\tau_c} = \frac{1}{\eta} \cdot \frac{\tau_o}{F_c \cdot l_c} \ , \tag{2.2.4}$$

where τ_o should be the output torque of the motor at the specified output power, P_o, with due regard for maximum ratings. The gearbox efficiency may be deliberately decreased to add a factor of safety to the calculations.

2.2.2 Electrical Drives

Controlling electrical motors is achieved with relevant power electronic circuits. The throttle actuator servo has a built-in power circuit and accepts positions command via a pulse-coded signal. The remaining three electric motors are simple automotive parts and require 12 V H-bridge drive circuits.

The H-bridge driver's current specifications were determined from the motor specifications and output torque requirements. The electric motor constant, k, of each motor was empirically characterised. Using the relationship of current to torque for DC motors,

$$\tau = k \cdot I, \tag{2.2.5}$$

the peak currents were calculated. The driver-circuits output is controlled with a single ended pulse width modulation (PWM) signal (approximately 20 kHz, biased at 50% duty cycle), has an $\overline{\texttt{enable}}$ line and returns a bi-directional current sensing signal. For added safety, the 12 V power supply to the motor drivers (as with all other circuits) has been appropriately protected with fuses.

An energy budget ensures that there is enough electrical power available to power all the output circuits. The alternator may charge the battery at its internally-governed rated current and the actuators will draw current at varying operating cycles. For continuous operation, the sum of average current drawn should not exceed the charging current. An estimated energy budget is shown in Table 2.2 with the anticipated currents and operating cycles. The estimate shows that there is approximately 1 A surplus current available to power additional circuits. The sum of the active currents is the peak current that the battery should be able to buffer and is used to select the battery.

Table 2.2: Example energy budget for currents drawn from the battery

Circuit	Active Current I[A]	Operating Cycle Duty [%]	Average Current I_{avg}[A]
Alternator	-8	95	-7.6
Controller	2.5	100	2.5
Steering	15	15	2.25
Brakes	10	5	0.5
Gears	5	2	0.1
Throttle	0.75	50	0.375
Total	25.25		-0.925 [a]

[a] A negative value of *current drawn* indicates a surplus of energy.

2.2.3 Linking Actuators to Controls

In order to transfer the energy from the actuator to the control, the systems has to be physically connected. Instead of redoing the stock mechanical levers on the vehicle, the

actuators were designed to be additional to the original levers. This requires the least amount of work and leaves the vehicle essentially intact.

Using simple mechanisms with the fewest linkage reduces friction which improves the efficiency of energy transfer. As a practical advantage, simple systems are easily serviced and finding replacement components for damaged parts are easier if they are "common" parts. See Appendix E for photographs of the designs described below.

The steering shaft of is accessible from the front of the vehicle. The chosen motor with worm gear assembly is mounted with its output shaft parallel to the steering shaft and connected with a chain and a pair of cogs (adapted bicycle parts). The relative size of these cogs were used to effect the exact gear ratio.

The ATV front and rear wheel brakes can be applied separately. The rear wheels have a disc brake, known for precise and predictable braking and are the braking system of choice. The cable was removed from the brake lever and rerouted to its actuator motor. The chosen motor has an integrated worm gear assembly and the output torque is sufficient to directly tension the cable using a short (30 mm) crank lever, since it requires very little motion (see Table 2.1).

Similarly, the throttle cable was rerouted to its actuator motor. Again the motor also has a built-in gearbox with sufficient torque to directly pull the cable. From the actuator, the cable was extended to the original throttle lever, so that the test driver can operate the vehicle when the controller is disabled.

The gear lever has three slots, namely Forward, Neutral and Reverse. The slots were removed so that it may slide directly from one setting to the next. The chosen motor has an integrated worm gear with enough output torque such that a simple crank-and-lever connection to the gear lever is sufficient to push/pull the gear lever to the required setting. The moment arm of the lever is calculated to deliver the required force over the required distance.

2.2.4 Safeguarding

The experimental nature of the vehicle makes it susceptible to occasional malfunction. A runaway vehicle can be dangerous and indicates the need for an emergency stopping

mechanism. In case of a malfunction, the ability to override the steering and stop the vehicle is sufficient. Table 2.3 lists the actuated controls and the physical safeguarding applicable.

Table 2.3: Safeguards for vehicle actuators

Control	Safeguard
Steering	Shear pin fails when excessive force is applied
Brake	Alternative method of braking – stock on ATV
Gear Lever	Engine cuts out on accidental gear change – stock on ATV
Throttle	Original throttle lever operates in unison with actuator; Installed an emergency cut-out switch on the ignition

To allow a test driver to override the steering actuator, a shear pin is used to transfer the torque from the cog to the steering shaft. The shearing force of the pin is calculated from the maximum torque needed for the steering under normal usage and the moment arm over which the pin acts. The dimensions of the pin are calculated from the shear pressure associated with the material, so that it will give way when overloaded. It is noted that a shear pin is a rudimentary solution and could be replaced by a more sophisticated system.

In terms of thrust, the test driver can only add additional braking or additional throttle. In the event of a malfunction, the test driver can add additional braking and stop the engine via the emergency cut-out switch.

To protect the vehicle gearbox, the lowest level controller checks that the vehicle is stationary and that brakes are applied (via a monitoring circuit on the brake light), before allowing a gear shift operation. The ATV circuitry also cuts out the ignition if the gear lever leaves neutral whilst the brakes are not active.

2.3 Sensors

To facilitate closed-loop control, all actuator motors are equipped with position feedback potentiometers. These are sampled and digitised by the vehicle controller board. The custom motor drivers include bi-directional current sensing, which is sampled by the controller and implemented as hiccup-mode over-current protection. These may also be used as torque feedback, as per equation 2.2.5.

To facilitate dead reckoning, it is required to measure forward displacement of the vehicle. A hall-effect rotation encoder connected to the transmission output provides accurate encoding of the distance travelled via equation 2.3.1. To obtain velocity, the controller simply computes the discreet derivative.

$$\text{displacement} = 2\pi(\text{wheel radius})(\text{drivetrain ratio}) \times \frac{\text{counts}}{\text{counts per revolution}} \tag{2.3.1}$$

Engine speed (RPM) is measured via a non-invasive tap on the electronic ignition system. By timing the period between successive current pulses delivered to the spark plug every revolution, the RPM is calculated using

$$\text{RPM} = \frac{60\,\text{s}}{1\,\text{min}} \times \frac{1}{T_{\text{revolution}}}. \tag{2.3.2}$$

To determine whether the clutch is engaged as well as the current ratio of the CVT drivetrain, the ratio of RPM to wheel speed is calculated. At higher speeds, this ratio should be factored into the velocity feedback controller as it changes the dynamics of the vehicle. If the ratio is outside the bounds achievable by the CVT, it may be deduced that the centrifugal clutch is disengaged and the appropriate response can be made.

Monitoring circuits have also been added to measure actuator battery voltage level and determine whether or not the test driver has activated the brakes. Some additional inputs and outputs are available to allow future expansion, such as remotely starting and stopping the engine.

2.4 Actuator Controller

To modularise the architecture, the vehicle is equipped with a vehicle specific controller board (shown in Figure 2.2). It presents a general interface to higher level controllers on a standard controller area network (CAN-bus). Furthermore, it samples and converts the sensor data, monitors vehicle status, such as battery level and test driver interference, and enforces actuator limitations to prevent accidental damage.

The inner-loop controller takes servo-like position inputs for actuators and generates the required control signals for the motor drivers. It accepts logic-based control signals for

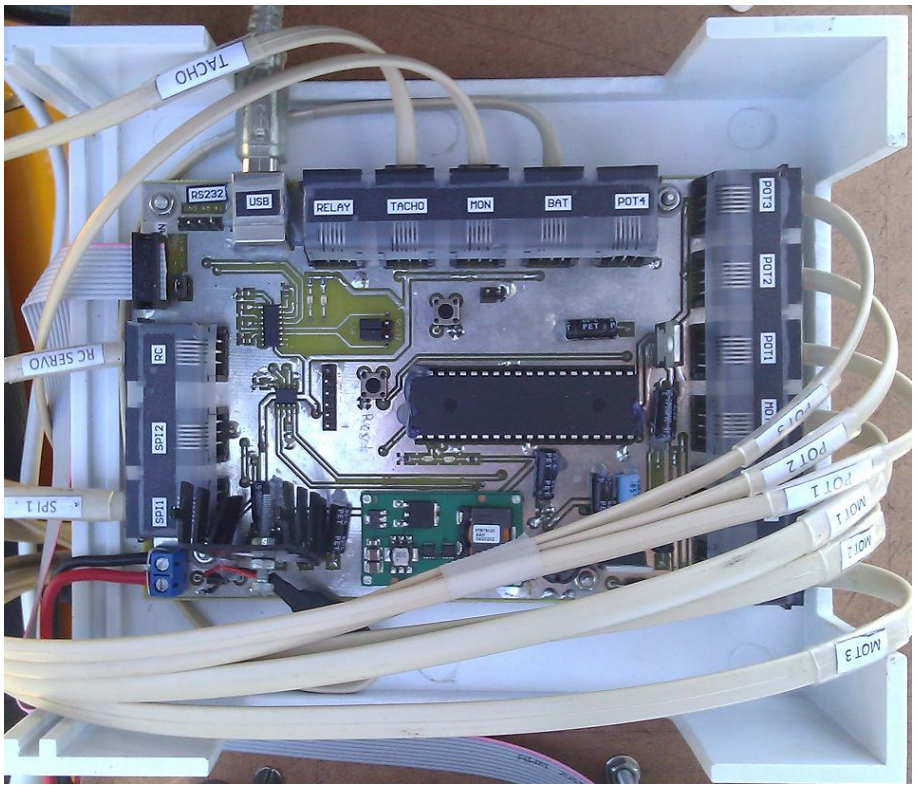


Figure 2.2: Photo of the actuator controller board, showing the input and output cabling.

gear shifts. The respective controllers reject any commands outside of the operating range of their actuators and raise error flags. The true actuator position is returned to the high-level controller, such that it may be used for monitoring or estimation algorithms.

The CAN-bus network has been implemented to synchronise with a high-level controller that polls at a 50 Hz[1] rate, using a digital phase-locked loop (DPLL). This ensures that returned sensor data is sampled at a precise 20 ms sample period, irrespective of minor congestion on the CAN-bus which influences polling frequency. A status flag is set when the frequency and the phase are synchronised.

A serial interface (also avaiblable via USB) provides additional debugging functionality via a PC user interface (shown in Appendix A). From the interface the actuators can be directly commanded and position controllers tested with position reference inputs. Sensor data can be be plotted and logged to investigate responses. The controller also keeps a limited message/error log that can be downloaded.

[1]The use of a 50 Hz polling rate is dictated by the CAN-bus interface of existing hardware systems that are available within the AutoNav group. This rate has proven to be sufficiently high to control various vehicles with dynamics that are faster than that of the ATV and is thus suitable for this application.

2.5 Vehicle Summary

The result of this section is a practical demonstrator vehicle. It has been converted to be fully drive-by-wire (steering, throttle, brakes, gear shift), whilst maintaining test driver input capabilities. Mechanical safeguarding as well as an emergency cut-out switch were added. The controller provides actuator, wheel odometry and engine tachometer sensor feedback to higher level controllers for estimation, as well as some additional monitoring services. The vehicle controller accepts position reference input commands for actuators to separate actuator-specific dynamics from the higher level control.

Chapter 3

Environment Sensor

To navigate in real time, the vehicle needs information about the layout of the operating environment. If this information is not completely known at the start of the mission or can change over time, the vehicle will have to explore the environment and update its knowledge through environmental sensors.

This chapter briefly considers different types of sensors which may be used to map the environment. After selecting a scanning lidar, ways to use this type of sensor to its full potential are investigated.

3.1 Smart sensors

For the vehicle to be independent of external data sources, all environmental sensors need to be carried on the vehicle. This implies that measurements are made relative to the vehicle's state and are only meaningful if interpreted as such.

To modularise sensors, the vehicle state is fed to the sensor system, so that it may internally pre-process the raw data with respect to the vehicle state and thus may hide the underlying implementation. This type of sensor is referred to as a *smart sensor*, to separate it from sensors which do not perform internal state-relative processing.

3.2 Sensor Selection

There are a number of sensor technologies capable of detecting surroundings. The sensors are most useful if they can provide information about the location and shape of a present object, so that they may be included in a map. If the environment is not constrained to be static, additional information may be necessary, but is beyond the scope of this project. Known sensors that provide shape/location data are listed, described and then compared below.

- Radar (radio detection and ranging)
- Sonar (sound navigation and ranging)
- Lidar (light detection and ranging)
- Computer vision

3.2.1 Radar

Radar emits electromagnetic (radio) waves and receives reflections off objects. The time-of-flight (round-trip time of transmitted wave to reflect and return) is used to calculate the distance to the object, given the speed of electromagnetic propagation. To have the relative direction of the detected object, one has to focus the radiated energy into a beam which can be steered in the direction of interest.

By choosing the proper frequency band, radar can be designed to operate very accurately over a couple of meters or to detect targets tens or hundreds of kilometres away. Unfortunately, forming a narrow beam with high angular resolution requires large antennae, larger than what can be fitted to a vehicle.

Doppler radars can distinguish moving objects from stationary ones with high precision. This enables the reliable detection of dynamic objects against a stationary background, which greatly reduces the need for a high angular resolution.

3.2.2 Sonar

Sonar emits sound waves and receives echoes off objects. The methods used to process the echoes are similar to that of radar in Section 3.2.1, but because of the lower bandwidth, is much more affordable on a portable scale.

The propagation of sound is heavily distorted by atmospheric disturbances and is therefore not reliable over longer distances. Also the the slow speed of sound (relative to radio waves) limits its applications to low speed or short range use.

3.2.3 Lidar

Lidar emits (laser) light, which is a particular portion of the electromagnetic spectrum and therefore very similar to radar. It also uses the time-of-flight of the reflected signal to determine distance. Since laser light can be focused into an almost arbitrarily narrow beam, the angular resolution of a measurement may be selected. Lidar operation is discussed in detail in Section 3.3.1 and is therefore not duplicated here.

3.2.4 Computer Vision

Computer vision is a system whereby the images from a digital camera system are processed by a computer to extract the required information, commonly to mimic animal-like image recognition. Common techniques include stereoscopic cameras, optical flow and texture sectioning.

Computer vision shows great promise for use in autonomous navigation and SLAM. Current techniques, however, require high computational loads and advanced processing and implementation is therefore beyond the scope of this thesis. Computer vision is indeed a subject of research for the AutoNav research group and will be added to the vehicle by follow-up researchers.

3.2.5 Comparison

In Table 3.1 the sensors types are compared. Ideally a vehicle should combine many sensor types to exploit the specific strengths of each sensor. Sonar and radar can be used

Table 3.1: Comparison of sensors that provide distance and angular information of objects as a means to map the environment.

Attribute	Radar	Sonar	Lidar	Computer Vision
Distance resolution	Excellent	Good	Excellent	Good
Angular resolution	Fair	Poor	Excellent	Good
Application strength	Large or moving targets	All short range obstacles	Profile scanning	Sectioning
Application range	10 . . . 200 m	1 . . . 5 m	1 . . . 50 m	0 . . .infinity
Cost	High	Low	Mid	Low

to effectively identify obstacles, respectively of short and long range [7]. Their lack of fine angular resolution makes it unsuitably to map the profile of a road surface.

As a midrange sensor, the angular resolution of lidar makes it possible to map the profile of a road surface, with obstacles protruding from it. The accuracy is limited mainly by the uncertainty of the sensor pose estimation. Computer vision hopes to provide the entire spectrum of short and long range measurements, but is not yet available. As such, a scanning lidar was chosen as primary environmental mapping sensor.

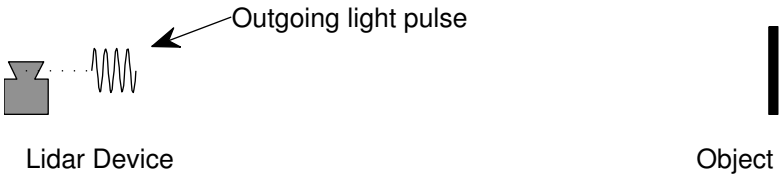
3.3 Lidar Mapping

The most widely used type of lidar device is a planar scanning sensor [7, 8], effectively delivering two-dimensional data at high update rates (> 25Hz). Three-dimensional lidar units are available, such as the Microsoft Kinect and MESA SwissRanger^{TM} SR4000, but those capable of outdoor (sunlit) operation with usable range are not yet affordable for practical use. As such, the remainder of this chapter will be devoted to the two-dimensional scanning lidar.

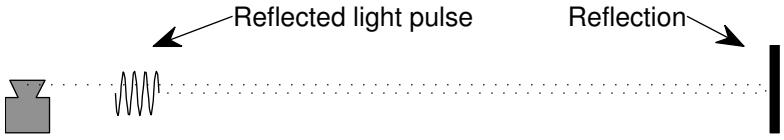
Firstly the capabilities and processing considerations of lidar data will be discussed, then the influence of mounting on the usability of the retrieved data will be considered. In Chapter 4 we define mapping techniques and verify these techniques using simulation.

3.3.1 2D Lidar Capabilities

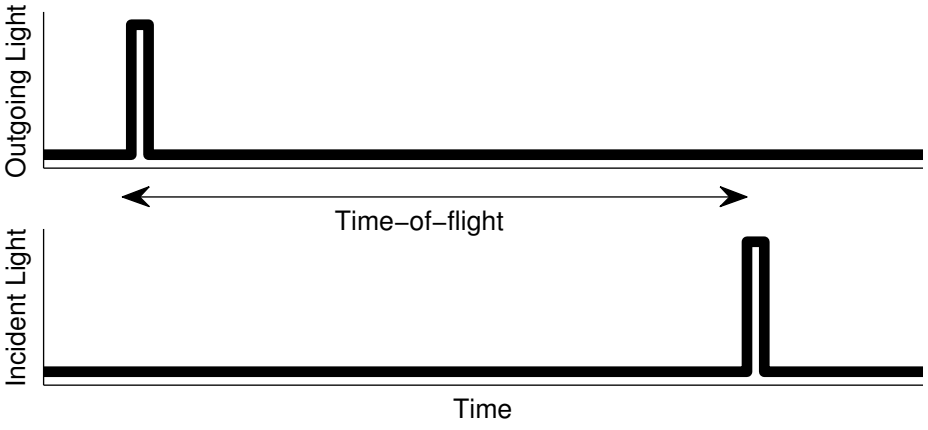
To understand mapping with a lidar, the operation and capabilities of a typical two-dimensional scanning lidar (adapted from [9]) are discussed. The device outputs a modulated pulse of focused laser light (see Figure 3.1a). A receiver tuned to the same wavelength of light detects and digitises the incident light intensity (Figure 3.1b). If an object is present in the outgoing beam path, some portion of light is reflected from its surface back towards the device. Since light has a finite, known speed of propagation, the incident reflection will be delayed with respect to the outgoing pulse (Figure 3.1c). The length of the delay represents twice the distance between the device and the reflecting surface.



(a) Device generates outgoing pulse of light



(b) Surface reflection causes an incident pulse of light



(c) Time scale showing round trip time of incident pulse

Figure 3.1: Illustration of operating principle of a 2D scanning lidar.

By rotating the beam and taking measurements at convenient increments in rotation, a disc-shaped area about the device can be scanned. This is typically achieved by using a rotating mirror inside the device [9]. If the rotational increment (angular resolution) is matched to the beam spread, an uninterrupted band can be scanned (see Figure 3.2).

The acquired data is represented by an array of distances coupled to their rotation index. A profile of the scan may be constructed in polar coordinates, with origin at the device centre.

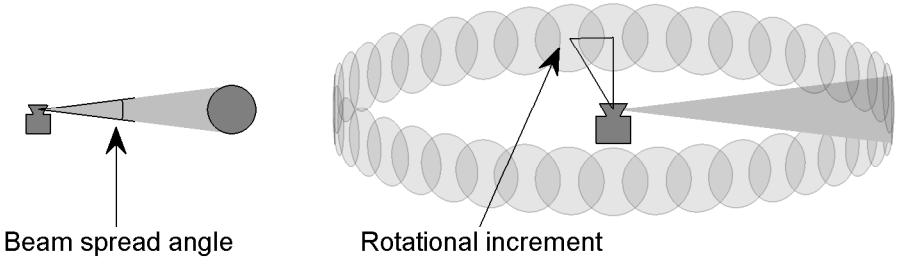


Figure 3.2: Illustration of scanning lidar beam spread and angular resolution.

Some important considerations include:

- A closer surface may obstruct the view of a surface behind it.
- A lidar detects *only* a surface, not the volume of an object.
- A physical device will have limited detection range.
- A surface with low remission values may not reflect a sufficient amount of light to be detectable.

A single lidar detection has three regions to be interpreted, as depicted by Figure 3.3. Firstly, the space between the sensor and the detected surface are known to be clear. Secondly, there is a known surface at the detected distance, within the measurement uncertainty of the device. And lastly, the area behind the surface is unknown, since its view is blocked.

3.3.2 Lidar Mounting

The position and orientation that a scanning sensor is mounted on a vehicle greatly influences the viewable environment profiles. Apart from static mounting, a scanner may also be actuated using a pan/tilt unit (PTU) to direct the sensor to view a different area. We discuss some of the possibilities and motivate our selection.

Static horizontal (bumper) mounting is the simplest method of mounting, as depicted in Figure 3.4a. It is very effective in structured environments where the obstacles

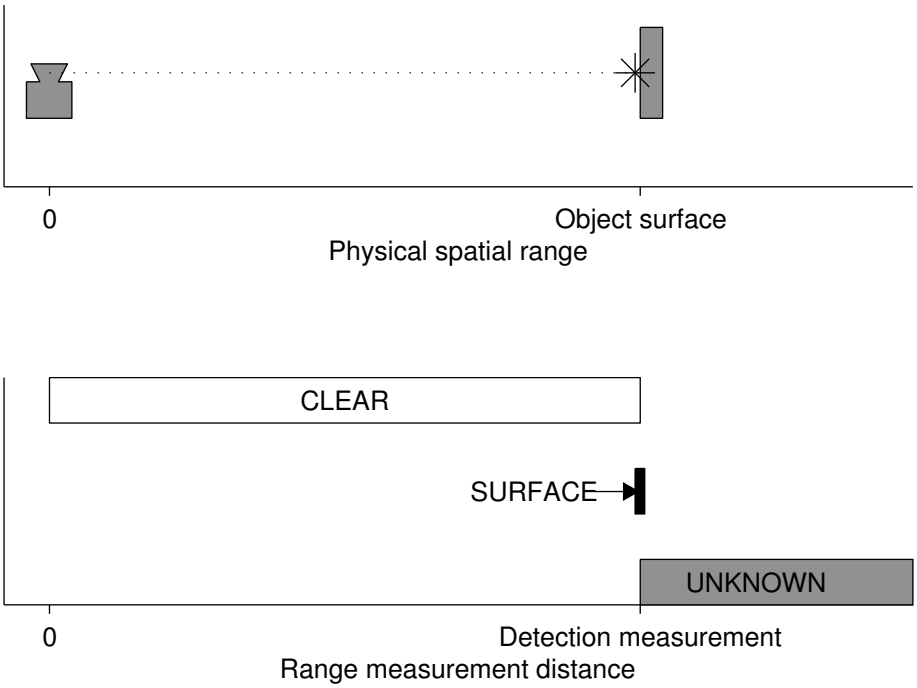
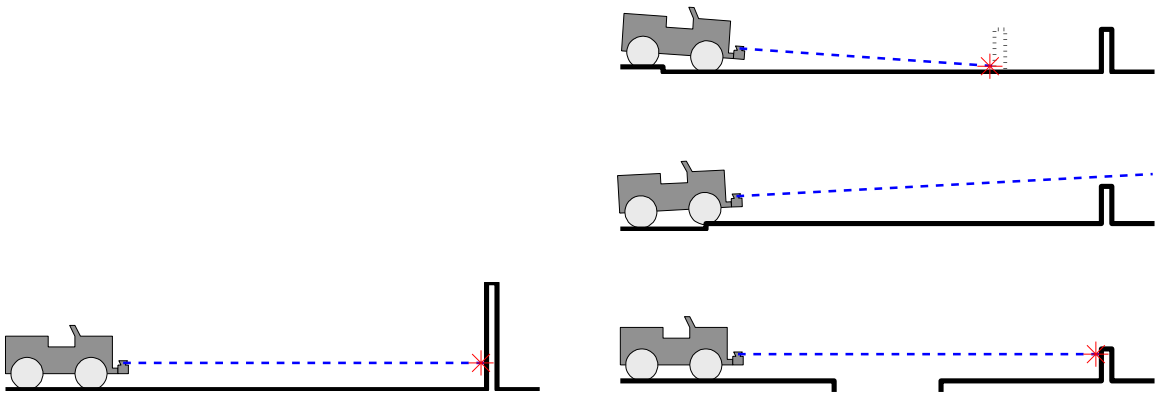


Figure 3.3: Illustration depicting the three regions of interpretation for a lidar detection.

are known to protrude significantly higher than the sensor mount height. All detections indicate obstacles and non-detections indicate clear area. It is unfortunately very sensitive to pitch and yaw motions of the vehicle (Figure 3.4b). A small angular change is amplified over distance. Pitching upwards may overlook a short obstacle and pitching downwards may detect the driving surface as a phantom obstacle. There is also no way of asserting the presence of a driving surface (Figure 3.4b bottom)and is therefore unacceptably unreliable.



(a) Effective detection in a structured environment (b) Erroneous conditions due to pitch and lack of surface scanning

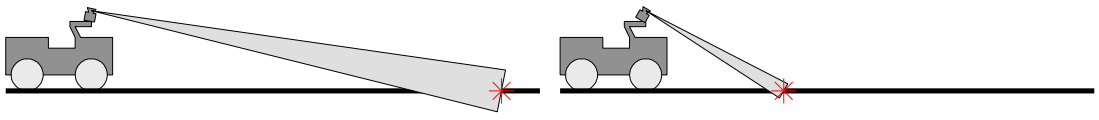
Figure 3.4: Illustration of mounting lidar horizontally on a bumper.

If one were to compensate for pitch and roll motions and include scanning of the driving surface, it implies building a complete 3D representation of the environment. There is no longer a need to rely on any guarantee about the "structuredness" of the environment. When using sensor pose data, it becomes essential to include the statistical uncertainty of the vehicle estimator in the measurement interpretation.

Having the sensor deliberately **mounted pitched downwards** ensures profile slices being taken of the driving surface as per Figure 3.5. The particular slice will still be affected by vehicle roll and pitch, but pitch and roll is measured by the vehicle estimator for mapping and is not a cause of error. The main source of error will be the limited accuracy of the vehicle estimator [7]. Again, the effect of angular error is amplified over distance.

There arises a trade-off between the accuracy of our map and our ability to look ahead towards the horizon, given that the angular estimation is fixed. Looking further ahead enhances our ability to plan ahead, but the detail is lost in estimation error (Figure 3.6a) and visa versa (Figure 3.6b. One way to avoid striking a compromise is to use multiple lidar devices and set them up for different viewing distances as in [7] (five lidar scanners), but this a costly solution. It is desirable to achieve usable results using only a single lidar device.

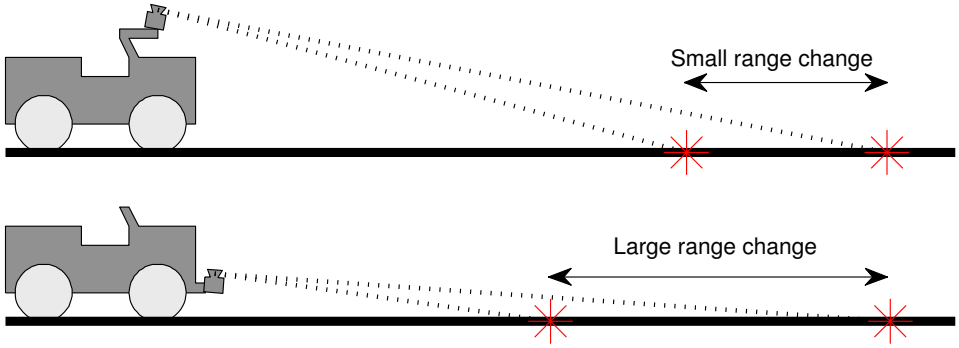
Figure 3.5: A downwards pitched scanner taking profile slices of the driving surface.



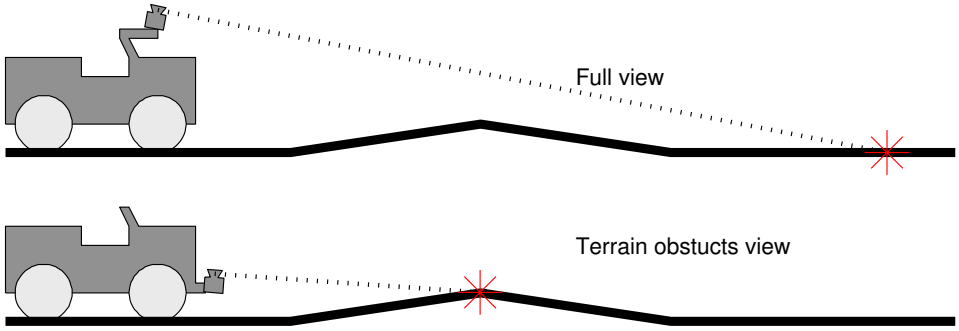
(a) Long look ahead mount (b) High accuracy mount

Figure 3.6: Illustration of mounting a lidar at different downward pitch angles, with measurement uncertainty indicated.

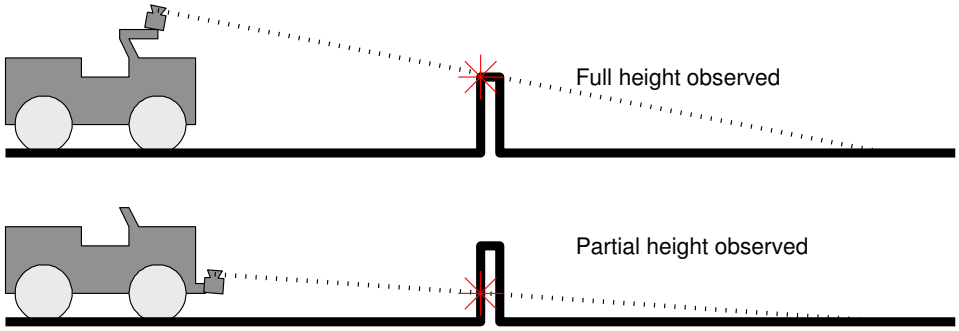
When using a fixed mount, mounting the lidar device in a elevated position has several advantages as portrayed in Figure 3.7. Firstly, a higher mounting position desensitises the viewing distance from unwanted pitch and roll motions (Figure 3.7a). Secondly, it makes it less likely that driving surface deformation will intervene with viewing distance (Figure 3.7b). And most importantly, it makes it possible to discern height as the vehicle nears an obstacle (Figure 3.7c).



(a) Range sensitivity to equal pitch changes



(b) View obstruction by equal terrain



(c) Height discretion at equal range

Figure 3.7: Illustrations of comparison between a fixed pitch roof mounted and fixed pitch bumper mounted lidar.

It is possible to control the lidar orientation via a **PTU mount**. This gives us the option to switch, in real time, between the high accuracy of short distance scanning and the

longer distance of reduced accuracy scanning. Similar results are obtained as with using multiple devices, but with the limitation of having the sensing bandwidth of a single device. At the same time, having lower sensor bandwidth may be a benefit to processing requirements.

Various different mounting and actuation strategies may be used, each with different effects. This includes the ability to direct the sensor to scan areas previously missed or where greater accuracy is required. It could also be used to stabilise unwanted pitch and roll motions incurred by vehicle motions.

3.4 Sensor Summary

Of the various types of sensors, a scanning lidar was chosen based on its excellent range and angular resolution. The scanning lidar can effectively detect obstacles and assert the presence of a driving surface. Ideally the lidar should be used in conjunction with the long range capabilities of radar and computer vision.

The choice was made to mount the lidar at a fixed downward pitch, as this facilitates a profile scan of the driving surface. The precise pitch angle is a trade-off between the desired accuracy and the viewing distance. Bumper mounting a lidar provides good obstacle detection in a structured environment, but its shortcomings are unacceptable for unstructured environments. It has been shown that mounting the device high on the vehicle has some desirable effects. For practical construction reasons, the lidar will be approximately 1.2 m above the driving surface.

It was noted that using multiple lidar devices or actively aiming the device(s) using a PTU may provide considerable improvements, but is beyond the scope of this thesis.

Chapter 4

Mapping

It is necessary to reconstruct the environment from the acquired lidar data and store it in a map which is usable for path planning. One way to achieve this, would be to store all scanned profiles with their associated sensor pose information. Then, for every planning query, all the data would have to be processed and checked for conflicts. This could require infinite memory and processing, as the amount of gathered data would continually increase over time, which is not a feasible solution.

The practical approach is to process the incoming data in real time, extract the required information and store only that information in a map-like structure. Instead of storing an increasing amount of data over time, the map data is updated as new information becomes available.

For the reasons explained in Section 3.3, a full 3D mapping setup was opted for in order to overcome the limitations of a 2D setup. Ultimately we want to be able to profile the driving surface and distinguish obstacles in a manner suitable for path planning. In this chapter known approaches to represent an environment are considered and a detailed discussion of the implementation of the chosen 3D occupancy grid representation is given. The final section describes the lidar simulation setup that was used as a development tool.

4.1 Map Representation

To model the 3D environment in which operation is desired, a representation of the real world is needed that is both efficient to update and efficient to interpret. This section

describes some known methods, working up towards three-dimensional occupancy grids.

4.1.1 Triangular Mesh Representation

As sensors mostly detect surfaces, it follows to represent the environment as a collection of surfaces. From computer visualisation we know that a practical method to discretise an arbitrary surface is to approximate the surface with a triangular mesh [3] (Figure 4.1). In areas of high detail the mesh is refined and where there is little detail only a few vertices are needed. This saves on storage space and computational expense [10].

Although a meshed representation is very general, updating it and using it as input to a path planner can become very involved. A simple and convenient method is preferred and therefore this topic is not pursued in this thesis.

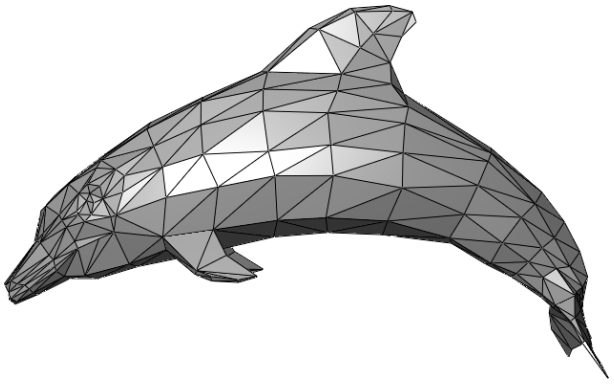


Figure 4.1: Example of a triangular meshed surface

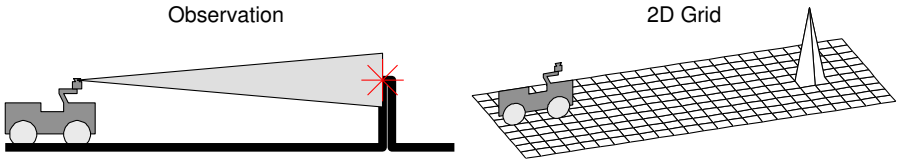
4.1.2 Grid-based Representation

Since the vehicle is to be bound to a driving surface, the representation in this thesis may be constrained to a relatively flat surface. A surface with no discontinuities can be approximated by a regular grid of heights, stored in a two-dimensional matrix. Heights in the area between stored vertices may be interpolated as required.

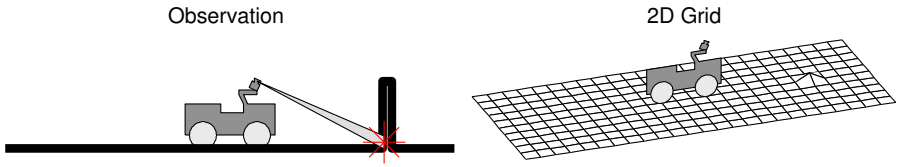
This conveniently suits the desire to represent a smooth driving surface. Intuitively, if a vertex within the grid protrudes significantly in height from its neighbours, it may be considered an obstacle.

As the sensors gather information about the environment, the relevant vertices in the grid are updated with their heights. To allow for measurement uncertainty, the perceived accuracy of the height estimation may be stored with the height. Any system of update that accommodates uncertainty may be used, such as the popular Kalman filter.

There is, however, one pitfall that may arise when we sense a vertical surface. Consider a pole protruding from the ground. If an observation of the pole were made near its top (see Figure 4.2a), it may correctly be identified as an obstacle. If at a later time, many new observations are made at the base of the pole (Figure 4.2b), with higher accuracy, such that the vertices are updated with the height of the base observation, the process may mistakenly erase the pole from the map.



(a) First observation recognises object



(b) Second observation mistakenly decreases height of object

Figure 4.2: Illustration of gridbased height profiling of a pole.

In the implementation of Probabilistic Terrain Analysis (PTA) gridbased maps in [11], they overcome this dilemma by maintaining two heights per cell – the lowest and the highest observed points. With this added information, they define three possible states for a cell.

A cell is unknown if it does not contain both a minimum and a maximum height i.e. at least two observations. A cell is drivable if the difference in height between minimum and maximum does not exceed some defined threshold (uncertainty accounted for via a Markov model). This implies a level surface. If the difference in height does exceed the threshold, then the surface is assumed discontinuous and marked undrivable.

The PTA method was implemented successfully, but the authors explicitly note that the PTA method does not reconstruct a 3D model [11]. Instead, it uses a statistical test on a per-cell basis to determine whether a cell is drivable.

It is noted that these surface-mapping techniques only use the surface data returned by the lidar device. In Section 3.3.1 we state that lidar data can also observe a clear space, which remains unused when mapping surfaces. This surface mapping assumes that,

1. if an obstacle-like surface exists, it will be observed in two distinct heights and
2. the absence of such distinct observations is sufficient to declare a cell drivable.

These conditions cannot be guaranteed, nor does the method of [11] indicate a means to determine whether a cell has been sufficiently probed to be obstacle free. The successful use of the PTA method in [11] was aided by the use of five lidar devices, increasing the probability of observations. As this project only uses a single device, this is not a feasible solution.

4.1.3 2D Occupancy Grid Representation

A popular method for classifying a 2D terrain for drivable and undrivable areas, is the occupancy grid (OG) [1]. It divides the map into a spatial lattice and maintains a stochastic estimate of the occupancy state of each cell [1]. An example OG from [2] is shown in Figure 4.3. When planning a path, the vehicle should not enter an occupied cell.

Bayesian estimation procedures allow the incremental updating of the OG using readings from several sensors over multiple points of view [1]. In keeping with the terminology of [1], each cell ,C, is described by a discrete random variable, $s(C)$, with two states, *occupied* and *empty*, denoted by OCC and EMP respectively. It is further said that the two states are exclusive and exhaustive, such that $P[s(C) = \text{OCC}] + P[s(C) = \text{EMP}] = 1$.

They define a probability based sensor model for a range data sensor, $p(r \mid z)$ which relates the distribution of reading r to the true spatial range z. It may easily be updated to include uncertainty in angular data, by expanding it to $p(r \mid z, \theta)$. To facilitate incremental updating, Bayes' theorem is used to estimate the cell occupancy state of cell C. Given

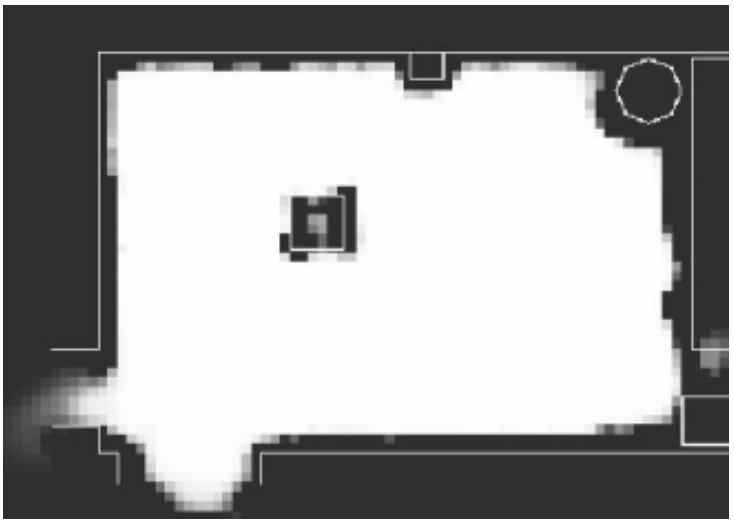


Figure 4.3: A 2D occupancy grid example, created by a sonar robot scanning a room. The white area indicates clear space and the grey unknown. The room and obstacles outline has been overlayed.

an estimate of the state of a cell,

$$P[s(C) = \text{OCC} \mid \{r\}_t]$$

$$\text{where } \{r\}_t = \{r_1, \ldots, r_t\}$$

and given a new observation, r_{t+1}, the improved estimate is given by

$$P[s(C) = \text{OCC} \mid \{r\}_{t+1}] = \frac{p\left[r_{t+1} \mid s(C) = \text{OCC}\right] \cdot P\left[s(C) = \text{OCC} \mid \{r\}_t\right]}{\sum_{s(C)} p\left[r_{t+1} \mid s(C)\right] \cdot P\left[s(C) \mid \{r\}_t\right]}. \quad (4.1.1)$$

The Bayesian formulation requires a prior, which in this case is the estimate of the cell for all previous readings, $P[s(C) = \text{OCC} \mid \{r\}_t]$. The updated cell estimate replaces the prior value and is stored with the cell. [1] obtains the distribution of sensor readings from the sensor model, $p[r_{t+1} \mid s(C)]$, using Kolmogoroff's theorem. The process from sensor reading to map is shown in Figure 4.4 (derived from [1]).

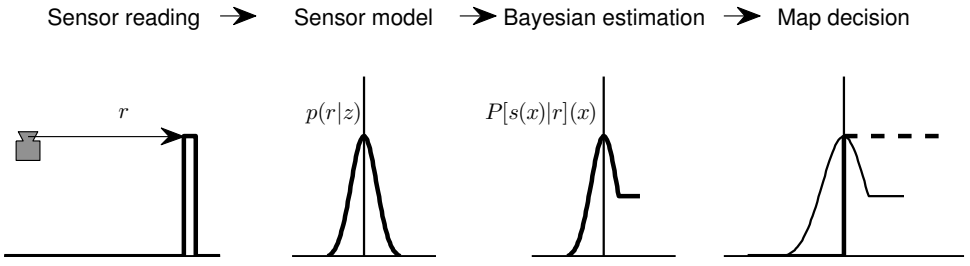


Figure 4.4: Diagram of estimating the occupancy grid from range sensor data

To distinguish occupied cells from free cells, [1] suggests using the maximum a posteriori decision rule, where a cell is:

- occupied if $P[s(C) = \text{OCC}] > P[s(C) = \text{EMP}]$,
- empty if $P[s(C) = \text{OCC}] < P[s(C) = \text{EMP}]$ and
- unknown if $P[s(C) = \text{OCC}] = P[s(C) = \text{EMP}]$.

The author of [1] also notes that the thresholding may be adjusted to include a band of unknown area, or that many tasks may be performed directly on the OG, without the need for explicit thresholding.

Occupancy grids are also a convenient method for combining different types of sensors. The different sensors may each update the same grid using their respective sensor models, or maintain separate grids, which can be overlayed for decision making.

It is further noted that OG makes explicit use of all three regions of a sensor, as per Section 3.3.1. This is desirable, as we would like to exploit all the available information.

The original method of OG, as presented here, makes use of the inverse sensor model to update the cells. As explained in [12], this inverse procedure creates some inconsistencies in the obtained map. The authors of [12] derive a method that uses only *forward* sensor models. Unfortunately, this is not an incremental approach. Incremental approaches are convenient for planning algorithms which need the map as it is discovered.

4.1.4 3D Occupancy Grid

Naturally the methods of 2D OGs may be extended to three dimensions. Instead of each cell representing a square of possible driving surface, each cell now represents a volume cube of operating space. As for the 2D case, the occupancy state of each cell is estimated. The driving surface can be thought of as a layer of perfectly stacked boxes and obstacles as stacked piles on top of the surface layer (Figure 4.5). All the remaining space is filled with ghost boxes.

A boundary between an empty and an occupied cell represents an object's surface. Extracting this boundary over the driving surface can be used to profile the driving surface, such as with the gridbased heights (Section 4.1.2).

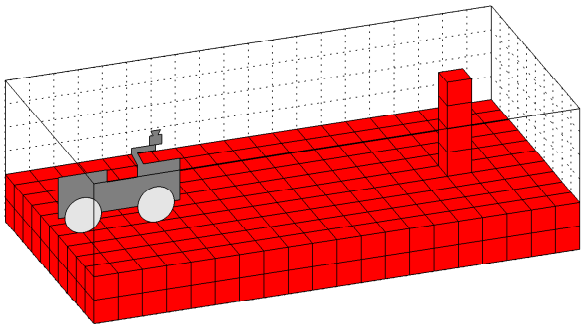


Figure 4.5: Example of 3D occupancy grid surface with obstacle. (Empty cells not drawn).

Fortunately, errors such as in Figure 4.2 can be excluded, since it is possible to check the vertical dimension for "hovering" occupied cells. Since cells can indicate an unknown state, it can be conveniently determided whether a cell has been observed or not, thus overcoming the assumptions of Section 4.1.2.

4.1.5 Representation Summary

Due to the unstructured nature of an outdoor operating environment, it is inevitable that the vehicle must profile the driving surface to account for (moderately) sloped surfaces. The assumptions of pure surface-based modelling poses problems and does not guarantee that all obstacles have been observed. As such, the OG representation was chosen, which overcomes these problems.

The 3D OG can model an arbitrary environment and is the method of choice for this thesis. It captures data from all three sensor regions, which enables the labelling of space as empty, occupied or unknown, which can be used to guarantee empty space. Bayes' theorem provides a statistically sound method to incrementally update the OG as data is gathered. If so desired, it would be possible to extract an approximated surface representation from the OG and add to it the aforementioned guarantee.

4.2 Implementation of the 3D Occupancy Grid

This section gives a detailed explanation of the implementation of 3D occupancy grids as used in this thesis. The goal is to make a practical mapping module, which could be

used as a real-time on-board system. There are a number of design parameters which are problem specific. The factors that influence the design choices are discussed and are used as a guide to selecting appropriate parameters for the sensor model, 3D OG resolution and mounting pitch.

4.2.1 Axis System

We define four axis systems, namely inertial axis, body axis, map axis and sensor axis, denoted by subscripts $_I$, $_B$, $_M$ and $_S$ respectively. Although any axis system may be transformed to any of the others and therefore seem redundant, it is used as a convenience to concisely express certain relative quantities [13].

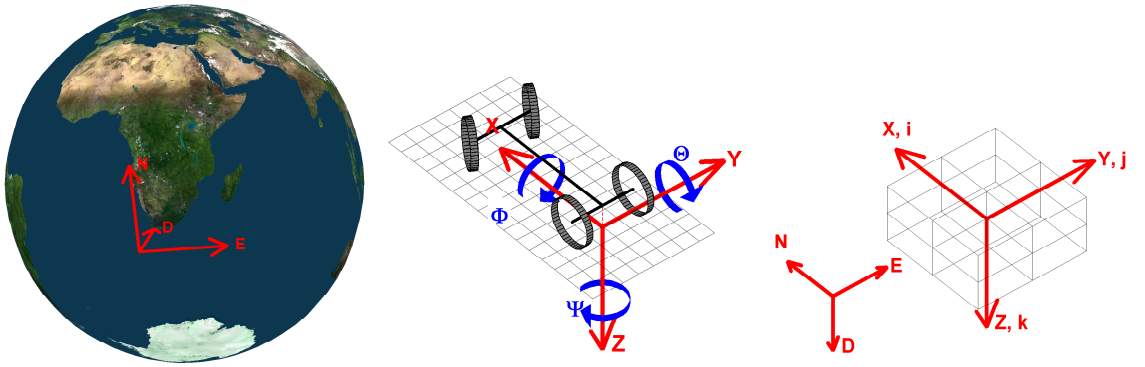
The inertial axis system (sometimes called the *NED axis*, short for North-East-Down axis) is stationary with respect to the earth's surface, with right-handed orthogonal axes labelled N, E and D (see Figure 4.6a). As the names suggest, N points due north, E east and D "down" along the gravitational vector at that location. The origin may be chosen anywhere on the earth surface, therefore the starting point of the vehicle mission is chosen for convenience.

The body axis system is an axis system whose origin and orientation coincides with the origin and orientation of the vehicle (see Figure 4.6b). Right-handed orthogonal axes are labelled x, y and z, with the x-axis parallel to the vehicle longitudinal centreline, the y-axis parallel to the rear axle and the remaining z-axis points down. The origin is tied to the vehicle chassis and the specific location is chosen as the midpoint of the contact points of the rear wheels when the vehicle is set on a flat, level surface.

Vehicle rotations are defined about these axes using the right-hand-rule. Positive roll, Φ, is about the positive x-axis. Similarly pitch, Θ, is about y-axis and yaw, Ψ, about the z-axis. In layman's terms, yaw is a 'right', pitch is a 'up' and roll a 'clockwise' motion. Since the order in which rotations are applied is not commutative, rotations are applied successively in the order yaw-pitch-roll, each separately in body axes. Note that a vehicle, positioned at its starting point, headed due north on a level surface shall have its body axes coinciding with the inertial axes.

The map axis system is used to provide coordinates within the digitised environment representation (see Figure 4.6c). The map shall be considered stationary, like the inertial

system and have its axes' orientation aligned to that of the inertial system. The origin, however, need not coincide with the inertial axis system. When referencing real world units, they are indicated by x, y and z, and when using logical indices, they are denoted by i, j and k.



(a) Inertial axes (b) Vehicle body axes (c) Map axes (shown with NED)

Figure 4.6: The three axis system definitions.

The sensor axes is fixed to the origin of the lidar beams, similar to the manner in which the body-axis system is fixed to the vehicle. The x-axis lies along the centre beam of the device and the z-axis is perpendicular to the plane in which the device scans, pointing "downwards". The y-axis completes the orthogonal axis system.

Transformations between two axis systems may be performed on coordinates and vectors using a direction cosine matrix (DCM), denoted by $\mathbf{T}$, and appropriate translations of coordinates. Subscripts in the form of $\mathbf{T}_{\text{from}\rightarrow\text{to}}$ indicates the source and destination axis systems.

4.2.2 Sensor Model

The OG requires a sensor model and this is in the form of a distribution function of readings returned, given the real world spatial setup, $p[r|z,\theta]$. Occupancy grids were mainly developed for sonar sensors with wide angular and range uncertainty, in the order of 12° and 1.2 m in [2], but can equally be applied to lidar devices with smaller uncertainty.

A SICK LMS-111 lidar was used with the key measurement specifications summarised in Table 4.1. For each lidar beam, the device returns a range reading, r, and an angular index, θ_i. From the combined beam divergence and angular increment, the angular uncertainty

is appoximated to be 0.5° and all range measurements fall within ±30 mm. On the scale of the operating environment, this sensor can almost be considered an ideal sensor.

Table 4.1: SICK LMS-111 lidar measurement specifications [9].

Description	Specification
Maximum range	20 m
Range error	±30 mm
Maximum scanning angle	270°
Angular resolution	0.25°
Beam divergence (full angle)	0.86°
Scanning frequency	up to 50 Hz

The sensor model is mapped to the cells, as a distribution over the cells – a process which is dependent on the vehicle state. This adds additional measurement error and we augment the uncertainty of the sensor model with the uncertainty of the state estimate. The available IMU, DGPS and estimator system is said to be accurate to 50mm in position data [14] and 1° in angular data [15].

The combined distribution is a convolution of the lidar and estimator distributions. Since neither distribution is accurately known and the process to characterise the distribution is tedious and an empirical approximation at best, no formal attempt was made to formally acquire such a distribution. Instead, a simple function is used that captures the essence of uncertainty in angular and range data, assuming these effects to be independent.

The angular distribution, $\Gamma(\theta)$, is approximated by a triangular function that peaks at unity on the lidar beam centre, decays linearly with divergence from beam centre and settles at zero for an offset beyond σ_θ,

$$\Gamma(\theta_{\text{div}})_{\text{sensor model}} = \begin{cases} \left|1 - \frac{\theta_{\text{div}}}{\sigma_\theta}\right| & \text{for } -\sigma_\theta < \theta_{\text{div}} < \sigma_\theta \\ 0 & \text{otherwise,} \end{cases} \tag{4.2.1}$$

where θ_{div} is divergence from the actual beam centre. Note that the angular divergence may equally represent an angle on a 2D plane or an angle in 3D space.

We approximate the range function, $\Delta(\rho)$, with a smooth transition inspired by [2], with unity at the surface distance and a smooth falloff over a selectable distance, $2\sigma_\rho$,

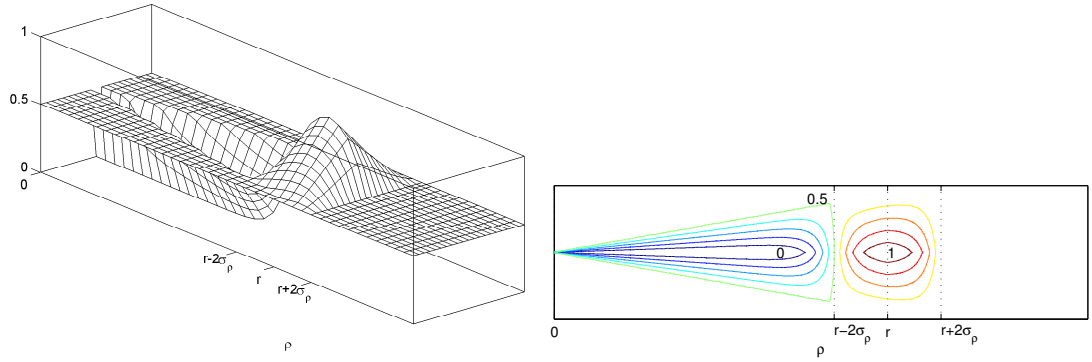
$$\Delta(\rho_{\text{div}})_{\text{sensor model}} = 0.5 - 0.5 \tanh\left(\frac{\rho_{\text{div}} - 2\sigma_\rho}{\sigma_\rho}\right), \tag{4.2.2}$$

where ρ_{div} is the divergence from the actual beam range.

Combining equations 4.2.1 and 4.2.2 into $p[r|s(C) = \text{OCC}]$ needed for equation 4.1.1 results in

$$p[r \mid s(C) = \text{OCC}] = \begin{cases} 0.5 + (\Delta(|\rho - r|) - 0.5)\Gamma(\theta_{\text{div}})) & \text{for } 0 < \rho < r + \sigma_\rho \\ & \text{and } -\sigma_\theta < \theta_{\text{div}} < \sigma_\theta \\ 0.5 & \text{otherwise.} \end{cases} \quad (4.2.3)$$

This function can be visualised as a 2D section through the projected lobe in Figure 4.7, with σ_ρ and σ_θ exaggerated to highlight their effect. In the Bayesian update of equation 4.1.1, all cells with $p[r|s(C) = \text{OCC}] = 0.5$ will be unaffected and are considered outside the lobe.



(a) Sensor model as a 3D mesh.

(b) Sensor model as a set of contours.

Figure 4.7: Two-dimensional sensor model for a beam. It depicts the distribution about the beam axis with range reading r, $p[r|s(C) = \text{OCC}]$.

For this sensor model, the two parameters, namely σ_ρ and σ_θ, needs to be specified. As a rough approximation, simply adding the approximate error bounds of the lidar device to the error bounds of the estimator gives:

$$\begin{aligned} \sigma_\rho &= 30\,\text{mm} + 50\,\text{mm} \\ &= 80\,\text{mm} \end{aligned} \quad (4.2.4)$$

$$\begin{aligned} 2\sigma_\theta &= 0.5^\circ + 1.0^\circ \\ &= 1.5^\circ \end{aligned} \quad (4.2.5)$$

4.2.3 Resolution of Occupancy Grid

The resolution of the OG is the volume unit used to represent the environment. The finer the resolution, the more detail it is possible to capture. On the other hand, a fine grid implies an estimation of more cells and will require more processing power. With a 3D implementation, the computational requirements increase with $O(n^3)$. This leads to a trade-off between processing speed and feature detail.

It is desrable to derive a minimum grid resolution that would render reasonable accuracy, with the advantage of the reduced computational load. The two key factors to consider are:

1. the minimum recognisable obstacle's size
2. and the accuracy of the sensor.

It is noted that, for a terrestrial vehicle, a relatively small error in vertical measurement can incorrectly classify terrain as drivable (e.g. driving up a curb 0.25 m high will cause damage to most ordinary cars). Whereas the same measurement error, but horizontally towards an obstacle, is small with respect to the average range measurement of obstacles which may be several metres.

It would seem like there is a greater tolerance of horizontal errors. From this, it can be deduced that having different grid resolutions for the vertical and the horizontal dimensions, denoted by α_{xy} and α_z, may be appropriate. From the project scope (Section 1.3.1) an obstacle is defined as an extrusion from the driving surface with minimum dimensions $\Delta h_{\text{obs}} > 0.5\,\text{m}$ and $a_{\text{obs}} > 0.5\,\text{m}$.

In the horizontal dimension, it is ensured that an obstacle will be represented in the 3D OG by ensuring that at least one column of cells is entirely occupied (Figure 4.8a). This is achieved by setting the horizontal resolution to half the minimum obstacle dimension,

$$\begin{aligned} \alpha_{xy} &= \frac{a_{\text{obs}}}{2} \\ &= 0.25\,\text{m/cell}. \end{aligned} \tag{4.2.6}$$

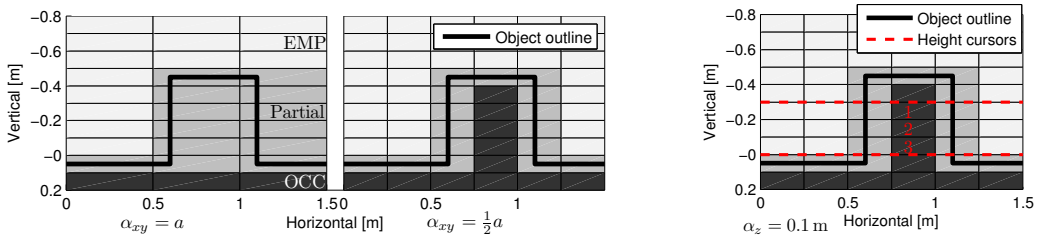
For argument's sake, choosing a vertical grid resolution $\alpha_z = 0.1\,\text{m/cell}$, leads to

$$\begin{aligned}\Delta h_{\text{obs}}{}^{\text{cell units}} &= \frac{\Delta h_{\text{obs}}}{\alpha_z} \\ &= \frac{0.5}{0.1} \\ &= 5 \text{ cells.}\end{aligned} \tag{4.2.7}$$

At best, extracting the height from the OG is accurate to one cell's dimension. In the worst case, if the driving surface appears one cell higher and the obstacle one cell shorter, we are left with a mere three cell difference to distinguish the obstacle (Figure 4.8b). From this we deduce that

$$\alpha_z = 0.1\,\text{m/cell} \tag{4.2.8}$$

is an acceptable resolution in height.



(a) $\alpha_{xy} = \frac{1}{2}a_{\text{obs}}$ ensures OCC cells

(b) $\alpha_z = 0.1\,\text{m}$ ensures three cell detection height

Figure 4.8: Illustrations of horizontal and vertical grid resolution.

Before the parameters of equations 4.2.8 and 4.2.6 are accepted, a sanity check is done against the accuracy of our measurements. Absolute positional estimate is within 50 mm, which is within the confines of a single cell, that is, the lidar origin with respect to cell indices is considered exactly known.

As mentioned before, angular error is multiplied by distance. With an angular accuracy of 1.5°, it is desirable to determine the maximum range at which an obstacle can be discerned. Using the three cell discretion height as guideline, it indicates the maximum usable range as

$$\begin{aligned}\rho_{\max} &= \frac{\Delta h_{3\text{ cell}}}{\arctan \sigma_\theta} \\ &= \frac{0.3}{\arctan 1.5^\circ} \\ &\approx 11.4\,\text{m},\end{aligned} \tag{4.2.9}$$

which is acceptable for a midrange application.

4.2.4 Mounting Pitch

Mounting pitch determines the range of the viewing horizon, which in turn affects the accuracy of the 3D OG map. As stated in Section 4.2.3, the viewing horizon goal is $\rho = 11.4\,\text{m}$. As stated in Section 3.3.2, the lidar will be mounted at a height of $z_{\text{B}}^S = -1.2\,\text{m}$, which is above the driving surface, with a fixed downwards pitch angle Θ_B^S. The sensor pitch angle as is determined to be

$$\begin{aligned} \Theta^S &= \arctan\left(\frac{z_B^S}{\rho_{\max}}\right) \\ &= \arctan\left(\frac{-1.2}{11.4}\right) \\ &= 6^\circ \end{aligned} \tag{4.2.10}$$

Figure 3.7a shows that vehicle pitch, Θ_I^V, has an influence on viewing distance. For the mounting setup described here, the change in viewing distance is showed in Figure 4.9. It is clear that small vehicle motions cause a considerable offset, especially for positive pitch angles. Note also how an increased mounting height decreases the sensitivity (dashed lines in Figure 3.7a).

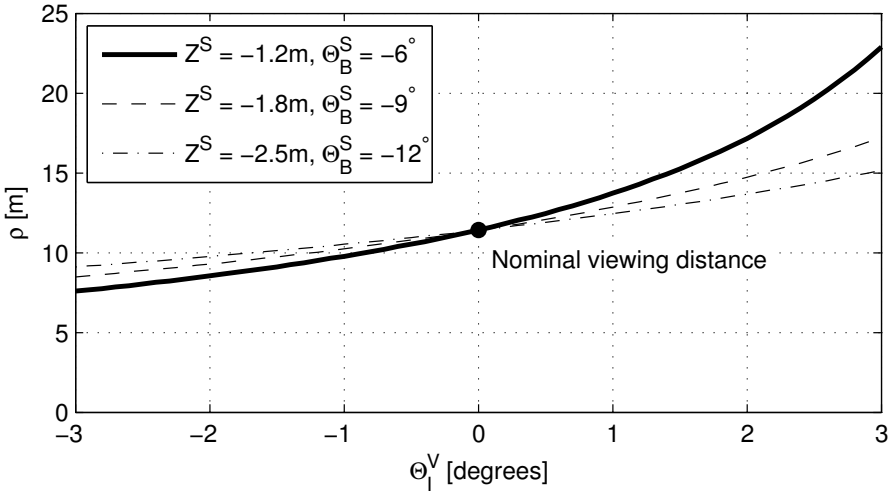


Figure 4.9: The influence of vehicle pitch on viewing distance for fixed sensor pitch.

The mounting also determines the maximum discernible height over distance, as portrayed in Figure 3.7c. As the vehicle nears an obstacle, the discernible height increases. For the

mounting setup described here, the full height of the minimum size obstacle is observable at

$$\begin{aligned}\rho_{\text{rec}} &= \frac{\left|z^{\text{S}}\right| - h}{\tan\Theta_{\text{I}}^{\text{S}}} \\ &= \frac{1.2 - 0.3}{\tan 6^\circ} \\ &= 8.5\,\text{m}.\end{aligned} \tag{4.2.11}$$

This calculation becomes irrelevant when a PTU can be used, but for the purpose of this thesis, equation 4.2.11 is assumed to be the nominal usable distance at which obstacles may be recognised.

4.2.5 Extent of Occupancy Grid

The extent of the occupancy grid determines the amount of memory required to hold the cell estimates. The naive approach would be to make a grid that covers the entire operating environment – the World for earthbound vehicles. The alternative is to confine the grid to a local area.

From the scope definition of Section 1.3.2 it is deduced that the vehicle's route will be uni-directional and therefore, the vehicle will pass through an area once and not return. Using memory to store global information that will not be used again is wasteful.

The extent of local map is defined to move along with the vehicle. For ease of digital storage and indexing, the 3D OG is maintained as a square prism, such that the horizontal sides are of equal length, with the vehicle near the centre of the grid.

The horizontal extent of the 3D OG should exceed the nominal viewing distance, so that all relevant surface detections are inside the map. The vertical dimension may be limited to the vertical space above the driving surface, up to the maximum height of the vehicle and down to just below the driving surface, to include the presence of the driving surface. The vehicle origin is bounded to the driving surface, therefore P_z^V is the driving surface height.

$$OG_x = OG_y = [-15\,\text{m}, 15\,\text{m}] \text{ relative to } P_{xy}^V \tag{4.2.12}$$

$$OG_z = [-1.0\,\text{m}, 0.2\,\text{m}] \text{ relative to } P_z^V \tag{4.2.13}$$

With the 3D OG extent and resolution known, the amount of digital storage required to store the map cab be calculated. Assuming each state is stored in a double-precision floating-point number of 8 bytes and substituting the the values of α and OG:

$$\begin{aligned}\text{Number of Cells} &= (\Delta OG_x/\alpha_{xy}) \cdot (\Delta OG_y/\alpha_{xy}) \cdot (\Delta OG_z/\alpha_z) \\ &= 675 \times 2^8 \,\text{cells}\end{aligned}$$

$$\begin{aligned}\text{Storage} &= (\text{Number of Cells}) \cdot (8\,\text{bytes}) \\ &\cong 1.3\,\text{MB}\end{aligned} \tag{4.2.14}$$

4.2.6 Beam Projection and Indexing

To update map cells with range measurements, sensor beams are projected into the discretised map, in order to determine each cell's relation to the sensor model. Measurements are made in the continuous real world, $\mathcal{W}$, and map cells are a 3D matrix in computer memory, $\mathcal{M}$, to be updated. This is essentially a mapping $\mathcal{W}(x, y, z) \mapsto \mathcal{M}(i, j, k)$.

In $\mathcal{W}$, a beam is measured from the lidar origin $\mathbf{P}_I^S$ with orientation $\mathbf{O}_I^S$, and the device returns a range and angular increment reading (r_n, ψ_n) for each beam n. A scan with N beams consists of $\left\{\mathbf{P}_I^S, \mathbf{O}_I^S, (r_1, \psi_1) \ldots (r_N, \psi_N)\right\}$.

As the 3D OG is defined in Cartesian coordinates, the sensor polar coordinates are converted to vectors in the xy Cartesian plane of the sensor axis system.

$$\mathbf{r}_{nS} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}_S = \begin{bmatrix} r_n \cos(\psi_n) \\ r_n \sin(\psi_n) \\ 0 \end{bmatrix}_S \tag{4.2.15}$$

In practice, $\mathbf{P}_I^S$ and $\mathbf{O}_I^S$ are not directly known in inertial axes. Instead, the mounting setup fixes the device origin in body axes, $\mathbf{P}_B^S$ and $\mathbf{O}_B^S$. The beam origin is the sensor origin, and is transformed to the inertial axes via

$$\mathbf{P}_I^S = \mathbf{T}_{B\to I}\mathbf{P}_B^S + \mathbf{P}_I^V, \tag{4.2.16}$$

where $\mathbf{P} = [x\ y\ z]^T$ and $\mathbf{T}_{B\to I}$ is the DCM from $\mathbf{O}_I^V$. The beam tip is transformed to inertial axes via nested transformations, from sensor axes, to body axes, to inertial axes:

$$\mathbf{r}_I = \mathbf{T}_{B\to I}\left(\mathbf{T}_{S\to B}\mathbf{r}_S + \mathbf{P}_B^S\right) + \mathbf{P}_I^V, \tag{4.2.17}$$

where $\mathbf{T}_{S\to B}$ is the DCM from $\mathbf{O}_B^S$ and $\mathbf{r} = [\mathbf{r}_1\ \mathbf{r}_2\ \ldots\ \mathbf{r}_N]$ is a row vector of the beam vectors.

The OG method updates the estimate of each cell with the probabilistic evidence of each reading. This implies that each and every cell has to be visited for each scan and updated with respect to each beam, which is a computationally expensive process. As can be seen from the sensor model (Section 4.2.2), the majority of cells fall outside the beam and will remain unaffected by the update process and need not be included in the calculation process.

To find the cells that are affected by a particular beam, it is necessary to know the extent of the beam's lobe in the sensor model. Updating only a subset of cells will significantly decrease the number of calculations needed for the sensor model and Bayesian update, at the cost of calculating the bounds of the lobe.

The sensor model's lobe is conical with its base at the detected surface, but the beam cone can be enclosed in a pyramid (see Figure 4.10). Since a pyramid has only straight edges, they are much more efficient to project. This projection can be achieved by defining a set of four rotation matrices, $\mathbf{T}_{\mathrm{BD}_{v=1\ldots4}}$, that describe the 3D rotations between each of the pyramid edges and the beam. Finding the vertices uses the result of equation 4.2.17, with one additional transformation:

$$\mathbf{v}_{Iv} = \mathbf{T}_{BDv}\mathbf{r}_I \tag{4.2.18}$$

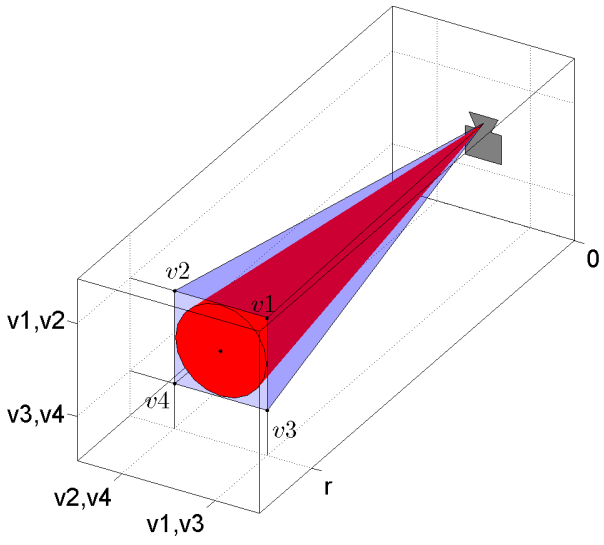


Figure 4.10: Using an elongated pyramid to bound the beam deviation cone.

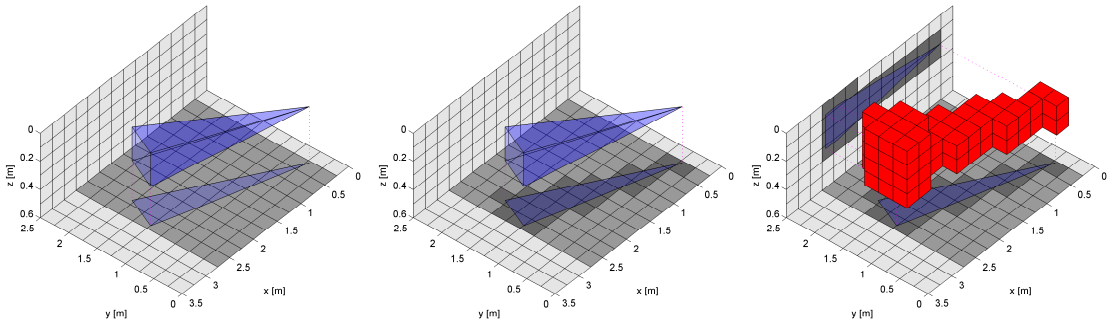
If the base of the pyramid is defined to be a vertical plane, it is possible to exploit the fact that vertices $v1$ and $v2$ have the same z value and likewise $v3$ and $v4$. Similarly, $v1$ and $v3$ have the same (x, y) coordinate as does $v2$ and $v4$. Thus projecting only for $v1$ and $v4$ generates all the necassary data and this can be used as optimisation. This concludes beam projection in $\mathcal{W}(x, y, z)$.

The map origin, $\boldsymbol{\mathcal{M}}_0$, is defined as the vertex in the map with the minimum coordinates in $\mathcal{W}$, such that all cell vertices have positive coordinates. Mapping a coordinate from inertial axis is now a simple translation

$$\mathbf{P}_M = \mathbf{P}_I - \boldsymbol{\mathcal{M}}_{0I}$$

and is applied to all beam projection geometry.

The next step is to identify all cells that are enclosed, or partially enclosed, by the beam projection. The process can be broken down into three tiers, as set out in Algorithm 4.1. Figure 4.11 visualises the three tiers, first calculating the x-range, then the y-range and finally the z-range of indices. Each range is then transformed to indices. A coordinate $(x, y, z)_M$ belongs to cell $\mathcal{M}(i, j, k)$, where $i = \lfloor x/\alpha_{xy} \rfloor$, $j = \lfloor y/\alpha_{xy} \rfloor$ and $k = \lfloor z/\alpha_z \rfloor$.



(a) Defining x-range (b) Defining y-range (c) Defining z-range

Figure 4.11: Projection of beam pyramid onto indexed cells. Note that the selected cells completely encapsulate the beam pyramid. Projections on the xy-plane and xz-plane aid visualisation.

4.2.7 Sensor Model Computation

Each cell is visited as returned by Algorithm 4.1 and needs its $p[r \mid s(C) = \text{OCC}]$ calculated. The distribution is dependant on ρ and θ_{div} of the specific cell. In the formulation

Algorithm 4.1 Compute i, j, k indices for a projected beam

```
Project beam into M(x, y, z). (equation 4.2.18)
Project pyramid onto xy-plane
  {results in an isosceles triangle}
Use minimum and maximum x coordinate of projection as x-range
Convert x-range to i index range with α_xy
  {clip i indices to the extent of map}
for all i in range i-range do
    Find the y-range of the projection corresponding to index i
    {using the projected triangle}
    Convert y-range to j index range with α_xy
    {clip j indices to the extent of map}
    for all j in range j-range do
        Find the z-range of the projection corresponding to indices i and j
        Convert z-range to k index range with α_z
        {clip k indices to the extent of map}
        for all k in range k-range do
            Compute p[r|s(C) = OCC] for cell C_ijk from sensor model
            Compute P[s(C) = OCC|{r}_t] with Bayes' theorem
        end for
    end for
end for
```

of the Bayesian update, we assumed each cell as an independent state. As such, cells may be visited in any ordering.

The centre of the cell may be used as representative location of the cell. Its coordinates are calculated by adding 0.5 to indices (i, j, k) and multiplying each with its respective resolution, α.

$$\mathbf{P}^{C_{ijk}} = \begin{bmatrix} (i+0.5)\alpha_{xy} & (j+0.5)\alpha_{xy} & (k+0.5)\alpha_z \end{bmatrix}^T \tag{4.2.19}$$

Calculating distance ρ from the origin to the cell is achieved with simple euclidean distance:

$$\rho_{ijk} = \left\| \mathbf{P}^{C_{ijk}} - \mathbf{P}^S \right\| \tag{4.2.20}$$

The angular deviation off the beam centre is calculated using vector algebra. Normalizing both the vector representing the beam, $\mathbf{r}$, and the vector from the sensor origin to cell's centre, $(\mathbf{P}^{C_{ijk}} - \mathbf{P}^S)$, results in two unit vectors, $\mathbf{n}_{\text{beam centre}}$ and $\mathbf{n}_{\text{cell centre}}$. The magnitude of the cross product of these two unit vectors is the sine of the angle between them. Therefore:

$$\theta_{\text{div}} = \arcsin\left(\left\| \mathbf{n}_{\text{beam centre}} \times \mathbf{n}_{\text{cell centre}} \right\| \right). \tag{4.2.21}$$

4.2.8 Bayesian Update

The new cell estimate for an indexed cell, C, can now be computed using Bayes' theorem as described in Section 4.1.3. Although the theorem makes provision for calculating the joint probabilities of cells, [2] motivates why cells may be assumed independent. From this assumption, they arrive at the following simplifying derivation:

$$\begin{aligned}\sum_{s(C)} p[r|s(C)=s(C)]P[s(C)=s(C)] &= p[r|s(C)=\text{OCC}]P[s(C)=\text{OCC}] \\ &\quad + p[r|s(C)=\text{EMP}]P[s(C)=\text{EMP}],\end{aligned}$$

but $P[s(C)=\text{EMP}] = 1 - P[s(C)=\text{OCC}]$,

$$\begin{aligned}\sum_{s(C)} p[\ldots]P[\ldots] &= p[r|s(C)=\text{OCC}]P[s(C)=\text{OCC}] \\ &\quad + (1-p[r|s(C)=\text{OCC}])(1-P[s(C)=\text{OCC}]).\end{aligned} \tag{4.2.22}$$

Substitute equations 4.2.20 and 4.2.21 into equation 4.2.3 and in turn equation 4.2.3 into 4.2.22 to obtain the denominator for Bayes' theorem. The numerator is the product of equation 4.2.3 and the previous cell estimate (refer to equation 4.1.1). The initial cell estimate (before any observations have occurred) is set to $P[s(c) = \text{OCC}] = 0.5$, to indicate an unknown estimate.

4.2.9 Local Map Shifting

The extent of the OG was defined to move with the vehicle in Section 4.2.5, such that when the vehicle moves, the extent of the map changes. For an observer fixed in the map axes, it would seem like the contents of the map had shifted a distance equal to the displacement of the vehicle, but in the opposite direction. A portion of the map will be discarded and on the opposite side of the map, a new region of unknown space is included in the map.

If the map extents were to change by an arbitrary distance, a problem arises when grids before and after the movement are not aligned. For a map of discrete cells, the contents of the map are fixed to the resolution of the grid, α. At each time step the grid contents are resampled, which results in some mixing of cell content with its neighbours. After a few

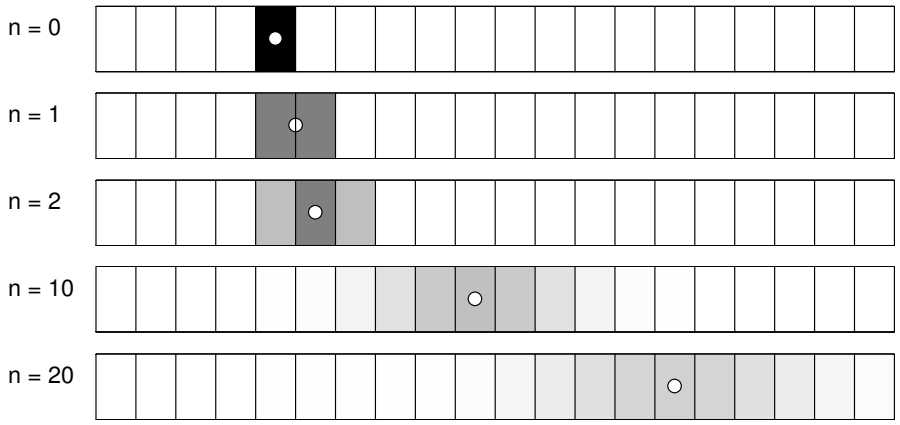


Figure 4.12: Illustration of the effects of resampling. Using a one-dimensional image, the original source is tracked for n iterations of worst-case resampling.

repetitions of resampling, serious degradation of fidelity occurs as depicted in Figure 4.12.

To remedy the resampling problem, the map is only shifted in integers of the map resolution α. This removes the need to mix cells during resampling and retains full fidelity. For the purpose of specifying the map extent, $\mathcal{M}_0$, using equations 4.2.12 and 4.2.13, the vehicle position coordinates are quantised to the grid resolution using

$$\begin{aligned} \hat{P_{xy}^V} &= \alpha_{xy} \cdot \left\lfloor \frac{P_{xy}^V}{\alpha_{xy}} \right\rfloor \\ \hat{P_z^V} &= \alpha_z \cdot \left\lfloor \frac{P_z^V}{\alpha_z} \right\rfloor . \end{aligned} \tag{4.2.23}$$

In practice, each time the map extent changes, the resampling operation is a batch memory copy operation that moves the contents with an integer number of index changes. After the shift, one or more edges of the map represent unknown environment, which were not assigned during the shift. These cells are initialised with $s(C) = 0.5$ to indicate the unknown state.

The copy operation can become computationally expensive if the size of the 3D OG becomes large. In [16] the authors suggest that the memory may be left stationary and instead, the indexing is changed to create an apparent shift. The indexing scheme calculates the new index by adding the amount of shift to the original index and uses modular maths to wrap indices outside the address space back into the addressable region:

$$\hat{i} = (i + \Delta_i) \bmod (\text{number of elements}). \tag{4.2.24}$$

The size of the OG used in this thesis does not warrant the use of this scheme, as the batch copy process takes approximately two milliseconds on a standard desktop computer[1]. For high resolution OG maps, this scheme may produce significant performance enhancements, provided that the cost of calculating the indices does not outweigh the benefits. This should be verified by using benchmarking.

The use of a local, shifting map also opens the possibility to maintain maps of different resolutions. A low resolution OG may be defined for a larger region of the operating world. When the local map shifts, instead of setting the cell state along the new edge to the unknown value, the state from the corresponding global map may be used. Similarly, the cells that are about to be discarded may be used to update the corresponding states in the global map.

4.2.10 Implementation Summary

Through the definition of various axis systems, it was shown that convenient transformations exist to project sensor beam orientation into the discretised map. The use of a bounding pyramid to approximate the beam spread cone can be used to acquire indices to the affected cells.

It was shown that for the specific class of sensor, a high angular accuracy lidar, that the sensor model significantly depends on the vehicle pose estimation accuracy. A simple sensor model was derived and the necessary computations shown to find the required range and angular deviation information. The Bayesian update formulation is completed with the assumption that cells may be evaluated separately.

To reduce computation cost and memory requirements, the 3D OG was defined to be localised to the vehicle position. Limiting resampling to integer shifts maintains the fidelity of the 3D OG. For moderately sized grids, memory shifting is appropriate, but for larger grids, using modular calculations on shifted indices may prove useful.

[1] Intel® Core™2 Duo CPU, E8500 @ 3.16 GHz, 3 GB RAM

4.3 Simulation

In order to develop, test and benchmark the 3D OG algorithms, a scanning lidar simulation was created. The algorithms for simulation and mapping were implemented in MATLAB™ for ease of visualisation during simulations. This section describes how the input lidar datasets were created, discusses some observations made during the creation of datasets and end with the execution of the mapping algorithm. The need for optimisation is also motivated.

4.3.1 Computing Lidar Datasets

Having stored sets of precomputed lidar simulation data allows the running of several mapping algorithms on the same dataset to compare results. A dataset consists of the vehicle states for position and orientation, the lidar device mounting position and orientation, and the simulated lidar range measurements. Datasets are created at the maximum rate and precision that the device offers and can be down sampled if required. The steps to create a dataset are given in Algorithm 4.2. The subsections to follow describe how to represent a simple ground truth environment, model vehicle motion over terrain and project lidar beams to find their intersection with the environment surface.

The advantage of simulating lidar datasets over collecting datasets with the actual sensor is that we have precise control over all parameters. The designer has an exact ground truth model of the environment to compare with mapping results. The vehicle path can be set exactly and the vehicle position and orientation data is not dependant on an estimator. Simulating different mounting positions requires no mechanical effort. Visualising beams within the simulation environment gives the designer insight that would otherwise be difficult to obtain.

4.3.1.1 Environment Model

In Section 4.1.1 it is explained that using a triangular mesh is convenient for representing surfaces. The MATLAB™ plotting environment has convenience functions for plotting surface height over a regularly spaced 2D grid of quadrilaterals, which the underlying implementation then converts to a triangular mesh for visualisation. The model parameters

Algorithm 4.2 Computing a simulated lidar dataset

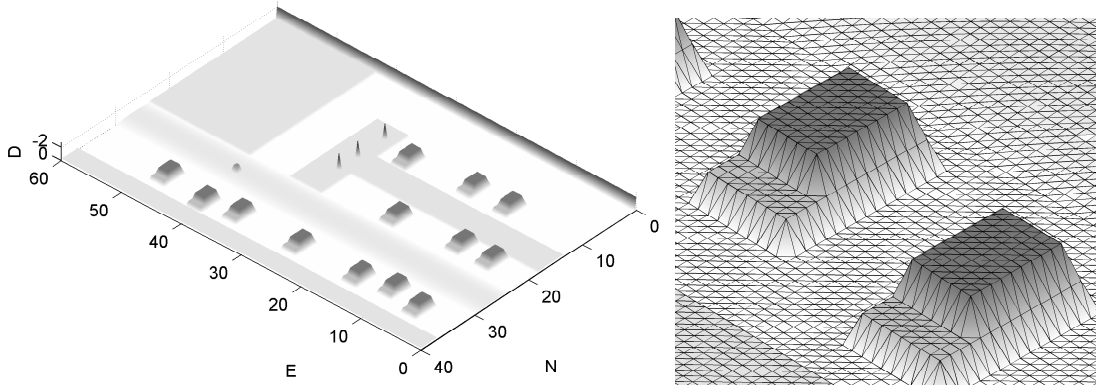
```
Define an environment model with a meshed grid
Define the route as waypoints and connecting lines/curves
Interpolate vehicle positions, P_I^V, along the route using simulation time
for all positions P_I^V do
    Calculate vehicle pose states (yaw, height, pitch and roll) and transformation matrix T_{B-I}
    if lidar mounted on PTU then
        Calculate sensor orientating and transformation matrix T_{S-B}
    else
        Use fixed orientation and T_{S-B}
    end if
    for all beams r_n do
        Project beam and find possible intersection with the model.
    end for
    Store the range data with vehicle and sensor data,
    {P_I^V, O_I^V, P_I^S, O_I^S, (r_1, ψ_1) ... (r_N, ψ_N)}
end for
```

are defined using the inertial axis system, so that the software components can be directly transferred from simulation to implementation.

To create an environment, the grid surface is initialised with all vertices set to zero height. To add an object, a group of adjacent vertices are set with an offset in the negative D-axis, so that the group extrudes from the driving surface. A test model was created, based on a parking lot, which includes large objects like cars, thin objects like lamp posts and tree trunks, a kerb outlining parking bays and a vertical wall along one edge. A curved road surface and a traffic cone were added to display certain effects. The meshed environment can be seen in Figure 4.13.

4.3.1.2 Vehicle Motion

To simulate the movement of the vehicle, a route is defined by hand as waypoints in the N-E axes, coupled to a time of arrival at each waypoint. The sections connecting waypoints are specified as either straight lines, or arcs of a circle. During simulation, the vehicle coordinates are determined by interpolating between the waypoints with time as parameter. The vehicle exerts Dubins-car behaviour [17] when using the origin of the vehicle body axis system as anchor point to trace the route and setting the vehicle yaw (the x_B-axis) tangent to the route curve. See Figure 4.14a.



(a) Model representing a parking lot(colour scaled to elevation)

(b) Enlarged portion of model to show triangular mesh of a "car"

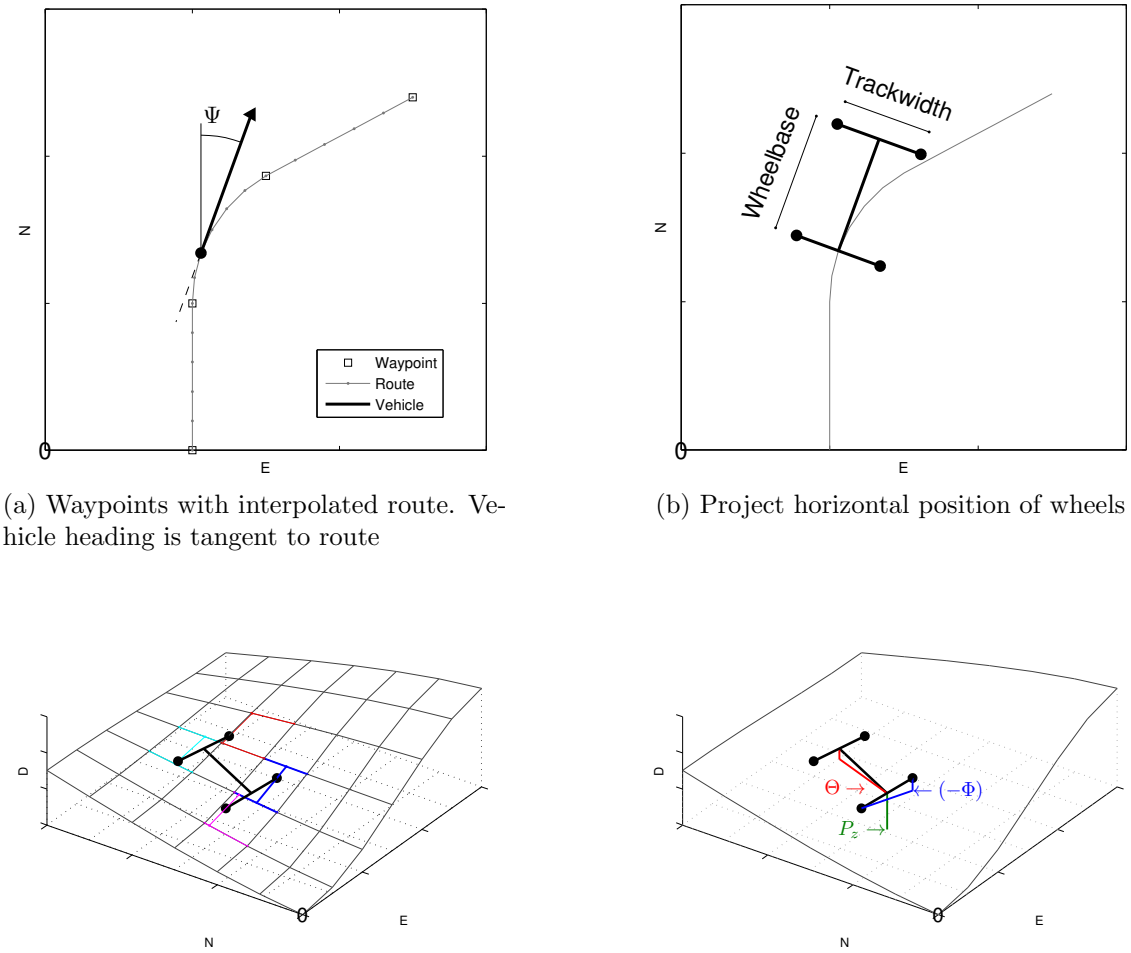
Figure 4.13: Triangular mesh model representing a parking lot

Calculating the remaining states (height, roll and pitch) requires the height of each wheel to be known. The ATV has a solid rear axle, but independent front wheel suspension. As such, the vehicle roll is dictated by the difference in height of the rear wheels (Figure 4.14d). The body axis origin is simply the midpoint between the rear wheel contact points and defines the height. To determine pitch, the vehicle nose is assumed to be at the average height of the front wheels. Vehicle pitch is determined from the difference between nose and rear height.

Using the horizontal position and yaw heading, the horizontal coordinates of the wheels can easily be projected using the wheelbase and trackwidth parameters (Figure 4.14b). For the purpose of this projection, the vehicle is assumed to be perfectly level. The height of the driving surface under each wheel, measured in the D-dimension, is bilinearly interpolated from the four corners of the quadrilateral that contains the wheel coordinates (Figure 4.14c). Applying the appropriate trigonometry to relative heights solves the vehicle states. For typical values of vehicle pitch and roll, the coupling between pitch and roll is neglected.

4.3.1.3 Beam Projection and Surface Intersection

For the purpose of the simulation, the lidar beams are projected without beam divergence. The projection angles of the lidar beams are transformed from the the sensor axis system, via the vehicle body axes, to world coordinates. The transformation is identical to the transformation used for the beam centre in Section 4.2.6 (equations 4.2.15-4.2.17) with the beam length, r_n, set to the maximum distance the lidar device can report.



(a) Waypoints with interpolated route. Vehicle heading is tangent to route

(b) Project horizontal position of wheels

(c) Interpolate wheel height from quadrilaterals

(d) Extract vehicle height, pitch and roll

Figure 4.14: Illustrations for simulating the vehicle motion

The calculation of intersections between the beam and the model surface is an expensive per-triangular-face operation. Since the model is meshed with a regularly spaced grid, we can project the beam "shadow" onto the grid and only test the surfaces that the beam passes over(Figure 4.15a). The beam is projected onto the N-E plane by discarding the D coordinate. A quadrilateral is selected if the projection of beam intersects any of the four sides of the grid surrounding it.

Starting at the quadrilateral closest to the beam origin and working towards the tip of the beam, we test for intersections along the beam path and stop at the first encounter. Each quadrilateral may be split into two triangles (in accordance with the MATLAB^{TM} visualisation). The face of each of these triangles are extended to represent an infinite plane (Figure 4.15b). If the intersection between the projected beam and this plane falls within the triangle, the beam has hit an object.

The point of intersection between a plane and a line can be calculated using vector algebra [18]. The formulation uses two points on the line, for which the beam origin, $\mathbf{P}_I^S$, and tip, $\mathbf{r}_I$, are used to construct a parametric line equation with parameter u (equation 4.3.2). A point on the plane (any vertex), along with a vector normal to the plane, constructs the plane. The normal, $\mathbf{n}$, can be obtained by the cross product of two vectors in the plane, for which the two sides of the triangle (equation 4.3.1) are used. The triangle vertices are labelled $v0$, $v1$ and $v2$, with corresponding coordinates $\mathbf{v}0$, $\mathbf{v}1$ and $\mathbf{v}2$.

$$\mathbf{n} = (\mathbf{v}1 - \mathbf{v}0) \times (\mathbf{v}2 - \mathbf{v}0) \tag{4.3.1}$$

$$\mathbf{P}_I = \mathbf{P}_I^S + u \cdot (\mathbf{r}_I - \mathbf{P}_I^S) \tag{4.3.2}$$

Solving the parameter u_{inter} for the intersection is given by equation 4.3.3, which uses two dot products. The result is substituted in equation 4.3.2 to obtain the point of intersection, $\mathbf{P}_{\text{inter}}$.

$$u_{\text{inter}} = \frac{\mathbf{n} \bullet (\mathbf{v}_0 - \mathbf{P}_I^S)}{\mathbf{n} \bullet (\mathbf{r}_I - \mathbf{P}_I^S)} \tag{4.3.3}$$

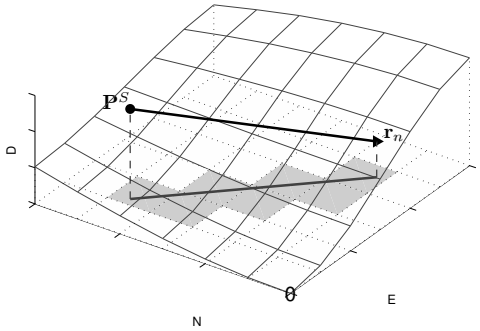
$$\mathbf{P}_{\text{inter}} = \mathbf{P}_I^S + u_{\text{inter}} \cdot (\mathbf{r}_I - \mathbf{P}_I^S). \tag{4.3.4}$$

If $\mathbf{P}_{\text{inter}}$ lies within within the triangle face, the beam distance is calculated using the euclidean distance from $\mathbf{P}_I^S$ to $\mathbf{P}_{\text{inter}}$. The test whether the point lies on the face can be done in the N-E plane using simple relational tests between the point and the three sides of the triangle, by discarding the D dimension of the vertices and the point of intersection (Figure 4.15c). If the intersection is not within the face, the next triangle is tested. If no intersection occurs over the length of the beam, or the edge of the model is reached, maximum distance is reported.

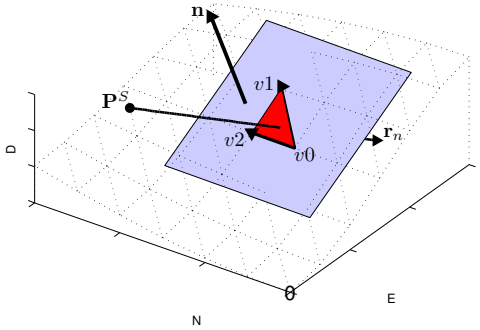
4.3.1.4 Comparing Point Clouds

The simulation environment allows the visualisation of the beams of the lidar device and the intersections within the environment model. The mapping algorithm can only map objects that are in the view of the lidar. Using the simulation, it is possible to investigate mounting strategies that ensure that critical obstacles will come into view in time to act upon them.

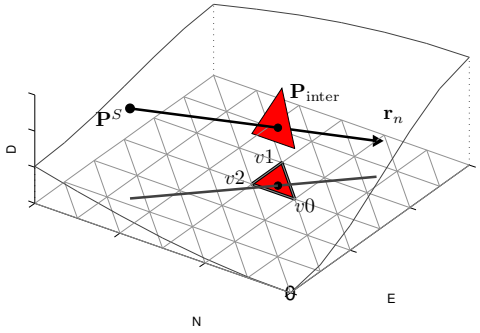
Plotting the set of surface detections, also known as a point cloud, onto the environment model shows the density of surface data collected. Colouring the point cloud with range



(a) Index relevant quadrilaterals on the 2D N-E plane



(b) Calculate line-plane intersection for a particular triangle



(c) Test in N-E axes if the point of intersection lies within triangle face

Figure 4.15: Illustrations for calculating the point of beam intersection.

information for each point adds a visual indication of accuracy to the points. An evenly spread point cloud minimises the amount of unknown area in the map. Also, the detection distance of the points should be in the usable range to ensure adequate accuracy and viewing horizon.

Although this type of investigation is not the primary focus of this thesis, some experiments were done with various mounting strategies. Point clouds were created for the following strategies:

- Fixed pitch slanted mount (as implemented)
- Higher and lower mounting positions
- Varying the pitch angle (emulating a PTU)
  - Sinusoidal motion with time
  - Triangular motion with time

- Varying both the pitch and roll angles (sinusoidal, 90° out of phase)
- Mounting the device on its side and rotating (vertical profile scan)

The qualitative results are summarised in Table 4.2. A common factor is that the periodicaly actuated strategies have to divide their scanning time between scanning the horizon and scanning the nearby driving surface. A "sampling period" appears to arise in viewing range equal to that of the actuation period, or alternatively stated, there are blind periods. If the vehicle drives at a steady pace, this results in periodic distribution of density in the point cloud. It is foreseen that the path planner will avoid unknown areas in the map and this therefore may result in unexpected planning behaviour. Of the periodic strategies, the coupled pitch/roll strategy has the best all-round distribution, using only a single input.

The greatest advantage of actuated mounts, are the ability to look far ahead for large obstacles and scan the nearby profile for accurate terrain mapping. Secondly, actuated mounts can increase point cloud density by "staring" at a surface while the vehicle is stationary. This is particularly useful when there are too many "unknown" areas in the map for practical path planning.

Table 4.2: Qualitative comparison of point clouds generated by using various mounting strategies in simulation

Strategy	Simple	Short Range	Mid Range	Long Range	Stare	Remarks
Fixed pitch	***	*	**	*	no	Only optimised for one range. Weak side scan.
Varying pitch	**	**	**	**	**	Weak side scan. "Divided attention".
Vertical, rotating	**	***	**	*	**	Long periods of blindness while device is pointed backwards, but omni-directional.
Active pitch/roll	*	**	**	**	***	"Focused attention" in useful range and direction.
Pitch/roll at 90° phase	**	**	**	**	**	"Divided attention", but omni-directional.

For the purpose of this thesis, the advantage of an actuated device does not outweigh the complexity and cost of a PTU mount. This is further supported by the relatively even

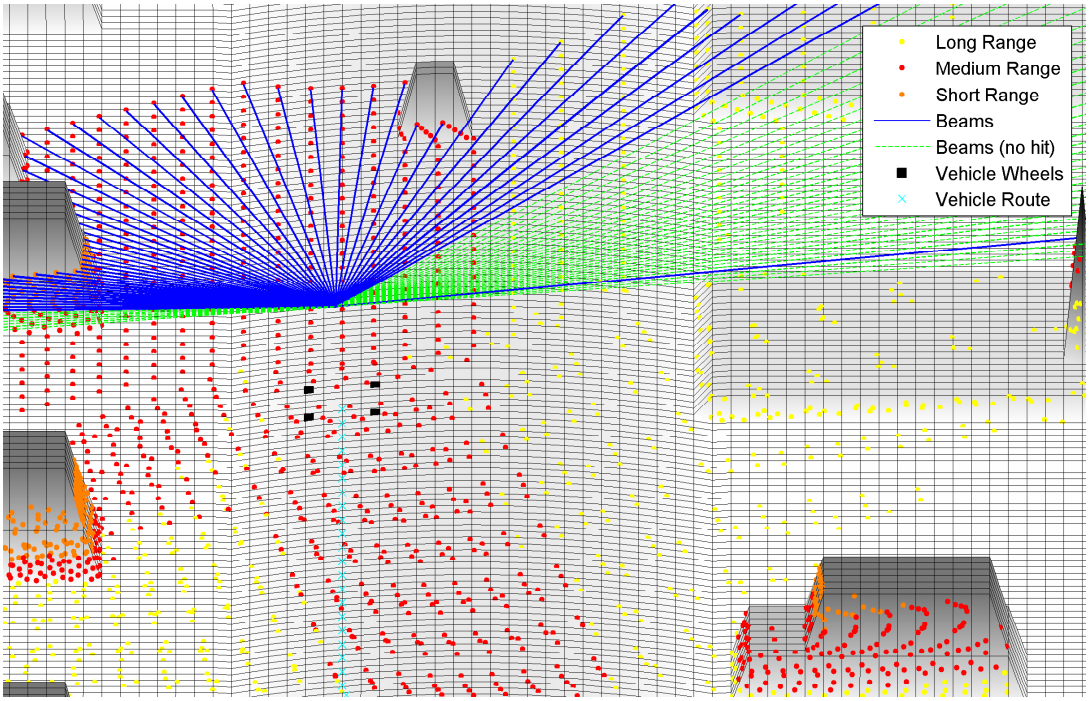


Figure 4.16: Simulated point cloud over environment model. The vehicle position and beam projections for the most recent time step are indicated. Points detected within the most usable range ($8.5\,\mathrm{m} < r_n < 12\,\mathrm{m}$)are indicated in red.

distribution of a fixed mount point cloud. It was found that setting the device mount as high as practical achieved the best results. It is still apparent that moderately uneven terrain causes roll and pitch actions that significantly influence the viewing distance. If a PTU were to be added, it is suggested that the PTU is used to maintain the viewing distance during normal driving conditions, with the benefit of "staring" when the vehicle is stationary.

4.3.2 Execute Mapping Algorithm

4.3.2.1 Software Structure

The simulation is implemented in modules so that it can easily be ported to a format suitable for real-time execution. The main program reads a dataset, initialises the map parameters with an empty map and then starts to iterate the mapping algorithm over the dataset. The main program also contains visualisation aids, showing the current position and pose of the vehicle in the environment model as well as an overview of the OG along with the drivability map for the 3D OG. Timekeeping markers are inserted to exclude the visualisations from the benchmarking as shown in Algorithm 4.3.

The 3D OG update algorithm is in function format and takes as its arguments the current 3D OG, the map extent, grid resolution and the sensor position, pose and range data with associated angles. This function will update the 3D OG independent of whether the map is local to the vehicle or bound in inertial space. A separate block of code is used to shift the map contents for a local map and initialise the new cells to the unknown state. It calculates the number of cells to shift the contents from the vehicle position data and updates the map extent information.

Band-limited noise was added to the vehicle orientation data to simulate the effects of sensor noise on the vehicle estimator. The 2σ bound was set at 1° in accordance with the accuracy expected from the estimator.

Algorithm 4.3 Simulation Structure

```
Load lidar dataset
Initialise 3D OG grid memory and map extent parameters
for t = simulation time steps do
    Sample from lidar dataset
    Start timekeeping
    if using a local map then
        Shift map contents and update extent parameters
    end if
    Run 3D OG update function
    Stop timekeeping
    Display visualisation of model and 3D OG
end for
```

4.3.2.2 Optimising Execution Time

The execution of the 3D OG update function has to be fast enough for real-time implementation. A successful team in the 2007 Darpa Urban Challenge [8] suggests a 10 Hz update rate as the minimum to allow sufficient time to respond to detections. From the simulation timekeeping results, it was clear that the MATLAB^{TM} implementation of 3D OG had to be optimised considerably to reach this goal.

The code was optimised to use batch computations for beams that use the same transforms and precomputing data that is shared amongst different code blocks. Where appropriate, calculating projections were reduced to 2D transforms, using $\mathbf{T}_{[2\times2]}$ instead of $\mathbf{T}_{[3\times3]}$, with considerable speed gain.

The final changes were to reduce the number of beams to include in the 3D OG. The execution time is $\mathcal{O}(n)$ proportional to the number of beams projected. It was found that the current implementation can project approximately 90 beams at the $10Hz$ rate[2]. The simulated lidar device (SICK LMS100) has a 270° field of view ($-135° \ldots 135°$ relative to Ψ^S) at a 0.25° angular increment, accounting for 1081 beams. Only the beams in the range, $-90° \ldots 90°$ relative to Ψ^S scans forward and are considered useful. It was found that subsampling the angular increment to 2°, and thus retaining 91 beams, still gives good results.

4.3.2.3 Findings

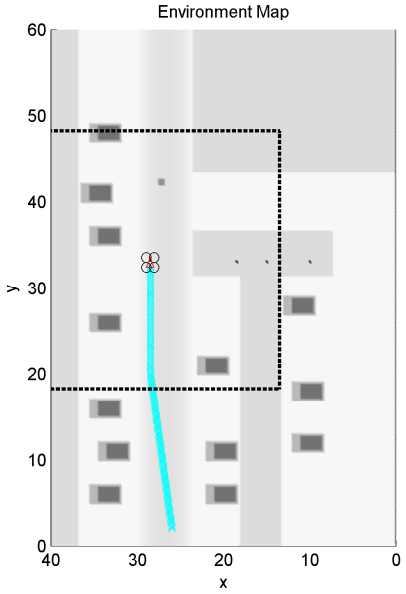
The simulation showed that a usable 3D OG map could be constructed in real time using MATLAB^{TM} scripts. Representative results are shown with the environment model in Figure 4.17. A simple search function is used along the z dimension to extract the boundary of cleared volume and the boundary of the surface mapping. Through simulation, important traits associated with the 3D OG method were identified.

The implementation of beam spread in the Bayesian update has difficulty mapping a surface where the angle of incidence between the beam and the surface are small. The beam tip is the only part of the beam that updates the 3D OG with the surface information, $s(C) = \text{OCC}$. As the vehicle drives forward, the subsequent beams pass over the surface with the length of the beam updating the 3D OG with $s(C) = \text{EMP}$ (Figure 4.18). Beam spread "bleeds" the EMP state into the cells representing the surface.

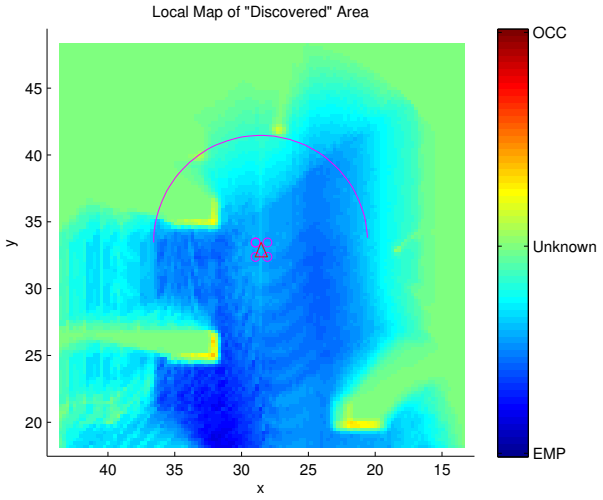
As many more beams pass over a surface than hitting it, some surface cells eventually turn into empty cells. When the angle of incidence is perpendicular to the surface, this does not happen. As a result, vertical obstacles are reliably detected, but mapping the driving surface is somewhat degraded. Again, a higher mounting position would prove advantageous to the angle of incidence.

It was found that narrowing the beam spread of the sensor model with respect to the clear region of the beam lessened this effect. This is done so that less area of the driving surface is updated with EMP observations. This effect warrants further investigation into a refined sensor model.

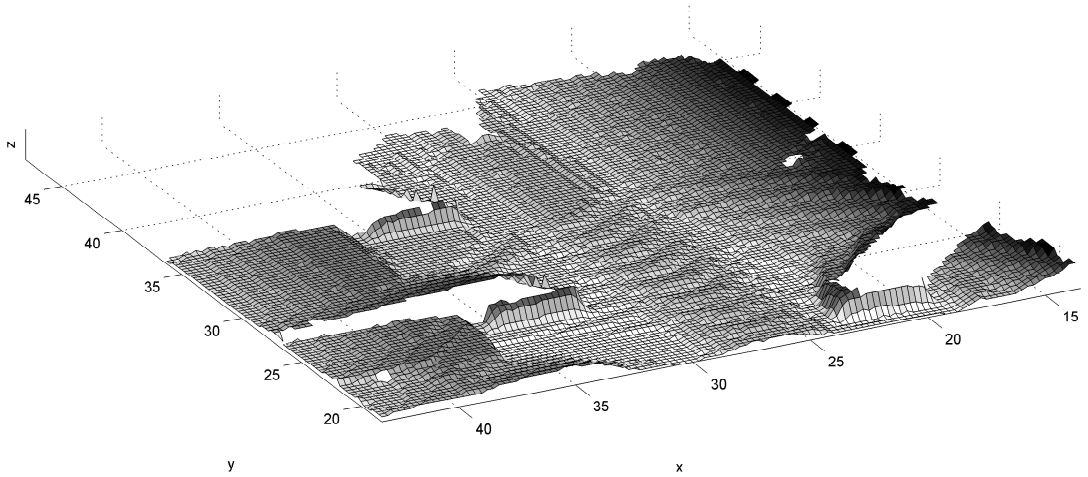
[2]Executed on a desktop PC running Windows XP: 32-bit, 3.1 GHz CPU, 3 GB RAM.



(a) Vehicle path and current position indicated in the simulated environment.



(b) Occupancy grid local to the vehicle. Columns of 3D OG is collapsed for display as a 2D OG. Arc indicates lidar field of view at ρ_{rec}.



(c) Surface extracted from 3D OG

Figure 4.17: Simulation visualisations of mapping algorithm

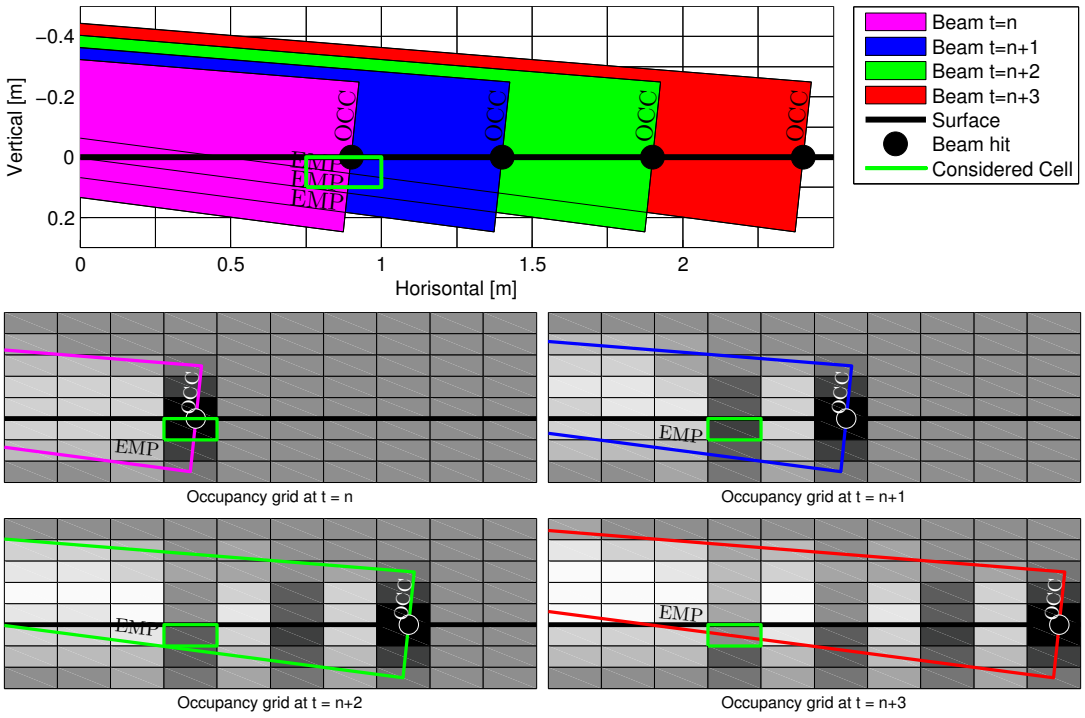


Figure 4.18: Illustration to show a surface cell being "EMP-tied" by successive beams. The beam at $t = n$ set the cell as OCC, but the following beams updates it with EMP.

Beam spread also causes a ridge in the mapped driving surface at the most recently updated surface cells (visible in OGs of Figure 4.18 at OCC text). The slight elevation is quickly levelled if subsequent beams passes over the area, as is the case when driving forward. This is not uncommon to the occupancy grid method, but can cause some surfaces to become ragged and erroneously be classified as undrivable.

4.4 Mapping Summary

A three-dimensional occupancy grid is the method chosen to map the lidar range data. The OG method is preferred as it explicitly models the empty space discovered by the lidar sensor. Surface mapping methods lack this property and can therefore not guarantee the absence of obstacles where none have been mapped. This is important information if the path planner is to generate safe paths.

The 3D OG is an incremental Bayesian estimation process that operates on stacked layers of cells. Each cell is represented by a variable of statistical probability, stored in a 3D matrix. Cells are updated with the information supplied by lidar scans. For this purpose, a

representative sensor model was constructed for use in the Bayesian update, that includes measurement noise.

Lidar beams are projected into the grid and cell estimates are updated, based on their relation to the beam. The projection is achieved by transforming the beam orientations from the sensor relative axis system to a earth fixed axis system, using the state information provided by the vehicle. To aid real-time execution speed, only cells that are intersected by a beam are updated. The conical beam shape is bounded within a pyramid that is efficiently projected and used for cell indexing.

The mapping algorithm was tested using a custom-developed lidar simulation environment. The simulation accepts a user-supplied 3D map and a simple vehicle route. It models vehicle motion over the terrain and returns virtual lidar range data. A visualisation was added that displays the lidar beams on the 3D map to gain insight into the spread of surface detections.

The simulation makes it possible to compare the mapping results with the artificial ground truth map. In general, the mapping algorithm returns good results. It is noted, however, that mapping a surface where the beams are incident nearly parallel to the surface, delivers a poor representation of the surface. We therefore suggest that the 3D OG is used primarily to map free space. Nevertheless, it was found that the mapping module is adequate to extract a drivability map for use in the path planner, as it recognises vertical obstacles with good reliability. A layer view of 3D OG is displayed in Appendix C, created with actual lidar data.

Chapter 5

Path Planning

An automated vehicle on a mission may encounter obstructions on its route and will have to navigate its way about these to avoid collision. In Section 1.2 path planning was defined as the detailed planning step that outputs the exact trajectory that the vehicle should follow to avoid conflict with its environment. For the purpose of path planning, the availability of an environment model (map) and a vehicle controller is assumed.

This chapter starts with an overview of the fundamental planning concepts, such as the configuration space, obstacles representation, planning constraints and the definition of completeness in path planning. The next section describes the problem-specific elements, namely the input map and the vehicle constraints. It is followed by the details of the sample based rapidly-exploring random tree (RRT) planner, tailored to the specific problem. Finally some simulation results are presented.

5.1 Path Planning Concepts

The path planning problem aim is essentially to find a collision-free path from an initial state to a goal state. This section will cover fundamental concepts of path planning using terminology from [3]. It introduces the configuration space and how to represent obstacles, defines planning constraints and explains the difference between continuous and sample-based planning.

5.1.1 Configuration Space

The vehicle operates in the infinite world, $\mathcal{W}$, that is the three-dimensional space $\mathbb{R}^3$. Some regions of the space $\mathcal{W}$ are occupied by obstacles, $\mathcal{O}$. $\mathcal{O}$ is defined as the closed set of points in space within, and including, the obstacle boundaries, such that $\mathcal{O} \subset \mathcal{W}$. The vehicle is a closed set of points, $\mathcal{V}$, that can reside in the remaining space, such that $\mathcal{V} \cap \mathcal{O} = \emptyset$. Expressing the collision-free space concisely is fundamental to planning.

To describe the set of points in $\mathcal{W}$ occupied by $\mathcal{V}$, only the shape of $\mathcal{V}$ in $\mathbb{R}^3$ and a mapping function, $h : \mathcal{V} \to \mathcal{W}$ is needed, that transforms each point in $\mathcal{V}$ to a point in $\mathcal{W}$. Such a function is termed a *rigid-body transformation* if 1) the distance between each pair of points in $\mathcal{V}$ are preserved and 2) the relative orientation of points in $\mathcal{V}$ are preserved [3].

If it is possible to parameterise the rigid-body transformation, this parametrisation is termed the *configuration space*, $\mathcal{C}$, of $\mathcal{V}$. In general a vehicle can translate along three dimensions ($\mathbb{R}^3$) and rotate about three axes ($\mathbb{S}^3$)[1]. When the six parameters are combined, however, they are more accurately described as $\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3$. Fortunately, for the case of a terrestrial vehicle translation is bound to a driving surface[2], the local area of operation is a manifold in $\mathbb{R}^3$, which is homeomorphic to $\mathbb{R}^2$ and rotation is limited to yaw, which is in $\mathbb{S}^1$. This implies the number of parameters were reduced to three (2D translation and rotation) and the dimension of the $\mathcal{C}$-space is three. Note that a single configuration $q \in \mathcal{C}$ maps all the points of $\mathcal{V}$ to $\mathcal{W}$ and that $\mathcal{C}$ represents *all* the possible configurations bounded to the driving surface.

5.1.1.1 Obstacle Region

Until now, the space in $\mathcal{W}$ that is occupied by obstacles, $\mathcal{O}$, has not been considered. The obstacle region, $\mathcal{C}_{obs} \subseteq \mathcal{C}$, is the set of all configurations, $q \in \mathcal{C}$, for which the vehicle and the obstacle sets intersect and are in conflict, expressed as

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{V}(q) \cap \mathcal{O} \neq \emptyset\}\,. \tag{5.1.1}$$

[1] Refer to [3], Part II, for an in-depth discussion of configuration spaces and set theory.

[2] We assume the driving surface never crosses over itself, as would be the case with a flyover or underpass. For a crossed over case, constructing a manifold may still be possible using appropriate identifications in the topological space.

5.1.1.2 Free Space

Conversely, $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$, is the set of *free space* in $\mathcal{C}$ for which the configurations do not cause conflict. By definition the obstacle and vehicle regions are closed sets, thus it follows that $\mathcal{C}_{obs}$ is a closed set. The remaining configurations $\mathcal{C}_{free}$ thus form an open set. This implies that a configuration can come arbitrarily close to the boundary of the free region without being in conflict with the obstacle region.

5.1.1.3 Path in $\mathcal{C}$ Space

An obstacle–free path is defined as the continuous trajectory, $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, such that $\tau(0) = q_I$ (the *initial configuration*) and $\tau(1) = q_G$ (the *goal configuration*), with $q_I \in \mathcal{C}_{free}$ and $q_G \in \mathcal{C}_{free}$. [3]

5.1.2 Representing Obstacle Regions

The manner in which the obstacle regions are defined is important, as it has a great influence on whether certain planning algorithms are efficient or not. Although the set theory used to formulate the configuration space results in concise expressions, in practice it is impossible to list the uncountably infinite number of points in a continuous configuration space. On the other hand, if the obstacle regions consists of a finite set of discrete configurations, it would be possible. In general, obstacle regions can be constructed using explicit or implicit representations.

5.1.2.1 Explicit Obstacle Regions (Geometric Modelling)

For explicit representations, the vehicle boundary, $\mathcal{V}$, and obstacle boundary, $\mathcal{O}$, are modelled with semi-algebraic expressions in $\mathcal{W}$. The boundaries of contact between the two sets can be algebraically computed and used to find $\mathcal{C}_{obs}$, the transformations that result in conflict. This works well with low order $\mathcal{C}$ spaces and using polygonal expressions for the regions. As the expressions for the boundaries and the order of the $\mathcal{C}$ spaces become more complex, the expression for $\mathcal{C}_{obs}$ soon becomes inefficient.

5.1.2.2 Implicit Obstacle Regions

To overcome the complexities of explicitly constructing the obstacle region, an implicit obstacle representation is searched by probing the configuration for free paths, by means of a sampling strategy. The implicit representation is in the form of a collision detection module which, to the planning algorithm, acts as a "black box". Separating planning from collision detection makes the planning algorithm independent of the (geometric) modelling used to test collision. When the number of primitive descriptors in the geometric model becomes large, the sampling based strategies can solve problems that would have been impractical using explicit strategies [3].

5.1.3 Planning Constraints

For many practical planning problems, the actor (vehicle in our case) cannot execute an arbitrary locus in $\mathcal{W}$. For a path to be a valid solution, the shape of the trajectory has to conform to certain *constraints*. The most common constraints of practical systems are kinematic constraints and dynamic constraints. These constraints may be applied simultaneously, termed kinodynamic constraints.

5.1.3.1 Kinematic Constraints

Kinematic constraints limit the motions that a vehicle can perform, without considering the force applied or the time needed to complete the motion. It may be presented as the range of motion a body can achieve with respect to translation and rotation expressed in geometric or algebraic form. The expression is often in the form of differential constraints over space, limiting the curvature of the locus.

5.1.3.2 Dynamic Constraints

Dynamic constraints adds the dependence on time to the problem. Since practical objects have inertia and the forces applied to incur motion have finite magnitude, motions are subject to acceleration and deceleration. Motion under dynamic constraints are often expressed as second (or higher) order differential constraints with respect to time.

5.1.3.3 Holonomic

For many problems it is possible to integrate the kinematic and/or dynamic constraints and obtain closed-form expressions for the locus over space. This category of constraints is termed holonomic constraints. If the expression cannot be integrated in closed form, it is termed non-holonomic.

5.1.4 Completeness

A planning algorithm is said to be *complete* if the following are true:

1. If a solution exists, it must find the solution in finite time;
2. If a solution does not exist, it must correctly report it.

There are two other classes of algorithms that can provide weaker guarantees of being complete [3]. A deterministic algorithm that samples the search space densely is called *resolution complete*, since it will find a solution in finite time if it exists, but keeps on refining the resolution infinitely if there is no solution. A dense random sampling algorithm is *probabilistically complete*, since the probability of finding the solution becomes one with enough samples, but will also continue indefinitely in the absence of a solution.

5.1.5 Continuous and Sampled Planning

Sampling-based motion planning attempts to separate planning from the representation of the conflict space, by using implicit representations. The general concept involves a discrete motion planner that tests candidate paths for conflict using the collision detection "black box". The collision detection module reports individual queries as conflicted or not.

Queries are sampled by the planning algorithm according to some search strategy. Since samples are drawn from an infinite set, it would take an infinite number of queries to probe the entire sample space and can at best be resolution complete or probabilistically complete. This leads to the fact that the solution is an approximation based on the resolution of the sampling strategy.

Continuous motion planning solves the planning problem algebraically with the semi-algebraic explicit representations of the conflict space and vehicle models. The resulting solution is exact. The algebraic implementation will execute in finite time and also report if no solution exists and is therefore complete.

Since sampling-based planning methods perform motion planning separately, it is often simple to include differential vehicle constraints, even of the non-holonomic class, in the planner. Continuous planning methods solve the motion analytically and differential constraints add considerable complexity. Solving non-holonomic problems without approximation in the continuous space remains a daunting problem.

5.2 Problem-Specific Elements

The practical path planner should execute in real time and generate paths suitable for the chosen vehicle, using the information supplied by the mapping module. The restrictions imposed by the vehicle are its differential kinodynamic constraints and its rigid body design for $\mathcal{V}$. The map is the local three-dimensional occupancy grid from Chapter 4.

This section starts with a discussion of the vehicle constraints for the automated ATV and how they can be simplified. Then the 3D OG map is collapsed to a 2D map and used to find the vehicle configuration space. The use of a sampling-based planner is motivated and its implementation is detailed.

5.2.1 Vehicle Constraints

To understand the possible trajectories that the ATV can execute, the vehicle kinematics are first studied in the absence of dynamics. After the kinematics have been defined, the influence of dynamics on vehicle motion will be discussed.

5.2.1.1 Simple Car Kinematics

The ATV belongs to the class of vehicle described as a *simple car* [3]. It has four wheels that roll (without skidding) on the driving surface, of which the two rear wheels are mounted in parallel on a fixed axle and the front wheels are steered together (Figure 5.1).

The steering angle, ϕ, is limited to a maximum deviation, $|\phi| \leq \phi_{\max}$, from the vehicle longitudinal axis. The rear wheel drive speed and the steering angle can be commanded.

The defining property for a simple car is that its wheels can rotate freely and may not skid sideways. For a fixed steering angle, this constraint is satisfied when the vehicle drives in a circle with constant radius. The geometric construction anchors the centre of the turning circle at the intersection of the extended rear and front axles and is dependent on the steering angle and the vehicle wheelbase, L_{WB}, such that $\rho = L_{\text{WB}}/\tan\phi$. The vehicle may also drive in a straight line ($\phi = 0$).

The configuration space parameters for a simple car are indicated on Figure 5.1. The origin of $\mathcal{V}$ is the centre of the rear axle and can be translated to $(x, y) \in \mathbb{R}^2$. The vehicle longitudinal axis can rotate with $\theta \in \mathbb{S}^1$, where $\mathbb{S}^1$ is any range that signifies one revolution e.g. $[-\pi/2, \pi/2]$ with the end-points identified. The constraints in the configuration space can be expressed as velocity constraints over the $\mathcal{C}$-space.

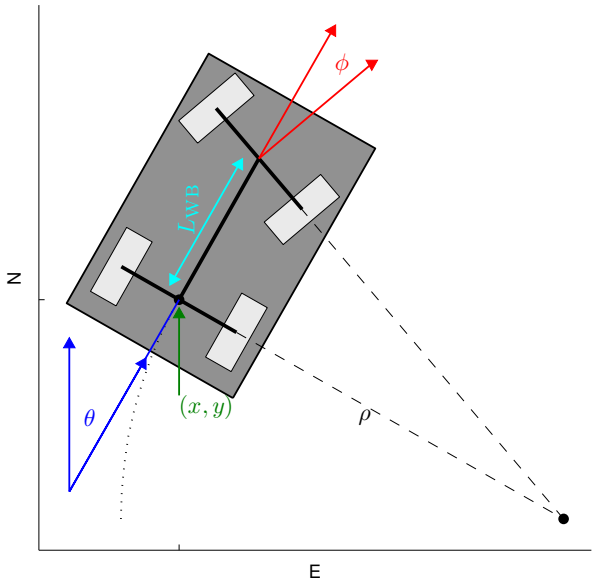


Figure 5.1: The simple car model. The configuration parameters, $q(x, y, \theta)$, and geometric construction for the turn anchor point, as a function of ϕ, are indicated.

Although the simple car kinematics have been assumed for the ATV, the following practical matters are noted that may degrade the model:

- The simple car model assumes that no translation is possible perpendicular to a wheel's rolling direction, but tyres have finite stiffness that, even without wheel slip, can cause a perpendicular velocity component while it rotates.

- The ATV has a solid rear axle without a differential drive. During a turn, the inside wheel has a shorter distance to travel than the outside wheel, but their rotational speed is equal, forcing some wheel slip. Fortunately, when driving at speed, centrifugal force causes some body roll and the resulting weight shift increases the frictional force of the outside wheel, concentrating wheel slip on the inner rear wheel. The simple car kinematics stay intact, based on the remaining three wheels.
- The ATV does not have a single front axle as with the simple car, but the steering mechanism has been imposed with non-linear linkage that approximately aligns the individual front axles to intersect the rear axle at the same anchor point.
- On a slippery surface, all the wheels could skid, in which case the kinematic constraints are wholly replaced by dynamic constraints. This situation is avoided.

5.2.1.2 Dynamics

The practical vehicle has two dynamic constraints, namely finite longitudinal acceleration and finite angular slew rate on steering. Acceleration to displacement is a second order differential constraint with respect to time. A maximum speed is also set. Steering angle slew rate is a first order differential constraint with respect to time.[3]

5.2.1.3 Incorporating Constraints

Without dynamic constraints, the simple car can instantaneously change speed, or steering angle. This implies that a simple car can change between a sharp left-hand turn and sharp right-hand turn without any transitional phase. For the simple car, the forward speed does not alter the turning radius. The steering command solely changes the turning radius. For constant steering angles, the kinematics for the path results in simple geometric curves, i.e. arcs of a circle or straight line segments, stringed together. For smooth transitions of steering angle, the simple car is however non-holonomic and is avoided.

The steering slew rate causes smooth transitions and results in a non-holonomic car. The simple geometric curves can be regained if the vehicle is restricted to only change steering

[3]This paragraph only describes dominant behaviour. In fact, both constraints (acceleration and slew rate) have higher order dynamics than stated here, but those are significantly faster and small in magnitude in comparison with the dominant behaviour. For the purpose of planning, the higher order dynamics may be safely neglected.

angles while it is stationary [17]. Since the vehicle cannot stop instantaneously, it would have to decelerate to stop precisely at the end of the current segment, adjust its steering angle and then start the next segment. This type of movement may be acceptable while performing strictly confined operations, such as parking, but is not practical for normal driving.

To simplify planning, we use simple geometric curves, but make the following adjustments to the vehicle model:

1. The vehicle is restricted to its minimum speed, since at low speeds, the distance travelled during the transitional phase is small relative to the distance of the segment. The lower the speed, the more closely the vehicle resembles a simple car.

2. During planning, a smaller value of $\phi_{\max}$ than the mechanical limit of the vehicle is used as a safety margin. The vehicle controller is assumed to be able to correct for minor deviations from the intended path, using the safety margin as headroom to actuate.

The resulting plan will be a sub-optimal approximation, but the use of simple expressions proves useful for implementation.

5.2.2 Obstacle Regions from 3D OG

Finding $\mathcal{C}_{obs}$ for the planning step is dependent on the shape of $\mathcal{V}$, the possible configurations and the obstacles, $\mathcal{O}$, in $\mathcal{W}$. In the current formulation, $\mathcal{W}$ and $\mathcal{V}$ are 3D spaces and $\mathcal{O}$ is mapped in a 3D OG, but $\mathcal{C}$ is translation and rotation on a 2D plane. First the 3D space is collapsed to a 2D plane, and then some assumptions are made to obtain an approximate $\mathcal{C}_{obs}$.

5.2.2.1 Collapsing 3D to 2D

For a vehicle confined to a manifold homeomorphic to a 2D plane, only the conflict that can occur while the vehicle is on the manifold need be considered. For the level operating surface defined in Section 1.3.1, the vehicle pitch and roll angles, induced by the surface deformation, are assumed negligible.

The ATV dimensions can be bounded in a rectangular prism, fixed in the body axis reference frame. The shape of the ATV fills most of the prism and therefore does not warrant the use of more complex polyhedra. The prism projects as a simple rectangular box onto the 2D plane that can translate and rotate when transformed through $h : \mathcal{V} \rightarrow \mathcal{W}$.

When collapsing the 3D OG onto the 2D plane, only cells covering the same z-dimension as the vehicle bounding prism have to be considered for conflict. For each vertical column of cells in the map, if *any* cell is in the occupied state ($P[s(C) = \text{OCC}] > 0.5$), the corresponding cell in the 2D OG grid is marked undrivable:

$$s(C_{ij}) = \begin{cases} \text{OCC} & \text{if } \exists\, s(C_{ijk}) = \text{OCC},\ k \rightarrow \mathcal{V}_D \\ \text{EMP} & \text{otherwise} \end{cases} \tag{5.2.1}$$

where C_{ij} is the cell in the 2D OG, C_{ijk} is a cell in the 3D OG and $k \rightarrow \mathcal{V}_D$ indicates the range of indices for k that maps to the D-range of the vehicle bounding prism.

5.2.2.2 Calculate Obstacle Regions

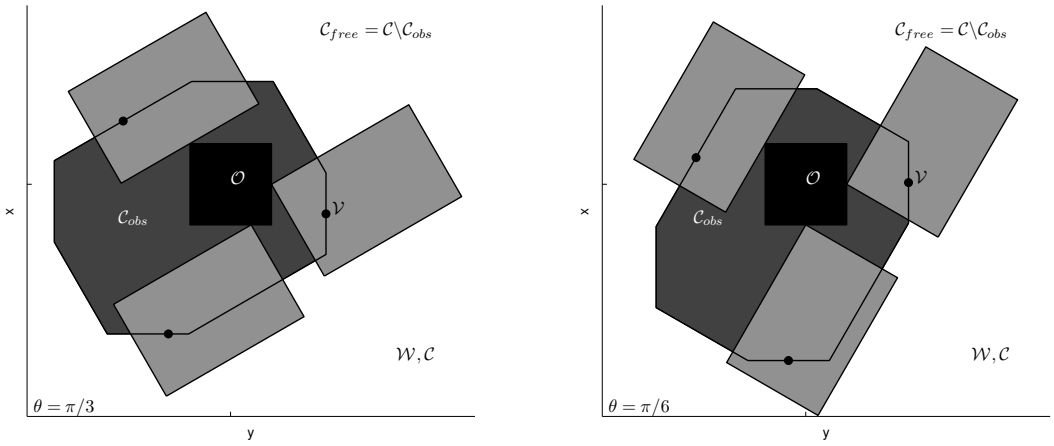
For configurations with a fixed rotation, say $\theta = \frac{\pi}{4}$, the obstacle region for the remaining two dimensions of $\mathcal{C}$ can be computed using the Minkowski difference [3],

$$\mathcal{C}_{obs} = \mathcal{O} \ominus \mathcal{V} \tag{5.2.2}$$

where $\mathcal{O}$ is the occupied cells in the 2D OG and $\mathcal{V}$ the rectangular box. This is illustrated in Figure 5.2, for a single occupied cell. It appears as though the obstacle has expanded with dimensions of the vehicle.

Observe however, that the vehicle will be driving forwards during the mission, passing obstacles at its sides, never at its nose or rear. As such, the longitudinal dimension of the vehicle is neglected to obtain a line with the length equal to the width of the original rectangle. Rotating the line through one revolution (θ over $\mathbb{S}$) results in a circle that contains all the points that the width of the vehicle can reach for any θ.

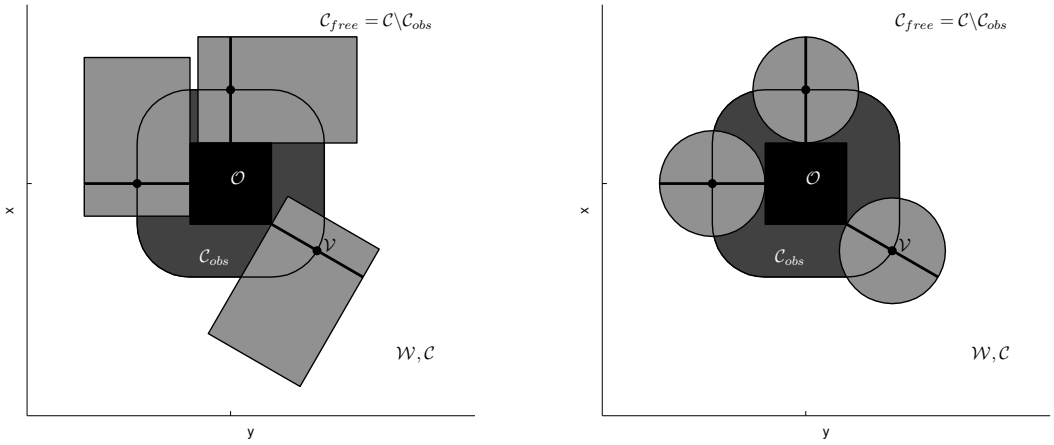
For the purpose of finding $\mathcal{C}_{obs}$, the configuration space has been reduced to translation of a circle in $\mathbb{R}^2$, that is simple to represent and compute using the Minkowski difference (example shown in Figure 5.3). Adding a safety margin to the radius of the circle ensures



(a) Minkowski difference at $\theta = 60°$ (b) Minkowski difference at $\theta = 30°$

Figure 5.2: Minkowski difference between a square obstacle and a rectangular bounding box shown at discreet values for rotation.

that the vehicle maintains some additional clearance from obstacles in the face of multiple approximations. To overspecify the obstacle region results in suboptimal paths (by excluding valid solutions), but is preferred to collision.



(a) Minkowski difference with bounding rectangle width. θ adjusted for "passing" $\mathcal{O}$

(b) Identical Minkowski difference with circle. Obtained independent of θ

Figure 5.3: Minkowski difference for a translating circle and a square obstacle.

For this simplified $\mathcal{C}_{obs}$, the planner is not allowed to drive up to an obstacle and stop just before it, since the vehicle length has been neglected. This can happen if the goal configuration is close to an obstacle or if the vehicle stops to change direction (reverse). For this reason, the vehicle is additionally limited to forward motion and it is specified

that the user supplied q_G will have sufficient clearance on all sides. A simple car limited to forward motion is termed a *Dubins car* [3].

5.2.3 Motivation for a Sampled Planner

With approximations, the vehicle kinodynamics can be made holonomic, which makes them eligible for continuous planning. Since approximations lead to suboptimal solutions, it would be preferred to utilise accurate vehicle models in future work. This may not be possible for continuous planning. Also, changing the vehicle model requires a complete redesign of the continuous planner, since it relies on algebraic models. This implies that it would be hard to reuse the planner on different vehicles.

The sampling-based strategy puts the vehicle model in a "black box" that can be easily swapped when a different vehicle is used. Some knowledge of the system can be included in the motion planning phase to improve the computation time of solutions, but it is not a requirement. The possibility of extending the sampling based strategies to any planning problem makes it the preferred choice.

Since the occupancy grid map already encodes samples of the configuration space it is well suited to sampled path planning. It will also be shown that the occupancy grid can be used as a risk function for the sampled planner. On the other hand, continuous planning methods are order dependent on the number of semi-algebraic primitives in the map representation. Converting the occupancy grid cells into algebraic primitives creates thousands of primitives, which will likely degrade the performance of continuous planners.

5.3 Adapted RRT Planner

The sampling based strategy is preferred, as it allows kinematic and dynamic constraints to be incorporated efficiently. In this presentation of the RRT, approximations are used to optimise some of the computations for a real-time implementation, but approximations are not essential to make the algorithm applicable to the problem.

5.3.1 Standard RRT Planner

The RRT is a specific implementation of rapidly-exploring dense trees (RDT) [3]. The idea is to rapidly explore the configuration space by incrementally connecting paths to sampled configurations in $\mathcal{C}$ and inserting them into a tree-like structure, called the search graph, $\mathcal{G}$. The graph consists of vertices and connecting edges. This approach yields good results as a search algorithm and the standard implementation requires no parameter tuning. The only requirement is a dense sampling strategy that will cover the search space.

5.3.1.1 RRT Algorithm

The basic algorithm for growing a tree consists of the steps in Algorithm 5.1. It does not account for obstacles, nor does it try to connect the goal, it simply covers the search space. The tree starts with only the initial configuration. In line 3, a new configuration is returned by the sampler, $\alpha(i)$, and inserted as vertex into the tree. If the sample $\alpha(i)$ is drawn at random, as opposed to drawn from an infinite sequence of deterministic samples, the RDT is labelled an RRT.

The NEAREST function selects the configuration, q_n, from all the configurations already included in the tree, that is the $q \in \mathcal{G}$ nearest to $\alpha(i)$. In line 6 the configurations along the path that connects q_n to $\alpha(i)$, are inserted in the tree as an edge. If the q_n lies on an edge, q_n is converted to a vertex that splits its former edge into two smaller parts, before it is connected. The NEAREST function requires that there is a suitable metric for the given search space. For each iteration i, the metric is applied to every vertex and edge already in the search graph with respect to $\alpha(i)$.

Algorithm 5.1 Standard RDT (without obstacles)

1: Initialise $\mathcal{G}$ with q_0
2: **for** $i = 1$ **to** k **do**
3: Sample a vertex, $\alpha(i)$
4: $q_n \leftarrow \text{NEAREST}(S(\mathcal{G}), \alpha(i))$
5: Insert $\alpha(i)$ in $\mathcal{G}$
6: Insert an edge from q_n to $\alpha(i)$ in $\mathcal{G}$
7: **end for**

5.3.1.2 RRT as Planner

The RRT functions as a path planner with some small additions. To accommodate obstacles, in lines 5 and 6, the path is tested for conflict before it is inserted in the tree. If it is in conflict, only the portion of the edge, starting at q_n, up to but not including the first point of conflict[4], is inserted. The use of the NEAREST function is motivated by the assumption that a shorter path will be less likely to contain a point of conflict than a longer path [5].

To connect the goal configuration, q_G, a modification is made to line 3. After every number of iterations, say every 100^{th} i, instead of sampling from α, q_G is substituted. If the goal can be connected to $q_n = \text{NEAREST}(\mathcal{G}, q_G)$ without conflict, a solution has been found and the algorithm terminated. If not, the algorithm continues as normal for 99 more iterations and then connecting the goal is retried. Sampling q_G regularly directs the search to expand towards the goal, and makes the search "focused", instead of wandering over all the space. Sampling q_G too often can prevent the tree from exploring $\mathcal{C}_{free}$ properly and delays solving a "difficult" route [3].

5.3.2 RRT with Differential Constraints

The power of the RRT lies with the ability to cope with differential constraints. Up to now, it has been assumed that, given two configurations, they can be simply connected since there were no constraints. This connection ignores the presence of obstacles and is termed a local planning method (LPM) [3].

5.3.2.1 Kinodynamic Constraints in State Space

The dynamic state of the vehicle can be described using state space methods from control theory. For most systems, the state, x, consists of the configuration, $q \in \mathcal{C}$, of the system and all the relevant time derivatives that drive the rate of change in the configuration, such as velocities, $\dot{q}$, or accelerations, $\ddot{q}$. The set of all states forms the state space, $x \in X$, with $x = \{q, \dot{q}, \ldots\}$. For dynamic systems, the search space is the state space.

[4]With $\mathcal{C}_{free}$ an open set, q can come arbitrarily close to the edge of $\mathcal{C}_{obs}$ [3]. How close is determined by the resolution of the practical conflict detection.

As with configuration spaces, a conflict region, X_{obs}, exists and the remaining space is $X_{free} = X \backslash X_{obs}$.

Connecting two states under differential constraints implies solving the boundary value problem (BVP) that describes the constraints. If a solution to the BVP exists, it is used as the LPM. The straight edge in line 6 of Algorithm 5.1 is replaced by the trajectory returned by the LPM. The trajectory is then tested to be in X_{free} before it is inserted into $\mathcal{G}$. Constructing an LPM is possible for the holonomic version of the Dubins car and is discussed in Section 5.3.2.3.

If an analytical solution for the LPM does not exist, or it is very expensive to calculate, numerical methods are used to approximate the trajectory. The most general method to obtain candidate trajectories, is to simulate various inputs to the system over various periods of time and select the trajectory which delivers the approximate final state [4].

It becomes difficult to guarantee that the state space is covered densely, since $\alpha(i)$ will not be inserted in $\mathcal{G}$ if the trajectory does not precisely includes $\alpha(i)$. We avoided using the dynamic model of the ATV with the simplifications of Section 5.2.1.3, but note that methods exist to include the dynamics in future work.

5.3.2.2 Cost Function as Metric

Differential constraints adds complexity to the selection of the "nearest" state from $\mathcal{G}$ with respect to $\alpha(i)$. It is desirable that the NEAREST function expresses the efficiency of connecting two states. It is therefore common to use a *cost function* to find the state to which the connection will have the lowest cost. A cost function accumulates cost over its path. The most common cost function accumulates a fixed cost per unit distance travelled, therefore penalising the path length.

It is possible to give different weights to different sources of cost, for instance, if travelling in a particular dimension requires more energy than in another, increasing the weight of that dimension minimises its use. This may result in a sense of optimal paths, as the planner will attempt to connect the path with the lowest cost first.

It is important that the cost function can be calculated efficiently, as it is calculated for every vertex and every edge in $\mathcal{G}$ at each iteration i. For this reason, the cost function

is often an approximation of the actual cost. Problems arise when the cost function underestimates the actual cost and care should be taken when an approximation method is used.

Computing the cost between an edge and $\alpha(i)$ may not be possible, as the edge can be a non-holonomic trajectory returned by a system simulator. A possible approach is to insert vertices along the trajectory at some sampling resolution and use these in the NEAREST function. This increases the number of primitives in $\mathcal{G}$ and emphasises the need for an efficient NEAREST implementation.

5.3.2.3 Shortest Path LPM and Cost Function for the Dubins Car

In order for the RRT to execute in real time, it is imperative that the LPM and NEAREST function be as efficient as possible. This section constructs a shortest path LPM for the Dubins car. The shortest path consists of basic geometric shapes, namely arcs of a circle and straight lines, that have trivial analytical path lengths. The computational cost of the construction is therefore shared between the LPM and NEAREST function.

Without regard for dynamics, the search space is simply the configuration space, $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$. A sample, $\alpha(i)$, is drawn from $\mathbb{R}^2$ to represent the next candidate vertex translation [5]. The rotation is left unspecified and acts as a target region to the LPM. The query to solve is thus an initial configuration, $q_1 = (x_1, y_1, \theta_1)$, and target region, $q_2 = (x_2, y_2) \times \mathbb{S}$.

The geometric construction for computing the shortest path from q_1 to q_2 is shown in Figure 5.4. The problem is first transformed, such that q_1 lies on the origin (now $\tilde{q}_1$) and θ is aligned with the x-axis [5]. If $\tilde{q}_2$ lies in the half plane, $y < 0$, it is mirrored about the x-axis, since the construction is symmetrical for left- and right-hand turns. The shortest path solution is based on two manoeuvres, first an arc at the minimum turning radius, ρ, followed by a straight line or another arc segment, that precisely connects the goal. The authors of [5] label the area within the turning circle $\mathcal{D}_\rho^+$ (shaded area in Figure 5.4) and then compute the minimum path length

$$L_\rho(q_1, q_2) = \begin{cases} f(\tilde{q}_2) & \text{for } \tilde{q}_2 \notin \mathcal{D}_\rho^+ \\ g(\tilde{q}_2) & \text{otherwise,} \end{cases} \tag{5.3.1}$$

where

$$f(\tilde{q}_2) = \rho \cdot \left(\theta_c(\tilde{q}_2) - \arccos \frac{\rho}{d_c(\tilde{q}_2)} \right) + \sqrt{d_c^2(\tilde{q}_2) - \rho^2} \tag{5.3.2}$$

$$g(\tilde{q}_2) = \rho \cdot \left(2\pi - \alpha(\tilde{q}_2) + \arcsin \frac{x}{d_f(\tilde{q}_2)} + \arcsin \frac{\rho \sin(\alpha(\tilde{q}_2))}{d_f(\tilde{q}_2)} \right) \tag{5.3.3}$$

with

$$d_c(\tilde{q}_2) = \sqrt{x^2 + (y - \rho)^2)} \tag{5.3.4}$$

$$\theta_c(\tilde{q}_2) = \arctan2(x, \rho - y), \ \theta_c(\tilde{q}_2) \in [0, 2\pi] \tag{5.3.5}$$

$$d_f(\tilde{q}_2) = \sqrt{x^2 + (y + \rho)^2} \tag{5.3.6}$$

$$\alpha(\tilde{q}_2) = \arccos \frac{5\rho^2 - d_f(\tilde{q}_2)^2}{4\rho^2}. \tag{5.3.7}$$

The two terms in equation 5.3.2 is the path length of a curved segment followed by a straight segment that reaches target configurations outside the turning circle, $\mathcal{D}_\rho^+$. When the target configuration lies within the turning circle, the shortest path is two curved segments with path length given by equation 5.3.3.

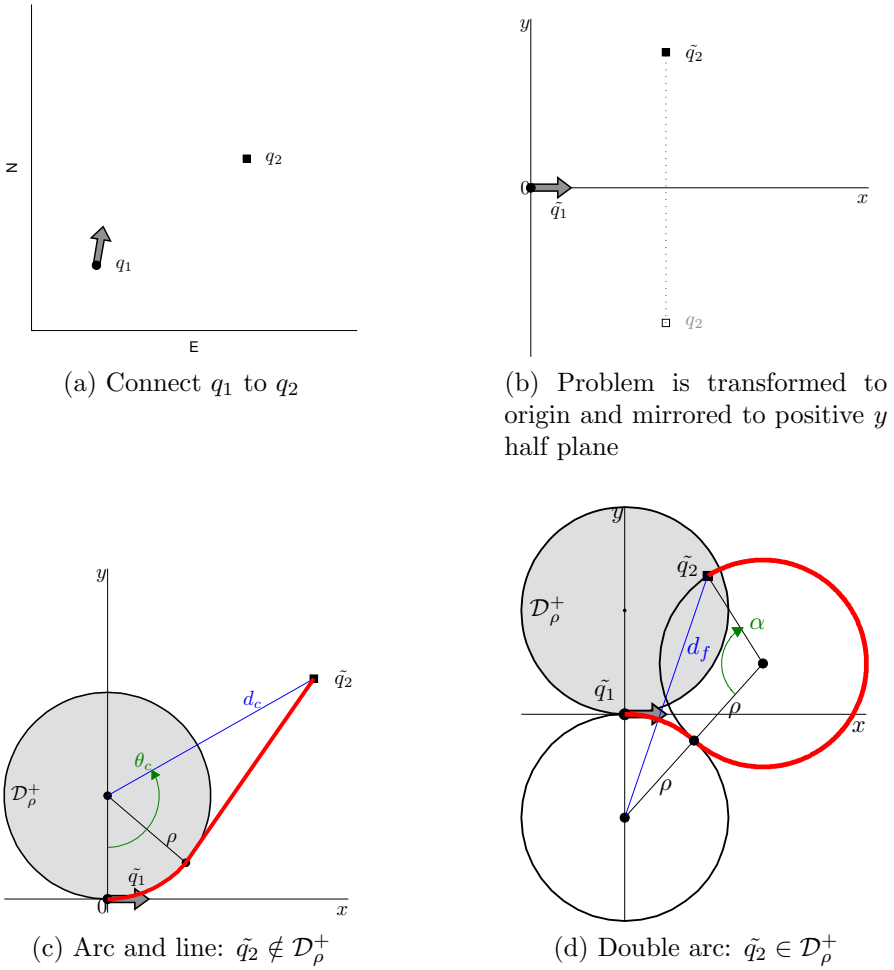
5.3.2.4 Conflict Detection on OG

The conflict detection algorithm is used to test each sampled configuration and path generated by the LPM for conflict, before it is inserted into $\mathcal{G}$. The region of conflict, $\mathcal{C}_{obs} \in \mathbb{R}^2$, is obtained obtained as a 2D OG, using the theory from Section 5.2.2. The region is stored in a 2D drivability map that uses the same cell-structure as the 2D OG.

The drivability map can be probed for a single configuration in constant, time$\mathcal{O}(1)$. The configuration translation, (x, y), is converted to the discrete indices, (i, j), that provide direct access into the map. To test a trajectory, the trajectory is traced over the grid and each cell crossed is probed. This process is $\mathcal{O}(n)$ dependent on grid resolution. For non-holonomic trajectories, the trajectory is sampled and assumed piecewise-linear between samples.

5.3.3 Implementation

The RRT for planning the vehicle path is shown in Algorithm 5.2. The tree is initialised and the obstacle region computed. Line 4 selects the next configuration that will be



(a) Connect q_1 to q_2

(b) Problem is transformed to origin and mirrored to positive y half plane

(c) Arc and line: $\tilde{q}_2 \notin \mathcal{D}_\rho^+$

(d) Double arc: $\tilde{q}_2 \in \mathcal{D}_\rho^+$

Figure 5.4: Geometric constructions for the Dubins car optimal paths

connected to $\mathcal{G}$. The sampling strategy (line 4) will be expanded in Section 5.3.3.3, Algorithm 5.3.

The path lengths from all vertices in $\mathcal{G}$ to the candidate vertex are calculated. This implies that the LPM is computed for all vertices. The trajectory in $\mathcal{C}_{free}$ with lowest cost is inserted into $\mathcal{G}$. In contrast to the unconstrained RRT, only trajectories without any conflict are considered for insertion in the tree. This is explained in Section 5.3.3.1. If no such trajectories exist, line 11 resets to the sampling algorithm.

The conflict-free trajectory is inserted in the tree. If the inserted vertex was sampled from the goal configuration, the path is complete and returned. The solution path consists of all the edges leading up to the final vertex, and therefore the tree is traversed in reverse order to extract the path.

Algorithm 5.2 RRT for Dubins Car on 2D OG

1: Initialise $\mathcal{G}$ with q_I
2: Calculate $\mathcal{C}_{obs}$ from the 2D OG and store as drivability map
3: **repeat**
4: $q2 \leftarrow$ candidate vertex, sampled from $\mathcal{C}_{free}$
5: **for all** $q1 \in \mathcal{G}$ **do**
6: Compute $L(q2)$ for LPM from $q1$ to q_2
7: Test the Dubins LPM trajectory for conflict in drivability map
8: **end for**
9: $q_n \leftarrow$ NEAREST$(\mathcal{G}, q2)$, such that the trajectory lies in $\mathcal{C}_{free}$
10: **if** $q_n = \emptyset$ **then**
11: Return to line 4 and acquire a new sample
12: **end if**
13: Insert vertex q_2 in $\mathcal{G}$
14: Insert edge from q_n to q_2 in $\mathcal{G}$
15: **if** $q_2 \in q_G$ **then**
16: **return** solution
17: **end if**
18: **until** terminated

5.3.3.1 Optimal Path

The standard RRT planner does not attempt to find optimal paths, instead its focus is on quickly exploring the search space for a *feasible* path. Selecting the nearest vertex from the tree shortens the new trajectory, which in turns makes it less likely to cross an obstacle region. It does not imply that the total path length, following the edges in the tree up to the current vertex, is the shortest path.

If the NEAREST function is altered to select the vertex, q_n, such that the total cost to reach the new state is minimised, a sense of optimality has been created. The cost to reach a vertex, $C(q)$, is the cost of executing all the edges leading up to it, and is stored with the vertex. To calculate the cost of a new vertex, the cost stored at the previous vertex and the cost of the new edge are summed:

$$C(q_2) = C(q_1) + L_\rho(q_1, q_2). \tag{5.3.8}$$

This formulation can cause the standard RRT planner to lose completeness (recall Section 5.1.4), as it will not insert an edge that makes the necessary detour around obstacles. The shortest edge to a state "behind" an obstacle will always pass through the obstacle region and will therefore be truncated. This is circumvented in the adapted RRT, as it

only considers conflict-free paths in the NEAREST function, hence a detour edge may be inserted if it is conflict-free.

The author of [3] notes the region of *inevitable conflict*. This is the set of states, for which the configuration is not in conflict, but due to differential constraints, all trajectories originating from it will inevitably cause a conflict. This area typically surrounds the existing obstacle region. [3] therefore discourages inserting partial trajectories up to the border of $\mathcal{C}_{obs}$, as in Section 5.3.1.2, which will likely result in inevitable conflict.

5.3.3.2 Risk as Additional Cost

The optimal planning strategy in Section 5.3.3.1 can be used to avoid high risk regions, by attaching a cost penalty to risk. A risk region is defined as the configuration space in close proximity to the obstacle region, or configurations spanning under mapped area (where $P[s(C_{ijk}) = OCC] = (0, 0.5]$). The state variable of the drivability map is extended from the boolean OCC/EMP representation, to the range $R(C_{ij}) = [0, 1]$, where $R(C_{ij}) = 0$ is risk free ($P[s(C_{ijk}) = OCC] = 0$) and $R(C_{ij}) = 1$ is maximum risk ($P[s(C_{ijk}) = OCC] \geq 0.5$).

The risk for a particular cell is chosen as the maximum risk within its area of proximity. The risk associated with a risk source, $R(RS)$, is scaled with proximity, decreasing with distance between the cell and the source,

$$R(C_{ij}) = \begin{cases} \max(1 - \frac{1}{\sigma_R}\|C_{xy} - RS_{xy}\|) \cdot R(RS) & \text{for } \|C_{xy} - RS_{xy}\| \leq \sigma_R \\ 0 & \text{otherwise,} \end{cases} \tag{5.3.9}$$

where the subscript xy indicates 2D position and σ_R the range deemed proximal. An example of a 2D OG with associated drivability map is shown in Figure 5.5.

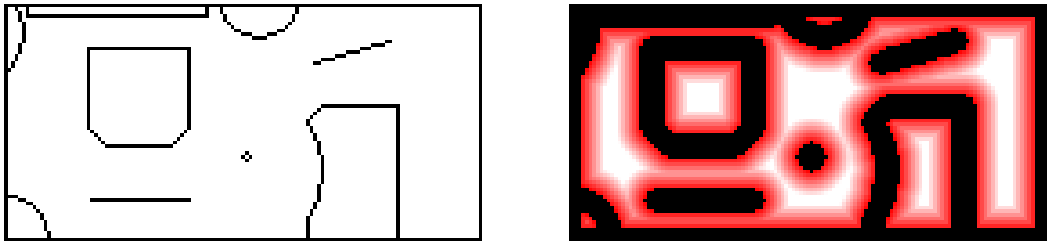


Figure 5.5: Risk conscience drivability map. The yard map base is shown on the left and the drivability map on the right. Red = high risk, White = risk free and Black = $\mathcal{C}_{obs}$.

To include the risk of an edge in the cost function, cost has to be accumulated when executing an edge. It was chosen to accumulate risk over unit distance travelled through the risk region. Thus the longer the distance a trajectory remains in a risk region, the greater the cost of the trajectory and the less likely that it will be inserted into the tree. The computation is included in the conflict detection module, as it already probes the drivability map for $\mathcal{C}_{obs}$.

The weight of risk (w_R) versus path length (w_L) in the planner cost function determines the distance of detours that will be taken to avoid risk. Since the risk is bound on the interval $[0, 1]$ and accumulated over unit length, the weights adjust the ratio of detour path length to risk path length. Equation 5.3.8 is now extended to

$$C(q_2) = C(q_1) + w_R \cdot R(q_1, q_2) + w_L \cdot L_\rho(q_1, q_2). \tag{5.3.10}$$

5.3.3.3 Sampling Sequence

For a real-time reactive path planner, the planning algorithm will be rerun constantly to react on new information added to the drivability map. This implies that the RRT planner will calculate a completely new path each time the RRT is executed, even if no changes occurred. During execution, it may seem like the vehicle is driving erratically as it switches from one random tree to the next at each run. It would be preferable if the vehicle could remain on the same path, unless that path becomes obstructed.

The solution path from the previous RRT can be used as waypoints that guide the current tree expansion to include a similar solution [6]. When the tree is initialised, the first n candidate samples to be inserted are the n vertices of the previous solution path, excluding the goal. If there were updates to the drivability map, this step automatically re-checks for conflict and calculates the updated cost of reaching the vertices.

To promote the search for more optimal paths, the search space is explored for a set amount of time, before the goal state is sampled. During this time, many randomly selected states are added to the tree that may or may not be connectible to the goal. The waypoints from the previous solution improve the likelihood of connecting states near the previous solution, inserting many paths similar to it.

When the goal is eventually sampled, there may be many alternatives to connect from. Only if such a path has lower cost (closer to optimal), will the vehicle switch its path.

If the goal is sampled, but not yet reachable, the exploration process is continued as per the standard RRT planner, resampling the goal every number of samples. The complete sampling strategy is shown in Algorithm 5.3.

Algorithm 5.3 Sampling Strategy for the Adapted RRT

```
1: waypoint_list ← vertices from previous solution path
2: Start timer: EXP_TIME ← T_exploration period
3: i ← 0
4: for sample request do
5:     increment i
6:     if i ≤ number of waypoints then
7:         q_2 ← waypoint_list[i].q
8:     else if i is a multiple of goal sampling period and EXP_TIME expired then
9:         q_2 ← q_G
10:    else
11:        q_2 ← uniform pseudorandom sample over ℝ^2
12:    end if
13:    if q_2 ∈ C_obs then
14:        return to line 6
15:    end if
16:    return q_2
17: end for
```

5.3.3.4 Sampling Space Extent

In Section 1.2 it is explained that path planning is limited to the practical range over which the map is considered complete. Planning beyond this distance is futile, since the information is not trustworthy and likely to change. It is desirable to plan within the viewing range of the environment sensor.

The planner is restricted to exploration within a radius of the vehicle considered the viewing horizon. This is accomplished by including the space outside of the viewing horizon in $\mathcal{C}_{obs}$. As such, the conflict detection module will reject all LPM trajectories that leave the viewing horizon. Similarly, the sampling strategy (Algorithm 5.3) only creates pseudorandom candidate configurations in $\mathcal{C}_{free}$.

An exception is made when attempting to connect the goal, when all space outside the viewing horizon is considered part of $\mathcal{C}_{free}$. This is necessary, since the goal need not lie within the viewing horizon and the drivability map may contain false obstacles beyond the viewing horizon.

From the findings in Section 4.3.2.3, it is known that the map contains some artefacts at the most recent profile scanline, due to beamspread in the Bayesian update process. These may erroneously appear as obstacles, which is nominally at a distance of 11.4 m (equation 4.2.9). Also note, the nominal reliable detection distance, $\rho_{rec} = 8.5\,\text{m}$ ahead (equation 4.2.11). From this, ρ_{rec} was chosen as the appropriate planning horizon.

5.3.3.5 RRT Completeness

An important factor that defines the success of the RRT algorithm is whether or not it is complete. Recall that a complete algorithm will find a solution if it exists. A sampled algorithm is resolution complete if it will tend to cover the entire configuration space given infinite time. The standard RRT is resolution complete and although planning problems with very narrow passes can take long to solve, they will be solved.

The planning algorithm presented here is not complete, since it does not ensure that the rotational dimension of the configuration space is sampled densely. This was sacrificed for a simpler LPM and to limit the sample space to two dimensions. For the kinematic constraints in limited space, many configurations are unreachable and considerable time will be wasted on attempts to connect them. The current LPM connects to a target region (any θ), which improves the likelihood of inserting edges and expanding the tree quickly. Despite its shortcomings, it was used successfully in [5].

5.4 Simulation

The planner as designed is implemented in MATLAB$^{\text{TM}}$, where the planning process can be simulated with visual confirmation. For release as a practical path planning module, the planner is encapsulated in a internet protocol (TCP/IP) host interface. The mission planner uploads a planning query and the planner returns the solution path. In this section the interface presented to the mission planner is shown and the findings from simulation results are reflected upon.

5.4.1 Interface

The planner interface is shown in Figure 5.6. The planner is compiled with the minimum vehicle turning radius as constraint, the vehicle width and distance deemed proximal for the risk region, the planning horizon distance and the cost function weights. The map grid resolution and extents and the periods for search space exploration and failure time-out are set during initialisation.

For each replanning iteration, the planner receives the vehicle configuration (q_I), the goal coordinates (q_G), the updated 2D OG and its extent in NE-axis. The waypoints that guide the next iteration are maintained internally by the planner. When a solution is found, the planner returns the list of vertices that forms the solution path, in the form of configurations, $\tau = \{q_1, q_2, \ldots, q_G\}$. These are passed to the vehicle controller.

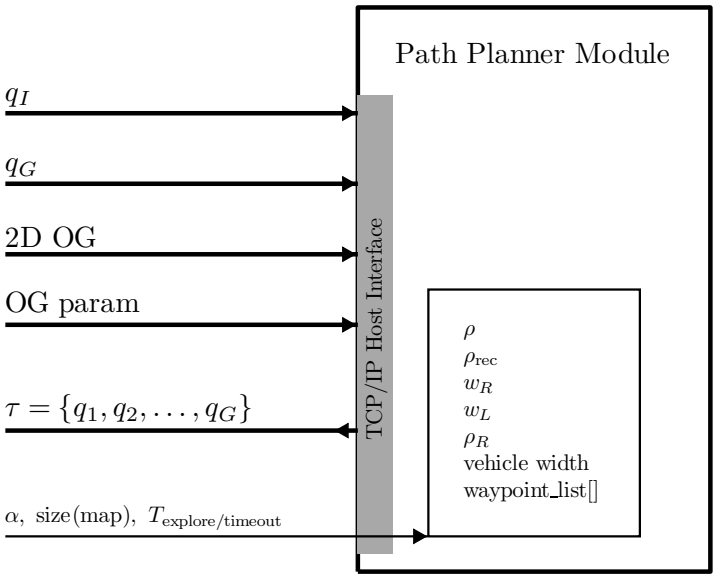


Figure 5.6: Block diagram of path planner interface

The path planner host follows the implementation in Algorithm 5.4.

5.4.2 Findings

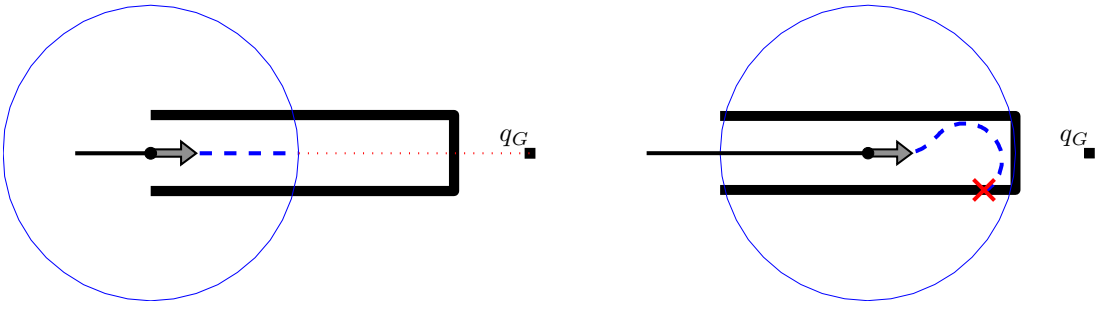
The adapted RRT finds feasible solutions in a variety of obstacle scenes (see Appendix D for an example). The use of waypoints significantly improved the execution time of successive queries, which in turn results in more time to find more optimal solutions. It was found that the optimisation tends to insert intermediate vertices on the path, that

Algorithm 5.4 Path planner host

1: Initialise internal variables
2: Accept connection {α_{xy}, Map dimensions and $T_{\text{exploration time}}$}
3: **loop**
4: Receive planning request {q_I, q_G, 2D OG and extents}
5: Transform to map axis system
6: Compute drivability map {$\mathcal{C}_{obs}$ and risk region}
7: Run RRT Planner {Algorithm 5.2}
8: Trim redundant vertices from solution path
9: Transform to NE-axis system
10: waypoint_list← solution path
11: **return** waypoint_list
12: **end loop**

only account for minute cost improvements. In practice, these would make little difference to the path the vehicle drives. After a few iterations, the number of vertices can grow substantially and are often clustered together. In line 8 of Algorithm 5.4 the list of vertices is scrutinised for very short edges and these are removed.

Since the vehicle has been constrained to forward driving, it is simple to construct obstacle scenes that can cause the vehicle to become trapped in a dead end without sufficient free space to turn around (Figure 5.7). Including reverse motion in the path planner will therefore greatly enhance its usability.



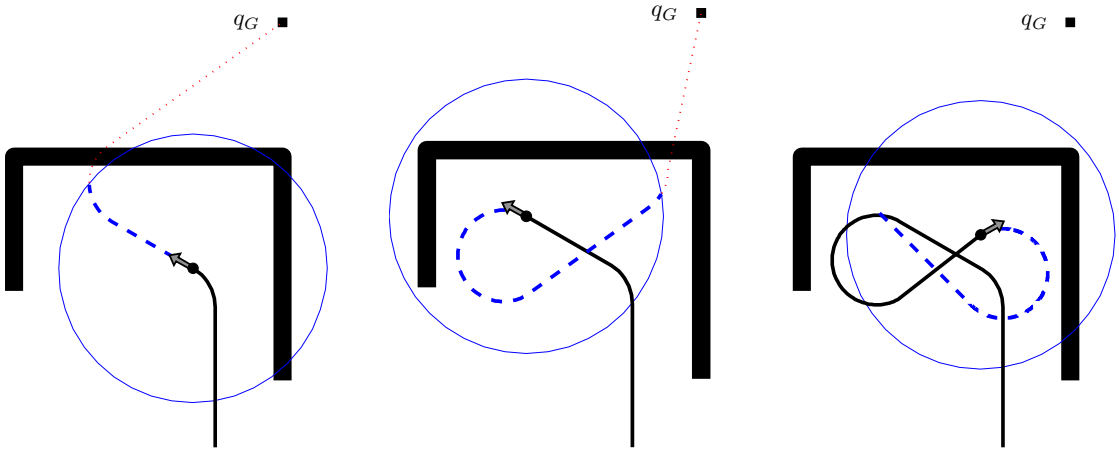
(a) Enter a corridor. Dead end outside viewing horizon.

(b) Dead end within viewing range. Trapped.

Figure 5.7: A dead end obstacle region traps the vehicle if it cannot drive in reverse

The path planner viewing horizon is confined to the near vicinity of the vehicle. This has the effect that the path planner "forgets" obstacles that leave its vicinity. When the vehicle has been forced to take a wide detour (Figure 5.8a), the original obstacle leaves its viewing horizon (Figure 5.8b). On a significant detour, the path planner may attempt to return to the original course, only to rediscover the same obstacle and take the

same detour path (Figure 5.8c), creating an infinite loop. A proper route planner should recognise this situation on a global map and reroute.



(a) Evade obstacles visible in viewing horizon

(b) Original obstacles no longer visible. Turn back

(c) Rediscover obstacles. Close infinite loop

Figure 5.8: Obstacles that extend beyond the path planner view cause infinite loops. This can be dealt with in the route planner, which considers the global map.

The simulation of the path planner in MATLAB^{TM} returns usable paths, with a median execution time of 20 ms for the RRT planner on a set of typical scenes that was constructed for simulations. With $T_{\text{exploration}} = 100\,\text{ms}$, the paths generated are smooth and maintain a good relation with respect to the risk area, even in cluttered environments. The practical system has some additional overhead and returns a path in approximately $250 - 500\,\text{ms}$.

The vehicle keeps on moving during the computation time and thus the planned path may be outdated by the time the solution is returned. To account for the delay, the vehicle position is propagated with the expected computation time and this position is set as the initial configuration in the path planner. In this way, when the path is returned, the vehicle is approximately at the origin of the path.

5.5 Path Planning Summary

An RRT sampling based planning strategy was chosen to cope with vehicle kinematic constraints. It is explained that if the speed of vehicle is limited, the vehicle can be approximated as a Dubins car, for which there exists optimal shortest-path holonomic trajectories. If the vehicle is assumed to be driving forward, some simplifications can

be made in the calculation of the obstacle region, which lowers the dimension of the configuration space. In this case rotation can be neglected. It is remarked that these assumptions are not required for the RRT planner, but are used here to improve execution time.

The standard RRT planner has been adapted to effectively find near optimal solution paths. Included in the cost function is path distance and an associated risk factor. Regions of risk are defined about obstacles and where the map is not yet properly discovered. This makes good use of information provided by the mapping module. This has the effect that paths will generally avoid these regions, but can still pass through them if essential to the solution.

The RRT planner module was tested in simulation. It was found that most solutions can be obtained in well under 0.5 s, which is satisfactory for a real-time system. It is noted that a planner with a local planning search space can easily become trapped in compound obstacle regions due to the kinematic constraints. Future work should include reverse driving manoeuvres, implementing the dynamic constraints and interfacing to a route planner.

Chapter 6

Vehicle Controller

The vehicle controller orchestrates the actuators on the vehicle to follow the safe path generated by the path planner. In its simplest form, the vehicle controller is responsible for tracking the path and velocity references it receives from the mission controller, using the actuator systems on the vehicle.

6.1 Architectural Relation

The vehicle controller is subordinate to the mission planner (Figure 6.1). The mission planner can give the vehicle logical commands to start and stop driving. The vehicle reports its state estimation information to the mission planner at regular intervals. To start following a path, the mission planner relays the waypoints generated by the path planner to the vehicle controller and signals the controller that it may start. Velocity reference values are added to the waypoints.

It is required that the path starts at the current location of the vehicle and is constrained to the abilities of the vehicle. The vehicle controller can correct for minor differences between the reference and vehicle constraints, provided that it has some actuator headroom.

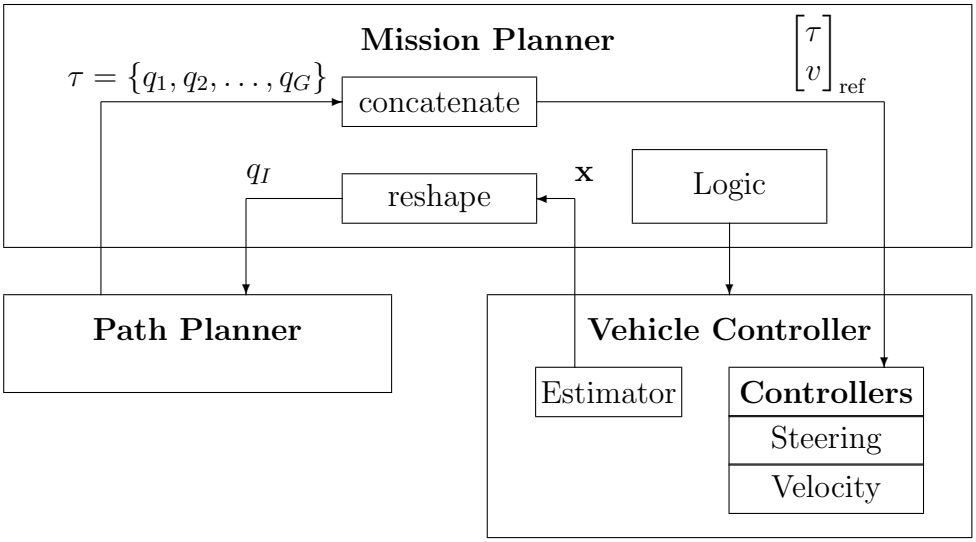


Figure 6.1: Block diagram of the relation between the mission planner and the vehicle controller and path planner

6.2 Vehicle State Estimator

In order for the vehicle to know its whereabouts in relation to the reference path, the on-board controller includes a kinematic state estimator that updates at 50 Hz. The estimator system takes measurements from a variety of sensors and applies a real-time extended Kalman filter (EKF) that combines the data to obtain a least-squares best estimate of the current vehicle state. The sensors available to the state estimator are listed in Table 6.1. The estimated vehicle state vector is

$$\mathbf{x} = \begin{bmatrix} P_N & P_E & P_D & V_N & V_E & V_D & \Psi & \Theta & \Phi \end{bmatrix}^T, \tag{6.2.1}$$

consisting of three positions, three velocities and three angles in the inertial axis system.

Table 6.1: Estimator sensors and descriptions

Sensor	Description of measurement
3-axis Accelerometer	Acceleration in body axis. Requires correction for gravitational vector.
3-axis Gyroscope	Accurate angular velocities about body axis.
3-axis Magnetometer	Approximate orientation, relative to earth's magnetic field.
DGPS	Accurate NED position. Velocity in each dimension. Maximum 10 Hz update rate.
Wheel encoder	Forward speed in body axes. Subject to wheel slip.

6.3 Velocity Controller

The vehicle has three controls that are required to control velocity, namely throttle, gear lever and brakes. The different controls have to be used in unison to achieve and maintain the desired velocity. To ensure the correct use of the controls, a finite state machine is implemented to orchestrate the process. Although only forward driving is used in this thesis, reverse is easily included in the state machine. The state machine selects the appropriate feedback controller, based on the vehicle state.

6.3.1 Velocity Controller State Machine

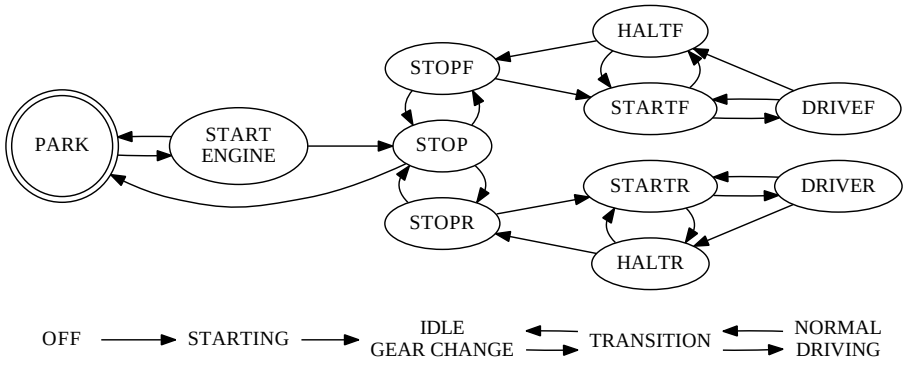


Figure 6.2: Velocity controller finite state machine for bi-directional driving. Group descriptions are indicated below the relevant states.

A mission begins and ends with the vehicle in the PARK state. To begin, the engine is started and if successful, enters the STOP state, idling in neutral. From here the procedure is split into two symmetric branches for forward driving and reverse driving (add suffix "-F"/"-R" to state names respectively). When the vehicle receives a forward velocity reference, it issues the forward gear change command and waits in the STOPF state. Upon completion of the gear change, the parking brakes are relaxed and the throttle controlled to pull away using the STARTF state.

The STARTF state controls the transitional phase, defined as a speed envelope, when the centrifugal clutch is not fully engaged. If the vehicle speed exceeds the transitional envelope, normal driving continues in the DRIVEF sate. It will re-enter STARTF if the speed falls back within the transitional evelope. The STARTF state may also be used to drive at reference speeds below the minimum speed for the clutch to engage.

When a zero velocity reference (or reverse) is received while driving, the HALTF state brings the vehicle to a stop, and when it is stationary, enters the STOPF state and sets the brake. From here it may initiate the symmetric sequence for driving in reverse, or be commanded to PARK and switch off. Passing through STOP issues the neutral gear change command. If during any time the engine should cut out, a zero velocity reference is assumed which brings the vehicle to STOPF, STOP and then PARK.

6.3.2 Velocity Feedback Controllers

From the state machine it can be seen that there are three driving states per direction, each requiring a different control strategy. Since the forward and reverse gears of the ATV have slightly different gear ratios, the forward controller gains will need to be adjusted for use in reverse.

During normal driving, the clutch is fully engaged and the system model is simply throttle input which results in a torque to the drive wheels, accelerating the vehicle inertia. The engine can provide adequate acceleration force and limited deceleration. On a steep descent, or if the velocity reference is quickly decreased, some additional braking may be required. The brakes are activated based on a maximum velocity error margin and when the throttle saturates at the closed position.

In the transitional START phase, the engine is not directly connected to the drive wheels. The amount of torque transferred, via the centrifugal clutch to the drive wheels, is a non-linear function of the engine RPM. It is approximately linear after the clutch makes contact at a minimum RPM. An outer-loop controller with velocity feedback commands acceleration force and uses the inverse torque function to calculate the RPM reference. The inner-loop controller with direct RPM feedback tracks this reference.

A minimum RPM reference keeps the clutch in contact at all times to avoid large non-linearities. To track slow velocity reference commands, the brakes are kept in contact, since their response time is much quicker than the engine RPM controller. Prolonged use of the START state will cause unnecessary wear of the clutch and is therefore timed out.

To stop the vehicle during the HALT state, the throttle is completely closed and maximum safe braking force is applied. This prevents the minimum RPM reference command of the START state during deceleration in the transitional speed envelope.

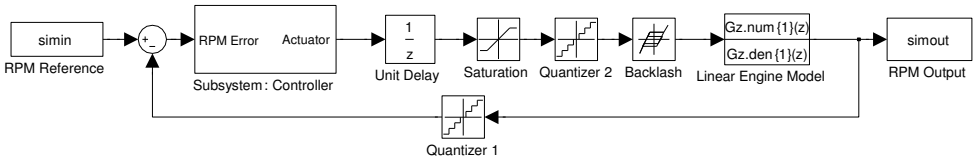


Figure 6.3: An example of a controller simulation model for RPM controller

The discrete controllers are designed using the MATLAB^{TM} control toolbox. The actuator to output dynamics of the vehicle are logged while providing input that attempts to excite the relevant modes of the drivetrain. A non-linear model is derived by fitting the appropriate order model to the logged data [19]. The model is linearised at the nominal operating points and stabilised in a root-locus analysis design. The discrete controller design is simulated with the non-linear model (Figure 6.3) before it is implemented on the on-board controller.

6.4 Steering Controller

To drive along the path returned by the path planner, the vehicle controller has to steer appropriately to remain on track. This controller is burdened with task of following trajectories that were created using approximate vehicle constraints.

This controller is implemented using the cross-track error, x, indicated in Figure 6.4, and the difference in heading between the path and vehicle, ψ, to set the steering angle reference command

$$\phi = \psi + \arctan\left(\frac{kx}{U}\right), \tag{6.4.1}$$

where k is a positive gain and U the vehicle speed [7].

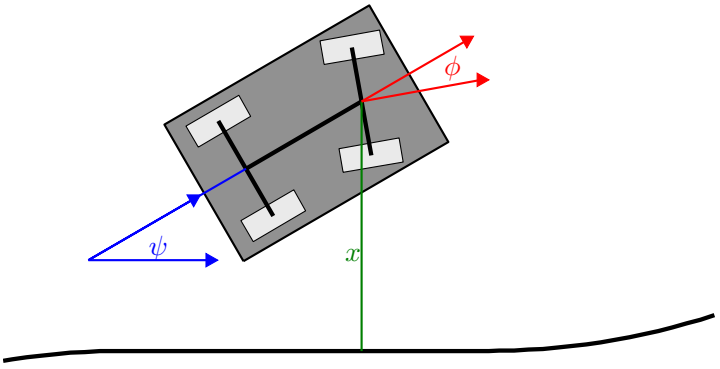


Figure 6.4: Construction for steering controller, based on cross-track error and heading

The first term sets the steering angle reference parallel to the path trajectory. The second term turns the reference towards the path as a function of cross-track error. The arctan function saturates at $\pm 90^\circ$ when x becomes large, which relates to the wheels pointing straight at the path when the vehicle is far off track. The authors of [7] have proved that equation 6.4.1 results in convergence with exponential decay for a continuous ideal car with small cross-track error. The gain, k, adjusts the rate of convergence and the inclusion of drive speed, U, makes the rate of convergence independent of speed.

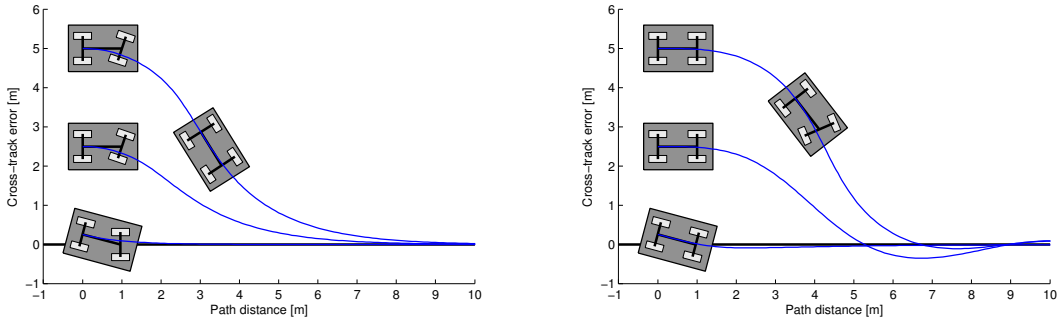
The minimum turning radius limits the value of k. Setting k too high generates a theoretical trajectory where the cuvature exceeds the turning radius and results in overshoot. A ballpark figure can be obtained by substituting typical values for ϕ, x and ψ, based on the turning radius, and a given U in equation 6.4.1 and solving k. The curves in Figure 6.5a are created with $U = 2\,\mathrm{m/s}$ and $k = 1.5$ for $\rho = 3\,\mathrm{m}$.

Some adjustments are needed for a practical controller. The discrete controller has finite response time and the steering actuator has inertia and a maximum slew rate which detracts from the stability of the control law. In [7] it is suggested to add some damping to the error signal. For the same k and U as before, damping was added to x and the resulting dynamic trajectories are shown in Figure 6.5b.

The coefficients of the damping filter can be computed by modelling slew rate as an appropriately slow pole and choosing stable closed-loop poles on the resulting root locus given a fixed gain. This approximation works well with small errors. Simulation is used tune the parameters to wider range of initial errors. It is noted that without proper damping, the dynamic system easily becomes unstable. The filter coefficients are dependent on k and therefore sensitive to velocity. The coefficients were designed for the minimum ATV velocity.

6.4.1 Steering Control Simulation

The discrete control law is tuned using a simulation that models the kinodynamic effects of the steering actuator system, including slew rate and saturation. The simulation is used to validate the amount of headroom the path planner should leave for the vehicle steering controller.

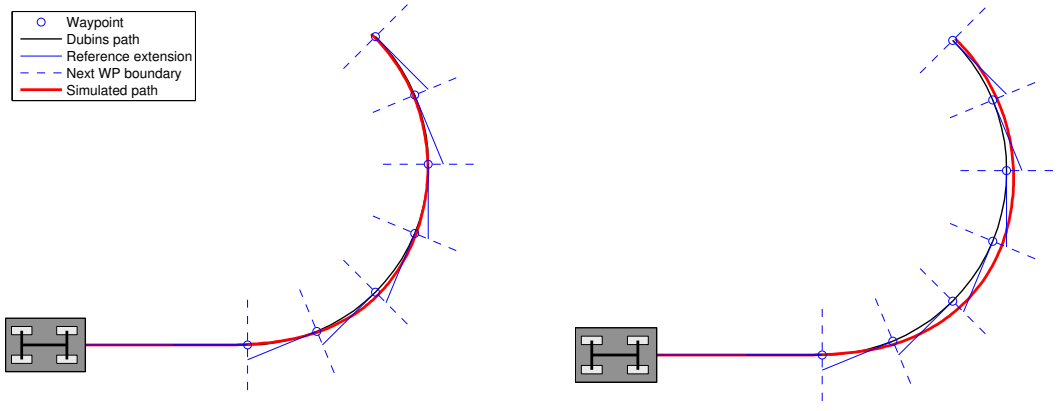


(a) Trajectories of a continuous controller without slew rate. Stable for all cross-track errors. Note that the front wheels are fully turned at the first instant, compared to (b).

(b) Trajectories of a discrete controller with limited slewrate. x is critically damped with a finite impulse response filter for small x. Note that the front wheels turns over time.

Figure 6.5: Typical steering controller trajectories for various starting conditions

The path planner outputs a set of waypoints that consists of NE coordinates and heading, Ψ. The cross-track reference line is generated by expanding each waypoint into a line segment that passes through the waypoint at the given heading. The steering controller selects the next segment as its reference when it passes the current waypoint. Figure 6.6 shows consecutive waypoints and their extensions, the ideal Dubins car path that connects them and the true path resulting from the steering controller.



(a) Simulated path at nominal design speed displays good tracking

(b) Simluated path at 130% of design speed displays initial overshoot

Figure 6.6: Simulated path over waypoints that are extended into reference paths.

It is noted that the slew rate of the steering system results in initial overshoot. Through trial and error in simulation, it was found that setting the planner radius to 4 m, with waypoints around the bends at intervals between 15° and 30° apart, gives good tracking at the minimum design speed.

Increasing the speed results in considerably more overshoot, as expected (Figure 6.6b) and proves that the simple path planner is only valid at low speeds. An experimental simulation was conducted where the next waypoint is selected as reference, some distance before the current waypoint is reached. This early selection allows the controller to "look ahead" to compensate for slew rate, but only delivered minor improvements.

The use of simple waypoints to represent curves is motivated with the following formulation:

1. Each of the sections in the path resembles the curves in Figure 6.5.
2. Assume a successful steering controller passes each waypoint with negligible cross-track and heading errors,
3. thus a waypoint can be seen as the initial condition for the steering controller in terms of the *following* section.
4. If the initial condition (the previous waypoint) for a section lies on a true trajectory that eliminates cross-track error before it reaches the waypoint, the assumption in line 2 holds.

The condition in line 4 is satisfied within the path planner constraints. The formulation holds and proves that the simple waypoints provide an adequate representation of the path.

6.5 Vehicle Controller Summary

The vehicle controller receives a list of waypoints from the mission planner. Each waypoint consist of 2D coordinates, a reference heading and a velocity. It is the task of the steering controller to track the path indicated by the waypoints. The velocity controller maintains the reference speed for the current waypoint.

The velocity controller consists of a state machine and feedback controllers. The state machine schedules the gear shifts and throttle and brake feedback controllers. There are parked states, gear changing states and driving states. The three driving states separates the low speed, normal driving speed and stopping situations to apply the relevant feedback controller.

The steering controller is based a non-linear feedback function of cross-track and heading error. The feedback function guarantees stability for unconstrained systems and exponential error decay when the error is small. Damping the cross-track error compensates for the dynamic constraints of the test vehicle. The simulation of the steering controller displays good tracking under the design conditions for speed and path curvature. For increased speed, the reference path should reflect the vehicle dynamic constraints or else considerable overshoot can occur due to the limited actuator response time.

Chapter 7

Practical Testing

The simulated systems were integrated into a real-time navigation and control system. The mission goal is to drive through each waypoint in a fixed list of user-defined waypoints, while evading simple obstacles. This chapter first describes the integrated structure and then the test scenarios used to validate the navigation system. This is followed by a discussion of the test drive results.

7.1 System Structure

The integrated system structure is shown in Figure 7.1. The top-most level is the combined mission planner and route planner. Directly below the mission planner is the path planner, mapping functions and vehicle controller. All the subsystems act as service providers to the mission planner, i.e. information is requested by the mission planner and the subsystems respond to commands.

7.1.1 Mission Planner

The mission planner is implemented as a MATLAB^{TM} m-script. For these simple tests, no actual planning is performed in the mission planner or route planner. The mission planner is used to initialise the subsystems and coordinate data transfer between these systems. When all the systems are ready, it waits for user confirmation to start the mission. During

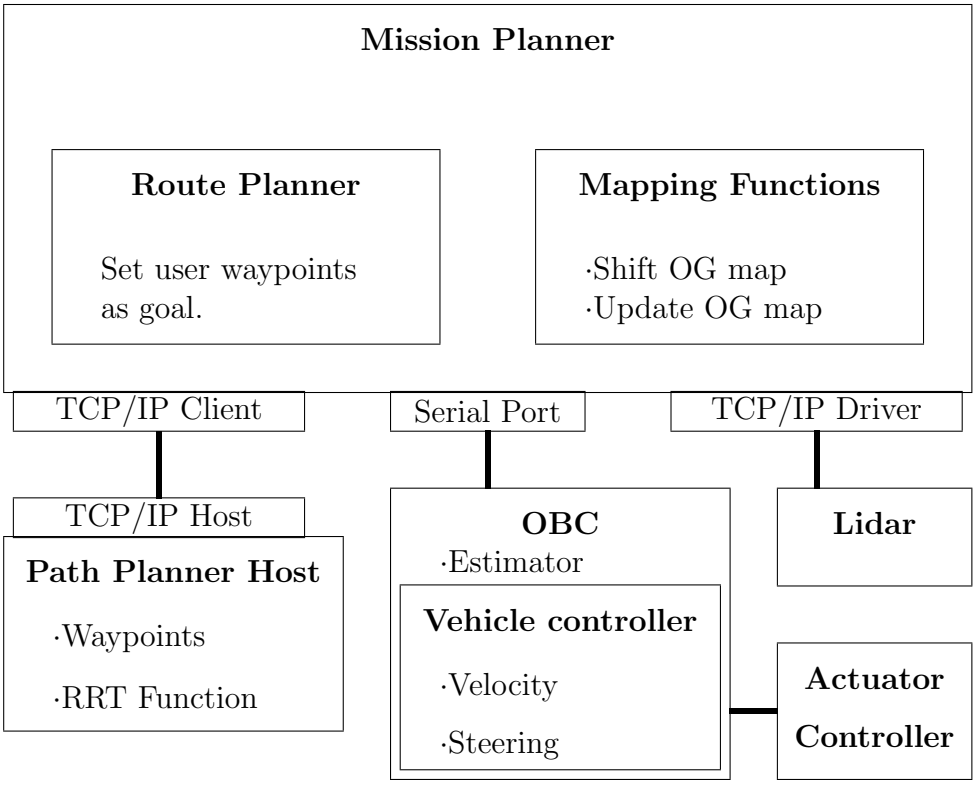


Figure 7.1: Hierarchy of modules used during testing

execution, it monitors the mission progress of the route planner and stops the mission when it has been completed.

7.1.2 Route Planner

The route planner is integrated into the mission planner and feeds the current waypoint from the user waypoint list to the path planner as an intermediate goal state. The next waypoint from the list is selected when the current waypoint is reached. This condition is relaxed to a radius around the waypoint to allow a tolerance for cross-track errors and to allow planning ahead for the next waypoint. When the last waypoint is reached, the reference speed is set to zero, which will bring the vehicle to a halt. For all other waypoints, a constant speed is set that is equal to the minimum vehicle operating speed.

7.1.3 Mapping Functions

The mapping functions are called directly from the mission planner script. The variable storage space for mapping is defined in the mission planner scope to simplify data

transfers between modules. The mission planner receives the vehicle state from the vehicle controller and requests a new scan from the lidar device. The lidar connects using an ethernet interface. A driver function that connects the TCP/IP network and parses the incoming data into MATLAB™ variables were written in C and compiled to the MATLAB™ executable "mex" format. The 3D OG is first shifted to remain local to the vehicle position and then updated using the (sub-sampled) lidar scan data.

Only when a new path planning request is submitted, is the 3D OG collapsed to a 2D drivability map. For efficiency, this is performed only on the portion of the map that is relevant to the path planner viewing horizon. The mapping algorithm is computationally intense and uses all the available computational power that MATLAB™ can source and therefore dictates the loop frequency of the mission planner.

7.1.4 Path Planner Host

The path planner host is also implemented as a MATLAB™ m-script. It receives initial parameters and planning requests via a TCP/IP host interface, shown in Section 5.4.1. Both sides of the interface (host and client) are written in C and compiled to the "mex" format. When the host receives a request, it runs the RRT planner m-function on the local 2D OG and returns the solution path on the TCP/IP interface. It also keeps a local copy of the waypoints for the next iteration.

The custom interface between mission planner and path planner, based on a standard computer network, allows these processes to execute in parallel, in separate MATLAB™ sessions. These sessions may either be on separate networked computers, or on the same computer if it has multiple processing cores. This allows the path planner and mapping functions to execute at their respective maximum rates.

7.1.5 Vehicle Controller

The vehicle controller is implemented on the standard on-board controller (OBC) available in the AutoNav group and connected to the mission planner via a serial connection. The mission planner can issue various commands to the vehicle controller that starts and stops

a mission and uploads the most recently computed set of waypoints. The mission planner receives the vehicle state and can poll some general status information from the vehicle.

The OBC is built around a 16-bit microcontroller (dsPIC30F6014A from Microchip). The design includes various peripherals to interface with the IMU, DGPS, actuator controller, mission planner and also a ground station. The OBC is accompanied by skeleton firmware that reads data from the various sensors and executes the EKF to obtain the vehicle state estimate. The OBC logs all data to a digital memory card. The vehicle controllers in Chapter 6 were integrated into the OBC framework.

7.2 Test Setup

Testing was carried out in a controlled environment to ensure safety and maintain the integrity of the tests. The ability of the vehicle controller to follow a reference path was tested first and then the complete system was tested.

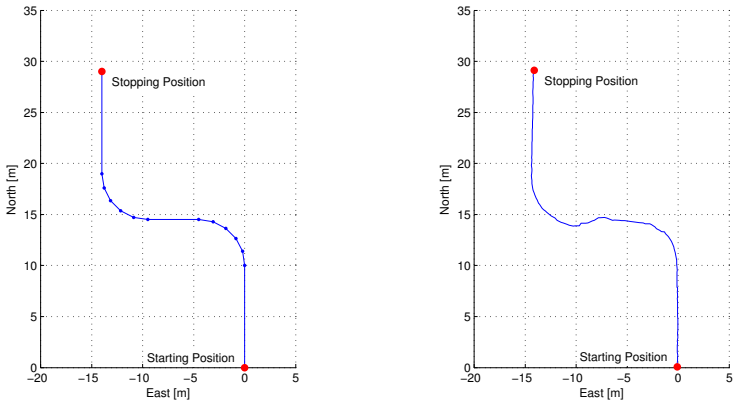
7.2.1 Steering Controller Test

To test the steering controller independently, the vehicle controller is initialised with a fixed set of waypoints using a mission planner emulator (see Appendix B). The path was chosen to illustrate the ability of the vehicle to follow both a straight segment as well as a curved segment. The reference path and waypoints are shown in Figure 7.2a and the actual ground track obtained is shown in Figure 7.2b.

Unfortunately, the differential GPS was not available during this test. The lower accuracy of ordinary GPS causes false cross-track errors (note the irregular curve in Figure 7.2b) which, combined with the high frequency gain of the damping filter, causes large unwanted steering commands. It was decided to remove the damping filter from the cross-track error and instead use yaw rate (which is unaffected by GPS errors) to damp the controller with a response similar to the original controller.

The steering controller test also showed that the vehicle turning radius is slightly wider than was designed for. This is attributed to the solid rear axle (refer to Section 5.2.1.1).

The lack of actuator headroom will likely cause some tracking errors on curved paths, unless the curve radius constraint is relaxed.



(a) Waypoint set to track (b) Actual GPS track

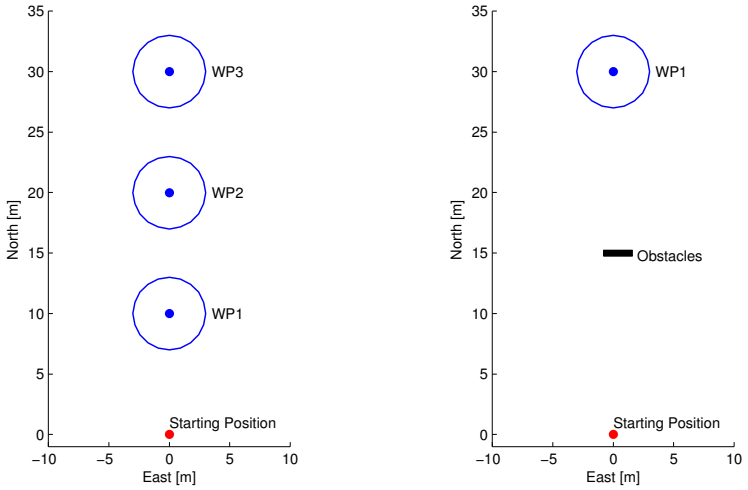
Figure 7.2: Vehicle steering controller test path and track

7.2.2 Integrated Test

The complete system was tested using two simple sets of user-defined waypoints, shown in Figure 7.3. The first set has three waypoints, 10 m apart in a straight line with no obstacles, and serves to verify the mission planner and its cooperation with all the subsystems. The second set is a single waypoint, but with an obstacle placed deliberately on the path connecting the starting position and goal position, to force the vehicle to display evasive action. Unfortunately, the size of the test space limited the scale of testing that was possible.

7.2.3 Velocity Controller

For safety reasons, the velocity controller was not tested during the navigation tests. Although all the required actuators were functional, the engine remained switched off and the vehicle was pushed by hand during tests. Since the brake actuator was active it maintained the reference velocity as set by the mission planner, despite attempts to push the vehicle faster. This also stopped the vehicle when the goal was reached.



(a) Waypoint set one (b) Waypoint set two

Figure 7.3: User-defined waypoints that were used to test the integrated system

If the engine is not turned on, the velocity controller state machine is not able to leave the PARK state. The state machine was forced to advance permanently to the STARTF state, which is appropriate for the speeds used during testing.

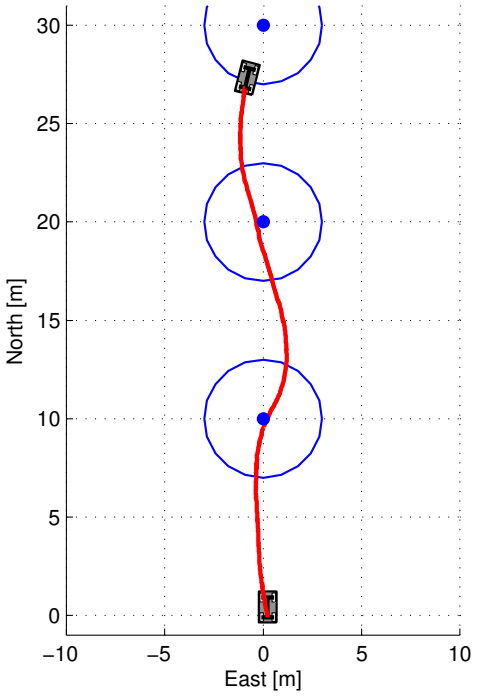
It was also found that the presence of a safety driver near the antenna of the GPS system significantly reduced its reception quality and caused loss of differential accuracy.

7.3 Integrated Test Results

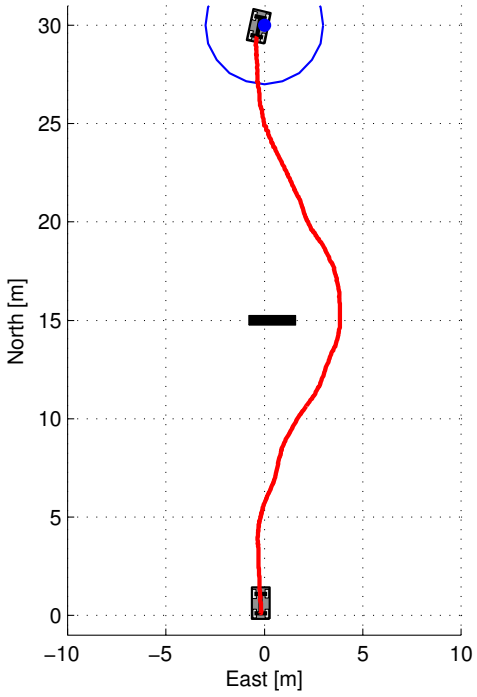
Four test runs were made using the test setup described in Section 7.2.2. Test One has no obstacles and the remaining three tests were conducted with a single obstacle. The tests were conducted on the lawn nearby the research labs. Although seemingly even, the lawn has considerable irregularities, which causes pitch and roll angles of up to two degrees and 25 cm of elevation difference.

7.3.1 Test One – No Obstacles

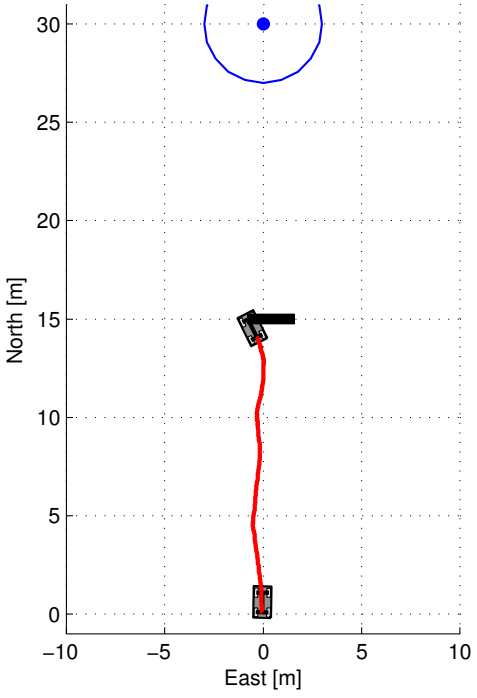
The ground track is shown in Figure 7.4a and it is clear that the vehicle did indeed reach all three waypoints. Keep in mind that the next waypoint is selected as soon as the vehicle enters the radius around the current waypoint. Between the first and the second waypoint, the vehicle swerved 1.2 m to the right, which may seem strange given that there are no



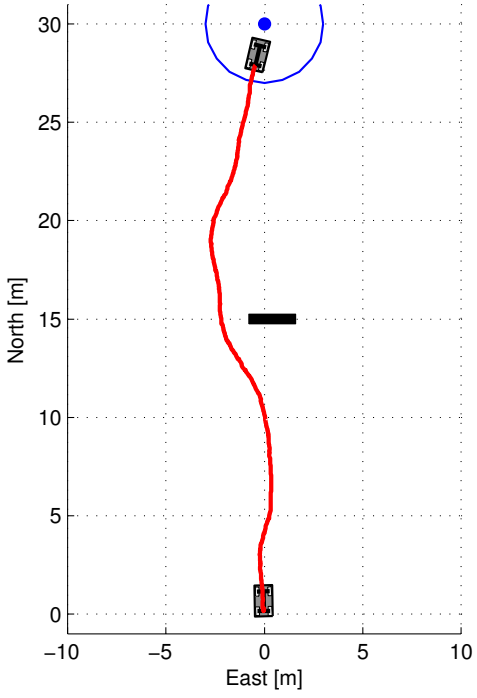
(a) Test One: The route waypoints and the radius about them are indicated on the route.



(b) Test Two: The goal area and approximate obstacle area are indicated on the track.



(c) Test Three: The late change in evasive direction caused a collision.



(d) Test Four: The goal area and approximate obstacle area are indicated on the track.

Figure 7.4: Ground tracks from practical test drives

obstacles. The reader is however reminded that considerable priority has been given in the RRT to follow "safe" (well mapped) paths. Should the motion of the vehicle over irregular terrain (see Section 3.3.2) cause an uneven distribution of scanned profiles, the path planner will actively avoid the under-mapped areas. The vehicle continued normally and stopped at the set radius from the final waypoint, as intended.

7.3.2 Test Two – Single Obstacle

The ground track is shown in Figure 7.4b and the vehicle successfully avoided the obstacle and made its way to the goal. The vehicle starts off heading straight towards the goal. When the obstacle comes within the planning horizon, the vehicle takes a smooth evasive action to the right of the obstacle with ample clearance from it and heads towards the goal.

7.3.3 Test Three – Single Obstacle (Collision)

The ground track is shown in Figure 7.4c and the vehicle was unable to avoid the obstacle and make its way to the goal. The vehicle starts off heading straight towards the goal and at 7.6 mNorth chose a path taking an evasive action towards the right of the obstacle. At 9.7 mNorth the path planner switched to an evasive action passing on the left. Due to the slew rate of the actuator, the steering controller was unable to track the two sudden changes in direction and overshot the planned path by more than 1.2 m, causing a collision with the edge of the obstacle.

It is clear that ignoring non-holonomic constraints can have severe effects on the validity of the planned paths. It should also be noted that the method used to propagate the vehicle state in the mission planner ignored the dynamic effects of the steering system. This causes discrepancies between the actual vehicle state and the state used for path planning, especially during sudden direction changes.

During the last 4 m leading up to the collision, the path planner was unable to find valid solution paths, since the vehicle had already entered the area of inevitable collision. Normally the vehicle would enter an emergency stopping mode in this situation, but this feature was disabled to simplify manual testing procedures.

7.3.4 Test Four – Single Obstacle

The ground track is shown in Figure 7.4b and shows that the vehicle successfully avoided the obstacle and made its way to the goal. The vehicle starts off headung straight towards the goal. When the obstacle comes within the planning horizon, the vehicle takes a smooth evasive action to the left of the obstacle with ample clearance and heads towards the goal. After passing the obstacle, it makes another deviation towards a "safe" area, similar to that of Test One. This is attributed to the fact that the area directly behind the obstacle was occluded from the sensor view on approach of the obstacle. This area is less likely to be mapped as well as the area towards the sides and is therefore avoided as expected.

7.3.5 General Observations

The accuracy of the estimation system is limited by the accuracy of the measurements. Due to time constraints and various technical issues, only a limited calibration of IMU sensors were performed. For instance it is not possible to fully calibrate the magnetometer on the vehicle, as it is only possible to rotate the vehicle about the yaw axis. Furthermore, zeroing the sensors biases requires the vehicle to be orientated perfectly level - a state which is difficult to ensure during field tests. The uneven nature of the testing terrain, coupled with slight attitude estimation errors, can easily cause some of the higher portions of terrain to be erroneously classified as obstacle regions.

The steering actuation motor often failed during the preparations for testing, due to problems with its brushes not making proper contact, which were aggravated by the generated heat from high control currents. For this reason the current limit was lowered considerably for the final tests, which slows its dynamic response time. These dynamics were not modelled in the steering controller and decreases the steering controller stability.

The system was reset before each test. This implies that there is no prior mapped data available to the path planner. It is therefore not allowed to start a test with an obstacle closer to the vehicle than the viewing horizon of the lidar, since it had no opportunity to map such objects. To overcome this, it is possible to to enable the mapping module and manually driving a few metres as a means to initialise the map, before enabling the automatic vehicle controllers.

7.4 Tests Summary

Practical navigation tests were performed in a controlled environment. The tests were set up to show the feasibility of the complete system and to investigate the integrated interaction between the different navigational modules. For safety reasons and to simplify the testing procedure the velocity controller were overridden and the vehicle pushed by hand.

Three of the four tests showed successful automatic navigation. Successful navigation testifies that all the subsystems function satisfactory. The failed test is attributed to approximations made during the design process, namely neglecting the steering slewrate and dynamics in the path planner and the mission planner, which proved to be inadequate. With the current framework created during this thesis, it should be simple to derive more accurate models.

Chapter 8

Conclusion and Recommendations

8.1 Conclusion

This thesis presented the groundwork that was laid to enable and demonstrate autonomous navigation of a terrestrial vehicle and demonstrated the integrated system with practical tests. All the necessary systems for a full demonstration were designed. This required:

- Actuating the selected vehicle (ATV) with electric actuators and designing appropriate controller hardware
- Investigating the capabilities of a scanning lidar for optimum 3D mapping potential
- Creating a visual simulation for a lidar device that scans within a virtual 3D environment and outputs lidar datasets
- Applying the occupancy grid method on a three-dimensional grid, paying special attention to efficiently projecting multiple lidar beams into a local 3D OG in real time
- Developing a path planner based on the RRT, using the simple car kinematic model for computational efficiency
- Designing a vehicle controller for path following and setting up the velocity controller state machine specific to the vehicle

- Integrating the mapping module, path planner and vehicle controller with a basic mission/route planner for real time execution during practical tests.

The all terrain vehicle is fully actuated (steering, throttle, brakes, gear shift) and the actuator board provides inner-loop feedback control for position reference commands. The additional sensors (wheel-speed and tachometer) provides convenient and accurate data to high-level controllers. With safety features designed into all critical components, this vehicle is a valuable, ready-to-use test bed for future research.

The 3D OG mapping software performed well in simulation and in practical tests using a single lidar device. It is capable of mapping three dimensional obstacles reliably in real time. For best results, the use of an accurate state estimator, especially with respect to attitude, is a crucial requirement. The simple threshold for obstacle regions, based on height relative to the vehicle, works well while driving on even surfaces, but does make some false detections when applied to uneven terrain. The mapping process is therefore ready for most on-road situations and with intelligent thresholding, should work over moderately uneven terrain. It is, however, not well suited to mapping the driving surface, as the surface tends to contain ridges as a result of successive profile scans.

The RRT path planning method showed promise for use as a real-time path planner. Even when neglecting prominent vehicle steering dynamics, the path planner was able to create usable trajectories. It is clear however, that reliable path planning will only be achieved with proper regard for vehicle dynamics, including those of the vehicle controller. The RRT method does make provision for complex vehicle dynamics and is therefore still the author's method of choice.

It is the author's opinion that this project was successful at automating a terrestrial vehicle and implementing the complete framework for automated navigation. Areas of focus were: vehicle actuation, lidar mapping, path planning and the vehicle controller. The system reliability can easily be improved with the recommendations listed in the next section. With the framework in place, future research can focus on singular aspects, e.g. improving the path planner, and then test the upgraded component in the system.

8.2 Recommendations

Section 8.2.1 gives the recommendations for initial improvements, based on the results given in this thesis. It is followed by Section 8.2.2 which provides suggestions for future work.

8.2.1 Necessary Improvements

Vehicle:

- Replacing the relatively cheap steering actuator (automotive wiper motor) with a more reliable/powerful actuator will remedy many steering-related issues.

Lidar Mapping:

- Measuring the actual lidar sensor model may improve the mapping accuracy, especially with regard to ridges in the driving surface.
- The use of a plane-detection algorithm to identify the driving surface in the 3D OG and using this plane as reference to threshold obstacles, should improve obstacle detection on inclined terrain.

Path Planner:

- Extend the RRT planner to use the full dynamic steering model or a higher order approximation thereof, to obtain plans that are guaranteed to be executable by the vehicle.
- Use a more accurate model to predict the vehicle configuration to use as the initial configuration during planning – include the vehicle controller and actuator dynamics, based on the current reference trajectories.

8.2.2 Future Work

Suggestions for improving the vehicle:

- Replacing the shear-pin with a non-destructive override will ease testing and benefit safety.

- Provision has been made on the actuator board for an analogue input which could be used as input for a test-driver steering override.
- Provision has been made for an electronic cut-out relay, which should be fitted.
- Provision has been made for another hall-effect sensor to provide precise feedback of throttle position – the RC servo displays considerable hysteresis, which can then be suitably compensated for.

Suggestions for improving the lidar mapping:

- Porting the mapping implementation to a more efficient programming language can result in significant performance increase. The mapping algorithm can exploit multi-threaded computing to project multiple beams simultaneously and use GPU computing to accelerate the matrix multiplication to project beams. The performance boost can be used to map at a higher rate or to refine the grid/angular resolution.
- For missions that revisit the same area, implement a global map that initialises the local map.
- Further investigation into actuated mountings as per Section 4.3.1.4.

Suggestions for improving the Path planner:

- Including velocity (as well as forward and reverse driving) in the sample space will prevent the vehicle from becoming trapped.
- Adding another weight to the cost function, based on actuator motion, will result in “smoother” driving behaviour.

Suggestions for improving the vehicle controller:

- Consider using algebraic expressions for curved trajectories, instead of multiple waypoints. These algebraic expressions should honour the actuator dynamics.
- Use gain scheduling for the steering controller to remain stable at variable velocity.
- Replace the simple PID velocity controller with full-state feedback control for precise control over dynamic responses.

Suggestions for improving the estimator data:

- Mounting the IMU and GPS sensor set with the lidar will measure the lidar sensor position directly, reduce electromagnetic interference originating from the vehicle electronics and provide an open view of the sky for the DGPS antenna.

8.3 Final Words

This document describes the successful automation of a terrestrial vehicle (see Appendix E for photos of the vehicle) and the design and implementation of functional navigation modules for a complete terrestrial navigation system. The navigation demonstrations illustrates the appropriateness of the proposed architecture. This concludes the description of the framework which can be used for further research on the topic of autonomous navigation.

Bibliography

[1] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46 –57, jun 1989.

[2] M. Ribo and A. Pinz, "A comparison of three uncertainty calculi for building sonar-based occupancy grids," *Robotics and Autonomous Systems*, vol. 35, pp. 201–209, Jun 2001, 10.1016/S0921-8890(01)00116-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889001001166

[3] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. [Online]. Available: http://ijr.sagepub.com/content/20/5/378.abstract

[5] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. How, "Motion planning for urban driving using rrt," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, sept. 2008, pp. 1681 –1686.

[6] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, 2002, pp. 2383 – 2388 vol.3.

[7] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," in *The 2005 DARPA Grand Challenge*, ser. Springer Tracts in Advanced Robotics, M. Buehler, K. Iagnemma, and S. Singh, Eds. Springer Berlin / Heidelberg, 2007, vol. 36, pp. 1–43, 10.1007/978-3-540-73429-1_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73429-1_1

[8] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little Ben: The Ben Franklin racing team's entry in the 2007 DARPA Urban Challenge," in *The DARPA Urban Challenge*, ser. Springer Tracts in Advanced Robotics, M. Buehler, K. Iagnemma, and S. Singh, Eds. Springer Berlin / Heidelberg, 2009, vol. 56, pp. 231–255, 10.1007/978-3-642-03991-1_6. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03991-1_6

[9] "Operating instructions - laser measurement systems of the LMS100 product family," SICK AG, Erwin-Sick-Str. 1, 79183 Waldkirch, Germany, March 2010. [Online]. Available: https://www.mysick.com/saqqara/pdf.aspx?id=im0031331

[10] H. Hoppe, "Progressive meshes," in *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 99 – 108.

[11] S. Thrun, M. Montemerlo, and A. Aron. (2010) Probabilistic terrain analysis for high-speed desert driving. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.877&rep=rep1&type=pdf

[12] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Autonomous Robots*, vol. 15, pp. 111–127, 2003, 10.1023/A:1025584807625. [Online]. Available: http://dx.doi.org/10.1023/A:1025584807625

[13] I. Peddle, "Autonomous flight of a model aircraft," Master's thesis, University of Stellenbosch, 2005.

[14] "Compact, low power GNSS platform offers flexible positioning options," NovAtel, 4 2011. [Online]. Available: https://www.novatel.com/assets/Documents/Papers/OEMV-1_Series.pdf

[15] W. Hough, "Autonomous aerobatic flight of a fixed wing unmanned aerial vehicle," Master's thesis, University of Stellenbosch, 2007.

[16] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara, "Obstacle avoidance and path planning for humanoid robots using stereo vision," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 1, april-1 may 2004, pp. 592 – 597 Vol.1.

[17] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 4, pp. 498 –501, aug 2001.

[18] P. Bourke. (1991, August) Intersection of a plane and a line. [Online]. Available: http://paulbourke.net/geometry/planeline/

[19] J. Treurnicht, "Parameter estimation," University of Stellenbosch, Feb 2011, course notes. [Online]. Available: http://courses.ee.sun.ac.za/Beheerstelsels_414/images/paramestexamples.pdf

Appendices

Appendix A

Actuator Controller PC Interface

The actuator controller interface shown below is an interactive MATLAB^{TM} graphical user interface (GUI) that is used to test the actuator controller via a serial interface. It can poll (or request a stream of) all the sensor data and command the actuators or set position references for the controllers.

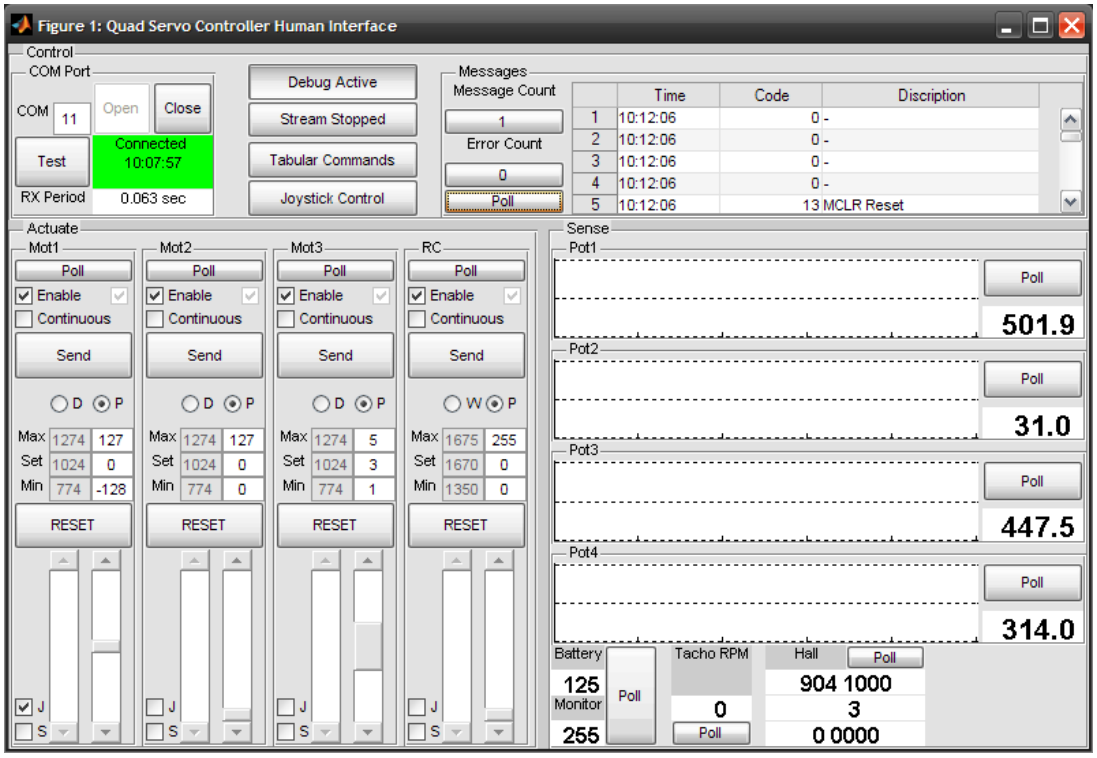


Figure A.1: Actuator controller PC interface

Appendix B

Mission Planner Emulator

The mission planner emulator shown below is an interactive MATLAB^{TM} graphical user interface (GUI) that is used to test the vehicle controller. It polls the status and estimator information at regular intervals and uploads a predetermined set of waypoints, mimicking the interface of the mission planner with respect to the vehicle controller.

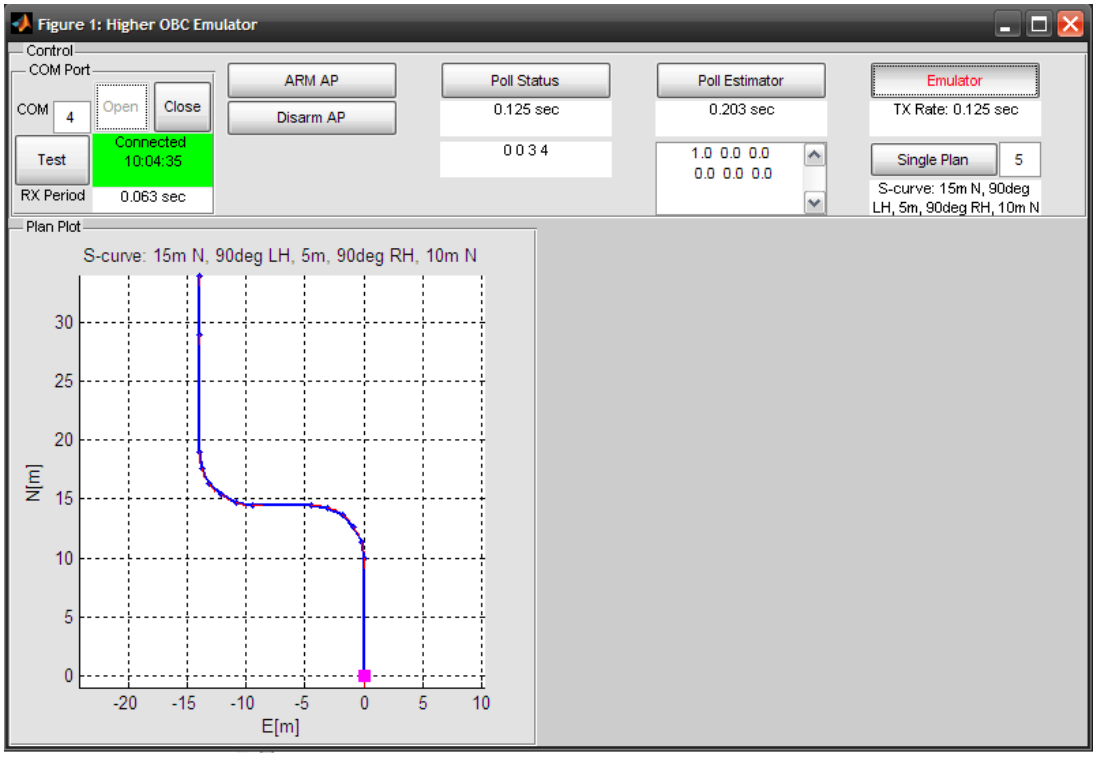


Figure B.1: Mission planner emulator

Appendix C

Inside View of 3D OG

The series of images in Figure C.2 provides a view of the data captured in a 3D OG during a practical test. The map was created in the parking area shown in Figure C.1. Each image represents a horizontal layer of cells in the 3D OG. The vertical height (as a Down coordinate in the inertial axis system) is indicated with each image. Green indicates unmapped area, red indicates occupied cells and blue indicates empty cells, with the approximate vehicle trajectory indicated in cyan. The vehicle was driven in a north-westerly direction. A surface extracted from the occupied cells is shown in Figure C.3.

Additional notes:

- The reader is encouraged to match large items from the photo to the series of images. E.g. the three cars to the left on the photo, visible at the lower-right corner in c-k. Also the elevated kerbs visible in l-n.
- The occupied cells in images p-s is the driving surface – the curvature of the road and gutters (image s) is clearly distinguishable.
- The most recent scan creates a "ridge" in the driving surface, (line of occupied cells at the top-left boundary of cleared area, visible in j-o).
- The slanted angle of the sensor implies a slanted boundary of mapped area – compare the upper-left edge of the empty area (blue) between images a-h.
- DGPS was unavailable for this test, therefore reduced estimator accuracy can be assumed.

Figure C.1: A photo taken from the starting location of the vehicle. The approximate path and final location of the vehicle is indicated.

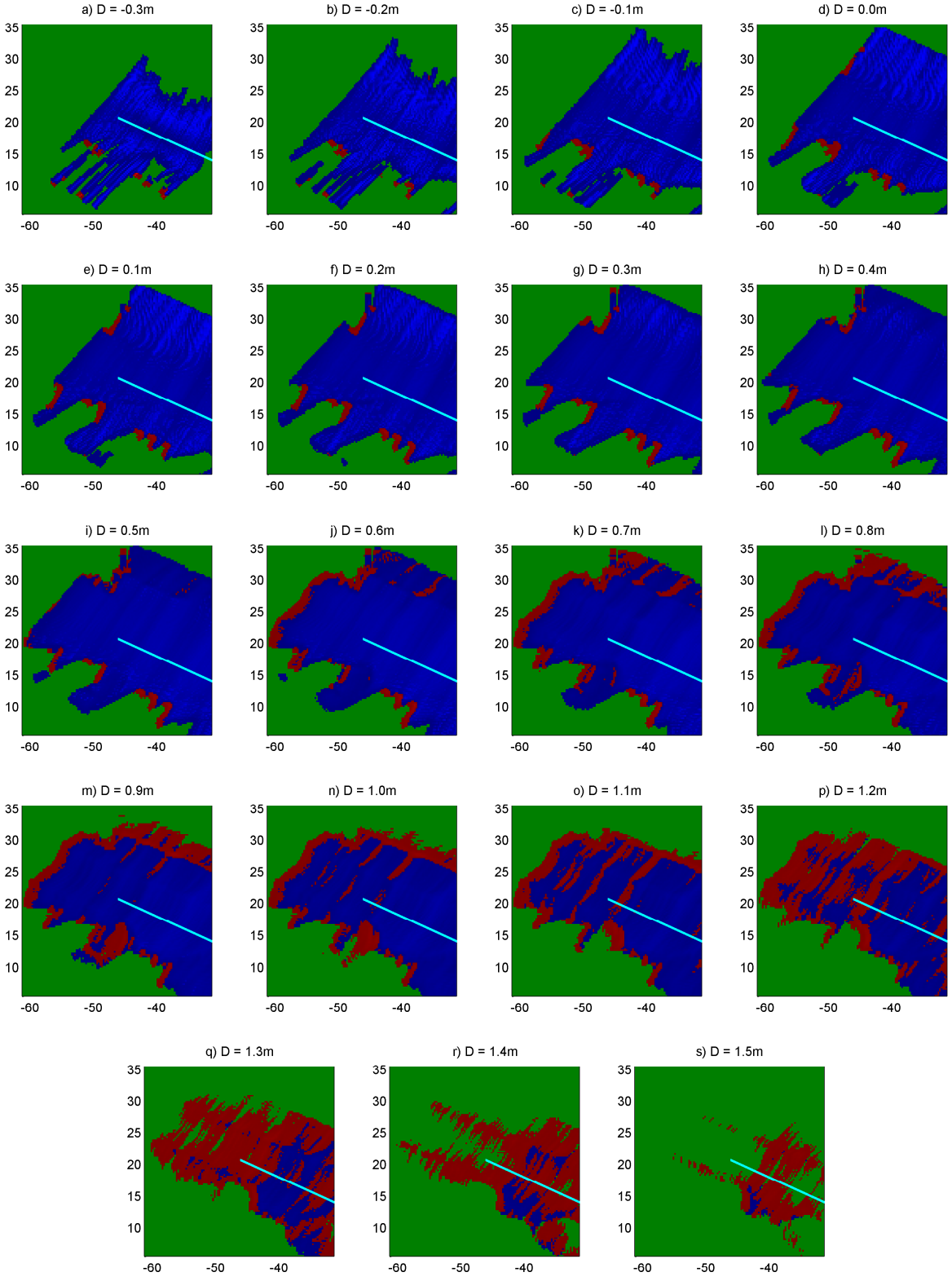


Figure C.2: This series of images show the occupancy state of cells in each horizontal layer in the 3D OG.The approximate path followed up to this point is shown. Axes are in metres with the vertical axis being distance North and the horizontal axis being distance East.

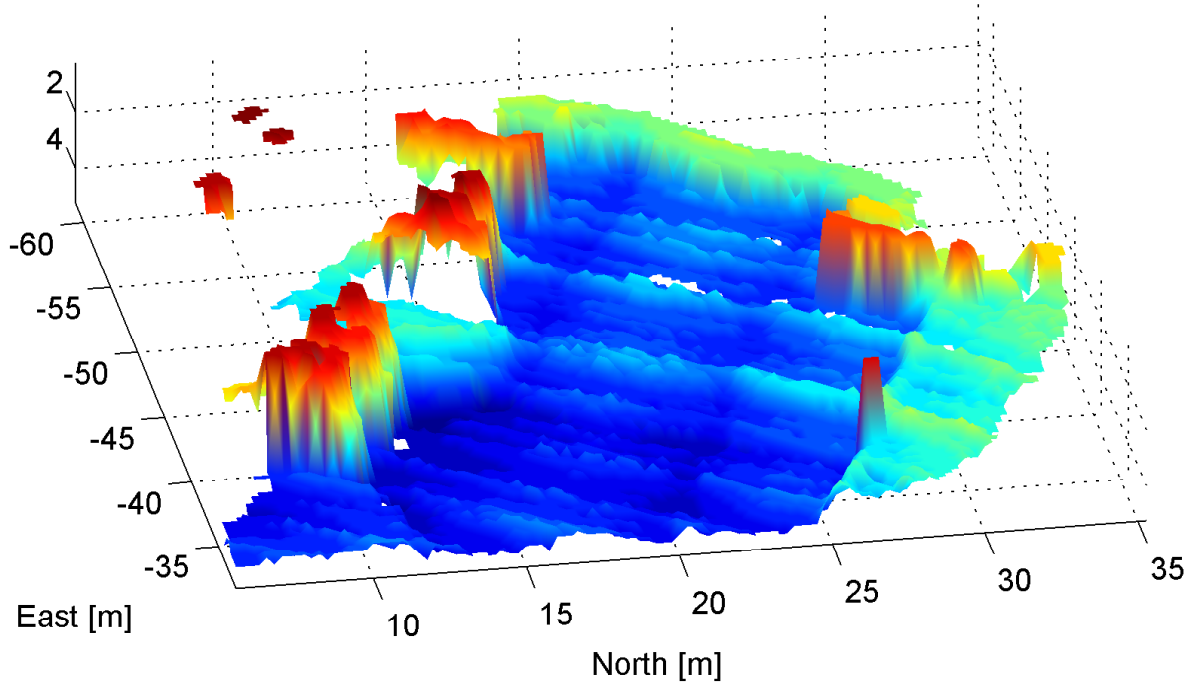


Figure C.3: Surface extracted from 3D OG. colour scaled with height. Note the groups of cars divided by a kerb (on the left) and the trees on the kerb (to the right).

Appendix D

Path Planner Trajectories

The images in Figure D.1, Figure D.2 and Figure D.3 shows three alternative paths computed for the same simulated planning problem. The series of images show the position of the vehicle at different points in time and the path computed at that particular time-step.

The track up to the particular time-step is shown in green and the computed waypoints are shown as blue dots. The planner viewing horizon is the radius about the vehicle that is included in the path planner obstacle detection computation and is indicated by the red circle. Any obstacles outside of the viewing horizon are ignored. Images are taken at every fifth time-step, as indicated in the title of each image, along with the computation time of that path.

Note the following:

- When applied to the same planning problem, the randomised path planner will generate different trajectories each time, as portrayed by these three paths.
- The planned trajectories maintain a minimum clearance from obstacles, based on the vehicle width (see time-steps 21 and 26 in Figure D.2).
- When possible within vehicle constraints, the cost-based planner will maintain additional clearance from obstacles, as evident from Figure D.1 and Figure D.3. Refer to Section 5.3.3.2 on page 83 for the risk conscience map.
- The planner can only optimise paths within the planning horizon, which may result in globally sub-optimal paths, such as Figure D.2.

- This simulation assumes the mapping process is ideal and the map is therefore known up to the planning horizon. It also ignores the effect that closer objects would normally obstruct the view of objects behind them.

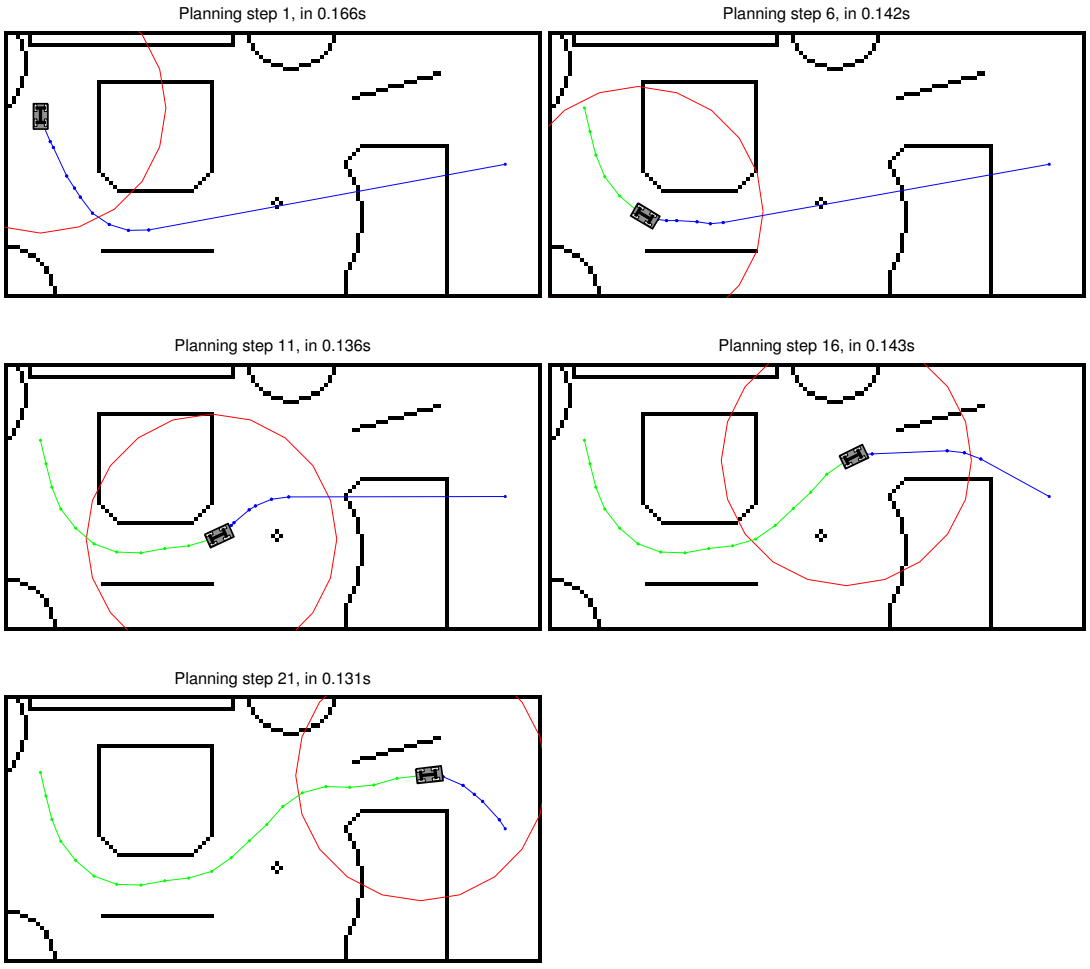


Figure D.1: The first series of images showing the successive path planner trajectories as the simulated vehicle moves along the path.

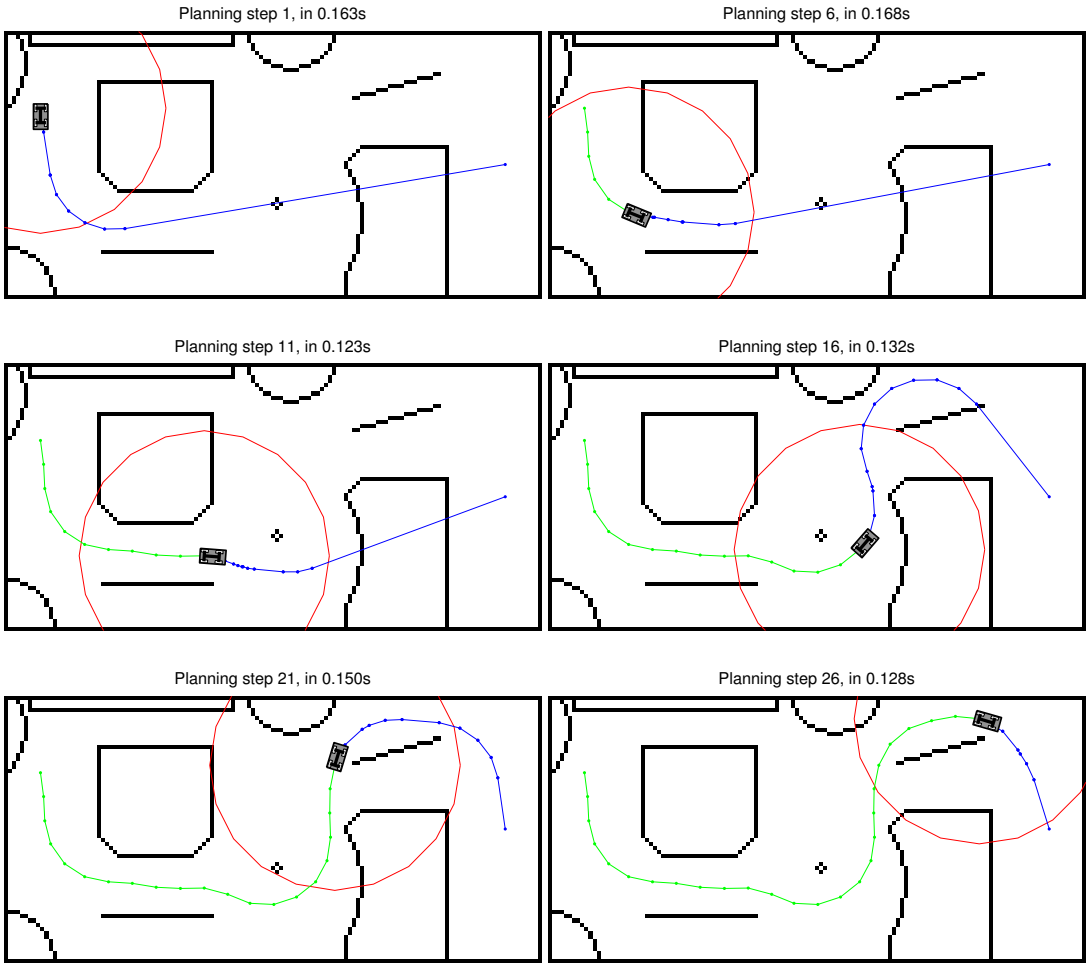


Figure D.2: The second series of images showing the successive path planner trajectories as the simulated vehicle moves along the path.

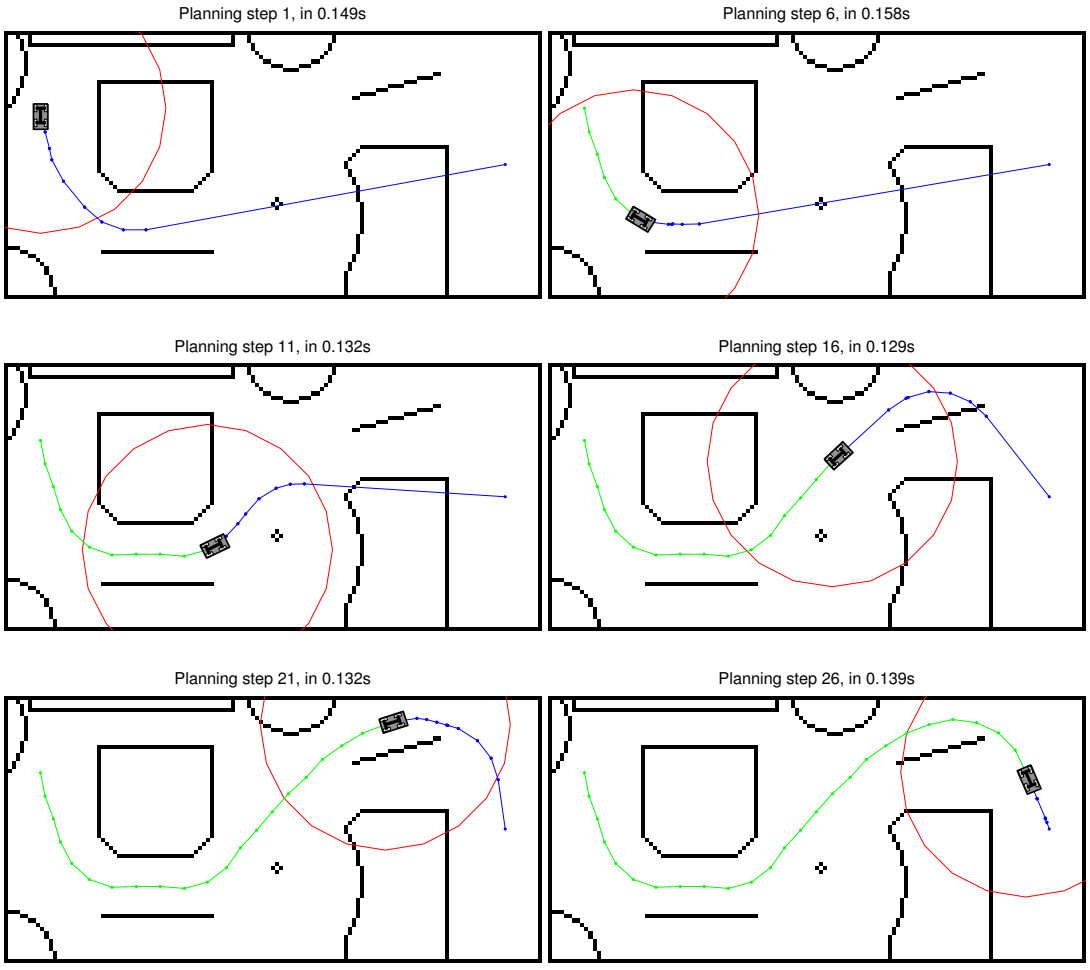


Figure D.3: The third series of images showing the successive path planner trajectories as the simulated vehicle moves along the path.

Appendix E

Photos of Vehicle Hardware

The most notable vehicle hardware is shown in Figure E.1. Actuators that are difficult to distinguish are outlined in red. The approximate mounting locations are indicated.

Figure E.1: Photos of the most notable hardware added and their location on the vehicle.