UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Path planning for an unmanned terrestrial vehicle in an obstacle ridden environment

by

Thomas Ignatius Ferreira

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

Masters of Engineering

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

Study leader: Dr I.K. Peddle

March 2009

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: ....
T.I. Ferreira

Date: ....

Abstract

This thesis relates to the successful development of an unmanned terrestrial vehicle (UTV) capable of operating in an obstacle ridden environment. The primary focus of the project is on the specific path planning algorithms. It is shown that specific methods of *populating* the obstacle-free space can be combined with methods of extracting the shortest path from these *populations*. Through use of such combinations the successful generation of optimal collision-free paths is demonstrated.

Previously developed modular architectures are combined and modified to create a UTV platform which meets all the requirements for implementation of navigational systems and path planning algorithms on board the platform. A two-dimensional kinematic state estimator is developed. This estimator makes use of extended Kalman Filter theory to optimally combine measurements from low cost sensors to yield the vehicle's state vector. Lateral guidance controllers are developed to utilize this estimated state vector in a feedback control configuration. The entire system is then successfully demonstrated within a simulation environment. Finally, practical results from two days of test runs are provided in both written and interactive form.

Opsomming

Hierdie tesis handel oor die suksesvolle ontwikkeling van ′n onbemande grond voertuig (OGV) wat instaat is om te funksioneer in ′n hindernis besaaide omgewing. Die primêre fokus van die projek is op die spesifieke pad beplannings algoritmes. Dit word gewys dat spesifieke metodes waardeur die hindernis-vrye gedeelte van die omgewing *bevolk* word gekombineer kan word met metodes waardeur die kortste pad vanuit hierdie *bevolking* ontgin word. Deur middel van sulke kombinasies word die suksevolle voortbringing van optimale botsings-vrye paaie gedemonstreer.

Vroeër ontwikkelde modulere argitekture word gekombineer en aangepas om ′n OGV platform te skep wat voldoen aan al die vereistes vir implementering van navigasie en pad beplanning stelsels aanboord die platform. ′n Twee-dimensionele kinematiese toestandsafskatter word voorgedra. Die afksatter maak gebruik van uitgebreide Kalman Filter teorie om op ′n optimale manier die metings van lae koste sensors te kombineer en sodoende die voertuig se toestandsvektor af te skat. Laterale beheerders word voorgedra wat hierdie afgeskatte toestandsvektor gebruik in ′n terugvoer beheer konfigurasie. Die hele stelsel word dan gedemonstreer in a simulasie omgewing. Laastens word praktiese resultate van twee dae van toetslopies voorgedra in geskryfde vorm asook in interaktiewe vorm.

Acknowledgements

The author would like to thank the following people for their contribution towards this project.

- Dr I.K. Peddle for his guidance and enthusiasm throughout the project as well as being a consistent driving force. It is much appreciated Iain.
- My parents. Dad, thanks for showing interest, proofreading this thesis and calling it a nice piece of "rocket science". More importantly, thanks for all the opportunities and for being a great dad. Mom, thanks for the countless cups of coffee, great food, continuous love and support, general concern for my health and for being a great mom.
- My grandmother, Hilda, for her love and support and continuous updates about rising or falling fuel prices. Without this I might have lost complete touch with the outside world during the writing up of this dissertation.
- My friend, Christo, whose house paid a heavy price during the practical preparations for this project. Thanks for aiming the frustration at the Lithium-Polymer battery and not at me Stof, you are a dear friend.
- The late Willie van Rooyen for his administrative and technical help which was always accompanied by a friendly face. We will all miss you dearly Willie.
- ESL friends for being a great bunch of guys.
- Last, but not least, Quintis Brandt for his help in tracking down components on more than one occassion

Contents

Nomenclature

Units

V	Volt
A	Ampere
Ah	Ampere hour
m	meter
cm	centimeter
Hz	Hertz
s	Seconds
ms	Miliseconds
ns	Nanoseconds
us	Microseconds
kHz	Kilohertz
$^\circ/s$	Degrees per second angular rotation
$^\circ$	Degrees measure of angle
rad/s	Radians per second angular rotation
m/s	Meters per second measure of translational speed
rad	Radians measure of angle

Acronyms

2D	Two-dimensional
AC	Alternating Current
ADC	Analog to Digital Converter
BST	Binary Search Tree
C	High-level Programming Language
CAN	Controller-area network

CCP	Capture/Compare/PWM
CPR	Counts per revolution
DARPA	Defense Advanced Research Projects Agency
DC	Direct Current
DLQR	Discrete Linear Quadratic Regulator
ECCP	Enhanced Capture/Compare/PWM
ECEF	Earth Centred Earth Fixed geocentric system
EKF	Extended Kalman Filter
ESL	Electronic Systems Laboratory, University of Stellenbosch
GPS	Global Positioning System
IC	Integrated Circuit
IMU	Inertial Meaurement Unit
ISA	Industry Standard Architecture
ISR	Interrupt Service Routine
LAN	Local Area Network
LQR	Linear Quadratic Regulator
MOSFET	Metal-oxide-semiconductor field-effect transistor
NE	North-East reference system
NED	North-East-Down reference system
OBC	Onboard Computer
PCB	Printed Circuit Board
PI	Proportional Integral
PIC	Programmable Intelligent Computer
PWM	Pulse width modulated
RF	Radio Frequency
SI	International System of Units
TUGV	Tactical Unmanned Ground Vehicle
UART	Universal Asynchronous Receiver and Transmitter
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UTV	Unmanned Terrestrial Vehicle
ZOH	Zero Order Hold
DVD	Digital Video Disc

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Background

In a modern age, where there is a general trend toward automization, there has been significant interest in the field of unmanned ground vehicles (UGV), also referred to as unmanned terrestrial vehicles (UTV). The use of UGV's to reach potentially hazardous and sometimes humanly unreachable environments is becoming an attractive and ever more realistic alternative. Examples include the 'Sojourner Rover': developed by NASA for exploration of the planet Mars during the 'Mars Pathfinder' mission, the 'Gladiator Tactical Unmanned Ground Vehicle': a remotely operated TUGV employed by the United States Marine Corps to support dismounted units in all environments and terrain with a modular design to allow for mission specific payloads, 'The Dragon Runner': developed for the US Marine Corps Warfighting Laboratory, designed to increase situational awareness, and Stanford University's Volkswagen-based autonomous vehicle 'Stanley' which won DARPA's Grand Challenge for autonomous vehicles in America in 2006. These UGV's can be seen in Figure 1.1

Motivated by these ongoing developments and several successes the Department of Electronic Engineering's UAV research group, at the University of Stellenbosch, have had recently, it was decided to investigate further and possibly contribute to this UGV field.

Figure 1.1: Dragon Runner (top left), Stanley (top right), Gladiator (centre), Crusher (bottom left), Sojourner Rover (bottom right)

1.2 UTV Overview

Although hardware implementation and control methods are vital elements which contribute to the success of the test vehicle's autonomous navigation, the core of this project relies heavily on the success of the respective path planning algorithms. The global aim of this project is to successfully demonstrate more than one path planning algorithm, whilst at the same time presenting a successful practical implementation of these algorithms in conjunction with the state estimation and control methods used.

Autonomous navigation of a UTV is defined as its ability to move from prespecified point A to point B without any human aid and taking into account that the area between points A and B is obstacle ridden. Several algorithms exist for successfully finding the shortest route between two points when obstacles are present between them. This project focusses on methods of creating *populations* in obstacle-free space and then finding combinations of line segments from these *populations* which provide the shortest possible route. Two *population* algorithms as well as two *shortest route* algorithms are investigated and, as will be evident in this thesis, the respective algorithms display different characteristics and inherently different advantages and disadvantages. Ultimately a trade-off exists between them which will be presented.

The scope of this project does not include obstacle detection and all obstacles are therefore pre-loaded onto the onboard computer from the ground station through RF communication. To a certain extent this lacks purpose and practicality and only seems feasible if unmanned ground vehicles were to download area information from strategically placed servers each time they entered new area grids. For this reason flexibility is increased and the UTV is made more accessible to follow-up projects by strategically implementing all algorithms in such a way as to allow for the addition of sensors for obstacle detection. All algorithms implemented allow for the interrupt, caused by the addition of an obstacles while the navigation systems are running, and therefore simulates the process of, for example, an ultrasonic sensor detecting a new obstacle while the UTV is moving toward its destination.

In order to maximize the autonomous capability of the UTV all processing is done onboard and the ground station merely serves as a monitoring medium, a way to define desired waypoints and a way to initialize all onboard systems. This implies that even if communications with the ground station is lost, during navigation, the UTV will simply continue on its calculated route until it reaches its destination after which it will come to a halt.

Although the testing of a UTV provides much more flexibility than the testing of a UAV, where errors or inaccuracies could lead to devastating results and possibly the complete destruction of equipment when an aeroplane crashes, the goal in this project is still set to fully exhaust simulation resources before attempting test runs. This is done for the purpose of maintaining a strategic approach throughout the project rather than simply trying to achieve results through a process of trial and error. As will be evident in this thesis, this process of simulation before testing proves to be highly advantageous. Through careful debugging within simulation and accurate modelling, very few test runs are necessary to duplicate what is seen in simulation in reality.

The realization of the actual UTV is divided into 5 stages, which are designed, simulated and implemented separately but with a continuous mutual awareness, throughout the respective design processes, in order to be successful in the final stage of integration. During the final integration process it is however still occasionally necessary to refer back to the individual designs when conflicts occur during integration. The order in which these counterparts are implemented in this project is shown in Figure 1.2.

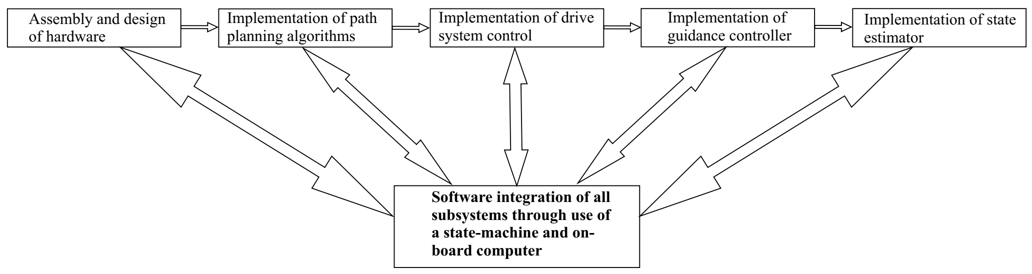

Assembly and design of hardware
Implementation of path planning algorithms
Implementation of drive system control
Implementation of guidance controller
Implementation of state estimator
Software integration of all subsystems through use of a state-machine and on-board computer

Figure 1.2: Project stages

The use of relatively low cost sensors and their inherent poor measurement characteristics lead to minor inaccuracies which does effect the overall performance of the UTV. It will be shown however that with the aid of an EKF, poor measurements can be compensated for with other measurements by optimally weighing the two measurements and, provided the testing grid is not too small, successful obstacle avoidance can still be achieved. It is evident however that the GPS used has a rather low accuracy in terms of this project and its requirements and the integration, of velocity and yaw rate measurements, is therefore vital.

1.3 Thesis Outline

This thesis covers the implementation of an autopilot for an unmanned terrestrial vehicle, operating in an obstacle ridden environment, and includes the preparation of hardware, investigation into path planning algorithms, the implementation of control algorithms and finally practical demonstrations. Chapter 2 is dedicated to a discussion on the hardware used and gives a sense of the physical UTV as a whole. Hardware problems that were encountered and the chosen solutions are also presented. Chapter 3 presents the path planning algorithms and features a discussion on the advantages and disadvantages of each algorithm. In Chapter 4 the focus is primarily on software implementation and the control strategies are introduced. This chapter also features a discussion on the non-linear simulator used to verify the entire system before practical implementation. Chapter 5 presents the development of a simplified two-dimensional state estimator and simulation results of the simulator from Chapter 4, with the state estimator included, are presented. Finally, a discussion on the practical results are provided in Chapter 6.

Chapter 2

Test Vehicle and Avionics

The implementation of an autonomous navigation system, onboard the UTV, requires a software counterpart which is carefully weaved with a hardware counterpart. This chapter focusses on giving an indepth look at the project from a hardware perspective and in doing so will also give clarity about how the software algorithms merge with the hardware.

A picture of the UTV in its entirety can be seen in Figure 2.1. The UTV makes use of batteries as source of power, charging circuitry, electrical motors, digital drive systems to achieve desired wheel speeds, an inertial measurement unit, onboard GPS and RF modules, CAN protocol hardware and an onboard computer.

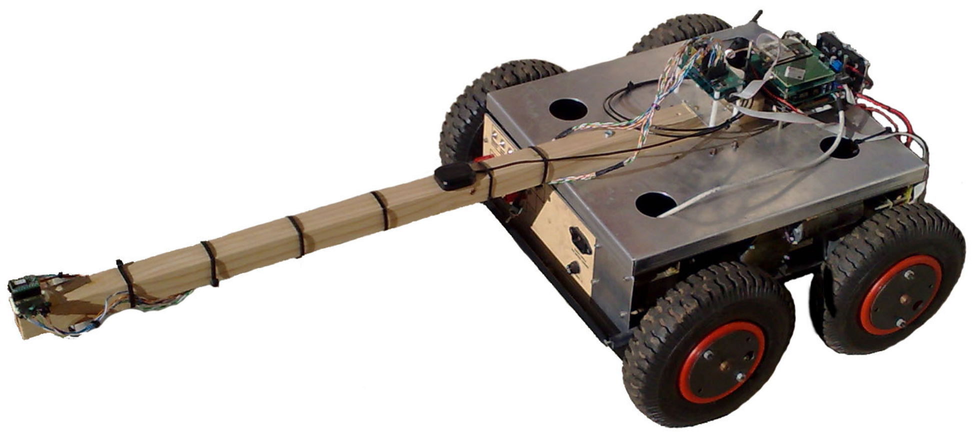


Figure 2.1: Binky

2.1 System Overview

Figure 2.2 shows an overview of the hardware system which is implemented on the test vehicle and how it interacts with the ground station. The sections to follow will give a discussion on each component and give more clarity about each component's relevance in this project. Hardware problems that were encountered and possible solutions are also presented.

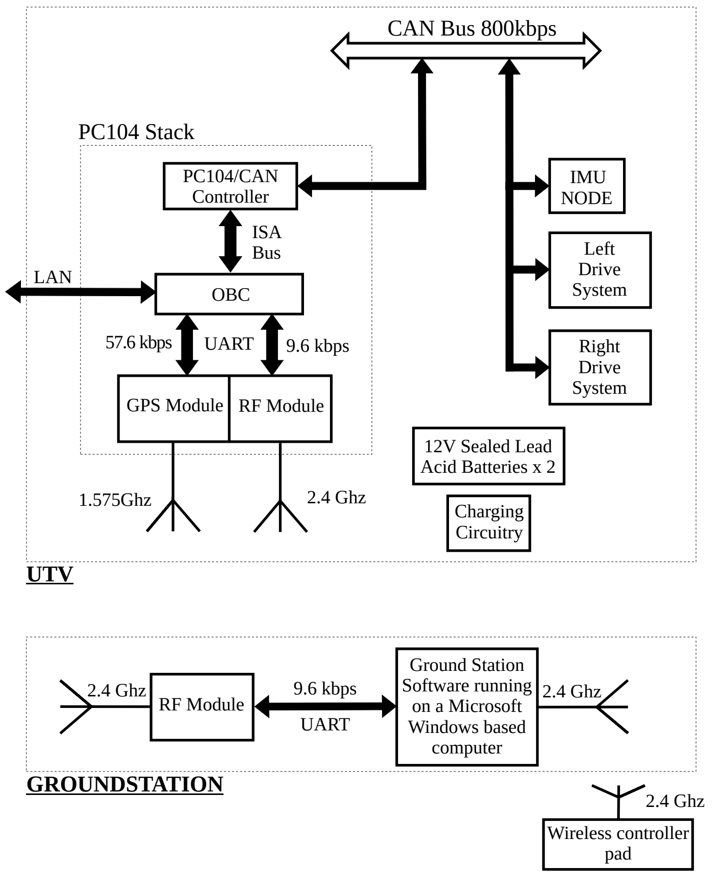


Figure 2.2: SystemOverview

2.2 PC104 Stack

The PC104 stack used in this project is based on a previous design in the UAV research group, at the University of Stellenbosch. The stack can be seen in Figure 2.3 and more detailed information about the architecture can be found in [12] and [8]. The only component which is changed is the OBC and this is done due to the previous model no longer being available at the time of the project. This architecture is adopted because of the flexibility and extendibility it inherits from its use of a CAN bus to communicate with various control nodes.

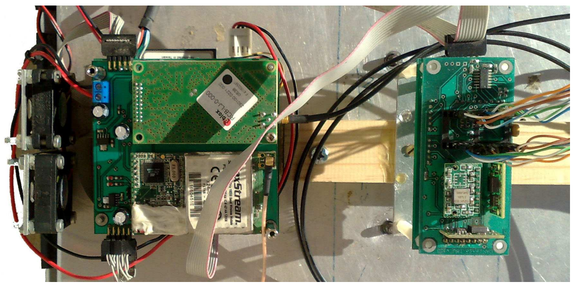


Figure 2.3: PC104 Stack(left) and IMU(right)

The PC104 stack consists of the elements listed below and the following subsections will give a brief discussion on each element and how it fits into this project.

- Onboard Computer (OBC)
- PC104/CAN Controller
- GPS and RF daughter board

2.2.1 OBC

The OBC serves as the main processing unit on board the UTV. All path planning algorithms and control algorithms, excluding wheel speed control and CAN protocol implementation, are implemented in C (programming language) on the OBC. The OBC is an Intel 400 MHz Ultra Low voltage Celeron processor, presented in PC104 form factor, and it supports a Compact Flash Disk of up to 4GB. The hard disk used in this project is a 256 MB Compact Flash disk which provides more than enough storage space for logging errors and telemetry data during UTV navigation. An indepth look at the technical specifications of the OBC can be found at [18].

Communications with the GPS module and RF module are done directly through use of the OBC's dual UART communication ports, while communications with the sensor node and drive system nodes are routed through the PC104/CAN Controller via the Industry Standard Architecture (ISA) bus. The OBC also has a Local Area Network Connection (LAN) which is used for downloading and uploading of data to and from a laptop or desktop computer.

As far as control algorithms go the OBC has more than enough processing power. Certain path planning algorithms implemented in this project are however more computationally expensive, especially when the area in which the UTV operates has a large number of obstacles. This implies that, while path planning algorithms are running on the OBC, periods might occasionally occur during which the processor is heavily loaded and the sample time for the control system might be exceeded. This only occurs during the addition of a new obstacle, in real time when obstacle detection is simulated, and a new path planning cycle is scheduled. One possible solution is to spread the calculations of the path planning algorithms over more than one sample time, by storing the current status of the path planning variables, when the sample time becomes threatened, and then continuing the process during the successive cycles. This is however not a trivial task with the path planning algorithms, discussed in Chapter 3, and an alternative solution is therefore considered. Since these processing *blackouts* can jeopardize the integrity and validity of the control system, the decision is made to make the UTV stop entirely and maintain its current position and attitude every time a new path planning cycle is scheduled. Justification for this is found in the fact that a UTV, unlike a UAV, can intermittently be completely stationary.

During practical tests however, it becomes evident that with relatively few obstacles, the margin with which the sample time is exceeded is negligible if not absent. Seeing as it only occurs during one cycle, the overall effect on the control system is also negligible. The significance is further lowered by the fact that the UTV moves relatively slowly in comparison with the sampling frequency on which the implemented control system is based. Nevertheless, with an increasing amount of obstacles the effects on the control system could become significant and it is therefore decided to leave the failsafe in place. As previously mentioned, in Chapter 1, the scope of this project does not include obstacle detection. The effect of the UTV stopping momentarily is therefore only seen in this project when a new obstacle is uploaded from the ground station during navigation. More information about this and the control algorithms implemented on the OBC can be found in Chapter 4.

2.2.2 PC104/CAN Controller

The CAN Controller board is positioned right above the OBC in the PC104 stack. It is responsible for the timing of the entire system and also includes voltage regulators, which step down the supplied 12 V rail to 5 V and 7 V respectively, and then supply power to the entire PC104 stack and nodes on the CAN bus. Due to the combined power consumption of the PC104 stack and CAN nodes these regulators tend to experience significant increases in temperature and two DC cooling fans are therefore directed onto the PC104 stack as can be seen in Figure 2.4.

The PC104/CAN Controller communicates with the OBC via the ISA bus and also controls the flow of data packets on the CAN bus which is connected to the drive systems and IMU node. The protocol onboard the PC104/CAN Controller was developed by [12] and takes 20 ms to complete one cycle. It is implemented on a PIC microcontroller onboard the PC104/CAN Controller and remains unchanged from [12], with the exception of the specific CAN packets which are issued during each 20 ms cycle.

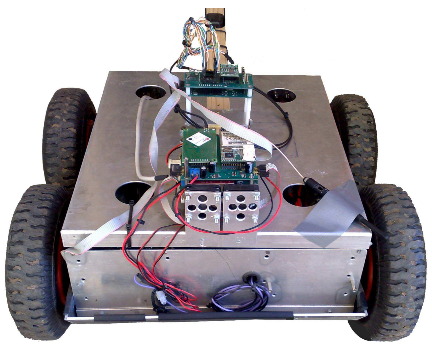


Figure 2.4: Rear view of UTV and Cooling Fans

In order to accommodate the left and right drive systems the CAN packet structure is altered in C (programming language) on board the PIC microcontroller. CAN packets are now addressed to the IMU node, left drive system and right drive system respectively while the global synchronization packet remains unchanged. In previous UAV applications, at the University of Stellenbosch, CAN packets were also issued for servo commands which are discarded in this project since they are replaced by two drive systems.

2.2.3 GPS and RF daughter board

The GPS and RF daughter board can be seen clearly at the top of the PC104 stack in Figure 2.3. The respective RF and GPS modules used are listed below.

- MaxStream XStream 2.4 GHz OEM Transceiver
- u-Blox RCB-LJ OEM Receiver

The RF module is used as means of wireless communication between the OBC and a laptop computer which serves as ground station, which is also connected to an RF transceiver. In previous UAV applications at the University of Stellenbosch this MaxStream module has proven itself to be highly reliable with a communications range far beyond what is necessary for the success of this project. Successful determination of the UTV's position is done by means of an Extended Kalman Fliter (EKF) which combines sensor measurements to obtain a best estimate of the vehicle's state vector in a noisy environment. The use of a GPS in this project is debatable and the size of the area grid in which the UTV operates plays a big role. Due to the limited accuracy of this GPS it only becomes useful when the UTV traverses a long distance where accumulated errors from the propagation of encoder and rate gyroscope measurements become significant. Over small distances however it relies heavily on these propagations and is not sensitive to GPS measurements. This insensitivity to GPS measurements creates the possibility for indoor usage of the UTV over small distances.

2.3 Inertial Measurement Unit

The IMU, seen in Figure 2.3, was developed by [5] and [12]. At the bottom it has a baseboard on which a PIC microcontroller resides. This baseboard is responsible for incoming sensor measurements from the printed circuit board (PCB) above it on which the inertial sensors reside. The measurements obtained are analogue filtered, sampled by an Analogue to Digital Converter (ADC) and then digitally low pass filtered to suppress noise. The inertial sensors on the top PCB include:

- 3 x ADXRS150 Single-Axis Rate Gyroscopes from Analog Devices
- 2 x ADXL210E Dual-Axis Accelerometers from Analog Devices
- 1 x HMC2003 Three-Axis Magnetometer from Honeywell

The sensors listed above provide enough measurements for motion in a full six degree of freedom application. Although this project only requires enough measurements to accommodate motion in three degrees of freedom, full functionality of the IMU was still implemented so as to allow for its use in future projects. The modularity of the design makes it easily transferable to another platform and it could therefore be re-used in future UAV applications if the need arises.

It should be noted that the rate gyroscopes used in this project, unlike the ones used in [15], are only capable of measuring maximum angular rates of $\pm 150^\circ/\text{s}$. The $\pm 150^\circ/\text{s}$ angular rate is also mapped over 3 V as in [15] and therefore provides a higher resolution. These rate gyroscopes are chosen to accommodate the UTV's much slower yaw rates.

Attitude of the UTV is only required in a two dimensional (2D) plane. For this reason and due to the additional availability of encoders to determine wheel speeds, the inertial sensors listed above are only partially used. The sensors used by the UTV are listed below.

- 1 x Single-Axis Yaw Rate Gyroscope
- 1 x Three-Axis Magnetometer (only 2 horizontal axes used)

These two sensors are used to determine the UTV's yaw angle (heading angle). Through use of an EKF, the measurement from a single yaw rate gyroscope is combined with the measurements from the two horizontal axes of the magnetometer to provide a best estimate for the current yaw angle of the UTV.

A measurement of the yaw angle is obtained from the magnetometer, seen in Figure 2.5. The magnetometer provides a vector measurement of the earth's magnetic field and in doing so, the UTV's heading angle relative to the horizontal component of this field can be determined. Pre-determined specifications of where exact North is, relative to Magnetic North, is taken into account and this then provides enough information for the successful determination of the yaw angle in a North-East (NE) reference frame. More technical information about how the HMC2003 magnetometer operates can be found in [15].

During implementation it was observed that strong magnetic fields, emanating from the UTV's chassis, cause disturbances in the magnetometer measurements. This can easily be observed by holding a compass in close proximity above the UTV and observing the incorrect measurement of North on the compass. Occasional random rotations, without settling, of the compass needle may also be observed. For this reason the magnetometer was mounted on a wooden extension of the UTV's chassis in order to isolate the magnetometer from these disturbances.

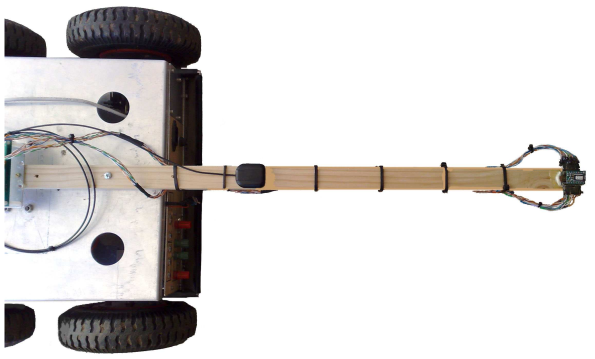


Figure 2.5: Magnetometer (at the end of wooden extension)

An indepth investigation to find the exact sources of these electromagnetic disturbances is beyond the scope of this project. Calculating the exact magnetic field around the UTV would require a complex use of Ampere's Integral Law of Magnetic Field Intensity and possibly Gauss' Integral Law of Electric Field Intensity [3], especially when the layout and non-uniform and non-symmetric wiring of the UTV's drive systems are considered. A brief investigation is however presented which gives a sound indication of what the source of the magnetic field disturbances might be.

According to [3], Ampere's Integral Law states that the line integral (circulation) of the magnetic field intensity **H** around a closed contour is equal to the net current passing through the surface spanning the contour plus the time rate of change of the net displacement flux density $\epsilon_\circ \mathbf{E}$ through the surface (the displacement current). This law is shown mathematically below and an illustration can be seen in Figure 2.6.

$$\oint_C \mathbf{H}\, d\mathbf{s} = \int_S \mathbf{J}\, d\mathbf{a} + \frac{d}{dt}\int_S \epsilon_\circ \mathbf{E}\, d\mathbf{a} \tag{2.3.1}$$

Note that **J** is the current density enclosed by the contour and an integral of this density yields the net current.

Using this law the magnetic field around a line current can be expressed in terms of specific radii around the line current. A line current is formally de-

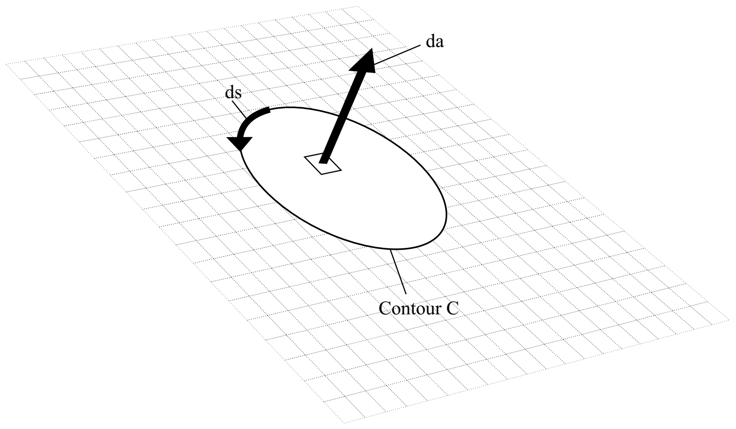


Figure 2.6: Ampere's Integral Law of Magnetic Field Intensity

fined as the limit of an infinite current density distributed over an infinitesimal area

$$i = \lim_{\substack{|\mathbf{J}|\to\infty \\ A\to 0}} \int_A \mathbf{J}\, d\mathbf{a} \tag{2.3.2}$$

where i is a current, constant over the length of a thin line.

If the assumption is made that one of the conductors connecting the UTV's motors to the power supply (through the drive systems) can be modeled as a line current, then a consideration of a single such conductor yields a good indication of what the induced magnetic field intensity might be at the magnetometer's mounting position on the UTV.

Wheel speeds and loads on the DC motors vary over time and the current through such a conductor is thus dependent of time. However, for the sake of this argument and to obtain an indication of one possible magnetic field strength at a specific moment in time, at a constant wheel speed, the assumption is furthermore made that the current through this wire is independent of time and therefore also the fields. This implies that the second term on the right of Equation 2.3.1 is zero and the equation reduces to

$$\oint_C \mathbf{H}\, d\mathbf{s} = \int_S \mathbf{J}\, d\mathbf{a} \tag{2.3.3}$$

Since the wire is modeled as a thin line current, substitution of 2.3.2 into 2.3.3 yields

$$\oint_C \mathbf{H}\, d\mathbf{s} = i \tag{2.3.4}$$

After calculation of the integral, having taken into account that **H** only has an azimuthal component [3], a final equation is obtained which yields magnetic field strength at a radius r.

$$H_\phi = \frac{i}{2\pi r} \tag{2.3.5}$$

The magnetometer measures flux density which is in units of Gauss ($\mu_\circ H_\phi$). The current at which the DC motors stall is 18 A [10] and will induce the maximum magnetic field intensity around the conductor. Using 17 A as the line current and noting that the magnetometer was initially mounted approximately 10 cm from the main conductors on the UTV, Equation 2.3.5 yields a flux density of 0.000034 Gauss.

When taking into account that the earth's magnetic field at Stellenbosch has components, with the magnetic flux densities listed below, it becomes evident that this calculated disturbance field is 3 orders of magnitude smaller and cannot solely be responsible for the disturbances.

Earth's Magnetic Field

- North = 0.093904 Gauss
- East = -0.04136.6 Gauss
- Down = -0.236304 Gauss

However, when considering the fact that there are multiple conductors running from the batteries to drive-systems to DC motors, inside the UTV's chassis, the figure calculated above is significantly increased.

As will be shown in Section 2.4.3, the UTV's drive systems control the DC motors' supply voltage by switching between -12 V and + 12V at a frequency of 25 kHz and varying the duty cycle. This implies that there is a change in current over time, even at a constant wheel speed, and more importantly a significant time rate of change of the net electric displacement flux density (displacement current). The previous assumption that the second term on the right of Equation 2.3.1 is zero, is therefore not entirely accurate and further contribute to a larger disturbance flux density. The flux density figure above is further altered by the fact that the conductors inside the UTV are not line currents in reality.

A final consideration is the fact that the plate, separating the UTV's undercarriage and drive systems from the IMU, is made from Aluminium which is paramagnetic [21] and therefore allows a magnetic field to pass through with ease. This is not a good magnetic shield and a ferromagnetic material used in its stead might decrease disturbance fields at the magnetometers mounting position.

Choosing a ferromagnetic material for this separation plate might have an influence on the measured Z (Down) component of the earth's magnetic field but this is acceptable when one considers the fact that in a 2D application only the East and North components of the earth's magnetic field are required for successful determination of the UTV's yaw angle.

2.4 Undercarriage and Drive Systems

Figure 2.7 shows the positioning of the relevant hardware components in the UTV's undercarriage. The remainder of this chapter will focus on these components as well as the vehicle chassis.

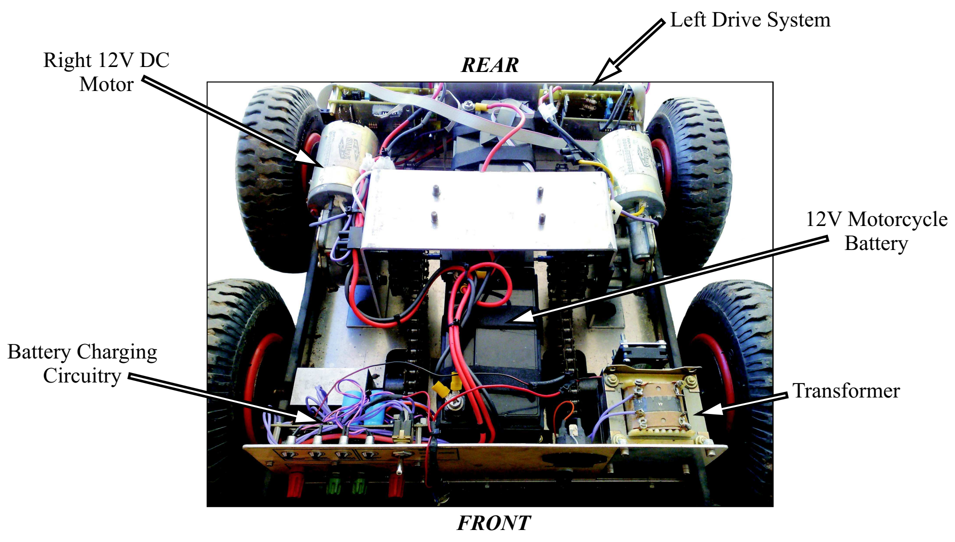


Figure 2.7: UTV Drive Systems

2.4.1 Vehicle Chassis

The chassis of the UTV, originally designed by [10], is manufactured from 3mm Aluminium sheeting. Movement of the UTV is made possible by four 200 mm wheels with pneumatic tyres which are mounted without a steering mechanism since the UTV steers itself using a skid steer principle. Using a skid steer platform provides the benefit of simplicity when the UTV is required to turn on one spot.

Torque is relayed from the motors to the wheels via a chain sprocket system. Since the motors provide more torque than what is required the sprocket system is designed with a turns ratio of 1.5 : 1 as shown in Figure 2.8. This leads to higher wheel speeds at lower corresponding motor speeds and results in a more efficient utilization of the motors' torque capabilities.

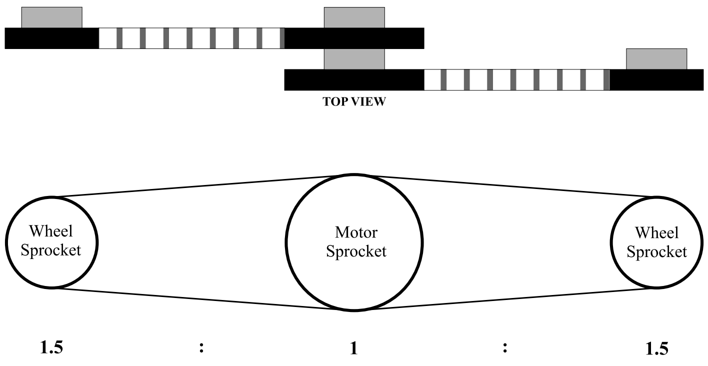


Figure 2.8: Turns Ratio of Sprocket System [10]

2.4.2 Batteries and Charging Circuitry

Power is supplied to all systems on board the UTV by two 12 V, 8 Ah motorcycle batteries which are located in the front and back of the UTV as can be seen in Figure 2.7. Since each motor is capable of drawing up to 18 A when it stalls these batteries are connected in a parallel configuration to maximize current output capacity.

Charging of the batteries is done through use of a Sealed Lead-Acid battery charger configuration, designed by [16]. The circuit configuration also includes a 5 V DC regulator and a full circuit schematic can be found in Appendix A.

A standard domestic 220 V AC power supply is connected to a transformer to obtain a 16 V AC output which is fed into an AC-DC converter to obtain a 23V DC output. Through use of a Texas Instruments UC2906 Lead-Acid Linear Charge Management IC this 23 V is then utilized to obtain appropriate voltages and currents for charging of the batteries. Light emitting diodes are also implemented which make the charging process more user friendly by indicating normal charging cycles, over charging cycles and completion of charging cycles. In addition to charging of the batteries the circuit configuration provides a 5 V regulated DC output alongside the unregulated 12 V battery output. These outputs can be seen on the front panel of the UTV in Figure 2.7.

2.4.3 Drive Systems

The purpose of this section is to introduce the drive system medium through which the UTV gains controlled mobility. The drive systems on board the UTV each consist of three core component categories. On each side the components include a DC motor, a microprocessor alongside a full bridge DC-DC converter, and an encoder and current transducer. These three component categories act as actuator, controller and measurement devices respectively and through their use a closed loop control system is implemented for each side of the UTV. The hardware architecture, originally designed by [10], is reviewed and modified in this project to improve performance and compatibility with the hardware discussed in Sections 2.2 and 2.3.

A basic layout of the drive system on each side of the vehicle and how the components interface with each other can be seen in Figure 2.9.

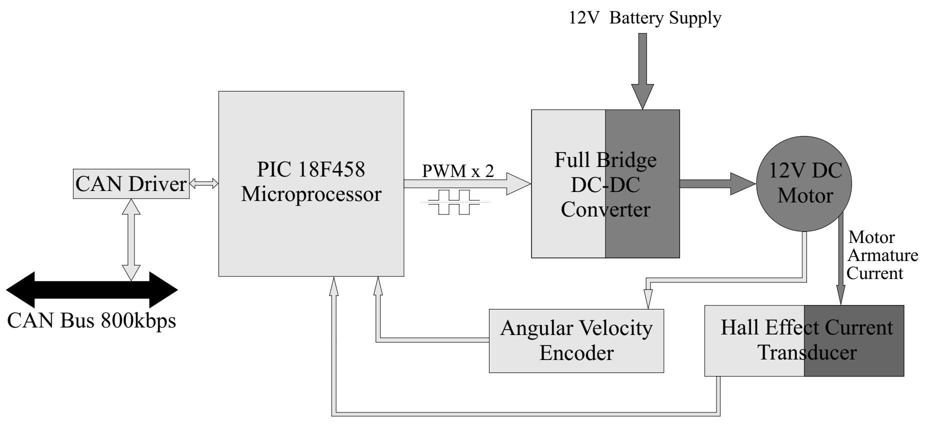


Figure 2.9: Drive System Overview

In Figure 2.9 all lighter shadings indicate the electronic, low current side of the system which is isolated from the heavy current side of the motor, Full Bridge DC-DC converter and Hall Effect current transducer. The purpose of this hardware configuration is to create a closed system which fully controls motion of the UTV while only requiring communication with higher level hardware through use of the CAN bus.

PIC Microprocessor

The PIC18F458 Microprocessor is responsible for internal communications within the drive system. It also implements two way communication with higher level hardware through use of an onboard CAN module and external CAN driver. Wheel rotation speeds are received digitally via the CAN bus and the PIC then uses this as reference speed. Through use of angular velocity encoders and current transducers, actual wheel speeds and DC motor armature currents are measured and used in control algorithms on board the microprocessor. Using this state feedback configuration, the control algorithms, which are discussed in Chapter 4, provide an appropriate pulse width modulated (PWM) input to the full bridge DC-DC converter which, in turn, produces an appropriate control input voltage to the DC motor.

Communication on the CAN bus with the PC104/CAN Controller (Section 2.2.2) is done through a protocol which includes the following data packets.

Received by Drive System

- Global Synchronization Package
- Disable/Enable Drive Systems
- Angular Velocity Setpoint

Sent by Drive System

- Rotation Speed, Direction and Status

The global synchronization packet is transmitted every 20 ms by the PC104/ CAN controller to all nodes on the CAN bus. This packet serves as a signal for all nodes to reply with their current measurement values and status. In the case of the drive systems a packet is transmitted, in reply to this message, which contains the current measured rotation speeds obtained from the encoders, the current direction of rotation and the current status of the specific drive system, whether it be enabled or disabled.

During each 20 ms cycle a setpoint value for each drive system is also obtained from the OBC and transmitted by the PC104/CAN Controller. The drive system receives this packet and extracts its commanded reference speed from the received data. A special command packet can also be received by the drive system which enables or disables the drive system. This happens during initialization of the UTV autopilot or when explicitly requested by the ground station during manual control of the UTV. More information on this protocol and the actual data packets can be found in Appendix A.

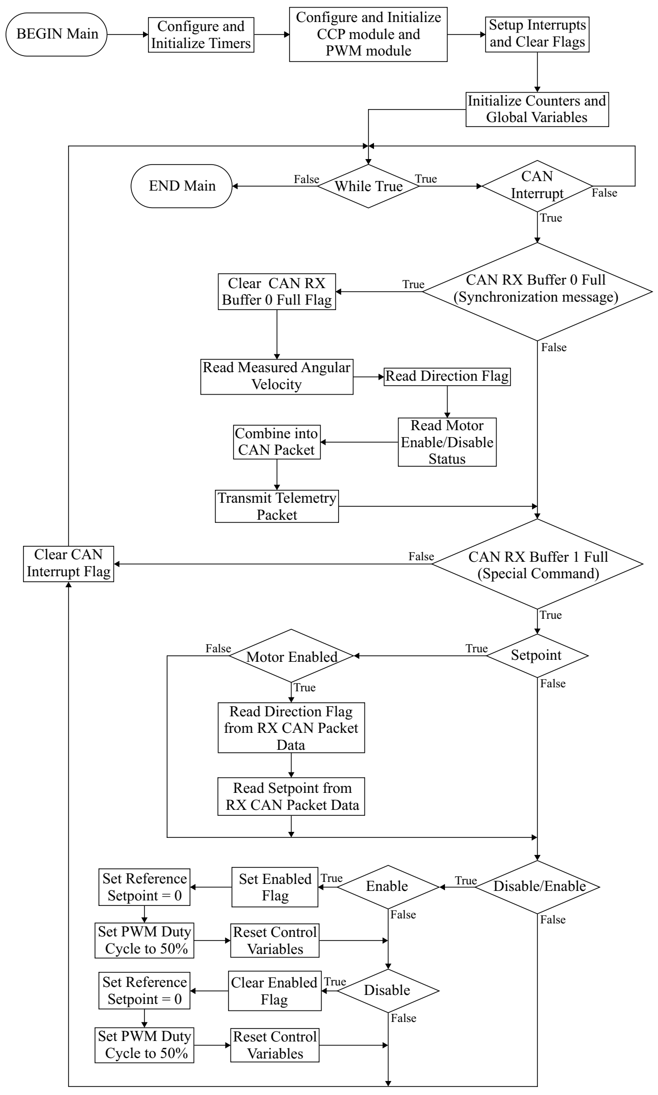

BEGIN Main
Configure and Initialize Timers
Configure and Initialize CCP module and PWM module
Setup Interrupts and Clear Flags
Initialize Counters and Global Variables
END Main
While True
CAN Interrupt
CAN RX Buffer 0 Full (Synchronization message)
Clear CAN RX Buffer 0 Full Flag
Read Measured Angular Velocity
Read Direction Flag
Read Motor Enable/Disable Status
Combine into CAN Packet
Transmit Telemetry Packet
CAN RX Buffer 1 Full (Special Command)
Clear CAN Interrupt Flag
Setpoint
Motor Enabled
Read Direction Flag from RX CAN Packet Data
Read Setpoint from RX CAN Packet Data
Disable/Enable
Enable
Set Enabled Flag
Set Reference Setpoint = 0
Set PWM Duty Cycle to 50%
Reset Control Variables
Disable
Clear Enabled Flag
True
False

Figure 2.10: Flowchart of Main Function on board the PIC microprocessor

Figure 2.10 shows a flowchart of the main repetitive loop implemented in C code on the PIC microprocessor. As seen this main function is responsible for servicing received CAN packets as well as transmission of telemetry packets whenever a global synchronization message is received from PC104/CAN Controller. All control algorithms and measurements are done in low priority and high priority service interrupts respectively and will be discussed in the sections to follow.

DC-DC Converter

A full bridge DC-DC converter is used to regulate the input voltage to each motor in accordance with the control algorithms executed on the PIC microcontroller. The PWM signal and its inverse, obtained from the PIC microcontroller, are used as logical inputs for two dual channel high speed IR2110 power MOSFET drivers. The outputs of these IR2110 drivers are in turn connected to four metal-oxide-semiconductor field-effect transistors (MOSFET). This configuration can be seen in Figure 2.11.

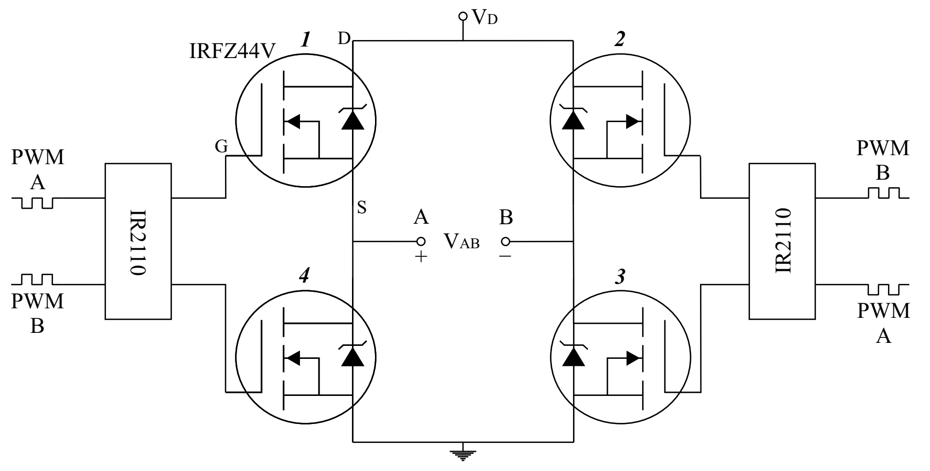


Figure 2.11: DC-DC Full Bridge Voltage Converter

Through utilization of the PWM signals obtained from the PIC microcontroller, and appropriate connections, controlled switching of the four individual transistors is achieved. At any moment in time only two of the transistors, also referred to as switches, will be closed and conducting while the other two will represent open-circuits. When taking a closer look at Figure 2.11 it becomes apparent that the switches are diagonally grouped in pairs with respect to their switching behaviour, i.e. transistor **1** and **3** will be conducting when **2** and **4** are open-circuits and vice versa. V_D represents the unregulated supply voltage from the batteries of the UTV while V_{AB} represents the voltage which is connected across the terminals of the DC motor. The voltage across the DC motor will therefore continuously be switched between $-V_D$ and $+V_D$. By varying the duty cycle of the supplied PWM signal the DC component, of this signal which is being switched between the negative and the positive of the battery supply, can be changed accordingly and this is ultimately how the rotation speeds of the DC motors are controlled.

In [10] a frequency of 9.8kHz is used for the PWM signal supplied to the IR2110 drivers from the PIC microcontroller. This leads to undesirable high frequency audible noise while the MOSFETS are switching. Since the audible range of humans is approximately 20 Hz - 20 kHz [22] a decision was made in this project to increase the switching frequency to 25 kHz. A further benefit of increasing the switching frequency is the reduction in ripple currents in the motors which in turn reduces torque pulsations. This can be shown with Equation 2.4.1 from [10]

$$(\Delta I_{p-p})_{max} = \frac{V_D}{2L_a f_s} \tag{2.4.1}$$

where I_{p-p} is the peak to peak ripple current, L_a is the armature inductance and f_s is the switching frequency. An increase in f_s clearly implies a decrease in ripple current I_{p-p}.

This increase in frequency is made feasible by the change in crystal oscillator to 9.6 MHz and usage of the High-Speed Crystal/Resonator Phase Locked Loop mode of the PIC18F458 which results in a clock frequency of 38.4 Mhz on the microcontroller. The higher clock frequency allows for higher PWM frequencies while still maintaining high PWM resolution. Equation 2.4.2, from the Microchip PIC18F458 datasheet, shows how no resolution is sacrificed from [10] while achieving higher PWM frequencies.

$$PWM_{MAX} = \frac{log(\frac{f_{osc}}{f_{pwm}})}{log(2)}\ bits \tag{2.4.2}$$

$$\begin{aligned} f_{pwm} &= 25\ kHz \\ f_{osc} &= 38.4\ MHz \\ PWM_{MAX} &= 10.585\ bits \end{aligned}$$

This 10 bit resolution leads to accurate speed control with 0.02 V voltage increments in supplied voltage to the DC motors as shown in Equation 2.4.3.

$$V_{increment} = \frac{24}{2^{10}}\ V \tag{2.4.3}$$

In addition to PWM resolution the timing limitations of the IR2110 drivers and IRFZ44V MOSFET's also have to be taken into account. The timing specifications for these components are obtained from the International Rectifier datasheets and presented below.

Symbol	Definition	Maximum	Units
t_{on}	Turn-on propagation delay	150	ns
t_{off}	Turn-off propagation delay	125	ns
t_{sd}	Shutdown propagation delay	140	ns
t_r	Turn-on rise time	35	ns
t_f	Turn-off fall time	25	ns
	Total	**475**	

Table 2.1: IR2110

Symbol	Parameter	Maximum	Units
t_{on}	Turn-on delay time	13	ns
t_{off}	Turn-off delay time	40	ns
t_r	Rise time	97	ns
t_f	Fall time	57	ns
	Total	**207**	

Table 2.2: IRFZ44V

The maximum accumulated delay of 0.475 us from the tables above is now compared to the 40 us period of the chosen 25 kHz PWM signal. This comparison shows that during each cycle of the PWM signal there must be an imposed dead time (1.2% of the PWM period) between switching of the MOSFETS to prevent cross over current. A dead time of 0.7 us is therefore set using the microprocessor's dedicated register for this. This dead time is decreased from the initial dead time of 1.2 us implemented by [10].

As mentioned in [10], the non-linearity due to blanking time is minimized by using an internal current loop where the armature current of the DC motor is also sampled for feedback. This measurement of current implemented by [10] was not functioning at the time and the printed circuit boards were in bad condition. The Full Bridge converter PCB was therefore redesigned and assembled.

Equation 2.4.4 shows how the supplied voltage to the DC motors is a function of duty cycle D, with range 0 to 1, and through use of this equation the PWM duty cycle and consequently the voltage to the DC motor, is automatically varied during control execution on board the microprocessor.

$$V_{AB} = (2D - 1)V_D \tag{2.4.4}$$

Encoders

As previously mentioned two Agilent angular velocity optical encoders are used for the measurement of wheel speeds on board the UTV. These encoders each contain a lensed LED source with a code wheel which rotates between an emitter and detector IC. Through use of this configuration two square waves in quadrature are obtained from each encoder which represent counts per revolution. The quadrature signals are connected to the PIC18F458 and by using the capture module on the microprocessor the time between rising edges of these signals are calculated to provide a measure of angular velocity. Since these signals are 90° out of phase with each other, it can be determined which signal is leading the other and consequently direction of rotation is also determined.

On the PIC18F458 microprocessor Timer 3 is setup for association with the capture/compare/PWM (CCP) module, which is used for the capturing of angular velocity from the encoders, while Timer 2 is setup for association with the Enhanced/Capture/Compare/PWM (ECCP) module which is used for the PWM output. Timer 1 is additionally utilized during the encoder measurements and a discussion on this will follow shortly.

Timer 3 is reset every time a rising edge of the incoming square wave from the encoder is detected and the timer then starts counting in increments of 833 ns until the next rising edge occurs. The capturing of rising edges occurs asynchronously and the measurement update rate is a function of the current wheel speed. Special attention therefore has to be given to extremely low wheel speeds where the measurement update rate becomes slow in comparison with the bandwidth of the drive system.

The measurement of slow wheel speeds is further complicated by a 16 bit limitation on the PIC18F458. Since Timer 3 on the microprocessor has a 16 bit width an overflow occurs after every 65535 counts and the counter is then reset to 0. The possibility therefore exists that extremely low wheel speeds will

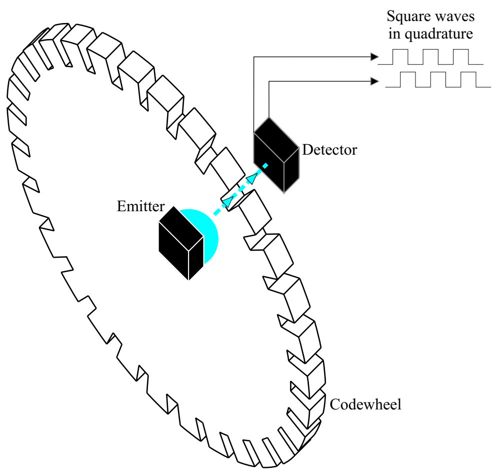


Figure 2.12: Basic operation of the Encoder

cause the timer to exceed 65535 counts, reset to zero and consequently measure an incorrect period between the corresponding rising edges. A simple solution to this would be to increase Timer 3's pre-scaler value and hence the counting period of 833 ns is also increased but according to the PIC18F458 datasheet the maximum prescaler for either Timer 1 or Timer 3, which can be associated with the CCP or ECCP modules, is 1 : 8. Raising the counting period is further not desired since it would decrease the resolution of time between rising edges and thus the resolution of the angular velocity measurement.

From the Agilent Technologies's datasheet for the HEDS-5500 A-13 encoder it is found that the code wheel has a resolution of 500 counts per revolution (CPR) and a count of 65535 therefore, when taking the 1.5 : 1 sprocket system ratio into account, corresponds to a wheel speed of

$$\omega_{wheel} = \frac{2\pi(1.5)}{500(65535)(833 \times 10^{-9})} \tag{2.4.5}$$

$$= 0.35\, rad/s$$

and a measurement update rate of

$$f_{meas} = \frac{1}{65535(833 \times 10^{-9})} = 18.32Hz \tag{2.4.6}$$

Wheel speeds below this threshold have to be accounted for by storing the amount of overflows in memory and adding a corresponding multiple of 65535 for every subsequent count until the next rising edge is detected, after which the overflow counter is reset again. However, theoretically this implies that should the UTV come to a complete halt or slow down drastically between rising edges the counter will increment itself indefinitely until the overflow counter overflows itself and a further overflow counter is required. While it has the advantage of accurate measurement of extremely low wheel speeds it has the disadvantage of an asymptotic approach to zero wheel speeds and hence a measurement of zero wheel speed is never obtained.

This asymptote at zero and measurement update rates below the system bandwidth at extremely low wheel speeds ultimately compromise the integrity of the control system. In this region the control system operates with significantly delayed measurements and the effect is most severe when the UTV is stationary and a zero measurement can not be obtained. This causes the control system to continuously reduce actuator input, when a zero setpoint is received via the CAN bus, in an attempt to achieve zero motor speed, until ultimately a negative non-zero speed occurs and the process is reversed causing a limit cycle. Slower measurement rates at slower wheel speeds can be accommodated by reducing the update rate of the control system, while staying well above the bandwidth of the drive system, or by modelling the variable measurement delay somehow but ultimately the asymptote at zero still remains.

Since the UTV in this project is only required to turn with a maximum yaw rate on one spot when finding the heading of a new path segment, and to track a straight line path segment at a chosen common mode speed by superimposing differential wheel speeds on top of this common mode speed, the decision is made to regard all measurements below the threshold from Equation 2.4.5 as zero since the operating range of the UTV falls outside this region.

However, since the OBC and PC104/CAN Controller operates at 50 Hz the new chosen sample rate has to be an integer multiple of 0.02 s to make easy implementation feasible without altering the already implemented protocols. A multiple of 3 would imply a sample rate of 16.67 Hz which is rather low in comparison with the system bandwidth. The decision is therefore made to choose a sample rate of 25 Hz which is easily implemented on the OBC by updating control after every second multiple of the 50 Hz protocol rate. This allows for the simple addition of a counter which allows update of UTV control at a slower rate by simply updating after every multiple of 2 of the 50 Hz rate. This implies that the threshold from Equation 2.4.5 has to be increased further so that no wheel speed measurement update rates slower than the chosen 25 Hz sample rate occurs. A sample time of 25 Hz equates to a minimum wheel speed of 0.48 rad/s from Equation 2.4.5 which is still outside the operating range of the UTV and this new threshold and sample rate is therefore decided on. A reference wheel speed between -0.48 rad/s and 0.48 rad/s is therefore serviced by a hard-coded exception function on the microcontroller which sets the PWM output to an exact 50 % duty cycle (resulting in zero wheel speeds after proper calibration) and any encoder measurement between -0.48 rad/s and 0.48 rad/s, which is indicated by an overflow in 16 bit Timer 3 or measurement update rate below the chosen threshold, is regarded as zero. This measurement threshold speed can be lowered as required by the specific application but at the expense of an overall decreased control sample rate. Within the scope of this project lower speed measurements are not required for the reasons mentioned, as well as the fact that ripple currents due to Mosfet switching and inadequate motor sensitivity to fine voltage increments limit the maximum resolution in controlled wheelspeeds which can be achieved. This measurement update rate of 25 Hz is relatively high in comparison with the drive system bandwidth which will be shown to be 3.15 Hz, and therefore makes the decision even more attractive.

A further complication arises from the fact that if the UTV were to come to a standstill with the code wheel in close vicinity of a transitional edge, ripple currents from the switching Mosfets and consequent torque pulsations could cause slight vibrations which lead to high frequency capturing of rising edges and a sudden deviation from zero in measured speed even though the UTV is stationary. During testing this phenomenon is best observed when the UTV is placed on an elevated platform and the wheels are not grounded giving them more freedom to rotate without the frictional load which is present when the UTV's weight rests on the wheels. The freedom of movement causes the wheels to be more sensitive to ripple currents and hence vibrations occur more easily. Disturbances are also present on encoder measurements during operation of the UTV when torsional vibration and terrain disturbances such as small rocks or slippery terrain cause non-uniform rotational motion of the wheels.

An illustration of the effects of these disturbances can be seen in Figure 2.13 where an encoder output for the minimum, maximum and an intermediary angular speed of the wheels are shown in conjunction with the corresponding calculated angular velocity of the wheels after each rising edge. Also illustrated in this figure is the variable sample rate which is a function of the angular velocity of the wheels.

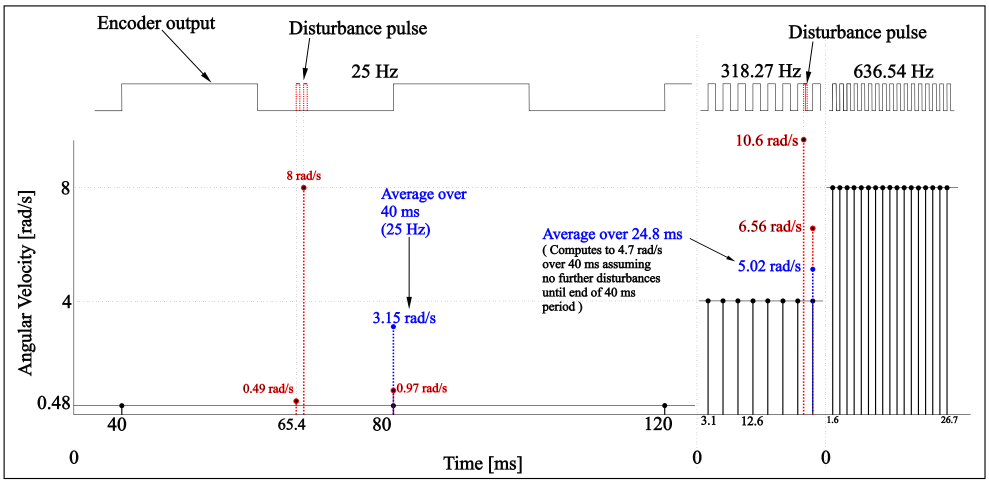


Figure 2.13: Effects of disturbances on encoder measurements

It is clearly visible that the measurement of angular velocity is more sensitive to disturbances at lower speeds since the measurement update rate is significantly slower. Assuming that the control rate is of the same order as the higher measurement rates, at higher speeds, it can be seen that the effects of an incorrect measurement due to disturbances has a significant duration at slow speeds where there is an increased measurement delay and the control system will thus react accordingly to an incorrect measurement. However at higher angular speeds with faster measurement update rates an incorrect measurement is quickly corrected by a successive measurement and the effects are therefore less significant on the control system. Since lower speeds also have to be accommodated, and it is difficult to model a variable measurement delay in the control system design, the maximum control rate is limited to 25 *Hz*. A solution therefore has to be found for utilizing the faster measurement rates at higher angular velocities while still maintaining control system integrity at lower speeds by keeping the control rate below or at 25 *Hz*.

To accomplish this the decision is made to do averaging of the measurements at a rate of 25 Hz. By doing this, measurement noise at higher angular velocities is significantly attenuated through utilization of the higher sample rates. At lower angular velocities noise is also attenuated but to a lesser extent since there exists no excess measurements for averaging and averaging is just achieved of measurements and the possible disturbance measurements between them. Figure 2.13 also illustrates this. It is clearly visible that at 4 rad/s the process of averaging attenuates noise more efficiently than at the minimum drive shaft angular velocity of 0.48 rad/s.

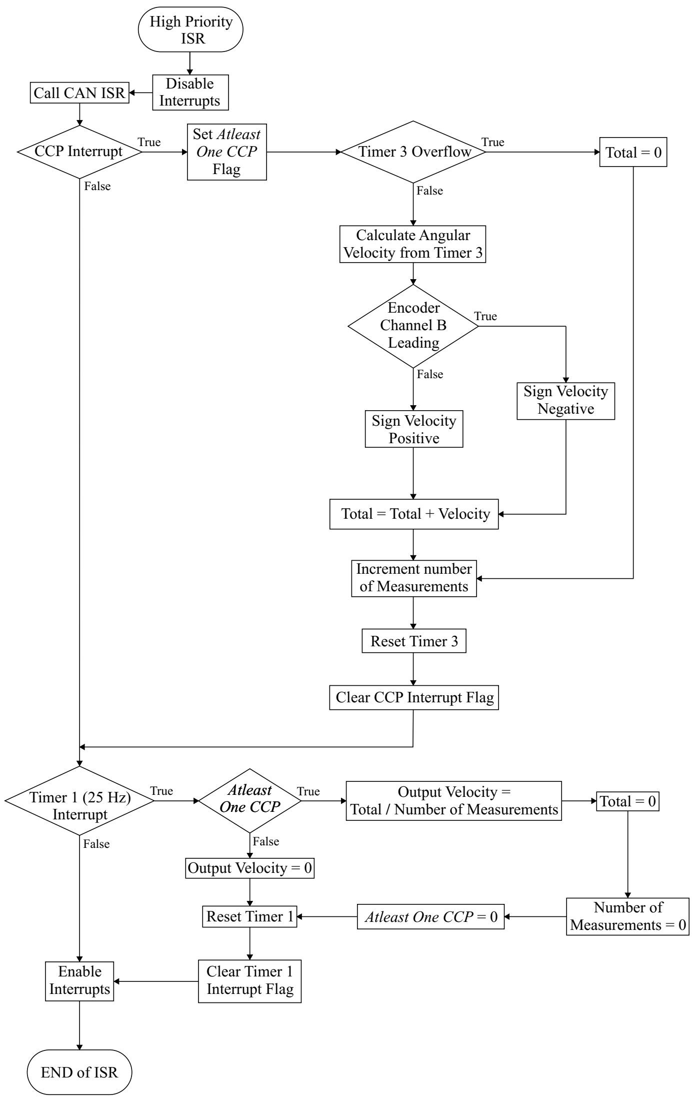


Figure 2.14: Flowchart of Angular Velocity Measurement ISR

Figure 2.14 shows the flow diagram of the Angular Velocity Measurement Interrupt Service Routine. This ISR has a high priority and therefore overrides any low priority interrupts which may be in the process of executing. Software polling is done within the ISR to distinguish between sources of the interrupts which occurred. As shown in Figure 2.14, all interrupts are disabled during execution of the ISR to avoid recursive interrupts. The software polling can however still accommodate the scenario where one interrupt source goes active shortly after the other has gone active since interrupt flags for specific sources are set when they occur, regardless of whether the particular interrupt is enabled, as pointed out in the PIC18F458 datasheet. The scenario of one interrupt source going active shortly after the other therefore does not require an exit and re-entry into the ISR and consequently processing time is not wasted.

An illustration of the averaging process previously mentioned is also shown. Averaging is done with signed (according to direction) angular velocity measurements and the disturbances due to vibrations, when the UTV is stationary, are therefore attenuated since they correspond to fast successive negative and positive measurements which cancel to approach a zero measurement. During UTV movement the direction of rotation is favoured by the measurements and sudden disturbance measurements of an opposite direction is attenuated by the the process of averaging.

Timer 1 is set up to overflow and cause a high priority interrupt at a rate of 25 Hz and determines the interval over which averaging is done. All counters and flags shown in Figure 2.14 are initialized to zero in the main function on the microprocessor. Timer 3 and Timer 1 is also reset and started in the main function and the first capture of a rising edge, when the UTV starts moving, will therefore correspond to the overflow flag of Timer 3 having been set, since no rising edges occurred for a significant time since initialization, and thus a zero measurement is obtained (assuming vibration disturbances while stationary did not cause premature captures of rising edges since the UTV was under frictional load). The very first sample of angular velocity once the UTV has started moving therefore only occurs after the second rising edge is captured.

Current Transducers

As previously mentioned, the armature currents of the DC motors are also measured for feedback. This is accomplished with the same Hall Effect current transducers used by [10]. These Hall Effect sensors provide a voltage output which is proportional to the current which flows through it. Figure 2.15 shows a flowchart of the Low Priority interrupt service routine implemented on the PIC mircoprocessor.

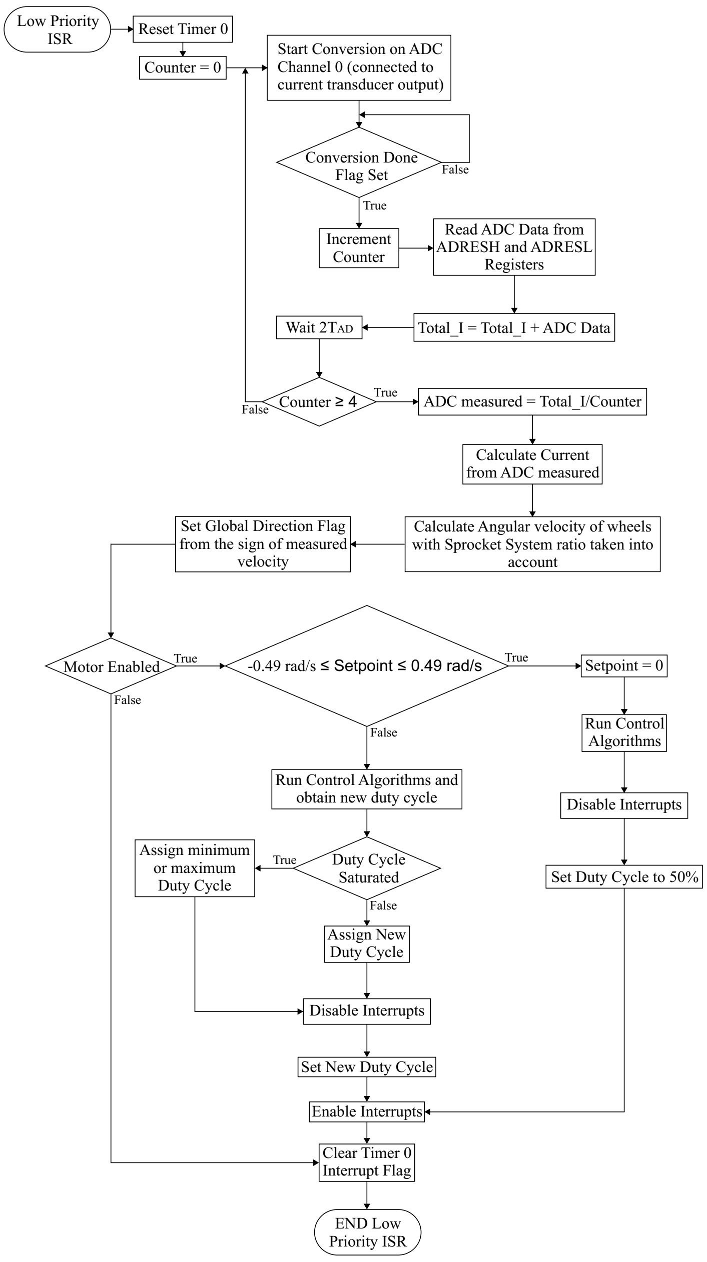

Low Priority ISR
Reset Timer 0
Counter = 0
Start Conversion on ADC Channel 0 (connected to current transducer output)
Conversion Done Flag Set
False
True
Increment Counter
Read ADC Data from ADRESH and ADRESL Registers
Wait 2TAD
Total_I = Total_I + ADC Data
Counter ≥ 4
False
True
ADC measured = Total_I/Counter
Calculate Current from ADC measured
Set Global Direction Flag from the sign of measured velocity
Calculate Angular velocity of wheels with Sprocket System ratio taken into account
Motor Enabled
True
False
-0.49 rad/s ≤ Setpoint ≤ 0.49 rad/s
True
False
Setpoint = 0
Run Control Algorithms
Run Control Algorithms and obtain new duty cycle
Disable Interrupts
Assign minimum or maximum Duty Cycle
True
Duty Cycle Saturated
False
Set Duty Cycle to 50%
Assign New Duty Cycle
Disable Interrupts
Set New Duty Cycle
Enable Interrupts
Clear Timer 0 Interrupt Flag
END Low Priority ISR

Figure 2.15: Flowchart of Low Priority ISR

Timer 0 on the microprocessor is setup to cause a low priority interrupt every

40 ms $(25Hz)$. When this low priority interrupt occurs the measurement of output voltage from the current transducer is done through use of the PIC microprocessor's Analog to Digital Converter (ADC) Channel 0. As can be seen in Figure 2.15, 4 successive measurements are taken and the average of these measurements calculated. This is done to decrease the effects of a possible incorrect analog to digital conversion measurement if one should occur.

There exists a slight delay during each analog to digital conversion and a repetitive while loop executes until the Conversion Done Flag has been set. According to the PIC18F458 datasheet the conversion time per bit (T_{AD}) must be bigger than or equal 1.6 us and a delay of $2T_{AD}$ must be allowed for between successive analog to digital conversions. T_{AD} is therefore setup as $64T_{OSC}$ resulting in a value of 1.67 us and as seen in Figure 2.15 a delay of $2T_{AD}$ is implemented between successive measurements. Since the ADC module on the microprocessor has a 10 bit resolution the total delay per measurement is therefore $12T_{AD}$ or 20.4 us. The averaging process requires 4 measurements and the total delay per Timer 0 cycle is therefore 80.16 us. In comparison with the sample rate of 40 ms this delay is a factor 499 times smaller and is therefore acceptable.

After a measurement of current is obtained the angular velocity obtained from the encoders is converted to a wheel speed by taking into account the ratio of the sprocket system. Depending on whether the motor is enabled or not and whether the setpoint is not within the previously discussed lowest threshold speeds the control algorithms are then executed to obtain an updated duty cycle for the PWM output. It should be noted that the minimum UTV reference speed is marginally increased from 0.48 rad/s to 0.49 rad/s so as to ensure that as soon as movement commences the encoder measurement update rate is faster than the 25 Hz averaging rate.

It should also be noted that a ripple current exists on the armature current of the DC motor, due to the previously mentioned MOSFET switching, and this causes high frequency components $(25kHz)$ on the current transducer's voltage output signals. The output of the current transducers is therefore analogue filtered before being fed into the microprocessor's ADC input pin. The filtering is done with a low-pass Butterworth filter at $25Hz$, implemented in hardware on the PCB.

2.5 Summary

In this chapter the hardware components used on the UTV were introduced as well as problematic areas which were encountered and their corresponding solutions. The hardware was categorized into three major components which includes the PC104 Stack, Inertial Measurement Unit and all systems relevant to the drive systems of the UTV. The interdependence of the systems were discussed while also presenting the UTV as a whole. Brief discussions were also given on where specific software algorithms fit into the hardware.

This chapter is therefore concluded and lays the foundation for control and path planning algorithms, from chapters to come, to be implemented upon.

Chapter 3

Path Planning Algorithms

In order to achieve successful navigation through an obstacle ridden environment path planning algorithms are a necessity. The concept of path planning has received significant attention in applications which range from motion planning for robots to route calculation for units in strategy computer games. This chapter is by no means a summary of all the methods available but aims to present a discussion on the methods chosen for this project. The specific algorithms used are introduced as well as a comparison between their respective performances. With the availability of the vehicle and hardware platform, discussed in Chapter 2, a decision is made to investigate algorithms which yield paths, consisting of straight line segments joined at successive nodes. Once this decision is made the path planning problem can be separated from the rest of the project with the only stipulation being that, paths yielded must consist of straight line segments where each line segment is fully defined by length, starting position, endpoint and heading angle in the commonly defined North-East(NE) axis system. This path data is required by the control algorithms presented in Chapter 4. The path planning counterpart of the project can then be viewed as a closed module with obstacle coordinates, starting point, and destination as inputs, and an optimal path as output.

3.1 Overview

The path planning problem can be solved through implementation of several different algorithms, each displaying different characteristics and inherently different advantages and disadvantages. These algorithms are required to make use of known obstacle coordinates and then calculate an optimal path from starting position to destination, which avoids all obstacles at all times. Obstacles are defined in terms of the coordinates of their vertices in a 2-dimensional plane and within the scope of this project the following assumptions are made,

- Obstacles are disjoint which implies that no two obstacles overlap and each obstacle is uniquely and separately defined. An example of two disjoint obstacles is shown in Figure 3.1.

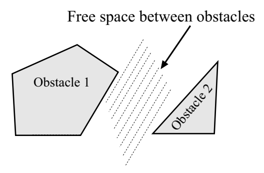


Figure 3.1: Two Disjoint Obstacles

- Obstacles are convex polygons. This concept is best visualized when thinking of the obstacle vertices as nails pointing upwards out of the plain. When an elastic rubber band is held around these nails and released it will take on a shape which represents the convex hull of the vertices. Stated in different terms, no two edges of the obstacle will meet at a vertex to form an exterior angle which is less than 180°. An example of a convex obstacle is shown below,

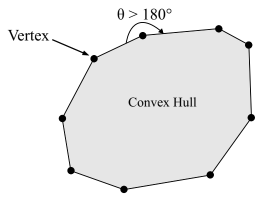


Figure 3.2: Example of a Convex Hull

These assumptions are necessary since the majority of computational geometry algorithms are based on these two stipulations. Extended techniques are available to accommodate obstacles which do not adhere to these stipulations but increases the complexity and computational cost of the problem. The reader is referred to [7] and [9] for more information on handling these special cases. Although these assumptions seem limiting, justification within the scope of this project is found in the fact that any two obstacles which are not disjoint can be redefined as one larger obstacle which encloses the two non-disjoint obstacles to form a new obstacle which is disjoint from the remainder of the obstacles. The second assumption is also justified since no limitations were placed on the size of the obstacles in this project and once again a non-convex obstacle can be represented by a larger convex obstacle which encloses the non-convex obstacle completely.

Through use of computational geometry techniques the obstacle-free space is populated with a map consisting of line segments joined at specified nodes within the free space. By extracting a certain combination of these line segments, from the populated set, a path through the obstacle ridden environment is found. However, several different feasible paths can be extracted from this populated set and won't necessarily always yield the shortest path. An additional *shortest path algorithm* is therefore used which determines the combination of line segments from the populated map which will yield the shortest possible path. It is important to note that the *shortest path algorithm* does not necessarily calculate the absolute shortest path through the obstacle ridden environment but merely the shortest possible path which can be constructed from the specific set of line segments generated by the specific *population* algorithm. The nature of the *population* algorithm therefore determines whether the final shortest path will be the actual shortest path through the environment.

The following two *population* algorithms are implemented in this project,

- *Voronoi* diagram
- *Visibility* graph

and both these *population* algorithms are tested in co-operation with both of the following *shortest path algorithms*,

- *Dijkstra's* algorithm
- *A*Star* search

An additional note should be made about the nature of the obstacles defined for the *Voronoi* algorithm. For reasons which will become apparent when the *Voronoi* algorithm is discussed, only square obstacles can be accommodated by this *population* algorithm. This has its origin in the fact that the *Voronoi* diagram is not specifically formulated as a solution to the path planning problem and has many other applications in diversified fields of interest. The problem therefore has to be translated in such a way as to allow the *Voronoi* diagram to offer a solution. Although inefficient, the stipulation of all obstacles being square is justified once again by the fact that any convex obstacle can be enclosed by a larger square obstacle and the new square obstacle is then used in the algorithm. Although not implemented in this project, an attractive alternative to the *Voronoi* diagram was found which is briefly discussed in Chapter 7.

3.2 *Population* Algorithms

This section will introduce the two *population* methods implemented in this project. The characteristics and reasoning behind both methods are presented and the problem is then translated from an intuitive idea to a computational algorithm which can be implemented on a computer. Arguments are based on the discussions in [9] and [7]. Flow diagrams will also be shown which display the basic structure of the implementations.

3.2.1 *Voronoi* Diagram

Theory of the Voronoi Diagram

The *Voronoi* diagram is chosen in this project due to its fundamental characteristic which makes it a candidate for an algorithm with maximum clearance from obstacles. The Euclidean distance between two points, q and p, can be denoted as,

$$R(q,p) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2} \tag{3.2.1}$$

where R is the Euclidean distance between the two points. Consider a 2-dimensional plane which contains n distinct points, $\{p_1, p_2, ..., p_n\}$, where these points are defined as sites. The *Voronoi* diagram is defined as a subdivision of this 2-dimensional plane into n cells, one for each site, with the unique property that at any point, q, within one cell corresponding to site p_i, the distance to p_i is always shorter than the distance to any other site, p_j, that is $R(q,p_i) < R(q,p_j)$. A simple example of this *Voronoi* diagram is shown in Figure 3.3 which shows how each site corresponds to a cell in which all points are closer to the specific site than to any other site.

From this definition of the *Voronoi* diagram another observation can be made which is ultimately exploited in this project during the implementation of the *Voronoi* diagram for path planning purposes. It follows from simple geometric intuition that 4 sites representing the vertices of a square will have the *Voronoi* diagram shown in Figure 3.4, and it is clearly visible that the *Voronoi* edges will always pass through the exact centre of the square in or-

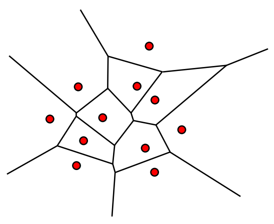


Figure 3.3: Simple Example of a *Voronoi* Diagram

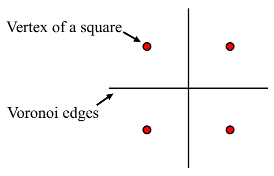


Figure 3.4: *Voronoi* diagram of the vertices of a square

der to satisfy the characteristics of the *Voronoi* diagram. If all obstacles in the path planning problem are considered as square then a simple method is found for applying the *Voronoi* diagram in such a way as to yield a populated graph which only has line segments in the free space of the plane. This is achieved by specifying each vertex of each square obstacle as a site during *Voronoi* calculation and then adding a pruning process after calculation which eliminates all *Voronoi* edges, which pass through the centre coordinates of the square obstacles. A populated line segment map is then achieved which approximates maximum clearance from all obstacles. It should be noted that this implementation is not always efficient and special scenarios exist where a discontinuity between start and goal is present, as well as scenarios where the maximum clearance characteristic disappears. These occurrences will be shown later in this section. As mentioned, an improved *population* method was found in hindsight which is briefly introduced in Chapter 7.

Fortune's Algorithm

With the definition of the *Voronoi* diagram available, the next step is the development of an algorithm which calculates this diagram. Several algorithms have been proposed over the years of which the most efficient one is based on a proposal from *Fortune* in 1985. A summary of this algorithm is presented below, which is based on the discussions in [7] and [9].

Before introducing *Fortune's* ideas a brief discussion is required on *plane-*

sweep algorithms. These kind of algorithms follow an approach where a virtual sweep line is passed over the plane, leaving at any point in time the problem solved for the portion of the plane already swept, and unsolved for the portion of the plane which still has to be swept. *Plane-sweep* algorithms are attractive since they yield lower computational complexities, generally of $O(n \log n)$. However, the difficulty of using a *plane-sweep* algorithm comes in when one considers the fact that in order to construct the *Voronoi* diagram, knowledge of the sites beyond the sweep line, yet to be encountered, is required. Stated in different terms, the sweep line will encounter edges of the *Voronoi* diagram before encountering the sites which are responsible for those edges.

Fortune proposes an ingenius solution to this problem through consideration of a 3-dimensional space while solving the 2-dimensional *Voronoi* problem. Consider the two cones erect over the xy-plane, with infinite height, and sides sloped at 45°, shown in Figure 3.5. These two cones each have their apex over a point, P_i, which represents one of the sites in the xy-plane, on which the *Voronoi* diagram is to be constructed. The bisector of P_1 and P_2 is defined as the bisector perpendicular to the line segment $\overline{P_1P_2}$. When one considers the fact that both cones are identical in rate at which the circular diameter increases, in an ascending horizontal xy-plane, it is easy to see that these two cones intersect in a vertical plane which is exactly the vertical projection of the bisector of P_1 and P_2. In turn it then follows from simple geometric intuition that this vertical intersection plane does indeed represent the *Voronoi* edge between the two sites, when viewed from $z = -\infty$. *Fortune's* proposal exploits the fact that the intersection of these cones, when viewed from $z = -\infty$, represents the *Voronoi* diagram. Instead of using a simple sweep line, as is the general case in *plane-sweep* algorithms, *Fortune* makes use of a *sweep line*, L, and *sweep plane*, π, which is slanted at 45° to the xy-plane, as shown in Figure 3.6.

When viewing the two cones and the plane in Figure 3.6 from $z = -\infty$, all sites and cones to the right of the sweep line, L, that is the portion $x > l$, are obscured by the slanted plane which cuts the xy-plane at L. These sites represent the portion of the plane which still needs to be swept. The portion of the plane, $x < l$, is however visible from $z = -\infty$, up to the intersection of the slanted plane, π, with the positive frontier of the cones. From the basic properties of conic sections the intersection of the slanted plane with any one of the cones is a parabola. The intersections of all cones with the slanted plane

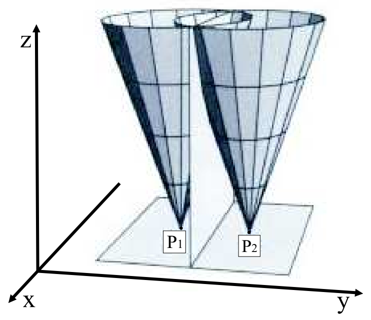


Figure 3.5: A line is projected by the curve of intersection of two cones [7]

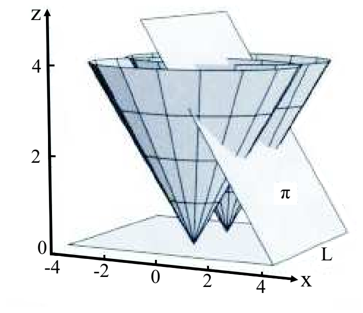


Figure 3.6: Plane π and L sweeps toward $x \to \infty$ and cuts the cones [7]

therefore represents a parabolic front, which consists of pieces of parabolas, when projected onto the xy-plane.

This idea is shown in Figure 3.7 which shows a view of the cones and sweep plane from $z = -\infty$, with only the $x < l$ portion of the plane visible. It is clearly visible how the two parabolas of intersection between the sweep plane, π, and each cone, together form a parabolic front. At the specific point where these two parabolas join, to form the parabolic front, the edge of the *Voronoi* diagram is traced out as the plane sweeps toward $x = \infty$.

Through use of this slanted plane *Fortune* solves the problem of the sweep line encountering *Voronoi* edges before the sites which generate them are encountered. The portion of the diagram to the left of the sweep line is therefore not constructed at all times but rather the portion of the diagram underneath the plane, π. The sweep plane is sloped at exactly the same angle as the sides of the cones and as soon as a new site is encountered by the sweep line, L, the sweep plane simultaneously encounters the cone for that site.

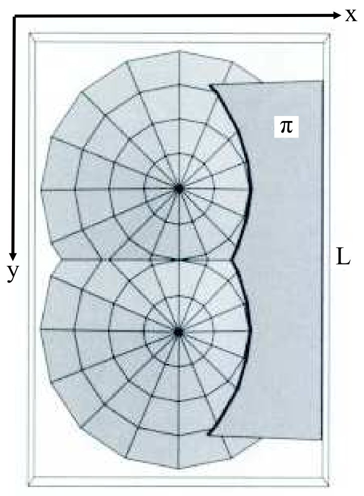


Figure 3.7: Viewed from $x \approx -\infty$, the bold curve represents the parabolic front [7]

The most valuable observation which is made from this 3-dimensional consideration of the problem is the fact that the *Voronoi* diagram can be traced out by the points where parabolas meet, in a parabolic front, as a sweep line is moved across the plane. When returning to the problem in 2-dimensional space the idea of this parabolic front can be utilized to develop an algorithm. The parabolic front, consisting of parabolic arcs is called the *beach line* and is represented by the bold arc in Figure 3.8. This arc represents the projections, of the intersections of 3-dimensional cones and the slanted plane, onto the xy-plane. The *beach line* also represents the locus of points that are closer to any site above the sweepline, L, than to the sweep line itself.

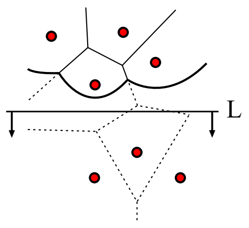


Figure 3.8: The parabolic front in the xy-plane [9]

The algorithm is now based on a process of maintaining this *beach line* as the sweep line moves. The *beach line* changes continuously, however, an explicit maintenance at all times is not required but merely an identification of special events which changes the structure of the *beach line* in terms of its parabolic arcs. Two such events exist, one being the event in which a new parabolic arc appears on the *beach line*, and the second being the event in which a parabolic arc shrinks to a point and then disappears.

The first event, where a new parabolic arc appears on the beach line, is caused when the sweep line encounters a new site. In this event the parabola which is defined by the site is at first a degenerate parabola as shown in Figure 3.9. As the sweep line continues to move downwards the new parabola grows and becomes wider as shown in Figure 3.10.

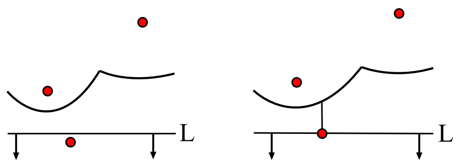


Figure 3.9: Degenerate parabola when new cite is encountered [9]

The breakpoints where this new parabola meets the old parabola start to trace out a new *Voronoi* edge in opposite directions which is at first not connected to the *Voronoi* diagram. Eventually this edge meets up with another edge and becomes connected to the *Voronoi* diagram. In [9] it is shown that this is indeed the only way a new arc can appear on the *beach line* and this event is referred to as a *site event*.

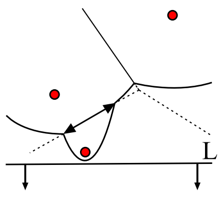


Figure 3.10: Degenerate parabola grows wider as sweep line moves [9]

The second type of event occurs when an existing arc of the *beach line* shrinks to a point and then disappears. Consider the 3 parabolic arcs, α, α_i and α_{ii}, shown in Figure 3.11. Three sites, p_i, p_j and p_k, are shown as well as the moving sweep line, L. The arc shown by the dotted line represents the parabola of site p_j denoted by α_i and is still shown in the second two diagrams of Figure 3.11 to indicate how this parabola "falls out" of the *beach line*. At the point in time denoted by the second diagram, from the left, in Figure 3.11, parabola α_i shrinks to a mere point, q, on the *beach line*. This point, q is equidistant from the sweep line and each of the three sites, and a circle can therefore be drawn through the three sites, with lowest point on the sweep line. This point, q, indeed represents a vertex of the *Voronoi* diagram and there can not be any sites within this circle. This is easy to comprehend when one considers the fact that the arc, α_i, disappearing implies the meeting of two breakpoints and hence the meeting of two *Voronoi* edges. This scenario where the sweep line reaches the lowest point of a circle through 3 sites which define 3 consecutive parabolic arcs is defined as a *circle event* as shown in Figure 3.12. In [9] it is shown that this is the only way an arc can disappear from the beach line.

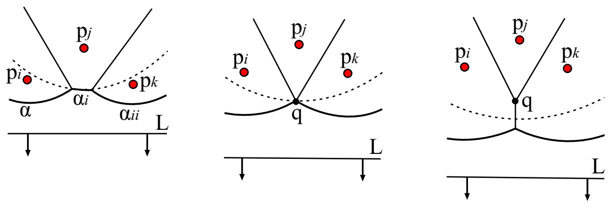


Figure 3.11: Parabolic arc shrinks and then disappears [9]

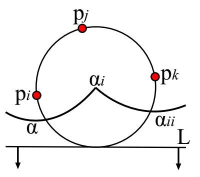


Figure 3.12: Circle event when parabolic arc disappears [9]

Two explicit events have thus been defined which represent the only two ways in which the structure of the *beach line* can change. These changes in structure in turn represent the discrete times at which information about the *Voronoi* diagram is updated. The problem is therefore reduced from a continuous problem of maintaining the ever changing *beach line* to a discrete problem where only certain events need to be identified and processed in order to calculate the *Voronoi* diagram.

It should be noted that the discussions of this section were by no means a detailed presentation of *Fortune's Voronoi* algorithm, but rather a brief summary of the core ideas. Several proofs and technical details have been omitted and the reader is referred to [9] for a detailed discussion. The implementation of an algorithm which makes use of the circle and site events, defined above, to calculate the *Voronoi* diagram requires a complex use of data structures.

Fortunately many versions of *Fortune's* algorithm have been implemented and perfected, in a wide variety of programming languages, over the years and are freely available. A decision was therefore made to make use of an already available C implementation of the algorithm which can be found at [24]. Only slight modifications were required to make this implementation, in C, fit into the project, as will be shown in the implementation section to follow.

A brief summary can however be given about the data structures used in the implementation [9]. Each component of the implementation is listed along with a brief description.

- The *Voronoi* diagram under construction - Stored in a doubly-connected edge list where the representation of the *beach line* allows access to relevant parts of the doubly-connected edge list.
- *The Beach line*- Represented by a balanced binary search tree where its leaves correspond to the arcs of the *beach line*. Each leaf stores the site of the arc which it represents. The internal nodes of this balanced binary tree represent the breakpoints where parabolic arcs meet by storing ordered couples of sites which correspond to the two consecutive parabolic arcs.
- The Event Queue - Implemented as a priority queue where the events are prioritized in terms of their respective y-coordinates from top to bottom. *Site events* are simply stored as the site itself since this is the only information required to know when this event will occur. *Circle events* are however not as easily identified. The algorithm makes sure that for every three consecutive sites (corresponding to three consecutive arcs on the *beach line*) the corresponding *circle event* is in the queue if the circle intersects the sweep line. This implies that the *circle event* is yet to occur since the sweep line has not yet reached the lowest part of the circle. The *circle event* is stored as the lowest point of the circle in terms of its y-coordinate. Any circle which contains another site in its interior or any circle which is completely above the sweep line is regarded as a false alarm and not added to the event queue.

It should be noted that during each event another event in the queue can be destroyed or created. A *site event* for instance will cause the appearance of a new arc on the *beach line* which implies new consecutive triples of arcs and hence new *circle events*. It is therefore imperative that all data structures are updated appropriately during each event which occurs. The way in which these structures are updated can be seen in [9].

The concepts of this algorithm remain abstract and difficult to visualize when merely presented in writing. The reader is therefore referred to [20], where an animated version of the algorithm is shown which greatly aids in understanding how the algorithm executes.

Implementation

The C implementation of *Fortune's* algorithm, found at [24], yields an output which fully defines the *Voronoi* diagram. It produces an output consisting of several rows of data where each row consists of an array which denotes specific information about the *Voronoi* diagram. One of four different types of arrays can be present in each row of the data output and these four types are summarized below,

- $[s\ a\ b]$ - Indicates that an input site, number s, was seen at x-coordinate, a, and y-coordinate, b.
- $[l\ a\ b\ c]$ - Indicates a straight line, number l, represented by $y = -\frac{a}{b}x + \frac{c}{b}$.
- $[v\ a\ b]$ - Indicates a vertex, number v, of the *Voronoi* diagram at x-coordinate, a, and y-coordinate, b.
- $[e\ l\ v_1\ v_2]$ - Indicates an edge, number e, of the *Voronoi* diagram which is a subsegment of line number l, and has end points at vertices numbered, v_1 and v_2. If v_1 or v_2 is -1 a line to infinity is indicated.

In the above summary all x-coordinates correspond to the East axis and all y-coordinates correspond to the North axis in the NE reference frame discussed in Chapter 5. This output of the algorithm is slightly modified so that each row of the output is defined by a number which denotes its type. From top to bottom, in the summary above, the integer numbers $1 - 4$ were used to denote the specific type of array and all arrays are then packed into a matrix which has its rows as these arrays.

A flow diagram of the entire *Voronoi* implementation is shown in Figure 3.13. The call for a *Voronoi population* is made from the *Path_Planner.c* module shown. The specific routines inside the *Path_Planner.c* module are obscured in this figure, and the figures of the following sections, in order to focus solely on the specific algorithm's implementation. These routines will be shown in more detail in Figure 4.9. It should be noted that the entire implementation shown in Figure 3.13 represents the contents of the *Calculate Voronoi Graph* block in Figure 4.11. The focus in this chapter is on the specifics of the individual algorithms and not on how these algorithms are integrated into the rest of the project. The reader is therefore referred to Figure 4.11 if at any time more information is required about the role of each algorithm within the entire UTV implementation.

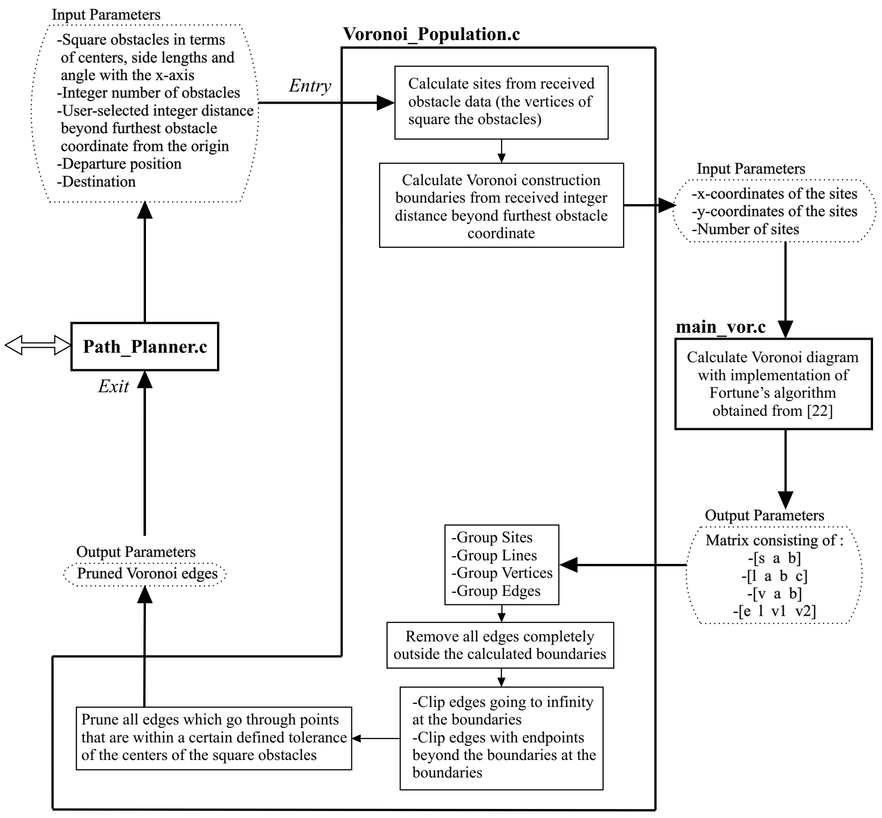

Input Parameters
-Square obstacles in terms of centers, side lengths and angle with the x-axis
-Integer number of obstacles
-User-selected integer distance beyond furthest obstacle coordinate from the origin
-Departure position
-Destination
Entry
Voronoi_Population.c
Calculate sites from received obstacle data (the vertices of square the obstacles)
Calculate Voronoi construction boundaries from received integer distance beyond furthest obstacle coordinate
Input Parameters
-x-coordinates of the sites
-y-coordinates of the sites
-Number of sites
main_vor.c
Calculate Voronoi diagram with implementation of Fortune's algorithm obtained from [22]
Output Parameters
Matrix consisting of :
-[s a b]
-[l a b c]
-[v a b]
-[e l v1 v2]
-Group Sites
-Group Lines
-Group Vertices
-Group Edges
Remove all edges completely outside the calculated boundaries
-Clip edges going to infinity at the boundaries
-Clip edges with endpoints beyond the boundaries at the boundaries
Prune all edges which go through points that are within a certain defined tolerance of the centers of the square obstacles
Output Parameters
Pruned Voronoi edges
Path_Planner.c
Exit

Figure 3.13: Call for a *Voronoi Population* from the Path Planning Module

Figure 3.13 shows how the C implementation of *Fortune's* algorithm, obtained from [24], is utilized to generate a preliminary *Voronoi* diagram, with the vertices of square obstacles defined as the sites for the calculation. The user specifies an integer value which represents a distance beyond the obstacle coordinate which is furthest from the origin of the two-dimensional reference frame. This value is then used to determine the bounding box of the *Voronoi* diagram and all *Voronoi* edges are clipped by this bounding box. *Voronoi* edges which fall completely outside the bounding box are removed from the populated set of line segments.

The final step then requires a pruning of all calculated *Voronoi* edges which pass through the centers of the square obstacles. Due to floating point operations minute offsets are occasionally present and thus line segments which are intuitively known to pass through the centres of the squares are not recognized as edges which should be pruned. A slight tolerance of 0.001 *m* was therefore implemented to ensure edges which should be pruned are not missed. The tolerance used is not a set constant value and depending on the scale of the area on which the *Voronoi* diagram is calculated the tolerance value can be increased and decreased accordingly. A value of 0.001 *m* yielded good results in this project. It should be noted however that when the square obstacles and the distances between them become infinitesimally small the possibility arises that other *Voronoi* edges will also fall within this tolerance and be pruned when they should not be pruned. The allowed tolerance is therefore a function of the size of the obstacles and the validity of the algorithm is bounded by the maximum accuracy which can be achieved with floating point operations. Within the scope of path planning for a UTV typical values are much larger and a minimum size can be stipulated for objects which are recognized as obstacles, during future implementations of obstacle detection. The algorithm therefore remains valid in this application, with the exception of singularities shown in the next section.

Results

With a *Voronoi population* algorithm implemented the only step remaining is an investigation into the performance of this algorithm. Routines were written in *Matlab* which write data to text files at different stages during execution. These text files were then used to write a routine which plots the execution of the algorithm in an animated fashion while also creating an *.avi* video. These videos can be seen on the DVD included with this thesis. Figure 3.14 shows the stage of the algorithm where the *Voronoi* diagram has been calculated and all appropriate edges marked for pruning.

After a removal of the pruned edges the populated diagram represents only line segments in the obstacle free space as shown in Figure 3.15. In this figure two characteristics of the algorithm are immediately identified. The first characteristic is the fact that the populated diagram, after pruning, displays characteristics very similar to the original *Voronoi* diagram, in the sense that segments tend toward the ideal situation where maximum distance is achieved from two adjacent obstacles simultaneously, by tracing out seg-

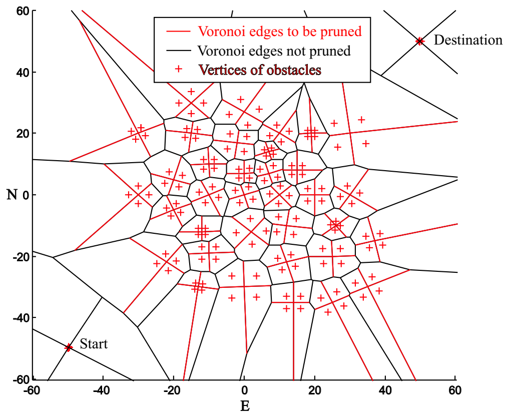


Figure 3.14: Pruning of *Voronoi* Edges

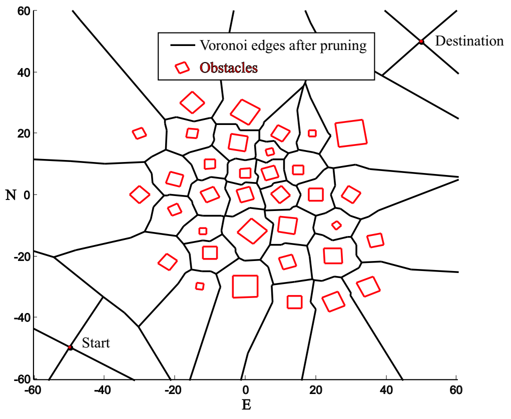


Figure 3.15: *Voronoi* Edges after Pruning

ments which are equidistant from obstacles on either sides of the segment. This in turn implies a maximum clearance from obstacles which makes the algorithm attractive, but at the cost of the second characteristic which is also identified. This second characteristic is the tendency of the populated diagram to be represented by a large number of short line segments with severe variation in heading angles. This ultimately implies a path which is not smooth and in turn implies less efficient motion of the UTV.

The UTV in this implementation is regarded as a point vehicle and the assumption is made that no two obstacles are in such close vicinity of each other that a *Voronoi* edge between them will imply a path segment with insufficient clearance for the UTV, due to the vehicle's size. Even under this assumption occasional failures of the algorithm can still occur. The algorithm generally tends to achieve maximum clearance from obstacles. However, when large

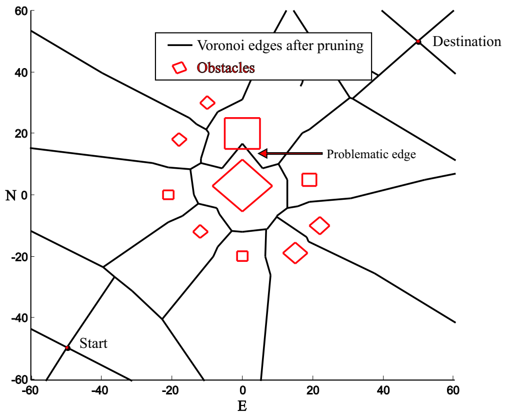


Figure 3.16: Incorrect collision with obstacle

obstacles are in close vicinity of each other and orientated as shown in Figure 3.16 the populated diagram can include an edge which enters the interior of an obstacle. This ultimately voids the validity of the algorithm since such a segment could imply a collision of the UTV with an obstacle. A method therefore needs to be found for pruning these edges as well. However, excessive pruning of edges is not desired since this could lead to scenarios where the start and destination points are no longer connected by the populated diagram. An example of this discontinuity is immediately evident when considering the fact that a single square obstacle would generate a *Voronoi* diagram which only consists of line segments going through its centre, and consequently have to be pruned, leaving no connection between start and destination. To accomodate this scenario a simple routine was implemented in both *shotest path algorithms* which are discussed later in this chapter. The routine constructs a rectangular path around the obstacle environment when no connection between start and destination is found. This *Voronoi* algorithm is therefore considered effective when the environment includes many obstacles which are not in close vicinity of each other and not orientated as shown in Figure 3.16.

A final note should be made about how the start and destination points are included in the algorithm. In order to connect these two points with the populated *Voronoi* diagram they are represented by two virtual square obstacles. These "obstacles" are defined with vertices which are 0.4 m apart, with the starting point and destination as their center points respectively. The value of 0.4 m is arbitrarily chosen, since this is the width of the UTV, but is of no consequence since any 4 vertices of a square will generate *Voronoi* edges exactly through the centre, if no other sites are in the interior of this square.

These virtual obstacles are orientated at the angle which is found when connecting the start and destination with a straight line. In doing this 4 *Voronoi* edges are yielded, for start and destination, which connects into the diagram and good results were also obtained with the start and destination amidst the obstacles. During the pruning process the *Voronoi* edges which are generated by these virtual squares are regarded as exceptions and not pruned.

This therefore concludes the *Voronoi* implementation of a *population* algorithm. As was noted in this section this algorithm is unfortunately not valid at all times but only yields good results under certain circumstances. The problematic areas therefore have to be addressed through modification of the algorithm or an entirely new algorithm should be considered. In Chapter 7 a better algorithm is briefly suggested as substitute for the *Voronoi* diagram.

3.2.2 *Visibility* Graph

Theory of the Visibility Graph

The previous section showed the implementation of a *population* method which yields line segments that approximate maximum clearance from obstacles, at the cost of longer paths and the limitation of only square obstacles being used. This section focuses on a *population* method in direct contrast to this. The *Visibility* graph is a method which can be applied directly to the path planning problem and accommodates not only square obstacles but convex polygon obstacles as well. The *Visibility* graph is a *population* method which is based on the concept of constructing straight lines between the vertices of obstacles in the two-dimensional plane. Consider the simple example of a *Visibility* graph shown in Figure 3.17.

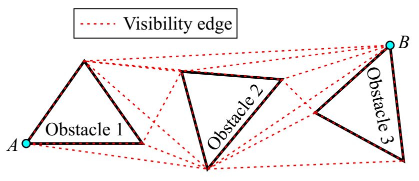


Figure 3.17: Simple *Visibility* Graph

In Figure 3.17 it is shown how the edges of the *Visibility* graph represent the straight lines from each vertex to each of the other vertices. An additional characteristic should however be noted which is the core reason why *Visibility* graphs can be considered for path planning purposes. As seen in Figure 3.17, each vertex is only connected, by means of the *Visibility* edges, to all other vertices which are visible. Edges which pass through obstacles are therefore ignored during the construction of the graph.

An additional characteristic can be identified when analyzing Figure 3.17. The populated Visibility graph represents a set of line segments which can be regarded as a *road map*. In order to get from any one vertex, *A*, to any other vertex, *B*, a path can be obtained from this *road map*. The important characteristic to note here is that, at all times the absolute shortest path between points *A* and *B*, without passing through obstacles, will be represented by a specific combination of the *Visibility* edges. This is best visualized when considering an elastic rubber band with endpoints fixed to points *A* and *B* respectively. If several such elastic bands are connected between *A* and *B*, each forced along the shape of a different possible path, and then released, all rubber bands will contract to become as short as possible. The shortest path from *A* to *B* will therefore be represented by one of these elastic rubber bands. All these rubber bands are indeed represented by the *Visibility* graph and by calculating the combination of *Visibility* edges which yields the shortest path in the populated diagram, the absolute shortest path is also found. For a formal proof of this refer to [9].

A note should be made about the obstacle clearance of the *Visibility* graph. As seen in Figure 3.17, a diagram which includes line segments which represent the absolute shortest path, between points *A* and *B*, comes at the cost of minimal clearance from obstacles. In fact, all line segments lie either on the edges of obstacles or pass through the vertices of obstacles, unless a straight line of visibility is present between points *A* and *B*. This implies paths which will not achieve obstacle avoidance but rather cause the UTV to collide partially or completely with obstacles. A method is therefore required to account for the fact that the UTV is not a point vehicle in reality and additional clearance from obstacles should be added. A simple method was used in the implementations of this project where all square obstacles were simply made virtually larger before being passed to the *Visibility* algorithm. This method is however not optimal and in the case of complex convex obstacles it becomes difficult to simply expand the size of the obstacles without an explicit algorithm. An efficient method for this purpose, the *Minkowski Sum*, was however found in hindsight, and is briefly discussed in Chapter 7.

Visibility Graph Algorithm

Several algorithms exist for calculating the *Visibility* graph. In this project an intuitive approach was followed during implementation of such an algorithm which will be discussed in the next section. This implementation yielded a computational cost of $O(n^3)$, where n is the total amount of obstacle vertices. In hindsight however, it was found that a more optimal algorithm exists which achieves a cost of $O(n^2 \log n)$. Since an optimal *Voronoi* algorithm was implemented, this section will introduce the optimal *Visibility* algorithm so that a comparison between the two *population* methods can be done without the one algorithm being unfairly disadvantaged due to poor implementation. This comparison is presented later in this chapter. The summary of this optimal algorithm, to follow, is based on the discussions in [9].

Consider a two-dimensional plane with disjoint polygonal obstacles, S, which have n vertices in total. In order to check whether any one vertex, w_i, is visible from a current vertex, P_j, it is intuitively obvious that an investigation into whether the line segment, $\overline{P_j w_i}$, intersects any of the obstacle edges, is required. All obstacle edges therefore have to be checked for intersection with line segment, $\overline{P_j w_i}$, yielding a computational cost of $O(n)$. Degenerate cases where $\overline{P_j w_i}$ intersects the interior of an obstacle, without intersecting it's edges and only passing through its vertices, also have to be checked for and will be addressed shortly. The computational cost is further increased when considering that all line segments, $\overline{P_j w_i}_{\ (i=1..\ n)}$, have to be checked against all obstacle edges, increasing the cost to $O(n^2)$. Furthermore, when considering that all obstacle vertices have to be used as the origin, $P_{j\ (j=1..\ n)}$, when checking visibility to all other vertices, the cost is increased even more to $O(n^3)$!

However, the optimal implementation of a *Visibility* algorithm attempts to reduce this computational cost of the problem by taking an ordered approach when considering the visibility between obstacle vertices. From each point, P_j, all surrounding vertices are sorted in clockwise order of angles with the starting position of a sweep line, ρ_0, shown in Figure 3.18.

The approach consists of moving the sweep line around the point, P_j, in a clockwise direction, while maintaining information about the edges which are currently intersecting the sweep line as well as information about the visibility of the previous vertex, w_{i-1}, which was checked. In doing so the process is accelerated since the visibility to the previous vertex, w_{i-1}, is used

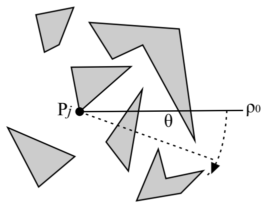


Figure 3.18: Sweep line at its initial position [9]

when deciding the visibility to the current vertex, w_i. Should both w_{i-1} and w_i therefore lie on the same segment, invisibility of w_{i-1} immediately implies invisibility of w_i. Visibility of w_{i-1} however, does not imply visibility of w_i, and additional tests have to be performed which will be discussed shortly.

The information about the obstacle edges which intersect the sweep line is stored in a balanced binary search tree, τ, where each leaf represents the edge which is intersected. From left to right the edges which are intersected are stored in ascending order of distance, along the sweep line, $\overline{P_j w_i}$, from the current vertex, P_j, from which visibility is checked.

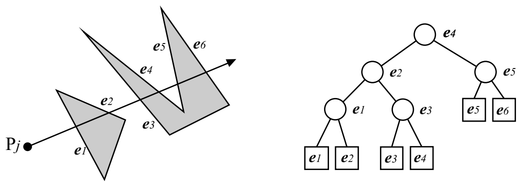


Figure 3.19: How obstacle edge intersections are stored in a BST [9]

This binary search tree (BST) can be seen in Figure 3.19. As shown, each internal node of the tree represents the rightmost leaf, e_{lr}, in the left subtree below it and all edges in the right subtree of this node are therefore further than e_{lr} from P_j, while all edges in its left subtree are closer or the same distance from P_j. Therefore when checking for the visibility of a vertex, instead of checking all obstacle edges for intersection with the sweep line, $\overline{P_j w_i}$, in a random manner, all that is now required is a check whether the edge in the leftmost leaf of the tree intersects the sweep line $\overline{P_j w_i}$.

As mentioned previously, degenerate cases where more than one obstacle vertex lie on the sweep line are indeed a possibility. Examples of these scenarios can be seen in Figure 3.20 where the previous vertex which was tested

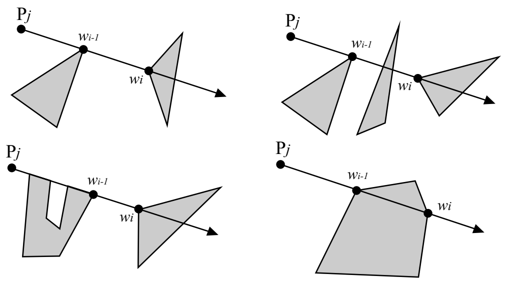


Figure 3.20: Special cases where the sweep line contains more than one vertex [9]

is denoted by w_{i-1} and the current vertex being tested is w_i.

These scenarios are treated as follows,

- If w_{i-1} is invisible, w_i is also invisible
- When w_{i-1} is visible there are two ways in which w_i can be invisible. If neither of these apply the vertex is visible. The two tests for invisibility are shown below
  - Is the segment $\overline{w_{i-1}w_i}$ intersected by an edge in τ?
  - Does $\overline{w_{i-1}w_i}$ lie within an obstacle of which both w_{i-1} and w_i are vertices?

A flow diagram of the entire implementation can be seen in Figure 3.21. It should be noted that the discussions in [9] do mention an added degenerate case. This is the case where the current vertex, P_j, falls on an obstacle vertex and the sweep line, $\overline{P_jw_i}$, falls entirely within an obstacle, since w_i falls on another vertex of the same obstacle. This case is however not addressed in the algorithm presented in [9] since the discussions are of a more general nature where P_j is not necessarily on one of the obstacle vertices.

A simple additional test is therefore required in the implementation shown in Figure 3.21. The additional test should test obstacle edges incident to P_j to determine whether the sweep line lies within an obstacle which has a vertex at P_j.

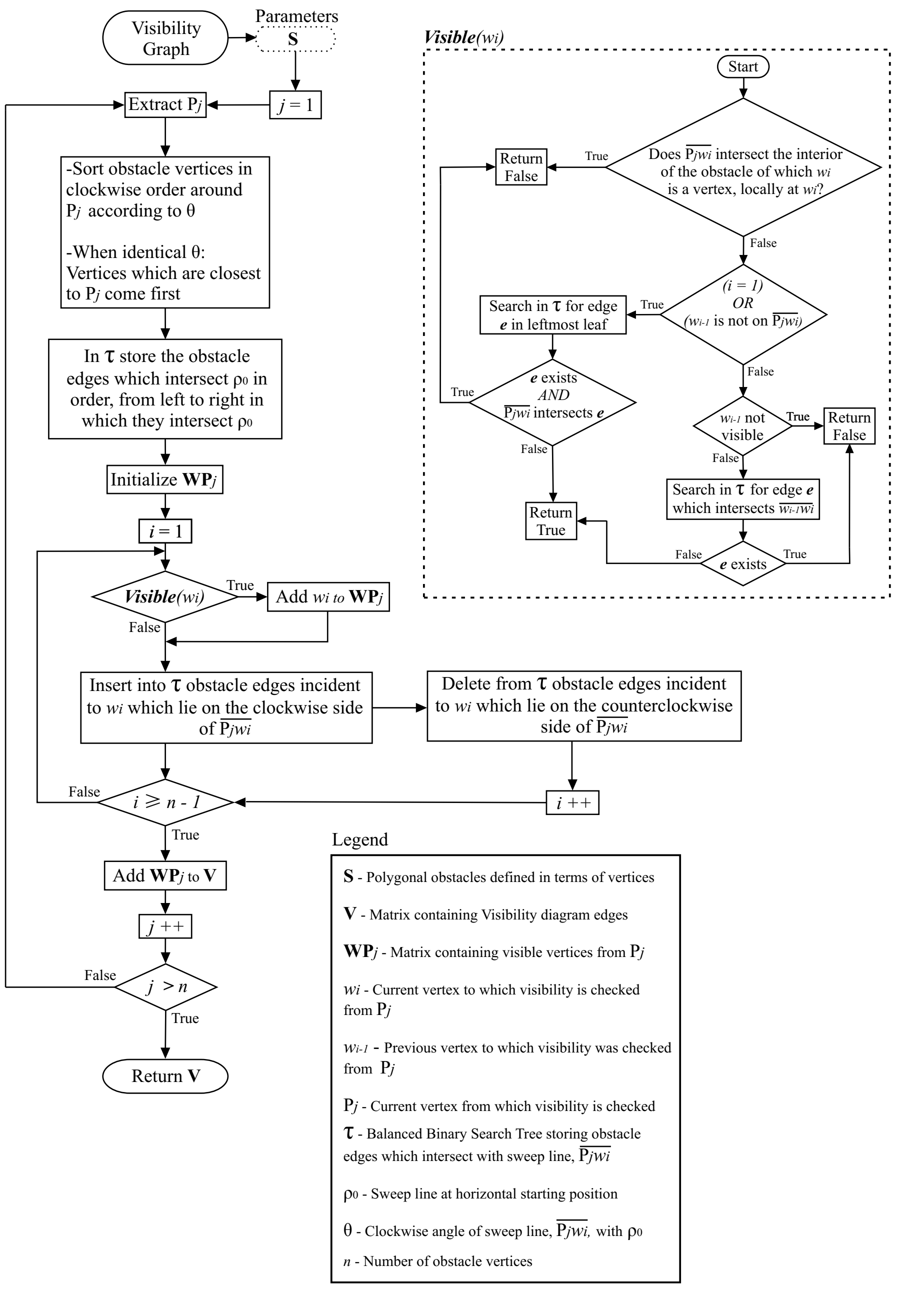

Visibility Graph
Parameters
S
Extract Pj
j = 1
-Sort obstacle vertices in clockwise order around Pj according to θ
-When identical θ: Vertices which are closest to Pj come first
In τ store the obstacle edges which intersect ρ0 in order, from left to right in which they intersect ρ0
Initialize WPj
i = 1
Visible(wi)
True
Add wi to WPj
False
Insert into τ obstacle edges incident to wi which lie on the clockwise side of Pjwi
Delete from τ obstacle edges incident to wi which lie on the counterclockwise side of Pjwi
i ++
False
i ⩾ n - 1
True
Add WPj to V
j ++
False
j > n
True
Return V
Visible(wi)
Start
Return False
True
Does Pjwi intersect the interior of the obstacle of which wi is a vertex, locally at wi?
False
Search in τ for edge e in leftmost leaf
True
(i = 1) OR (wi-1 is not on Pjwi)
False
True
e exists AND Pjwi intersects e
False
wi-1 not visible
True
Return False
False
Search in τ for edge e which intersects wi-1wi
Return True
False
e exists
True
Legend
S - Polygonal obstacles defined in terms of vertices
V - Matrix containing Visibility diagram edges
WPj - Matrix containing visible vertices from Pj
wi - Current vertex to which visibility is checked from Pj
wi-1 - Previous vertex to which visibility was checked from Pj
Pj - Current vertex from which visibility is checked
τ - Balanced Binary Search Tree storing obstacle edges which intersect with sweep line, Pjwi
ρ0 - Sweep line at horizontal starting position
θ - Clockwise angle of sweep line, Pjwi, with ρ0
n - Number of obstacle vertices

Figure 3.21: Flowchart of $O(n^2 \log n)$ *Visibility* Algorithm

Implementation

As previously mentioned the implementation of an algorithm to calculate the *Visibility* graph was done in an intuitive brute force manner with a computational cost of $O(n^3)$. Although not optimal, the *Visibility* graph was still constructed accurately, and with the relatively small amounts of obstacles used during simulation and practical tests, heavy loads were not present on the OBC or computer used for simulation. The entire implementation can be seen in Figure 3.22.

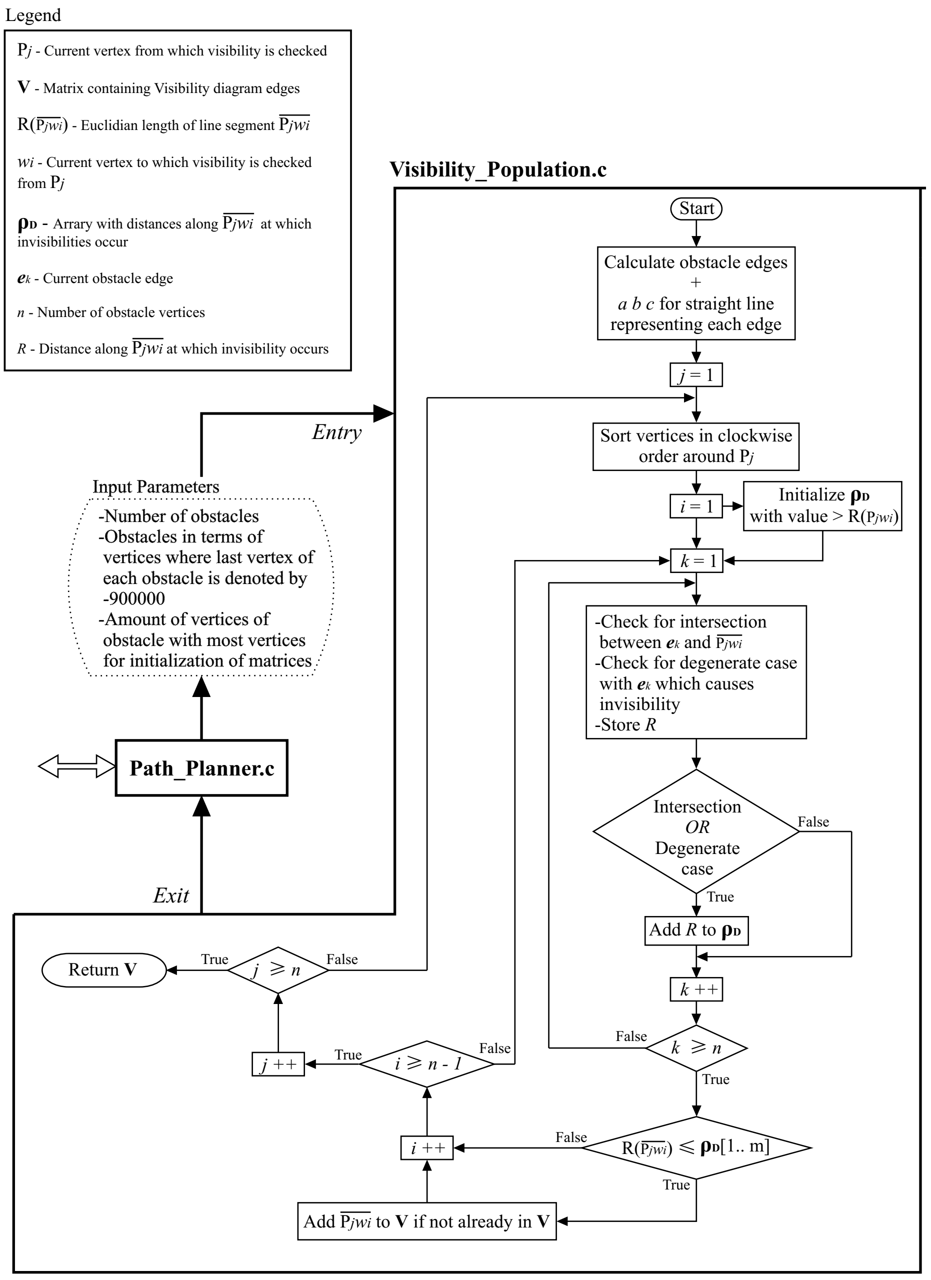

Legend
Pj - Current vertex from which visibility is checked
V - Matrix containing Visibility diagram edges
R(Pjwi) - Euclidian length of line segment Pjwi
wi - Current vertex to which visibility is checked from Pj
ρD - Arrary with distances along Pjwi at which invisibilities occur
ek - Current obstacle edge
n - Number of obstacle vertices
R - Distance along Pjwi at which invisibility occurs
Visibility_Population.c
Start
Calculate obstacle edges + a b c for straight line representing each edge
j = 1
Sort vertices in clockwise order around Pj
i = 1
Initialize ρD with value > R(Pjwi)
k = 1
-Check for intersection between ek and Pjwi
-Check for degenerate case with ek which causes invisibility
-Store R
Intersection OR Degenerate case
False
True
Add R to ρD
k ++
k ⩾ n
R(Pjwi) ⩽ ρD[1.. m]
Add Pjwi to V if not already in V
i ++
i ⩾ n - 1
j ++
j ⩾ n
Return V
Entry
Exit
Input Parameters
-Number of obstacles
-Obstacles in terms of vertices where last vertex of each obstacle is denoted by -900000
-Amount of vertices of obstacle with most vertices for initialization of matrices
Path_Planner.c

Figure 3.22: Flowchart of implemented $O(n^3)$ *Visibility* Algorithm

Similarly to the presentation of the *Voronoi* diagram, the routines inside the *Path_Planner.c* module are obscured to avoid complexity and the reader is once again referred to Figure 4.11. The nested while loops which lead to the $O(n^3)$ cost can clearly be seen in Figure 3.22. The basic approach toward computing the *Visibility* graph in this implementation is a calculation of all the points along the line segment, $\overline{P_j w_i}$, at which an obstacle edge causes invisibility. These invisibilities can either be caused by simple intersections of an obstacle edge, e_k, with the line segment, $\overline{P_j w_i}$, or by degenerate cases such as the ones discussed in the previous section.

Should an invisibility boundary be found along $\overline{P_j w_i}$ the corresponding distance from P_j is stored in an array, ρ_D. After comparing all obstacles edges, e_k, with the current sweep line, $\overline{P_j w_i}$, all values in the array, ρ_D, are compared to the distance of w_i from P_j. Should any of the values found in ρ_D be smaller than the Euclidean distance, $R(\overline{P_j w_i})$, an invisibility is implied and the edge is not added to the matrix representing the *Visibility* graph. In the opposite scenario $\overline{P_j w_i}$ is added to the *Visibility* graph under construction. This is then repeated for every vertex, representing $w_{i\ (i=1..\ n)}$, and then also repeated for every vertex, representing $P_{j\ (i=1..\ n)}$. In order to include the start and destination points in the *Visibility* graph they are simply defined as two additional obstacle vertices. It should be noted that the linear calculation methods used in this implementation only accommodate convex obstacles as stated in the beginning of this chapter. The *Visibility* algorithm can however be constructed for obstacles which are not convex, as shown in [9].

Results

Results that were obtained with the implemented $O(n^3)$ algorithm are now presented. Similarly to the *Voronoi* implementation, data is written to a text file during the execution of the algorithm, and then used in a different module to plot the *Visibility* edges against the obstacles. As shown in Figure 3.23 all possible *Visibility* edges are accurately constructed for a set of convex polygonal obstacles in the two dimensional plane. In contrast to the *Voronoi population* method there will always be a connection between start and destination, even in the degenerate case where there are no obstacles. In this scenario the *Visibility* graph simply consists of a straight line from start to destination. In this simple degenerate case the tendency of the algorithm to find the shortest possible path can already be identified. It is also seen how minimal clearances from obstacles are achieved and how some *Visibility* edges actually lie along the edges of the obstacles. The obstacles which are passed as parameters to the algorithm therefore have to be marginally larger than the actual obstacles during UTV operation and a method for ensuring this is suggested in Chapter 7.

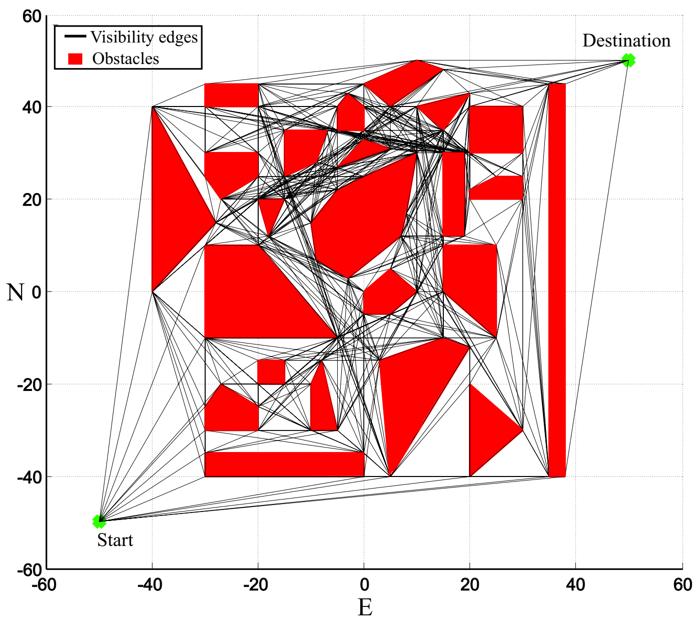


Figure 3.23: *Visibility* Graph generated by $O(n^3)$ algorithm with convex obstacles

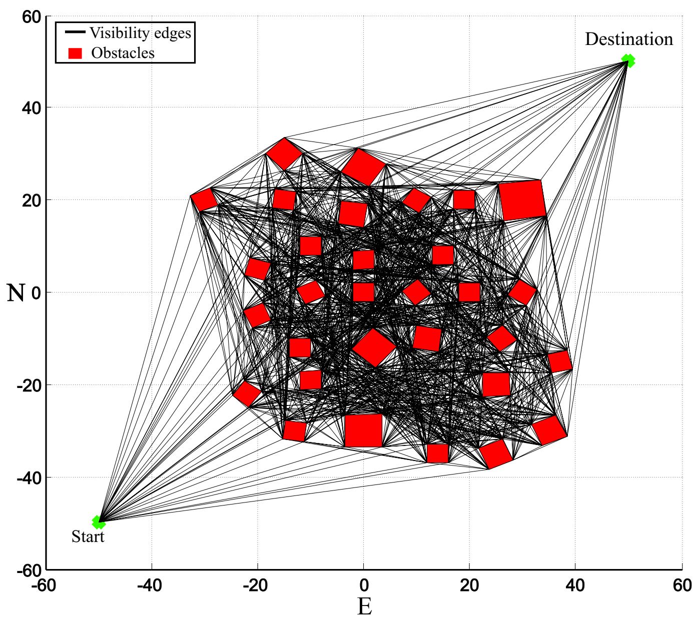


Figure 3.24: *Visibility* Graph generated by $O(n^3)$ algorithm with square obstacles

In Figure 3.24 the results of the same algorithm being used on a set of square obstacles is shown. At first glance, when comparing to the *Voronoi populations* previously presented, it is immediately evident how the computational cost of the algorithm has increased due to the increase in *population* edges which need to be found. Stated in different terms, for a certain amount of obstacle vertices, n, the *Visibility* graph consists of a significantly larger number of line segments than the *Voronoi* diagram. The guarantee of a path existing and that this is indeed the shortest path therefore comes at the cost of higher computational complexity. A final thing to note is the fact that in general the *Visibility* algorithm tends toward straighter paths with longer and less line segments in comparison with the *Voronoi* diagram, when trying to move from a point A to B. The *Visibility* graph *population* method is therefore attractive when smoother motion of the UTV is desired and the time it takes to move from A to B is important.

3.3 *Shortest Path* Algorithms

In the previous section the implemented *population* algorithms were discussed. However, the problem of finding an optimal path through the obstacle ridden environment is not yet solved. As mentioned in the beginning of this chapter, an additional step of finding the shortest path is required after having populated the free space in the two-dimensional plane. These algorithms are required to find a combination of the line segments from the populated sets, obtained from either the *Voronoi* or *Visibility* diagrams, which represents the shortest possible path available in the *population*, to get from points A to B. This section features a discussion on the two *shortest path algorithms* which were implemented in this project.

3.3.1 *Dijkstra*

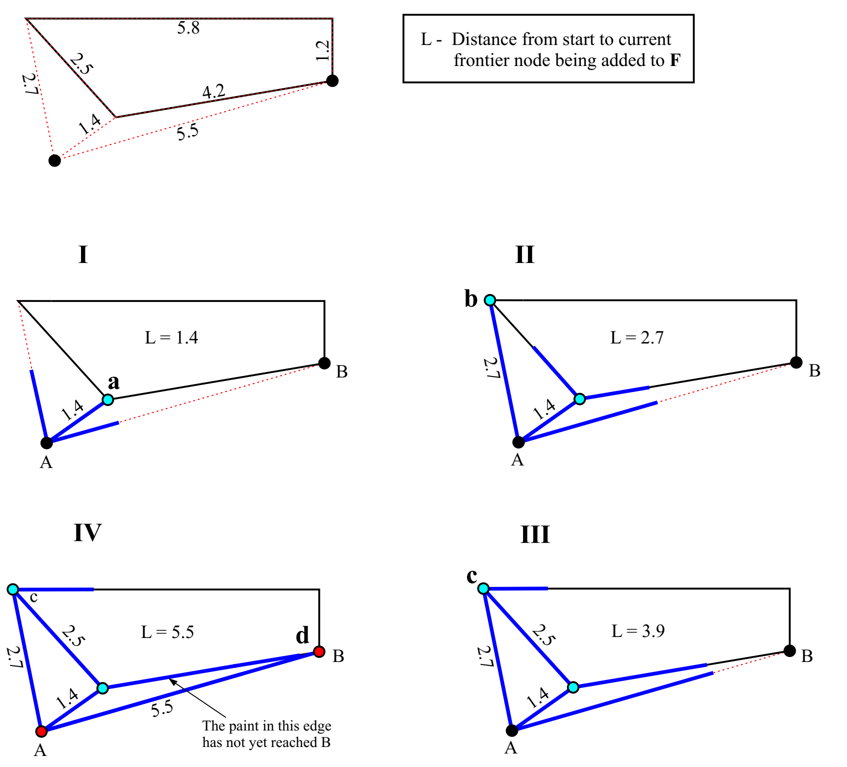
Algorithm

An attractive solution to the problem of finding the shortest path between two points, A and B, where both A and B represent specific nodes in a *population* set, is proposed by *Dijkstra* (1959). In [7] the idea behind *Dijkstra's* algorithm is discussed by means of an effective analogy which makes it easy to understand and conceptually visualize the behaviour of the algorithm. A similar approach is therefore followed in the discussions to follow.

Consider the simple example of a *Visibility* graph shown in Figure 3.25. All the *Visibility* edges are shown along with their respective Euclidean lengths. Suppose the shortest possible path is required between two nodes, A and B, which are both nodes of the *Visibility* graph, also shown in Figure 3.25. If all edges are considered as hollow pipes in the horizontal xy-plane, with identical diameters, then one can imagine a process of pouring paint into the pipes at the starting node, A. The paint is assumed to spread evenly through these imaginary pipes at a uniform rate of one unit of length for one unit of time. When thinking about this process intuitively it becomes obvious that as soon as the paint reaches a specific node for the first time, the time which it took to reach that node, **along the shortest possible path**, is known, since the paint moves through all pipes at the same rate, and more specifically the time which it took to reach that node represents the length of the shortest path. By storing information at each node about the node which paint reached it from, the path can be found as soon as the destination node is reached, by tracing back the path to the starting point A. In [7] an insightful comment is made, *'This is roughly equivalent to tagging each paint molecule with its path so far, so that when the first molecule reaches t, it's path is known'*, where t denotes B within the scope of these discussions.

Dijkstra exploits this by proposing an algorithm which simulates the process of paint spreading through the edges of the populated map. The algorithm he proposes further exploits the fact that the simulation of the paint spreading process can be done in discrete time steps rather than in continuous time. The basic principle of the algorithm is to maintain a *paint frontier*, **F**, which represents all the nodes which have currently been reached by the imaginary paint. The next step then requires looking for nodes, outside of **F**, which can be added to the *paint frontier* along one of the edges of the populated map. After finding all candidates the node which is added to **F** is the one or several which will yield the same shortest total distance increase from the origin. Throughout the process a tree, **T**, is also maintained which stores all the edges which have currently been filled with paint. These edges are added to **T** in the order in which the paint fills them. Edges which are first filled with paint therefore have a lower index in the tree. This is ultimately how the shortest path can be found. As soon as an edge is added to the tree, which has its one node (vertex) as the destination, the previous node from which paint reached it can be found by looking at the opposite node of this edge in **T**. A search can then be done in **T** from the opposite side (from the lowest index in **T**) for an edge which has an endpoint corresponding to this edge's starting point. The process is then repeated until the first edge connected to the origin is found.

Once again consider the four discrete time steps shown in Figure 3.25. As seen in **I**, paint is being poured into the pipes at the origin, A. The node which is first reached by the paint is node **a**. At this point in time **a** is added to **F** and the distance, 1.4, is stored in this index of **F**. At the same time the edge $\overline{A\mathbf{a}}$ is added to **T**. It is also seen how the paint has only partially filled the two adjacent *pipes*.



5.8
1.2
2.5
4.2
2.7
1.4
5.5
L - Distance from start to current frontier node being added to F
I
II
L = 1.4
L = 2.7
a
b
A
B
IV
III
L = 5.5
L = 3.9
c
d
The paint in this edge has not yet reached B

Figure 3.25: Paint spreading uniformly through *Visibility* Edges in discrete steps

After the next discrete time step, shown in **II**, the paint reaches node **b**. Node **b** is then added to **F** and the distance, 2.7, is stored in this index of **F**. The edge $\overline{A\mathbf{b}}$ is then added to **T**. In step **III** the same node is reached by the paint along a different *pipeline*. This indeed represents a special case which has to be accounted for in the algorithm. At this step the node **c** is not added to **F** since it is already in **F** with a corresponding shorter distance from the origin along a different path. The edge $\overline{\mathbf{ac}}$ is however marked as filled with paint and added to **T** with the node **a** as starting point since this node was first reached by the paint. In the last discrete step shown by **IV** the paint reaches the destination B. After this step the node **d** is added to **F** with the distance, 5.5, and the edge $\overline{A\mathbf{d}}$ is added to **T**. Since the destination has been reached by the paint the shortest possible path is immediately known.

Consider Table 3.1 which represents the current status of the tree, **T**. As mentioned all that is required is a check from the front (lowest index) of **T** for edges, with endpoint nodes corresponding to the starting node of the last

Index in T	*Visibility* edge	Starting node	Endpoint node
1	$\overline{A\mathbf{a}}$	A	**a**
2	$\overline{A\mathbf{b}}$	A	**b**
3	$\overline{\mathbf{ac}}$	**a**	**c**
4	$\overline{A\mathbf{d}}$	A	**d**

Table 3.1: Status of the tree **T** once the destination is reached

edge in **T**, and then repeating the process until the origin node is found. When looking at Table 3.1 no such search is required in this case and the shortest path is immediately identified as $\overline{A\mathbf{d}}$. Intuitively it is known that the shortest path between two points is a straight line between them and although this simple example does not claim to prove the validity of the algorithm it serves as insightful way of bridging the gap between the methods of the algorithm and simple intuition.

Implementation

An implementation of the algorithm was done in C programming language as shown in the Flowchart of Figure 3.26. The *Update **F** and **T*** block contains the routines shown in Figure 3.27 while the *Extract Path block* contains the routines shown in Figure 3.28. The flow diagram was split into these components to avoid clutter. When looking at these flow diagrams it becomes evident that it is not easy to identify the time complexity of the algorithm since the *while* loops are dependent on the amount of *population* edges passed to the algorithm. Also, some loops execute with varying repetitions since they are dependent on the size of the *frontier matrix*, **F**, which grows in size as the algorithm progresses toward the destination point.

If n is still regarded as the amount of obstacle vertices, and information is available about how many *population* edges are generally yielded by a specific *population* algorithm, then a crude bound of the time complexity of the implementations shown in these flowcharts can be determined. Consider the fact that a *Visibility* graph generally yields a quadratic number of edges, $O(n^2)$. [7] Assuming the worst case scenario where all nodes, except the destination node of the *Visibility* graph, are already in the *frontier* matrix, and considering the fact that there are n nodes in a *Visibility* graph, it becomes evident that the internal loop with regards to the size of **F**, shown in Figure 3.26, can have a time complexity of approximately $O(n)$. However this loop grows from $O(1)$ to $O(n)$ since at first there is only one node in **F**. When noticing this

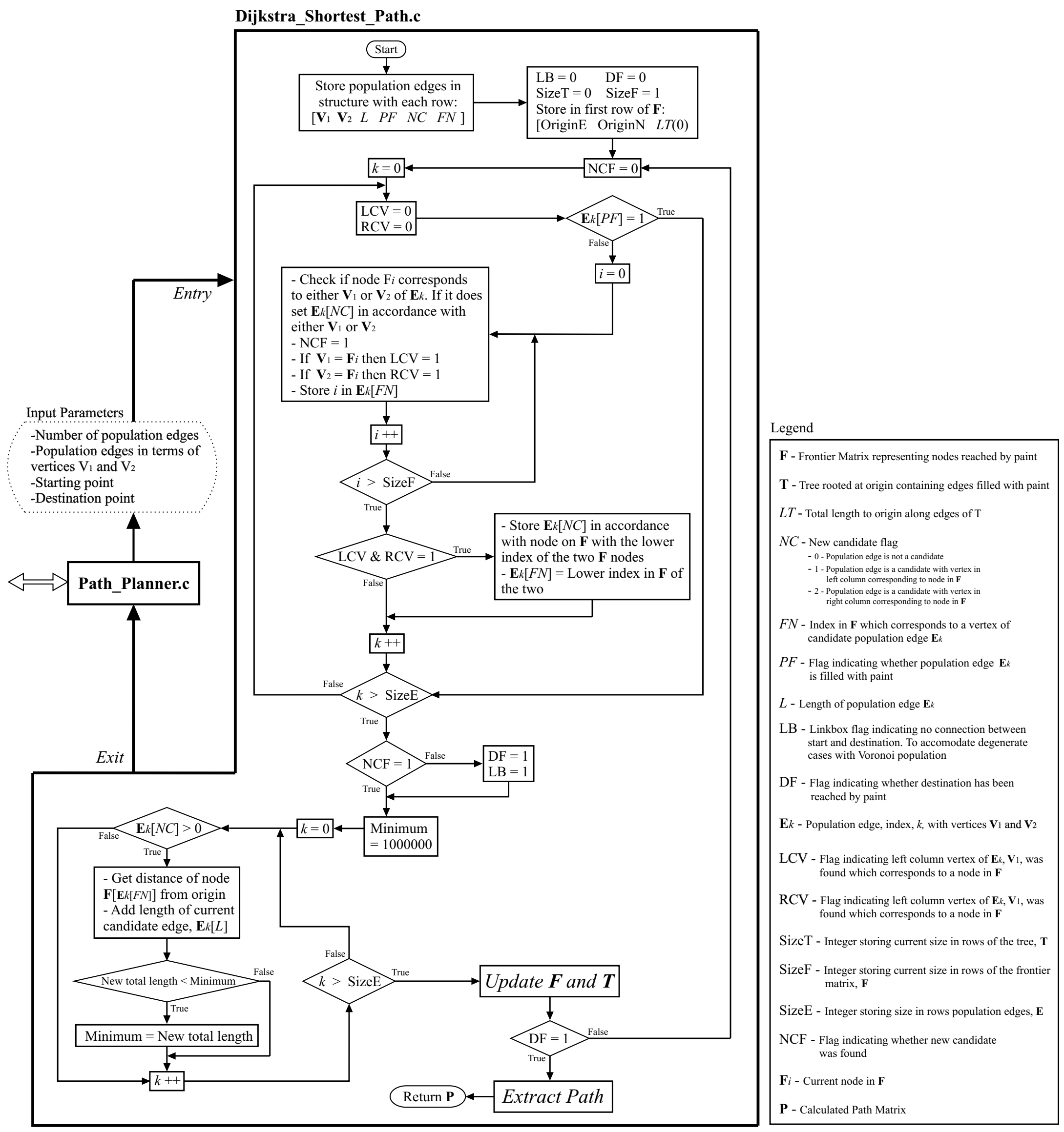

Dijkstra_Shortest_Path.c
Start
Store population edges in structure with each row: [V1 V2 L PF NC FN]
LB = 0 DF = 0
SizeT = 0 SizeF = 1
Store in first row of F: [OriginE OriginN LT(0)]
k = 0
NCF = 0
LCV = 0
RCV = 0
Ek[PF] = 1
True
False
i = 0
- Check if node Fi corresponds to either V1 or V2 of Ek. If it does set Ek[NC] in accordance with either V1 or V2
- NCF = 1
- If V1 = Fi then LCV = 1
- If V2 = Fi then RCV = 1
- Store i in Ek[FN]
i ++
i > SizeF
LCV & RCV = 1
- Store Ek[NC] in accordance with node on F with the lower index of the two F nodes
- Ek[FN] = Lower index in F of the two
k ++
k > SizeE
NCF = 1
DF = 1
LB = 1
Minimum = 1000000
k = 0
Ek[NC] > 0
- Get distance of node F[Ek[FN]] from origin
- Add length of current candidate edge, Ek[L]
New total length < Minimum
Minimum = New total length
k ++
k > SizeE
Update F and T
DF = 1
Extract Path
Return P
Entry
Exit
Input Parameters
-Number of population edges
-Population edges in terms of vertices V1 and V2
-Starting point
-Destination point
Path_Planner.c
Legend
F - Frontier Matrix representing nodes reached by paint
T - Tree rooted at origin containing edges filled with paint
LT - Total length to origin along edges of T
NC - New candidate flag
- 0 - Population edge is not a candidate
- 1 - Population edge is a candidate with vertex in left column corresponding to node in F
- 2 - Population edge is a candidate with vertex in right column corresponding to node in F
FN - Index in F which corresponds to a vertex of candidate population edge Ek
PF - Flag indicating whether population edge Ek is filled with paint
L - Length of population edge Ek
LB - Linkbox flag indicating no connection between start and destination. To accomodate degenerate cases with Voronoi population
DF - Flag indicating whether destination has been reached by paint
Ek - Population edge, index, k, with vertices V1 and V2
LCV - Flag indicating left column vertex of Ek, V1, was found which corresponds to a node in F
RCV - Flag indicating left column vertex of Ek, V1, was found which corresponds to a node in F
SizeT - Integer storing current size in rows of the tree, T
SizeF - Integer storing current size in rows of the frontier matrix, F
SizeE - Integer storing size in rows population edges, E
NCF - Flag indicating whether new candidate was found
Fi - Current node in F
P - Calculated Path Matrix

Figure 3.26: Main Flow Diagram of Implemented Dijkstra Algorithm

loop is nested within another loop with regards to the amount of *population* edges, and remembering there are generally n^2 edges in a *Visibility* graph, the total complexity of the two inner loops can possibly grow from $O(n^2)$ to $O(n^3)$. Both these loops are nested within the main loop which represents the flow of paint through the *pipes*. A crude worst case time complexity bound of more than $O(n^3)$ is therefore found for the implemented algorithm, when a *Visibility* graph is used as *population* method. This implementation is however far from optimal and in [7] it is mentioned that the algorithm can be implemented to run with a complexity of $O(n^2)$.

This optimal time complexity will therefore be used during comparisons

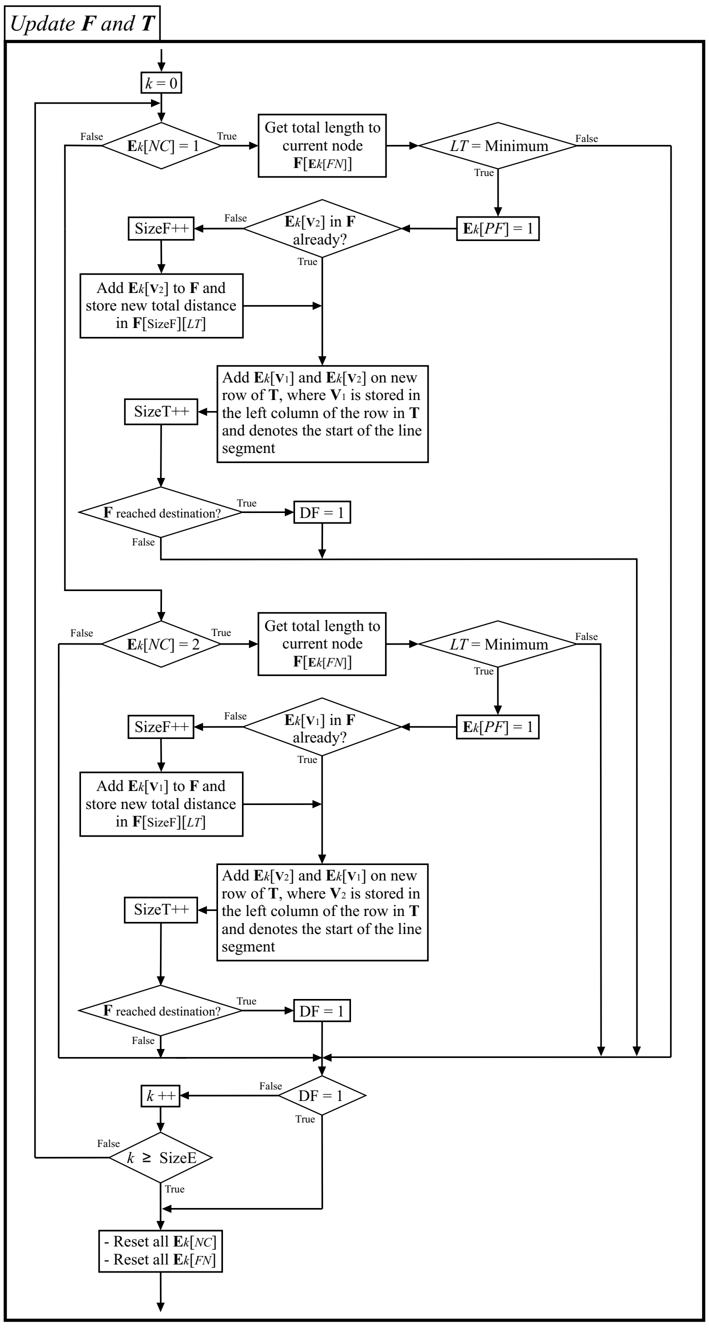

Update F and T
k = 0
Ek[NC] = 1
False
True
Get total length to current node F[Ek[FN]]
LT = Minimum
False
True
Ek[PF] = 1
Ek[V2] in F already?
False
True
SizeF++
Add Ek[V2] to F and store new total distance in F[SizeF][LT]
Add Ek[V1] and Ek[V2] on new row of T, where V1 is stored in the left column of the row in T and denotes the start of the line segment
SizeT++
F reached destination?
True
False
DF = 1
Ek[NC] = 2
Get total length to current node F[Ek[FN]]
LT = Minimum
Ek[PF] = 1
Ek[V1] in F already?
SizeF++
Add Ek[V1] to F and store new total distance in F[SizeF][LT]
Add Ek[V2] and Ek[V1] on new row of T, where V2 is stored in the left column of the row in T and denotes the start of the line segment
SizeT++
F reached destination?
DF = 1
DF = 1
k ++
k ≥ SizeE
- Reset all Ek[NC]
- Reset all Ek[FN]

Figure 3.27: Update **F** and **T**

since all algorithms thus far have been quantified in terms of their optimal implementation. When considering the case where the *Voronoi* diagram is used as *population* method the time complexity of the problem drastically decreases. As stated in [9] a *Voronoi* diagram has at most $3n - 6$ edges. Since constants are ignored when analyzing time complexities the amount of edges of the *Voronoi* diagram can be regarded as n. The total time complexity of the implemented *Dijkstra's* algorithm is therefore reduced to a crude bound above $O(n^2)$ in this non-optimal implementation of the algorithm. With the algorithm mentioned in [7] it can possibly be decreased even more toward $O(n)$. Once again this optimal time complexity will be used in comparisons.

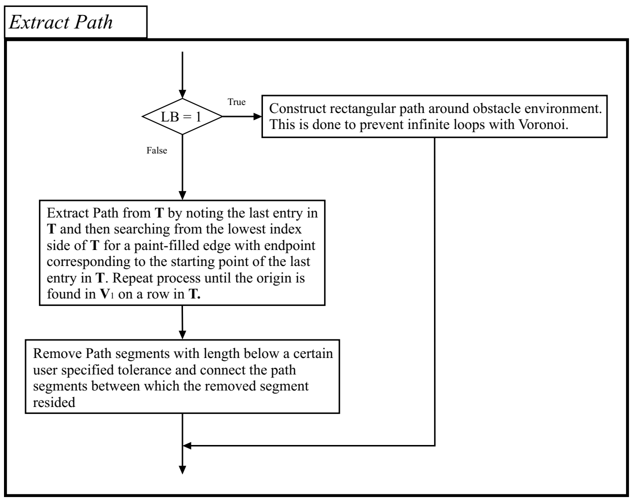

Extract Path
LB = 1
True
False
Construct rectangular path around obstacle environment. This is done to prevent infinite loops with Voronoi.
Extract Path from T by noting the last entry in T and then searching from the lowest index side of T for a paint-filled edge with endpoint corresponding to the starting point of the last entry in T. Repeat process until the origin is found in V₁ on a row in T.
Remove Path segments with length below a certain user specified tolerance and connect the path segments between which the removed segment resided

Figure 3.28: Extract Path from **T**

A final note can be made about the *Extract Path* block with the contents shown in Figure 3.28. These routines represent the final stages of the algorithm where the populated tree, **T**, is used to extract the shortest path calculated by the algorithm. This is done in a manner as discussed previously where the path is traced back from the destination to the origin, by finding lowest index entries in **T**, which have endpoints as the starting points of the previous edge treated in **T**, where the last index in **T** is treated first. Two additional routines were implemented to accommodate the degenerate cases of the *Voronoi* algorithm where the populated set yields no connection between starting point and destination. In such a case the linkbox flag is set and a rectangular path is simply constructed around the entire two-dimensional reference system along the boundaries defined.

The other routine was implemented to remove path line segments from the extracted path which have lengths below a certain user specified threshold. This was implemented so that the UTV is not unnecessarily burdened with infinitesimally short path segments before stopping and turning to new headings. It should be noted that when this tolerance is too large path segments may be removed which were required to avoid a certain obstacle, since the previous path segment and next path segment in the sequence, before and after the path to be removed, are connected by simply making the endpoint of the previous segment the starting point of the next segment. Care should therefore be taken when deciding this tolerance and the scale of the obstacle ridden environment is an important factor to consider. This routine is generally only applicable when using a *Voronoi population* where very short line segments can occur. In the case of a *Visibility population* however paths are generally represented by longer line segments and short path line segments which may occur are most often compulsory for obstacle avoidance. The routine therefore becomes redundant in the case of a *Visibility population*.

Results

In order to test the implemented *Dijkstra* algorithm the entries in the tree, **T**, were written to a text file during execution. From the same plotting routines used for the *Voronoi* and *Visibility* diagrams, these edges of **T** are then extracted in the same order they were added to **T**, and superimposed, one by one, on the corresponding populated graph. The speed at which the edges in **T** (the edges filled with paint thus far) are plotted can be varied by the user so as to avoid excessively long delays when the populated graph has a large number of edges. The edges are therefore plotted in an animated fashion to give a visual representation of how the algorithm executes.

Consider Figure 3.29. The 4 plots in this figure can be thought of as "still frames" that were captured at 4 specific moments in time during the algorithm's execution. In plot **I** it is seen how the *paint frontier* has progressed only partially into the populated set. In plot **II** the *paint frontier* progresses further with more nodes added to **F** and more edges present in **T**. The *paint frontier* then progresses further through plots **III** and **IV**.

As soon as the *paint frontier* reaches the destination point the path is known and extracted from **T** in the manner discussed previously. Figure 3.30 shows this extracted path. As shown, the path approximates a straight line and gives a good indication that the algorithm functions correctly. In this figure the heavy, $O(n^2)$, load on *Dijkstra*, with respect to the amount of *Visibility* edges, can also be seen. This heavy computational cost of *Dijkstra's* algorithm can be contributed to the fact that the algorithm actually achieves more than what is required. When taking a closer look it is seen that this algorithm not only computes the shortest path to the destination but in fact the shortest path to any node in **F**! Stated in different terms the shortest path to any point,

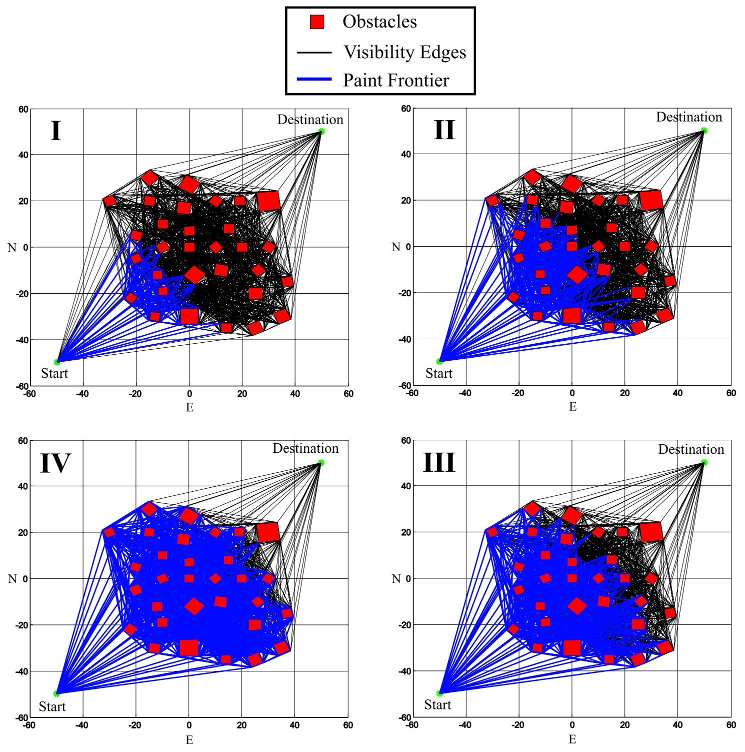


Figure 3.29: Different stages of implemented *Dijkstra* Algorithm

which is the node of any line segment already filled with paint, can be found through a simple search in **T**.

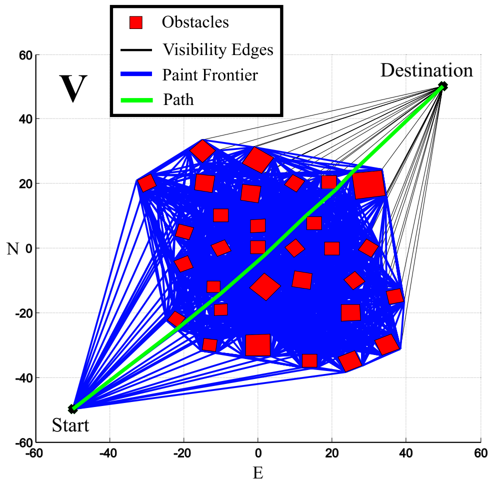


Figure 3.30: Path found with implemented *Dijkstra* Algorithm

Dijkstra's algorithm can therefore be considered as a brute force method for calculating more than what is required for a path planning application where the destination is known. In the next section it will be seen how *Dijkstra's* algorithm can be modified slightly to yield a new algorithm with a much lower computational cost. In Figure 3.31 it is seen how the implemented *Dijkstra's* algorithm also yields good results when convex polygonal obstacles are used in conjunction with the *Visibility* graph. In this plot it is seen how the larger convex obstacles imply less edges of visibility and therefore a marginally decreased load on *Dijkstra*. The computational cost is however still high.

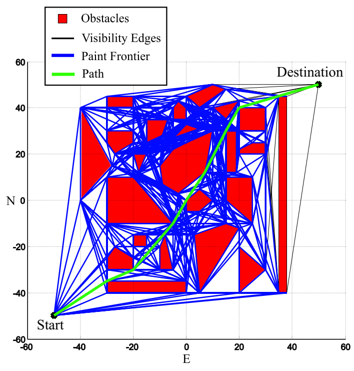


Figure 3.31: Path found with *Dijkstra* Algorithm for Convex obstacles

In order to see the drastically decreased load on *Dijkstra*, when the *Voronoi* diagram is used as *population* method, refer to Figure 3.32. It is clearly seen how the large decrease in *population* edges drastically decreases the load on the *Dijkstra* implementation. As mentioned previously, the time complexity of the algorithm is decreased (with an optimal implementation) toward $O(n)$ when *Voronoi* is used.

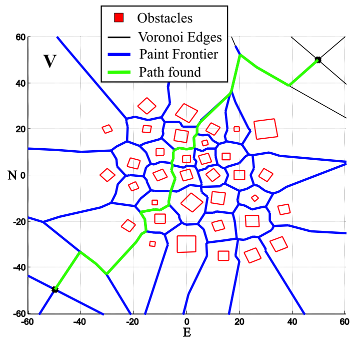


Figure 3.32: Path found with *Dijkstra* Algorithm on *Voronoi population*

The 4 still frames which show the progress of the *paint frontier* is also shown for the *Voronoi* case in Figure 3.33. This section on *Dijkstra's* algorithm can therefore be concluded with a brief summary on the characteristics of the algorithm. As shown, the algorithm is capable of accurately determining the shortest path from a populated set generated by either the *Voronoi* or the *Visibility population* method. The algorithm however proves cumbersome and computationally expensive when used in conjunction with the *Visibility* graph. In the case of the *Voronoi* diagram however, it is much more efficient and yields good results at an acceptable computational cost.

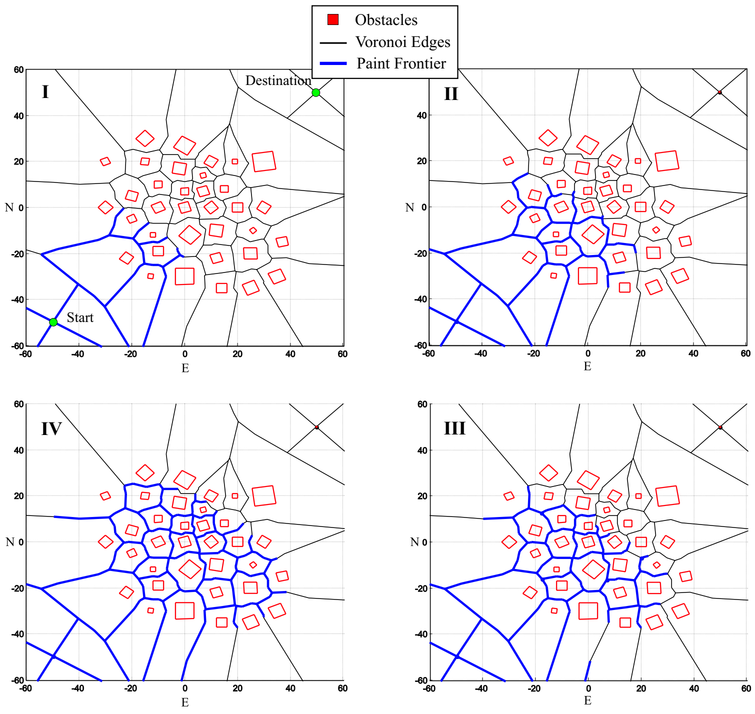


Figure 3.33: Different stages of *Dijkstra* Algorithm on *Voronoi population*

A final note can be made about *Dijkstra's* algorithm and the fact that it "over calculates" the problem by finding the shortest routes to all nodes in **F** rather than just to the destination. Although of no use in this project, this characteristic can be exploited in a different unmanned terrestrial vehicle application where several possible destinations exist and the route to the nearest one is required. An example of this would be a robotic UTV which is required to gather resources from any one of several known resource points. This is merely a vague example but encourages thought on the large number of possible applications of *Dijkstra's* algorithm.

3.3.2 *A*Star*

Algorithm

As noted in the previous section, *Dijkstra's* algorithm achieves more than what is required for this path planning application. In this project only a shortest path to a known destination is required and yet *Dijkstra's* algorithm calculates the shortest path to several nodes while finding the shortest route to the destination. This proves inefficient due to high computational costs, especially when used on the *Visibility* graph, and is not application specific but rather a solution to a more general problem. A highly attractive alternative is the *A*Star* search.

*A*Star* can also be visualized as an algorithm simulating the process of paint spreading through pipes, and is in fact very closely related to *Dijkstra's* algorithm. However, in the *A*Star* case the algorithm can conceptually be regarded as spreading paint through the pipes (*population* edges) in a biased manner by favouring certain pipes over others. *A*Star* accomplishes this by adding a certain cost function, also known as a *heuristic*, which is continuously minimized while deciding on which *population* edge should be filled with paint next. This cost function is referred to as a *heuristic* because it is generally an approximation of a certain cost yielded when a certain *population* edge is chosen. The *heuristic* function which is chosen differs from application to application and is strongly dependent on the environment in which the algorithm is used. Since an optimal path is the main consideration in this project, a *heuristic* was chosen which bases its cost on the Euclidean distance to the destination, should a certain *population* edge be chosen. It should be noted that this is not the only possible *heuristic* and in fact there are a wide variety of *heuristics* to choose from. In some applications for instance the cost is not only based on the length of the path but also on the specific terrain where a UTV would for instance prefer a path along a road rather than a path going through hazardous terrain. Another attractive possibility, for airborne applications, is a *heuristic* which adds a penalty to paths which imply severe changes in heading angle.

Since this algorithm no longer "spreads paint" uniformly and evenly through the populated grid a more appropriate way of referencing the paint frontier would be to consider it a closed set of edges which have been forcefully and selectively filled with paint. The nodes of **F** now represent the nodes of this closed set and all *population* edges incident to these nodes of **F** are regarded as the open set. The *closed set* indeed also represents the tree, **T**, from which the path is ultimately extracted in the same way as is the case in the *Dijkstra* implementation.

The *heuristic* function, f, used in this project, is defined as follows,

$$f = G + H \tag{3.3.1}$$

where G is defined as the cost to go, and H is the cost remaining after having gone. Stated in more specific terms, G is the new accumulated distance from the origin, along the *population* edges in **T**, should a specific new *population* edge candidate be selected and added to the *closed set*. It can already be noted that G is in fact merely the cost function minimized by *Dijkstra's* algorithm. The only difference with *A*Star* is the addition of the term, H, to the cost function, which is chosen in this project as the Euclidean distance from the other end of the new candidate edge (the node of this edge not yet in **F**) to the destination. When considering the fact that *A*Star* attempts to minimize this cost function with the selections it makes, it is already evident that the algorithm will favour straight paths over paths implying large detours, since the shortest distance between two points, A and B, is a straight line, and an optimal candidate to minimize this cost function is in fact a *population* edge lying on this straight line between start and destination.

Consider the simple example of a *Visibility* graph shown in Figure 3.34. The Euclidean lengths of all *Visibility* edges between starting point, A, and destination, B, are shown. Consider the discrete point in time, labeled **1**, in this figure. The first step of the algorithm is to initialize the *closed set frontier* matrix, **F**, with the starting node, A. At this point in time all edges incident to the nodes of **F** are considered, as indicated, and their corresponding *f scores* calculated. The *f scores* are calculated as the sum of each edge's length and the Euclidean length between its node, not in **F**, and the destination. The edge incident to **F** yielding the lowest *f score* is then added to the *closed set*, as seen in the second discrete time step. At this point in time all the new edges, incident to **F**, are considered again and their corresponding *f scores* calculated. Immediately the edge incident to **F**, from the destination, is found to yield the lowest *f score*, added to the *closed set* and the path is found.

When remembering the way in which *Dijkstra* executes, it becomes evident that several of the edges in Figure 3.34 would have been filled with paint before the path was found. This simple example illustrates how *A*Star*, with an Euclidean *heuristic*, always directs itself toward the goal along a path approximating a straight line.

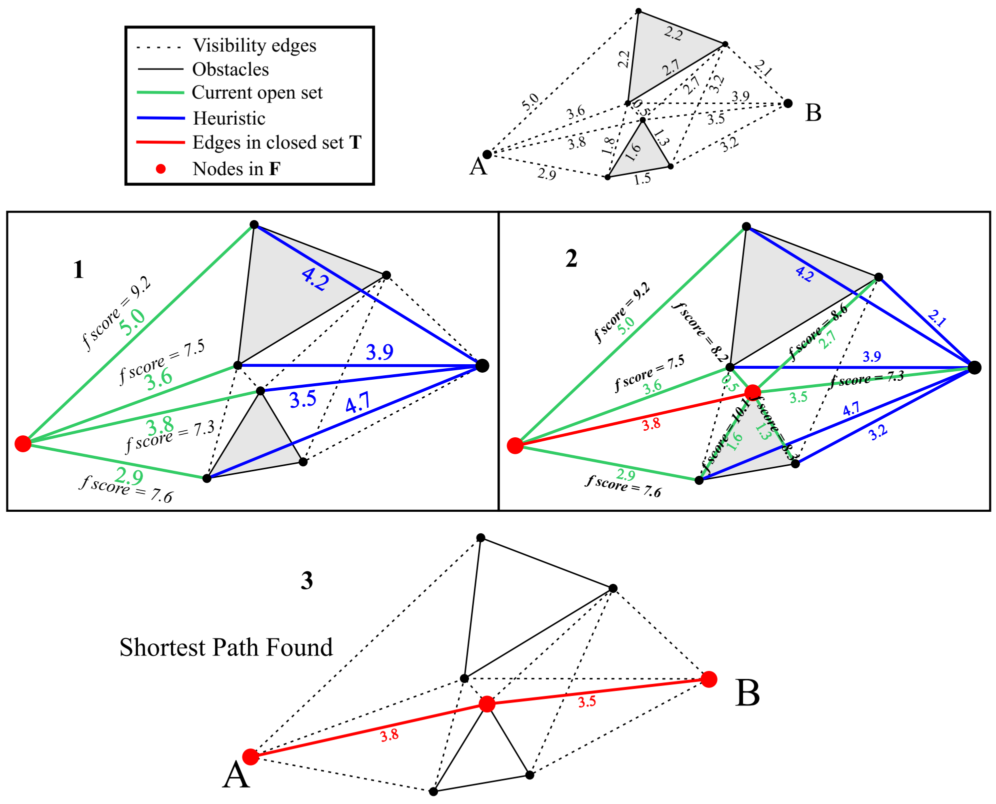


Figure 3.34: Simple example of an *A*Star* search with Euclidean Heuristic

In some cases however a path approximating a straight line does not exist due to large obstacles forcing shortest paths which consist of detours. Consider the discrete steps shown in Figure 3.35. In this figure a *Visibility* graph is shown for two obstacles where one large obstacle prevents straight line paths from being available. The progress of the algorithm is the same in these steps as in the previous example. It should however be noted how nodes, which are already in **F**, are not added to **F** again, as shown in discrete step **5**. This is done for the same reason stated during the discussions on *Dijkstra*. In steps **5** and **6** the new edges are therefore added to the *closed set*, **T**, with starting points defined as the node, of the specific edge, which was first added to the *closed set* but the nodes are not added to **F**.

In this figure it is seen how *A*Star* once again attempts to find a shortest path along a straight line but fails in doing so and then starts to add more edges to the *closed set* before finding the shortest path. The algorithm is therefore

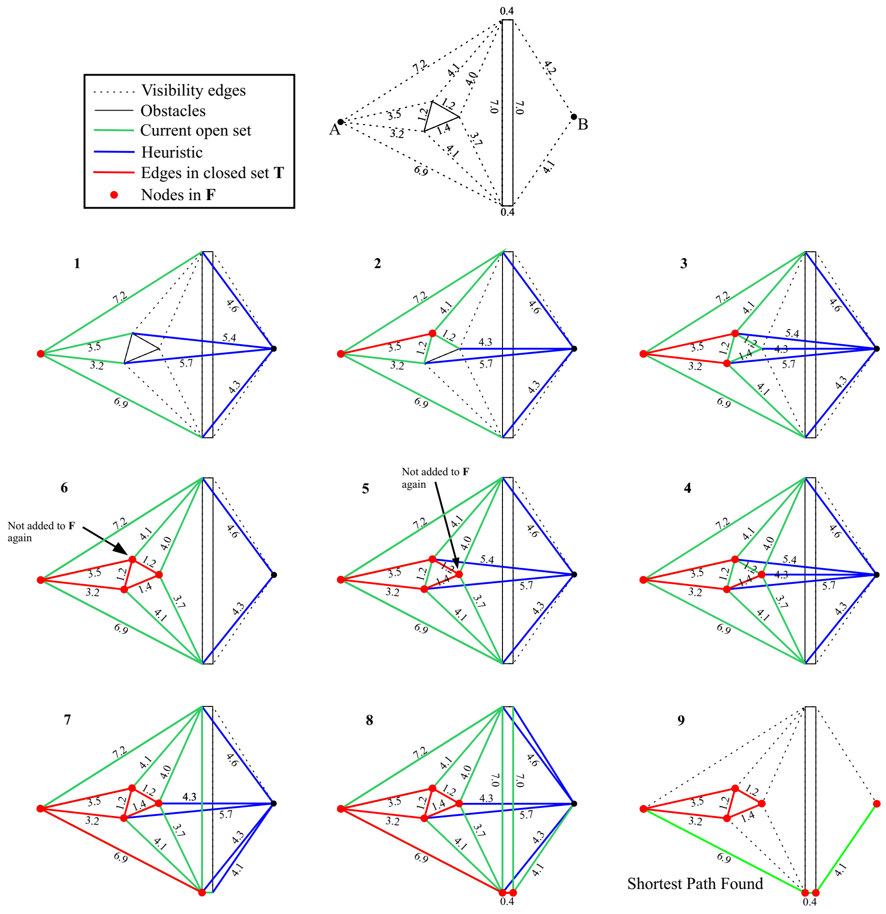


Figure 3.35: Example of an *A*Star* search with detours

slowed down and this can be contributed to the fact that in this case the approximations of the Euclidean *heuristic* function are inaccurate. More specifically the *heuristic* function drastically underestimates the true cost after having gone, H. When noting that *Dijkstra* is in fact a degenerate case of an *A*Star* algorithm, where H is simply zero, it becomes evident that a *heuristic*, which underestimates H, approaches a *Dijkstra* expansion. Relatively good results are however still obtained and although not as fast as in the first example the path is still found in less steps than what would have been the case if *Dijkstra* was used. To illustrate this the steps taken by *Dijkstra* in finding the shortest path in the same obstacle environment is shown in Figure 3.36. It can clearly be seen that *A*Star* accomplishes what *Dijkstra* accomplishes in less discrete time steps by adding fewer edges to the *closed set* whereas *Dijkstra* fills the majority of the edges with paint before finding the optimal path.

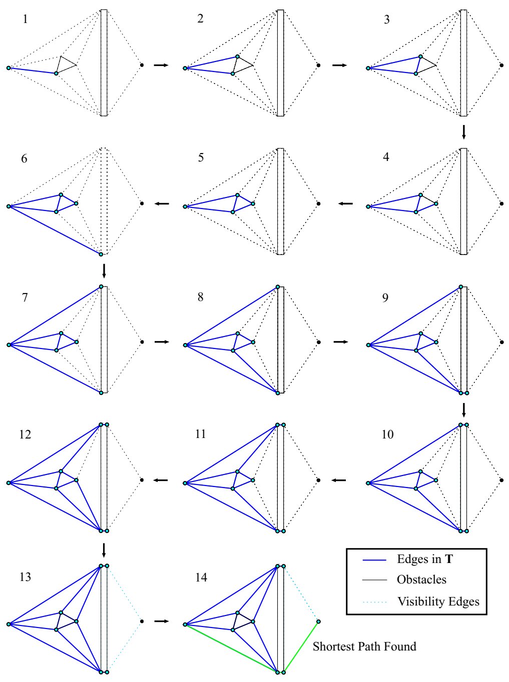


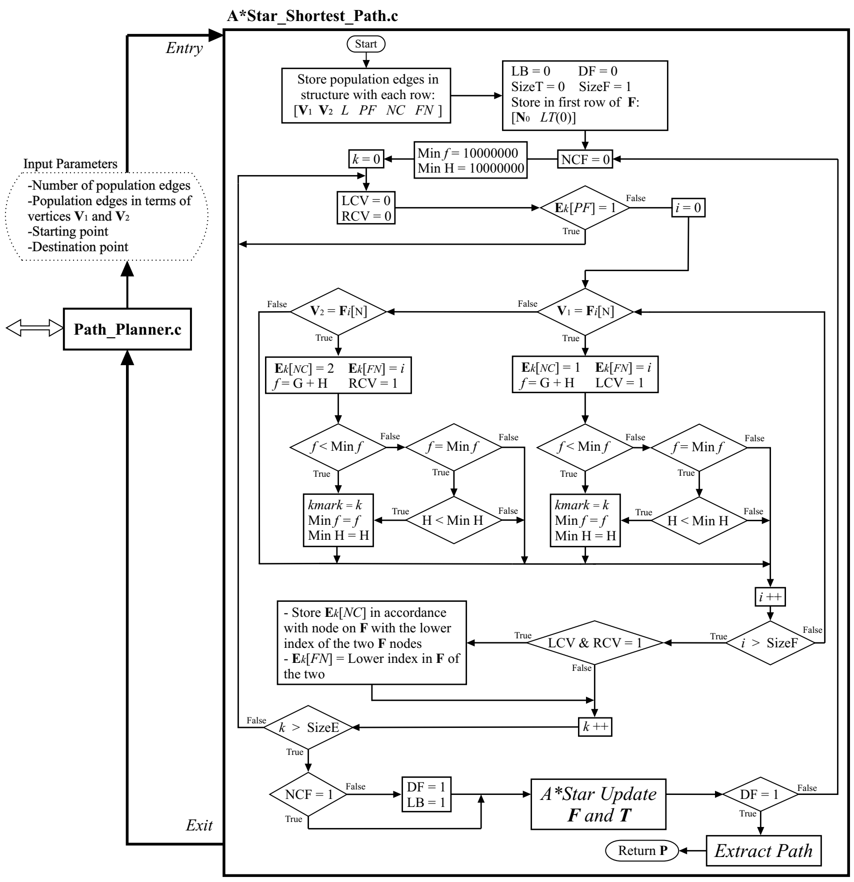
Figure 3.36: Equivalent Discrete *Dijkstra* steps

As mentioned at [19], another important characteristic to note is the fact that a *heuristic* which overestimates H will not always yield the shortest path. The Euclidean *heuristic* is therefore an effective choice in this implementation because it will never overestimate H, but only underestimate it and in some circumstances be a very accurate estimation. It therefore has a worst case bound with the same time complexity as *Dijkstra*, $O(n^2)$, with an optimal implementation of *Dijkstra*, but is generally much faster. As an added bonus it favours straight paths which is in accordance with the preferences for motion of the UTV.

Implementation

The implementation of this algorithm was done according to the flowchart shown in Figure 3.37. As seen the implementation is similar to the *Dijkstra* implementation with the exception of the added *heuristic* function. The routines contained in the *A*Star Update **F** and **T*** block are shown in Figure 3.38 while the routines of the *Extract Path* block are identical to the one used with the *Dijkstra* implementation.

An additional note should be made about the scenario where two candidates in the *open set* have identical *f scores*. To accommodate this scenario, which is most certainly a possibility, a certain *tie-breaker* function has to be implemented. Similarly to the choice in *Heuristic* this *tie-breaker* function is also application specific and can be chosen in many ways. The reader is referred to [17] for some suggestions. As shown in the flow diagram of Figure 3.37, ties are simply broken in this implementation by choosing the edge with the lowest heuristic score, H.



Legend

F - Frontier Matrix representing nodes on closed set

T - Tree rooted at origin containing edges in closed set

LT - Total length to origin along edges of T

NC - New candidate flag
- 0 - Population edge is not a candidate
- 1 - Population edge is a candidate with vertex in left column corresponding to node in **F**
- 2 - Population edge is a candidate with vertex in right column corresponding to node in **F**

FN - Index in **F** which corresponds to a vertex of candidate population edge $\mathbf{E}_k$

PF - Flag indicating whether population edge $\mathbf{E}_k$ is in closed set

L - Length of population edge $\mathbf{E}_k$

LB - Linkbox flag indicating no connection between start and destination. To accomodate degenerate cases with Voronoi population

DF - Flag indicating whether destination has been reached

$\mathbf{E}_k$ - Population edge, index, *k,* with vertices $\mathbf{V}_1$ and $\mathbf{V}_2$

LCV - Flag indicating left column vertex of $\mathbf{E}_k$, $\mathbf{V}_1$, was found which corresponds to a node in **F**

RCV - Flag indicating left column vertex of $\mathbf{E}_k$, $\mathbf{V}_1$, was found which corresponds to a node in **F**

SizeT - Integer storing current size in rows of the tree, **T**

SizeF - Integer storing current size in rows of the frontier matrix, **F**

SizeE - Integer storing size in rows population edges, **E**

NCF - Flag indicating whether new candidate was found

F_i - Current node in **F**

P - Calculated Path Matrix

f - Cost to new node >>> $f = G + H$
$G = \mathbf{F}_i[LT] + \mathbf{E}_k[L]$
H = Heuristic function. In this case the Euclidian distance to the destination from $\mathbf{E}_k[V_1]$ or $\mathbf{E}_k[V_2]$ depending on which node of $\mathbf{E}_k$ corresponds $\mathbf{F}_i$

N - Node stored in $\mathbf{F}_i$

kmark - Index of candidate $\mathbf{E}_k$ which yields the lowest f score

$\mathbf{N}_0$ - Node of the origin

Figure 3.37: Main Flowchart of implemented *A*Star* Algorithm

In the case of ties, *population* edges are therefore favoured which yield ending nodes closer to the destination. Should the heuristic scores also match, the first candidate in the *open set*, which was marked, is simply chosen. Ultimately the choice in *tie-breaker* simply has the ability to speed up the search if it is wisely chosen.

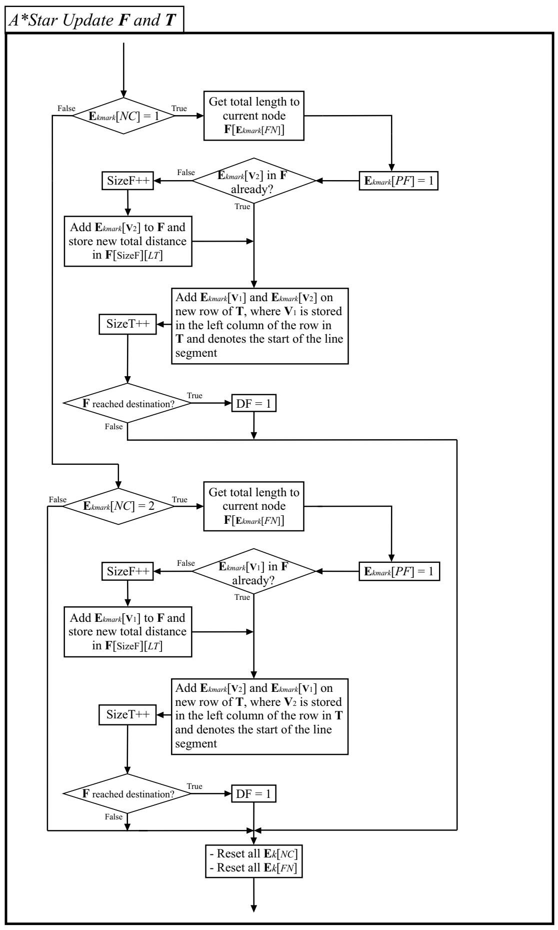

A*Star Update F and T
False
$\mathbf{E}_{kmark}[NC] = 1$
True
Get total length to current node $\mathbf{F}[\mathbf{E}_{kmark}[FN]]$
$\mathbf{E}_{kmark}[PF] = 1$
$\mathbf{E}_{kmark}[\mathbf{V}_2]$ in **F** already?
False
True
SizeF++
Add $\mathbf{E}_{kmark}[\mathbf{V}_2]$ to **F** and store new total distance in $\mathbf{F}[\text{SizeF}][LT]$
Add $\mathbf{E}_{kmark}[\mathbf{V}_1]$ and $\mathbf{E}_{kmark}[\mathbf{V}_2]$ on new row of **T**, where $\mathbf{V}_1$ is stored in the left column of the row in **T** and denotes the start of the line segment
SizeT++
F reached destination?
True
False
DF = 1
False
$\mathbf{E}_{kmark}[NC] = 2$
True
Get total length to current node $\mathbf{F}[\mathbf{E}_{kmark}[FN]]$
$\mathbf{E}_{kmark}[PF] = 1$
$\mathbf{E}_{kmark}[\mathbf{V}_1]$ in **F** already?
False
True
SizeF++
Add $\mathbf{E}_{kmark}[\mathbf{V}_1]$ to **F** and store new total distance in $\mathbf{F}[\text{SizeF}][LT]$
Add $\mathbf{E}_{kmark}[\mathbf{V}_2]$ and $\mathbf{E}_{kmark}[\mathbf{V}_1]$ on new row of **T**, where $\mathbf{V}_2$ is stored in the left column of the row in **T** and denotes the start of the line segment
SizeT++
F reached destination?
True
False
DF = 1
- Reset all $\mathbf{E}_k[NC]$
- Reset all $\mathbf{E}_k[FN]$

Figure 3.38: Flowchart of *A*Star Update **F** and **T*** block

Results

In order to illustrate the performance differences between the *A*Star* algorithm and the *Dijkstra* algorithm, simulation results are once again presented. Consider the results of the algorithm on a *Visibility population* with square obstacles, shown in Figure 3.39. As seen the *A*Star* algorithm completely out performs *Dijkstra* in this example. A bare minimum of the *population* edges are added to the *closed set* before the same optimal path is found and this can be attributed to the fact that a path approximating a perfect straight line is available in this *population* set. Once again the tendency of the Euclidean *heuristic* to approximate straight paths is demonstrated and the immense performance improvements in comparison with *Dijkstra*, when this Euclidean *heuristic* closely resembles the actual cost, H, is clearly visible. Since a *Visibility* graph generally has n^2 edges, and the inner loop with regards to the *closed set*, **F**, in the implementation shown in the previous section, only executes a small constant amount of times in this example (since **T** grows minimally), the complexity of the implementation is suddenly reduced from above $O(n^3)$ (with the non-optimal *Dijkstra* implementation) to possibly $O(n^2)$. Since this is a non-optimal implementation and the optimal implementation of *Dijkstra* has a time complexity, $O(n^2)$, the complexity could possibly be lowered even more toward $O(n)$!

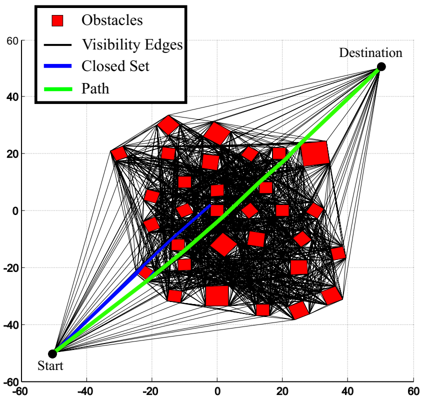


Figure 3.39: *A*Star* Search on Square Obstacle *Visibility Population*

To see how an Euclidean *A*Star* search becomes less effective and tends toward the slower performances of *Dijkstra*, when straight paths are not available, refer to Figure 3.40. In this figure it is clearly visible that there is only a marginal performance increase from the simulation on the same obstacle set presented during the discussions on *Dijkstra*. It is visible how the tendency of a *Voronoi population* to represent paths which are not straight degrades the performance of the Euclidean *A*Star* search. However, when considering the fact that *Voronoi populations* have a much lower complexity than *Visibility* graphs, in the order of n edges, as proposed in [9], the overall performance still comes close to a *Visibility* graph and *A*Star* combination, and might in some cases be even better when *A*Star* performance is degraded on a *Visibility population* (discussed in the next paragraph).

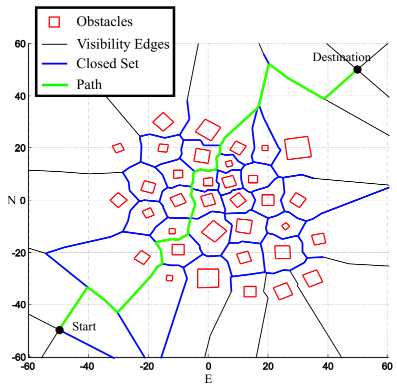


Figure 3.40: *A*Star* Search on Square Obstacle *Voronoi Population*

Two simulation results with a convex obstacle set are presented in Figures 3.41 and 3.42. These figures effectively illustrate the drastic performance degradation a mere change in start and destination around the same obstacle set can cause. Although a clear line of site is not available in Figure 3.41 it is seen how *A*Star* still performs relatively well with a minimal growth in, **T**, compared to the performance of *Dijkstra* on the same *population*, shown in Figure 3.31. However, when the start and destination is moved in such a way that the shortest path is forced into being a large detour, shown in Figure 3.42, the Euclidean *heuristic* drastically underestimates, H, and the algorithm degenerates toward the *Dijkstra* algorithm where, H, is simply zero. This is clearly seen in the amount of edges which are added to the closed set in Figure 3.42 which more closely resemble a *Dijkstra* expansion.

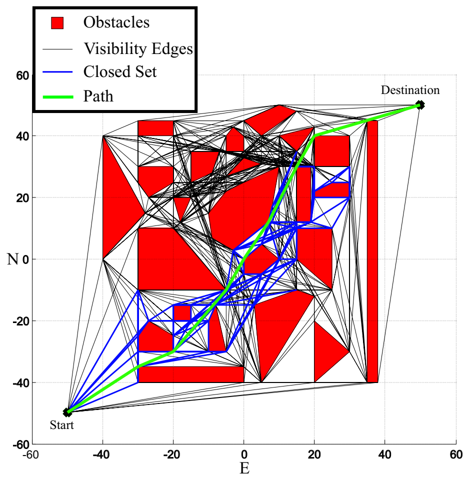


Figure 3.41: *A*Star* Search on Convex Obstacle *Visibility Population*

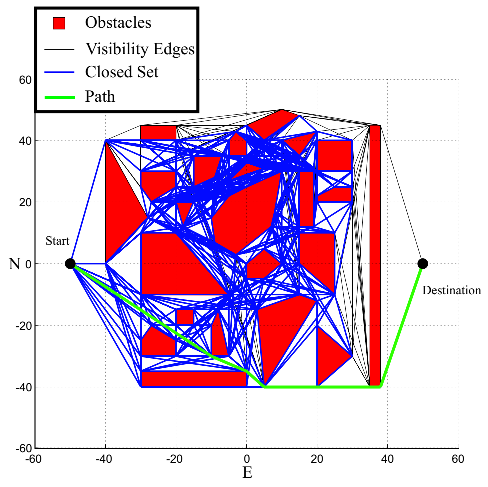


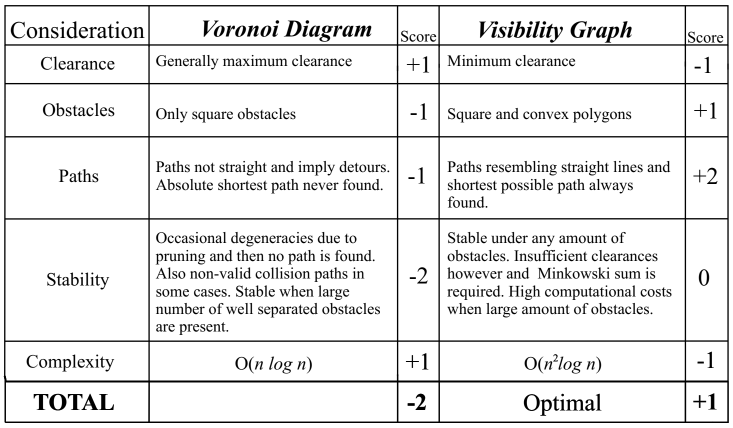
Figure 3.42: *A*Star* Search on *Visibility Population* with no line of sight

To summarize, the *A*Star* algorithm with an Euclidean *heuristic* greatly improves performance in an application where the destination is known. In the worst case it can only be as computationally expensive as *Dijkstra* and never worse and in most cases better. It still accurately determines the shortest possible path since the cost, H, is not overestimated but at most accurately approximated and generally underestimated. *A*Star* therefore proves to be the more attractive algorithm in this project. *Dijkstra* should however be given credit for its efficiency in other applications where one specific destination is not known (the type of application this particular *A*Star* implementation will not be able to accommodate).

3.4 Performance Summary

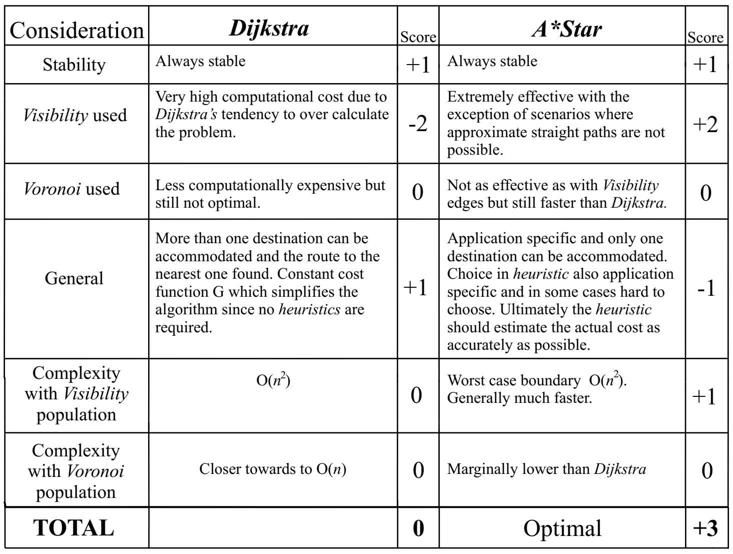
In order to give a summary of the respective algorithms investigated, tables are presented as shown in Figures 3.43 and 3.44. These tables are based on an intuitive scoring system to give a rough indication of which algorithm combination is optimal in this path planning application.

It should be noted that all time complexities listed are not those of the implemented algorithms but rather the complexities of these algorithms assuming optimal implementations, as discussed previously. This is done to ensure one algorithm is not unfairly disadvantaged. From these tables it is visible that as far as computational cost goes a *Voronoi* and *A*Star* combination is optimal. However, *Voronoi* yields non-optimal, non-straight paths as a price for achieving maximum clearance from obstacles. This alone could possibly be overlooked to place *Voronoi* higher on the scoreboard since *Visibility* graphs have the reciprocal downside of minimum clearance from obstacles in order to yield optimally straight and short paths, and significant increases in computational cost.



Consideration	***Voronoi Diagram***	Score	***Visibility Graph***	Score
Clearance	Generally maximum clearance	+1	Minimum clearance	-1
Obstacles	Only square obstacles	-1	Square and convex polygons	+1
Paths	Paths not straight and imply detours. Absolute shortest path never found.	-1	Paths resembling straight lines and shortest possible path always found.	+2
Stability	Occasional degeneracies due to pruning and then no path is found. Also non-valid collision paths in some cases. Stable when large number of well separated obstacles are present.	-2	Stable under any amount of obstacles. Insufficient clearances however and Minkowski sum is required. High computational costs when large amount of obstacles.	0
Complexity	O($n \log n$)	+1	O($n^2 \log n$)	-1
TOTAL		**-2**	Optimal	**+1**

Figure 3.43: Performance summary of *population* algorithms



Consideration	***Dijkstra***	Score	***A*Star***	Score
Stability	Always stable	+1	Always stable	+1
Visibility used	Very high computational cost due to *Dijkstra's* tendency to over calculate the problem.	-2	Extremely effective with the exception of scenarios where approximate straight paths are not possible.	+2
Voronoi used	Less computationally expensive but still not optimal.	0	Not as effective as with *Visibility* edges but still faster than *Dijkstra.*	0
General	More than one destination can be accommodated and the route to the nearest one found. Constant cost function G which simplifies the algorithm since no *heuristics* are required.	+1	Application specific and only one destination can be accommodated. Choice in *heuristic* also application specific and in some cases hard to choose. Ultimately the *heuristic* should estimate the actual cost as accurately as possible.	-1
Complexity with *Visibility* population	O(n^2)	0	Worst case boundary O(n^2). Generally much faster.	+1
Complexity with *Voronoi* population	Closer towards to O(n)	0	Marginally lower than *Dijkstra*	0
TOTAL		**0**	Optimal	**+3**

Figure 3.44: Performance summary of *shortest path* algorithms

Voronoi however has an added deficiency which scores it much lower than *Visibility* graphs. This is the possibility of it not finding a path or occasionally yielding paths which would imply collisions. From a stability point of view *Visibility* graphs are therefore the better choice and when considering the generally low computational cost of an *A*Star* search on a *Visibility* population, the increased $O(n^2 \log n)$ *population* complexity of the *Visibility* graph is a small price to pay. As an added bonus a method is available for ensuring improved obstacle clearance when a *Visibility* graph is used and will be discussed briefly in Chapter 7. The optimal combination, of the implemented algorithms in this project, is therefore a *Visibility* and *A*Star* combination, with the only requirement being, an algorithm to ensure sufficient clearances.

3.5 Summary

This chapter featured an in depth discussion on the specific path planning algorithms used in this project. It was shown that a *population* method is first used to construct a *roadmap* representing several feasible paths through the obstacle environment. Two *shortest path* algorithms were then introduced which are used on these *populations* to find the optimal path consisting of a combination of these line segments in the obstacle-free space. The basic theory behind these algorithms were presented, followed by a discussion on the implementations thereof and finally results were shown. A comparison was then given between the respective algorithms and of the algorithms implemented in this project it was decided that the optimal combination is a *Visibility population* used in conjunction with an *A*Star* search. For an interactive demonstration of these algorithms please refer to Appendix D. This chapter is therefore concluded and a closed path planning module is now available for use by the UTV's controllers. This is indeed the topic of the next chapter.

Chapter 4

Control and Simulator

This chapter features a discussion on the controllers implemented to steer the UTV along a calculated path consisting of straight line segments. All state measurements used by these controllers are obtained from a state estimator on the OBC, which is discussed in Chapter 5, with the exception of the wheel speed measurements and current measurements for the drive system controllers which are obtained from the encoders and current transducer respectively, as discussed in Chapter 2. The simulator for the entire system implemented in *Matlab* is also presented toward the end of this chapter.

4.1 Overview

The UTV's controller hierarchy consists of several levels, each level serving a specific purpose through utilization of controllers on lower levels. Before attempting the design of these controllers a clear understanding of what is required from the UTV, in order to achieve its purpose, is necessary. The UTV is required to perform the following tasks.

- Be manually operatable from a ground station through RF communication - *Appendix C*.
- Receive the autopilot initialization command and start all algorithms and control sequences - *Appendix C*.
- Calculate a path consisting of straight line segments through an obstacle ridden environment given a point of departure and destination - *Chapter 3*.
- Commence navigation of the calculated path. Throughout navigation, stop momentarily at the end of each line segment and turn on the cur-

rent position to achieve the heading of the next line segment - *Chapter 4*.

- Repeat this until the destination is reached, after which coming to a halt and going into standby, awaiting further instructions - *Chapter 4* and *Appendix C*.
- Be able to receive coordinates of additional obstacles during tracking of a path. In such a scenario come to a halt, recalculate the new best path and then resume tracking until the destination is reached - *Chapter 4* and *Appendix C*.
- While doing tasks log telemetry and provide a real-time stream of telemetry data to a ground station through RF communication - *Appendix C*.

To achieve these goals the UTV requires the design of several controllers which range from the lowest level of wheel speed control to the highest level of path planning and guidance control. The approach consists of designing a feedback controller on each level and then moving up to the next level with a successive loop closure, thereby obtaining a new higher level reference input to the system. At the level where only a specific heading angle and line segment length is required as input, decision making responsibility is given to a state-machine which activates and deactivates certain controllers in accordance with the current activity on the path-planning module and the current UTV state vector.

The chapter will start off at the lowest controller level which is the angular wheel velocity controller of the drive systems and then progress through higher levels up to the state-machine.

4.2 Drive System Controller

The lowest level of control is the drive system feedback control which serves the sole purpose of achieving a reference wheel speed under varying loads. The plant characteristics of this system is presented in Appendix B and a controller is designed with the assumption that both DC motors are identical.

The decision is made to implement full state feedback control to achieve a desired response since all states are available for measurement and an estimator is not required. It should be noted that this is the only control loop which does not make use of measurements obtained from the state-estimator, discussed in Chapter 5. This is done since the drive systems' control loops run on the PIC microcontrollers, separately from the OBC, as discussed in Chapter 2 and provide a proverbial black box for higher level controllers to interface with.

Integral control is further added to the full-state feedback configuration to provide disturbance rejection and eliminate steady-state errors on the response. Since this plant has a bandwidth of 2.83 *Hz* which is only a factor 8.8 times smaller than the sample rate of 25 *Hz*, as discussed in Chapter 2, the decision is further made to follow a direct discrete design approach. This is done since a factor 30 is recommended for design by emulation as mentioned in [4].

Figure 4.1 shows a block diagram of the controller configuration. The controller is designed in the discrete domain and then used to control the continuous plant through use of a zero-order hold D/A converter as can be seen. As mentioned in [1] this configuration of integral control, with the feed forward, $\overline{N}$, introduces a new zero to the system which can be made to cancel the added closed-loop pole of the integrator state and thereby cancels the excitation of this pole from the reference command input. The zero is introduced with a choice of $\overline{N}$ where, $\overline{N} = \frac{K_I}{1-z_I}$, and z_I is the closed loop integrator pole.

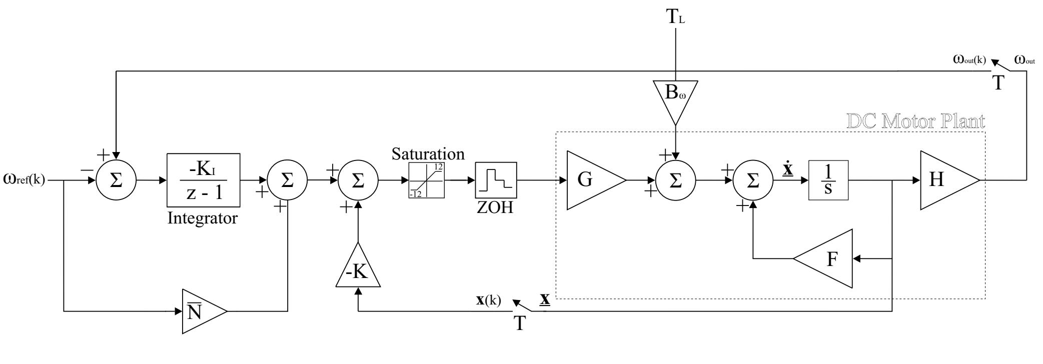


Figure 4.1: Integral Control with full-state feedback and added zero [1]

It should also be noted that the control input is limited to between -12 and 12 *V* by a saturation block since the maximum and minimum inputs to the system are 12 *V* and -12 *V* respectively from the Sealed Lead Acid Batteries introduced in Chapter 2. The controller therefore has to be designed with a balance between use of control and speed of the step response as main priority. For this reason *Matlab*'s *DLQR* function is used to design the feedback gain **K** for full state feedback.

The Linear Quadratic Regular (LQR) method is used since it serves as a way of finding an optimal balance between response and control usage. It does this by minimizing a cost function which consists of the weighted squares of the states of the system as well as the weighted squares of the inputs to the system. More information about this method can be found in [1] and the cost function is simply listed here for convenience.

$$J = \frac{1}{2}\sum_{k=0}^{N}[\ \mathbf{x}^T(k)\mathbf{Q}_1\mathbf{x}(k) + \mathbf{u}^T(k)\mathbf{Q}_2\mathbf{u}(k)\ ] \tag{4.2.1}$$

Before using this method the discrete state-space matrices of the system plant, from Appendix B, are augmented to include the integral state. This then yields the following augmented plant model

$$\begin{bmatrix} x_I(k+1) \\ \mathbf{x}(k+1) \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{H} \\ 0 & \mathbf{\Phi} \end{bmatrix} \begin{bmatrix} x_I(k) \\ \mathbf{x}(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{\Gamma} \end{bmatrix} u(k) - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} r(k) \tag{4.2.2}$$

and the augmented control law for state feedback is,

$$u(k) = -\begin{bmatrix} K_I & \mathbf{K} \end{bmatrix} \begin{bmatrix} x_I(k) \\ \mathbf{x}(k) \end{bmatrix} + \overline{N}r(k) \tag{4.2.3}$$

Substitution of Equation 4.2.3 into 4.2.2 then yields the state-space matrices for the closed loop drive system with $\omega_{ref}(k)$ as input and $\omega_{out}(k)$ as output as seen in Figure 4.1.

$$\begin{bmatrix} x_I(k+1) \\ \mathbf{x}(k+1) \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{H} \\ -\mathbf{\Gamma}K_I & \mathbf{\Phi} - \mathbf{\Gamma}\mathbf{K} \end{bmatrix} \begin{bmatrix} x_I(k) \\ \mathbf{x}(k) \end{bmatrix} + \begin{bmatrix} -1 \\ \overline{N}\mathbf{\Gamma} \end{bmatrix} r(k)$$

$$y(k) = \begin{bmatrix} 0 & \mathbf{H} \end{bmatrix} \begin{bmatrix} x_I(k) \\ \mathbf{x}(k) \end{bmatrix} \tag{4.2.4}$$

The goal is now to choose weighting matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$ in Equation 4.2.1 in such a way that a fast response is obtained while not exceeding the saturation limits, in Figure 4.1, on the actuator input. Through a process of trial and error in *Matlab* with the *DLQR* function the following matrices were found which yielded a feedback gain, **K**, which resulted in a satisfactory closed loop response while not exceeding actuator limits.

$$\mathbf{Q}_1 = \begin{bmatrix} 236 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \qquad \mathbf{Q}_2 = \begin{bmatrix} 100 \end{bmatrix} \tag{4.2.5}$$

The maximum rotational velocity of the wheels was recorded as 8 rad/s and the system must therefore be capable of achieving this step without saturating the controls. Figure 4.2 shows how a relatively fast step response is obtained while keeping the control usage below the saturation limit of 12 V. An angular velocity disturbance step, due to a load torque, is also applied to the system after 4 seconds and the disturbance rejection effect of the integral control on the steady state response can be seen.

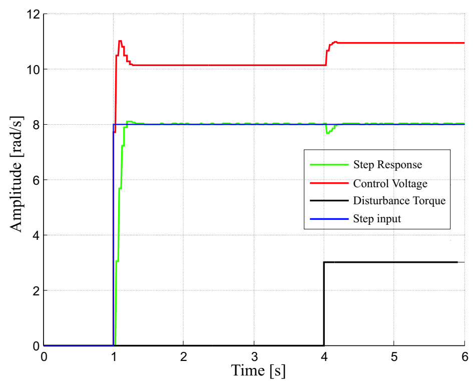


Figure 4.2: Closed Loop Drive System's Response

As confirmation of accuracy the response of the closed loop discrete state-space model of Equation 4.2.4 was compared to the response from the Simulink model shown in Figure 4.2. These responses matched and a transfer function, $G_{Drive}(\mathbf{z})$, is found for the closed loop drive system with the following poles,

$$\begin{aligned} z_1 &= 0.4231 + 0.2366i \\ z_2 &= 0.4231 - 0.2366i \\ z_3 &= 14.44 \times 10^{-12} \end{aligned}$$

and a system bandwidth of 3.15 Hz, as previously mentioned in Chapter 2.

4.3 Yaw Rate Controller

The next higher level of control is the yaw rate controller. The yaw rate controller is required to receive a certain yaw rate reference command and then force the UTV to achieve this yaw rate, with zero steady state error, by simultaneously sending reference angular wheel velocities to the right and left drive systems of the UTV which have closed loop characteristics as presented in the previous section.

Before continuing with the design of the yaw rate controller a discussion on the UTV's motion in a two-dimensional North-East (NE) reference frame is required. In order to fully define motion of the UTV in this plane the Northern, N, and Eastern, E, coordinates are required as well as the heading angle, ψ, of the UTV. In Figure 4.3 the differential equations of motion for the UTV are shown. From inspection it is visible that the inputs to the system are the forward velocity, **V**, and the yaw rate, **r**.

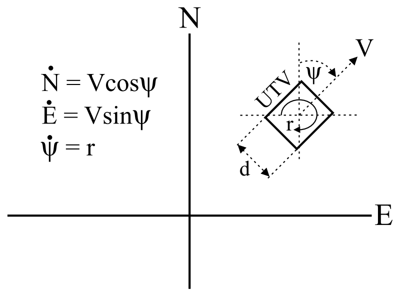


Figure 4.3: Motion of the UTV in 2 Dimensional Inertial Reference Frame

Since the UTV has no steering angle and simply steers using a skid steer principle the side-slip angle is regarded as zero and the velocity vector, **V**, is in line with the heading angle of the UTV. The task at hand is to find the relationship between the **V** and **r** reference inputs and the wheel speed references so as to be able to design a yaw rate controller, with a combination of the closed loop drive systems in the previous section as plant. The difficulty comes in when one considers the fact that wheel speeds are dependent on both the yaw rate command as well as the forward velocity command and both **V** and **r** affect both left and right wheel speeds of the UTV. The yaw rate, **r**, can be thought of as being proportional to a differential wheel speed, which is superimposed on a common mode wheel speed, **V**. As will be shown shortly, the system can be decoupled through definition of two virtual inputs and individual controllers can then be designed for **V** and **r** respectively.

However, when one considers that the UTV in this project is only required to track a straight line path at a set forward velocity the **V** input can be regarded as a constant over the durations of the yaw rate controller being active. Forward velocity is therefore operated in an open loop configuration since utmost accuracy is not required in the velocity at which each line segment is tracked, and the system reduces to a yaw rate controller superimposed on open loop forward velocity reference commands. It should be noted that a simple secondary controller is activated after each line segment, which does not make use of the yaw rate controller, to turn the UTV on its current location until the new line segment's heading is achieved. In hindsight it was realized that this secondary controller is redundant and one heading controller being used throughout will have sufficed. Since the state machines, presented later in this chapter, were already implemented at this time a decision was made to leave the secondary controller in place but it should be noted that it is not required. More about this secondary controller and the open loop operation of **V** will follow later in this chapter.

In order to aid with finding expressions for the left and right wheel speeds in terms of **V** and **r** the following virtual inputs are defined,

$$\begin{aligned} V_{cm} &= \frac{V_L + V_R}{2} \\ V_{dm} &= \frac{V_L - V_R}{2} \end{aligned} \tag{4.3.1}$$

where V_{cm} is the common mode forward velocity, V_{dm} is the differential mode velocity, V_L is the forward velocity of the left side of the UTV and V_R is the forward velocity of the right side of the UTV. From Equations 4.3.1 the left and right wheel speeds can now be expressed in terms of these two virtual inputs,

$$\begin{aligned} \omega_L &= \omega_{cm} + \omega_{dm} \\ \omega_R &= \omega_{cm} - \omega_{dm} \end{aligned} \tag{4.3.2}$$

where the common mode angular wheel velocity is denoted by ω_{cm} and the differential mode angular wheel velocity by ω_{dm}. **V** and **r** can be defined in terms of these virtual inputs and vice versa. The yaw rate of the UTV can be expressed as,

$$\mathbf{r} = \frac{V_L - V_R}{d} = \frac{0.2\omega_{dm}}{d} \tag{4.3.3}$$

where d is the width of the UTV's chassis and it is taken into account that the wheels have a radius of 0.1 m. It therefore follows that,

$$\omega_{dm} = \frac{\mathbf{r}d}{0.2} \qquad\qquad \omega_{cm} = 10V_{cm} = 10\mathbf{V} \tag{4.3.4}$$

The block diagram in Figure 4.4 shows a representation of these equations. The velocity input is a constant as previously mentioned and the constant tracking velocity for each straight line segment of the path is chosen as 4.5 rad/s or 0.45 m/s.

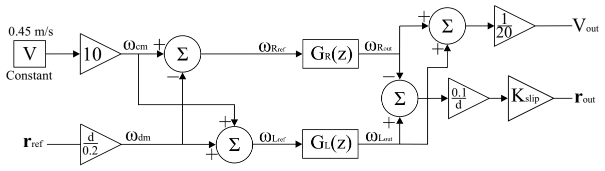


Figure 4.4: Block Diagram of Yaw Rate and Forward Velocity Plant

Also seen in this block diagram is a slip gain, K_{slip}, which is used to calibrate the system model by commanding certain yaw rates and recording actual measured yaw rates of the UTV. This gain serves as a means of accounting for the fact that the UTV does not operate with no slip of the wheels in reality and a value of $K_{slip} = 0.53$ was found through calibration. The calibration process is shown in Appendix B. $G_R(z)$ represents the transfer function of the right closed loop drive system while $G_L(z)$ represents the left drive system.

A transfer function from the reference yaw rate to the yaw rate output is found for this plant. From Figure 4.4 the following equation holds.

$$\mathbf{r}_{out} = \frac{0.1}{d}[10\mathbf{V}G_L(z) + \frac{\mathbf{r}_{ref}d}{0.2}G_L(z) - 10\mathbf{V}G_R(z) + \frac{\mathbf{r}_{ref}d}{0.2}G_R(z)]K_{slip} \tag{4.3.5}$$

With the previously mentioned assumption that both DC motors are identical and that the differences in their responses are negligible, $G_L(z) = G_R(z) = G_{LR}(z) = G_{Drive}(z)$, and Equation 4.3.5 reduces to,

$$G_r(z) = \frac{\mathbf{r}_{out}}{\mathbf{r}_{ref}} = K_{slip}G_{LR}(z) \tag{4.3.6}$$

The yaw rate controller is designed with Equation 4.3.6 representing the plant dynamics. Intuitively it can be seen in Equation 4.3.5 that differences in dynamics between $G_L(z)$ and $G_R(z)$ are amplified by an increase in reference forward velocity, V. Since slight inaccuracies exist in the plant models and due to the assumption that both closed loop drive systems are identical in dynamic response, which is not entirely true, the decision is made to design a proportional integral (PI) controller. The integral part of the controller serves as way of rejecting steady-state disturbances and also negates the effects of inaccuracies in the plant model.

Through use of *Matlab*'s root locus design tools on the plant in Equation 4.3.6 a PI controller is found. A PI compensator is defined in *Matlab*'s *SISOTOOL* and the proportional gain then varied along the root locus to yield a step response with minimum settling time without producing any overshoot. In reality the UTV has a maximum yaw rate which corresponds to the wheels on one side of the UTV rotating at maximum angular velocity and the wheels on the other side rotating at maximum angular velocity in the opposite direction. This maximum yaw rate was found during practical calibration as 1.06 rad/s. To protect against integrator wind-up in the yaw rate controller it must be ensured that yaw rate references above this saturation limit does not occur. Prolonged disturbances such as heavy torque loads, which prevent the UTV's wheels from turning, can however still cause integrator wind-up and it is assumed that such loads are not present during test runs. As will be seen in the next section, the heading controller which is responsible for yaw rate reference commands is merely a proportional controller and saturation limits can therefore be imposed there without any integrator wind-up occurring. The closed loop block diagram of the yaw rate controller is shown in Figure 4.5. The saturation limits on the yaw rate output is shown. Also present is a yaw rate disturbance input, w, which serves as elementary way of simulating possible steady state disturbances due to plant model inaccuracies and torque loads due to the terrain on which the UTV is operated.

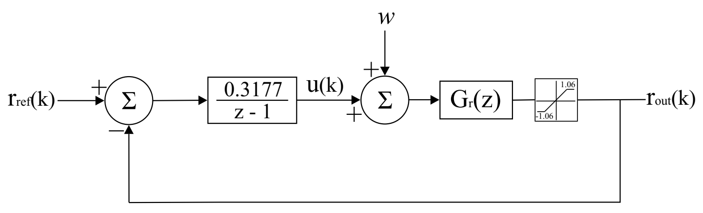


Figure 4.5: Closed Loop Yaw Rate Controller

This closed loop system has the step response shown in Figure 4.6.

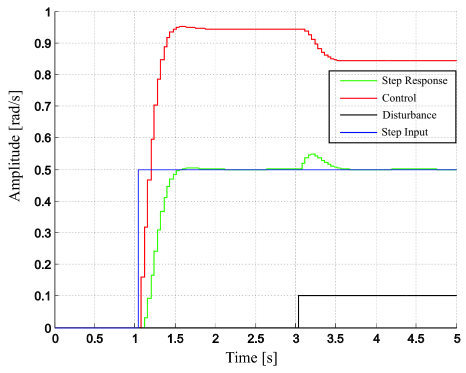


Figure 4.6: Step Response of Closed Loop Yaw Rate System

Also shown is the rejection of a step disturbance at 3 seconds by the integral term of the compensator and as seen the overshoot is minimal. The closed loop yaw rate input to yaw rate output transfer function, $G_{rcl}(\mathbf{z})$, yielded a bandwidth of 1.25 Hz.

4.4 Heading Controller

The next level of control is the heading controller. Similarly to the approach in [4] a yaw rate command is generated from the heading error signal. The task at hand is therefore to design a sufficient controller in another closed loop configuration with the yaw rate controller from the previous section as plant while just adding the natural integrator to yield heading angle from yaw rate.

Ideally a compensator integrator term is also optimal in this compensator to reject disturbances such as biases on the yaw rate gyro. However, if yaw rate gyro biases can be ignored, then the natural integrator is sufficient to allow tracking of heading commands with zero steady state error. Since the rate gyros on the UTV are calibrated (set to zero while the UTV is stationary with appropriate measurement gains and offsets) before each test run, the possibility of biases due to temperature differences are reduced. As will be seen in chapters to follow an Extended Kalman Filter is used in providing best estimates of all the UTV's states, heading angle included. Since this implies a heading measurement by weighing both the direct measurement from a magnetometer as well as the integral of the yaw rate gyro the possibility of steady state heading errors is further decreased. During practical demonstrations it was found that proportional control alone is sufficient in allowing the tracking of heading commands with negligible steady state heading errors.

A further advantage of using only proportional control is the feasibility of implementing saturation limits on the yaw rate command, which is obtained from the heading controller, and thus decreasing the possibility of integrator wind-up in the yaw rate controller. As mentioned in the previous section the UTV makes use of two heading controllers which are activated respectively depending on whether the UTV is at the end of a line segment or busy tracking a line segment. Figure 4.7 shows a block diagram of the closed loop heading controller which is active while the UTV is busy tracking a straight line segment.

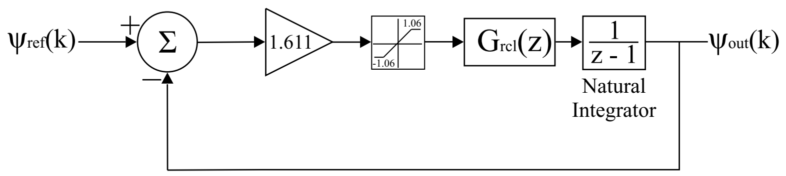


Figure 4.7: Block Diagram of closed loop Primary Heading Controller

The other heading controller is activated when the UTV has reached the end of a line segment and is required to turn on its current location to find the heading of the next line segment. As mentioned, this controller is unnecessary but was left in place due to the state-machine, to be discussed later in this chapter, already having been implemented at the time of realizing the redundance of a secondary heading controller. The alternative controller runs separately from the yaw rate controller in an open loop configuration. The controller simply checks the sign of the yaw rate reference obtained from the heading controller in Figure 4.7 and then demands $-3\ rad/s$ from the one closed loop drive system and $3\ rad/s$ from the opposite closed loop drive system accordingly. This command is issued until a heading measurement within a certain tolerance of the heading angle of the next line segment is obtained, after which control is given back to the guidance controller, which makes use of the heading controller in Figure 4.7, and tracking of the next line segment commences.

Figure 4.8 shows the step response of the closed loop system in Figure 4.7. The heading compensator proportional gain, K_H, was once again found using

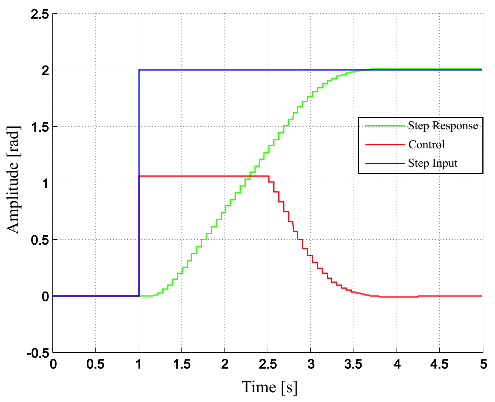


Figure 4.8: Step Response of Primary Heading Controller

Matlab's *SISOTOOL* to plot the root locus of the open loop system and then varying the gain along this root locus until the fastest step response with no overshoot was obtained. No overshoot is preferable since overshoot would imply the UTV displaying oscillatory motions when a step to a new heading occurs, before settling on the new heading in the steady state. As seen the the saturation limits on the control (the yaw rate command) simply enters the UTV into a constant maximum yaw rate for the turn, when large heading step commands are issued, and since only a proportional compensator is used no integral wind-up occurs, provided large prolonged torque loads are not present.

4.5 Guidance Controller

The guidance controller is based on the work done by [4]. The approach is to generate a heading command from a cross track error signal, which is simply the lateral distance of deviation from a straight line path. The calculation of this cross track error signal is done through use of the position measurements obtained from the state estimator discussed in Chapter 5. This is done by centering the coordinate system at the start of the current straight line segment, which the UTV is required to track, and then rotating it until the new OX-axis lies along the heading of the current straight line path segment. Rotation is done by setting the beginning of the current line segment as the origin of the coordinate system and then multiplying the new position vector with the following transformation matrix which follows from simple geometric inspection,

$$\textbf{Transformation} = \begin{bmatrix} cos\psi_{ps} & sin\psi_{ps} \\ -sin\psi_{ps} & cos\psi_{ps} \end{bmatrix} \tag{4.5.1}$$

where ψ_{ps} is the heading angle of the current path segment to be tracked. The cross track error is then simply the OY-axis coordinate of the transformed UTV location. The transformed OX-axis directly represents the distance traveled by the UTV along its current path segment. Checking for when the end of a line segment is reached is therefore simply done by checking whether the current transformed OX-axis measurement is equal to or exceeds the length of the path segment obtained from the path planner. When this occurs a flag is set and the state-machine, to be discussed in the next section, activates a new controller.

Through use of a closed loop feedback configuration, with the heading controller from the previous section as plant and a proportional compensator, K_{track} , and continuously commanding a zero reference, a guidance controller is implemented. A proportional gain, $K_{track} = 1$, was found to yield good results. This guidance controller forces the UTV to remain on its current straight line path by continuously adjusting the heading reference in an attempt to achieve the zero track error reference input. A block diagram of the closed loop guidance system can be found in [4]. Also mentioned in [4] is the effect of rate gyroscope biases and how the addition of an integral term to the heading controller alone can nullify these effects. Since satisfactory results were obtained during test runs, without an integral term in the heading controller, and since an integral term is undesirable due to previously mentioned integrator wind-up possibilities, the decision is made to make use of only a proportional heading compensator in this project. The rate gyroscopes are also calibrated before each test run, as previously mentioned, which further reduces the possibility of rate gyroscope biases.

The guidance controller therefore completes the final successive loop closure of the controllers and represents the highest level of control which encapsulates the dynamics of all lower level controllers. This controller makes use of the following inputs obtained from the path planning module still to be discussed,

- P_d - Departure coordinates of current path line segment
- P_a - Destination coordinates of current path line segment
- ψ_{ps} - Heading angle of current path line segment
- L_{ps} - Length of current path line segment

and the following measurements obtained from the state estimator,

- N - Northern location measurement
- E - Eastern location measurement
- ψ - heading angle measurement

and only operates during tracking of a straight line path segment and not during transitions from one line segment to the next. A final note should be made about the sample rate of these controllers. Although the PC104/CAN controller, has an update rate of 50 *Hz*, all UTV controllers can only have an update rate of 25 *Hz* due to the limitations presented in Chapter 2. All software routines on the OBC therefore still run at 50 *Hz* to accommodate the PC104/CAN Controller protocol, as in [15], but the UTV control and the state estimator are only updated in multiples of 2 of 20 *ms* (50 *Hz*) and as a result the required lower sample rate of 25 *Hz* is obtained.

4.6 Controller Scheduling

In order to accomplish the requirements stated in Section 4.1 the UTV needs a state-machine which continuously monitors the current mode of operation. This state-machine sets and clears flags accordingly such that the end of a current mode cycle is recognized, transitions from one mode to the next is triggered and special events are serviced. The operation of the UTV during calculation of a path and tracking of the calculated path is therefore controlled by this state machine. The state machine provides a pre-defined sequence in which routines must execute and in doing so enables the UTV to accomplish the tasks from Section 4.1 which are required during autopilot navigation.

The state-machine can be subdivided into two modules. The two modules are the Path Planner module and the Controller module. Flowcharts for these modules are shown in Figures 4.9 and 4.10 respectively. These two flowcharts show the logical flow of events of each module, during each 25 *Hz* cycle, where the code, represented by the flowchart of Figure 4.9, executes before the code represented by Figure 4.10. The approach consists of defining several boolean flags which are updated during each 25 *Hz* sample time interval. The status of these flags indicate the current mode of operation of the UTV, and control the order in which routines and algorithms execute.

The Path Planning module is responsible for the calculation of paths through use of specific user-selected algorithms as well as passing the path line segments to the Controller module incrementally. The Controller module is responsible for updating the UTV controllers as well as activating and deactivating controllers appropriately in order to track the path obtained from the Path Planning module.

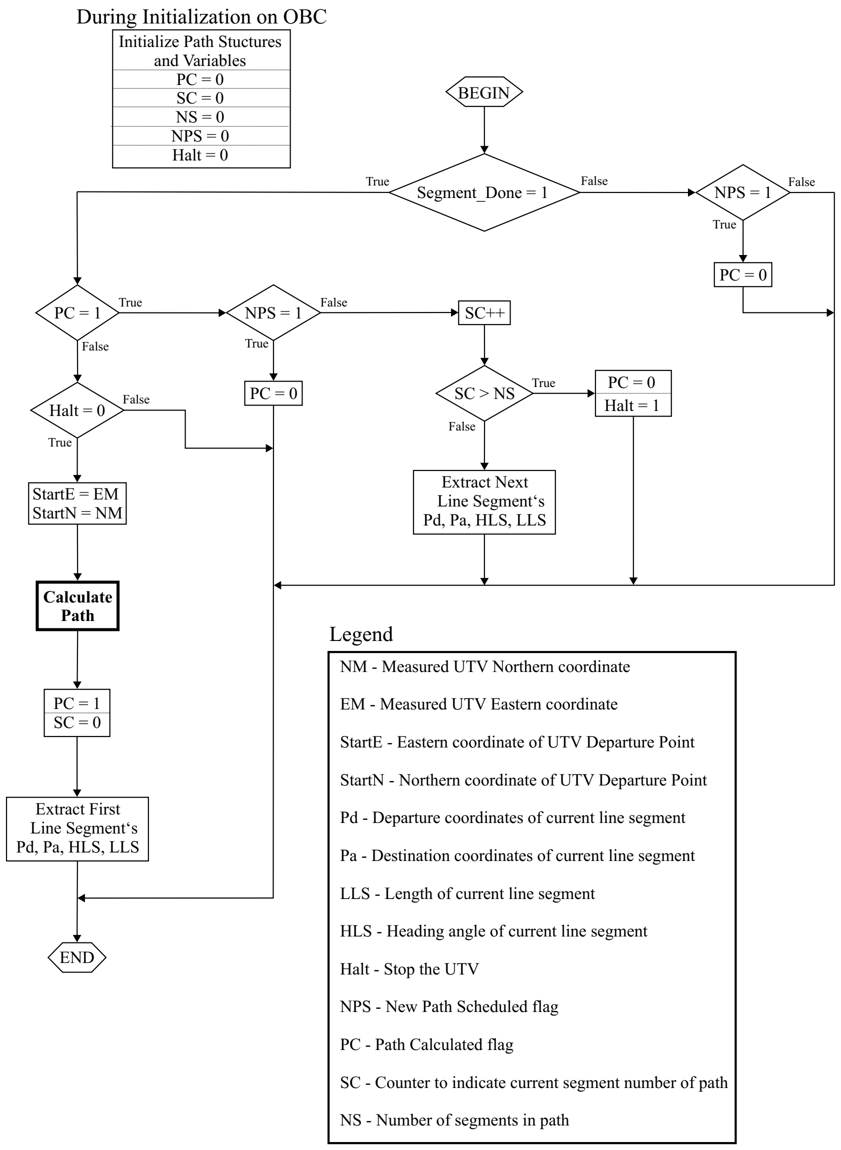

During Initialization on OBC
Initialize Path Stuctures and Variables
PC = 0
SC = 0
NS = 0
NPS = 0
Halt = 0
BEGIN
Segment_Done = 1
True
False
NPS = 1
True
False
PC = 0
PC = 1
True
False
NPS = 1
False
True
SC++
PC = 0
Halt = 0
False
True
SC > NS
True
False
PC = 0
Halt = 1
StartE = EM
StartN = NM
Extract Next Line Segment's Pd, Pa, HLS, LLS
Calculate Path
PC = 1
SC = 0
Extract First Line Segment's Pd, Pa, HLS, LLS
END
Legend
NM - Measured UTV Northern coordinate
EM - Measured UTV Eastern coordinate
StartE - Eastern coordinate of UTV Departure Point
StartN - Northern coordinate of UTV Departure Point
Pd - Departure coordinates of current line segment
Pa - Destination coordinates of current line segment
LLS - Length of current line segment
HLS - Heading angle of current line segment
Halt - Stop the UTV
NPS - New Path Scheduled flag
PC - Path Calculated flag
SC - Counter to indicate current segment number of path
NS - Number of segments in path

Figure 4.9: Flowchart of Path Planner Module State-Machine

The order of routines in one module is partially determined by the status of flags which are only set in the other module and vice versa. Certain flags are also set externally through user input from the UTV's RF ground station and further determine the flow of events. The most vital external flag being the *new path scheduled* flag which is set as soon as a new obstacle is uploaded from the ground station to simulate the process of obstacle detection.

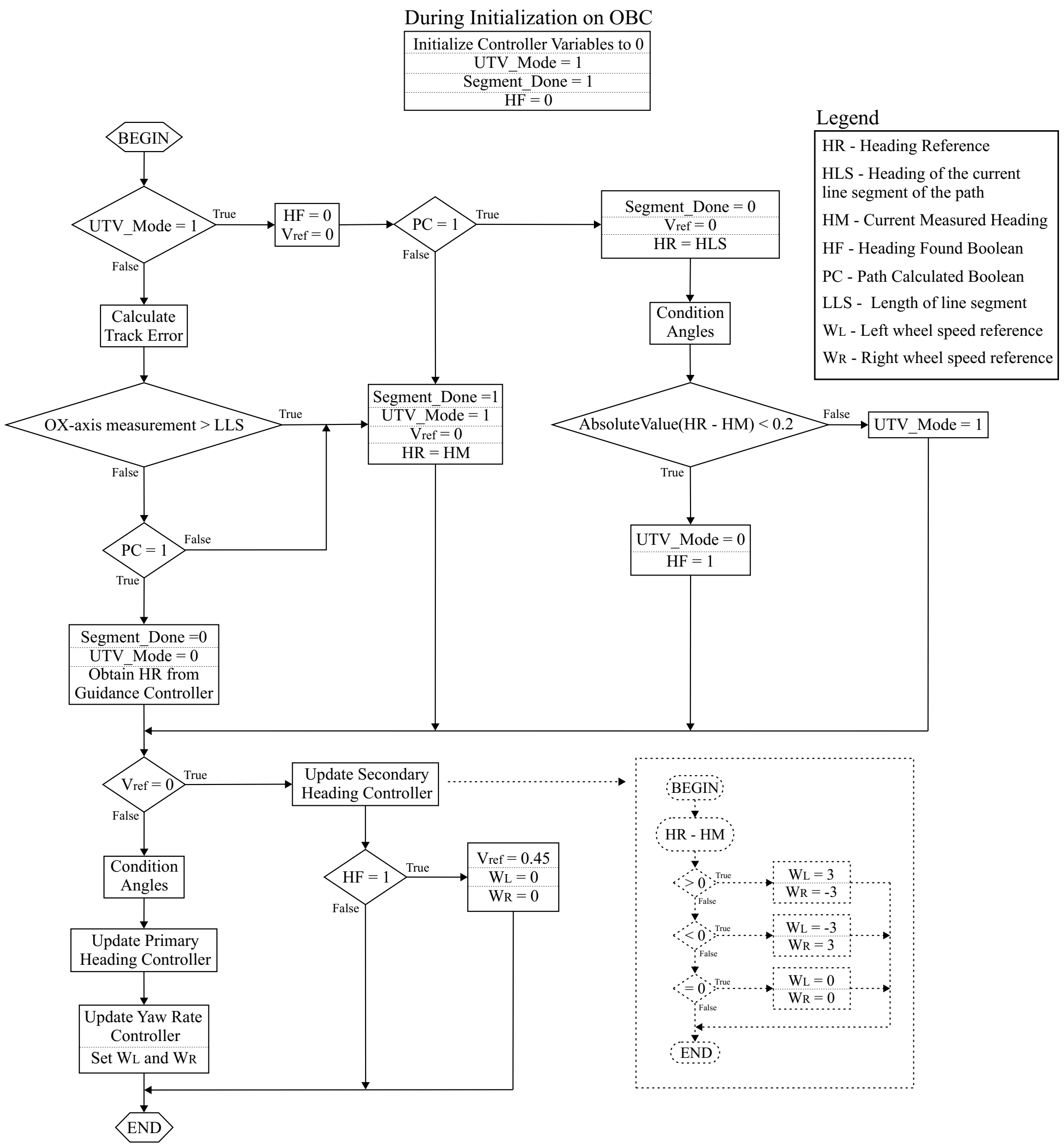

During Initialization on OBC
Initialize Controller Variables to 0
UTV_Mode = 1
Segment_Done = 1
HF = 0
Legend
HR - Heading Reference
HLS - Heading of the current line segment of the path
HM - Current Measured Heading
HF - Heading Found Boolean
PC - Path Calculated Boolean
LLS - Length of line segment
WL - Left wheel speed reference
WR - Right wheel speed reference
BEGIN
UTV_Mode = 1
True
False
HF = 0
Vref = 0
PC = 1
Segment_Done = 0
Vref = 0
HR = HLS
Calculate Track Error
Condition Angles
OX-axis measurement > LLS
Segment_Done =1
UTV_Mode = 1
Vref = 0
HR = HM
AbsoluteValue(HR - HM) < 0.2
UTV_Mode = 1
UTV_Mode = 0
HF = 1
PC = 1
Segment_Done =0
UTV_Mode = 0
Obtain HR from Guidance Controller
Vref = 0
Update Secondary Heading Controller
HF = 1
Vref = 0.45
WL = 0
WR = 0
Condition Angles
Update Primary Heading Controller
Update Yaw Rate Controller
Set WL and WR
END
HR - HM
> 0
< 0
= 0
WL = 3
WR = -3
WL = -3
WR = 3
WL = 0
WR = 0

Figure 4.10: Flowchart of Controller Module State-Machine

The selection between the primary and secondary heading controllers, discussed previously, can be seen in Figure 4.10. The primary controllers in this figure contain the difference equations digital implementation of the controllers previously discussed in this chapter. The yaw rate controller also contains the mathematical relationships to yield wheel speeds from reference common mode speed, **V**, and yaw rate controller output, $u(k)$, as shown in

Figure 4.4 and Figure 4.5.

Also shown is the guidance controller and how the end of a line segment is simply indicated by the OX-axis measurement exceeding the length of the current line segment obtained from the path planning module. The secondary heading controller consists of the simple routine shown in dotted lines. The flow diagram also indicates how it is determined whether the UTV has achieved the heading angle of the next line segment with this secondary controller, after which the *UTV_Mode* is changed to 0 to indicate that straight line segment tracking should be started with the primary controllers.

Figure 4.10 also shows the *Condition Angles* routine. These routines are implemented due to the non-unique nature of heading angle measurements. An angle of for instance 70° can also be represented with 430° due to the 360° duplicate mapping characteristic of angles. Since the state estimator, discussed in Chapter 5, makes use of integral routines on the yaw rate measurement to obtain heading measurements, the possibility of angles below $-360°$ and above 360°, which is equally representable by an equivalent angle between $-360°$ and 360°, is present. Since the path planning algorithms make use of the four quadrant inverse *tan* function all line segment headings will always be between $-180°$ and 180°. The UTV heading controller could therefore compare angles, smaller than $-360°$ or bigger than 360°, with angles below 360° to obtain heading error signals which would lead to full revolutions of the UTV to achieve the commanded heading reference, when less than a full revolution is always sufficient. A further difficulty is the fact that the UTV can either turn anti-clockwise or clockwise to achieve it's new heading but the one which is preferable is the one which causes the UTV to only turn through an acute or obtuse (smaller than 180°) angle. All angles are therefore conditioned to be between 0° and 360° and further conditioned so that heading differences are never bigger than 180° and the smallest possible turn to achieve the new heading is always chosen by the UTV. This was however just implemented in simulation and not on the actual UTV for reasons which will be stated in the *Practical Results* chapter. The flowchart of this routine can be seen in Appendix A.

The *Calculate Path* routine, indicated in bold print in Figure 4.9, is the core routine of the path planner module which makes use of the path planning algorithms, discussed in Chapter 3, to calculate an optimal path through the obstacle ridden environment. This routine is responsible for the selection of a certain *population* algorithm in conjunction with a specific *shortest path* algorithm, according to the user selection from the RF ground station.

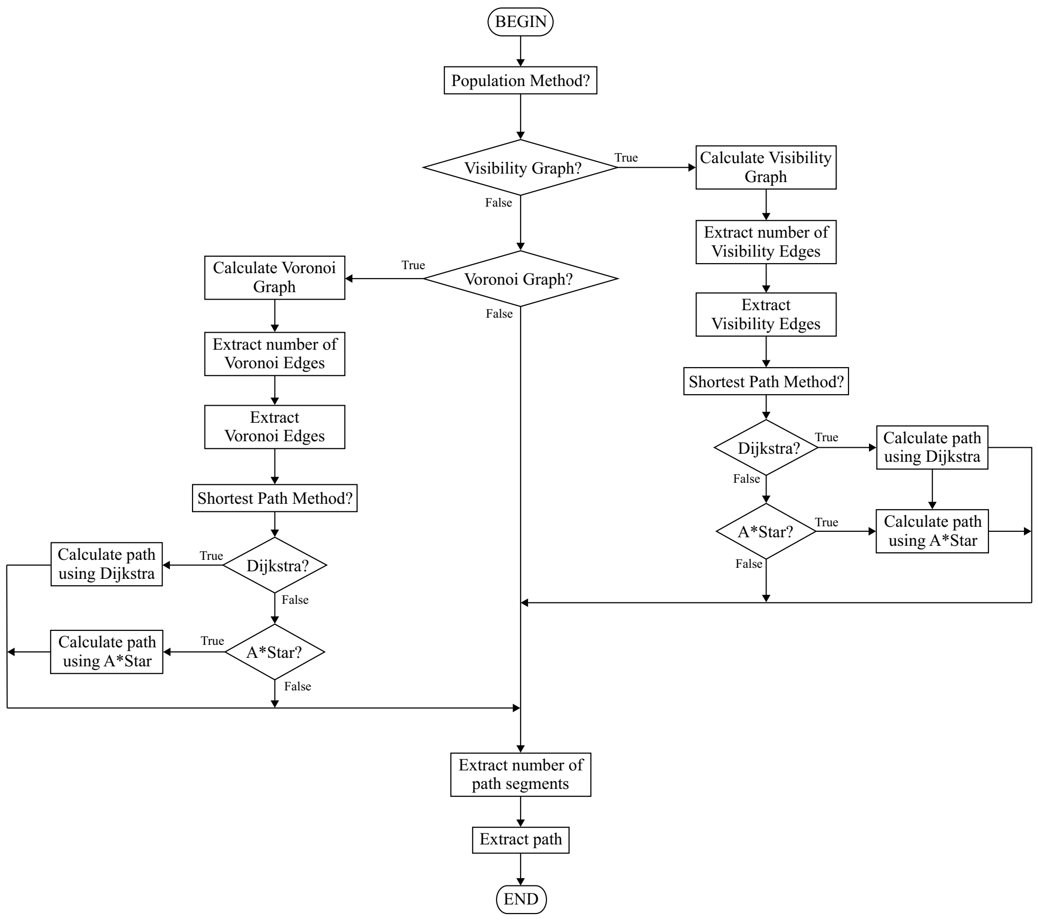

BEGIN
Population Method?
Visibility Graph?
True
False
Calculate Visibility Graph
Extract number of Visibility Edges
Extract Visibility Edges
Shortest Path Method?
Dijkstra?
A*Star?
Calculate path using Dijkstra
Calculate path using A*Star
Voronoi Graph?
Calculate Voronoi Graph
Extract number of Voronoi Edges
Extract Voronoi Edges
Shortest Path Method?
Dijkstra?
A*Star?
Calculate path using Dijkstra
Calculate path using A*Star
Extract number of path segments
Extract path
END

Figure 4.11: Flowchart of Calculate Path Function

A flowchart of this routine can be seen in Figure 4.11 which shows how the specific algorithms are selected. Upon exit from this routine the amount of segments present in the calculated path is returned, as well as the calculated path in matrix format with each line of the matrix representing one line segment of the path with its departure and destination coordinates, heading angle and length.

4.7 Non-linear Simulator

As mentioned in Chapter 1, simulation plays a big role in minimizing the amount of test runs which are necessary for the UTV to accomplish what is required. Instead of using a process of trial and error with direct implementation on the OBC, a preliminary stage of simulation is used to verify designs and perform as much debugging as possible before attempting actual implementation on the UTV. The simulator integrates all the controllers, algorithms and models developed and serves as way of judging the autopilot's performance as a whole. Simulation was done in Simulink where system blocks were predominantly implemented through use of *Matlab*'s S-Function blocks. The use of S-Function blocks is preferred since it implies implementation of all systems in C programming language which is the same language used for implementation on the OBC. This makes transfer of routines from simulation to actual hardware possible with minimal effort. The similarity in implementation also allows for consistent debugging on simulation and OBC alike.

4.7.1 Entire Simulation

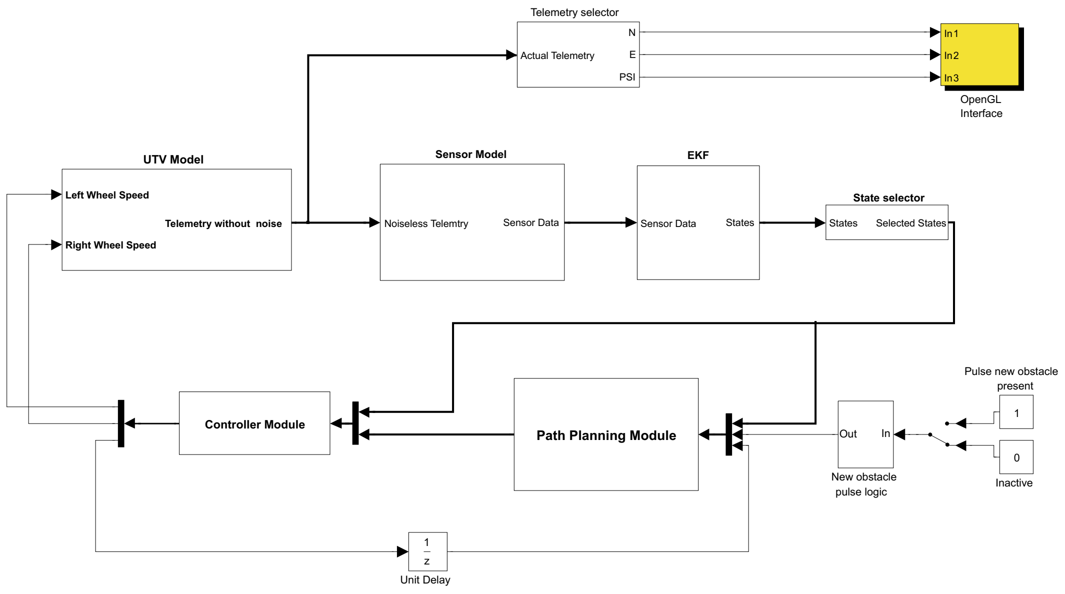


Figure 4.12: Highest Level of Simulink Simulator

The Simulink block diagram of the entire system can be seen in Figure 4.12. The simulator can be subdivided into two categories of components, these categories being, components which represent the algorithms to be implemented on the OBC, and components which act as a model of the physical UTV. The modelling components are implemented predominantly through use of lower level Simulink blocks with occasional use of S-Function blocks.

These components include the *UTV Model* and *Sensor Model* shown in Figure 4.12 which are not to be implemented on the OBC but merely serve as a means of predicting how the actual UTV will respond to the control algorithms during test runs.

The other category includes the *Path Planning*, *Controller* and *EKF* blocks in Figure 4.12. These system blocks are all simulated through use of *Matlab* S-Functions and are to be implemented on the OBC. Further notice should be taken of the unit delay block which represents the fact that the status of boolean flags updated on the Controller module state-machine, discussed previously in this chapter, are only recognized by the Path Planning module during the next 25 *Hz* cycle on the OBC, due to the fact that the code of the Controller module executes after the code of the Path Planning module.

Figure 4.12 also shows the *New Obstacle Pulse Logic* components. These blocks are used to simulate the process of a new obstacle being uploaded from the ground station, while the UTV is busy navigating a path, by setting the *new path scheduled* flag, in Figure 4.9. The logical block makes use of simple components to ensure the pulse is only present for the duration of one cycle, since this is what will happen during test runs. That is, a new obstacle is uploaded from the ground station, the *new path scheduled* flag is set and then cleared again after the Path Planning module has taken notice.

One final system block remains which is the *OpenGL Interface*. This block links the real-time telemetry of the UTV to a graphical interface and serves as way of monitoring the behaviour of the UTV model while the simulation executes. Since more attention is given to the two-dimensional path planning algorithms in this project, the graphical simulator is merely used as a supplementary way of debugging, and other graphical plots were used and discussed Chapter 3. The interface was therefore not altered from [4] and [15] and still contains a 3D aeroplane model. The aeroplane now merely taxi's around on the ground surface without taking off but still provides valuable insight into the path traveled when the tracking feature of the graphical interface is activated. Figure 4.13 shows the graphical interface with some color alteration for improved printing. The aeroplane represents the UTV in this case and the white path is drawn as the UTV navigates toward its destination, providing valuable insight into whether the path is being tracked well enough with the Extended Kalman Filter (state estimator) being used in conjunction with noisy measurements. Insight is also gained into how the UTV executes turns and whether the angle conditioning previously mentioned is functioning correctly.

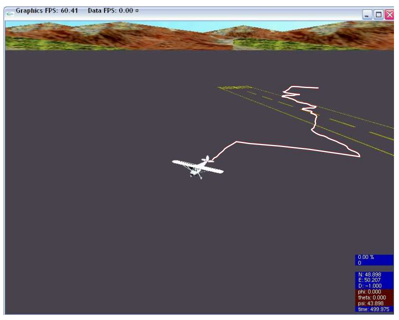


Figure 4.13: Graphical Simulator Interface

4.7.2 Path Planning, Controller and EKF Simulation Blocks

As mentioned the Path Planning, Controller and EKF simulator blocks are implemented with S-Functions. The S-Function for the Controller block is an implementation of the flowchart in Figure 4.10 in C programming language. This block has the *path calculated* flag and path line segment data from the Path Planning module as inputs, as well as the telemetry obtained from the Extended Kalman Filter S-Function. In turn the controller block updates all the controllers previously discussed in this chapter to arrive at appropriate wheel speed commands as outputs to the UTV Model. The *segment done* flag is also included in the outputs and is relayed back to the Path Planning block through a unit sample time delay.

The Path Planning block contains an S-Function implementation of the flowchart in Figure 4.9 and has the *segment done* flag from the Controller block, the telemetry from the EKF block and the *new path scheduled* flag from the external source as inputs. The choice in population algorithm and shortest path algorithm is passed to the block through the S-Function block parameters and these parameters are altered in a user entry M-file. The Path Planning Block has the outputs previously mentioned as the Controller block's inputs and arrives at these outputs by executing the user's choice of path planning algorithm, as shown in Figure 4.11. After this it continuously updates the state-machine flags, in accordance with the flowchart of Figure 4.9, until the destination has been reached or a new path is scheduled.

The EKF block is an implementation of the Extended Kalman Filter (state estimator) discussed in Chapter 5. It has the noisy sensor measurements from the Sensor Model as inputs and the estimated states of the UTV as output. All relevant data such as noise correlation matrices are passed to this block through use of the S-Function block parameters. These three S-Function blocks were easily transferred to the OBC after debugging and only minor adjustments and modifications to the structure of the code was required.

4.7.3 UTV Model Block

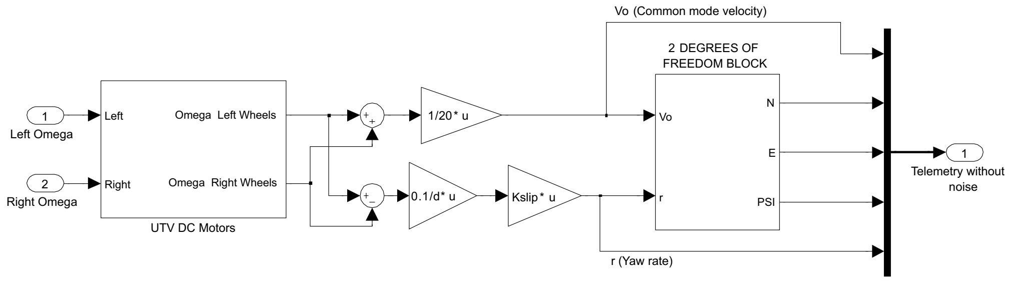


Figure 4.14: UTV Model Block of Block Diagram Simulator

The UTV Model block of the simulator consists of the components shown in Figure 4.14. The DC Motors block contains the model of the closed loop drive systems shown in Figure 4.1. Both DC motors are represented equally by this model and the slip gain can also be seen in Figure 4.14, which was found through calibration as previously mentioned. Also shown are the gains required to arrive at a yaw rate and common mode forward velocity output and these gains are the same as shown in Figure 4.4. The yaw rate and common mode velocity outputs enter a two degree of freedom block which calculates the UTV's telemetry through utilization of the equations shown in Figure 4.3. This then provides a sound model of the UTV, its behaviour and the actual telemetry which will be obtained in response to wheel speed references during test runs.

4.7.4 Sensor Model

In order to test the functionality of the Extended Kalman Filter, sensor measurements such as the ones used on the actual UTV need to be simulated. These sensor measurements are accompanied by noise which also needs to be included in the simulation. The approach is to take the telemetry values obtained from the UTV model, discussed in the previous section, and obtain sensor measurements with noise from these values.

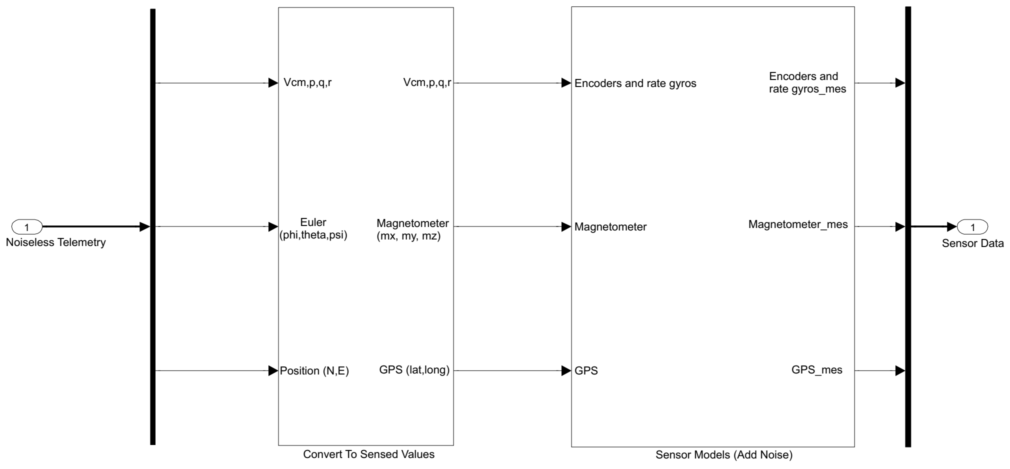


Figure 4.15: Sensor Model Block of Block Diagram Simulator

As shown in Figure 4.15 the process consists of two stages. During the first stage reverse calculation is done from the noiseless telemetry signals to arrive at the raw signals provided by the actual sensors. During the second stage band-limited white noise is added to these raw signals to represent the noise present on the measurements obtained from the UTV's low cost sensors.

The components of the first stage can be seen in Figure 4.16. The functions used to calculate the magnetometer's body reference measurements and GPS latitude and longitude measurements are both implemented with S-Functions. These two functions are the same as those used by [15]. A constant block is also shown which represents the Earth's Magnetic Field at Stellenbosch in the NED axis system. The *Trim GPS* block is used to set the reference origin of the navigational system and is chosen arbitrarily. During physical implementation the trim coordinates are obtained by taking the average of several successive GPS measurements at the UTV's first stationary position, before navigation commences. The pitch rate and roll rate of the vehicle as well as the pitch and roll Euler angles are assumed to be zero and simply connected to the zero constant blocks which are not shown in Figure 4.16.

The components of the second stage of the Sensor Model simply consist of band-limited white noise blocks with their outputs being added to the sig-

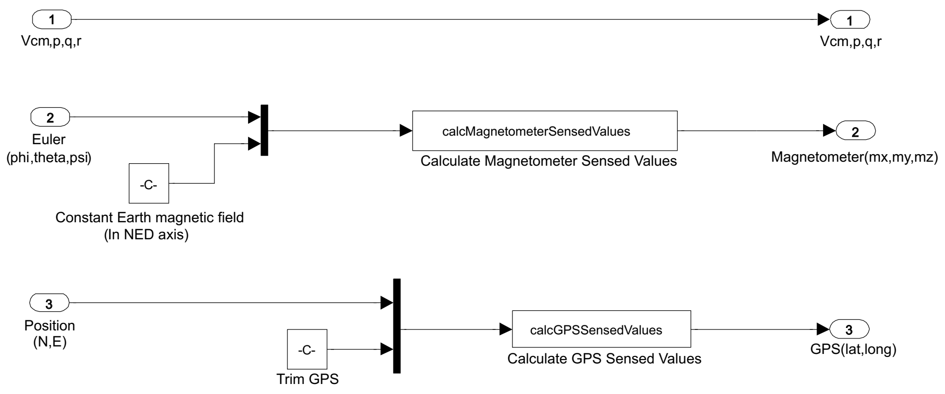


Figure 4.16: Convert to Sensed Values Block of Sensor Model Block

nals obtained from stage one of the Sensor Model. The covariances of the sensors on the UTV were determined partially from measurements and partially from datasheets and used as parameters in these band-limited white noise blocks.

4.7.5 Simulator Results

The simulator provided invaluable insight into the feasibility of the controllers, navigation algorithms and co-operation of the individual system components. Through careful debugging and several iterations in simulation the implementation of the controllers and algorithms on the OBC was made possible with minimal effort. During test runs the UTV displayed behaviour similar and to a certain extent even identical to what was seen in simulation and the importance of this intermediary simulation stage can therefore not be emphasized enough. Telemetry results obtained with this simulator were plotted and are presented with the development of the state estimator in Chapter 5 since they also provide valuable insight into the functionality of the estimator. Although the graphical simulator mentioned earlier in this chapter was not given much attention it should be credited for providing a very efficient, intuitive tool for identifying problem areas quickly and establishing immediately whether the algorithms implemented are fundamentally sound.

4.8 Summary

In this chapter a discussion was given on the controllers implemented, from the lowest level of drive system control to the highest level of guidance control. The design methods used during implementation of the controllers were presented as well as the step response characteristics of some of the controllers. The state-machine responsible for combining these controllers and scheduling UTV activity was presented and gives insight into how the UTV accomplishes the tasks, mentioned in the beginning of this chapter, which relate to movement of the UTV during navigation. The tasks mentioned which were not addressed is discussed in Appendix C on ground station implementation. Finally a non-linear simulator was presented which provided invaluable insight into the system as a whole and ensures proper preparation before implementation on the OBC.

Chapter 5

State Estimation and Simulation Results

The required states of the UTV are directly available from sensor measurements. In fact, an excess of measurements are available since the architecture adopted from [12] and [8] accommodates three dimensional motion, and encoder measurements are also available. Although it is possible to make use of only the required sensor measurements directly in the adopted control strategy, a more optimal approach would be to combine these measurements in an optimal way such that sensitivity to noise is reduced. The presence of noise is inevitable when considering the low cost sensors used. An efficient way of doing this is through implementation of a recursive extended Kalman Filter. A full six degree of freedom filter such as this has already been introduced by [15]. This chapter is therefore dedicated to the development of a simplified two dimensional version of this filter with new non-linear kinematic equations. The theory of this extended Kalman Filter is based on discussions in [13] and is only briefly introduced before the application thereof is shown.

5.1 Overview

Although optimal measurements in a noisy environment can be regarded as the main motivation for use of an EKF in this project, a more specific driving factor is also present. The GPS measurements alone have an accuracy too low to justify its sole use in obtaining position coordinates of the UTV, especially when considering the tolerances for obstacle avoidance which is in the order of a meter or less within the scale of the testing terrain used for practical demonstrations in this project. A supplementary method therefore needs to be introduced to provide position measurements of a higher resolution. The implementation of an EKF allows the addition of a "dead-reckoning" system while still making use of the GPS measurement updates as will be shown shortly.

As seen in Chapter 4 the control strategies used require measurements of the following states of the UTV for feedback

- N - Northern position coordinate
- E - Eastern position coordinate
- ψ - Yaw angle
- r - Yaw rate

In order to optimally obtain these measurements through use of a state estimator kinematic equations are derived which fully define the UTV's motion in a two dimensional NE reference fame. These kinematic equations are shown in Figure 4.3. When using these equations and considering the available measurements from sensors on the UTV, a structure for the kinematics of the state estimator is easily found. A representation of this kinematic structure is shown in Figure 5.1.

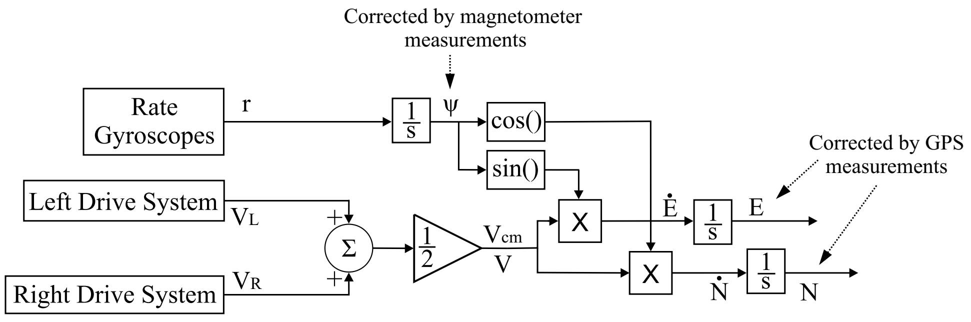


Figure 5.1: State Estimator Kinematics

The figure shows a measurement of the vehicle's yaw rate obtained from a platform, in a strapdown configuration, which is equipped with rate gyroscopes and accelerometers (not used in this project). The yaw rate obtained from the Z-body-axis gyroscope is integrated to yield the UTV's heading angle directly. The angular velocities of the wheels on both side of the UTV are obtained from the encoders, discussed in Chapter 2, and through a simple averaging sequence the common mode forward velocity, V_{cm}, of the UTV is found which is also defined as V. With the trigonometric relationships of the equations in Figure 4.3 the two axis components of the UTV's velocity is then found. The velocity components are then integrated to yield the two dimensional position of the UTV in the NE reference frame.

The heading angle state and position states are therefore functions of integration processes in terms of the rate gyroscope measurement, encoder measurements and propagated heading angle state. This implies that the accuracy of these states will deteriorate with time due to noise on the measurements and approximations made when using discrete integration routines. As shown in Figure 5.1 a solution to this is the correction of the accumulated deterioration in states, at slower intervals, by making use of the direct GPS measurements of position and the indirect measurement of the heading angle provided by the magnetometer. The benefit of a finer resolution in positional states at 25 Hz, through use of a "dead-reckoning" integration process, is therefore combined with the GPS measurements at 4 Hz to ensure positional state deterioration is bounded. It should be noted that the yaw rate is not estimated but simply used as an input to the kinematic plant of the EKF. The yaw rate controller, from Chapter 4, therefore makes use of the raw rate gyroscope measurement in a direct feed through configuration on the EKF.

5.2 Optimal State Estimation Theory

The theory presented in this section comes directly from the discussions of [13] and is presented in a fashion very similar to the presentation in [15]. Nevertheless, a presentation of the application of this theory will have no meaning to the reader if the basic concepts are not introduced first. The theory behind optimal state estimation is therefore briefly summarized again to provide a foundation for this chapter. Some equations will not be listed here and the reader is referred to [15] for a more detailed discussion.

A discrete linear plant is represented by the following equations,

$$\mathbf{x}(k+1) = \mathbf{\Phi}\mathbf{x}(k) + \mathbf{\Gamma}\mathbf{u}(k) + \mathbf{\Gamma}_w w(k) \tag{5.2.1}$$

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) + v(k) \tag{5.2.2}$$

where $v(k)$ represents the measurement noise and $w(k)$ represents the plant's process noise. These noises are randomn processes which have a zero mean and are uncorrelated. The discrete process noise covariance matrix is defined by $\mathbf{Q}(k)$ and the measurement noise matrix by $\mathbf{R}(k)$. The state error vector, $\mathbf{e}(k)$, is defined as the difference between the estimated and actual state vectors. An error covariance matrix can now be defined as shown in Equation

5.2.4.

$$\begin{aligned}\mathbf{e}(k) &= \mathbf{x}(k) - \hat{\mathbf{x}}(k) &(5.2.3)\\ \mathbf{P}(k) &= E[\mathbf{e}(k)\mathbf{e}^T(k)] &(5.2.4)\end{aligned}$$

where $E[\cdot]$ represents the expected value operator. A recursive algorithm is used which implements the Kalman Filter equations and performs real-time optimal estimation for the states of a discrete linear plant. The steps of this algorithm are summarized below.

1. Through use of the plant dynamics and the deterministic input, $\mathbf{u}(k)$, the previous best estimate, $\hat{\mathbf{x}}(k)$, is propagated forward in time to yield $\overline{\mathbf{x}}(k+1)$. The error covariance, $\mathbf{M}(k+1)$, of the propagated state is also calculated.

$$\begin{aligned}\overline{\mathbf{x}}(k+1) &= \mathbf{\Phi}\hat{\mathbf{x}}(k) + \mathbf{\Gamma}\mathbf{u}(k) &(5.2.5)\\ \mathbf{M}(k+1) &= \mathbf{\Phi}\mathbf{P}(k)\mathbf{\Phi}^T + \mathbf{Q}(k) &(5.2.6)\end{aligned}$$

2. The filter gain, $\mathbf{L}(k+1)$, is calculated whenever a measurement update from the GPS or magnetometer is available. This is done by noting the uncertainties of both the propagated state and the actual measurements and then yielding a filter gain which weighs one of the two more heavily in such a way as to optimize accuracy of the estimated state.

$$\mathbf{L}(k+1) = \mathbf{M}(k+1)\mathbf{H}^T[\mathbf{H}\mathbf{M}(k+1)\mathbf{H}^T + \mathbf{R}(k+1)]^{-1} \qquad (5.2.7)$$

3. By using the current calculated filter gain, $\mathbf{L}(k+1)$, and the error between the actual and propagated measurements in a recursive weighted least squares manner, the states of the estimator are now updated. The error covariance of the newly acquired best state estimate is also calculated,

$$\hat{\mathbf{x}}(k+1) = \overline{\mathbf{x}}(k+1) + \mathbf{L}(k+1)[\mathbf{y}(k+1) - \mathbf{H}\overline{\mathbf{x}}(k+1)] \qquad (5.2.8)$$

$$\begin{aligned}\mathbf{P}(k+1) &= [\mathbf{I} - \mathbf{L}(k+1)\mathbf{H}]\mathbf{M}(k+1)[\mathbf{I} - \mathbf{L}(k+1)\mathbf{H}]\\ &\quad + \mathbf{L}(k+1)\mathbf{R}(k+1)\mathbf{L}(k+1)^T &(5.2.9)\end{aligned}$$

where $\mathbf{I}$ represents the identity matrix. The algorithm then repeats during the next sample time instant and these newly acquired error covariance and state matrices are used once again to propagate to the next states and so the process continues.

As seen in Figure 4.3 the kinematic equations which define motion of the UTV in the NE reference frame are non-linear. A method for linearizing the non-linear system, so that the recursive algorithm just discussed can be applied, is therefore required. The Extended Kalman Filter provides a solution to this problem by approximating the non-linear system as a linear time variant system. The way in which this linearization process is approached will now be discussed briefly and is once again based on the discussions in [13] and [15].

A continuous non-linear system can be represented in the following form,

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) + w \tag{5.2.10}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) + v \tag{5.2.11}$$

where f is a non-linear function in terms of the state and control vector and $\mathbf{h}$ is a non-linear function in terms of the state vector. The continuous process noise, w, is white and has a power spectral density (PSD), $\mathbf{Q}_c$, while the measurement noise has a PSD, $\mathbf{R}_c$. The linearization and discretization steps required to implement an Extended Kalman Filter for a non-linear, continuous plant such as this one, which defines motion of the UTV in a NE reference frame, are now summarized.

1. The non-linear dynamics of the plant, represented by Equation 5.2.10, are linearized about the previous best estimate of the state vector by calculating the Jacobian matrices and evaluating them at $\hat{\mathbf{x}}(k)$ and $\mathbf{u}(k)$,

$$\mathbf{F_k} = \left[\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right]_{\hat{\mathbf{x}}(k), \mathbf{u}(k)} \tag{5.2.12}$$

$$\mathbf{G_k} = \left[\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right]_{\hat{\mathbf{x}}(k), \mathbf{u}(k)} \tag{5.2.13}$$

2. These linearized matrices are then converted to the discrete time domain using the discrete sample time, T_s, which is 40 ms in this project. Certain approximations can be used in this discretization process which are valid when the sample time is much shorter than the system time constants. As stated in [1] these approximations are as follows,

$$\mathbf{\Phi}(k) \approx \mathbf{I} + \mathbf{F_k} T_s \tag{5.2.14}$$

$$\mathbf{\Gamma}(k) \approx \mathbf{G_k} T_s \tag{5.2.15}$$

3. The continuous process and measurement noise covariance matrices now need to be converted to equivalent discrete covariance matrices. As mentioned in [15], the process noise for a system often includes both a process noise component, due to unmodeled kinematics, as well as a component originating from noise on the input vector, $\mathbf{u}$, which is not completely deterministic in nature. The low cost sensors used in this project which provide the input measurements, V and r, to the system is shown to display significant noise figures and the assumption is therefore made that noise contributions due to unmodeled kinematics are small in comparison. The discretization of the continuous process noise therefore only involves a discretization of the continuous inputs' PSD matrix, $\mathbf{Q_{uc}}$, to obtain $\mathbf{Q_u}(k)$. The noise properties of the matrix are then mapped onto the states of the system through use of the discrete input matrix $\mathbf{\Gamma}(k)$as follows,

$$\mathbf{Q}(k) = \mathbf{\Gamma}(k)\mathbf{Q_u}(k)\mathbf{\Gamma}(k)^T \tag{5.2.16}$$

According to [1] the continuous process noise PSD matrix is related to its discrete covariance counterpart, when the sample time is much shorter than the system time constants, as follows,

$$\mathbf{Q_u}(k) = \frac{\mathbf{Q_{uc}}}{T_s} \tag{5.2.17}$$

Since the transpose of a scalar is the same scalar, and the transpose, of a multiplication of a matrix and a scalar, is the same as the multiplication of the transposes of the scalar and the matrix respectively, Equation 5.2.16 can be rewritten, by making use of Equations 5.2.15 and 5.2.17, as follows,

$$\mathbf{Q}(k) = \mathbf{G_k}\mathbf{Q_{uc}}\mathbf{G_k}^T T_s \tag{5.2.18}$$

Equation 5.2.18 is ultimately used in the implementation of the EKF, as shown later in this chapter. According to Equation 5.2.11 the measurement updates are available continuously. As mentioned in [15] this is however not the case in practice and EKF's are mostly updated sporadically. The noise properties for the sensors are also usually provided as discrete noise covariances. A discretization of a continuous measurement noise covariance matrix is therefore assumed unnecessary and the measurement noise properties are directly represented with the discrete covariance matrix $\mathbf{R}(k)$.

4. The next step is to propagate the previous best state estimate forward

in time by integrating the non-linear system dynamics over one sample time with a simple Euler integration,

$$\bar{\mathbf{x}}(k+1) = \hat{\mathbf{x}}(k) + \dot{\hat{\mathbf{x}}}(k)T_s \tag{5.2.19}$$

where

$$\dot{\hat{\mathbf{x}}}(k) = \boldsymbol{f}(\hat{\mathbf{x}}(k), \mathbf{u}(k)) \tag{5.2.20}$$

The error covariance on the propagated states are also calculated by making use of the linearized discrete system dynamics, and noise properties obtained from Equation 5.2.18,

$$\mathbf{M}(k+1) = \mathbf{\Phi}(k)\mathbf{P}(k)\mathbf{\Phi}(k)^T + \mathbf{Q}(k) \tag{5.2.21}$$

5. With the propagation of states completed for this sample time instance the filter checks for availability of a measurement update from the magnetometer and GPS. If a new measurement update is available the non-linear measurement/output matrix, $\mathbf{h}(\mathbf{x})$, is linearized about the propagated state vector shown in the previous step,

$$\mathbf{H}(k+1) = \left[\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\right]_{\bar{\mathbf{x}}(k+1)} \tag{5.2.22}$$

and the new filter gain is calculated,

$$\mathbf{L}(k+1) = \mathbf{M}(k+1)\mathbf{H}(k+1)^T[\mathbf{H}(k+1)\mathbf{M}(k+1)\mathbf{H}(k+1)^T + \mathbf{R}(k+1)]^{-1} \tag{5.2.23}$$

6. Similarly to the innovation update for the linear Kalman Filter previously shown, the states of the estimator are now optimally updated with the newly acquired filter gain and the accompanying error covariance matrix is also updated,

$$\hat{\mathbf{x}}(k+1) = \bar{\mathbf{x}}(k+1) + \mathbf{L}(k+1)[\mathbf{y}(k+1) - \mathbf{h}(\bar{\mathbf{x}}(k+1))] \tag{5.2.24}$$

$$\begin{aligned}\mathbf{P}(k+1) &= [\mathbf{I} - \mathbf{L}(k+1)\mathbf{H}(k+1)]\mathbf{M}(k+1)[\mathbf{I} - \mathbf{L}(k+1)\mathbf{H}(k+1)] \\ &\quad + \mathbf{L}(k+1)\mathbf{R}(k+1)\mathbf{L}(k+1)^T \end{aligned} \tag{5.2.25}$$

When no measurement updates are available the output matrix, $\mathbf{H}(k+1)$, is simply set to zero and consequently the filter gain also goes to zero. In between measurement updates the current best state estimate by the EKF is simply the propagated state. This propagated state represents the "dead-reckoning" system which calculates position coordinates with an integral process on the encoder measurements and yaw rate measurement, as previously mentioned. The steps presented are followed to implement and extended Kalman Filter which estimates the states of the UTV, required for the control strategies of Chapter 4, in real-time. To start the process the following initial states are required,

- The continuous process noise covariance matrix, $\mathbf{Q_{uc}}$
- The discrete measurement noise covariance matrix, $\mathbf{R}(k)$
- The initial state vector, $\hat{\mathbf{x}}(0)$, and error covariance matrix, $\mathbf{P}(0)$

5.3 Approximated Inertial Reference Frame

Before presenting the implementation of the EKF a brief discussion is given on the reference frame used. The UTV's position and heading angle is defined in an approximated inertial reference frame identical to the North-East-Down (NED) system introduced by [15], with the exception of a few added stipulations and reductions for two-dimensional use. This NED system is fixed at a certain point on the earth's surface and can be defined at specific positions within an Earth Centered Earth Fixed (ECEF) geocentric system, which is also shown in [15]. The surface of the earth is assumed flat within the area of definition of the NED system and the UTV is assumed to operate on a surface small enough for this assumption to be justified. The assumption is also made that rotational velocities of the vehicle are much larger than the earth's angular rotation and the duration of operation of the UTV is small in comparison and the NED system is therefore assumed to have a non-rotating surface.

During initialization of the EKF a fixed trim position is therefore required from the GPS, in ECEF coordinates, which defines the point at which the NED system is fixed. The aim of this EKF is to estimate the North and East states directly and with the fixed trim position determined, the measurements from the GPS can therefore be converted to NED coordinates, prior to being used as a measurement update in the EKF. This conversion requires the radius of the earth at the origin of the NED system and within the scope of this project a round earth model is assumed which implies that the radius is a constant denoted by R. Since the UTV will always remain grounded on the earth's surface the Down component of the NED system can be regarded as a constant and the system therefore reduces to a North-East (NE) system. This NE system is ultimately the two-axis reference frame used in the kinematic equations of the estimator.

5.4 Implementation of the EKF

5.4.1 Non-Linear State Equations

By making use of the defined NE reference frame non-linear equations can now be written for this system in the form of Equation 5.2.10. These equations are shown in Figure 4.3 and are stated again here,

$$\dot{N} = V\cos\psi \quad (5.4.1)$$

$$\dot{E} = V\sin\psi \quad (5.4.2)$$

$$\dot{\psi} = r \quad (5.4.3)$$

where r is the rate gyroscope yaw rate measurement and V is the common mode forward velocity from the encoders. These two terms represent the inputs to the system which are used for state propagation as earlier stated. The heading angle is denoted by ψ, and N and E denote the respective position components in the NE reference frame. These terms are in turn the states of the system and are to be updated with the magnetometer and GPS measurements respectively, when measurements become available.

As stated in [15] the GPS receiver used has a measurement time delay in the order of 310 *ms*. As mentioned in Chapter 2, the same architecture is adopted in this project and the same GPS receiver used. The delay is therefore also present in this project and should not be left unmodeled. Contrary to the high velocity platform of [15] the UTV platform of this project represents a much lower velocity platform and it could therefore be argued that the effects of this delay are negligible. With the exception of a little added effort, modelling the delay can only prove advantageous and the decision is therefore made to model the delay once again. As shown in [15] this delay is easily modeled with a first order Pad*é* approximation of the form,

$$\frac{x_d(s)}{x(s)} = \frac{-s+\tau_p}{s+\tau_p} \quad (5.4.4)$$

where τ_p denotes the approximation's zero/pole location and the subscript, d, indicates the delayed version of the signal. More detail about this approximation can be found in [15] where it shown that the approximation is valid for an airframe's position and velocity motion variables. It is therefore most certainly valid for this UTV platform where the bandwidths of position and motion variables are significantly less. The measurements obtained from the GPS which are affected by the delay are therefore the N and E position measurements. The difference equation of the Padé approximation, stated above, is,

$$\dot{x_d} = \tau_p x - \dot{x} - \tau_p x_d \tag{5.4.5}$$

and can be applied to Equations 5.4.1 and 5.4.2 to yield,

$$\begin{aligned} \dot{N_d} &= \tau_p N - V\cos\psi - \tau_p N_d \quad (5.4.6) \\ \dot{E_d} &= \tau_p E - V\sin\psi - \tau_p E_d \quad (5.4.7) \end{aligned}$$

The kinematics, excluding rate gyroscope biases, have now fully been modeled and the state vector can be defined as,

$$\mathbf{x} = [\ N \ \ E \ \ \psi \ \ N_d \ \ E_d \ ]^T \tag{5.4.8}$$

and the input vector as,

$$\mathbf{u} = [\ V \ \ r \ ]^T \tag{5.4.9}$$

The non-linear state equation for the UTV is thus,

$$\dot{\mathbf{x}} \ = \ \boldsymbol{f}(\mathbf{x}, \mathbf{u}) = [\ f_1 \ \ f_2 \ \ f_3 \ \ f_4 \ \ f_5 \ ]^T \tag{5.4.10}$$

where f_1 to f_5 are the non-linear functions in terms of the control and state vector, shown by Equations 5.4.1 to 5.4.3, as well as Equations 5.4.6 and 5.4.7, and is summarized below,

$$\begin{bmatrix} \dot{N} \\ \dot{E} \\ \dot{\psi} \\ \dot{N_d} \\ \dot{E_d} \end{bmatrix} = \begin{bmatrix} V\cos\psi \\ V\sin\psi \\ r \\ \tau_p N - V\cos\psi - \tau_p N_d \\ \tau_p E - V\sin\psi - \tau_p E_d \end{bmatrix} \tag{5.4.11}$$

5.4.2 Non-Linear Measurement Equations

The non-linear measurement equations for the system, denoted by $\mathbf{h}(x)$ in Equation 5.2.11, are now defined. As discussed in Section 5.3 the GPS latitude and longitude measurements are converted to NE measurements prior to being used in the EKF. The conversion is done as follows,

$$N_d = (\lambda_d - \lambda_{trim})R \tag{5.4.12}$$

$$E_d = (\phi_d - \phi_{trim})R\cos\lambda_{trim} \tag{5.4.13}$$

where λ_d and ϕ_d denote the delayed latitude and longitude measurements from the GPS, while the trim GPS latitude and longitude measurements, obtained during initialization, are denoted by λ_{trim} and ϕ_{trim} respectively. The constant round earth model radius of the earth is denoted by R. It should be noted, these equations are only valid under the assumption that the distances traveled by the UTV are small in comparison with the scale of the ECEF inertial reference frame and all assumptions made in Section 5.3 are valid.

The GPS can therefore be regarded as providing delayed measurements of the position states of the UTV directly and can therefore be treated as linear measurements. The GPS components of the non-linear measurement equations, represented by $\mathbf{h}(x)$ in Equation 5.2.11, are therefore,

$$h_1 = N_d \qquad\qquad h_2 = E_d \tag{5.4.14}$$

The magnetometer provides measurements of the earth's magnetic field in each body axis component of the UTV. These body axis component's consist of components along each of the following defined axes,

- OX_B-axis - An axis parallel to some longitudinal reference line which points toward the front of the UTV in line with the forward velocity vector V.
- OY_B-axis - Perpendicular to the OX_B-axis, going through the centre of gravity and pointing toward the starboard side of the UTV.
- OZ_B-axis - Perpendicular to the X_BY_B-plane, going through the centre of gravity and pointing downward.

and the magnetometer measurements are denoted by mx, my and mz respectively. Although the earth's magnetic field varies with position along the earth's surface the assumption is once again emphasized that the UTV travels relatively short distances and within the scope of this project the earth's magnetic field can therefore be regarded as constant. Through use of the Euler angle transformation matrix the earth's magnetic field in the NED reference frame can be coordinated in the body axes defined above,

$$\begin{bmatrix} mx \\ my \\ mz \end{bmatrix} = T_{\phi\theta\psi} \begin{bmatrix} B_N \\ B_E \\ B_D \end{bmatrix} \tag{5.4.15}$$

where $T_{\phi\theta\psi}$ denotes the Euler angle transformation matrix and B_N, B_E and B_D denote the earth's constant magnetic field components in the NED reference frame. Since the UTV is assumed to be operated on a level surface the pitch and roll Euler angles can be regarded as zero at all times. The transformation matrix therefore reduces to a matrix in terms of only the yaw angle, ψ, and only the mx and my magnetometer measurements are used. This reduced relationship between magnetometer measurements and the earth's horizontal magnetic field components is,

$$\begin{bmatrix} mx \\ my \end{bmatrix} = T_{\psi} \begin{bmatrix} B_N \\ B_E \end{bmatrix} \tag{5.4.16}$$

where T_ψ denotes the reduced transformation matrix in terms of only the yaw Euler angle as shown below,

$$T_\psi = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \tag{5.4.17}$$

The remaining components of the non-linear measurement equations are therefore determined as,

$$h_3 = \cos\psi B_N + \sin\psi B_E \tag{5.4.18}$$

$$h_4 = -\sin\psi B_N + \cos\psi B_E \tag{5.4.19}$$

and the non-linear measurement equations are summarized as,

$$\mathbf{h}(\mathbf{x}) = [\ h_1 \quad h_2 \quad h_3 \quad h_4\ ]^T \tag{5.4.20}$$

It should be noted that a simpler definition of the measurement equations was identified in hindsight. Consider the fact that the two magnetometer measurements in body-axes, mx and my, provide enough information to calculate the heading angle, ψ, prior to use in the EKF. This can be done with the following equation,

$$\psi = \tan^{-1}\left[\frac{mxB_E - myB_N}{myB_E + mxB_N}\right] \tag{5.4.21}$$

The measurement equations could therefore be redefined as making direct use of the heading angle measurement, ψ, and the entire measurement matrix, including the delayed GPS measurements, could therefore be represented with linear equations. Although not implemented in this manner it is definitely a simplification worth mentioning.

5.4.3 Implementation

With the non-linear continuous system defined an Extended Kalman Filter can now be implemented by using the optimal state estimation theory discussed earlier in this chapter. The first step requires a linearization and discretization of the non-linear plant dynamics of Equation 5.4.11. As mentioned the linearization is done by calculating the Jacobian matrices and evaluating them at the previous best estimate of the state vector and the current inputs. The evaluation simply requires a substitution of values which change continuously on the EKF and only the calculation of the Jacobian matrices are therefore shown here. A linearized discrete matrix for the plant dynamics is found from Equations 5.2.12, 5.4.11 and 5.2.14 as follows,

$$\mathbf{F_k} = \begin{bmatrix} \frac{\partial \dot{N}}{N} & \frac{\partial \dot{N}}{E} & \frac{\partial \dot{N}}{\psi} & \frac{\partial \dot{N}}{N_d} & \frac{\partial \dot{N}}{E_d} \\ \frac{\partial \dot{E}}{N} & \frac{\partial \dot{E}}{E} & \frac{\partial \dot{E}}{\psi} & \frac{\partial \dot{E}}{N_d} & \frac{\partial \dot{E}}{E_d} \\ \frac{\partial \dot{\psi}}{N} & \frac{\partial \dot{\psi}}{E} & \frac{\partial \dot{\psi}}{\psi} & \frac{\partial \dot{\psi}}{N_d} & \frac{\partial \dot{\psi}}{E_d} \\ \frac{\partial \dot{N}_d}{N} & \frac{\partial \dot{N}_d}{E} & \frac{\partial \dot{N}_d}{\psi} & \frac{\partial \dot{N}_d}{N_d} & \frac{\partial \dot{N}_d}{E_d} \\ \frac{\partial \dot{E}_d}{N} & \frac{\partial \dot{E}_d}{E} & \frac{\partial \dot{E}_d}{\psi} & \frac{\partial \dot{E}_d}{N_d} & \frac{\partial \dot{E}_d}{E_d} \end{bmatrix}_{\hat{\mathbf{x}}(k),\mathbf{u}(k)} \tag{5.4.22}$$

$$\begin{aligned} \mathbf{\Phi}(k) &\approx \mathbf{I} + \mathbf{F_k} T_s \\ &\approx \begin{bmatrix} 1 & 0 & -VT_s \sin\psi & 0 & 0 \\ 0 & 1 & VT_s \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \tau_p T_s & 0 & VT_s \sin\psi & 1-\tau_p T_s & 0 \\ 0 & \tau_p T_s & -VT_s \sin\psi & 0 & 1-\tau_p T_s \end{bmatrix}_{\hat{\mathbf{x}}(k),\mathbf{u}(k)} \end{aligned} \tag{5.4.23}$$

A linearized discrete version of the input matrix is found from Equations 5.2.13, 5.4.11 and 5.2.15 as follows,

$$\mathbf{G_k} = \begin{bmatrix} \frac{\partial \dot{N}}{V} & \frac{\partial \dot{N}}{r} \\ \frac{\partial \dot{E}}{V} & \frac{\partial \dot{E}}{r} \\ \frac{\partial \dot{\psi}}{V} & \frac{\partial \dot{\psi}}{r} \\ \frac{\partial \dot{N}_d}{V} & \frac{\partial \dot{N}_d}{r} \\ \frac{\partial \dot{E}_d}{V} & \frac{\partial \dot{E}_d}{r} \end{bmatrix}_{\hat{\mathbf{x}}(k),\mathbf{u}(k)} \tag{5.4.24}$$

$$\begin{aligned} \mathbf{\Gamma}(k) &\approx \mathbf{G_k} T_s \\ &= \begin{bmatrix} T_s \cos\psi & 0 \\ T_s \sin\psi & 0 \\ 0 & T_s \\ -T_s \cos\psi & 0 \\ -T_s \sin\psi & 0 \end{bmatrix}_{\hat{\mathbf{x}}(k),\mathbf{u}(k)} \end{aligned} \tag{5.4.25}$$

The continuous process noise covariance matrix needs to be converted to its discrete equivalent. As mentioned earlier the process noise is assumed to have its origin only in the noise on the input vector and noise due to unmodelled kinematics are neglected. The continuous inputs noise covariance matrix is therefore defined as,

$$\mathbf{Q_{uc}} = \begin{bmatrix} \sigma_V^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix} \tag{5.4.26}$$

where σ_V and σ_r denote the standard deviations on the forward velocity measurement from the encoders and yaw rate measurement from the rate gyroscope respectively. These standard deviations were determined by recording yaw rate data from the rate gyroscope and common mode velocity data from the encoders and then analyzing the sets to determine their standard deviations. The following values were found,

$$\begin{aligned}\sigma_V &= 0.02\, m/s \\ \sigma_r &= 0.496\, ^\circ/s\end{aligned}$$

This continuous input noise is then mapped onto the states of the system by making use of the continuous input matrix of the plant, $\mathbf{G_k}$, and then discretized, as shown in Equation 5.2.18, to yield $\mathbf{Q}(k)$.

As discussed earlier the measurement noise is assumed to be directly represented as discrete and the measurement noise covariance matrix is found as,

$$\mathbf{R}(k) = \begin{bmatrix} \sigma_N^2 & 0 & 0 & 0 \\ 0 & \sigma_E^2 & 0 & 0 \\ 0 & 0 & \sigma_{mx}^2 & 0 \\ 0 & 0 & 0 & \sigma_{my}^2 \end{bmatrix} \tag{5.4.27}$$

where σ_N and σ_E represent the standard deviations on the converted N and E measurements from the GPS. The standard deviations on the magnetometer measurements are denoted by σ_{mx} and σ_{my} respectively. With a method similar to the rate gyroscopes and encoders these values were found as,

$$\sigma_N = \sigma_E = 4\, m$$

$$\sigma_{mx} = \sigma_{my} = 0.02\, Gauss$$

After propagation of the previous best state estimate as well as the corresponding state covariances, the non-linear measurement matrix, $\mathbf{h}(\mathbf{x})$, is linearized about the propagated state vector, $\bar{\mathbf{x}}(k+1)$. From Equations 5.2.22 and 5.4.20 the linearized matrix is found, as shown in Equation 5.4.28. The updated filter gain is then found using Equation 5.2.23.

The final step involves an optimal update of the estimator states with Equation 5.2.24 and an update of the corresponding state error covariances matrix with Equation 5.2.25.

$$\mathbf{H}(k+1) = \begin{bmatrix} \frac{\partial h_1}{N} & \frac{\partial h_1}{E} & \frac{\partial h_1}{\psi} & \frac{\partial h_1}{N_d} & \frac{\partial h_1}{E_d} \\ \frac{\partial h_2}{N} & \frac{\partial h_2}{E} & \frac{\partial h_2}{\psi} & \frac{\partial h_2}{N_d} & \frac{\partial h_2}{E_d} \\ \frac{\partial h_3}{N} & \frac{\partial h_3}{E} & \frac{\partial h_3}{\psi} & \frac{\partial h_3}{N_d} & \frac{\partial h_3}{E_d} \\ \frac{\partial h_4}{N} & \frac{\partial h_4}{E} & \frac{\partial h_4}{\psi} & \frac{\partial h_4}{N_d} & \frac{\partial h_4}{E_d} \end{bmatrix}_{\bar{\mathbf{x}}(k+1)}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -B_N \sin\psi + B_E \cos\psi & 0 & 0 \\ 0 & 0 & -B_N \cos\psi - B_E \sin\psi & 0 & 0 \end{bmatrix}_{\bar{\mathbf{x}}(k+1)} \tag{5.4.28}$$

With all the necessary matrices and sensor data at hand a recursive extended Kalman Filter algorithm, as previously discussed, was implemented. The filter was implemented with an execution rate of 25 *Hz* with GPS measurement updates occurring at 4 *Hz*. Magnetometer measurement updates are available at much higher rates and the option therefore exists to update magnetometer measurements only, when GPS measurements are not available. It is however important to ensure the system is observable when implementing any form of estimator, prior to calculating the filter gain.

A brief investigation into this observability is therefore required to determine whether updating only the magnetometer measurements at a faster rate is justified. Consider the linearized measurement matrix when only magnetometer measurements are updated and not the GPS measurements. This implies a reduced measurement matrix where the 1 entries on the 1*st* row, 4*th* column and on the 2*nd* row, 5*th* column are forced to zero when no GPS measurements are available and the rest of the matrix remains unchanged.

To ensure observability the observability matrix, defined as,

$$\boldsymbol{\varphi} = \begin{bmatrix} \mathbf{H} \\ \mathbf{H\Phi} \\ \vdots \\ \mathbf{H\Phi}^{n-1} \end{bmatrix}$$

has to have full rank, n, where n is the number of rows in $\mathbf{\Phi}$. To investigate the observability matrix was setup with the reduced measurement matrix, $\mathbf{H}(k+1)$, and the linearized and discretized plant matrix, $\mathbf{\Phi}(k)$. For the test a control and state vector were chosen, with values which can be expected during operation of the UTV, in order to evaluate the matrices. These values were,

- $V = 0.45\, m/s$
- $\psi = \frac{pi}{4}\, rad$
- $\tau_p = 6.452$
- $T_s = 0.04\, s$
- $B_N = 9390.4 \times 10^{-5}$
- $B_E = -4136.6 \times 10^{-5}$

where the constant NE magnetic field components of the earth, at Stellenbosch, were used for B_N and B_E and the pole/zero location of the Padé approximation was determined from the 310 ms measurement delay of the GPS. The observability matrix is now calculated for both the non-reduced measurement matrix, with both GPS and magnetometer measurements available, as well as the reduced measurement matrix with only magnetometer measurements available. The rank of these two observability matrices were then calculated to yield the following,

- Non-reduced $\mathbf{H}(k+1)$ yields an observability matrix with full rank
- Reduced $\mathbf{H}(k+1)$ yields an observability matrix with incomplete rank

From this simple sample test it can therefore already be deduced that the system is not always observable when only the magnetometer measurements are updated. It was therefore decided to update both the GPS and magnetometer measurements at 4 Hz only. It should be noted that this test does not attempt to prove the observability of the non-reduced measurement matrix when both magnetometer and GPS measurements are updated. In that regard the observability is only confirmed for this specific sample with the chosen parameter values. The non-reduced measurement matrix did however perform effectively and several parameter values were used in this observability test which all yielded a confirmation of observability. A formal proof of observability for the non-reduced measurement matrix is therefore omitted but notice is taken that on stricter terms, with a more complex EKF, observability should be investigated vigorously.

Initialization of the state vector, during practical implementation, is done by taking the average of several successive GPS and magnetometer measurements, while the UTV is stationary. More information about the initialization procedure can be found in Appendix C.

5.5 Simulation

The simulator of the entire system is presented in Chapter 4 and shows how the state estimator is included in the simulation through use of a Simulink S-Function, which is an implementation of the discussions in this chapter. The performance of the state estimator is evaluated by making use of the UTV model in the simulation to generate a noiseless reference state vector. This vector is then corrupted by white noise and the appropriate delays are introduced on the GPS signals. The estimator is driven with these corrupted and delayed signals and comparisons are then made between the noiseless reference state vector and the estimated state vector. While evaluating the performance of the estimator the validity of the entire system is also evaluated under the influence of measurements corrupted with noise, since the estimated states are used for feedback in the controllers. This section presents the performance of the implemented EKF, along with the rest of the system, and ultimately the successful navigation of a path obtained from the path planning algorithms.

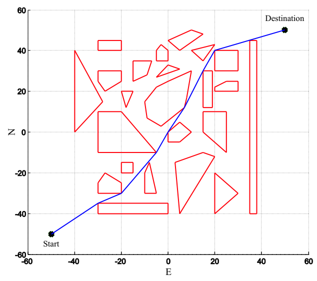


Figure 5.2: Path Obtained from a *Visibility population*

In order to gain insight, the entire system was simulated, with a set of obstacles as shown in Figure 5.2, where the path calculated by the path planning algorithms is also shown. Figure 5.3 shows the estimated states obtained from the EKF as well as the error between these estimated states and the noiseless reference state vector. Also shown in dashed lines are the 2σ error bounds which were obtained from the state error covariance matrix, $\mathbf{P}(k)$, which is calculated on the EKF throughout the simulation. The simulation was run for a duration of 350 s in order to monitor EKF performance throughout navigation of the entire *Visibility* path. As stated in [15], by definition 95 % of the state errors should fall within the 2σ bounds if a filter is completely linear but this does not hold for an EKF though since the non-linear transformation of a Guassian probability density function is no longer Guassian. In Figure 5.2 it can be seen that this stipulation does however indeed hold in this implementation.

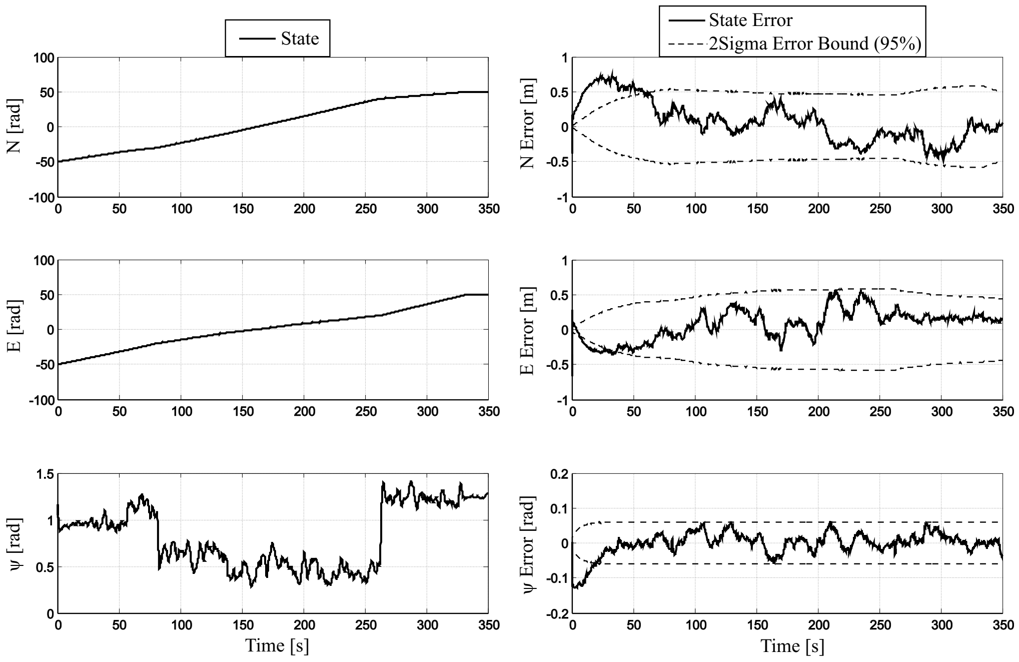


Figure 5.3: EKF States (left) and State Errors (right) - *Visibility* Path

To gain insight into the overall performance of the system and the kind of inaccuracies which can be expected from the EKF, the estimated path of the UTV is plotted versus the actual path obtained from the noise-free state vector. As seen in Figure 5.4 the actual path is very accurately estimated with only slight occasional deviations. This figure also represents the successful functioning of the control algorithms since the path traveled is clearly an accurate representation of the desired path from Figure 5.2.

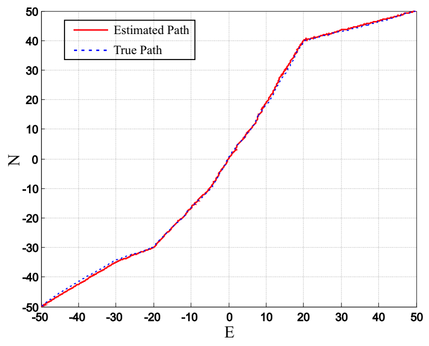


Figure 5.4: Estimated Path vs True Path during *Visibility* Graph Navigation

As seen in Chapter 3, there is a vast difference in the type of paths which can be expected from the two different *population* algorithms. Since the *Voronoi* algorithm usually yields paths which are less straight, with rigorous turning manoevres required, the same simulation procedure was also followed for the obstacles and path shown in Figure 5.5. This path is obtained from the *Voronoi* algorithm and shows the jagged edges characteristics of the paths this algorithm yields.

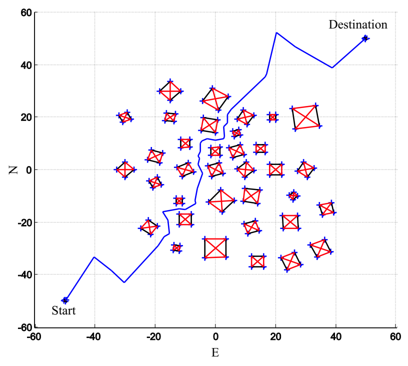


Figure 5.5: Path Obtained from a *Voronoi population*

As seen in Figure 5.6 the EKF again yields satisfactory results with positional errors in the order of less than 0.5 m. The different characteristics of this path is also evident in the higher contrasts of ψ measurements.

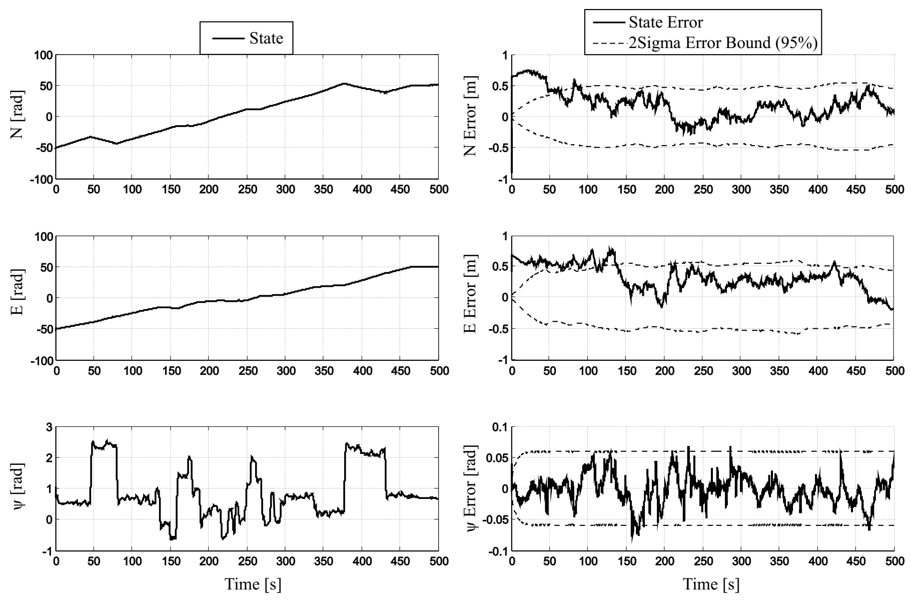


Figure 5.6: EKF States (left) and State Errors (right) - *Voronoi* Path

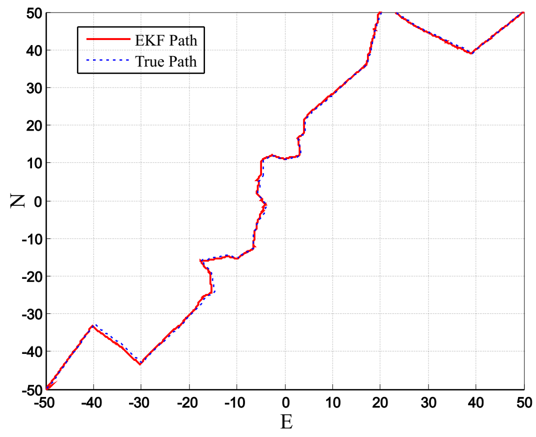


Figure 5.7: Estimated Path vs True Path during *Voronoi* Navigation

Figure 5.8 shows a zoomed version of the East error state in Figure 5.6. In this figure it can be seen how the error covariance grows in between GPS measurements, at 4 Hz, and is continuously pulled back toward zero when GPS measurements become available on the EKF. As mentioned in Chapter 2, the estimator relies heavily on propagation of the encoder, $\mathbf{V}$, and rate gyroscope, $\mathbf{r}$, measurements and is not sensitive to GPS update measurements over short distances. This is evident in this chapter when comparing the noise variance figures of the encoders and yaw rate gyroscopes with the variances of the GPS measurements.

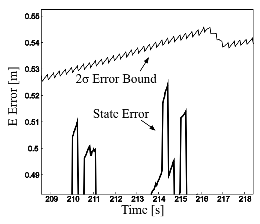


Figure 5.8: EKF Estimated East Position State Error

Although the noise variances can not be compared directly, due to them being quantified in terms of different units, it is still evident that a significant duration of propagation is required before variances on positional states, obtained from propagation, become as large as the variance of 4 m on the GPS measurements. Stated in different terms, the certainty of propagation is higher than the certainty of the measurements, and consequently the EKF "trusts" the propagation process more. Although indefinite indoor use of the UTV will not be possible, due to the accumulating effect of errors during integration (propagation), it could be argued that the UTV will function well enough without GPS updates, given the time of operation and area of operation is relatively small. This was confirmed during test runs since the EKF still functioned relatively well when no GPS fix and hence no GPS measurements were available. The UTV is therefore capable of outdoor operation with occasional GPS blackouts occurring. This creates the possibility for simultaneous indoor and outdoor usage, provided the UTV does not operate within a building for too long before exiting it again in such a fashion that a GPS fix is once again available.

A final note should be made about rate gyroscope biases. This EKF was simulated under the assumption that gyroscope biases can be ignored since temperature variations are compensated for, as discussed in Appendix C. Biases can however have deteriorating effects on performance and should be kept in mind and modeled when unexpected results are obtained during practical implementation. Within the scope of this project, where only one rate gyroscope is used, biases proved to have little or no effects during test runs and are therefore left unmodelled.

5.6 Summary

In this chapter, the implementation of a two-dimensional Extended Kalman Filter was presented. An overview of the estimator kinematics was shown and the structuring of the kinematics within the measurement platform of the project was introduced. Next the optimal estimation theory used for the implementation was briefly summarized. A discussion was given on the NE reference frame used and the application of the optimal estimation theory was presented. Finally simulation results and corresponding discussions were given.

Chapter 6

Test Runs

This chapter features a presentation of the results obtained from practical tests. An overview of the tests which were performed is given, followed by a discussion on the results of each day's tests and finally an analysis is provided of the telemetry obtained during the navigation of two paths, one calculated by the *Visibility population* algorithm and the other by the *Voronoi population* algorithm. Both these *population* algorithms were tested in conjunction with the *A*Star shortest path* algorithm, since it was established in Chapter 3 that this is the optimal choice, as far as computational cost goes, and yields exactly the same results as use of the *Dijkstra shortest path* algorithm.

6.1 Overview

Test runs were conducted on a sports recreational field (rugby field), on Sunday the 29th of June 2008, as well as on Monday the 30th of June 2008. All systems of the UTV were tested within a practical obstacle environment to confirm the UTV's ability to navigate itself from a starting position to a destination, autonomously (with the exception of initialization procedures on the ground station GUI) while avoiding all represented obstacles on the rugby field.

It should be noted that these tests were performed under adverse circumstances. On the 18th of May 2008, the uncompiled code of the ground station GUI and OBC user control application (UCA) was lost in a fire caused by a faulty Lithium-Polymer battery. Only the versions of these two software modules which were already loaded onto the OBC and ground station PC, at the time, were therefore available for test runs, and improvements could not be made on the practical implementations. Relatively good results were however still obtained with the exception of minor anomalies which will be addressed and appropriate solutions provided. Estimated telemetry was not directly available on these preliminary implementations but an alternative representation of the estimated telemetry is available on the ground station GUI as will be seen shortly. To supplement the limited telemetry which is available, video recordings are presented on the DVD included with this thesis. The reader is strongly urged to view these videos since it provides an effective interactive medium through which the results of practical tests can be evaluated. It should be noted that all videos referenced from this chapter can be found in the *practical demonstrations videos* folder on the DVD.

6.1.1 Sunday the 29th of June 2008

On Saturday the first step was a definition of a set of obstacles for the test runs. The positional coordinates of these obstacles' vertices, within the NE reference frame discussed in Chapter 5, were then recorded for use during the tasks of the following day. On Sunday morning all equipment was taken to the test site. The first practical step was a definition of an origin for the NE axis system. The origin was marked by means of a threaded rod. The N-axis and E-axis were then represented with red and white hazard tape through use of an accurate compass and this marked position as origin.

Figure 6.1: Setting up the obstacles

The next task was accurately locating all the defined obstacle vertices, through use of a 50 m measuring tape and the compass, and marking each obstacle vertex with another threaded rod. Hazard tape was wrapped around the four vertices of each obstacle to represent the edges (sides) of each obstacle clearly. Only square obstacles were defined to avoid complexity during this process.

With a complete representation of the obstacles in place, the attention was shifted to the actual task of navigation. The ground station laptop computer was set up, a video camera was prepared and the UTV was placed at the starting location on the rugby field and its power switched on. From the ground station the OBC executable was started, screen capturing software initialized, and all necessary initialization procedures were performed on the ground station GUI. These initialization procedures are discussed in Appendix C and can be seen on the DVD discussed in Appendix D. After initialization the estimator was armed and a green dot on the virtual NE axis system, on the ground station GUI, immediately indicated the UTV's current position with good accuracy. For the first test the *Voronoi population* algorithm was selected on the ground station GUI. The autopilot was armed, a *Voronoi* path calculated on board the UTV, and the coordinates of the line segments of this path were almost instantaneously communicated back to the ground station, and displayed on the GUI. This confirms the correct functioning of the path planning module and the insignificant computational load on the OBC, due to a combination of the *Voronoi population* and *A*Star shortest path* algorithms.

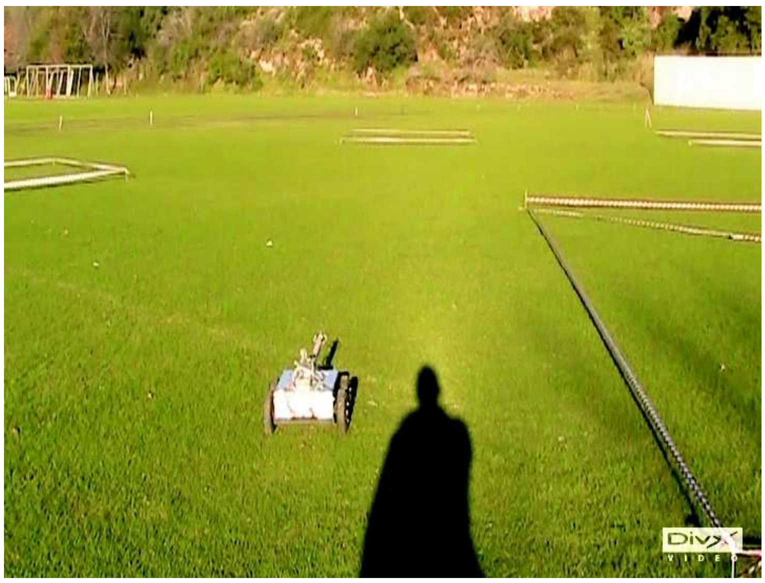


Figure 6.2: UTV Navigation in Progress

After this the focal point was shifted to the control algorithms. Immediately after the path was calculated the UTV started turning, on its current location, to aim itself in the direction of the first line segment's heading. This heading was achieved and tracking of the first path line segment commenced. During the navigation of the first straight line segment the correct functioning of the guidance controller was confirmed.

Throughout the navigation the UTV stopped momentarily at the end of each line segment and then turned to the heading of the next line segment. It should be noted however that an anomaly was observed during the transition from one of the line segments to the next. As stated in Chapter 4, all heading angles are conditioned so as to prevent full rotations of the UTV during transitions. The OBC executable which was loaded onto the UTV at the time did not yet have this conditioning implemented and therefore displayed such a full revolution during navigation. To confirm this was indeed the cause, the exact same obstacle environment was simulated after the day's tests to try and duplicate the anomaly in simulation. The full revolution was indeed observed on the *OpenGL* graphical simulator. It is at this point that the conditioning routines were implemented in simulation and after implementation the simulation executed flawlessly. These conditioning routines are shown in Appendix A. The anomaly is also indicated at 2 minutes into the video, labeled *1st*, in the *DAY ONE* folder on the DVD. Navigation did however continue successfully after the anomaly and the UTV reached its destination to within an accuracy of 1.5m on the scale of a 50 by 50 m rugby field.

The effectiveness of the state estimator was shown by the green dot which is continuously superimposed, in real-time, on the virtual NE axis system on the ground station GUI, in accordance with the estimated positional state vector. This green dot accurately traced out the calculated path on the ground station GUI with occasional minor deviations, especially during the full rotation anomaly just discussed. A vast improvement from the raw GPS positional measurements to the estimated positional states is evident as will be seen in the *Telemetry Analysis* section to follow shortly. It will additionally be shown that the estimated path indeed represents the actual path traveled by the UTV.

The next test was dedicated to the results of a *Visibility population* and *A*Star shortest path* combination. To ensure obstacle avoidance, with this *population* algorithm being used, all square obstacles were virtually increased in size with the addition of 2 m to the length of the sides of each square obstacle. Similarly to the *Voronoi* case all systems functioned correctly and during this test run no full revolution anomalies were displayed. The computational load was slightly increased on the OBC but to no significant extent and navigation also commenced shortly after the path was displayed on the ground station GUI. The longer straight line characteristics, with minimal changes in heading throughout the navigation, due to the *Visibility population* method being used and a path approximating a straight line being available from the specific obstacle environment, was clearly evident. Successful obstacle avoidance was still achieved since all obstacles passed to the path planning module were virtually larger than their actual sizes and this can be seen in the video, labeled *2nd*, in the *DAY ONE* folder.

The focus was then shifted to a test run during which an additional obstacle is uploaded in real-time during navigation, to simulate the process of obstacle detection. This test was performed with the use of the *Visibility population* algorithm. The new obstacle was uploaded while the UTV was in close vicinity of the obstacle to be added and therefore realistically represents the process of obstacle detection. It should be noted that the additional obstacle was not physically represented on the rugby field but merely virtually added to test the UTV's response. This test was partially successful and can be seen in the video, labeled *4th*, in the *DAY ONE* folder. A minor anomaly should however be mentioned. The version of the OBC code, loaded onto the UTV at the time, was not yet correctly configured to use the UTV's current position as the starting point for the path planning algorithms in the case of a new path planning cycle being scheduled. The new path being calculated is therefore once again a path from the original starting point to the destination, with the new obstacle avoided. It is however shown in the video that relatively good results were still obtained. Instead of the UTV commencing navigation from a new starting point, the UTV now sees the change in path, to avoid the obstacle in its close vicinity, as a sudden track error and the guidance controller forces the UTV onto the new line segment. Although this is not in accordance with the intended operation of the algorithms and the author does not claim the unconditional validity of this, it still gives an indication of the robustness of the guidance controller. Test runs for Sunday the 29th of June 2008 were then concluded.

6.1.2 Monday the 30th of June 2008

Upon arrival at the test site the following day, a disappointing discovery was made. All the threaded rods and hazard tape which represented the obstacles had been removed. With the successes from the previous day a decision was therefore made to spend the day testing other aspects of the system. The first test was based on testing the UTV's ability to navigate a *Visibility* path to the destination and then navigate its way back to the starting point, all with the stipulation of no human interference except for configurations on the ground station GUI. The navigation to the destination was conducted in the same manner as the tests of the previous days. However, once the UTV reached its destination the OBC executable was not stopped. Instead the autopilot was disarmed as well as the estimator. The start and destination positions were then exchanged on the ground station GUI and uploaded to the UTV by means of RF communication. The estimator was once again initialized and the autopilot armed. Once again the calculated path was communicated back to the ground station and the UTV started navigating itself toward the original starting position. After the navigation the UTV stopped within reasonable distance of the new destination (the previous point of departure) but with decreased accuracy. This can be attributed to the fact that the UTV was not precisely at the original destination when the estimator was initialized. A certain offset is therefore imposed which causes degraded accuracy during navigation back to the original departure point. Relatively good results were still obtained which can be seen in the video, labeled *1st*, in the *DAY TWO* folder. This test therefore confirms the continuity of the system.

The next test was dedicated to monitoring the estimator's performance in real-time during navigation. This was done by initializing a *Voronoi* navigation process and then switching to the *Estimator* tab on the ground station GUI during navigation. The test is seen in the video, labeled *2nd*, in the *DAY TWO* folder and shows the correct functioning of the estimator. A diagram which indicates the UTV's heading in real-time can also be seen. It should be noted that a full revolution anomaly was also recorded during this *Voronoi* test, for the same reasons stated previously.

The day was concluded by demonstrating manual control of the UTV from the ground station GUI, seen in the video labeled *3rd*, in the *DAY TWO* folder on the DVD. The remainder of this chapter is dedicated to an analysis of the telemetry results obtained from a *Visibility* and *Voronoi* navigation respectively.

6.2 Telemetry Analysis

This section provides a presentation of telemetry results obtained during two test runs. For the reasons previously mentioned direct estimator telemetry is not available. An effective substitute is however available which is a screen capture of the ground station GUI after the completion of each path. This diagram, updated in real-time on the GUI, makes use of the estimator positional states to plot a green dot in a NE reference frame during navigation. The green dot represents the current estimated position of the UTV while the estimated path traveled thus far is traced out with a cyan coloured line. This path is superimposed on the NE diagram on the ground station GUI which also displays the calculated path plotted in blue.

6.2.1 Navigation of a *Visibility* path

The telemetry presented in this section was obtained during the test run shown in the video, labeled *1st*, in the *DAY TWO* folder on the DVD. As seen this test run was conducted without actual representations of the obstacles on the rugby field. The actual path followed by the UTV in practice is therefore not easily visible for comparison with state estimator and GPS positional states. This telemetry section therefore focuses on the the heading angle measurements and the performance of the guidance controller while the next section will focus on a comparison between the performances of the state estimator and GPS. Consider the path traced out on the ground station GUI in accordance with the estimated positional states, shown in Figure 6.3.

Next, consider the raw magnetometer heading angle measurements obtained during navigation, shown in Figure 6.4. These heading angles were calculated through use of Equation 5.4.21. In this figure the three distinct heading angles of the calculated path can be observed. The line segments of this path are shown in Table 6.1 for clarity. A distinct observation can be made when considering Figure 6.4. As shown, a sinusoidal like signal is superimposed on the the three distinct steps in heading angle. This indeed represents the continuous attempts of the guidance controller to achieve a zero track error. It is noticeable how these oscillations are relatively large in amplitude which suggests lateral guidance controller gains which are too high. These gains could therefore be iterated upon and the system *tuned* in order to obtain an improved response.

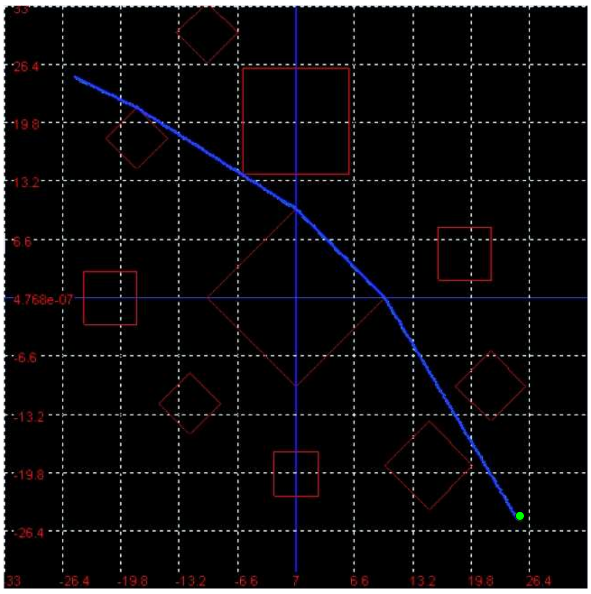


Figure 6.3: Estimator positional states accurately trace out the *Visibility* path in real-time on the ground station

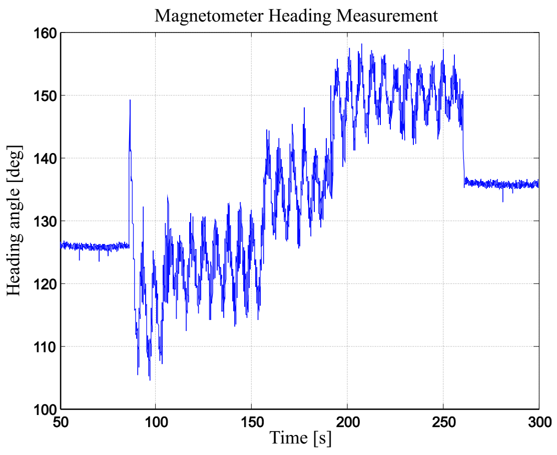


Figure 6.4: Magnetometer Heading during navigation of a *Visibility* path

North Start	East Start	North End	East End	Heading Angle	Length
25.10 *m*	-24.83 *m*	21.54 *m*	-18.00 *m*	2.05 *rad*	7.70 *m*
21.54 *m*	-18.00 *m*	9.90 *m*	0.00 *m*	2.14 *rad*	21.43 *m*
9.90 *m*	0.00 *m*	0.00 *m*	9.90 *m*	2.36 *rad*	14.00 *m*
0.00 *m*	9.90 *m*	-25.00 *m*	25.00 *m*	2.60 *rad*	29.21 *m*

Table 6.1: Path segments of practical *Visibility* path

6.2.2 Navigation of a *Voronoi* path

To observe the insufficient accuracy of the GPS measurements alone consider Figure 6.5. In this figure it is seen how the raw GPS measurements deviate quite significantly from the calculated path. In fact, the accuracy of these GPS measurements are even worse than expected and could possibly be attributed to a poor GPS fix being available at the time.

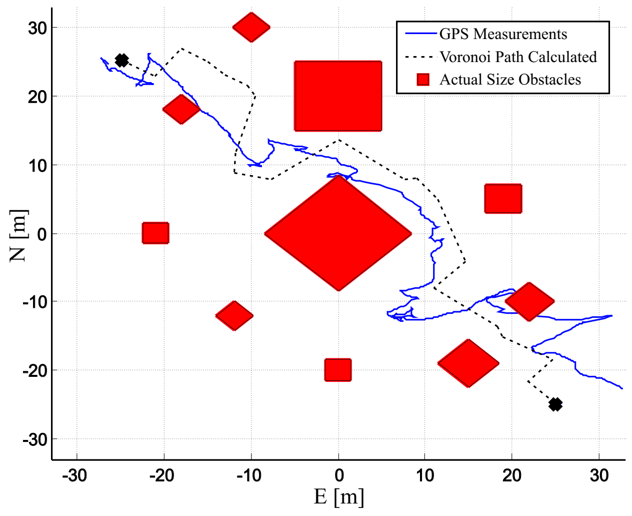


Figure 6.5: GPS Measurements during navigation of a *Voronoi* path

Again consider the estimated positional states traced out in cyan on the ground station GUI, shown in Figure 6.6. It is seen that the estimated path closely resembles the calculated path. Now refer to the video, labelled *1st*, in the *DAY ONE* folder on the DVD, at exactly 2 minutes and 56 seconds. In this video the position of the path, indicated in Figure 6.6, is clearly represented by the actual UTV at this stage during its navigation. When comparing this portion of the path with the non-existence of that specific portion, in the GPS path of Figure 6.5, it is confirmed that the estimator indeed accurately represents the UTV's actual position more so than the GPS. The insensitivity of the estimator to GPS measurements is therefore demonstrated and it is seen that the estimator still performs relatively well under the circumstances of degraded GPS accuracy. Stated in different terms, the certainty of propagation of yaw rate and encoder measurements are higher and consequently weighed more heavily by the estimator.

As previously mentioned all angles are not conditioned properly on the version of the OBC executable, loaded at the time, and consequently full revolu-

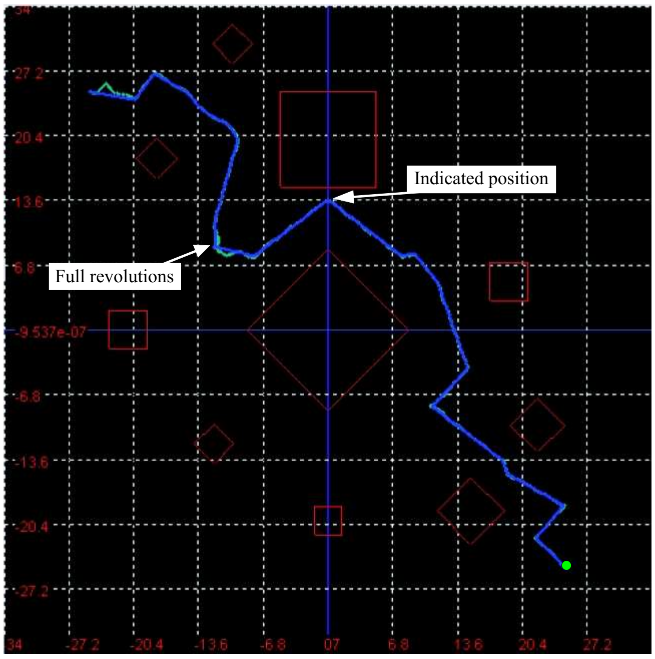


Figure 6.6: Estimator positional states accurately trace out the *Voronoi* path in real-time on the ground station

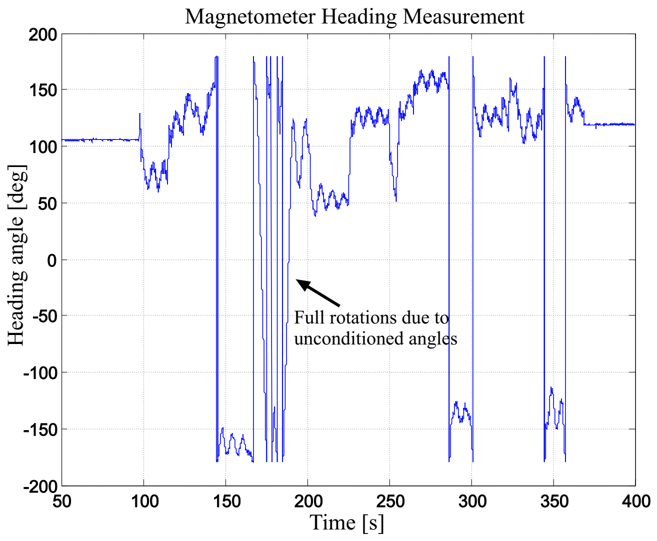


Figure 6.7: Magnetometer Heading Measurements during navigation of a *Voronoi* path

tions occur. The point at which this occurs can be seen in Figure 6.6. This is also evident in the magnetometer heading measurements, shown in Figure 6.7. In Figure 6.7 the cause of this revolution is clearly illustrated at approximately 140 *s* where the four quadrant inverse tan function causes an instantaneous jump from an angle above a 180° to its duplicate negative mapping of more or less -175°. It is therefore purely a numerical problem which was easily solved in simulation by conditioning all angles, as stated in Chapter 4. The flowchart of these conditioning routines can be seen in Appendix A.

6.3 Summary

This chapter featured a presentation of the practical results obtained during test runs conducted with the fully integrated UTV system. It was shown that even with immature practical implementations good results were still obtained which can largely be attributed to the vigorous exhaustion of simulation resources before attempting test runs. Anomalies that were encountered were discussed and addressed. Limited telemetry results were available and reference was therefore made to video recordings, included with this thesis, where necessary.

Chapter 7

Summary and Recommendations

7.1 Summary

This thesis reported the work done on the development of an autonomous unmanned terrestrial vehicle, operating in an obstacle ridden environment. The project was the first to focus on the automation of a ground vehicle at the Electronic Systems Laboratory, at the University of Stellenbosch.

The preparation of a UTV hardware platform was presented. It was shown how a previously developed modular architecture, which makes use of the CAN protocol, can be utilized and modified to serve the purpose of integrating all systems required for autonomous navigation of a UTV. The hardware was introduced from the lowest level of drive systems, which interface with two direct current motors, to the highest level of an on board computer which interfaces with a ground station GUI via a RF communication link, while serving as main control node for all systems on board the UTV.

With a hardware platform in place attention was shifted to the software implementation of path planning algorithms which yield optimal paths through a two-dimensional obstacle ridden environment. The first step was an investigation into algorithms which populate the obstacle-free space of the environment with straight line segments connected at specified nodes. Two such algorithms were investigated and it was determined that the *Visibility* graph is a more stable and appropriate solution than the implemented *Voronoi population* method. The next step required the investigation into algorithms which filter out the shortest path, consisting of a combination of the line segments from these *populations*. It was established that in this type of application, where the destination of the path is defined, the *A*Star* search is optimal as far as computational cost and stability goes. Both *population* algorithms and both *shortest path* algorithms were implemented and successfully demonstrated. Four closed modules were then available for use by a path planning module.

The focal point was then shifted to the development of controllers which translate the data obtained from the path planning module into reference commands which ultimately determine the wheel speeds on each side of the UTV. It was shown how the dynamics of lower level controllers can be encapsulated through a process of successive loop closures until only reference command inputs are required for a guidance controller. The hierarchy of controllers were then integrated into a controller module which interfaces with the path planning module via a state-machine. It was shown that through use of this state-machine an obstacle detection unit is accommodated in future projects.

A simplified two-dimensional state estimator was then introduced. It was shown how all required states of the UTV are accurately estimated through the use of an extended Kalman Filter which combines the measurements from low cost sensors in an optimal manner. It was further shown how the estimator displays an insensitivity to GPS measurements and is able to function well without GPS measurements for a limited duration of operation.

Through use of a non-linear simulator, which makes use of a developed dynamic model of the UTV, the path planning and controller modules were evaluated in conjunction with the state estimator. After numerous simulations the desired results were obtained. A ground station GUI was then developed to accommodate all initialization procedures and interactions which are required with the UTV and its on board modules. At the same time the controller and path planning modules were implemented on the UTV platform. Finally, practical tests were conducted and it was shown that relatively good results were obtained. This rapid success during test runs can largely be attributed to a thorough process of development within simulation, prior to implementation on the actual hardware platform.

7.2 Recommendations

Recommendations on how the UTV can be improved and extended are now given,

7.2.1 *Trapezoidal Map*

As noted in the discussions of Chapter 3 the *Voronoi* implementation in this project yields occasional collision paths and at times no path at all. An attractive alternative to using a *Voronoi population* is the *Trapezoidal Map*. This section will give a brief description of how this algorithm can be used for path planning purposes. For additional information refer to [9]. The basic principle of a *Trapezoidal Map* is shown in Figure 7.1.

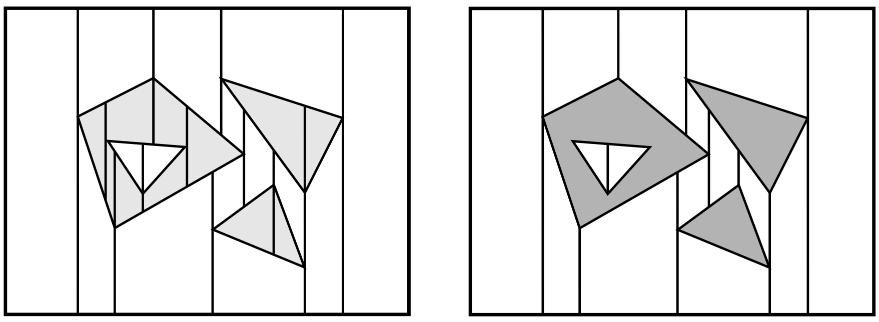


Figure 7.1: Construction of the *Trapezoidal Map* [9]

The method consists of first defining a bounding box around the obstacle environment. Vertical extensions are then drawn from each vertex of the obstacles, represented by polygons, upward until another obstacle edge or the bounding box is encountered. Extensions are also drawn downward in the same manner. This then yields a subdivision as shown on the left side of Figure 7.1. It becomes clear why the subdivision is called a *Trapezoidal Map*. As shown in this figure, the *faces* represented by the space between these vertical extensions, with the horizontal boundaries being the edges of the obstacles, or the bounding box, represent trapezoidal-like polygons, where some are triangles which is in fact a degenerate trapezoid with one edge of zero length. By making use of these vertical extensions a *population* can be constructed in the obstacle free space of the environment. Consider the diagram on the right in Figure 7.1. In this diagram all *Trapezoidal Map* edges which lie in the interior of obstacles have been removed. This is easily done when considering that the algorithm presented in [9] provides information about the edge bounding each trapezoid from the top. A simple check therefore has to be done on whether this same edge bounds the obstacle from the top or from the bottom. If it bounds the obstacle from the top it is known that the vertical extensions of this trapezoid, also bound from the top by this edge, are in the interior of an obstacle and need to be removed. For more information refer to [9].

With the remaining *Trapezoidal Map*, representing only vertical extension in the obstacle free space, a *population* can now be calculated which will yield yield a *roadmap* with similar characteristics to the *Voronoi* implementation of this project but without the degenerate cases where collisions are implied or no connection found between start and destination. It achieves this with the only stipulation being that the start and destination are both not unrealistically defined within the interior of an obstacle, and are located in the obstacle free space.

In Figure 7.2 it is shown how a node was placed in the centre of each of the remaining trapezoids, after removing vertical extensions inside obstacles, as well as in the middle of each vertical extension. The method for finding a *populated set* now simply requires a connection between each node in the middle of a trapezoid and the nodes surrounding it on the edges of that specific trapezoid. After this has been done a population is yielded as shown in Figure 7.2. All that now remains is simply connecting the start and the destination to the node in the centre of the respective trapezoids, in the obstacle-free space, in which the start and destination reside.

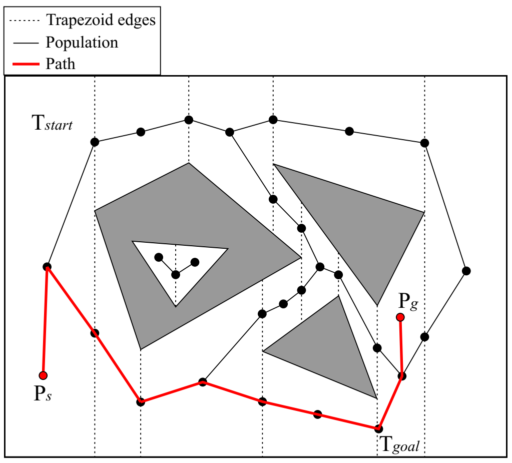


Figure 7.2: Path found from a *Trapezoidal Map* [9]

It is shown in [9] that an implementation of this algorithm will always find a collision free path for a point vehicle, if one exists. It therefore suffices to say that this algorithm is a better option than the implementation of the *Voronoi population* presented in this project since it also yields paths with maximum clearance from obstacles, due to the nodes being placed exactly in the middle of the edges of the *Trapezoidal Map* in free space. In fact the *population* closely resembles the *Voronoi populations* presented in Chapter 3, with the added advantage that collision free paths are guaranteed and convex obstacles are also accommodated. Furthermore, in [9], it is shown that this algorithm can be implemented with a $O(n \log n)$ complexity which is exactly the same as the *Voronoi* diagram's complexity, with the further advantage that pruning is not required afterward as was the case with the *Voronoi population* method presented in this thesis. It is therefore recommended that either this algorithm be investigated further or an alternative implementation of the *Voronoi* algorithm be found for path planning purposes.

7.2.2 *Visibility-Voronoi Complex*

It suffices to mention one such possible alternative implementation of the *Voronoi* algorithm. In fact, not a purely *Voronoi* orientated implementation but a hybrid implementation referred to as the *Visibility-Voronoi-Complex*. This *population* method has received significant attention recently. The method is based on a hybrid implementation of the *Voronoi* diagram and *Visibility* graph where a variation in a certain parameter varies the output of the algorithm between a pure *Voronoi* diagram and a pure *Visibility* graph. The algorithm seems very attractive since with an optimal choice of this parameter an optimally short smooth path with maximum clearance from obstacles can be found which combines the strengths of both *population* methods. This algorithm could possibly outweigh any of the algorithms discussed thus far, as far as optimal paths go, and more information can be found at [23].

7.2.3 *Minkowski Sum*

It was shown in Chapter 3 that a *Visibility population* yields insufficient clearances from obstacles. A pre-processing algorithm, known as the *Minkowski Sum*, exists which can be used in conjunction with the *Visibility* graph to yield an optimal path with sufficient obstacle clearances. This algorithm can potentially be used in conjunction with any *population* method and any *shortest path* algorithm. It is not an algorithm specifically involved in the process of determining an optimal path through the obstacle environment but rather an algorithm which *pre-processes* the original obstacle environment, referred to as the *work space*, to yield a modified obstacle environment, the *configuration space* [9]. The *pre-processing* algorithm makes use of the *Minkowski Sum* between the polygon representing the vehicle or object, which path planning is done for, and the original obstacle environment. In doing this the obstacle shapes are virtually modified and marginally increased in size to ensure that any *population* method used on this virtual *configuration space*, even the *Visibility* graph with minimal clearances, will yield a *population* which only represents paths in the obstacle free space and ensures that collisions will never occur. This algorithm is strongly recommended, especially in the case of the *Visibility* graph being used as *population* method. To gain a general idea of the approach refer to Figure 7.3.

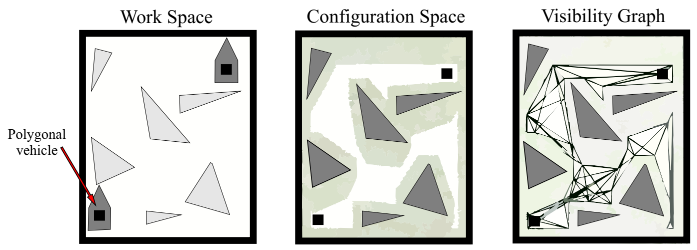


Figure 7.3: Calculating the *Configuration Space* from the *Work Space* [9]

7.2.4 Rate Gyroscope and Encoder Combination

A further improvement is recommended which will aid in reducing the effects of possible rate gyroscope biases. As discussed in Chapter 4, a slip gain is introduced, during simulation, which adjusts the theoretical yaw rate of the UTV (obtained from a linear expression in terms of the respective left and right wheel speeds) to more closely resemble actual recorded yaw rates of the UTV at certain wheel speeds. There is however still a level of uncertainty when yaw rates are calculated from the wheel speeds since terrains on which the UTV operates differ. A valuable observation can however be made. Although the wheel speed measurements from the encoders do not provide a high level of certainty when measuring non-zero yaw rates they do indeed represent an exact accurate measurement of zero yaw rates. An improved implementation of yaw rate measurements would therefore be a combination of encoder and rate gyroscope measurements where yaw rates are regarded as zero when indicated as zero by the encoders and simply taken as the rate gyroscope measurement when the encoders do not yield zero yaw rates. In doing this the effects of bias drifts on the rate gyroscope during zero yaw rates are eliminated.

7.2.5 Reducing Computational Cost

As seen in Chapter 3, all algorithms were not implemented in an optimal manner. Computational cost can therefore be reduced through optimal implementations.

7.2.6 Future Implementations of Obstacle Detection

It was shown in this project how the process of obstacle detection is simulated by setting a flag which triggers a new path planning cycle. In future applications where obstacle detection is added the system can be initialized by defining a preliminary path which simply consists of a straight line between start and destination. As the UTV then progresses along this straight line obstacles will or will not be detected. As soon as an obstacle is detected, all which is required is that the *new path scheduled* flag be set and the obstacle added to a data structure which stores the current set of known obstacles in the environment. This will then bring the UTV to an immediate halt during which a new path is calculated, which avoids this obstacle and all previously stored obstacles, after which navigation continues.

7.2.7 Extensions to a 3D Environment

Although there is a vast difference between the control of a UAV in three-dimensional space and the control of a UTV in two-dimensional space, it was shown in this project how the same hardware platform, used by the UAV Research Group, is easily utilized for the automation of a UTV. Through use of such modular hardware platforms an environment is created where a UTV group can possibly serve as preliminary testing platform for ideas which could be extended to the three-dimensional problem of path planning for UAV's. As stated in Chapter 3, the path planning module can be regarded as a closed module which requires a certain set of inputs to yield a specific set of outputs. With the addition of certain *path-smoothing* techniques, and the implementation of an *A*Star heuristic* function which introduces penalties on paths which imply large heading angle changes, the path planning module could therefore be extended to accommodate way-point navigation for a UAV in an elevated two-dimensional plane. It should further be noted that a *Visibility* graph and *A*Star shortest path* combination can be implemented in three-dimensional space, at the cost of increased computational cost, and therefore creates the possibility of three-dimensional path planning for use in the UAV Research Group.

Appendices

Appendix A

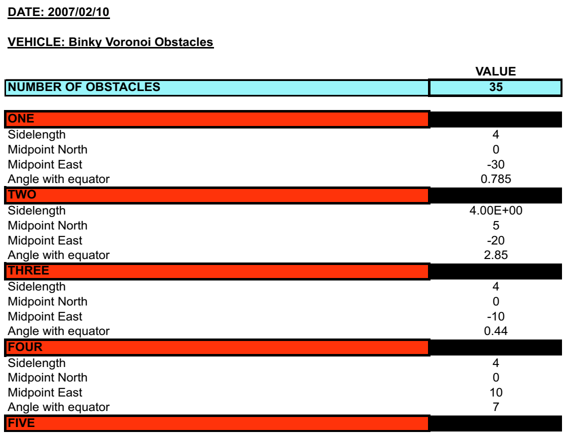
Hardware and Software Details

This appendix features details specific to the hardware of Chapter 2 and software of Chapter 4 and Appendix C.

A.1 Data Files

Since this project does not include obstacle detection, obstacles have to be defined prior to simulation. These obstacles are defined in a *Microsoft Excel* spreadsheet, imported into MATLAB and also written to a *.dat* file during simulation. The *.dat* file is then copied to a folder named *C:\Vehicle* on the computer which is used for the simulation. This *Vehicle* folder is where the finalized OBC executable (written in C code) resides and the vehicle and estimator specifications file is also written to this folder during simulation. During use of the ground station GUI the folder needs to be present on the ground station PC and located in the *C:* root. For convenience the *.dat* obstacle files, which are written during simulation, can be imported on the ground station GUI during UTV operation. The reader is referred to the DVD included with this document for a demonstration.

As mentioned in Appendix C, the two obstacle files are defined in different formats for the respective *population* algorithms. This is done to provide a convenient way of defining square obstacles in the case of the *Voronoi population* algorithm. The format in which both obstacle files are defined in an *Excel* spreadsheet is shown in Figures A.1 and A.2 respectively. In Figure A.1 it is shown how the square *Voronoi* obstacle coordinates are defined in terms of side length, centre location and angle with the horizontal reference line. Figure A.2 on the other hand shows how the vertex locations of convex polygons are defined as obstacles for the *Visibility* algorithm. These obstacles are separated by the number 1000000 since the amount of vertices of each obstacle can vary and when extracting the data the end of each obstacle needs to be indicated. The number 1000000 is used since it falls well outside the boundaries of expected coordinates within the NE reference frame discussed in Chapter 5.

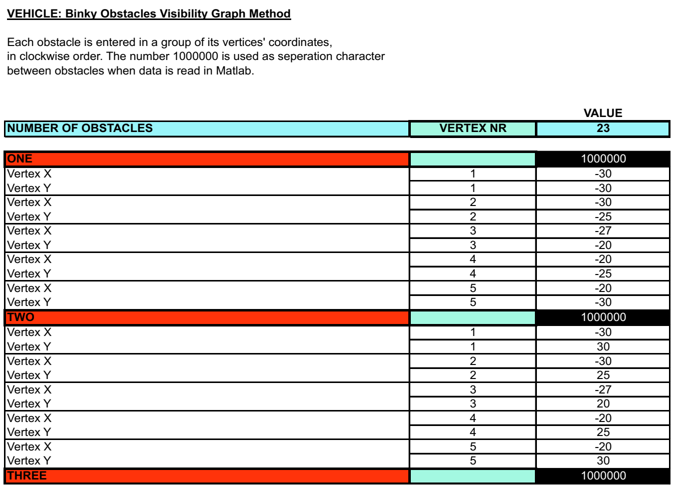


DATE: 2007/02/10

VEHICLE: Binky Voronoi Obstacles

	VALUE
NUMBER OF OBSTACLES	35
ONE	
Sidelength	4
Midpoint North	0
Midpoint East	-30
Angle with equator	0.785
TWO	
Sidelength	4.00E+00
Midpoint North	5
Midpoint East	-20
Angle with equator	2.85
THREE	
Sidelength	4
Midpoint North	0
Midpoint East	-10
Angle with equator	0.44
FOUR	
Sidelength	4
Midpoint North	0
Midpoint East	10
Angle with equator	7
FIVE	

Figure A.1: Format of *Voronoi* Obstacles Spreadsheet



VEHICLE: Binky Obstacles Visibility Graph Method

Each obstacle is entered in a group of its vertices' coordinates,
in clockwise order. The number 1000000 is used as seperation character
between obstacles when data is read in Matlab.

		VALUE
NUMBER OF OBSTACLES	VERTEX NR	23
ONE		1000000
Vertex X	1	-30
Vertex Y	1	-30
Vertex X	2	-30
Vertex Y	2	-25
Vertex X	3	-27
Vertex Y	3	-20
Vertex X	4	-20
Vertex Y	4	-25
Vertex X	5	-20
Vertex Y	5	-30
TWO		1000000
Vertex X	1	-30
Vertex Y	1	30
Vertex X	2	-30
Vertex Y	2	25
Vertex X	3	-27
Vertex Y	3	20
Vertex X	4	-20
Vertex Y	4	25
Vertex X	5	-20
Vertex Y	5	30
THREE		1000000

Figure A.2: Format of *Visibility* Obstacles Spreadsheet

The physical parameters of the vehicle are also defined in a *Microsoft Excel* file and then extracted for use in simulation, as well as written (along with estimator and natural constant parameters) to a *.dat* file in the *Vehicle* folder previously mentioned. This *.dat* file is uploaded to the OBC as discussed in Appendix C.

A.2 Communication

A.2.1 Serial Communication Protocol

The serial communication between ground station and RF module on board the UTV is established through use of a protocol introduced by [4]. This protocol ensures data integrity during transfers and functions well in this project under the load of up to 30 square obstacles' coordinates being uploaded sequentially while idle ground station operations are in progress.

A.2.2 Ground Station Telemetry

Description	**Bytes**
Time stamp	4
Rate gyroscope measurements	6
Accelerometer measurements	6
Magnetometer measurements	6
TOTAL	22
Percentage of bandwidth	4.6 %

Table A.1: Contents of Primary Packet

Description	**Bytes**
Latitude	4
Longitude	4
Altitude	2
Heading	2
NED velocities	6
Fix and differential GPS indicator	1
TOTAL	19
Percentage of bandwidth	4 %

Table A.2: Contents of GPS Packet

Similarly to the discussion in [15] the packets which are transmitted from the UTV to the ground station, during navigation, are analyzed. These packets are sent sequentially at a rate of 2 *Hz* and a limit therefore exists on how many bytes can be sent during each 2 *Hz* cycle.

As stated in [15] the RF link can handle approximately 480 bytes of data at 2 *Hz*. Each telemetry packet transmitted from the UTV is shown in Tables

Description	**Bytes**
Rate gyroscope reference voltage and temperature	2
Magnetometer reference voltage	1
Main battery voltage	1
CAN bus voltage	1
OBC voltage	1
OBC temperature	1
Megabytes logged	1
Autopilot, Estimator and Fan status	2
TOTAL	10
Percentage of bandwidth	2.1 %

Table A.3: Contents of Secondary Packet

Description	**Bytes**
Estimated N	2
Estimated E	2
Estimated ψ	2
r measurement	2
V measurement	2
State error covariances	6
TOTAL	16
Percentage of bandwidth	3.5 %

Table A.4: Contents of EKF Packet

A.1 to A.6, along with the corresponding usage of the available bandwidth. When summing the bandwidth usage of all the individual packets it is evident that the transmission of telemetry ultimately makes use of approximately 25.5 % of the available bandwidth. A relatively large portion of the bandwidth is therefore still available. This proves advantageous due to the presence of the *Upload all obstacles* button on the ground station GUI. Usage of this button implies the transmission of a large amount of obstacle coordinates at once and could require significant bandwidth. As was seen during practical implementation, not all obstacle coordinates are always successfully received by the UTV, when this button is used, and a threshold of approximately 30 square obstacles was found. When an amount of obstacles beyond this threshold needs to be uploaded to the OBC it is recommended that only the first 30 obstacles be uploaded in a batch and the remainder of the obstacles uploaded individually.

Description	**Bytes**
Number of *Voronoi* obstacles	4
Number of *Visibility* obstacles	4
Population algorithm selected	1
Shortest path algorithm selected	1
Starting point *N*	4
Starting point *E*	4
Destination *N*	4
Destination *E*	4
Voronoi calculation span	4
Visibility calculation span	4
TOTAL	34
Percentage of bandwidth	7.1 %

Table A.5: Contents of Path Planning Packet

Description	**Bytes**
Wheel speed	4
Translational speed	4
Direction	1
Motor enabled/disabled	1
TOTAL	10
$\times$ 2 (Left and Right)	20
Percentage of bandwidth	4.2 %

Table A.6: Contents of Encoder Packet

A.2.3 CAN Bus Communication

As mentioned in Chapter 2 communication between drive systems, OBC and the IMU is established through use of the CAN bus. CAN messages are transmitted and received by these nodes and all messages are uniquely identified by a 29-bit CAN 2.0B identifier. Each CAN message can also contain a maximum of 8 bytes of data in addition to the identifier. An indepth discussion of the the original CAN protocol can be seen in [12] and the messages in this project is only slightly adjusted to accommodate the drive systems. This section presents the CAN messages used as well as a short description of each message. Each node is shown along with the messages which are generated by the specific node.

PC104/CAN Controller Node

- 0x1E8F010A: Received by left drive system node where 1*st* byte of data indicates the following,

  - 0x01 - Reference wheel speed
  - 0x02 - Disable drive system
  - 0x03 - Enable drive system

- 0x1A8F010C: Received by right drive system node where 1*st* byte of data indicates the following,
  - 0x01 - Reference wheel speed
  - 0x02 - Disable drive system
  - 0x03 - Enable drive system
- 0x010101FF: Received by all nodes and requests each node to reply with its telemetry
- 0x0B8F0102: Received by IMU node and toggles self-test function
- 0x0B8F0104: Received by IMU node and triggers magnetometer set/reset function

IMU Sensor Node

- 0x0A800201: Received by PC104/CAN Controller Node and contains IMU ADC filtered measurements
- 0x0A810201: Received by PC104/CAN Controller Node and contains IMU ADC filtered measurements

Left Drive System

- 0x0E800A01: Received by PC104/CAN Controller Node and contains the following data,
  - Byte 2 - Angular wheel velocity
  - Byte 4 - Direction flag
  - Byte 7 - Enabled/disabled status

Right Drive System

- 0x0A810C01: Received by PC104/CAN Controller Node and contains the following data,
  - Byte 2 - Angular wheel velocity
  - Byte 4 - Direction flag

– Byte 7 - Enabled/disabled status

It should be noted that the angular wheel velocities, shown above, represent a factor 16 scaled version of the actual measured angular wheel speed. Since the wheel speeds are comparatively low, with a maximum value of 8 rad/s, resolution in measurements would be insufficient if the measurements were simply transmitted in the envelope of the unsigned character variable used. This would only yield an integer resolution in speed measurements since decimal places would be truncated during cast from floating point variable to unsigned character variable on the PIC microprocessor. The measurement is therefore multiplied by 16 on the microprocessor, before transmission, and decimal places are included in the integer transmitted value. By dividing the received values on the OBC by 16 again the decimal places are extracted and a resolution of $\frac{1}{16}$ rad/s is achieved. This is feasible since the maximum wheel speed of 8 rad/s is known and hence a multiplication by 16 will never cause the unsigned character data byte to overflow.

A.3 Batteries

A.3.1 Sealed Lead-Acid Batteries

Figure A.3 shows a schematic of the Sealed Lead-Acid battery charger implemented by [16]. This charger proved to be very efficient and the discussions on this charger, in Chapter 2, are useful when viewing the schematic.

A.3.2 Lithium-Polymer Batteries

As mentioned in Chapter 7, the author of this document lost valuable data during a fire, on the 18th of May 2008, at the residence where he was completing this dissertation. The author would like to emphasize the importance of correct use of Lithium-Polymer batteries during charging cycles and discharging cycles as it was established that this was the cause of the fire. It is highly recommended that Lithium-Polymer batteries never be charged unattended and that a concrete fire resistant surface is used as platform for the battery during charging cycles. A reliable microprocessor charger should also be used, which is designed for the specific battery, and batteries should never be overcharged!

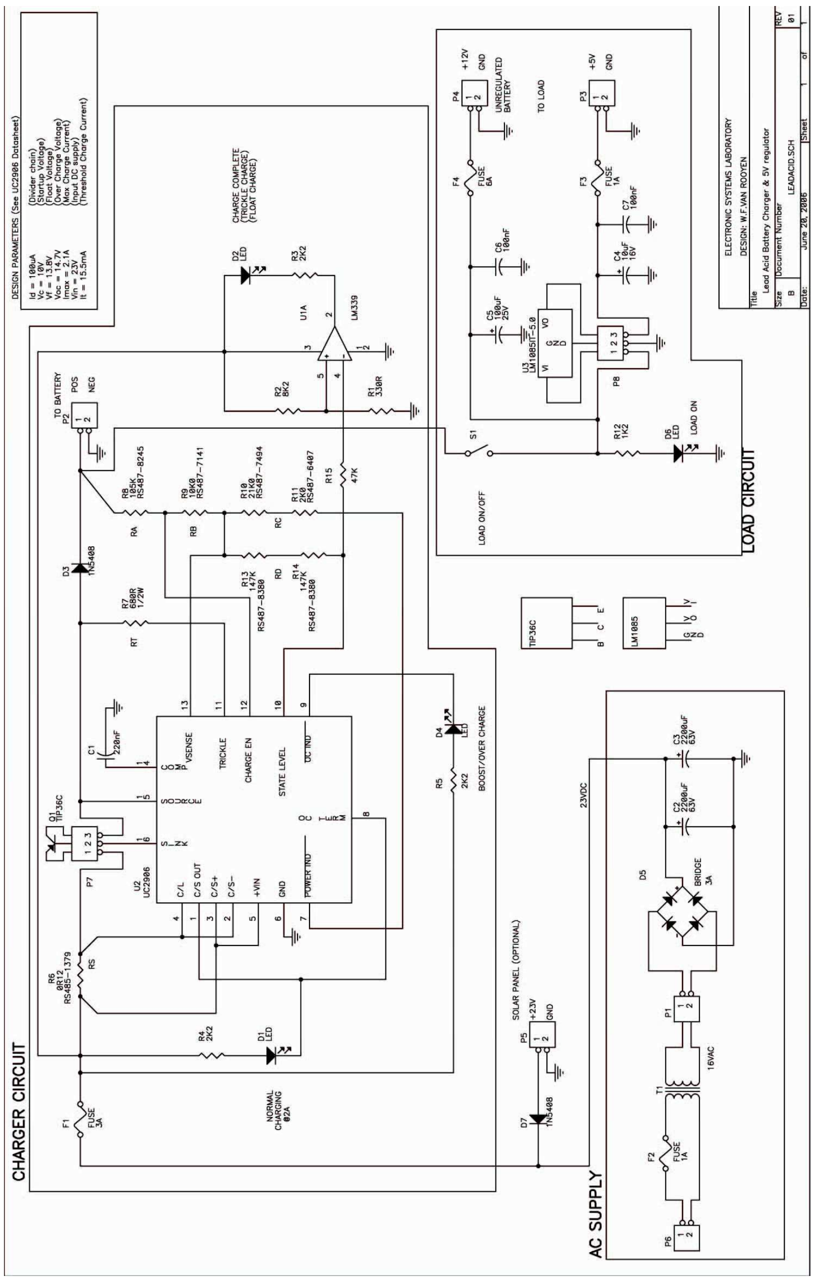


Figure A.3: Battery Charger Schematic in memory of Willie van Rooyen

The Lithium-Polymer battery was being used as light weight high density power source for the ground station RF module discussed in Appendix C. Successful test runs were however still conducted with the use of an alternate RF module, courtesy of [4]. This RF module was left unaltered and only the RF module on the UTV was reconfigured for communication with the newly acquired ground station RF module.

A.4 Angle Conditioning

The flowchart shown in Figure A.4 represents the conditioning routines which were implemented on the heading angles of path line segments and the current measured heading of the UTV. The approach consists of representing all angles with their equivalent mapping within 0° and 360°, and these conditioned angles are then adjusted in such a way that the heading difference between measurement and reference is never bigger than 180°. This ensures the UTV will only turn through acute and obtuse angles when trying to achieve a new headings.

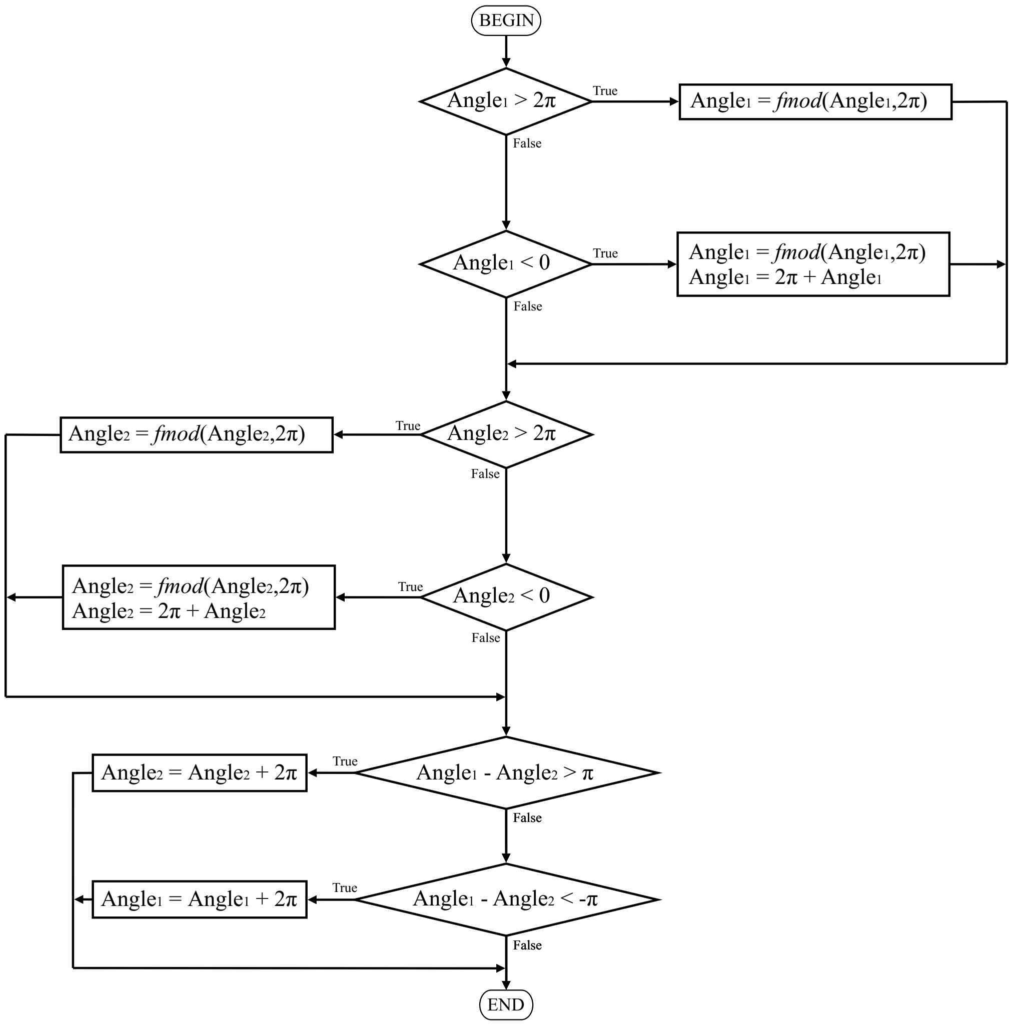


Figure A.4: Flowchart of the *Condition Angles* Routine

Appendix B

UTV Parameters

This appendix has the purpose of presenting the development of a theoretical model of the UTV's DC motors for use with the control algorithms of Chapter 4. The approach consists of mathematically modelling and finding a transfer function for one of the DC motors on the UTV, with a supply voltage as input and angular wheel velocity as output. The transfer function is further presented in state-space form so as to allow for state-space control design techniques. The determination of the slip gain discussed in Chapter 4 is also shown.

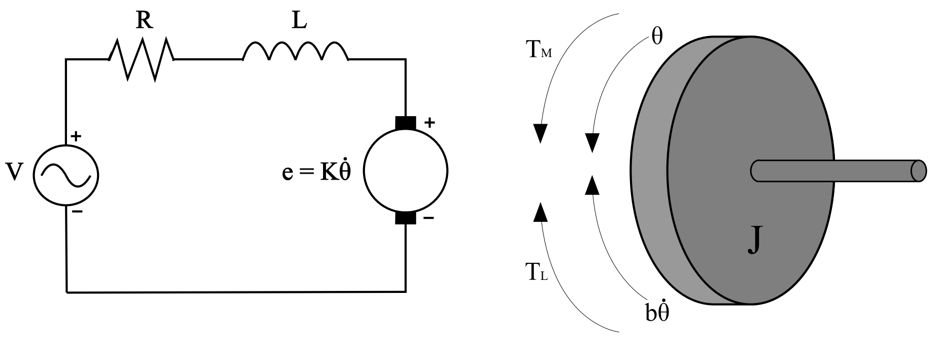


Figure B.1: Schematic representation of DC Motor

Figure B.1 shows a representation of the electric circuit of the armature and a free body diagram of the rotor of a DC motor acting under load. The rotor and the shaft are assumed to be rigid. Through use of Kirchoff's voltage law and Newton's law of rotational motion, equations can be derived to represent the operation of the DC motor. These equations are used to form a transfer function which relates the rotational output speed of the motor to the supplied input voltage. In Figure B.1 the symbols shown represent the physical parameters listed on the next page.

- J - moment of inertia of the rotor [$\frac{kg\ m^2}{s^2}$]
- b - Damping ratio of the mechanical system [Nms]
- $K = K_e = K_t$ - Electromotive force constant [$\frac{Nm}{A}$]
- R - Electrical resistance [Ohm]
- L - Electrical inductance [H]
- V - Input voltage [V]
- i - Armature current [A]
- θ - Output position of rotor shaft [rad]
- $\dot{\theta}$ - Output angular velocity of rotor [$\frac{rad}{s}$]
- T_M - Motor torque [Nm]
- T_L - Load torque [Nm]
- e - Back EMF voltage [V]

As shown in [25] the motor torque is related to the armature current, i, by a constant factor K_t as shown in the following equation.

$$T_M = K_t i \tag{B.0.1}$$

The back EMF, e, is related to the angular velocity of the rotor by

$$e = K_e \dot{\theta} \tag{B.0.2}$$

Since K_t is equal to K_e in SI units one constant variable K can be used to represent both. Through use of Kirchoff's voltage law and Newton's law of rotational motion the following equations can now be written from Figure B.1 by making use of Equations B.0.1 and B.0.2. The angular velocity of the wheels are represented by ω which equals $\dot{\theta}$. The 1.5 : 1 sprocket system ratio is also taken into account and the factor 1.5 is apparent in Equations B.0.3 and B.0.4.

$$Ki - T_L = J\frac{\dot{\omega}}{1.5} + b\frac{\omega}{1.5} \tag{B.0.3}$$

$$L\frac{di}{dt} + Ri = V - K\frac{\omega}{1.5} \tag{B.0.4}$$

B.1 Transfer Function

By taking the Laplace Transforms of Equations B.0.3 and B.0.4 new equations are obtained in terms of the complex Laplace operator, **s**, and after substitution and rearranging of these equations a transfer function is yielded which relates output angular velocity of the UTV's wheels to input voltage and gives insight into the frequency characteristics of the system. The equations after taking the Laplace Transforms are shown below.

$$(\frac{1}{1.5})\omega(s)[sJ + b] = KI(s) - T_L \tag{B.1.1}$$

$$I(s)[sL + R] = V(s) - (\frac{K}{1.5})\omega(s) \tag{B.1.2}$$

The following equation is then found by eliminating $I(s)$ from the equations above through substitution.

$$\omega(s) = \frac{1.5K}{(sL + R)(sJ + b) + K^2} V(s) - \frac{1.5(sL + R)}{(sL + R)(sJ + b) + K^2} T_L$$

As stated in [10] the damping is negligibly small and is therefore ignored. The equation therefore reduces to

$$\omega(s) = \frac{1.5K}{s^2JL + sJR + K^2} V(s) - \frac{1.5(sL + R)}{s^2JL + sJR + K^2} T_L \tag{B.1.3}$$

In Equation B.1.3 the second term on the right represents the way in which the load torque, T_L, couples into the system. T_L is regarded as a disturbance torque on the system and the open loop block diagram of this equation is shown in Figure B.2. A voltage input to angular velocity output transfer function, for the system plant, is therefore obtained and shown in Equation B.1.4. A determination of the physical system parameters remains. These parameters were determined by [10] and a brief discussion is presented on how this was accomplished.

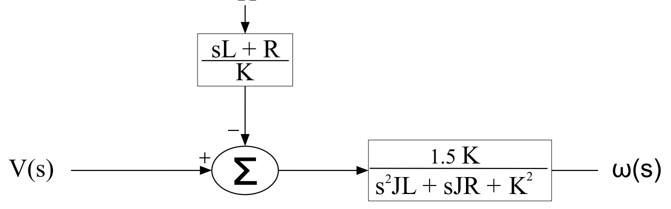


Figure B.2: Open loop plant and disturbance torque of DC motor

$$G(s) = \frac{\omega(s)}{V(s)} = \frac{1.5K}{s^2 JL + sJR + K^2} \tag{B.1.4}$$

The resistance and armature inductance of each motor was determined experimentally through use of an accurate multimeter. The electrical motor constant, K_e, was determined by applying several voltages to the terminals of the motor and recording an angular velocity of the motor shaft at each applied voltage with a tachometer, which measures revolutions per minute. These RPM measurements are easily converted to rad/s measurements and by taking an average of the calculated K_e values at each applied voltage an accurate representation of the electrical motor constant is obtained. As previously mentioned, the electrical motor constant and armature constant are equal in SI units and an experimental determination of K_t is therefore not required. A measurement of the armature current at each applied voltage was also taken to aid in calibration of current measurements by the current transducer mentioned in Chapter 2. A summary of the parameters obtained is listed below. As can be seen in Table B.1, there exists slight differences between the parameters of DC motor A and B. These differences are however negligible when considering the hierarchy of higher level controllers discussed in Chapter 4. The parameters of DC motor B are therefore used and the assumption is made that the motors are identical in response.

Parameter	**Motor A**	**Motor B**
R	$0.7\ \Omega$	$0.5\ \Omega$
L	$766\ \mu H$	$785\ \mu H$
K_e	$1.9\ \frac{V}{rad/s}$	$1.9\ \frac{V}{rad/s}$
K_t	$1.9\ \frac{V}{rad/s}$	$1.9\ \frac{V}{rad/s}$

Table B.1: Measured Physical Motor Parameters

One last parameter remains which is the motor inertia, J. As mentioned in [10] the inertia of the vehicle in its entirety is substantially larger than the inertia of the motor itself. The inertia of the vehicle reflected back to the motors is therefore used in determining the transfer function of the DC motors. The assumption is made that weight is distributed uniformly over the UTV's chassis and a mass of 6.25 kg therefore rests on each wheel. By approximating the point of contact between each wheel and the ground as a point mass of

6.25 kg, acting at the radius of the wheel, as shown in Figure B.3, the inertia can be found with Equation B.1.5.

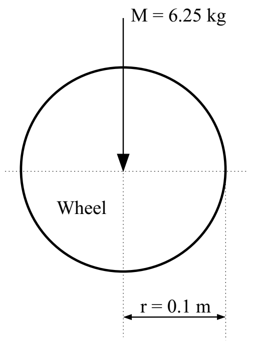


Figure B.3: Uniform weight distribution on wheels of UTV

$$\begin{aligned} J_{wheel} &= Mr^2 \qquad \text{(B.1.5)} \\ &= 0.0625\ kg.m^2 \end{aligned}$$

The total inertia seen by each motor on the UTV is calculated by summing the reflected wheel inertia, of the front and rear wheels on one side of the UTV, at the motor axle. For utmost accuracy the inertia of the sprocket system, wheel axles and motor axles also need to be taken into account and the exact weight on each wheel calculated. This higher accuracy is however sacrificed to avoid complexity. Calibration and practical measurements provide a reference and once the plant model is theoretically determined it can be compared to actual plant measurements and calibrated so as to match the physical plant exactly with a step response. Practical methods of determining the exact inertia also exists but is beyond the scope of this project.

The total reflected inertia at the motor axle is therefore determined with the following equation which makes use of the sprocket system ratio N.

$$\begin{aligned} J_{rotor} &= \frac{J_{rear\ wheel}}{N^2} + \frac{J_{front\ wheel}}{N^2} \\ N &= \frac{1}{1.5} \\ J_{rotor} &= 0.281\ kg.m^2 \qquad \text{(B.1.6)} \end{aligned}$$

All physical system parameters have now been determined and are substituted into Equation B.1.4 to yield the transfer function of the system plant for frequency analysis.

$$G(\mathbf{s}) = \frac{\omega(\mathbf{s})}{V(\mathbf{s})} = \frac{12920}{\mathbf{s}^2 + 637\mathbf{s} + 16360} \tag{B.1.7}$$

The step response of this plant is compared to the actual step reponse of a single UTV motor. The actual step response is recorded with the hardware configuration discussed in Chapter 2. Since the inertia parameter is most likely to be responsible for inaccuracies in the plant model, the inertia parameter is varied until the simulated plant step response matches the measured step response.

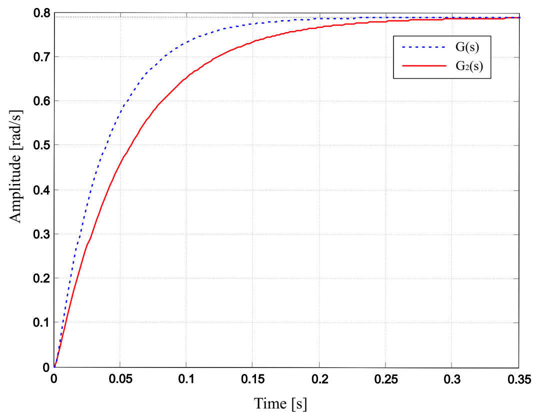


Figure B.4: Step responses of system plant and modified system plant

In Figure B.4 the original theoretical plant step response, $G(s)$, is shown alongside the actual measured step response, $G_2(s)$. The actual plant clearly has a slower settling time which indicates an inaccuracy in the plant model. The inertia parameter, J, is varied and the corresponding step responses obtained in MATLAB. Through a process of trial and error an inertia parameter value was found which matches the simulated step reponse with the measured step reponse. This value for J and the new plant transfer function is shown below.

$$J_{rotor} = 0.4157\ kg.m^2$$

$$G_2(\mathbf{s}) = \frac{\omega(\mathbf{s})}{V(\mathbf{s})} = \frac{8734}{\mathbf{s}^2 + 636.9\mathbf{s} + 11060} \tag{B.1.8}$$

Figure B.5 shows the pole locations of the original plant and the matched plant. The bandwidth of the matched plant is reduced to 2.83 Hz.

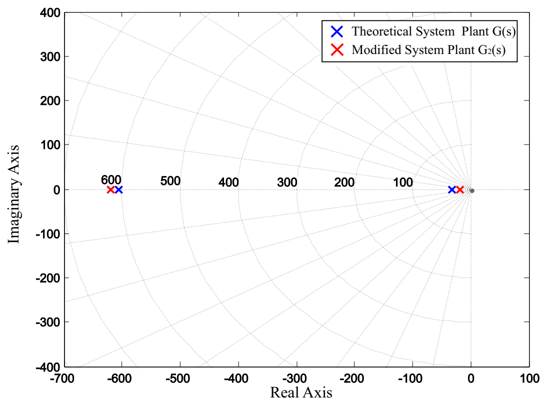


Figure B.5: Pole locations of system plant before and after modification

B.2 State-Space

In order to make use of state-space control methods during the design of the drive system controller the system plant is also presented in state-space form. The states of the second order system are chosen as

- i - Armature Current
- ω_{wheels} - Angular velocity of wheels

The input to the system is the applied voltage on the motor

- V

The disturbance input to the system is the load torque

- T_L

Equations B.0.3 and B.0.4 are rearranged to yield the following differential equations which are in standard state-space format,

$$\begin{aligned} \frac{di}{dt} &= -\frac{K}{1.5L}\omega - \frac{R}{L}i + \frac{1}{L}V \\ \frac{d\omega}{dt} &= \frac{1.5K}{J}i - \frac{1.5}{J}T_L - \frac{b}{J}\omega \end{aligned}$$

and the standard state-space matrices are easily found as,

$$\mathbf{F} = \begin{bmatrix} -\frac{R}{L} & -\frac{K}{1.5L} \\ \frac{1.5K}{J} & -\frac{b}{J} \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 0 & 1 \end{bmatrix} \qquad \mathbf{B_W} = \begin{bmatrix} 0 \\ -\frac{1.5}{J} \end{bmatrix} \tag{B.2.1}$$

where

$$\begin{aligned} \dot{\underline{\mathbf{x}}} &= \mathbf{F}\underline{\mathbf{x}} + \mathbf{G}u + \mathbf{B_W}W \\ y &= \mathbf{H}\underline{\mathbf{x}} \end{aligned}$$

The assumption is made that the measurement noise is negligibly small with the averaging process during encoder measurements, and filtering process during current measurements, taken into account (as described in Chapter 2).

B.3 Discrete State-Space

The continuous state-space model is further represented in discrete state-space to allow for discrete control design methods to be applied. The reason for this is stated in Chapter 4. With the help of MATLAB a discrete equivalent plant is found by including a zero order hold circuit with the sample time of 0.04 s, as discussed in Chapter 2, to yield the following discrete state-space matrices

$$\mathbf{\Phi} = \begin{bmatrix} -0.01454 & -1.313 \\ 0.00558 & 0.5038 \end{bmatrix} \qquad \mathbf{\Gamma} = \begin{bmatrix} 1.037 \\ 0.3917 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 0 & 1 \end{bmatrix} \qquad \mathbf{\Gamma}_\omega = \begin{bmatrix} 0 \\ -\frac{1.5}{J} \end{bmatrix} \tag{B.3.1}$$

where

$$\begin{aligned} \underline{\mathbf{x}}(k+1) &= \mathbf{\Phi}\underline{\mathbf{x}}(k) + \mathbf{\Gamma}u(k) + \mathbf{\Gamma}_\omega \omega(k) \\ y(k) &= \mathbf{H}\underline{\mathbf{x}}(k) \end{aligned}$$

The step response of this discrete state-space plant is finally investigated in MATLAB and the expected response obtained as shown in Figure B.6. The bandwidth is once again 2.83 Hz and the effect of the zero-order hold discretization process is visible.

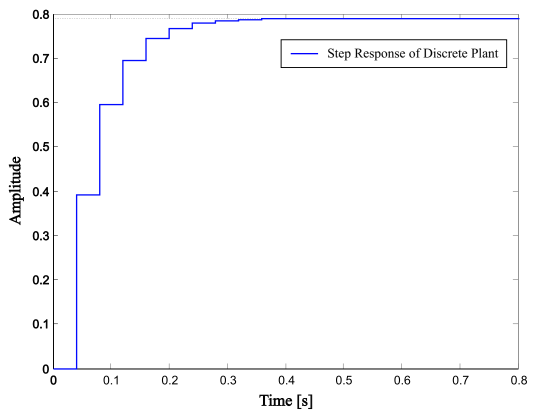


Figure B.6: Step Response of Discrete Plant with Sample Time 0.04 *s*

B.4 Calculating the Yaw Rate Slip Gain

In Chapter 4 the slip gain for the yaw rate controller was introduced. The calculation of this gain is done by commanding certain reference wheel speeds from the UTV which make it turn clockwise and anti-clockwise at varying yaw rates. With the equations presented in Chapter 4 the theoretical expected yaw rate is calculated and then compared to the actual yaw rate of the UTV. The actual yaw rate is determined by recording the time it takes for 10 full rotations of the UTV to be completed and then calculating the yaw rate. By making use of 10 rotations instead of just one more accurate results are obtained due to the inherent averaging characteristics of this process. Table B.2 shows a comparison of these recorded yaw rates and the theoretical yaw rates at corresponding wheel speeds. It is clearly visible that the actual yaw rates are consistently less than the theoretical yaw rates. These figures were also checked on different surfaces and on a carpet in the ESL laboratory for instance, which has a high frictional constant, the recorded yaw rates dropped significantly. The gain can therefore also be interpreted as a load gain seeing as it was evident that slip of wheels was not the only cause for reduced yaw rates but also frictional loads. Ultimately the specific origin of the reduced yaw rates is beyond the scope of this project and an accurate slip gain is only required for the surface on which the UTV is tested.

By plotting these theoretical values and actual recorded values opposite each other, a value for the slip gain can be found by doing a linear fitting on this plot, as shown in Figure B.7. The offset of this linear fitting is negligibly small and therefore omitted in the design of the yaw rate controller due to the integral terms in higher level controllers, as shown in Chapter 4.

W_L	W_R	Measured r $[rad/s]$	$\frac{0.1(W_L - W_R)}{d}$ $[rad/s]$
-4	4	-1.151	-2.286
-3	3	-1.005	-1.714
-2	2	-0.671	-1.143
2	-2	0.671	1.143
3	-3	0.976	1.714
4	-4	1.109	2.286

Table B.2: Theoretical vs Measured Yaw Rate

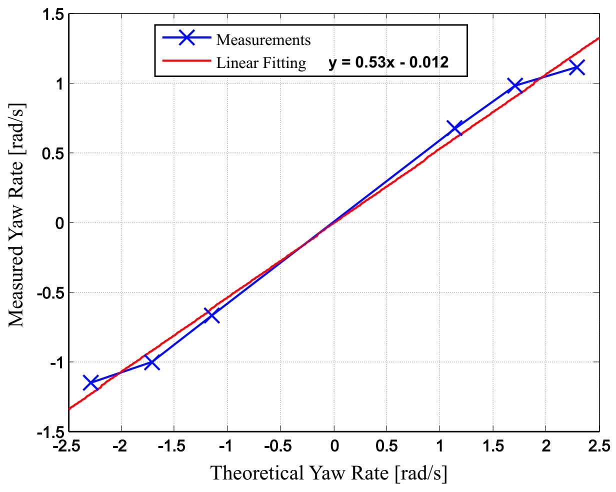


Figure B.7: Linear Fitting of Measured Yaw Rate vs. Theoretical Yaw Rate

Appendix C

Ground station

This appendix is dedicated to the ground station implementation which is used to communicate wirelessly with the UTV. The ground station plays a significant role in this project and serves as the medium through which the user interfaces with the UTV. The ground station is responsible for activation of initialization routines, provides real-time data and telemetry from the UTV during autopilot navigation, makes manual wireless operation of the UTV possible and gives a general visual indication of activity on the UTV which would not otherwise be easily accessible during operation.

C.1 Overview of Components

As shown in Figure 2.2 the ground station consists of a PC (preferably a Laptop for increased mobility) which runs the ground station software, a RF communications module [4] and a RF controller pad. The RF communications module provides a wireless link with the RF module on the UTV while the ground station PC communicates serially with the RF module via the UART protocol at 9600 baud. The ground station software provides a GUI to the UTV's onboard computer and allows real-time updating of variables on the OBC. Since the UTV does not make use of a RC transmitter, as is the case in previous UAV projects [4][15], an alternative form of manual control of the UTV needed to be implemented. Although manual control of the UTV is easily established through software implementation on the ground station PC, as will be discussed shortly, it proved inconvenient to be restricted to within the vicinity of the ground station PC and the PC keyboard during manual wireless control of the UTV, since the ground station PC is not easily carried around like a RC controller. A RF Logitech Cordless Rumblepad which also operates at 2.4 *Ghz*, shown in Figure C.1, was available for use. This cordless controller pad comes with manufacturer software which allows the rerouting of controller buttons to simulate specific keyboard buttons on the ground station PC being pressed. By setting up this software to intercept the controller buttons which are pressed and then simulating certain keyboard buttons being pressed, while the ground station software is running, manual control of the UTV is relayed from this controller pad to the UTV's onboard RF module via the ground station software. Both the RF controller pad and ground station RF module was found to operate simultaneously without conflicts.

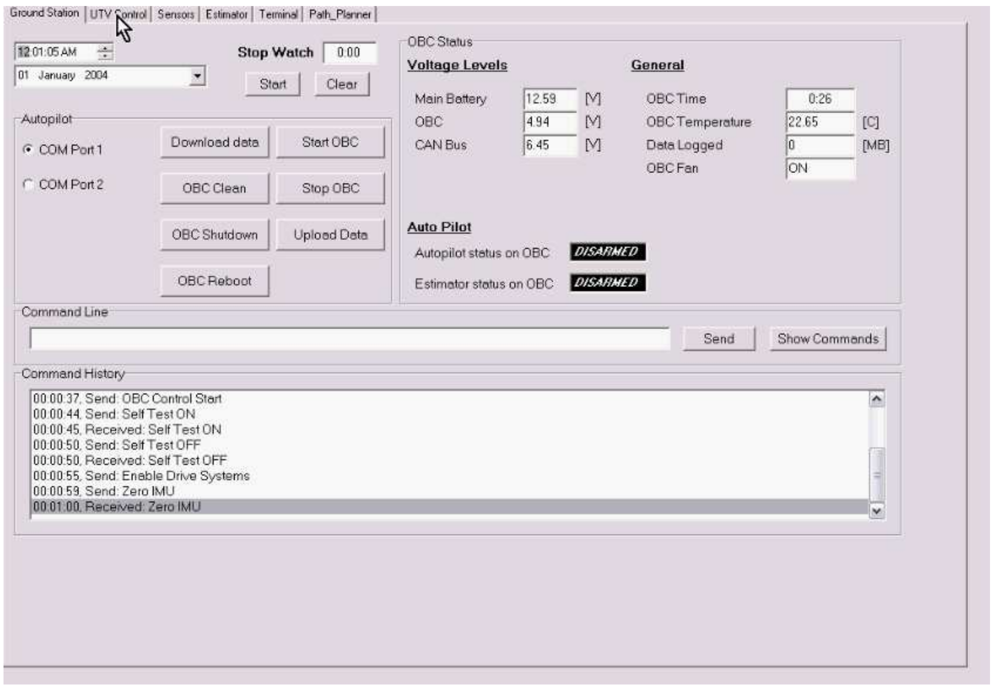
Figure C.1: Logitech Cordless Console Used for Manual Operation of the UTV

C.2 Ground station Software and GUI

As mentioned the ground station software provides a Graphical User Interface to the onboard computer of the UTV. This GUI is implemented with Borland C++ Builder 6 and the GUI implemented by [15] was used as a basic foundation to work from with a communications protocol already implemented. The GUI is a vital component used during test runs in this project. Due to the geometrical and graphical nature of path planning algorithms and the complexity of these algorithms being used in conjunction with control algorithms, the GUI had to be modified significantly to accommodate path planning, manual control of the UTV, and a two dimensional state estimator instead of the six degree of freedom estimator used in previous UAV projects. The GUI is also made more graphical to provide intuitive insight into activity on the UTV. Each interface page of the ground station GUI is now presented with a discussion on the various responsibilities of the components on each page.

C.2.1 Main Ground Station Page

The first page of the ground station GUI is shown in Figure C.2. This page is minimally altered from the one used by [15]. It shows basic information from the sensors on the PC104/CAN Controller board on the UTV such as voltage and temperature levels. A timer is also used to indicate how long the OBC control loops have been running and how many megabytes of telemetry data have been logged. These basic information boxes give an indication of the overall health of systems on board the UTV. The *OBC Fan* box is inactive in this project since alternative fans were used for the avionics package, as discussed in Chapter 2.



Main Battery	12.59	[V]	OBC Time	0:26	
OBC	4.94	[V]	OBC Temperature	22.65	[C]
CAN Bus	6.45	[V]	Data Logged	0	[MB]
			OBC Fan	ON	

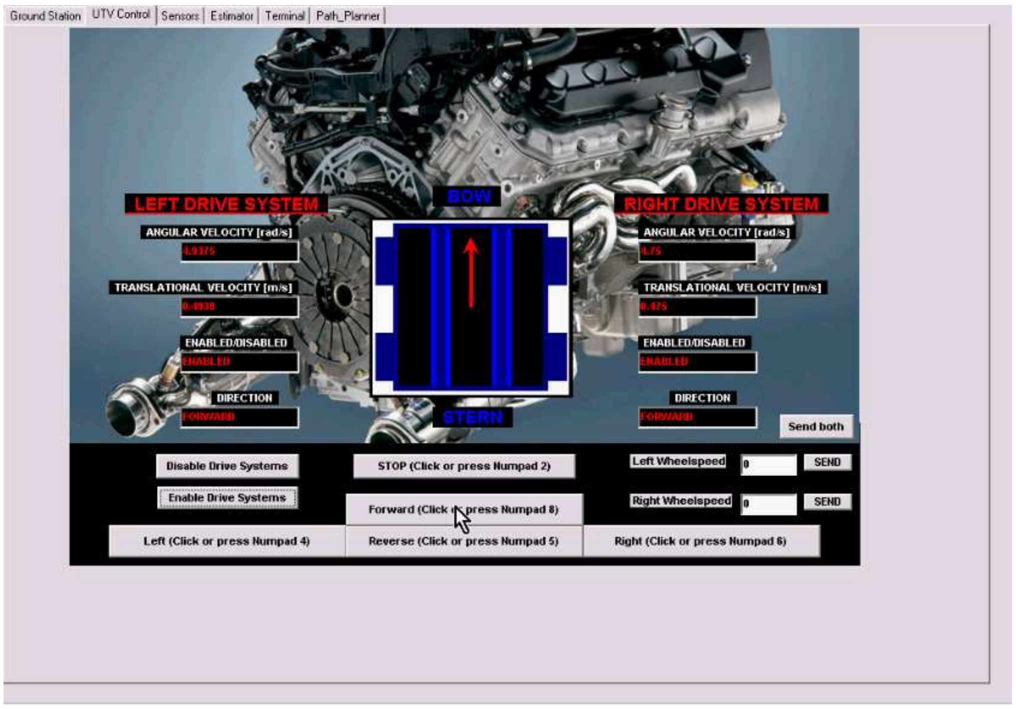
12:01:05 AM
01 January 2004
Stop Watch 0:00
Autopilot status on OBC: DISARMED
Estimator status on OBC: DISARMED

```
00:00:37, Send: OBC Control Start
00:00:44, Send: Self Test ON
00:00:45, Received: Self Test ON
00:00:50, Send: Self Test OFF
00:00:50, Received: Self Test OFF
00:00:55, Send: Enable Drive Systems
00:00:59, Send: Zero IMU
00:01:00, Received: Zero IMU
```

Figure C.2: Main Page of the Ground station GUI

Buttons are also shown which are used to start the OBC autopilot executable file, stop the autopilot and clean the logged data after use. The *Download data* button is used to download the C executable autopilot file, created and compiled on a *Linux* operating system with the *gcc* compiler, from the ground station PC to the UTV's avionics. This button also downloads the reference data file to the OBC which contains physical constants and estimator noise covariances. The *Upload data* button is used for uploading the error log file and telemetry log file from the UTV to the ground station PC after test runs. A *Reboot* button was added for quickly rebooting the OBC during debugging and the *OBC Shutdown* button is still present. The Command History list box displays all the commands which are issued by the pressing of any of the buttons on the ground station GUI, and also displays confirmation messages which are received from the OBC if a command is recognized or an error message if the command is not recognized. General error messages are also displayed here. The Command Line box is used to upload predefined commands to the OBC. Finally two text boxes are also shown which display the current status of the estimator and autopilot on the OBC, whether it be *Armed* or *Disarmed*.

C.2.2 UTV Manual Control Page


Ground Station
UTV Control
Sensors
Estimator
Terminal
Path_Planner
BOW
STERN
Send both
Disable Drive Systems
Enable Drive Systems
STOP (Click or press Numpad 2)
Forward (Click or press Numpad 8)
Left (Click or press Numpad 4)
Reverse (Click or press Numpad 5)
Right (Click or press Numpad 6)
SEND
SEND

LEFT DRIVE SYSTEM

ANGULAR VELOCITY [rad/s]: 4.9375

TRANSLATIONAL VELOCITY [m/s]: 0.4938

ENABLED/DISABLED: ENABLED

DIRECTION: FORWARD

RIGHT DRIVE SYSTEM

ANGULAR VELOCITY [rad/s]: 4.75

TRANSLATIONAL VELOCITY [m/s]: 0.475

ENABLED/DISABLED: ENABLED

DIRECTION: FORWARD

Left Wheelspeed: 0

Right Wheelspeed: 0

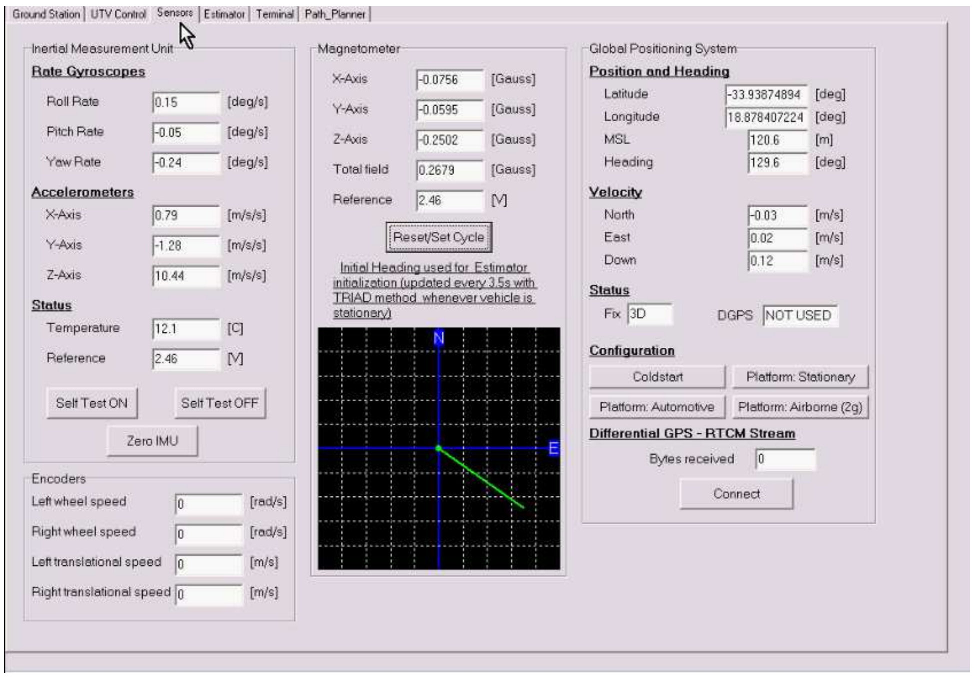
Figure C.3: Manual Control Page of the Ground station GUI

Manual control of the UTV is established through the page of the ground station GUI shown in Figure C.3. The autopilot on board the UTV is set up to recognize the instructions, which are transmitted when any of the buttons on this page is clicked, and then reacts by setting the respective wheel speeds of the UTV accordingly. The *Forward* button, when pressed, simply increases both the left and right commanded wheel speeds of the UTV in increments of 0.625 rad/s but does not exceed 8 rad/s. The *Reverse* button decreases both wheel speeds in the same increments but does not command wheel speeds below -8 rad/s. The *Left* button increases right wheel speed and decreases left wheel speed in the same increments and the *Right* button does the opposite. The *Stop* button brings the UTV to an immediate halt. These commands can also be sent to the UTV by pressing the corresponding keys on the Numpad of the keyboard, as shown in Figure C.3, and these keys can be pressed with any page of the GUI being active and still be registered. This provides immediate manual control of the UTV during any stage of navigation and activity on the GUI and also makes the manual control with the controller pad, previously mentioned, possible. Additional text boxes are also provided to allow immediate upload of exact left and right wheel speed references which is convenient during calibrations. Before any of these keys can be used to control the UTV the drive systems need to be activated with the *Enable Drive Systems* button. The drive systems can also be disabled should the need arise. Finally this page provides real-time telemetry of the left and right wheels speeds of the UTV in rad/s as well as forward velocity of the UTV in m/s. The direction in which the wheels are turning is also shown as well as the enabled or disabled status of the drive systems.

C.2.3 Sensor Page

The sensor page of the GUI is shown in Figure C.4 and is based on the one implemented by [15]. This page provides real-time measurements of all sensors on the UTV which were discussed in Chapter 2. Buttons are also present for calibration of the IMU sensors by calculating the appropriate gains and offsets for all sensor measurements on the OBC, while the UTV is stationary before any navigation is attempted. This provides good compensation for rate gyroscope biases due to temperature variation. The magnetometer also has a *Reset/Set Cycle* button which activates a routine in which the magnetometer is purged of any residual magnetic polarizations.

The only additions on this page are the drive systems telemetry which is also shown here for convenience and the diagram which indicates the current heading of the UTV. This diagram is updated every 3.5 seconds whenever the UTV is stationary and displays a line vector corresponding to the current heading of the UTV, calculated through use of the TRIAD method, discussed in [15], through utilization of the magnetometer and accelerometer measurements in 3 axes. This algorithm therefore provides the exact 3 dimensional attitude of the UTV when stationary. A similar diagram is used on the estimator page of the GUI as shown in the next section. This diagram also shows the heading of the UTV but is however updated with the heading obtained from the 2 dimensional state estimator which only makes use of the horizontal X-axis and Y-axis measurements of the magnetometer. Therefore, should the UTV be on non-level terrain the Z-component of the earth's magnetic



Inertial Measurement Unit

Rate Gyroscopes

Roll Rate	0.15	[deg/s]
Pitch Rate	-0.05	[deg/s]
Yaw Rate	-0.24	[deg/s]

Accelerometers

X-Axis	0.79	[m/s/s]
Y-Axis	-1.28	[m/s/s]
Z-Axis	10.44	[m/s/s]

Status

Temperature	12.1	[C]
Reference	2.46	[V]

Encoders

Left wheel speed	0	[rad/s]
Right wheel speed	0	[rad/s]
Left translational speed	0	[m/s]
Right translational speed	0	[m/s]

Magnetometer

X-Axis	-0.0756	[Gauss]
Y-Axis	-0.0595	[Gauss]
Z-Axis	-0.2502	[Gauss]
Total field	0.2679	[Gauss]
Reference	2.46	[V]

Initial Heading used for Estimator initialization (updated every 3.5s with TRIAD method whenever vehicle is stationary)

Global Positioning System

Position and Heading

Latitude	-33.93874894	[deg]
Longitude	18.878407224	[deg]
MSL	120.6	[m]
Heading	129.6	[deg]

Velocity

North	-0.03	[m/s]
East	0.02	[m/s]
Down	0.12	[m/s]

Status

Fix 3D DGPS NOT USED

Configuration

Differential GPS - RTCM Stream

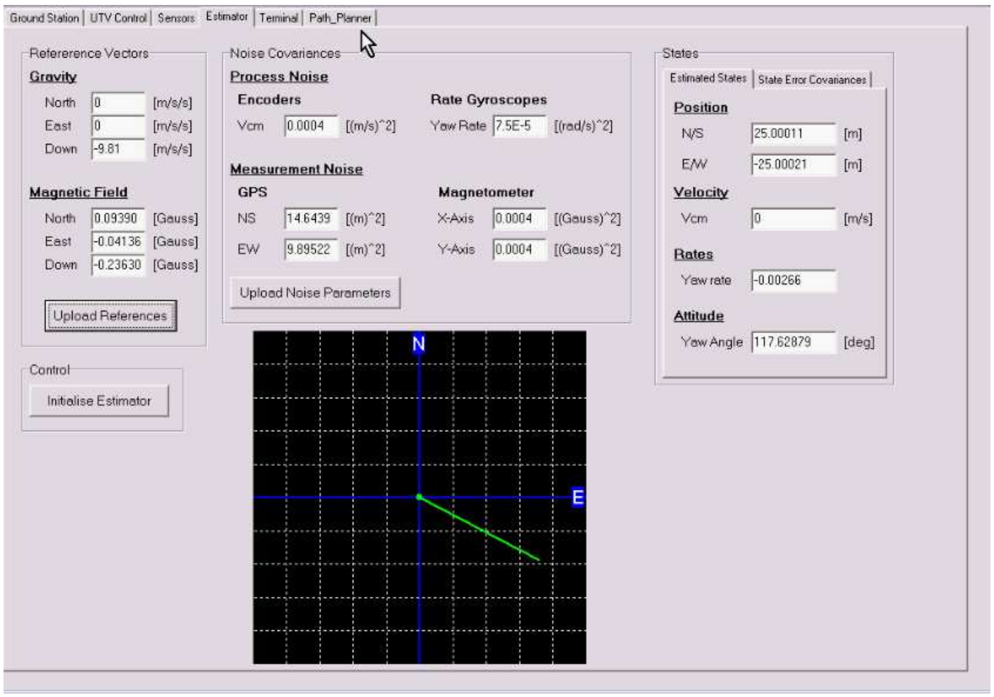
Bytes received 0

Figure C.4: Sensor Page of the Ground station GUI

field will couple in on the horizontal measurement axes of the magnetometer and consequently the estimator will yield an incorrect heading. A comparison of these two diagrams, before the UTV's autopilot has been armed and only the estimator has been armed, therefore provides an elementary way of checking whether the UTV is indeed on a level surface, which is a requirement of this project where all algorithms are based in 2 dimensional space. Should the vector on the diagram on this page therefore differ from the one on the estimator page of the GUI, a reconsideration of the UTV's testing grid is necessary. During test runs it was seen that a significant slope in testing terrain is required to corrupt the 2 dimensional algorithms of this project and successful testing was accomplished on a rugby field which is not perfectly level. A comparison between these two diagrams is therefore redundant in this project but still provides insight and emphasizes the 2 dimensional restriction on the algorithms implemented.

C.2.4 Estimator Page

The estimator page of the UTV, shown in Figure C.5, is dedicated to the operation of the two-dimensional estimator discussed in Chapter 5. The page provides text boxes for entry of reference vectors and a button to upload these references to the UTV through the RF link. These reference vectors are however already uploaded within the reference data file which is uploaded when the *Download Data* button is pressed on the main page of the ground station GUI and new references are therefore only uploaded should the UTV be operated in an area other than Stellenbosch. The page also contains text boxes and a button to upload noise covariances, for the process noise and measurement noise of the estimator, should the need arise to fine tune the estimator's performance.



Refererence Vectors

Gravity

North	0	[m/s/s]
East	0	[m/s/s]
Down	-9.81	[m/s/s]

Magnetic Field

North	0.09390	[Gauss]
East	-0.04136	[Gauss]
Down	-0.23630	[Gauss]

Noise Covariances

Process Noise

Encoders: Vcm 0.0004 [(m/s)^2]

Rate Gyroscopes: Yaw Rate 7.5E-5 [(rad/s)^2]

Measurement Noise

GPS: NS 14.6439 [(m)^2]; EW 9.89522 [(m)^2]

Magnetometer: X-Axis 0.0004 [(Gauss)^2]; Y-Axis 0.0004 [(Gauss)^2]

States

Position

N/S	25.00011	[m]
E/W	-25.00021	[m]

Velocity

Vcm 0 [m/s]

Rates

Yaw rate -0.00266

Attitude

Yaw Angle 117.62879 [deg]

Figure C.5: Estimator Page of the Ground station GUI

The *Initialize Estimator* button is used prior to arming the estimator and autopilot and commands the OBC to run an initialization routine which calculates the initial values of the states of the estimator. These initial states are calculated by taking the average of several successive GPS measurements and magnetometer measurements. The starting position of the UTV is also defined and uploaded on the Path Planning page of the GUI, prior to initialization of the estimator, and places the UTV at a certain point in a virtual two-dimensional grid used for the path planning algorithms. When the *Initialize Estimator* button is pressed the initial averaged GPS measurements are therefore used as the trim coordinates and all further GPS measurements are taken relative to these initial coordinates, then converted to meters and then shifted with the offsets obtained from the starting position defined on the

Path Planning page of the GUI to arrive at the UTV's position in the virtual path planning axis system.

After the *Initialize Estimator* button has been pressed a short delay occurs for the averaging process on the OBC and the initial values of the states are then seen in the corresponding text boxes. These text boxes also display the states of the estimator in real-time while the estimator is armed. As mentioned in the previous section a diagram is also present which only displays when the estimator is armed and shows the real-time heading obtained from the estimator in a NE axis system.

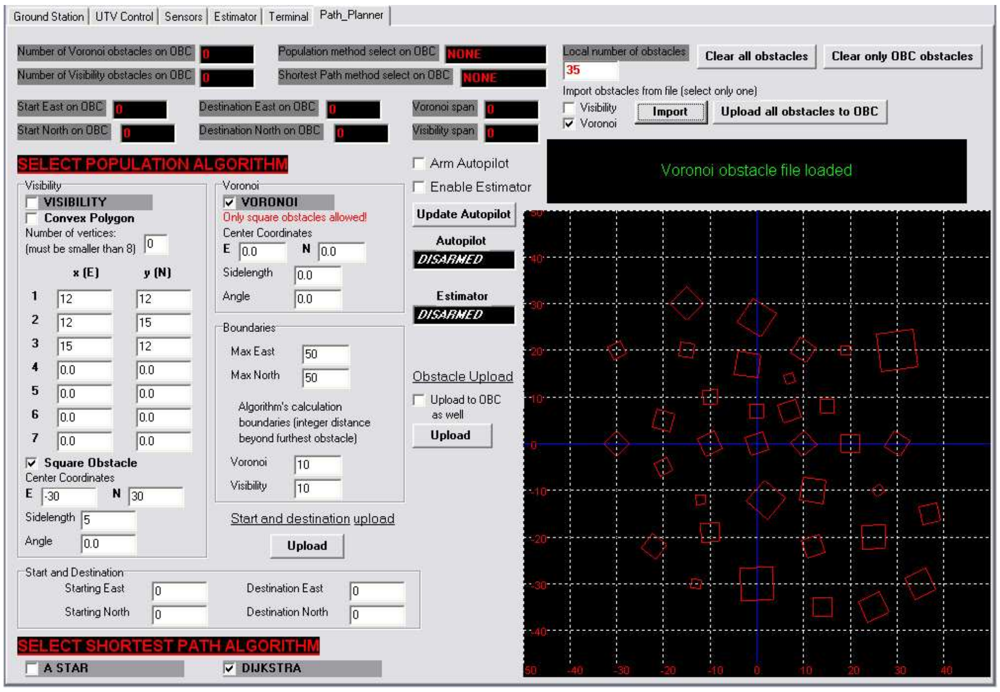
C.2.5 Terminal Page

The terminal page of the GUI is the same as the one used by [15] with the exception of the buttons, on the main page of the GUI, also included here for convenience during configuration of the OBC. When the sealed lead acid batteries of the UTV go flat during operation of the UTV unexpected shutdowns can occasionally cause corrupted data on the flash disk since the error log file and telemetry log file on the flash disk were not properly closed. This file corruption is easily remedied through this page of the ground station GUI by running the OBC Linux Kernel's extended file system check with the *e2fsck* command and then rebooting the system.

C.2.6 Path Planning Page

The core of the ground station GUI in this project resides on the Path Planning page shown in Figure C.6. This page is responsible for the wireless configuration of all variables on the OBC regarding the path planning algorithms. It also serves as a graphical medium through which UTV activity can be monitored throughout the autopilot navigation. Monitoring all these variables and ensuring a proper order in which variables are uploaded to the OBC can become rather complex and could lead to corrupted data on the OBC, should the proper sequence not be followed, which will ultimately cause failure of the path planning algorithms. A text box is therefore implemented which shows confirmation messages and warning messages throughout the process when the upload of data is attempted in an incorrect manner. This section will go through the steps required to set up the path planning algorithms on the UTV through use of the Path Planning page of the GUI.

Due to the nature of the path planning algorithms, discussed in Chapter 3, a predefined format for how obstacle coordinates are uploaded is necessary for



Ground Station | UTV Control | Sensors | Estimator | Terminal | Path_Planner

Number of Voronoi obstacles on OBC	0	Population method select on OBC	NONE
Number of Visibility obstacles on OBC	0	Shortest Path method select on OBC	NONE
Start East on OBC	0	Destination East on OBC	0
Start North on OBC	0	Destination North on OBC	0
Voronoi span	0	Visibility span	0

Local number of obstacles: 35

Clear all obstacles | Clear only OBC obstacles

Import obstacles from file (select only one)

[ ] Visibility
[✓] Voronoi

Import | Upload all obstacles to OBC

Voronoi obstacle file loaded

SELECT POPULATION ALGORITHM

Visibility

[ ] VISIBILITY

[ ] Convex Polygon

Number of vertices: (must be smaller than 8) 0

	x (E)	y (N)
1	12	12
2	12	15
3	15	12
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0

[✓] Square Obstacle

Center Coordinates E -30 N 30

Sidelength 5

Angle 0.0

Voronoi

[✓] VORONOI

Only square obstacles allowed!

Center Coordinates E 0.0 N 0.0

Sidelength 0.0

Angle 0.0

Boundaries

Max East 50

Max North 50

Algorithm's calculation boundaries (integer distance beyond furthest obstacle)

Voronoi 10

Visibility 10

Start and destination upload

Upload

[ ] Arm Autopilot

[ ] Enable Estimator

Update Autopilot

Autopilot: DISARMED

Estimator: DISARMED

Obstacle Upload

[ ] Upload to OBC as well

Upload

Start and Destination

Starting East	0	Destination East	0
Starting North	0	Destination North	0

SELECT SHORTEST PATH ALGORITHM

[ ] A STAR [✓] DIJKSTRA

Figure C.6: Path Planning Page of the Ground station GUI

each specific algorithm. As discussed in Chapter 3 this project implements 2 different *population* algorithms and two different *shortest path* algorithms. In order to calculate a path, one of the *population* algorithms need to be chosen and one *shortest path* algorithm need to be chosen. The *shortest path* algorithm is used on the results of the *population* algorithm to yield the *shortest path*.

The first step on the GUI page of Figure C.6 is to set the values for the boundaries of the virtual NE axis system in which the UTV will be operating. A message will be displayed in red in the black text box which indicates that all boundaries need to be set. Included in the *boundaries* group box is the *Voronoi* and *Visibility* boundary text boxes. These two boxes are used to set the *population* algorithms' calculation boundaries, in meters beyond the obstacle vertex which is the furthest from the origin of the NE axis system. After these boundaries have been set the ground station software has enough information to draw the NE axis system. This is done automatically when the *Upload* button is pressed, or the page is refreshed, in the larger black box shown and a grid is drawn which is marked in increments up to the boundaries defined. At this stage the message box will state that the OBC executable needs be started, if it has not already, and this is done on the main page of the ground station GUI or on the terminal page.

The next step is to define the starting point of the UTV as well as the desired destination within the boundaries previously defined. A simple form of error checking is implemented which disallows the upload of an identical starting and destination location and the message box displays a warning message when the user attempts this. A confirmation message is shown if these coordinates were successfully uploaded. The user can verify correct upload of the algorithm calculation boundaries, starting location and destination at this point by viewing the text boxes at the top of the page which displays the current variables on the OBC.

The most important step on this page of the GUI is the upload of obstacle coordinates. This can either be done by uploading the obstacles one at a time or with a batch upload of more than one obstacle. The GUI gives the user the option, with the *Upload to OBC as well* check box, to choose whether the current obstacle is just being uploaded to the local obstacle database on the ground station, for later upload to the OBC, or whether the obstacle is directly being uploaded to the OBC as well. All obstacles in the local database are drawn on the GUI's NE axis system while the current number of obstacles on the OBC is shown in the text boxes at the top of the page. The user can therefore monitor these two interfaces to ensure the obstacles on the OBC are synchronized with the obstacles on the ground station.

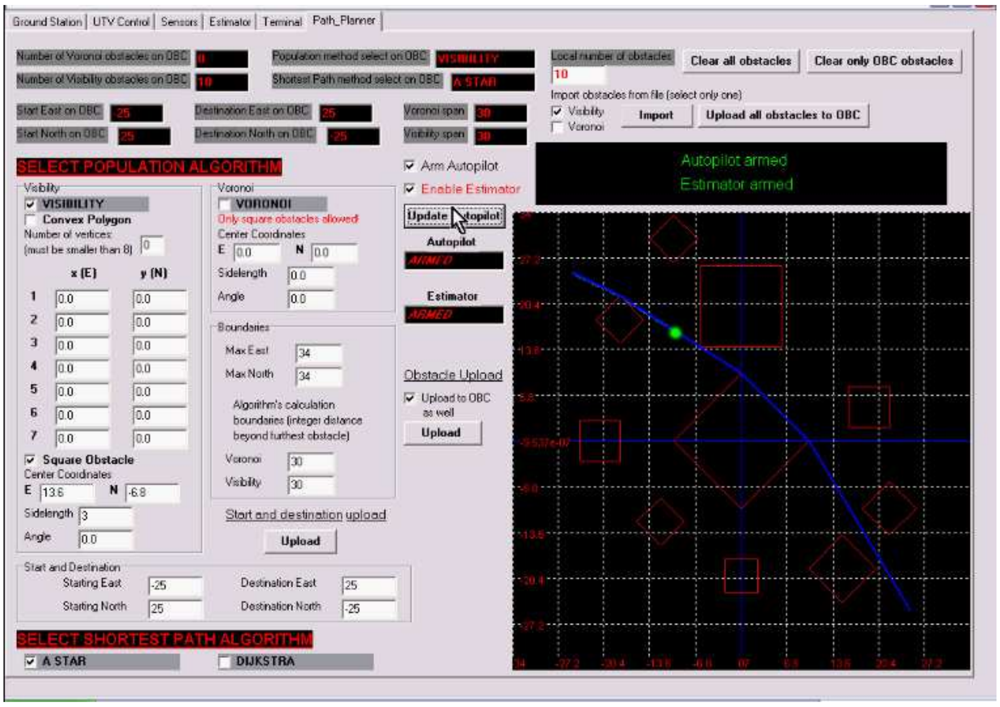
The obstacles are further uploaded in a specific format according to the chosen *population* algorithm, whether it be the *Voronoi* or *Visibility* graph algorithm. *Voronoi* obstacles are always square in form, as discussed in Chapter 3 and are defined by the, Northern and Eastern coordinates of its center, length of one of its edges and the angle which it makes with the N-axis. *Visibility* obstacles can take on the form of any convex polygon and the user has the option of defining the vertex locations of the obstacle or defining a square obstacle in the same format as the *Voronoi* obstacles, which is then converted to vertex format. The selection in *population* algorithm is made in the group boxes underneath the *Select Population Algorithm* heading by checking the appropriate check box. Error checking with corresponding warning messages are implemented which prevents the user from selecting neither of the *population* algorithms, both of the algorithms or the opposite algorithm once obstacles have already been uploaded in the format of the one algorithm. If an obstacle was uploaded incorrectly or the user wishes to use the alternate *population* algorithm, all obstacles have to be cleared first with the

Clear all obstacles button. The GUI also ensures that at least one and not both of the *shortest path* algorithms is selected and displays appropriate warning messages. The current selection in *population* algorithm and *shortest path* algorithm on the OBC is also shown in the OBC status text boxes at the top of the page.

As mentioned, several obstacles can also be uploaded in a batch. This is done with the *Upload all obstacles to OBC* button which transmits the coordinates of all obstacles, currently shown locally on the NE axis system, to the OBC in the format of the selected *population* algorithm. The obstacles can also be imported in a batch from an obstacle data file which is generated during *Matlab* simulation. The user has the option of importing a pre-defined *Voronoi* or *Visibility* obstacle file and care should be taken to ensure the corresponding *population* algorithm is selected in the check boxes under the *Select Population Algorithm* heading. Single obstacles can then be added to these imported obstacles locally before all obstacles are uploaded with the *Upload all obstacles to OBC* button. Should the data transmission be corrupted when large amounts of obstacles are uploaded to the OBC, with the *Upload all obstacles to OBC* button, the user also has the option to keep obstacles in the local database and clear only the obstacles on the OBC with the *Clear only OBC Obstacles* button. This is convenient when many obstacles have been added individually to an imported file and quick retransmission of all obstacles is all which is desired.

Once these steps have been followed correctly the UTV is ready for navigation. The first step is to initialize the estimator after calibrating all sensors, as previously mentioned, and enabling the drive systems. The estimator is then armed by checking the *Enable Estimator* check box and clicking the *Update Autopilot* button. A confirmation message will be displayed showing that the estimator has been armed and the autopilot is still disarmed. The current estimated position of the UTV is also drawn with a green dot on the NE axis system and the estimated heading of the vehicle can be seen visually on the Estimator page of the GUI as previously mentioned.

At this point all systems are ready for navigation to be started. The autopilot is now armed by checking the *Arm Autopilot* check box and clicking the *Update Autopilot* button. The path planning and control algorithms execute immediately after this, on the OBC, in accordance with the flowcharts of the state-machine discussed in Chapter 4 and the calculated path is transmitted back to the ground station and drawn on the NE axis system. The UTV then



Number of Voronoi obstacles on OBC: 0
Number of Visibility obstacles on OBC: 10
Population method select on OBC: VISIBILITY
Shortest Path method select on OBC: A STAR
Local number of obstacles: 10
Start East on OBC: -25
Start North on OBC: 25
Destination East on OBC: 25
Destination North on OBC: -25
Voronoi span: 30
Visibility span: 30

Import obstacles from file (select only one)
[✓] Visibility
[ ] Voronoi

[✓] Arm Autopilot
[✓] Enable Estimator

Autopilot armed
Estimator armed

SELECT POPULATION ALGORITHM

Visibility
[✓] VISIBILITY
[ ] Convex Polygon
Number of vertices: (must be smaller than 8) 0

	x (E)	y (N)
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0

[✓] Square Obstacle
Center Coordinates
E 13.6 N -6.8
Sidelength 3
Angle 0.0

Voronoi
[ ] VORONOI
Only square obstacles allowed!
Center Coordinates
E 0.0 N 0.0
Sidelength 0.0
Angle 0.0

Boundaries
Max East 34
Max North 34
Algorithm's calculation boundaries (integer distance beyond furthest obstacle)
Voronoi 30
Visibility 30

Start and destination upload

Autopilot: ARMED
Estimator: ARMED

Obstacle Upload
[✓] Upload to OBC as well

Start and Destination
Starting East -25
Starting North 25
Destination East 25
Destination North -25

SELECT SHORTEST PATH ALGORITHM
[✓] A STAR
[ ] DIJKSTRA

Figure C.7: Path Planning Page after the Estimator and Autopilot have been Armed

starts navigating this path and it's current location along the path is drawn on the NE axis grid with the help of the estimator telemetry. At this point the state-machine previously discussed has taken over and the UTV is fully on autopilot. Figure C.7 shows a typical display of the GUI at this stage. The calculated path is shown in blue and the current position of the UTV can be seen as the small green dot. The user now has the option of uploading an additional obstacle, while navigation is in progress, which triggers an immediate halt of the UTV and a recalculation of the path, after which navigation continues automatically along the new path.

Included with this thesis is a DVD featuring recordings of the actual UTV as well as the activity on the ground station during navigation. The reader is urged to watch this DVD in order to supplement the discussions of this appendix. It is recommended that reference be made to Appendix D before attempting to view these videos.

Appendix D

DVD Videos

All the videos mentioned in this thesis can be viewed on the DVD attached to the back page of this document. The DVD contains three main folders,

- 1 - A folder with videos regarding the practical demonstrations which relate to the discussions of Chapter 6 and 7
- 2 - A folder with videos regarding the path planning algorithms which relate to the discussions of Chapter 3
- 2 - A folder with supplementary software which may be required to play these videos

Please note that the videos regarding practical demonstrations are best viewed with *VLC Media Player*, included in the *Supplementary Software* folder. Alternatively the *DivX 5.0.5* codec, also in the *Supplementary Software* folder, can be installed and the practical demonstrations videos then viewed with *Windows Media Player*. Please be patient during installation of the *DivX 5.0.5* codec since it might install slowly.

The videos regarding the path planning algorithms are best viewed with *Windows Media Player*. The *TechSmith Screen Capture Codec (TSCC)* is however required and can also be found in the *Supplementary Software* folder. When viewing the path planning videos with *VLC Media Player* unexpected artifacts might be displayed which inhibits proper viewing. On some systems these videos did however display correctly in *VLC* as well.

Bibliography

[1] G.F. Franklin, J.D. Powell, M. Workman. *Digital Control of Dynamic Systems - Third Edition*. Addison Wesley Longman. 1998.

[2] F.P. Emami-Naeini. *Feedback Control of Dynamic Systems - Fourth Edition*. Prentice Hall. 2002.

[3] H.A. Haus, J.R. Melcher. *Electromagnetic Fields and Energy*. Prentice Hall. 1989.

[4] I.K. Peddle. *Autonomous Flight of A Model Aircraft*. Masters dissertation, University of Stellenbosch, 2005.

[5] J. Bijker. *Development of an Attitude Heading Reference System for an Air Ship*. Masters dissertation, University of Stellenbosch, 2006.

[6] J.G. Proakis, D.G. Manolakis. *Digital Signal Processing - Principles, Algorithms and Applications*. Prentice Hall. 1996.

[7] J. O'Rourke. *Computational Geometry in C - Second Edition*. Cambridge University Press. 2001.

[8] J. Venter. *Development of an Experimental Tilt Wing VTOL Unmanned Aerial Vehicle*. Masters dissertation, University of Stellenbosch, 2006.

[9] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer. 1997.

[10] M. Köpke. *The Development of a Drive System for an Unmanned Terrestrial Vehicle*. Mechatronic Project 488, University of Stellenbosch, 2005.

[11] R.L. Shackelford. *Introduction to Computing and Algorithms*. Addison Wesley Longman. 1998.

[12] S. Groenewald. *Development of a Rotary-Wing Test Bed for Autonomous Flight*. Masters dissertation, University of Stellenbosch, 2006.

[13] T. Jones. *Advanced Estimation 813 - Course Notes*. University of Stellenbosch. 2006.

[14] T. Wildi. *Electrical Machines, Drives, and Power Systems*. Prentice Hall. 2002.

[15] W. Hough. *Autonomous Aerobatic Flight of a Fixed Wing Unmanned Aerial Vehicle*. Masters dissertation, University of Stellenbosch, 2007.

[16] W. Van Rooyen. *Design of a Sealed Lead Acid battery charger.* Electronic Systems Laboratory Project, University of Stellenbosch, 2005/2006.

[17] Amit Patel's web page on the use of Heuristics.
http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#S12. 2008.

[18] OBC datasheet on Arbor-USA web page.
http://www.arbor-usa.com/pub/datasheet/computer_on_module/Em104-i613.pdf. 2008.

[19] Web page on A* Pathfinding for Beginners.
http://www.policyalmanac.org/games/aStarTutorial.htm. 2005.

[20] A visual implementation of Fortune's Voronoi algorithm.
http://www.diku.dk/hjemmesider/studerende/duff/Fortune/. 2001.

[21] Web page on Ferromagnetism by Glenn Elert.
http://hypertextbook.com/physics/electricity/ferromagnetism/. 1998-2008.

[22] Web page on Sensitivity of the Human Ear by R Nave.
http://hyperphysics.phy-astr.gsu.edu/Hbase/sound/earsens.html. 2008.

[23] The Visibility-Voronoi Complex by Ron Wein and Dan Halperin.
http://acg.cs.tau.ac.il/projects/internal-projects/the-visibility-voronoi-complex/project-page. 2008.

[24] Fortune's 2D Voronoi diagram C code.
http://www.cs.sunysb.edu/~algorith/implement/fortune/implement.shtml. 2008.

[25] Web page on DC Motor Speed Modeling by the University of Michigan.
http://www.engin.umich.edu/group/ctm/examples/motor/motor.html. 1997.