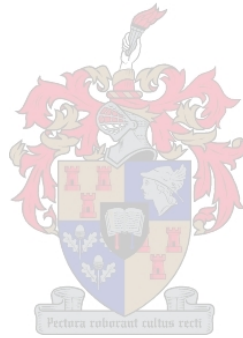


Vector Field Histogram Star Obstacle Avoidance System for Multicopters

by

Ruan Jacobus van Breda



*Thesis presented in partial fulfilment of the requirements for
the degree of Master of Engineering (Mechatronic) in the
Faculty of Engineering at Stellenbosch University*

Supervisor: Dr. W.J. Smit

December 2016

Declaration

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

Date: 2016/9/13

Copyright © 2016 Stellenbosch University
All rights reserved.

Abstract

Vector Field Histogram Star Obstacle Avoidance System for Multicopters

R.J. van Breda

*Department of Mechanical and Mechatronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.*

Thesis: MEng (Mech)

December 2016

In order to allow a multicopter to move about autonomously and detect and avoid obstacles in a two-dimensional plane, a prototype obstacle avoidance system, that implements the VFH* algorithm and uses a LIDAR sensor, was developed. The algorithm was coded in MATLAB and verified through simulations. Modifications were made to the VFH* algorithm to account for uncertainties with regard to the multicopter's position. Thereafter, the obstacle avoidance system was successfully implemented in a test environment and managed to navigate a multicopter to its goal position for various obstacle configurations.

Uittreksel

Vektor Veld Histogram Ster Hindernis Vermydings-sisteem vir Hommeltuie

(“Vector Field Histogram Star Obstacle Avoidance System for Multicopters”)

R.J. van Breda

*Departement Meganiese en Megatroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.*

Tesis: MIng (Meg)

Desember 2016

Ten einde 'n hommeltuig onafhanklik te laat rondbeweeg, terwyl dit versperings in 'n twee-dimensionele vlak waarneem en ontwyk, is 'n prototipe hindernis vermydings-sisteem, wat die VVH* algoritme en 'n LIDAR sensor gebruik, ontwikkel. Die algoritme is in MATLAB gekodeer en deur simulaties bevestig. Veranderinge is aan die VVH* algoritme aangebring om vir onsekerhede ten opsigte van die hommeltuig se posisie voorsiening te maak. Daarna is die sisteem suksesvol in 'n toetsomgewing geïmplementeer en het dit die hommeltuig suksesvol deur verskeie konfigurasies van versperrings genavigeer.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- Dr. W.J. Smit, my supervisor, for his guidance, support and interest showed in my research.
- SANRAL for granting me a bursary for my studies.
- My family, for their ongoing support.

Contents

Declaration	i
Abstract	ii
Uittreksel	iii
Acknowledgements	iv
Contents	v
List of Figures	viii
List of Tables	xi
Nomenclature	xii
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	3
1.3 Layout of the Study	3
2 Literature Review	4
2.1 Obstacle Avoidance Algorithms	4
2.1.1 Bug-1 Algorithm	5
2.1.2 Bug-2 Algorithm	6
2.1.3 Tangent Bug Algorithm	9
2.1.4 Potential Field Algorithms	13
2.1.5 The Virtual Force Field (VFF) Algorithm	14
2.1.6 Vector Field Histogram (VFH) Algorithm	18
2.1.7 Vector Field Histogram (VFH+) Algorithm	23
2.1.8 Vector Field Histogram Star (VFH*) Algorithm	25
2.1.9 Conclusion: Obstacle Avoidance Algorithm	26
2.2 Sensors	27
2.2.1 Ultrasonic Sensors (SONAR)	27
2.2.2 RADAR	28

2.2.3	LIDAR - Scanning Laser	28
2.2.4	Vision-Based Approaches	29
2.2.5	Conclusion: Sensors	30
3	VFH* Algorithm	31
3.1	Histogram Grid for Obstacle Representation	31
3.2	First Stage - The Primary Polar Histogram	32
3.3	Second Stage - The Binary Polar Histogram	34
3.4	Third Stage - The Masked Polar Histogram	35
3.5	Fourth Stage - Determining of Primary Candidate Directions . .	39
3.6	Look-ahead Verification	40
3.6.1	Projected Candidate Directions - Cost Function	42
3.6.2	Projected Candidate Directions - Heuristic Function	45
3.7	Conclusion	46
4	Code Validation: Simulations	47
4.1	Conclusion	52
5	Modifications to VFH*	54
5.1	Adding Safety Distance to Range Data	54
5.2	Compensating for Multicopter Position Uncertainties	55
5.3	Compensating for Multicopter Tilt Angle	58
5.4	Compensating for Multicopter Yaw Uncertainties	60
5.5	Conclusion	61
6	Hardware and Software	63
6.1	Robot Operating System (ROS)	64
6.2	Intel Edison	67
6.3	Pixhawk Flight Control Unit (FCU)	68
6.4	MATLAB	68
6.5	Lidar-Lite	69
6.6	Conclusion	70
7	Implementation: Test Flights	71
7.1	Results	73
7.2	Conclusion	78
8	Conclusion	79
	Appendices	81
A	How Multicopters Work	82
B	A* Search Algorithm	86
B.1	Heuristic	86

B.2 A* Search Algorithm Example	87
C Intel Edison Datasheet	90
D Lidar-Lite Datasheet	93
E Arduino Mega Datasheet	96
F Pixhawk Datasheet	99
G GPS Datasheet	102
H Piksi GPS Datasheet	105
List of References	108

List of Figures

1.1	Gemasolar (19.9 MW) CSP power plant in Spain	2
2.1	Obstacle avoidance using the Bug-1 algorithm.	6
2.2	Situation in which the Bug-1 obstacle avoidance algorithm is unable to reach the goal.	6
2.3	Obstacle avoidance using the Bug-2 algorithm.	7
2.4	Situation where the Bug-2 algorithm is less efficient than the Bug-1 algorithm.	9
2.5	Tangent Bug algorithm manoeuvring to the goal.	10
2.6	Robot moving towards the goal with sensor range indicated.	11
2.7	Tangent Bug algorithm encounters local minimum.	12
2.8	Tangent Bug algorithm with sensor range of zero.	13
2.9	Tangent Bug algorithm with infinite sensor range.	13
2.10	Updating of histogram grid using range data from an ultrasonic sensor.	15
2.11	Robot being steered towards the goal by the VFF algorithm.	16
2.12	Robot using VFF algorithm encounters a local minimum: u-shaped obstacle.	17
2.13	Wall following method (WFM) concept.	17
2.14	Robot using VFF algorithm encounters a local minimum: two obstacles with a gap in between.	18
2.15	Robot surrounded by obstacles.	19
2.16	Histogram grid representation of the robot's environment.	20
2.17	Construction of polar histogram by mapping active cells to sectors.	21
2.18	Polar histogram around robot's current position with threshold indicated.	22
2.19	Determining candidate sectors from polar histogram using thresholds.	22
2.20	Polar histogram: candidate steering directions.	23
2.21	Obstacle being enlarged by robot width and safety region.	24
2.22	Histogram representation of the primary polar histogram, binary polar histogram and the masked polar histogram.	24
2.23	Robot trajectories.	25
2.24	Robot using look-ahead verification to determine optimal path.	26

3.1	Primary polar histogram constructed around robot.	34
3.2	Binary polar histogram constructed around robot with two candi- date directions.	35
3.3	Robot trajectories.	36
3.4	Accounting for robot trajectories	37
3.5	Histogram representation of the primary polar histogram, binary polar histogram and the masked polar histogram	38
3.6	Robot using look-ahead verification to determine optimal path. . .	41
3.7	Look-ahead verification with search depth $n_g = 3$ and projected step distance d_s	42
3.8	Robot that's effective direction of motion is making no clear progress towards the target/goal.	44
3.9	Situation in which discount factor λ is required for the robot to choose the optimal path.	45
4.1	Overview of simulation code.	48
4.2	Simulation 1: The VFF algorithm encountering a local minimum. . .	49
4.3	Simulation 2: The VFH algorithm overcoming the local minimum that the VFF algorithm could not.	50
4.4	Simulation 3: The VFH+ algorithm accounting for the robot's width and a safety distance.	51
4.5	Simulation 4: A comparison between the VFH and the VFH+ al- gorithms.	51
4.6	Simulation 5: The VFH* algorithm implementing look-ahead veri- fication to choose the optimal path.	52
5.1	Multicopter position uncertainty.	56
5.2	Enlarged obstacles with multicopter as point-like vehicle.	57
5.3	Effect of various position uncertainties on multicopter trajectory. . .	57
5.4	Effect of position uncertainty on multicopter trajectory in a clut- tered environment.	58
5.5	Effect of tilt angle on range readings (side view).	59
5.6	Effect of tilt angle on multicopter trajectory.	60
5.7	Effect of yaw angle on range readings and obstacle uncertainty (top view).	60
5.8	Combined effect of tilt and yaw angles on range readings and ob- stacle uncertainty.	61
6.1	Hardware interface overview.	64
6.2	Node registration to ROS master.	65
6.3	ROS network overview.	66
6.4	<code>/mavros/local_position/pose</code> topic with Pixhawk as publisher and VFH* as subscriber.	66

6.5	<i>/lidarReadings</i> topic with Lidar-Lite as publisher and VFH* as subscriber.	66
6.6	<i>/newSetPoints</i> topic with VFH* as publisher and Setpoints node as subscriber.	67
6.7	<i>/mavros/setpoint_position/local</i> topic with Setpoints node as publisher and Pixhawk as subscriber.	67
6.8	Obstacle avoidance algorithm overview.	69
7.1	Multicopter with Intel Edison and Arduino encased with the Lidar-Lite mounted on-top.	72
7.2	Test setup with two panels used as obstacles.	72
7.3	Test 1 (gap size 5 m).	74
7.4	Test 2 (gap size 5 m).	74
7.5	Test 3 (gap size 10 m).	75
7.6	Test 4 (gap size 10 m).	76
7.7	Test 5 (gap size 8 m).	76
7.8	Multicopter approaching two obstacle panels.	77
7.9	Multicopter's path blocked by two obstacles due to the enlargement angle $\gamma_{i,j}$	77
A.1	Primary polar histogram constructed around robot.	83
A.2	Quadcopter lift manoeuvres.	83
A.3	Quadcopter performing pitch manoeuvres.	84
A.4	Quadcopter performing roll manoeuvres.	84
A.5	Quadcopter performing yaw manoeuvres.	85
B.1	Example of heuristic: Euclidean distance to the goal.	87
B.2	Example to illustrate how the A* search algorithm determines the optimal path to the goal node.	88
B.3	Priority queue expansion.	89
B.4	Determining the optimal path from priority queue.	89

List of Tables

4.1	VFH* algorithm simulation parameters.	47
7.1	VFH* algorithm test parameters.	73
7.2	Tests average error and standard deviations.	78

Nomenclature

Variables

c_n	Candidate direction (narrow)	[rad]
c_r	Candidate direction to the right	[rad]
c_l	Candidate direction to the left	[rad]
c_t	Candidate direction to target	[rad]
$d_{i,j}$	Distance between obstacle and RCP	[m]
d_r	Distance	[m]
d_l	Distance	[m]
d_{enlarged}	Enlarged range measurement	[m]
d_s	Step size	[m]
d_t	Total projected distance	[m]
k_e	Effective direction of motion	[rad]
k_t	Target direction	[rad]
r_{active}	Active window radius	[m]
r_{r+s}	Enlargement radius	[m]
r_{robot}	Robot radius	[m]
r_{safety}	Safety region	[m]
$r_{\text{multicopter}}$	Multicopter radius	[m]
r_{position}	Position uncertainty	[m]
r_{yaw}	Yaw uncertainty	[m]
r_{tilt}	Tilt uncertainty	[m]
s	Range measurement	[m]
x_o	Current x-coordinate	[m]
x_i	Projected x-coordinate	[m]
x_r	Right trajectory center x-coordinate	[m]
x_l	Left trajectory center x-coordinate	[m]
y_o	Current y-coordinate	[m]
y_i	Projected y-coordinate	[m]
y_r	Right trajectory center y-coordinate	[m]

y_l	Left trajectory center y-coordinate	[m]
α	Sector width	[rad]
$\beta_{i,j}$	Obstacle angle	[rad]
$\gamma_{i,j}$	Enlargement angle	[rad]
θ	Direction of motion	[rad]
θ_{tilt}	Tilt uncertainty	[rad]
θ_{yaw}	Yaw uncertainty	[rad]
μ	Average Error	[m]
σ	Standard deviation	[m]
ϕ_r	Right limit angle	[rad]
ϕ_l	Left limit angle	[rad]

Vectors and Tensors

F_r	Repulsive force vector
F_t	Attractive force vector
R	Resultant force vector

Subscripts

b	Binary
m	Masked
p	Primary
i	Row
j	Column
r	Right
l	Left
n	Narrow
s	Step
t	Target
e	Effective
o	Initial

Dimensionless Numbers

$c_{i,j}$	Cell certainty value	[]
g_0	Primary candidate direction cost	[]
g_i	Projected candidate direction cost	[]
H	Polar histogram	[]

h_i	Heuristic	[]
$m_{i,j}$	Magnitude	[]
n_g	Search depth	[]
s_{\max}	Sector threshold	[]
μ_1	Cost function first weight	[]
μ_2	Cost function second weight	[]
μ_3	Cost function third weight	[]
τ_{high}	High threshold	[]
τ_{low}	Low threshold	[]

Acronyms

CSP	Concentrated solar power
FCU	Flight control unit
GPS	Global positioning system
LIDAR	Light RADAR
MATLAB	Matrix laboratory
MCP	Multicopter center point
PC	Personal computer
POD	Polar obstacle density
RADAR	Radio detection and ranging
RCP	Robot center point
RTK	Real time kinematics
SONAR	Sound navigation and ranging
STERG	Solar thermal energy research group
VFH	Vector field histogram
VFH+	Vector field histogram: plus
VFH*	Vector field histogram: star
VFF	Virtual force field
WFM	Wall following method

Chapter 1

Introduction

In recent years multicopters have been implemented in many different applications. From search and rescue operations, photography, surveying, emergency aid and even multicopter racing, it would seem their uses are limited only by our own imagination. However, there are inherent dangers posed by multicopters. The loss of control over such a drone can easily result in injuries occurring from the falling mass or its spinning rotors. As safety concerns grow over the use of such devices the need for autonomous obstacle avoidance capabilities have become essential.

Autonomous robots are being used widely for various kinds of applications ranging from self-driving cars to humanoid robots such as the famous ASIMO robot manufactured by Honda. It is only reasonable to believe that even more novel applications will be found in the near future that require obstacle avoidance capabilities.

One application that is currently being investigated by the Solar Thermal Energy Research Group (STERG) at Stellenbosch University is the use of autonomous multicopters to inspect and calibrate heliostats in a concentrated solar power (CSP) plant. The goal of which is to reduce the costs associated with the manual inspection and calibration of heliostats.

Against the above-mentioned background and in order to ensure the safe movement of a multicopter in a CSP plant, the development and implementation of a prototype obstacle avoidance system will be investigated during this study. The latter will contribute towards the ongoing research in this field of study by STERG.

In the remainder of this chapter the use of multicopters in a CSP plant will be motivated in Section 1.1, the aims and objectives will be stated in Section 1.2 and the layout of the study will be discussed in Section 1.3.

1.1 Motivation

As the global demand for electricity increases it is important to find alternative resources for power generation. A lot of focus is being placed on utilising renewable energy to meet the global demand. A promising alternative renewable resource is the sun.

The sun's power can be utilised through concentrated solar power (CSP) plants like the 50 MW Khi Solar One central receiver power plant near Upington in the Northern Cape. A central receiver power plant uses mirrors called heliostats which reflect the light from the sun onto a central receiver tower as shown in Figure 1.1. A heat transfer fluid (HTF) is passed through the tower where it is heated to high temperatures (540 to 840°C) and then used to generate high pressure steam which is used in a turbine to generate electricity (Swinton & Campbell, 1920).

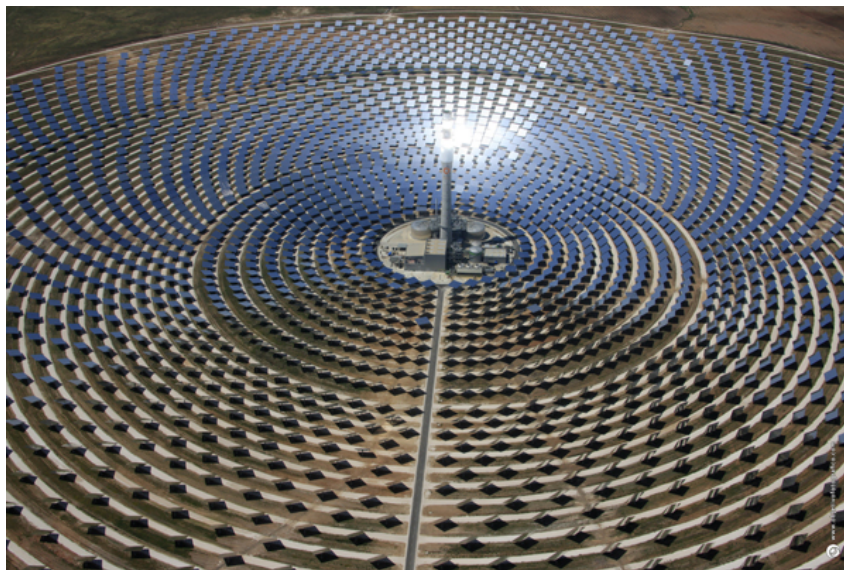


Figure 1.1: Gemasolar (19.9 MW) CSP power plant in Spain (Torresol, 2016).

At the moment one of the biggest challenges faced in the development of CSP plants is finding ways to reduce their costs. One of these CSP plants can have thousands of heliostats, as shown in Figure 1.1, that require cleaning and inspection on a regular basis. Currently this is done manually and is time consuming and expensive.

In order to address the problem at hand the use of multicopters for inspection of the heliostats is being investigated. But, for multicopters to be able to move about freely in the CSP plant and inspect heliostats some safety precautions will undoubtedly be required. Thus, the design of an obstacle avoidance system for multicopters is required. This will allow multicopters

to move about the plant freely and inspect heliostats without endangering themselves or anything in their surrounding environment.

1.2 Objectives

The main aim of the study is to develop a prototype obstacle avoidance system to be utilized by a multicopter to successfully avoid obstacles in a test environment. In order to realize the over-arching aim, the following objectives should be addressed:

- Explore literature on obstacle avoidance algorithms and relevant sensors, suitable for multicopters.
- Select a suitable obstacle avoidance algorithm as well as sensors to allow a multicopter to avoid obstacles in a two-dimensional environment.
- Code the selected algorithm and verify the code through simulations.
- Address the need for any modifications to the selected algorithm.
- Determine suitable hardware and software to implement the algorithm on a multicopter.
- Implement the prototype obstacle avoidance system on a multicopter in a test environment.

1.3 Layout of the Study

In Chapter 1 the context and reason for the study was explored and the over-arching aim as well as the objectives, to reach this aim, were formulated. In order to make informed decisions, a literature study has been undertaken regarding different obstacle avoidance algorithms and sensors in Chapter 2. The reasons for the selection of the VFH* algorithm and LIDAR sensor is also discussed in this chapter.

Chapter 3 provides a detailed discussion on the VFH* obstacle avoidance algorithm. The coding of the selected algorithm and verification of the code through simulations are discussed in Chapter 4.

Chapter 5 will address the modifications made to the VFH* algorithm to suit the needs of the multicopter application. Chapter 6 contains an overview of the hardware and software that were utilized to implement the VFH* algorithm on a multicopter. Chapter 7 discusses the test flights that were performed to confirm that the prototype obstacle avoidance system worked. The test environment as well as the data received from the tests are also discussed. Chapter 8 provides the findings, conclusions and recommendations of the investigation.

Chapter 2

Literature Review

There exists a wide variety of obstacle avoidance algorithms and sensors that can be utilized, depending on the needs of the obstacle avoidance application. In Section 2.1 the distinctions between the main types of obstacle avoidance algorithms are discussed. The most relevant obstacle avoidance algorithms are also discussed in more depth in the section. An overview of the different types of sensors used for obstacle avoidance and their applications are given in Section 2.2. The choice of the most suitable obstacle avoidance algorithm and the choice of sensor to implement the algorithm is provided in Section 2.1.9 and 2.2.5 respectively.

2.1 Obstacle Avoidance Algorithms

Obstacle avoidance can either be done locally, globally or through a combination of the two. A local obstacle avoidance system uses real-time sensor data to detect and avoid obstacles in the immediate vicinity of a robot. Examples of such algorithms include the curvature-velocity algorithm that makes use of constrained optimization in the robot's velocity space (Simmons, 1996). Algorithms such as the Bug-1, Bug-2, and Tangent Bug algorithms have also been used for local obstacle avoidance without the need to construct a configuration space (Howie Choset *et al.*, 2005). The potential function algorithm directs a robot as if it is a particle moving in a gradient vector field and directs the robot from a start position to its goal position (Howie Choset *et al.*, 2005).

Global obstacle avoidance systems determine the most appropriate path for a robot to follow based on a map of the robot's known environment (Ulrich & Borenstein, 2000). Such algorithms include the breadth-first, depth-first, A* and D* search algorithms (Howie Choset *et al.*, 2005). The use of artificial potential fields for global path planning for mobile robots was suggested by Warren (1989). This, however, requires that the global workspace be known at the time of planning.

Systems exist that combine local and global obstacle avoidance. A high

speed global dynamic window approach that combines real-time obstacle avoidance and global path planning was suggested by Brock & Khatib (1999). Another method that combines local and global obstacle avoidance is the VFH*.

Most algorithms presented here have been developed with mobile ground robots in mind though the principals are still relevant to this study. For the purpose of this literature review the algorithms will be discussed in terms of their applicability to robots in general. However, the term robot still includes, but is not limited to autonomous multicopters. Since multicopters in the current application will need to move in unknown environments only local obstacle avoidance algorithms were considered, with the exception of the VFH* algorithm which uses partial global path planning combined with local obstacle avoidance.

2.1.1 Bug-1 Algorithm

The Bug-1 algorithm uses two behaviours to navigate an obstacle course; a motion-to-goal and boundary-following behaviour. During motion-to-goal behaviour the robot moves towards the goal until an obstacle is encountered. The point where the robot encounters the obstacle is called the hit-point (Howie Choset *et al.*, 2005). At this point the robot switches to boundary-following behaviour which guides the robot around the obstacle. After the obstacle has been circumnavigated the closest point towards the goal on the perimeter of the obstacle, called the leave-point, is determined and the robot then traverses to that point (Howie Choset *et al.*, 2005). When the robot reaches the leave-point it switches back to motion-to-goal behaviour and continues in the direction of the goal once more until it either reaches the goal or another obstacle is encountered.

For example, Figure 2.1 shows a robot using the Bug-1 algorithm and contact sensors to avoid obstacles. There are two obstacles, WO_1 and WO_2 , between the robot and the goal. The robot is initially positioned at q_{start} and since no obstacles have been encountered it proceeds towards q_{goal} in motion-to-goal behaviour. The robot moves towards the goal until it encounters obstacle WO_1 at the hit-point q_1^H . The robot switches to boundary-following behaviour at this point and consequently circumnavigates the obstacle. After the obstacle has been circumnavigated the leave-point q_1^L on the obstacle perimeter is determined. Thereafter the robot traverses to the leave-point and then continues to move towards the goal once more. The process is repeated when the robot encounters the second obstacle WO_2 at hit-point q_2^H . The robot switches to boundary-following behaviour and circumnavigates the obstacle. Thereafter the leave-point q_2^L is calculated. Finally, the robot traverses to leave-point q_2^L and then continues to q_{goal} in motion-to-goal behaviour.

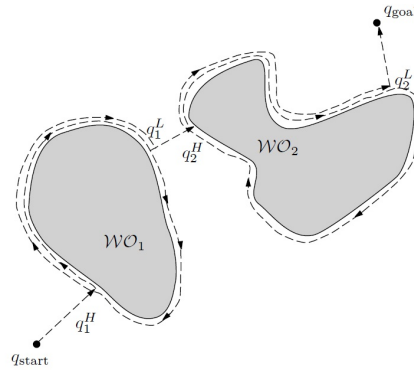


Figure 2.1: Obstacle avoidance using the Bug-1 algorithm (Howie Choset *et al.*, 2005).

The Bug-1 algorithm worked in the above case, however, there exist situations where no path towards the goal can be found. The Bug-1 algorithm identifies these situations by checking whether the line that connects the goal and leave-point intersects the obstacle after it has been circumnavigated. If it is found to intersect the obstacle no path to the goal can be found (Howie Choset *et al.*, 2005).

For example, in Figure 2.2 the robot is enclosed by an obstacle. The robot initially moves towards the goal in motion-to-goal behaviour and then encounters the obstacle at hit-point q_1^H . The robot switches to boundary-following behaviour and circumnavigates the obstacle. Thereafter the leave-point q_1^L is determined. However, since the line connecting q_{goal} and q_1^L intersects the obstacle, the algorithm is able to determine that no path to the goal exists.

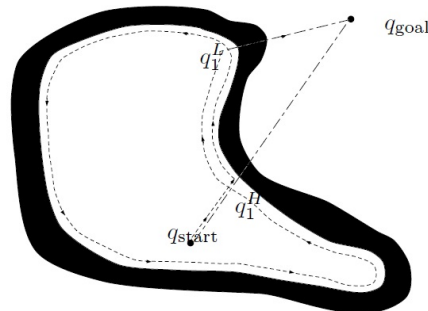


Figure 2.2: Situation in which the Bug-1 obstacle avoidance algorithm is unable to reach the goal (Howie Choset *et al.*, 2005).

2.1.2 Bug-2 Algorithm

Unlike the Bug-1 algorithm, the Bug-2 algorithm does not need to circumnavigate the obstacle to determine a leave-point. The algorithm works by calcu-

lating the shortest line that connects the robot start and goal positions, called the m -line. The robot, while in motion-to-goal behaviour, follows the m -line towards its goal position until an obstacle is encountered. When an obstacle is encountered the robot switches to the boundary-following behaviour and departs from the m -line. Here, the boundary-following behaviour guides the robot around the obstacle boundary until it can continue with its original path on the m -line, from a point closer to the goal than the initial point of contact with the obstacle (hit-point). The robot then switches back to motion-to-goal behaviour and proceeds towards the goal once more. In the case where the robot re-encounters the original departure point from the m -line (i.e. circumnavigates the obstacle) no path to the goal can be calculated (Howie Choset *et al.*, 2005).

For example, the concept of the Bug-2 algorithm is illustrated in Figure 2.3. The robot starts at q_{start} and moves towards the goal at q_{goal} following the m -line, which is the shortest distance between q_{start} and q_{goal} . The robot is in motion-to-goal behaviour until it encounters obstacle WO_1 at the hit-point q_1^H . It then switches to boundary-following behaviour and moves along the obstacle perimeter until it reaches the leave-point q_1^L from where the robot continues on its original path on the m -line. Next the robot encounters the second obstacle WO_2 at the hit-point q_2^H and consequently follows the obstacle boundary until it can continue on its path towards the goal at leave-point q_2^L on the m -line. Thereafter the robot continues to follow the m -line in motion-to-goal behaviour until it reaches the goal.

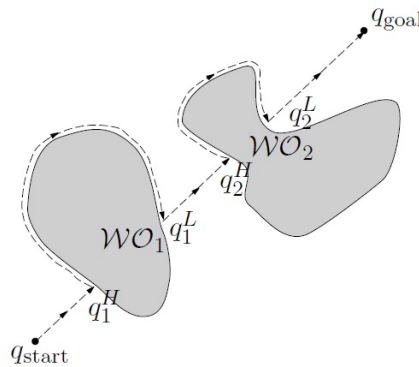


Figure 2.3: Obstacle avoidance using the Bug-2 algorithm (Howie Choset *et al.*, 2005).

It would seem that the Bug-2 algorithm is more efficient than the Bug-1 algorithm due to the fact that the Bug-2 algorithm does not require the robot to circumnavigate each obstacle and thus produces shorter path lengths in general. However, this is not always the case as certain scenarios exist in which the Bug-1 algorithm produces shorter path lengths than the Bug-2

algorithm. The path lengths of the two algorithms are compared to illustrate the latter.

The upper-bound for the path length $L_{\text{Bug-1}}$ of the Bug-1 algorithm can be calculated using Equation 2.1. The equation consists of two parts, the first part $d(q_{\text{start}}, q_{\text{goal}})$ is the direct distance between the robot's start and goal positions. In the second part of the equation the path length the robot traverses while circumnavigating obstacles is determined and is dependant on n , the number of obstacles, and p_i , the perimeter of each obstacle. A factor of 1.5 is used because the robot circumnavigates each obstacle once and then traverses to the leave-point on the perimeter of the obstacle. The distance to the leave-point will at most be half of the obstacle perimeter.

The maximum path length of the Bug-2 algorithm $L_{\text{Bug-2}}$ can be calculated using Equation 2.2. Again the direct distance between the robot's start and goal position $d(q_{\text{start}}, q_{\text{goal}})$ is the first term. Similarly to Equation 2.1 the second term in the Equation 2.2 is dependant on n , the number of obstacles, and p_i , the perimeter of each obstacle. Additionally, n_i is the number of times the m -line intersects each i 'th obstacle. The factor is 0.5 for the Bug-2 algorithm because half of the intersection points are not valid leave-points since half of the leave-points lie on the "wrong side" of the obstacle and would lead to collisions if the robot were to move straight towards the goal from these points (Howie Choset *et al.*, 2005).

$$L_{\text{Bug-1}} \leq d(q_{\text{start}}, q_{\text{goal}}) + 1.5 \sum_{i=1}^n p_i \quad (2.1)$$

$$L_{\text{Bug-2}} \leq d(q_{\text{start}}, q_{\text{goal}}) + 0.5 \sum_{i=1}^n n_i p_i \quad (2.2)$$

Although both equations are upper-bounds one can still observe that $L_{\text{Bug-2}}$ can be arbitrarily longer than $L_{\text{Bug-1}}$ (Howie Choset *et al.*, 2005). This will occur in situations where the m -line intersects an obstacle multiple times as is the case illustrated in Figure 2.4.

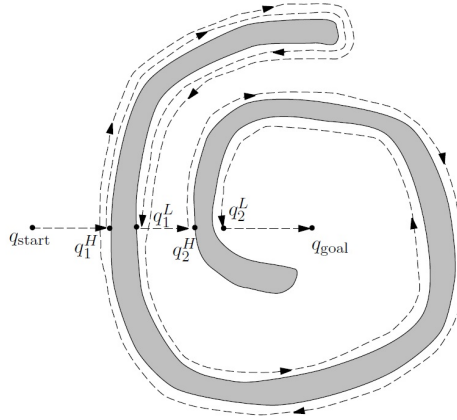


Figure 2.4: Situation where the Bug-2 algorithm is less efficient than the Bug-1 algorithm (Howie Choset *et al.*, 2005).

As the complexity of an obstacle increases so does the likelihood of the Bug-1 algorithm outperforming the Bug-2 algorithm. Therefore, Howie Choset *et al.* (2005) described the Bug-1 algorithms as an exhaustive search to find an optimal leave-point while the Bug-2 algorithm is an opportunistic approach since it takes the first valid leave-point available on the m -line. Despite the Bug-2 algorithm leading to shorter path lengths for simpler obstacles the more conservative Bug-1 algorithm often yields better performance (Howie Choset *et al.*, 2005).

2.1.3 Tangent Bug Algorithm

The Tangent Bug algorithm is an improvement on the Bug-2 algorithm, since it makes use of range sensors to determine a path to the goal, rather than arbitrarily moving around the obstacle boundary until a leave-point on the m -line is found (Howie Choset *et al.*, 2005). This results in a more efficient path to the goal. For instance, Figure 2.5 (left) shows a robot, with infinite sensor range, initially at position x reacting to the obstacles WO_1 and WO_2 . The O_i 's are the endpoints of the intervals detected by the robot's sensors. The robot moves towards the O_i that maximally decreases the heuristic distance to the goal (Howie Choset *et al.*, 2005). An example of such a heuristic distance is $d(x, O_i) + (O_i, q_{\text{goal}})$ where x is the robot's position (Howie Choset *et al.*, 2005). Hence the robot moves towards O_2 because it minimizes $d(x, O_i) + (O_i, q_{\text{goal}})$.

Figure 2.5 (right) shows a similar situation as before except that q_{goal} has been moved. The robot in this case moves to the point O_4 although O_2 would have been optimal if it was possible to travel directly to q_{goal} afterwards. However, due to the robot's infinite sensor range it can determine that WO_2 would cause the robot to move from O_2 to O_3 and then to q_{goal} . Thus, even though the distance travelling from O_2 to the target when neglecting WO_2 is less than

the distance of first moving to O_4 the algorithm assigns an infinite cost to O_2 since it will produce a suboptimal path due to the presence of WO_2 .

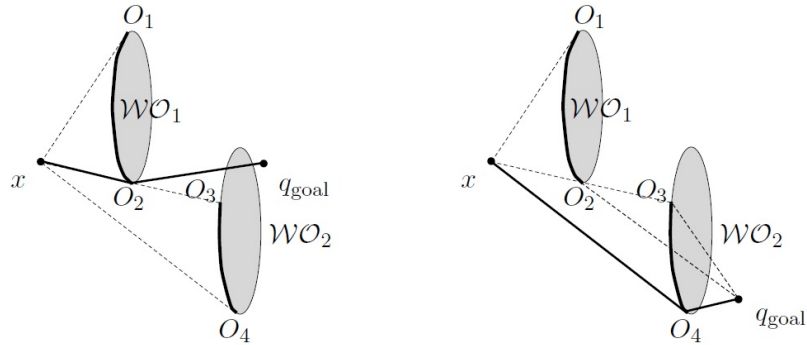


Figure 2.5: (Left) Tangent Bug algorithm selects O_2 as subgoal for the robot (Howie Choset *et al.*, 2005). (Right) Tangent Bug algorithm selects O_4 as subgoal for the robot (Howie Choset *et al.*, 2005).

The Tangent Bug algorithm's motion-to-goal and boundary-following behaviours differ from the Bug-1 and Bug-2 implementations. The motion-to-goal behaviour in this case directs the robot towards the goal but might have a phase where the robot follows the boundary, while the boundary-following behaviour may have a phase where the robot does not follow the obstacle boundary (Howie Choset *et al.*, 2005). For example, Figure 2.6 shows a robot moving between two obstacles using 360° range sensors, with a finite range R , shown as a dotted circle around the robot. As the robot initially approaches the goal in motion-to-goal behaviour the circle around the robot becomes tangent to the obstacle WO_1 . Thereafter the tangent point splits into two O_i 's which are the endpoints of the interval of the circle that touches the obstacle. The robot moves towards the O_i that maximally decreases the heuristic distance to the goal, O_2 in this case. Since obstacle WO_2 does not block the robot's path towards the goal it has no effect on the motion-to-goal behaviour. The robot stays in the motion-to-goal behaviour until it either has a clear path towards the goal or it reaches a local minimum where no point O_i minimizes the heuristic distance to the goal.

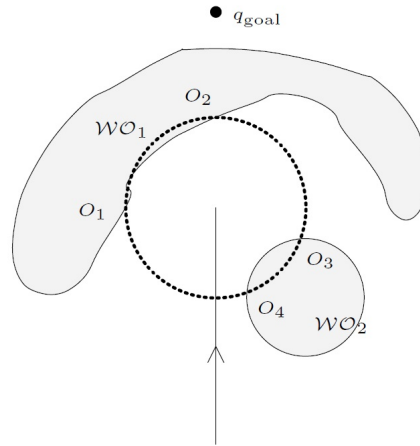


Figure 2.6: Robot moving towards the goal with dotted circle representing the sensor detection range (Howie Choset *et al.*, 2005).

The behavioural changes of the Tangent Bug algorithm are illustrated in Figure 2.7. The robot is initially in motion-to-goal behaviour heading towards the goal as shown by the solid line. When the obstacle is detected the robot continues in motion-to-goal behaviour and moves in the direction that minimizes the heuristic distance to the goal (O_2 in Figure 2.6) as shown by the dashed lines in Figure 2.7. The robot reaches a local local minimum at point M and switches to boundary-following behaviour. In boundary-following behaviour the robot chooses a direction to follow around the obstacle and continues to move around the obstacle while updating the values d_{followed} and d_{reach} . The value d_{followed} is the shortest distance between the goal and the known boundary while d_{reach} is the distance between goal and closest point on the obstacle that is within line of sight of the robot (Howie Choset *et al.*, 2005). When $d_{\text{reach}} < d_{\text{followed}}$ the robot terminates the boundary-following behaviour and switches back to motion-to-goal behaviour.

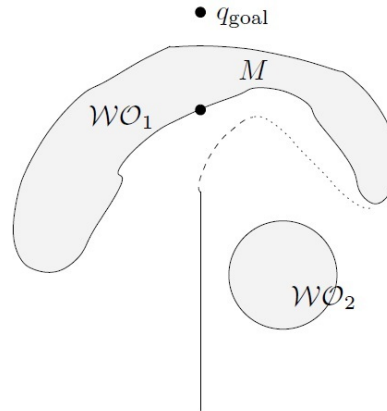


Figure 2.7: Robot initially in motion-to-goal behaviour before obstacle is detected (solid path), then moves to minimize heuristic distance to goal in motion-to-goal behaviour (dashed path), encounters local minimum and switches to boundary-following mode (dotted path) (Howie Choset *et al.*, 2005).

The Tangent Bug algorithm is affected by the range of its sensors. Two cases of the Tangent Bug algorithm are used to illustrate the effect of sensor range on the algorithm. In the first case the robot uses contact sensors (zero sensor range) and in the second case the robot's sensors have infinite range.

The first case where only contact sensors are used is illustrated in Figure 2.8. The robot, in motion-to-goal behaviour, moves towards the goal until it encounters the first obstacle at hit-point H_1 , from where it moves around the obstacle and departs from the obstacle boundary at depart-point D_1 . Next the robot moves towards the goal until it encounters the second obstacle at H_2 , from where it continues to move around the obstacle boundary in motion-to-goal behaviour in order to minimize the heuristic distance of the robot to the goal. This process continues until the robot eventually reaches a local minimum at M_3 where the robot is unable to minimize the heuristic distance any more. At this point the robot switches to the boundary-following behaviour until it reaches the leave-point at L_3 where $d_{\text{reach}} < d_{\text{followed}}$ is satisfied. Subsequently, the robot moves towards q_{goal} until it encounters another obstacle at hit-point H_4 . The robot continues in motion-to-goal behaviour until it reaches a local minimum at M_4 from where it switches to boundary-following behaviour. Lastly, the robot reaches leave-point L_4 , where $d_{\text{reach}} < d_{\text{followed}}$ is satisfied, and continues to q_{goal} .

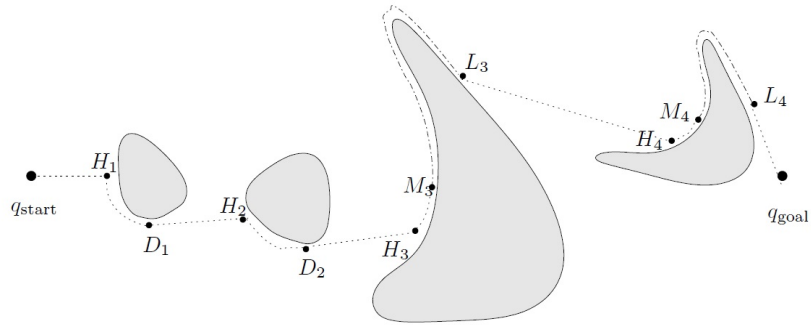


Figure 2.8: Tangent Bug algorithm with sensor range of zero. Motion-to-goal behaviour is indicated with dotted lines, while boundary-following behaviour is indicated with dashed lines (Howie Choset *et al.*, 2005).

In the second case, illustrated in Figure 2.9, the robot's infinite sensor range allows it to react immediately to the first obstacle and thus q_{start} and H_1 coincide. The robot moves around the obstacle and departs from it at departure-point D_1 which also coincides with hit-point H_2 . Then in the same manner as before the robot moves around the second obstacle and departs from it at D_2 , which coincides with hit-point H_3 . Next the robot is able to move around the third obstacle without falling into a local minimum as was the case when contact sensors were used. Finally, the robot departs from the third obstacle at D_3 and continues to the goal.

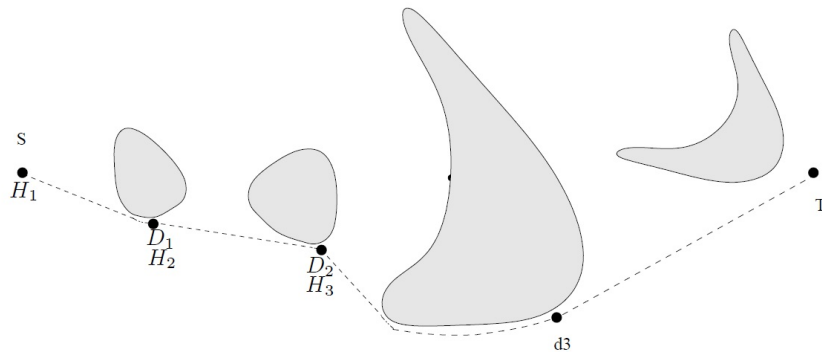


Figure 2.9: Tangent Bug algorithm with infinite sensor range (Howie Choset *et al.*, 2005).

2.1.4 Potential Field Algorithms

Potential field algorithms use potential functions to assign a gradient vector to each point on the manifold (Howie Choset *et al.*, 2005). Intuitively gradients can be viewed as forces acting on the robot. The robot is seen as a positive charge that is attracted to the negatively charged goal. Obstacles are seen as positive charged and thus obstacles induce repulsive forces on the robot

(Borenstein & Koren, 1991b). The sum of the attractive force to the goal and the repulsive forces from obstacles guide the robot to the goal under ideal conditions.

Alternatively potential functions can be viewed as landscapes (Howie Choset *et al.*, 2005). The robot moves across the landscape from regions with "high" values to regions with "low" values. This is similar to a rock rolling downhill from a high elevation to a lower elevation. When the robot follows such a path it is often referred to as gradient descent (Howie Choset *et al.*, 2005).

Potential field algorithms are however susceptible to local minimums and can thus become "stuck" in one position. A local minimum is a point where all gradient values around the robot's current position is higher than the gradient value where the robot resides (Howie Choset *et al.*, 2005).

2.1.5 The Virtual Force Field (VFF) Algorithm

Borenstein & Koren (1989) developed a real-time obstacle avoidance algorithm that steers a robot towards a goal position while avoiding obstacles. The algorithm, entitled the virtual force field (VFF), makes use of a histogram grid for obstacle representation and potential fields for navigation (Borenstein & Koren, 1989). It was developed for fast mobile robots and addresses some local minimum situations by implementing wall following techniques (Borenstein & Koren, 1989). The VFF algorithm consists of three parts:

Part 1 - A two-dimensional Cartesian histogram grid C is used to represent obstacles. Each cell (i, j) within the histogram grid has its own certainty value $c(i, j)$. A cell's certainty value represents the confidence that an obstacle resides within that cell (Borenstein & Koren, 1991b). Only one cell's certainty value is increased for each range reading. For example, when ultrasonic sensors are used the certainty value of the cell that is located on the acoustic axis of the sensor at a measured distance is increased as shown in Figure 2.10. A probabilistic distribution is obtained by continuously sampling each sensor at high speed while the robot is moving. This means that the cell on the acoustic axis and its neighbouring cells' certainty values are repeatedly increased as shown in Figure 2.10.

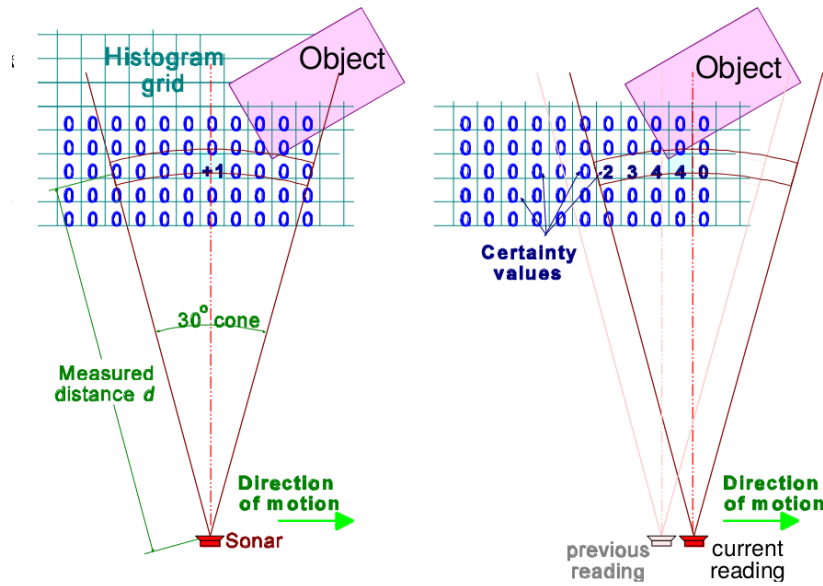


Figure 2.10: Updating of histogram grid using range data from an ultrasonic sensor (Borenstein & Koren, 1991b).

Part 2 - The potential field algorithm is used to steer the robot by using the histogram grid's probabilistic sensor information as illustrated in Figure 2.11. A region, called the active window, is centred around the robot's momentary location. The active window, denoted as C^* , moves with the robot as it moves about the histogram grid C . Even though a circular window would be more appropriate geometrically a square window is less computationally intensive (Borenstein & Koren, 1989). The cells within the active window C^* are called active cells and are denoted as $c^*(i, j)$. Each active cell is assigned a virtual repulsive force $F(i, j)$ that is directed towards the robot. The magnitude of each force is proportional to each active cells' certainty value $c^*(i, j)$ and inversely proportional to d^x , where d is the distance between an active cell and the center of the robot. x is a positive real number and was chosen to be 2 for implementation on a mobile ground robot by Borenstein & Koren (1989). At each point in time the active cells' repulsive forces are summed together to form a single repulsive force vector \mathbf{F}_r . A virtual attractive force vector \mathbf{F}_t is assigned to guide the robot towards the goal position. By summing the virtual attractive and repulsive force vectors a resultant force vector \mathbf{R} is calculated. The resulting force vector \mathbf{R} is then used to steer the robot towards its target position while avoiding obstacles as shown in Figure 2.11.

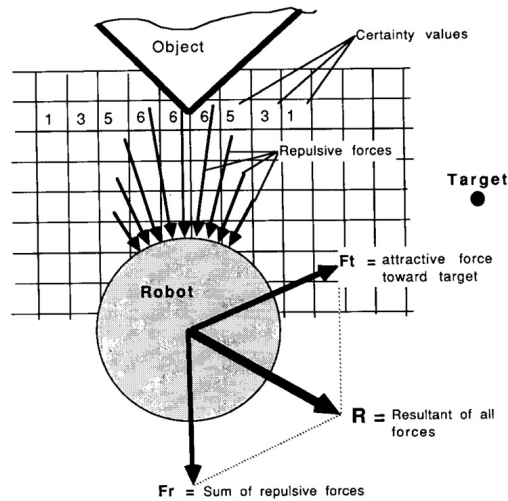


Figure 2.11: Robot being steered towards the target (goal) by the VFF algorithm (Borenstein & Koren, 1989).

Part 3 - Combining the procedures of Part 1 and 2 in real-time allows sensor data to have an immediate effect on the steering control of the robot. In practice each range reading is used to update the histogram grid and subsequently calculate the resulting force vector \mathbf{R} (Borenstein & Koren, 1989). This allows the robot to respond quickly when obstacles appear suddenly and results in the robot displaying fast reflexive behaviour which is important at high speeds (Borenstein & Koren, 1989).

However, there are some shortcomings of the VFF algorithm. The robot can still get stuck in local minimums under certain conditions. For example, when an obstacle is in the direct path of the robot as it moves towards the goal as shown in Figure 2.12. In this scenario the robot initially moves towards the goal as the repulsive force is less than the attractive force and thus the resultant force vector is towards the goal. As the robot moves closer towards the obstacle the repulsive force from the obstacle becomes greater until the resultant force vector either becomes zero or changes direction by 180° . If the resultant force vector changes direction oscillatory motion ensues where the robot keeps on moving back and forth towards and away from the goal.

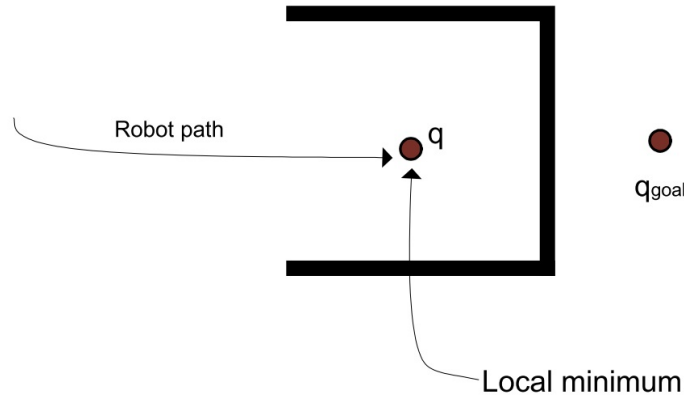


Figure 2.12: Robot using VFF algorithm encounters a local minimum: u-shaped obstacle (Zohaib *et al.*, 2013).

To address the problem of local minimums the wall following method (WFM) is used to guide the robot around obstacles when a local minimum has been encountered (Borenstein & Koren, 1989). The WFM works by first summing all the repulsive forces into one repulsive force vector \mathbf{F}_r as shown in Figure 2.13. Then a new virtual attractive force is created that replaces the original attractive force towards the goal temporarily while the robot is in WFM. The direction of the virtual attractive force is determined by either adding or subtracting an angle α to or from the direction of \mathbf{F}_r . This will guide the robot around the obstacle either to the left or right of the obstacle in a direction parallel to the obstacle boundary at a fixed distance from the wall (Borenstein & Koren, 1989). Borenstein & Koren (1989) suggested that α be chosen according to the condition: $90^\circ < \alpha < 180^\circ$. In their work α was chosen to be 145° . The robot stays in WFM until the difference in the robot's direction of motion and the target direction becomes less than 90° .

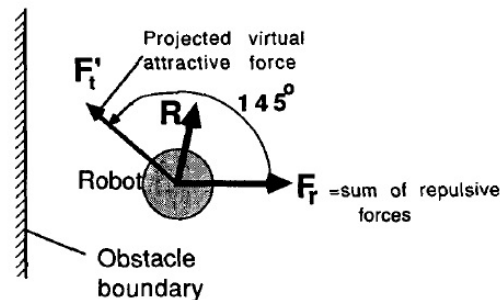


Figure 2.13: Wall following method (WFM) concept (Borenstein & Koren, 1989).

Additionally, when two obstacles are placed close to one another (e.g. a doorway) the VFF algorithm does not allow the robot to pass through the

doorway. This is because the two sides of the doorway cause repulsive forces that push the robot away as shown in Figure 2.14. This is similar to the above scenario, however in this case the robot might be able to progress towards the goal if it was not for the repulsive forces. In this case the WFM would guide the robot around one of the two obstacles and then allow the robot to continue to the goal. However, in cases where the obstacles are larger or tedious to circumnavigate the WFM might be unable to guide the robot around the obstacle. In such cases the robot would not reach the goal even though a clear path to the goal was initially available.

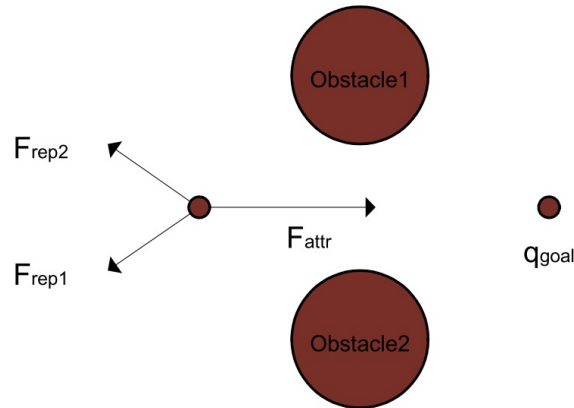


Figure 2.14: Robot using VFF algorithm encounters a local minimum: two obstacles with a gap in between (Zohaib *et al.*, 2013).

Furthermore, when the robot travels through a narrow corridor and the robot strays slightly to either side of the centreline between the corridor's walls the robot experiences a strong repulsive force from the closer wall. This causes the robot to be pushed to the other side of the centreline and the process is repeated with the other wall. Borenstein & Koren (1991b) noticed that under the right conditions this resulted in unstable oscillatory motion of the robot.

2.1.6 Vector Field Histogram (VFH) Algorithm

After careful analyses, the vector field histogram (VFH) algorithm was developed by Borenstein & Koren (1991b) to improve on the VFF. The inherent problem of the VFF algorithm was found to be a drastic reduction of data that occurs when the repulsive forces from the histogram grid and attractive force towards the goal are summed to calculate a resultant force vector (Borenstein & Koren, 1991b). Consequently, in only one step hundreds of data points are reduced to only a single vector (Borenstein & Koren, 1991b). As a result detailed information about the obstacle distribution around the robot is lost.

The VFH algorithm uses a two-stage data reduction process instead of the single-step process used in the VFF algorithm. The VFH algorithm uses three levels to represent data:

Highest level - a two-dimensional Cartesian histogram grid C is used to describe the robot's environment as with the VFF algorithm. On-board range sensors are used to continuously sample range data to create a real-time model of the robot's environment. An example of an obstacle course with the robot located between various obstacles is shown in Figure 2.15. The robot's on-board range sensors are used to build a two-dimensional Cartesian histogram grid of the robot's environment as shown in Figure 2.16. Cells with larger certainty values are shown with greater intensities and indicate that there is a greater chance of an obstacle residing in that cell location.

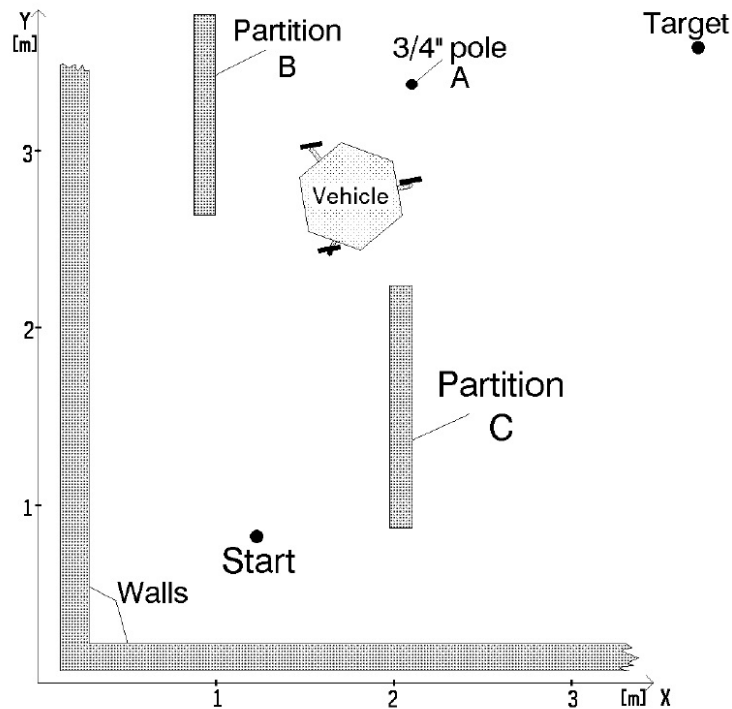


Figure 2.15: Robot surrounded by obstacles (Borenstein & Koren, 1991b).

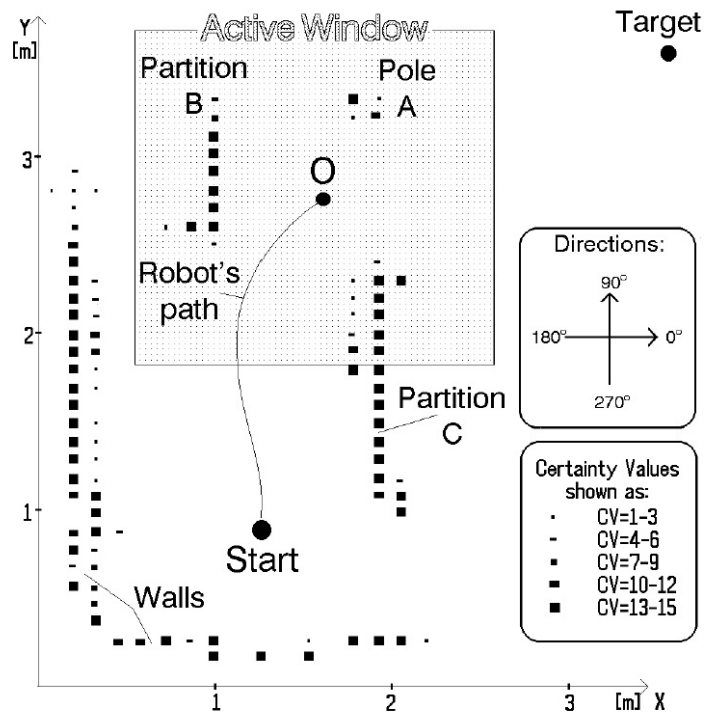


Figure 2.16: Histogram grid representation of the robot's environment (Borenstein & Koren, 1991b).

Intermediate level - the active region C^* , which is centred around the robot and lies on the two-dimensional Cartesian histogram grid C , is used to construct a one-dimensional polar histogram \mathbf{H} as shown in Figure 2.17. \mathbf{H} is located around the robot's momentary location and consists of n angular sectors of width α . The certainty values of the cells in each sector of the active region around the robot is used to construct \mathbf{H} . This allows each sector k to have a value that is representative of the polar obstacle density (POD) in the direction of that sector.

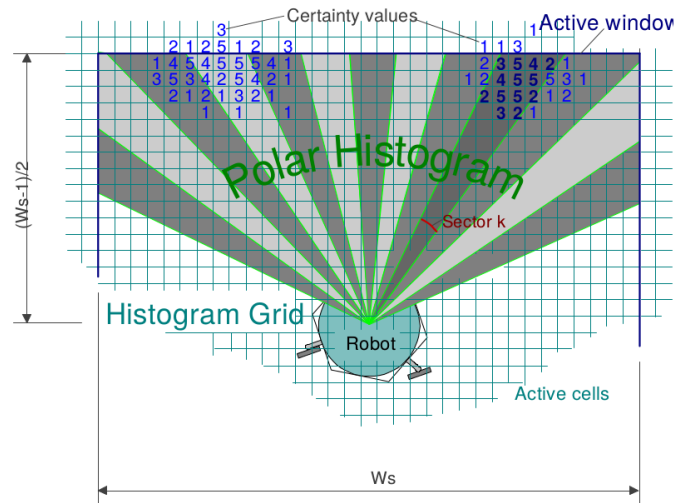


Figure 2.17: Construction of polar histogram by mapping active cells to sectors (Borenstein & Koren, 1991b).

The polar histogram, showcasing the polar obstacle densities, is drawn around the robot as shown in Figure 2.18. The sectors form a 360° circle around the robot. The obstacle densities for each sector is also shown as a histogram in Figure 2.19. Additionally, the threshold level which is used to assess whether the POD for each sector represents an obstacle or noise is indicated on both figures. If a sector's POD is above the threshold the sector is deemed to be "closed" or inaccessible, while on the other hand if the sector's POD is below the threshold the sector is deemed "open" or accessible to the robot.

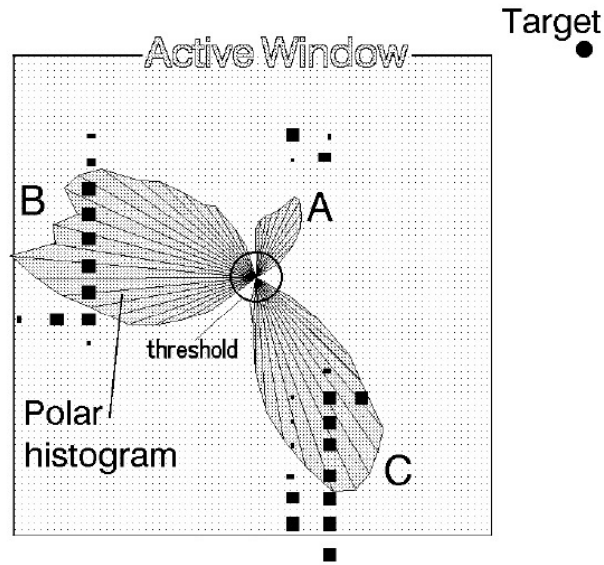


Figure 2.18: Polar histogram around robot's current position with threshold indicated (Borenstein & Koren, 1991b).

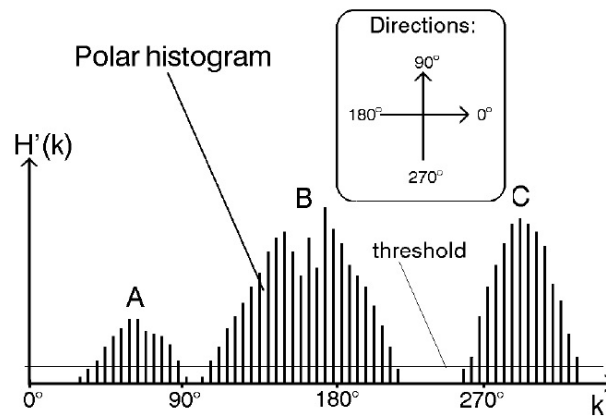


Figure 2.19: Determining candidate sectors from polar histogram using thresholds (Borenstein & Koren, 1991b).

Lowest level - the output of the VFH algorithm, namely the reference values for the drive and steering controllers of the robot are calculated from the polar histogram. Candidate directions are identified from the open sectors in the polar histogram grid as shown in Figure 2.20. The new direction of motion is selected from the candidate direction based on which candidate direction better guides the robot towards the target.

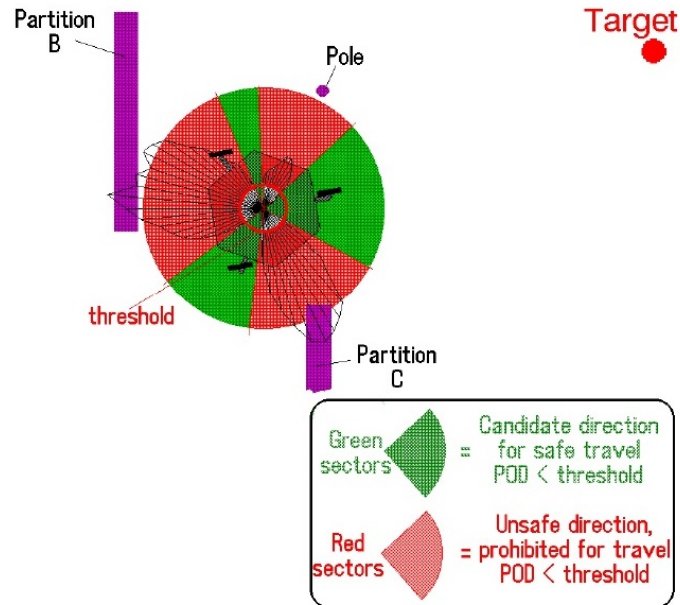


Figure 2.20: Polar histogram: candidate steering directions (Borenstein & Koren, 1991*b*).

2.1.7 Vector Field Histogram (VFH+) Algorithm

The vector histogram plus (VFH+) algorithm improves on the VFH algorithm by taking into account the robot's width and available trajectories. To accomplish this the VFH+ algorithm uses a four-stage data reduction process rather than the two-stage data reduction used by the VFH algorithm (Ulrich & Borenstein, 1998). In the **first stage** the VFH+ algorithm creates an one-dimensional primary polar histogram, that accounts for the robot's width, from the two-dimensional Cartesian histogram grid. To account for the robot's width each obstacle cell in the active region is enlarged when the primary polar histogram is created. Each obstacle cell is enlarged by the robot's width r_{robot} and an additional safety region r_{safety} . Thus, the safety region around the obstacle is $r_{\text{r+s}}$ where $r_{\text{r+s}} = r_{\text{robot}} + r_{\text{safety}}$ as shown in Figure 2.21.

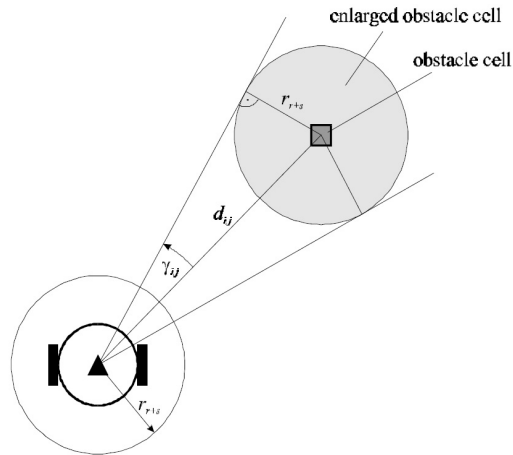


Figure 2.21: Obstacle being enlarged by robot width and safety region (Ulrich & Borenstein, 1998).

In the **second stage** the primary histogram grid, shown in Figure 2.22 a), is converted to a binary polar histogram grid by using thresholds to make sectors either free or blocked based on their polar obstacle densities as shown in Figure 2.22 b).

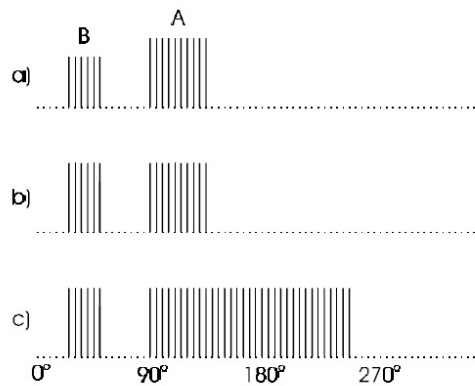


Figure 2.22: Histogram representation of a) the primary polar histogram, b) binary polar histogram and c) the masked polar histogram (Ulrich & Borenstein, 1998).

In the **third stage** the binary polar histogram is converted to a masked polar histogram by accounting for the trajectories that the robot can follow. This is done to account for the fact that most robots cannot change their direction of motion instantly, as shown in Figure 2.23 a), but rather are subject to turn circles or trajectories based on the robot's dynamics and speed, as shown in Figure 2.23 b). This reduces the possible steering directions the algorithm can choose from as shown in Figure 2.22 c). In the **fourth stage**

the VFH+ algorithm selects the appropriate direction in which to steer the robot based on the candidate directions available.

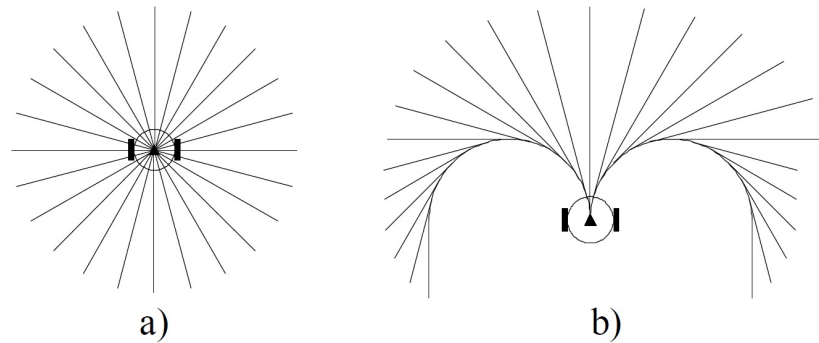


Figure 2.23: Robot trajectories (Ulrich & Borenstein, 1998).

2.1.8 Vector Field Histogram Star (VFH*) Algorithm

Further improvements to the VFH+ algorithm has lead to the development of the vector field histogram star (VFH*) algorithm. The VFH* algorithm addresses situations that are problematic for purely local obstacle avoidance algorithms. The algorithm works by assessing candidate directions in advance in order to choose the best one to guide the robot towards the goal. For example, in Figure 2.24 a robot is shown using the VFH+ algorithm to avoid obstacles. The obstacles are shown in black and the grey areas around the obstacles are the configuration space. The configuration space is added around the obstacles to take into account the robot's width. The dashed circle around the robot represents the robot's active region. Two candidate directions A and B have been identified by the VFH+ algorithm. Because the VFH+ algorithm only considers the active region when calculating the candidate directions it will on average only select the appropriate direction (B) 50% of the time.

The VFH* algorithm solves this problem by combining the VFH+ algorithm with the A* search algorithm to project the robot's trajectory several steps ahead and evaluate the consequences. Thus, the robot is able to look ahead at the robot's subsequent steps and evaluate each candidate direction based on heuristic and cost functions. In the case shown in Figure 2.24 the algorithm would be able to determine that candidate direction A would lead to a dead-end while candidate direction B can be succeeded by candidate direction C which decreases the heuristic and cost functions used to choose the most appropriate path. Consequently, the VFH* algorithm would choose candidate direction B.

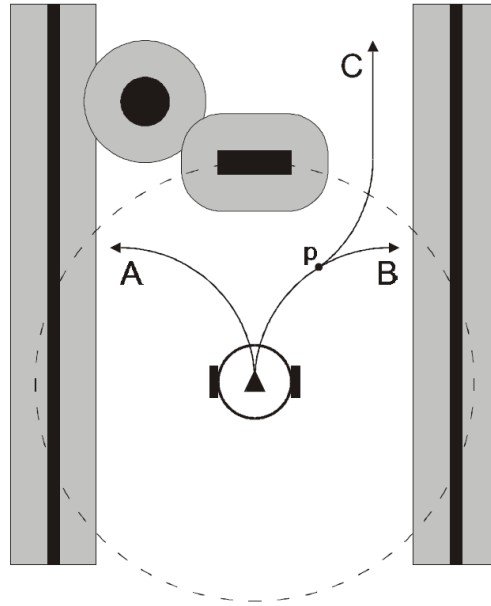


Figure 2.24: Robot using look-ahead verification to determine optimal path (Ulrich & Borenstein, 2000).

2.1.9 Conclusion: Obstacle Avoidance Algorithm

For this study an obstacle algorithm was required that considers the robot's width and dynamics and is not prone to get stuck in local minimums. Bug algorithms did not fulfil this criteria since they arbitrarily follow obstacle contour. This makes them prone to fall into local minimums. Although the Tangent Bug algorithm does try to minimize some heuristic distance to find its way to the goal position and addresses local minimums it does not consider certain constraints such as the robot's width.

Potential field algorithm also have a tendency to get caught up in local minimums and were thus not considered. The VFF algorithm does use the WFM to address local minimums but is unable to move through narrow openings and thus takes sub-optimal paths around obstacles even when a path to the goal is available.

The VFH* algorithm was selected as the algorithm best suited for this study for the following reasons. The VFH* algorithm considers the robot width which gives it an advantage over the VFH algorithm. Additionally the use of a partial global path planner, the A* algorithm to verify its route which enables it to successfully navigate obstacles that would be troublesome for the VFH+ algorithm. A more in depth description of the VFH* algorithm will follow in Chapter 3.

2.2 Sensors

In order for an obstacle avoidance algorithm to avoid an obstacle the obstacle first needs to be accurately detected. In the case of the VFH* algorithm the range readings from a sensor are used to update the histogram grid which is used by the algorithm to determine and select the most suitable candidate direction. An overview of the different types of sensors that can be used to detect obstacles and update the histogram grid are discussed next with examples of their real-life applications.

2.2.1 Ultrasonic Sensors (SONAR)

Sound navigation and ranging, or SONAR, is a commonly used method for obstacle detection. SONAR works by using echoes to detect obstacles. Sound waves are transmitted by an ultrasonic sensor. These ultrasonic sound waves reflect off obstacles in their path and some of these reflected echoes are reflected back to the sensor that made them. By measuring the time intervals between transmitted and received sound waves the obstacles' distances can be approximately calculated.

A number of researchers have used ultrasonic sensors for obstacle avoidance applications. Navarro-Serment *et al.* (1999) used ultrasonic sensors to create a localization system for a team of robots. The robots explore an unknown environment and use range data from the ultrasonic sensors to estimate their positions relative to one another and their environment. Soto *et al.* (1999) used sonar as a safety system for autonomous All Terrain Vehicles (ATVs) used for reconnaissance and surveillance. The ultrasonic sensors were used to detect nearby obstacles that could be overlooked by the vision system they implemented.

E.Prassler *et al.* (1999) created a robotic wheelchair MAid (Mobility Aid for elderly and disabled People) to assist the elderly and disabled in rapidly changing environments such as shopping malls, airports or railway stations where large amounts of people and objects move around. The robotic wheelchair uses a sonar system and laser range finder to perceive its surrounding environment. Similarly Thrun *et al.* (1999) describes two robot systems which were deployed as interactive tour-guides in two museums. Both these robots, the Rhino and Minerva used ultrasonic sensors along with other sensors to avoid obstacles.

The following shortcomings of ultrasonic sensors were identified by Borenstein & Koren (1991b):

- **Poor directionality** limits the accuracy to which the spatial position of an edge can be detected to 10-50 cm. The accuracy is dependent on the distance from the sensor to the obstacle and the angle from the sensor's acoustic axis to the surface of the obstacle.

- **Frequent misreadings** due to ultrasonic noise caused by external sources or stray reflections from other ultrasonic sensors, called crosstalk. Because these types of misreadings cannot always be filtered out they cause the false detection of edges.
- **Specular reflections** is when an obstacle is not detected or partially detected due to the sensor's ultrasound being reflected away from the sensor. This happens when the angle between an ultrasonic wave front and the normal of a smooth surface is too large.

2.2.2 RADAR

Radio detection and ranging, or RADAR as it is more commonly known, is an obstacle detection system. RADAR systems makes use of radio waves to determine the range, angle and even the velocity of objects. A RADAR system transmits radio waves that are reflected of any obstacle in the path of the radio waves. The reflected radio waves are received by a receiver and processed to determine the object's properties. Normally the radar transmitter and receiver make up one physical system.

In general these systems are quite expensive and thus not many researchers have access to such systems. One application of such a system was the use of a short-range millimetre-wave RADAR in a polar environment. The harsh polar terrain holds many restrictions for obstacle avoidance sensors, such as flying ice and snow, changing illumination and lack of contrast which degrades stereo and laser sensing (Foessel *et al.*, 1998). However, since it is not affected by the mentioned environmental conditions millimetre-wave RADAR was efficient.

2.2.3 LIDAR - Scanning Laser

LIDAR systems works in a similar fashion to RADAR and thus the term LIDAR was created as a portmanteau of the terms light and RADAR. A laser beam is emitted by a LIDAR sensor and then reflected from obstacle surface. The time between transmission and reception of the laser beam is used to calculate the obstacle's distance from the scanner (Ye & Borenstein, 2002).

Ye & Borenstein (2002) identified three main types of LIDAR systems. The first is a two-dimensional LIDAR system that scans for obstacles in one plane. The second system is a three-dimensional LIDAR system that scans in a similar fashion than what a two-dimensional LIDAR system does but adds a "nod" movement to add another dimension to the scan. The third type of system is Flash LIDAR which has no moving parts and rather focuses on producing a range image of a specific area.

Fröhlich & Mettenleiter (2004) and Fröhlich *et al.* (2000) used laser scanners for terrestrial surveying and three-dimensional modelling of real world environments, respectively. Apostolopoulos (2000) used an autonomous robot

to search for meteorites in Antarctica. The autonomous robot named Nomad used a combination of stereo-vision and laser range finding for obstacle avoidance and navigation.

The autonomous robot Nomad was tested under several weather (bright sun, various degrees of cloudiness and snow) and terrain (rocky regions and ice fields) conditions. The stereo-vision was tested on snow, blue ice and rocky terrain under clear, overcast and snowy conditions. Apostolopoulos (2000) found that under all conditions mentioned the stereo-vision system could not find enough texture in the scene to create disparity maps dense enough for navigation. A single-line-scan laser unit was tested under the same conditions as the stereo system. It was found that terrain type had no effect on the laser. The only scenario in which the laser system encountered problems was during periods of blowing snow. The laser beam was reflected off snow flakes causing either misreadings when the beam was reflected away from the laser system or incorrect range readings when the beam was reflected back to the laser unit.

Ye & Borenstein (2002) believed that the best choice to create a three-dimensional map would be a three-dimensional LIDAR system, however due to the costs involved with these systems they rather made use of a two-dimensional LIDAR aimed forward and downward at the front end of a mobile robot. The two-dimensional LIDAR proved to be a cost-effective alternative for the three-dimensional LIDAR system.

2.2.4 Vision-Based Approaches

A wide variety of vision-based approaches to obstacle avoidance and navigation exist. Such approaches include stereo-vision, visual match-making algorithms, and the use of a monocular camera for exploration of unknown environments. Some examples of where these approaches have been implemented follows.

Bischoff (1999) created the humanoid service robot HERMES to act as an experimental platform for research into human-friendly man-machine interaction. The robot's main sensor was stereo-vision used for mobile manipulation, environmental exploration and navigation. The robot was however restricted to working environments where the floor does not have bright reflections, shadows, or big patterns on it.

Matsumoto *et al.* (1999) proposed a visual view-based navigation method. This method called the View Sequence uses an existing model of the route consisting out of front views memorised during an initial teaching run. This concept was further developed and extended to include an omnidirectional vision sensor and was called the Omni-View Sequence. The inclusion of an omnidirectional sensor allowed for improved accuracy and navigation, more robust match making and it enabled the robot to return to its starting position back tracking on the path it travelled. The robot was however restricted to indoor use. Matsumoto *et al.* (2003) further developed this idea by introducing a map system named the view-sequenced map. This map is automatically

created based on the robot's exploration of a corridor using stereo and omnidirectional vision.

Santosh *et al.* (2008) proposed a novel image-based exploration algorithm using only a monocular pan-tilt camera. The robot is capable of autonomously exploring typical indoor environments without any prior knowledge thereof.

2.2.5 Conclusion: Sensors

For this study a sensor was required that is not expensive, has range capabilities in the range of 10 m to 20 m and can be used outdoors in various environmental conditions. The sensors discussed in this chapter were all evaluated at the hand of this criteria.

Vision-based sensors are not overly expensive and could potentially fulfil the range requirements. However vision-based approaches are not always reliable in all weather conditions and environments; in most cases the use of these sensors have been limited to indoor environments. Thus, these type of sensors were not considered.

SONAR sensors are relatively cheap, however, they have limited range capabilities and were thus not considered. RADAR on the other hand has great range capabilities and can function in most environmental conditions. However, RADAR systems were too expensive for this study and thus not considered.

It was decided to use a two-dimensional LIDAR sensor for this study since LIDAR sensors work well in most environmental conditions. Additionally, LIDAR met the range requirements required and was less expensive than RADAR.

Chapter 3

VFH* Algorithm

The VFH* algorithm was selected as the obstacle avoidance algorithm for this study. A detailed overview of the VFH* algorithm is presented in this chapter.

To determine the primary candidate directions in which the robot can be steered to avoid colliding with obstacles the VFH* algorithm makes use of a four stage data reduction process. The algorithm compensates for a robot's width, trajectory and a safety region when determining candidate directions for the robot to move in. The algorithm also makes use of look-ahead verification to select the optimal candidate direction.

In Section 3.1 the manner in which the histogram grid is updated to represent obstacles is discussed. The four stage data reduction process used to determine the primary candidate directions is discussed in Sections 3.2, 3.3, 3.4 and 3.5. The last part of the chapter, Section 3.6, discusses how look-ahead verification is used to determine the optimal candidate direction.

3.1 Histogram Grid for Obstacle Representation

The VFH* algorithm uses a two-dimensional Cartesian histogram grid C to represent obstacles (Borenstein & Koren, 1991b). Each cell of the histogram grid C has its own certainty value $c(i, j)$. On-board range sensors are used to continuously and rapidly update the histogram grid in real-time.

The range reading from the LIDAR sensor is used to increase the applicable cell's certainty value $c(i, j)$ by I^+ while all other cell's on between that cell and the robot are decreased by I^- . A cell's certainty value is bound by an arbitrarily chosen upper limit $CV_{\max} = 15$ and lower limit $CV_{\min} = 0$ (Borenstein & Koren, 1991a). I^+ was determined experimentally in relation to CV_{\max} . $I^+ = +3$ was chosen by Borenstein & Koren (1991a) since too large a value would cause the robot to react to single false readings. Contrarily a smaller value would not build up a cell's CV in time for an avoidance manoeuvre. I^- was determined relative to I^+ since it must be smaller than I^+ because

only one cell is updated for each reading whereas multiple cells might be decreased for one reading (Borenstein & Koren, 1991a). Thus, if an obstacle is detected multiple times it will have a high certainty value while random noise or misreadings will fade away with time (Borenstein & Koren, 1991b).

3.2 First Stage - The Primary Polar Histogram

In the first data-reduction stage the active region C_{active} , located on the histogram grid C , is mapped onto the primary polar histogram H^p (Ulrich & Borenstein, 1998). The active region C_{active} is located around the robot's momentary location and has a radius of r_{active} . Each cell acts as an obstacle vector with a magnitude $m_{i,j}$ and direction $\beta_{i,j}$ (Ulrich & Borenstein, 1998). The vector direction $\beta_{i,j}$ is based on each active cells' position relative to the robot center point (RCP) as shown:

$$\beta_{i,j} = \arctan \left(\frac{y_o - y_i}{x_o - x_i} \right) \quad (3.1)$$

where:

x_o, y_o : Coordinates of the RCP.

x_i, y_i : Coordinates of active cell $C_{i,j}$.

Additionally, the vector magnitude $m_{i,j}$ of each active cell $C_{i,j}$ is calculated as shown:

$$m_{i,j} = c_{i,j}^2 (a - b d_{i,j}^2) \quad (3.2)$$

where:

$c_{i,j}$: Certainty value of active cell $C_{i,j}$.

$d_{i,j}$: Distance from active cell $C_{i,j}$ to the RCP.

The parameters a and b are chosen according to:

$$a - b \left(\frac{r_{\text{active}} - 1}{2} \right)^2 = 1 \quad (3.3)$$

The certainty value, $c_{i,j}$, is squared when the vector magnitude is calculated (Ulrich & Borenstein, 1998). This means that recurring range readings will result in high certainty values and thus we can be more confident that an obstacle resides in the obstacle cell $C_{i,j}$. In contrast noise or once-off range readings do not result in high certainty values. The distance between the active cell and RCP, $d_{i,j}$, is also squared when calculating the vector magnitude. This means that occupied cells will have greater vector magnitudes the closer they are to the robot (Ulrich & Borenstein, 1998).

The robot is prevented from cutting corners by compensating for the robot's width (Ulrich & Borenstein, 1998). This is done by enlarging occupied cells by r_{r+s} as shown in Figure 2.21. The radius r_{r+s} consists out of two components.

The first is the robot's width r_{robot} and the second is a safety distance between the robot and obstacle defined as r_{safety} . Thus radius r_{r+s} is defined as:

$$r_{r+s} = r_{\text{robot}} + r_{\text{safety}} \quad (3.4)$$

The primary polar histogram H^p is constructed by dividing the active window surrounding the robot into sectors (Ulrich & Borenstein, 1998). An arbitrarily chosen angular resolution α is used so that the number of sectors is always an integer, i.e. $n = 360^\circ/\alpha$. In our simulations we chose $\alpha = 5^\circ$. Each angular sector corresponds to a discrete angle $\rho = k \cdot \alpha$.

Instead of updating only the sectors in which occupied cells fall all histogram sectors that are affected by the enlarged occupied cells are updated while building the primary polar histogram. The enlargement angle $\gamma_{i,j}$ is calculated as follows:

$$\gamma_{i,j} = \arcsin\left(\frac{r_{r+s}}{d_{i,j}}\right) \quad (3.5)$$

The polar obstacle density for each sector k is calculated as follows:

$$H_k^p = \sum_{i,j \in C_a} m_{i,j} \cdot h'_{i,j} \quad (3.6)$$

with:

$$\begin{aligned} h'_{i,j} &= 1 && \text{if } k \cdot \alpha \in [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}] \\ h'_{i,j} &= 0 && \text{otherwise} \end{aligned} \quad (3.7)$$

The h' function acts as a low-pass filter in that it smooths the polar histogram (Ulrich & Borenstein, 1998).

The primary polar histogram is drawn around the robot indicating sectors with high polar density values or in other words sectors that contain obstacles. The effect that the enlargement angle $\gamma_{i,j}$ has on the primary polar histogram is also indicated. The sectors to the sides of each obstacle that fall within the enlargement angle are also deemed blocked and assigned a polar obstacle density value using Equation 3.6. The sectors that are blocked due to the enlargement value are the filled sectors shown in Figure 3.2.

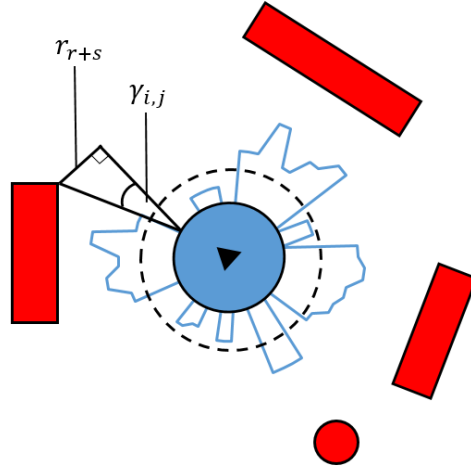


Figure 3.1: Primary polar histogram constructed around robot.

3.3 Second Stage - The Binary Polar Histogram

A robot moving in an environment with several narrow openings may behave in an indecisive manner due to openings alternating between open and blocked states during consecutive readings (Ulrich & Borenstein, 1998). This causes the indecisive behaviour since the robot's steering direction changes as the openings change states. To address this indecisive behaviour the polar histogram is reduced to a binary polar histogram.

The binary polar histogram H^b is constructed from the primary polar histogram using a hysteresis based on two thresholds τ_{low} and τ_{high} (Ulrich & Borenstein, 1998). By using these threshold values the primary polar histogram H^p 's sectors are reduced from polar obstacle density (POD) values to binary values; open (0) or blocked (1). The set of rules used to update the binary polar histogram H^b is shown:

$$H_{k,i}^b = 1 \quad \text{if} \quad , H_{k,i}^p > \tau_{\text{high}} \quad (3.8)$$

$$H_{k,i}^b = 0 \quad \text{if} \quad H_{k,i}^p < \tau_{\text{low}} \quad (3.9)$$

$$H_{k,i}^b = H_{k,i-1}^b \quad \text{otherwise} \quad (3.10)$$

The indecisive behaviour mentioned above is addressed through the process of reducing the polar histogram to the binary polar histogram since noise and misreadings are filtered out. When a sector in the primary polar histogram has a large POD (greater than τ_{high}) the sector is deemed blocked as shown in Equation 3.8. Conversely, a sector with a small POD (less than τ_{low}) is deemed open in the binary polar histogram, as shown in Equation 3.9, since noise and misreadings lead to small POD values. Noise and misreadings lead

to small POD values since they result in low certainty values in the histogram grid. Finally, if a sector's value falls in between the two threshold values the previous binary polar histogram is used to decide whether that sector is open or blocked as shown in Equation 3.10.

The concept is illustrated in Figure 3.2 where the polar histogram from Figure 3.1 has been reduced to a binary polar histogram where the sectors are either open or blocked. Consequently, the noise in Figure 3.1 is filtered out and the binary polar histogram will not change a sector's state from to open or blocked if the sector's threshold is not greater or less than τ_{high} and τ_{low} respectively.

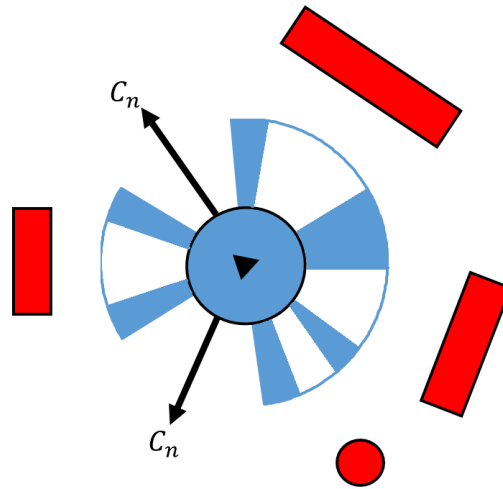


Figure 3.2: Binary polar histogram constructed around robot with two candidate directions.

3.4 Third Stage - The Masked Polar Histogram

The masked polar histogram is introduced as part of the VFH* to account for the robot trajectories, unlike the original VFH method that implicitly assumes that a robot is able to make instantaneous changes in its trajectory (direction of motion). For example, when a robot's dynamics and kinematics are ignored any of the trajectories shown in Figure 3.3 (a) would be valid trajectories. However, most mobile robots are unable to change their direction of motion instantaneously and are subject to turning circles as illustrated in Figure 3.3 (b). To address this the VFH* method approximates robot trajectories as circular arcs (constant curvature curves) (Ulrich & Borenstein, 1998).

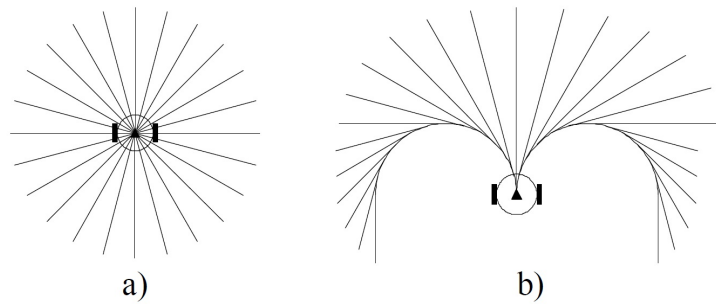


Figure 3.3: Robot trajectories (Ulrich & Borenstein, 1998).

To account for a robot's trajectories the robot's minimum steering radii to the left and right are used. For instance, Figure 3.4 shows a robot approaching two obstacles. The two obstacles are enlarged by r_{r+s} to account for the robot's width and safety distance. If one of the robot's trajectory circles and an enlarged obstacle cell overlap all directions of motion to that side, between the obstacle and the robot's backwards direction of motion, are blocked.

In the figure the robot's trajectory circle to its left overlaps with enlarged obstacle A as shown in Figure 3.4. Consequently, all sectors to the robot's left between the enlarged obstacle and the backwards direction of motion are blocked (masked). However, the robot can still turn to the right of enlarged obstacle B since its trajectory circle to the right does not overlap with enlarged obstacle B.

To approximate a robot's trajectories, as circular arcs, the values for the minimum steering radii of a robot has to be ascertained experimentally. The steering radii to the right and left of the robot are defined as r_r and r_l respectively and are functions of velocity (Ulrich & Borenstein, 1998). The masked polar histogram shows which directions are accessible at the robot's current speed. If all sectors are blocked the robot would have to decrease its speed and reconstruct the masked polar histogram (Ulrich & Borenstein, 1998). By decreasing its speed the robot reduces its steering radii and thus new candidate directions may be found.

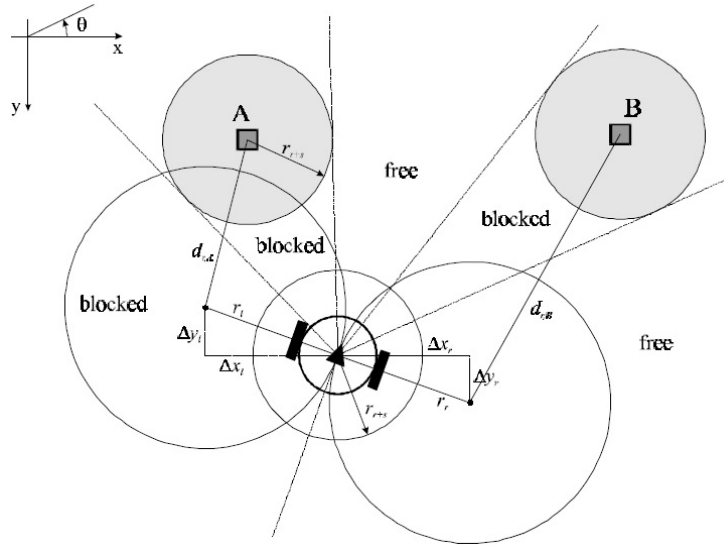


Figure 3.4: Accounting for robot trajectories (Ulrich & Borenstein, 1998).

The robot's right and left trajectory centers are given by:

$$\begin{aligned} \Delta x_r &= r_r \cdot \cos\theta & \Delta y_r &= r_r \cdot \sin\theta \\ \Delta x_l &= -r_l \cdot \cos\theta & \Delta y_l &= -r_l \cdot \sin\theta \end{aligned} \quad (3.11)$$

Distances from the two trajectory centers to an active cell $C_{i,j}$ are given by:

$$\begin{aligned} d_r^2 &= (\Delta x_r - \Delta x(j))^2 + (\Delta y_r - \Delta y(i))^2 \\ d_l^2 &= (\Delta x_l - \Delta x(j))^2 + (\Delta y_l - \Delta y(i))^2 \end{aligned} \quad (3.12)$$

Directions to the robot's right are blocked if:

$$d_r^2 < (r_r + r_{r+s}) \quad [\text{condition 1}] \quad (3.13)$$

Directions to the robot's left are blocked if:

$$d_l^2 < (r_l + r_{r+s}) \quad [\text{condition 2}] \quad (3.14)$$

The sectors which would be inaccessible for a robot due to its left and right turn circles are deemed blocked (masked). To determine which sectors are inaccessible every cell $C_{i,j}$ in the active window C_a is evaluated according to the above two conditions. Consequently two limit angles can be determined; ϕ_r , to the robot's right, and ϕ_l , to the robot's left. Additionally, the backwards direction of motion is defined as $\phi_b = \theta + \pi$ where θ is the robot's current direction of motion. To determine the limit angles the following procedure is implemented:

1. The backwards angle of motion ϕ_b is first determined. Then the limit angles ϕ_r and ϕ_l are set equal to ϕ_b .
2. This process is then repeated for every cell in the active window C_a with a certainty value satisfying a threshold value $c_{i,j} > \tau$:
 - a) If θ is to the left and $\beta_{i,j}$ is to the right of ϕ_r , condition 1 is checked. If the condition is satisfied ϕ_r is set equal to $\beta_{i,j}$.
 - b) On the other hand if θ is to the right and $\beta_{i,j}$ is to the left of ϕ_r , condition 2 is checked. If the condition is satisfied ϕ_l is set equal to $\beta_{i,j}$.

The masked polar histogram H^m can be constructed after the limit angles ϕ_r and ϕ_l have been determined from the above procedure. If a sector lies between one of the robot's two limit angles and the robot's backwards direction of motion that sector's state is changed to blocked as shown:

$$\begin{aligned} H_k^m &= 0 & \text{if } H_k^b = 0 \text{ and } (k \cdot \alpha) \in \{[\phi_r, \theta], [\theta, \phi_l]\} \\ H_k^m &= 1 & \text{otherwise} \end{aligned} \quad (3.15)$$

The data reduction process is illustrated in Figure 3.5. First the the primary polar histogram is constructed as shown in Figure 3.5 (a). Secondly, the primary polar histogram is reduced to the binary polar histogram, as shown in Figure 3.5 (b), using threshold values. Finally, to account for the robot's turn circles the masked polar histogram is constructed from the binary polar histogram as shown in Figure 3.5 (c). A number of sectors can be seen that have become blocked due to the robot's mobility constraints. If the masked polar histogram is not used the robot might very well choose a path that would lead to a collision with an obstacle due to the robot's turn circles.

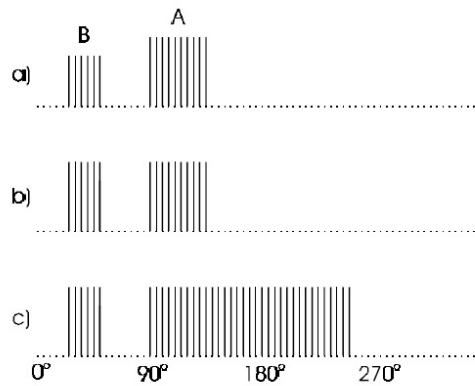


Figure 3.5: Histogram representation of a) the primary polar histogram, b) binary polar histogram and c) the masked polar histogram (Ulrich & Borenstein, 1998).

3.5 Fourth Stage - Determining of Primary Candidate Directions

The primary candidate directions are determined from the masked polar histogram's openings (Ulrich & Borenstein, 2000, 1998). Each opening is classified as either wide or narrow based on its size. The right and left borders, k_r and k_l , of all openings are used to establish whether an opening is wide or narrow (Ulrich & Borenstein, 1998). An opening is considered wide if the difference between its two borders is larger than s_{\max} sectors ($s_{\max} = 18$ in our simulations). Conversely, if the difference between its two borders is less than s_{\max} sectors the opening is considered narrow. The new direction of motion can be determined by evaluating each of the candidate directions (Ulrich & Borenstein, 2000).

Candidate directions are determined differently for wide and narrow openings. A narrow opening has only one candidate direction c_n which is the center of the opening, as illustrated in Figure 3.2. The candidate direction for a narrow opening is defined as:

$$c_n = \frac{k_r + k_l}{2} \quad \text{centered direction} \quad (3.16)$$

A wide opening has two potential candidate directions; one to the right of its left border, c_l , and one to the left of its right border, c_r (Ulrich & Borenstein, 1998). If the target direction k_t lies between the wide opening's two candidate directions a third candidate directions c_t , the direction straight to the goal, is also considered. The two candidate directions c_r and c_l guide the robot along the obstacle contour at a safe distance, while c_t leads the robot directly towards the target (Ulrich & Borenstein, 1998). The candidate directions for a wide opening are defined as:

$$c_r = k_r + \frac{s_{\max}}{2} \quad \text{towards the right} \quad (3.17)$$

$$c_l = k_l - \frac{s_{\max}}{2} \quad \text{towards the left} \quad (3.18)$$

$$c_t = k_t \quad \text{if } k_t \in [c_r, c_l] \quad (3.19)$$

Multiple primary candidate directions might exist. To select the most appropriate candidate direction a cost function is used to evaluate each candidate direction. The candidate direction with the lowest cost is selected as the robot's new direction of motion. Ulrich & Borenstein (1998) proposed the following cost function for a primary candidate direction c_0 :

$$g_0(c_0) = \mu_1 \cdot \Delta(c_0, k_t) + \mu_2 \cdot \Delta\left(c_0, \frac{\theta_n}{\alpha}\right) + \mu_3 \cdot \Delta(c_0, k_{d,n-1}) \quad (3.20)$$

with:

$$\Delta(c_1, c_2) = \min \left\{ |c_1 - c_2|, |c_1 - c_2 - 360^\circ\alpha|, \left| c_1 - c_2 + \frac{360^\circ}{\alpha} \right| \right\} \quad (3.21)$$

where:

- α : angular resolution of polar histogram
- θ_n : current robot direction of motion
- k_t : target / goal direction relative to robot's current position
- $k_{d,n-1}$: previously selected direction of motion

The **first term** of the cost function, $\Delta(c_0, k_t)$, is the cost associated with the difference between a candidate direction and target direction. The larger the difference between the two directions, the more the robot is guided away from the target direction.

The **second term**, $\Delta(c_0, \frac{\theta_n}{\alpha})$, is the cost associated with the difference between a candidate direction and the robot's wheel orientation. The larger the difference between the two, the larger the required change of robot orientation.

The **third term**, $\Delta(c_0, k_{d,n-1})$, is the cost associated with the difference between a candidate direction and the robot's previously selected direction of motion. The larger this term, the larger the change in steering direction will be.

Each of the three terms in the cost function is given a weight μ . The magnitude of the μ values are not significant but rather their magnitudes relative to one another. Consequently, a higher μ_1 will lead to the robot displaying more goal-oriented behaviour (Ulrich & Borenstein, 1998). A higher μ_2 value leads to more efficient paths since excessive changes in the direction of motion are avoided. Thirdly, a higher μ_3 value leads to smoother trajectories since the the robot commits to a direction as it attempts to head towards the previously selected direction of motion (Ulrich & Borenstein, 1998). By tuning the μ values relative to one another, more or less emphasis is placed on the different parts of the cost function to affect which candidate direction is chosen as the robot's new direction of motion (Ulrich & Borenstein, 1998). The following condition must be satisfied to ensure goal-oriented behaviour:

$$\mu_1 > \mu_2 + \mu_3 \quad [\text{condition 3}] \quad (3.22)$$

3.6 Look-ahead Verification

The process described up until this point is the process followed by the VFH+ which works well in most cases. However, due to the fact that the VFH+ algorithm is a local obstacle avoidance algorithm it does not always select the most appropriate candidate direction. This is illustrated in Figure 3.6 where the robot encounters multiple obstacles (shown in black). The configuration

space is indicated as the gray region around the obstacles while the robot's active window is indicated as a dashed circle. This is the region wherein an obstacle triggers an avoidance manoeuvre. With the information available two primary candidate trajectories are determined, trajectory *A* and *B*. Trajectory *A* leads to a dead-end, while trajectory *B* would turn into trajectory *C* at point *p* if that route were to be followed. If no additional information is available the robot would choose trajectory *B* only 50% of the time. To address this the VFH* algorithm builds on the VFH+ by adding look ahead-verification to verify which primary candidate trajectory would be the most suitable direction of motion (Ulrich & Borenstein, 2000).

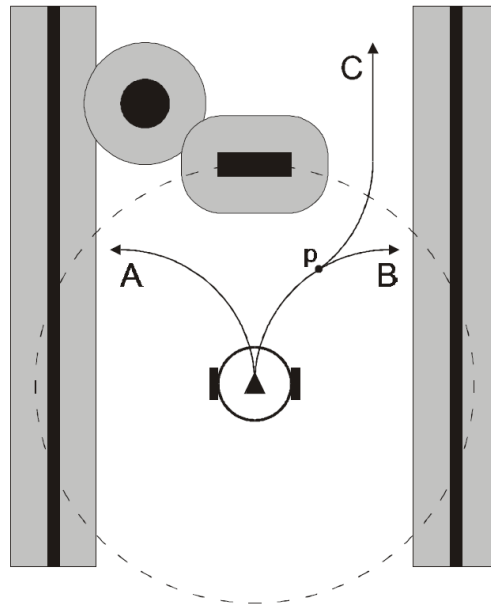


Figure 3.6: Robot using look-ahead verification to determine optimal path (Ulrich & Borenstein, 2000).

Look-ahead verification works by projecting the robot's trajectory several steps ahead and evaluating the consequences if each of the candidate directions were to be followed (Ulrich & Borenstein, 2000). The robot's projected position and orientation, if it were to move a projected step distance d_s in each of the primary candidate directions, is determined (Ulrich & Borenstein, 2000). At every projected position a new primary polar histogram is constructed based on the histogram grid information available. Next, the data-reduction process as described previously is repeated. New candidate directions are found, called projected candidate directions. The process is repeated n_g times until finally a search tree of depth n_g remains, where the end nodes correspond to the total projected distance $d_t = n_g \cdot d_s$ as illustrated in Figure 3.7 (Ulrich & Borenstein, 2000).

The larger the total projected distance d_t the greater the total look-ahead. If selected too high the algorithm is slowed down considerably while if its chosen too small the most optimal path might not be identified. Thus, selection of d_t is a trade-off between the speed and quality of the algorithm (Ulrich & Borenstein, 2000).

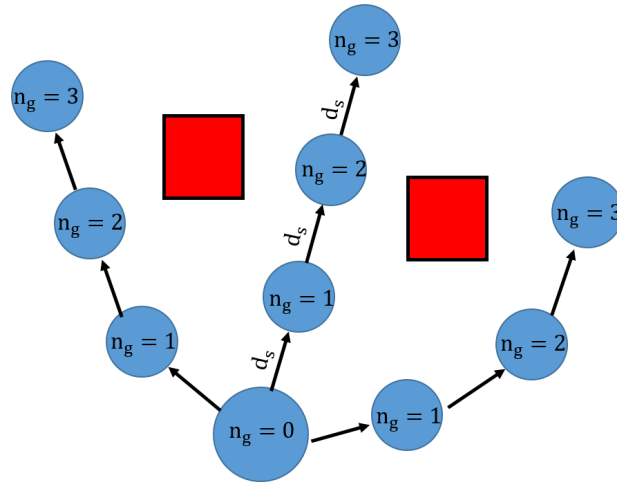


Figure 3.7: Look-ahead verification with search depth $n_g = 3$ and projected distance d_s .

The A* search algorithm is used to select the optimal primary candidate direction. This is done by assigning a cost to the nodes of every candidate direction c . The cost of a node is the sum of the costs of the nodes leading back to the start node (Ulrich & Borenstein, 2000). A priority value $f(c)$ is then assigned to each node based on the node's cost and a heuristic function $h(c)$. Thus, the priority value of a node is defined as $f(c) = g(c) + h(c)$ (Ulrich & Borenstein, 2000). The primary candidate direction that leads to the end node with highest priority (smallest $f(c)$) is then selected as the new direction of motion (Ulrich & Borenstein, 2000).

For more information on the A* search algorithm the reader is referred to Appendix B. The selection of the cost and heuristic functions used for the projected candidate directions is discussed in Section 3.6.1 and 3.6.2 respectively.

3.6.1 Projected Candidate Directions - Cost Function

The cost function of the projected candidate directions is based on the cost function of the primary candidate directions. However, there are some distinct differences too. The modified cost function for projected candidate directions

is shown:

$$g_i(c_i) = \lambda^i \left[\mu'_1 \cdot \max \{ \Delta(c_i, k_t), \Delta(k_e, k_t) \} + \mu'_2 \cdot \Delta\left(c_i, \frac{\theta_i}{\alpha}\right) + \mu'_3 \cdot \Delta(c_i, c_{i-1}) \right] \quad (3.23)$$

with:

$$k_e = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \quad (3.24)$$

and

$$0 < \lambda \leq 1 \quad (3.25)$$

The **first term** of the cost function represents the cost associated with the robot deviating from the target direction thereby ensuring goal-oriented behaviour. This is similar to the first term of the cost function used for the primary candidate directions but it does however differ slightly.

For a projected candidate direction the **first term** of the cost function also considers the effective direction of motion k_e . This is representative of the forward progress of a trajectory. It is ideal if both the candidate direction c_i and the effective direction of motion k_e correspond with the target direction k_t . However, this is not always the case as shown in Figure 3.8. In the figure the candidate direction c_i is aligned with the target direction k_t and thus the first term of the cost function is zero. The effective direction of motion however is not aligned with the target direction k_t and thus the robot makes very little to no forward progress.

If the effective direction of motion is not considered it can result in a situation where a projected trajectory has a low cost even though it makes no forward progress. Thus, by including the effective direction of motion trajectory costs become high if they make no forward progress (Ulrich & Borenstein, 2000).

The primary candidate direction's cost function did not make use of an effective direction of motion because the robot has no control over its current orientation. On the other hand the robot does have control over its projected trajectory and therefore it is necessary to include the effective direction of motion in the cost of a projected candidate direction (Ulrich & Borenstein, 2000).

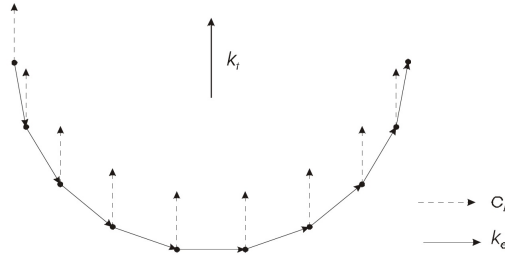


Figure 3.8: Robot that's effective direction of motion is making no clear progress towards the target/goal Ulrich & Borenstein (2000).

In the cost function used for the projected candidate directions the **second and third terms** no longer represent the cost associated with the robot committing to a direction. Instead these two terms represent the cost associated with making the projected trajectory smoother (Ulrich & Borenstein, 2000).

Weights are again assigned to the three terms in the cost function namely μ'_1 , μ'_2 , and μ'_3 . A greater μ'_1 value ensures the robot's goal-oriented behaviour while greater μ'_2 and μ'_3 lets the robot favour smoother trajectories. To ensure goal-oriented behaviour the following condition must be satisfied:

$$\mu'_1 > \mu'_2 + \mu'_3 \quad [\text{condition 4}] \quad (3.26)$$

Ulrich & Borenstein (2000) prescribed the following condition to emphasize the importance of each of the primary candidate directions over projected candidate directions:

$$\mu_1 \geq \mu'_1 \quad [\text{condition 5}] \quad (3.27)$$

Ulrich & Borenstein (2000) found the following set of parameters worked well for a goal-oriented mobile robot:

$$\begin{array}{lll} \mu_1 = 5 & \mu_2 = 2 & \mu_3 = 2 \\ \mu'_1 = 5 & \mu'_2 = 1 & \mu'_3 = 1 \end{array} \quad (3.28)$$

Ulrich & Borenstein (2000) suggested the discount factor λ in Equation 3.25 be set to 0.8. This factor is used so that not all candidate directions carry equal weight. Instead their weights decrease as their depth i increases since they are weighed by a factor λ^i .

This decreases the problem of having a fixed goal depth that results in a sharp cut-off. Without this factor the obstacle avoidance algorithm would not always behave as desired since all branches would carry equal weight. This is illustrated in Figure 3.9 where the robot's projected trajectory is shown with a search depth of $n_g = 7$. The target direction is shown by k_t . It can be seen that if the discount factor λ is ignored the cost of trajectory B would be cheaper than the cost for trajectory A causing the robot to continue to the right without making significant progress towards the target. This is because

the algorithm has an undesired tendency to favour trajectories that stop shortly before being affected by obstacles. If the search depth n_g is increased by one trajectory B encounters an obstacle and thus trajectory C would become the cheapest trajectory to follow. However, if the discount factor λ is used the cut-off is less sharp as the node depth increases leading to trajectory A becoming cheaper than trajectory B .

The discount factor λ to some extent helps to compensate for uncertainties in the map information. The map building process is stochastic in nature and thus the robot tends to be more certain about its immediate environment than positions further away. Thus it makes intuitive sense to give more weight to nodes closer to the robot than those further away or of greater depth. Finally with λ there is more control over the weights of the branches at different depths and thus more control over the behaviour of the search algorithm.

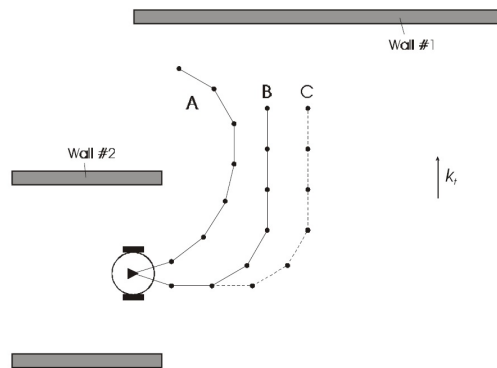


Figure 3.9: Situation in which discount factor λ is required for the robot to choose the optimal path (Ulrich & Borenstein, 2000).

3.6.2 Projected Candidate Directions - Heuristic Function

The cheapest cost from a node n_i to the goal node is estimated using a heuristic function $h(c)$. For the heuristic function to be admissible it must never overestimate the cost to reach the goal node. Ulrich & Borenstein (2000) suggested that if condition 4 is satisfied the cost would be minimal if the robot moved in the target direction k_t at every subsequent node. Thus a simple admissible heuristic was suggested that is based on the cost function but replaces the candidate direction c_i with target direction k_t as shown:

$$h_i(n_i) = \lambda^i \left[\mu'_2 \cdot \Delta \left(k_t, \frac{\theta_i}{\alpha} \right) + \mu'_3 \cdot \Delta (k_t, c_{i-1}) \right] \quad (3.29)$$

The effective direction of motion is ignored in the above heuristic and only the costs associated with the next branch is considered (Ulrich & Borenstein, 2000). This heuristic is admissible and computationally efficient, however it is

not optimal in that it underestimated the minimum cost of reaching the goal node.

To account for the effective direction of motion Ulrich & Borenstein (2000) suggested the following heuristic:

$$h_i(n_i) = \lambda^i \left[\mu'_1 \cdot \Delta(k_e, k_t) + \mu'_2 \cdot \Delta\left(k_t, \frac{\theta_i}{\alpha}\right) + \mu'_3 \cdot \Delta(k_t, c_{i-1}) \right] \quad (3.30)$$

with:

$$k_e = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \quad \text{based on } c_i = k_t \quad (3.31)$$

The heuristic shown above considers the effective direction of motion, however the additional term makes it computationally more expensive. This heuristic is still not optimal though since it only considers the minimum cost associated with the next branch (Ulrich & Borenstein, 2000).

To get an optimal heuristic one would have to expand the current node, without building the polar histogram and finding the candidate directions, until the goal depth is reached. Additionally the target direction k_t would have to be used as the candidate direction at each node (Ulrich & Borenstein, 2000). Then by summing the node costs the optimal heuristic value could be obtained. This however requires more computational power.

It is clear that many admissible heuristics exist and that a trade-off between quality and speed has to be made. Ulrich & Borenstein (2000) found that the heuristic had no influence on the resulting robot direction of motion and opted for the first heuristic for their particular application.

3.7 Conclusion

A detailed overview of the VFH* algorithm was presented in this chapter. The VFH* algorithm uses information from a range sensor to update the histogram grid. The active region on the histogram grid is then used to construct the primary polar histogram that also accounts for the robot's width and a safety region. To prevent indecisive behaviour the primary polar histogram is reduced to a binary polar histogram. Furthermore, to account for the robot's trajectories (turning circles) the binary polar histogram is reduced to the masked polar histogram. The masked polar histogram is used to determine the primary candidate directions. Look-ahead verification is performed using the A* search algorithm. Finally, the primary candidate direction that produces the end node with the lowest cost (highest priority) is selected as the robot's new direction of motion.

Chapter 4

Code Validation: Simulations

The VFH* obstacle avoidance algorithm was coded using MATLAB. To verify the correctness of the code, and to illustrate some key concepts of the VFH* algorithm, simulations were performed. The simulations were done in a step-wise manner starting with the VFH algorithm and modifying it to the VFH+ algorithm to take into account the robot's width and safety region. Finally, the VFH+ algorithm was modified to the VFH* algorithm by adding the A* search algorithm with cost and heuristic functions. The parameters in Table 4.1 were used for the simulations.

Table 4.1: VFH* algorithm simulation parameters.

Parameter	Unit	Value
Cell size	[m]	1
Active window width	[m]	5
s_{\max}	[–]	18
α	[°]	5
r_{safety}	[m]	1
n_g	[–]	10
r_{robot}	[m]	0.6

The VFH* algorithm was coded in MATLAB since MATLAB's Robotics Toolbox allows MATLAB code to be run as part of a Robot Operating System (ROS) network. The significance of this is discussed in more detail in Chapter 6. Furthermore, the use of MATLAB allows relatively easy debugging and the simulation code can be modified for real-time implementation on a multicopter with relative ease.

An overview of algorithm structure used for the simulations is shown in Figure 4.1. The simulation code was given a start and goal position. The area around the robot is scanned for obstacles and the histogram grid is updated if obstacles are encountered. The primary polar histogram is then constructed

from the active region on the histogram grid. Data reduction is performed and the binary polar histogram is constructed from the primary polar histogram. Next the masked polar histogram is constructed to account for the robot's turning circle/trajectory. The masked polar histogram is then used to determine the primary candidate directions. To select the optimal candidate direction the A* search algorithm was implemented with cost and heuristic functions as described in Chapter 3. Consequently the primary candidate direction with the lowest cost (highest priority) is chosen as the new direction of motion. Thereafter, using the new direction of motion the robot's new position is determined. The robot moves to the new position and the process is repeated to find the next position until the robot reaches the goal position.

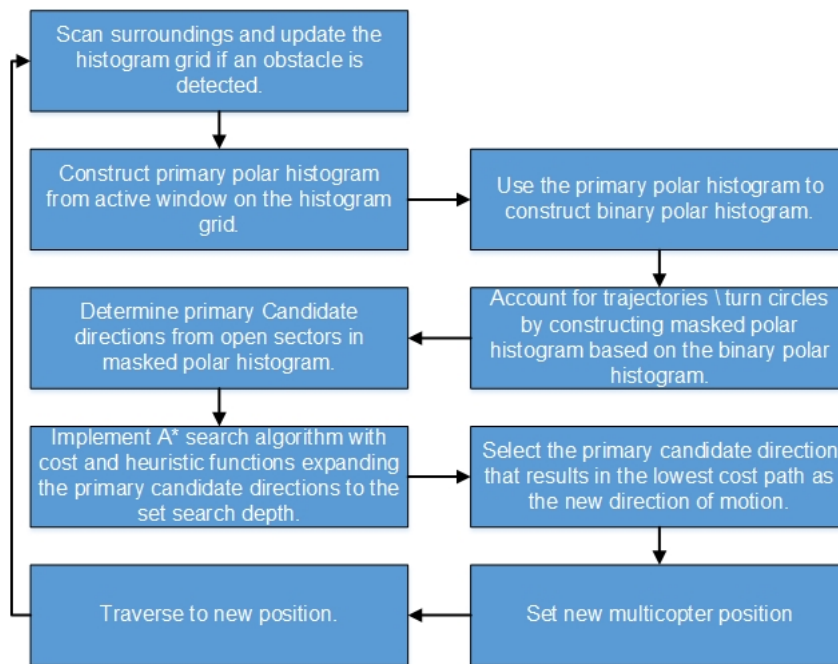


Figure 4.1: Overview of simulation code.

The first simulation scenario is that of an opening between two obstacle walls. To reiterate the importance of the use of histograms the VFF algorithm is first demonstrated. Figure 4.2 shows a robot moving towards a target using the VFF obstacle avoidance algorithm with its wall following disabled. The robot initially progresses towards the goal, however, it falls into a local minimum when it gets close enough to the opening. This is due to the fact that the attractive force guiding the robot towards the target is negated by the repulsive forces from the two obstacles as was mentioned in Section 2.1.6. Even if the WFM was used in this case there is no certainty that the robot would reach a point where it can continue to the goal. Additionally, this would be time consuming and inefficient as a clear path to the target is available.

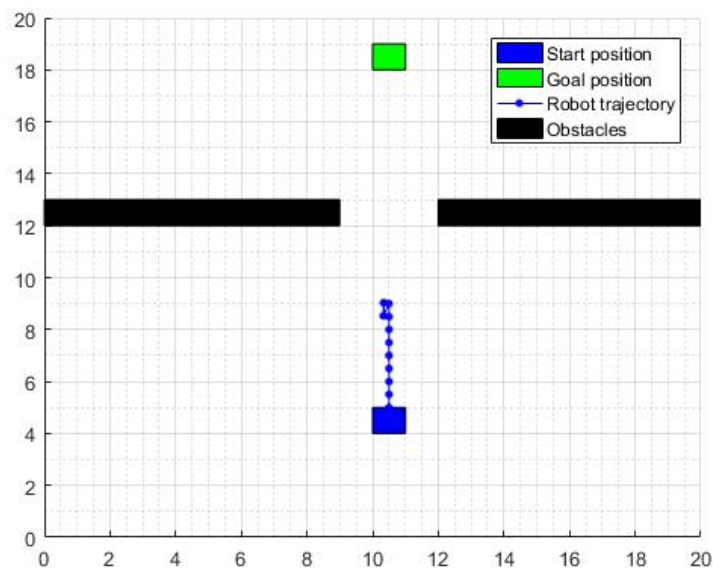


Figure 4.2: Simulation 1: The VFF algorithm encountering a local minimum.

As was mentioned in Chapter 3 the VFH algorithm addresses this issue by making use of a polar histogram for navigation. This allows the robot to move through the gap between the two obstacles as shown in Figure 4.3. However, if the robot is wide or the space between the two obstacles is narrow the robot might still collide with the obstacles. The VFH+ algorithm addresses this problem by accounting for the robot's width and safety region.

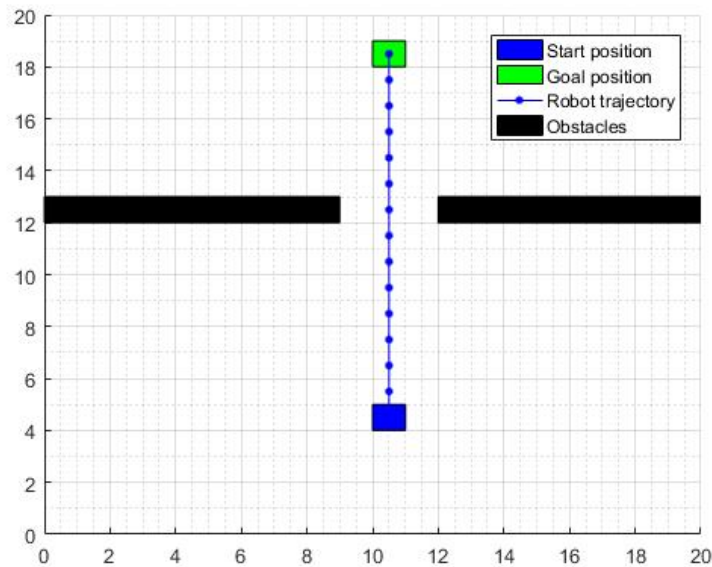


Figure 4.3: Simulation 2: The VFH algorithm overcoming the local minimum that the VFF algorithm could not.

Figure 4.4 shows a robot using the VFH+ obstacle avoidance algorithm. In this case the robot is able to establish that due to its width and safety region it will not be able to pass through the gap between the two obstacles. However, if the robot does not consider its width and safety region, as was the case with the VFH algorithm, it would have continued to move toward the target and collided with the sides of the obstacle.

Figure 4.5 illustrates how the VFH and VFH+ algorithms move around an obstacle to the goal. The figure shows how the VFH+ algorithm keeps a much larger distance between the robot and the obstacle, than the VFH algorithm does, since it is compensating for the robot's width and safety region. If the algorithm does not compensate for the robot's width and safety region in this manner the robot could potentially collide with the obstacle as it moves around the obstacle.

Finally, the VFH+ algorithm does not always make the correct choice of candidate directions when more complex obstacles are encountered as is shown in Figure 4.6. The VFH* algorithm addresses this by using look-ahead verification. The figure shows that the VFH* algorithm is able to determine the optimal path and progress to the goal. If look-ahead verification was not used the robot would have a 50% chance of choosing the correct direction, otherwise the robot would traverse to the right of the obstacle in the figure. In that case the robot would either end up travelling further than necessary to reach the goal or end up making no progress towards the goal or potentially encounter more obstacles.

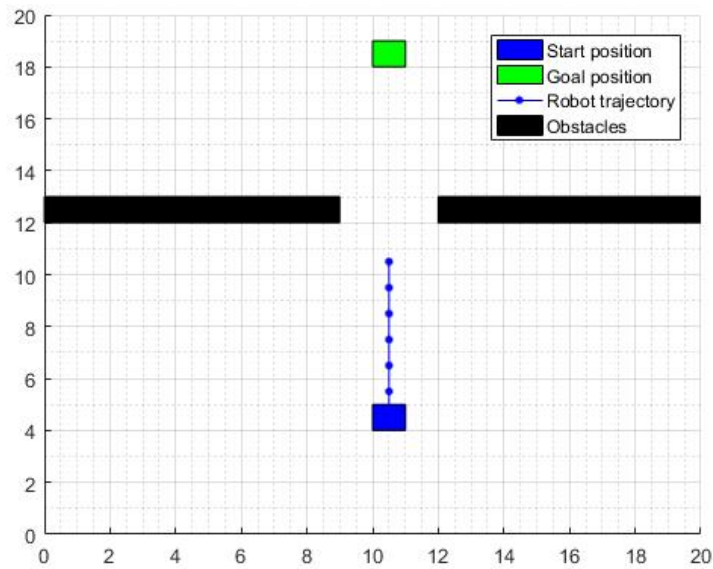


Figure 4.4: Simulation 3: The VFH+ algorithm accounting for the robot's width and a safety distance.

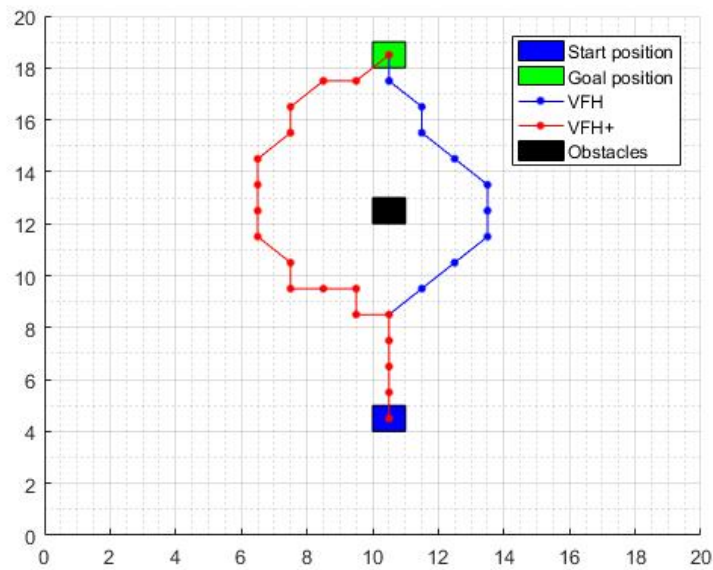


Figure 4.5: Simulation 4: A comparison between the VFH and the VFH+ algorithms.

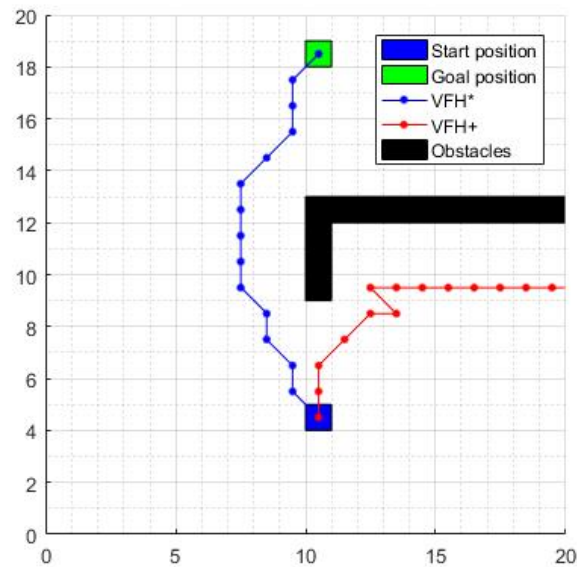


Figure 4.6: Simulation 5: The VFH* algorithm implementing look-ahead verification to choose the optimal path.

4.1 Conclusion

The MATLAB code of the VFH* algorithm was validated through a series of simulations. First the VFH algorithm was programmed and simulated in an environment that the VFF algorithm finds troublesome. The VFF algorithm is unable to move to the goal when confronted by two obstacles placed closely together with a gap in between. However, the VFH algorithm was able to navigate to the goal when it encountered the same two obstacles with a gap in between. This showed the usefulness of using a polar histogram for navigation.

The second simulation showed that the VFH algorithm could potentially cause the robot to collide with obstacles if the robot's width and safety region is not considered. Thus the code was modified to the VFH+ algorithm. Similar to the previous simulation the obstacles were placed close to one another with a gap in-between. The robot's width and a safety distance was however implemented since the VFH+ algorithm was used. Thus the algorithm determined that the robot would not be able to fit between the two obstacles and did not allow the robot to proceed.

The third simulation was a comparison between the VFH and VFH+ algorithms as the robot moved around an obstacle. Both algorithms successfully navigated to the goal. However, the VFH+ algorithm maintained a greater distance between itself and the obstacle due to the inclusion of its width and safety distance. In the event that the robot is rather wide or that the obstacle's

position is not certain and slightly closer to the robot, the risk of the robot colliding with the obstacle is increased when the VFH algorithm is used.

Finally, the code was further modified to the VFH* algorithm that includes look-ahead verification using the A* search algorithm along with cost and heuristic functions. The final simulation showcased the advantage of using look-ahead verification when a primary candidate direction is selected. Due to the use of look-ahead verification the robot was able to choose the primary candidate direction that guided the robot to the left of the obstacle at each point. If look-ahead verification was not used the robot would have selected the correct path around the obstacle 50% of the time. The code was deemed to be correct since it successfully accounted for robot width, a safety distance, and verified its path using look-ahead verification.

Chapter 5

Modifications to VFH*

Multicopters differ from ground vehicles in the sense that they are subject to inaccurate sensor data that cause uncertainties with regard to their position, yaw, and tilt estimations. Additional information on the basic concepts of how multicopters work is provided in Appendix A. The original VFH* algorithm does not take into consideration uncertainties in position, yaw, or tilt estimation since mobile ground robots make use of motor encoders and other mechanisms to more accurately determine the robot's position and orientation (yaw and tilt). However, as will be shown, the VFH* algorithm can be modified to account for these uncertainties by updating the way in which the multicopter builds its histogram grid and primary polar histogram.

The multicopter's tilt angle is used to collectively refer to the multicopter's pitch and roll angles. These parameters are important to consider when range measurements are taken since an obstacle may appear to be further away than what it is due to the angle of measurement.

5.1 Adding Safety Distance to Range Data

The original VFH* algorithm makes use of a safety distance d_{safety} when constructing the primary polar histogram as was discussed in Section 3.2. Though this safety distance works well when the multicopter is moving around an object it has no effect on the initial distance between the multicopter and an obstacle. This is because the safety distance d_{safety} is used to calculate only the enlargement angle $\gamma_{i,j}$ in Equation 3.5 and is not accounted for in terms of the direct distance between the multicopter and the obstacle. Using the original VFH* algorithm would mean that the multicopter would only react to an obstacle once the obstacle is within the multicopter's active region. This means that the safety distance is only considered as the multicopter moves around the obstacle. In general this works well but to ensure that the multicopter starts reacting at the appropriate distance from an obstacle we account

for the safety distance d_{safety} when updating the histogram grid C as follows:

$$d_{\text{enlarged}} = d_{i,j} - d_{\text{safety}} \quad (5.1)$$

where:

$d_{i,j}$: Distance from active cell $C_{i,j}$ to the RCP.

By using d_{enlarged} when updating the histogram grid instead of $d_{i,j}$, the multicopter will start reacting to an obstacle exactly a distance of d_{safety} sooner than it would when d_{safety} is only included in the enlargement angle $\gamma_{i,j}$. This allows the histogram grid to be updated before an obstacle enters the multicopter's active region. The correction made in Equation 5.1 means that the multicopter will detect the obstacle a distance of d_{safety} sooner than it would usually do. This means that the effective distance between the multicopter's center point and the obstacle when it is detected is:

$$d_{\text{effective}} = r_{\text{active-region}} + d_{\text{safety}} \quad (5.2)$$

5.2 Compensating for Multicopter Position Uncertainties

Most multicopters make use of a Global Positioning System (GPS) to determine their positions. Accurate GPS and vicon systems are often very expensive. This poses a problem for the VFH* algorithm since it constructs its primary polar histogram from the histogram grid C based on the multicopter's estimated position on the histogram grid. In this study a u-blox LEA-6 GPS was used to determine the robot's position, refer to Appendix G for additional information.

To account for the uncertainties in a multicopter's position it is first acknowledged that the multicopter can be anywhere within a circular uncertainty region around its estimated position as shown in Figure 5.1. From the figure one can see that the multicopter would fit through the gap between the two obstacles if its position is accurately known. But since that is unlikely one has to consider the uncertainty region around the multicopter which means that the multicopter might very well be on a collision course with one of the two obstacles. The radius of the uncertainty region may vary depending on the type and quality of sensor used as mentioned earlier. We define the radius of the position uncertainty region as r_{position} .

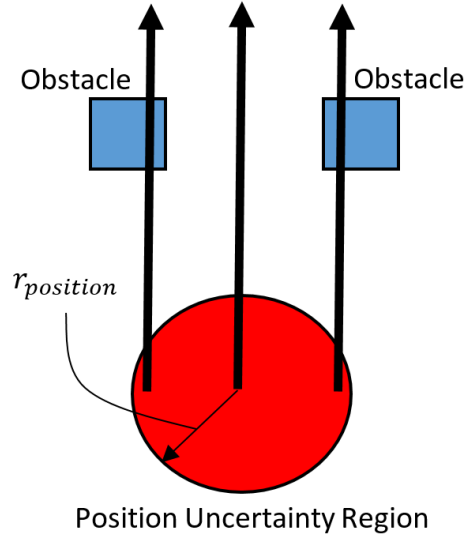


Figure 5.1: Multicopter position uncertainty.

The uncertainty region is accounted for in the VFH* algorithm by modifying the range readings from the two-dimensional LIDAR range sensor. This can be done by subtracting r_{position} from the LIDAR range measurement when updating the histogram grid C :

$$d_{\text{enlarged}} = d_{i,j} - d_{\text{safety}} - r_{\text{position}} \quad (5.3)$$

where:

$d_{i,j}$: Distance from active cell $C_{i,j}$ to the RCP.

This enlarges the area an obstacle occupies in the histogram grid and allows the uncertainty region to be shifted from the multicopter to the obstacles as illustrated in Figure 5.2. This is however only half the solution as this does not enlarge the sides of the obstacle to account for the position uncertainty. Therefore the enlargement angle $\gamma_{i,j}$ from Equation 3.5 is further modified as shown:

$$r_{\text{enlarged}} = r_{\text{multicopter}} + d_{\text{safety}} + r_{\text{position}} \quad (5.4)$$

$$\gamma_{i,j} = \arcsin\left(\frac{r_{\text{enlarged}}}{d_{\text{enlarged}}}\right) \quad (5.5)$$

Thus, the multicopter will start reacting sooner to an obstacle to compensate for inaccuracies in its position estimation. The multicopter's position uncertainty is also compensated for when it moves around obstacles since the r_{position} term has been included in the enlargement angle $\gamma_{i,j}$ as shown in Figure 5.2. In so doing the position uncertainty is taken into account when VFH* algorithm calculates candidate directions.

Figure 5.3 shows a simulation of the VFH* algorithm taking into consideration various position uncertainties. It can be observed that the multicopter

reacts sooner when the uncertainty is more due to Equation 5.3 while the increase in the size of the arcs as the robot moves around the obstacle are the consequence of Equation 5.5. The effective distance between the multicopter and the obstacle when the multicopter starts reacting to the obstacle can be updated as follows:

$$d_{\text{effective}} = r_{\text{active-region}} + d_{\text{safety}} + r_{\text{position}} \quad (5.6)$$

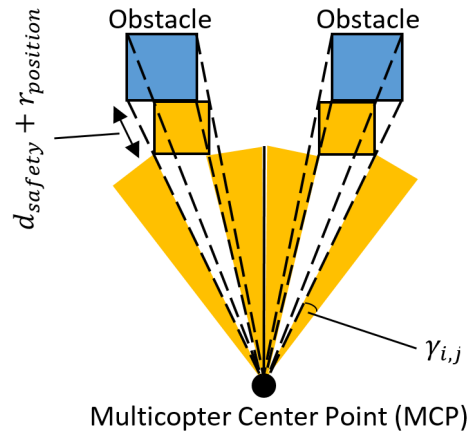


Figure 5.2: Enlarged obstacles with multicopter as point-like vehicle.

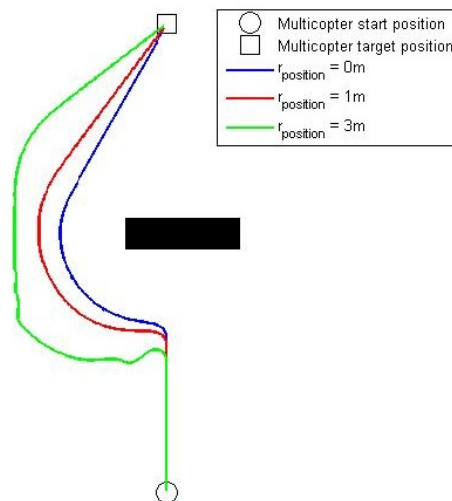


Figure 5.3: Effect of various position uncertainties on multicopter trajectory.

Figure 5.4 shows the effect of three position uncertainties when the multicopter attempts to traverse terrain with clustered obstacles. The multicopter manages the shortest route when no uncertainty is present, $r_{\text{position}} = 0\text{ m}$, while

it is still able to move in between two obstacles which are slightly further apart when $r_{\text{position}} = 1$ m. However, when the uncertainty is increased even more to $r_{\text{position}} = 4$ m the multicopter's navigation system can no longer be certain that it won't collide with any of the obstacles and thus appropriately chooses to go around the cluster.

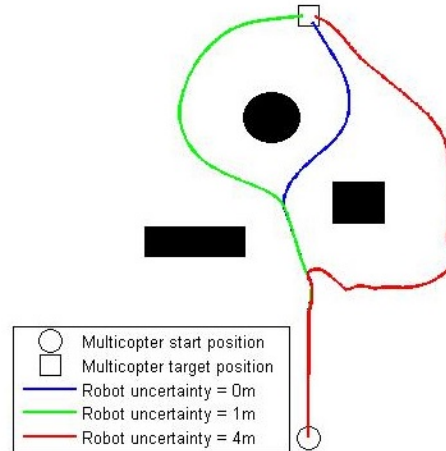


Figure 5.4: Effect of position uncertainty on multicopter trajectory in a cluttered environment.

Even though the concept of enlarging obstacles makes intuitive sense it is important to consider problems that might arise with enlarging obstacles too much or too little. If an obstacle is enlarged too much the multicopter might be forced to take sub-optimal routes as shown in Figure 5.4 when the robot's position uncertainty was made too large. Conversely, if the obstacle is enlarged too little or not at all the multicopter's navigation system might choose a path which will end in it colliding with an obstacle. To ensure that the multicopter will never collide with an obstacle it is suggested that r_{enlarged} be chosen to be as small as possible without underestimating any of the terms in Equation 5.4.

5.3 Compensating for Multicopter Tilt Angle

One way in which multicopters are entirely different from mobile ground robots is that they use pitch and roll manoeuvres to move forward or to the sides. This also means that unlike mobile ground robots they do not have to change their orientations (yaw) to change direction as they can easily move in any direction through a combination of pitch and roll manoeuvres.

A potential problem that arises with implementing VFH* on a multicopter using a two-dimensional LIDAR range sensor is that if a gimbal is not used the tilt angle of the drone will affect the range measurements made by the sensor. Figure 5.5 shows a multicopter as seen from the side. If the multicopter is

stationary while taking a range measurement the true distance to the object would be the distance R_1 , ignoring sensor inaccuracies of a few centimetres. However, as the multicopter starts to move either towards the target, or away from it, a tilt angle θ_{tilt} is introduced and the sensor range measurement s would indicate that the object is further away from the multicopter than it actually is. This error is dependant on the obstacle distance to the multicopter and may even be insignificant if the multicopter travels at low speeds, causing small tilt angles (θ_{tilt}), or if the obstacle is very close to the multicopter, causing short range readings (s).

The effective range of an obstacle R_1 is calculated as:

$$R_1 = s \cdot \cos(\theta_{\text{tilt}}) \quad (5.7)$$

The concept is exactly the same for pitch and roll manoeuvres. Thus, the error caused by a multicopter's tilt angle during either pitch or roll manoeuvres can easily be corrected by calculating the effective range R_1 and using the effective range instead of the actual measured range s to update the histogram grid C . While the one end of the multicopter might be lifted by an angle θ_{tilt} the other end will be pointing down by an angle θ_{tilt} and every other range reading will be at an angle in the range $0^\circ \leq \theta < \theta_{\text{tilt}}$.

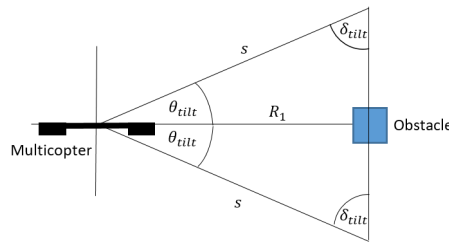


Figure 5.5: Effect of tilt angle on range readings (side view).

Figure 5.6 shows a simulation of the VFH* where the tilt angle was varied from 0° to 60° in increments of 15° . The obstacle is shown in the position where the two-dimensional LIDAR sensor initially measures it to be before correcting the measured distance. Thus the obstacle would make up a region much closer to the multicopter on the histogram grid for each of the cases where the tilt angle is greater than zero. In all the cases the multicopter only reacts to the obstacle once it is detected in its active region. The effect of the tilt angle correction can clearly be seen as the multicopter starts reacting sooner on the figure as the tilt angle increases. As the tilt angle is increased the measured distance is corrected and the object is detected closer to the multicopter's starting position than what was initially measured.

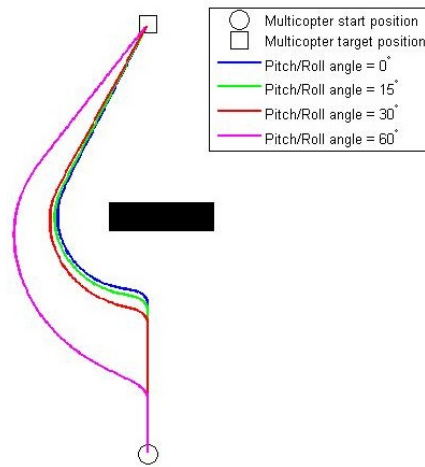


Figure 5.6: Effect of tilt angle on multicopter trajectory.

5.4 Compensating for Multicopter Yaw Uncertainties

Uncertainties in a multicopter's estimated yaw angle affect the VFH* algorithm in a similar fashion as pitch and roll do. Figure 5.7 shows a top view of a multicopter with the yaw uncertainty, $\theta_{\Delta yaw}$, defined as the angle by which the multicopter may differ from its estimated yaw angle. This is different from the roll and pitch angles in that errors in yaw angle estimation occur at random and have an influence on the perceived distance, position and width of an obstacle.

To account for the effect yaw uncertainty has on the perceived distance of an obstacle to the multicopter the process followed in Section 5.3 is repeated as shown:

$$R_2 = s \cdot \cos(\theta_{\Delta yaw}) \quad (5.8)$$

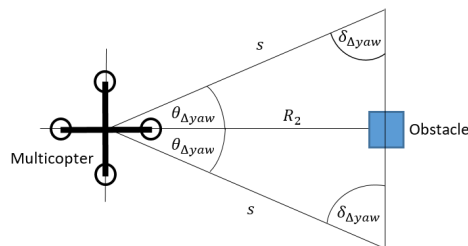


Figure 5.7: Effect of yaw angle on range readings and obstacle uncertainty (top view).

R_2 is then the corrected distance from the multicopter to an obstacle. However if the multicopter has a tilt angle as shown in Figure 5.8 then we also need to account for θ_{tilt} when calculating the corrected distance, R_2 :

$$R_2 = s \cdot \cos(\theta_{\text{tilt}}) \cdot \cos(\theta_{\Delta\text{yaw}}) \quad (5.9)$$

To account for the effect yaw uncertainty has on the width of an obstacle we include Δx , as shown in Figure 5.8, in the enlargement angle $\gamma_{i,j}$ as shown:

$$\Delta x = R_1 \cdot \sin(\theta_{\Delta\text{yaw}}) \quad (5.10)$$

then

$$\Delta x = s \cdot \cos(\theta_{\text{tilt}}) \cdot \sin(\theta_{\Delta\text{yaw}}) \quad (5.11)$$

$$r_{\text{enlarged}} = r_{\text{multicopter}} + d_{\text{safety}} + r_{\text{position}} + \Delta x \quad (5.12)$$

$$\gamma_{i,j} = \arcsin\left(\frac{r_{\text{enlarged}}}{d_{\text{enlarged}}}\right) \quad (5.13)$$

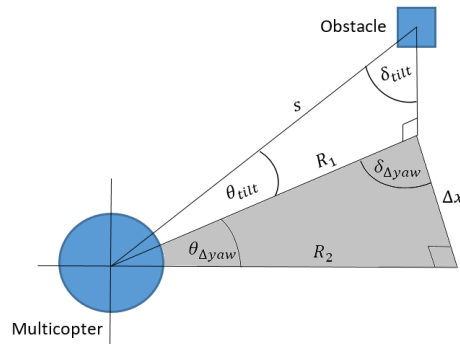


Figure 5.8: Combined effect of tilt and yaw angles on range readings and obstacle uncertainty.

The modified VFH* algorithm might lead to longer total path lengths compared to the original VFH* due to an over enlargement of obstacles. However, the enlargement of obstacles is required to ensure multicopters using the VFH* algorithm do not collide with obstacles when affected by uncertainties. Thus, even though longer path lengths may result from the modifications they are necessary to ensure that multicopters do not collide with obstacles.

5.5 Conclusion

The effects of position and yaw uncertainties were discussed in this chapter. The influence of tilt angles on the range measurement of obstacles when a two-dimensional LIDAR system is used was also addressed. It was found that the

VFH* algorithm can be modified to account for position and yaw uncertainties. It was shown that the algorithm can also be modified to consider the tilt angle of a multicopter as it builds its histogram grid with range measurements.

Without the modifications presented the original VFH* algorithm would not be capable of traversing the obstacle courses shown in Figure 5.3, 5.4 and 5.6 without there existing a strong possibility of a collision with an obstacle. There is no significant difference in the computational time of the improved VFH* algorithm compared to the original one due to the simplistic nature of the modifications in that only the range readings used to update the histogram grid and polar histogram grid are altered.

Chapter 6

Hardware and Software

Various hardware and software were used to implement the VFH* algorithm on a multicopter. A Robot Operating System (ROS) network was used to connect all the different hardware components. The network consisted of three main components, the Pixhawk flight controller unit (FCU), a LIDAR, and the obstacle avoidance algorithm (run on a PC with MATLAB). These components were all connected to an Intel Edison microcomputer which hosted the ROS master as shown in Figure 6.1. The ROS master manages all the communication between the different parts of the ROS network.

The multicopter's FCU was connected to the Intel Edison by means of a UART connection and the Intel Edison was placed on the multicopter itself. To detect obstacles a LIDAR was used which was controlled by an Arduino MEGA which hosted a ROS node that communicates with the ROS master. The Arduino MEGA was connected to the Intel Edison by means of an USB connection. The VFH* algorithm was run as a ROS node on a computer that communicated with the ROS master by means of WiFi.

The Pixhawk FCU sends information to the Intel Edison about all the multicopter parameters including the multicopter's position and orientation. This information can then be accessed by other nodes on the ROS network. The LIDAR publishes its range measurements and angles to the Intel Edison. This allows the VFH* node on the network to access information on the multicopter's position and orientation as well as update the histogram grid with information from the LIDAR sensor. After determining the new direction of motion the VFH* publishes new set-point values to the Pixhawk FCU through the Intel Edison. Consequently the Pixhawk FCU steers the multicopter to the new set-point position values. The core concepts of the ROS network are explained in the next section, followed by discussions of the main components that form part of the ROS network.

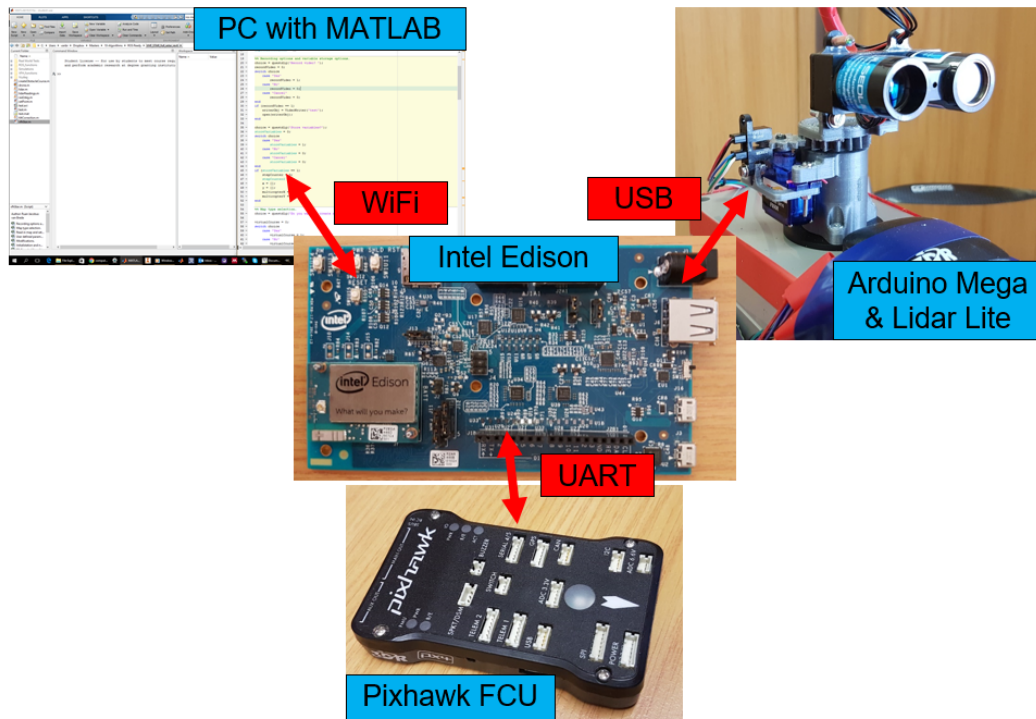


Figure 6.1: Hardware interface overview.

6.1 Robot Operating System (ROS)

Robot Operating System (ROS) is an open source framework that allows the relatively simple writing of robot software. It was created with the aim of encouraging collaborative robotics software development by providing a collection of tools, libraries and conventions (ROS, 2016). This allows for the creation of complex and robust robot behaviour across a wide variety of platforms with relative ease (ROS, 2016).

All the different parts of the ROS network are coordinated by a ROS master. The ROS master has its own unique URI (Uniform Resource Identifier) that specifies the IP address of the machine that hosts the master. All nodes, topics and services in the ROS network are registered with the ROS master as shown in Figure 6.2. If the nodes are not registered with the master node the nodes will not be able to find one another, exchange messages, or invoke services (ROS-Wiki, 2016).

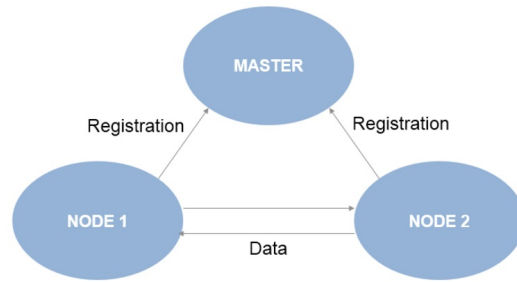


Figure 6.2: Node registration to ROS master (Mathworks, 2016).

A ROS network consists of various nodes. Each node is an entity that has a collection of ROS capabilities; publishers, subscribers and services. Nodes communicate through topics. A ROS network may have multiple topics and each topic may have multiple publishers and subscribers. Nodes that send messages to a topic by publishing to that topic are called publishers; nodes that receive messages from a topic by subscribing to that topic are called subscribers.

Publishers and subscribers are one way operations and are thus not appropriate for request/reply interactions. Request/reply operations are thus done through services. A service is defined by a pair of message structures, one for request and one for reply. A node may thus provide services which a client node might request and await a response from (ROS-Wiki, 2016).

For this research the ROS network consisted of a PC running MATLAB, a Intel Edison micro computer, and a Arduino MEGA connected to a LIDAR-Lite sensor. The network that was used is shown in Figure 6.3 where the three main nodes all register with the ROS master on the Intel Edison.

The Pixhawk FCU's parameters were synchronized with the ROS master. To access the multicopter's position parameters the VFH* algorithm was subscribed to the `/mavros/local_position/pose` topic as illustrated in Figure 6.4. This allowed the VFH* algorithm to determine the multicopter's position and orientation on the histogram grid. Similarly, the Arduino published the range and angle data from each Lidar-Lite measurement to the `/lidarReadings` topic hosted on the Intel Edison as illustrated in Figure 6.5. The VFH* algorithm was subscribed to the `/lidarReadings` topic and used the data from the topic to update the histogram grid.

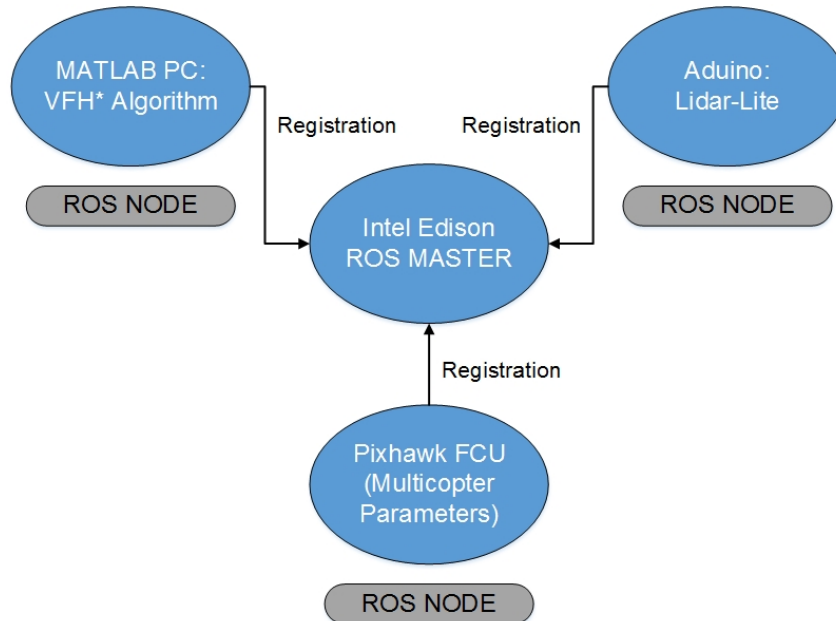


Figure 6.3: ROS network overview.

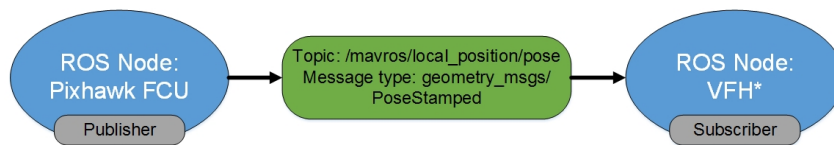


Figure 6.4: `/mavros/local_position/pose` topic with Pixhawk as publisher and VFH* as subscriber.

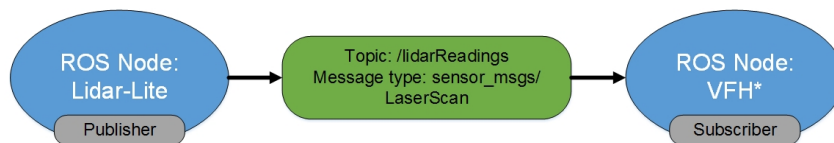


Figure 6.5: `/lidarReadings` topic with Lidar-Lite as publisher and VFH* as subscriber.

The histogram data was then reduced to find the primary candidate directions. After the optimal candidate direction was determined new setpoint values were calculated and published to the `/newSetPoints` topic as illustrated in Figure 6.6. A node called Setpoints was subscribed to the `/newSetPoints` topic. The Setpoints node repeatedly sent the new setpoint values to the `/mavros/setpoint_position/local` topic to which the Pixhawk FCU was subscribed as illustrated in Figure 6.7. This allowed the VFH* algorithm to

continue updating the histogram grid as the multicopter moved without interrupting the transmission of the setpoints. The Setpoints node continued to publish the setpoint values to the FCU until the multicopter reached its new coordinates. This process was repeated multiple times until the multicopter reached its goal position.

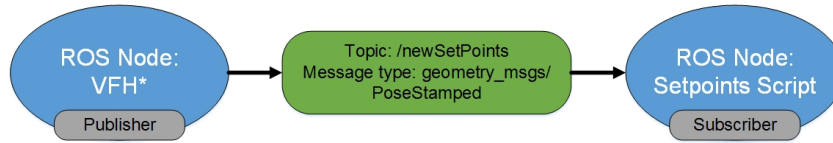


Figure 6.6: */newSetPoints* topic with VFH* as publisher and Setpoints node as subscriber.

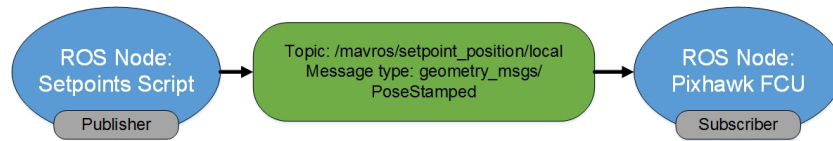


Figure 6.7: */mavros/setpoint_position/local* topic with Setpoints node as publisher and Pixhawk as subscriber.

6.2 Intel Edison

The Intel Edison was created for rapid prototyping and uses Ubilinux as an operating system. It provides various communication interfaces such as WiFi, UART and USB. The Intel Edison was chosen for its small size, low power consumption (35 mW) and its compatibility with ROS. For more detail on the Intel Edison the reader is referred to Appendix C.

The ROS master, which manages all the network communication, was hosted on the Intel Edison. Additionally, a Python script was used to run the Setpoints node on the Intel Edison. The script, called `sendSetPoints`, was used to ensure that a continuous stream of setpoint values were sent to the Pixhawk FCU. When new setpoint values were published by the VFH* algorithm to the */newSetPoints* topic the Setpoints node received the values and continuously published the setpoints to the FCU, as illustrated in Figure 6.6 and 6.7 respectively. This process was repeated until either new setpoint values were calculated by the VFH* algorithm or the goal had been reached.

6.3 Pixhawk Flight Control Unit (FCU)

The Pixhawk is an autopilot and controller that is suitable for mobile robotics platforms such as fixed-wing aircraft, cars and multicopters. It has a 168 MHz Cortex-M4 CPU with relatively low power consumption (250 mW). For more information the Pixhawk's data sheet can be found in Appendix F.

The Pixhawk is especially suited for research as users have full access to all the controller's parameters and the Pixhawk can be integrated into a ROS network with relative ease. Except for an external GPS that connects directly to the Pixhawk, the Pixhawk also provides a number of on-board sensors; a 3D gyro, accelerometer, magnetometer, and pressure sensor (barometer).

The Pixhawk uses the data from the sensors in combination to determine the multicopter's position and orientation. It also uses this information to control the multicopter's rotor speeds. The Pixhawk provides four main modes; manual, altitude-hold, position-hold and off-board mode. In manual mode the user uses a radio transmitter to change the multicopter's elevation, orientation, and position. In altitude-hold mode the Pixhawk maintains the multicopter's elevation while the user only has to adjust the multicopter's position and orientation. Similarly in position-hold mode the Pixhawk maintains the multicopter's elevation but also its position. Sometimes position-hold mode is also called GPS-hold.

Off-board mode is used in this study. In off-board mode the Pixhawk receives set-point values for its elevation, position and orientation and uses the control parameters used for position-hold mode to move to the specified position and maintain that position.

This is convenient since it separates the controller and the obstacle avoidance algorithm from one another. The VFH* algorithm publishes set-points to the Pixhawk which guides the robot to that set-point. Thus, the algorithm can be used for different controllers and does not actively participate in controlling the multicopter, except for issuing new position coordinates to the controller.

In the ROS network the Pixhawk publishes its parameters to the ROS master on the Intel Edison. If the connection between the Pixhawk and Intel Edison is lost the Pixhawk automatically switches back to position-hold mode.

6.4 MATLAB

MATLAB was used to program the VFH* algorithm. MATLAB was chosen because its Robotics System Toolbox provides an interface that allows MATLAB to communicate with, and become part of, a ROS network. This allows the user to easily develop, test and verify algorithms, especially those for autonomous mobile robotics applications (Mathworks, 2016). MATLAB also provides useful robot simulators such as Gazebo and V-REP that can be used to verify robotics algorithms. Additionally, MATLAB Embedded Coder

is another useful feature that allows the generation of C++ code from the existing MATLAB algorithm files.

A basic overview of the flow of the program used to implement the VFH* algorithm on a multicopter is shown in Figure 6.8. The multicopter's initial position and orientation was obtained from the `/mavros/local_position/pose` topic as previously mentioned. This served as a frame of reference for the multicopter. The algorithm then set its current position as its start position and stored any offsets in the position values that may have been caused by GPS drift. Then setpoint values were published to the Pixhawk FCU so that the multicopter could take-off to a pre-set height. The multicopter maintained its height until it eventually landed. After take-off the multicopter's environment was scanned by the two-dimensional LIDAR and its histogram was updated. The VFH* algorithm as described in Chapter 3 was then used to determine a new set of position coordinates that were sent to the Pixhawk FCU, for the multicopter to transition to. If the multicopter reached its goal position the program was ended by publishing setpoint values to the FCU to land the multicopter.

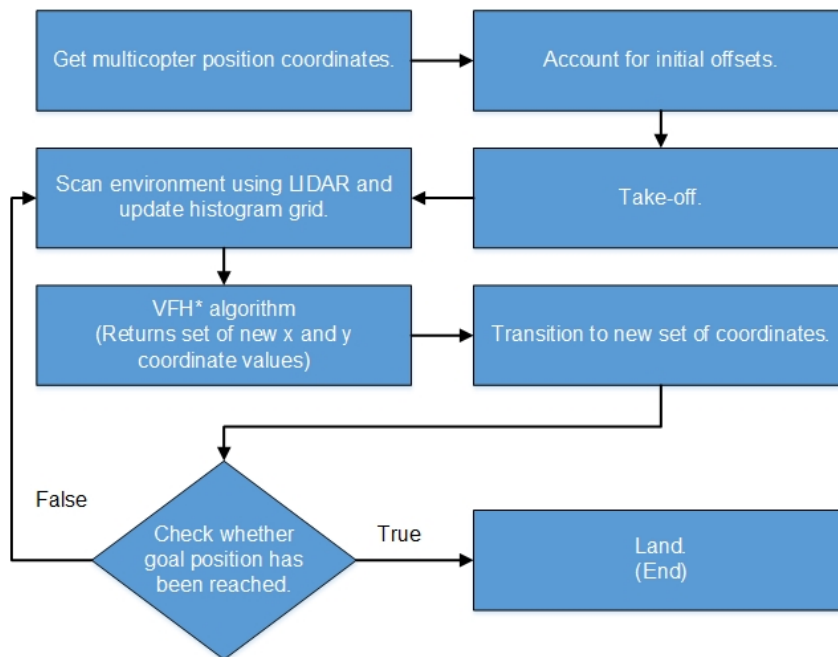


Figure 6.8: Obstacle avoidance algorithm overview.

6.5 Lidar-Lite

A two-dimensional LIDAR was used to update the histogram grid. The Pulsed-Light Lidar-Lite was selected due to its affordability, acceptable accuracy(\pm -

2.5 cm) and range (30 m - 60 m). For additional information the Lidar-Lite's data sheet can be viewed in Appendix D.

The Lidar-Lite was used to detect obstacles and update the histogram grid. It was controlled by an Arduino MEGA that has a 16 MHz ATmega 2560 processor. The ROS node on the Arduino published range and angle data to the */lidarReadings* topic. The acquisition time of the Lidar-Lite is less than 0.02 seconds.

A stepper motor was used to turn the Lidar-Lite and a pick-up sensor was used to determine the angle in which the LIDAR was turned/pointing. An Arduino MEGA was chosen since it could be integrated into the ROS network with relative ease and supported the Lidar-Lite sensor. The Arduino MEGA's data sheet can be viewed in Appendix E.

A callback function was used to receive data from the Lidar-Lite. Whenever the Lidar-Lite detected an obstacle (i.e. makes a range measurement within 20m of the multicopter) the callback function was triggered and the range readings and angles were sent to the VFH* algorithm on the PC which in turn updated the histogram grid with the new information.

The LIDAR configuration was designed so that it can be used for more than one project. Thus it was designed to be plug and play, i.e. it can just be connected to the ROS network and it will start publishing to its topic. The topic could then be accessed by other nodes in the ROS network.

6.6 Conclusion

An overview of all the different hardware and software components was provided in this chapter. ROS was used to integrate the various hardware components needed for autonomous flight and obstacle avoidance. The use of ROS and modular parts of the network has laid the framework for future work and research as different algorithms and sensor pairings can be implemented and tested. Additionally, the use of MATLAB has also made it convenient to make changes or modifications to code allowing in the field debugging.

Chapter 7

Implementation: Test Flights

The VFH* algorithm, with some of the modifications proposed in Chapter 5, was implemented on the multicopter as was described in Chapter 6. Position uncertainty was considered, however, yaw and tilt angles were not considered for these tests since the controller parameters used for these tests did not result in large changes in the multicopter's pitch and roll angles. The value of the position uncertainty r_{position} was increased to compensate for neglecting these parameters.

Test flights were performed to confirm that the prototype obstacle avoidance system worked as intended. A Real Time Kinematic (RTK) GPS was placed on the multicopter to accurately determine the multicopter's position in space while the Pixhawk and VFH* algorithm used the Pixhawk's estimated position to navigate the obstacle courses.

The Real Time Kinematic (RTK) GPS is more accurate than the Pixhawk's position estimation since it uses an additional ground station to more accurately determine its position (a horizontal accuracy of up to 2 cm). The RTK GPS's data sheet can be viewed in Appendix H.

At the start of each test the multicopter's yaw angle was locked. The VFH* algorithm's axes were aligned with the multicopter's orientation so that the positive x-direction was in the multicopter's forward direction and the positive y-direction was to the left of the multicopter. In all the test cases the VFH* algorithm was given the multicopter's current position as its start position and given a goal position 20 m in the direction of the algorithm's x-axis (multicopter's forward direction).

The multicopter with the attached hardware (Lidar-Lite, Arduino MEGA and Intel Edison) is shown in Figure 7.1. The test setup is shown in Figure 7.2. Panels were used to construct obstacle courses for the multicopter as shown in Figure 7.2. Table 7.1 shows the VFH* algorithm parameters that were used for the test flights.



Figure 7.1: Multicopter with Intel Edison and Arduino encased with the Lidar-Lite mounted on-top.



Figure 7.2: Test setup with two panels used as obstacles.

Table 7.1: VFH* algorithm test parameters.

Parameter	Unit	Value
Cell size	[m]	1
Active window width	[m]	5
s_{\max}	[–]	18
α	[°]	5
$r_{\text{multicopter}}$	[m]	0.6
r_{safety}	[m]	1
r_{position}	[m]	2.5
n_g	[–]	10

7.1 Results

The test setup used for Test 1 contained two panels placed in front of the multicopter with a gap in between as shown in Figure 7.3. The obstacle avoidance system successfully guided the robot around the obstacles. In all the test figures GPS refers to the Pixhawk’s position estimation.

However, the multicopter made swirling movements as it moved towards the goal. These swirling movements were caused by what is called the Toilet Bowl effect. The Toilet Bowl effect is common with multicopters and entails the multicopter making clockwise or counter-clockwise swirl movements. This is often caused by incorrectly tuned parameters or the multicopter’s compass that requires recalibration. It was found that tuning the Pixhawk controller’s parameters significantly reduced this effect, as can be seen in all the test results to follow.

A similar test setup was used for Test 2 (however the test was done after the controller’s parameters had been tuned). The test is shown in Figure 7.4. The obstacle avoidance system was able to successfully guide the multicopter around the obstacles. The GPS and RTK paths closely resembled one another. The region where the RTK GPS is a straight line shows a stretch where the RTK GPS lost its connection to its base station and then regained connection shortly thereafter. It can be observed that the Toilet Bowl effect was significantly less. This allowed the multicopter to reach the goal with a much smoother trajectory than before and also faster.

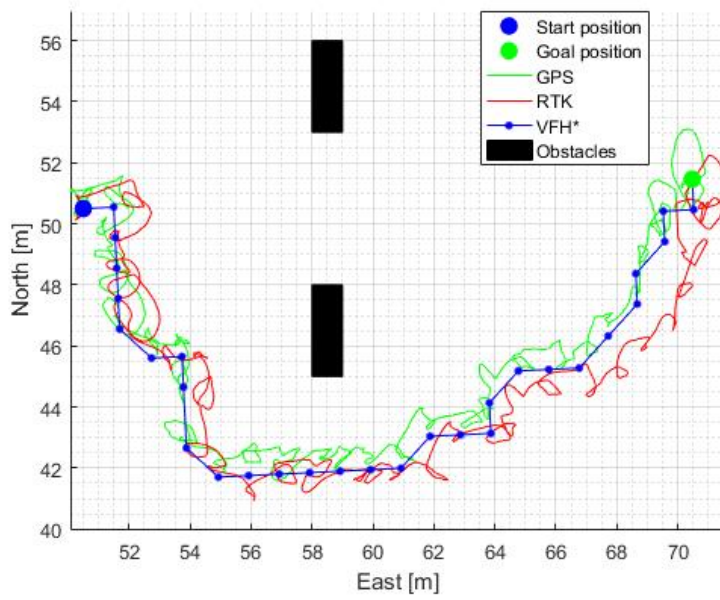


Figure 7.3: Test 1 (gap size 5 m).

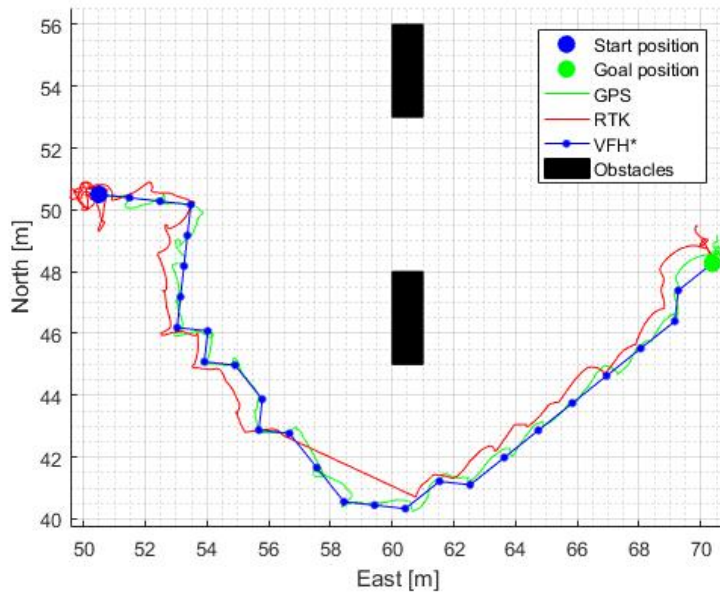


Figure 7.4: Test 2 (gap size 5 m).

Test 3 used a similar obstacle setup as the previous tests. However, the distance between the two panels was increased as shown in Figure 7.5. Using the prototype obstacle avoidance system the multicopter was successfully guided through the gap between the obstacles.

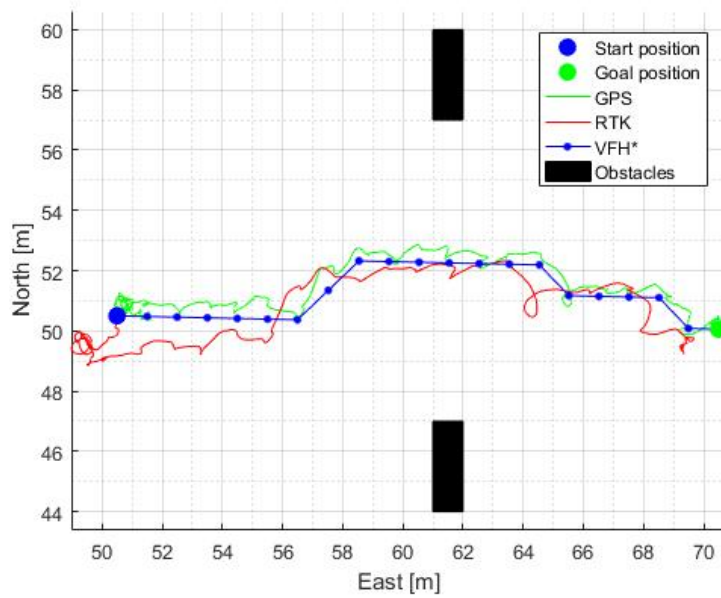


Figure 7.5: Test 3 (gap size 10 m).

For Test 4 a second run was done on the test setup used for Test 3 with the two panels placed in front of the multicopter with a gap in between as shown in Figure 7.6. This time, however, the multicopter was initially placed slightly more skew with regard to the obstacles resulting in it landing behind the obstacle south of the multicopter's starting position. However, regardless of the skew orientation the obstacle avoidance system guided the robot safely through the obstacles.

The test setup was altered again for Test 5. The obstacles were moved closer together than when the multicopter was guided through the gap, but further apart than when the multicopter was guided around the obstacles. The test is shown in Figure 7.7. The obstacle avoidance system initially guided the multicopter towards the gap between the obstacles. However, once as the multicopter neared the obstacles the obstacles avoidance system changed the multicopter's route and guided it around the obstacles. Then, as the multicopter moved around the obstacle the algorithm guided the multicopter in a circle and thereafter around the obstacle and to the goal.

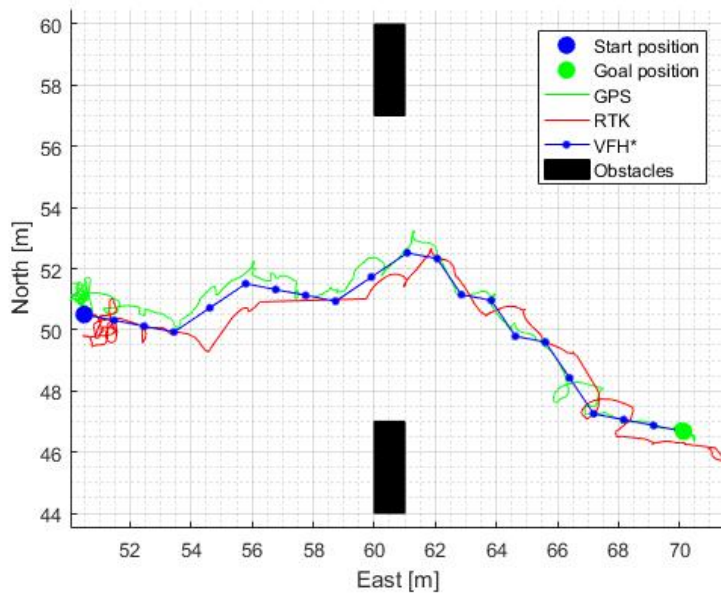


Figure 7.6: Test 4 (gap size 10 m).

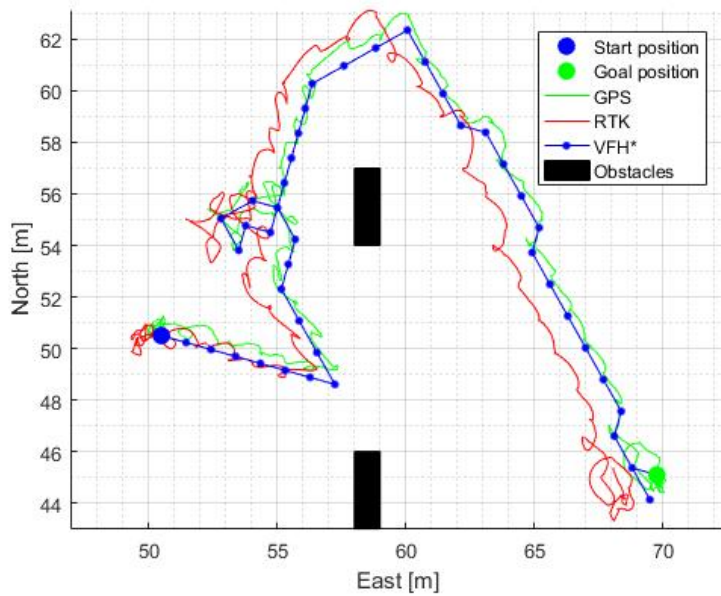


Figure 7.7: Test 5 (gap size 8 m).

To explain this behaviour Figure 7.8 illustrates the case where the multicopter was initially guided towards the gap. There were open sectors that lead to the goal position. However, as the multicopter neared the obstacles the enlargement angle $\gamma_{i,j}$ blocked the open sectors as illustrated in Figure 7.9.

Thus the algorithm started guiding the robot around the obstacle. The circular movement that was made thereafter is believed to have been caused by a misreading from the LIDAR detecting an obstacle where there was none. This would have potentially caused the algorithm to change the multicopters trajectory. However, subsequent LIDAR scans depreciated cells on the histogram grid, correcting the cells' certainty values that were increased by mistake. Thus, the algorithm was able to safely guide the multicopter around the obstacle once more.

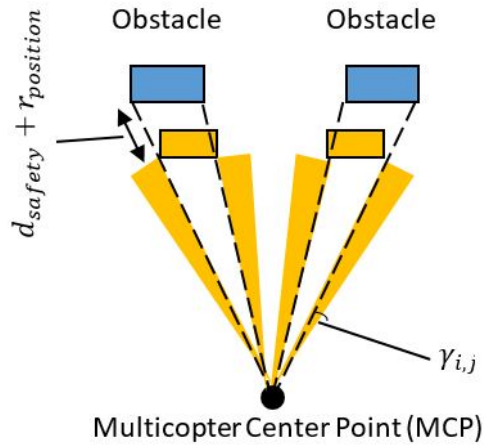


Figure 7.8: Multicopter approaching two obstacle panels.

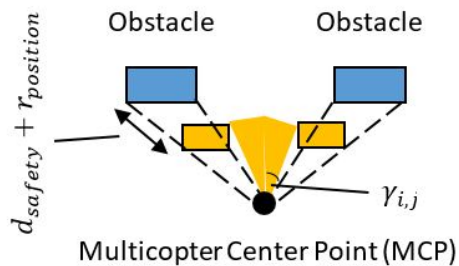


Figure 7.9: Multicopter's path blocked by two obstacles due to the enlargement angle $\gamma_{i,j}$.

Table 7.2: Tests average error and standard deviations.

	North		East	
	μ [m]	σ [m]	μ [m]	σ [m]
Test 1	-0.4115	0.2894	0.6085	0.3592
Test 2	-0.2555	0.3311	0.6684	0.2082
Test 3	-0.9194	0.5784	-0.0911	0.1626
Test 4	0.2327	0.3164	0.4611	0.2579
Test 5	0.2373	0.2820	-0.7810	0.2988

Table 7.2 shows the average error μ encountered in the multicopter's position along with the standard deviation σ for each of the test cases when the Pixhawk's position estimation data was compared to that of the RTK GPS. The average in all of the cases was less than 1 m though Test 3 had a rather large μ in the North direction. The standard deviation was only larger than 0.5 m for Test 3.

7.2 Conclusion

The prototype obstacle avoidance system, incorporating the VFH* algorithm with modifications from Chapter 5, successfully guided the multicopter to its goal position for all the test cases presented above. The Toilet Bowl effect was addressed by tuning the Pixhawk controller's parameters. In all the cases twirling movements were observed at the beginning and end. This is due to the multicopter taking-off and landing in those positions which meant it spent much more time in those positions. However the controller can not accurately keep the multicopter at any one point for more than an instance. Thus the multicopter "twirls" around the set position always being driven back towards the set position but never maintaining that position for an extended period.

The average error in the Pixhawk's position estimation for all the test cases was provided in Table 7.2. Implementing a more accurate GPS could reduce the average error, however the controller plays a big role in how well the multicopter can maintain its position, even when the multicopter's position is accurately known. Therefore, it is believed that an improved controller would reduce the standard deviation.

Chapter 8

Conclusion

The purpose of this study was to outline the different activities that contributed towards reaching the main objective of the thesis which was the development of a prototype obstacle avoidance system to be utilized by a multicopter to successfully avoid obstacles in a test environment.

A literature review on the various types of obstacle avoidance algorithms and sensors was done in Chapter 2. The main algorithms that were considered were discussed and after careful consideration the VFH* algorithm was selected. To implement the VFH* algorithm a two-dimensional LIDAR was selected as the sensor of choice due to its range capabilities, cost and the versatility of the sensor in varying weather and environmental conditions.

The VFH* algorithm was selected since it considers the multicopter's width and an additional safety region. Additionally, the VFH* algorithm implements look-ahead verification which allows the algorithm to select the most suitable candidate direction when more advanced obstacle configurations are encountered. Furthermore the algorithm could be modified to account for uncertainties with regard to the multicopter's position. A detailed discussion on how the VFH* algorithm works was done in Chapter 3.

The VFH* algorithm was coded in MATLAB and the code was verified through simulations in Chapter 4. After careful consideration it was decided that the VFH* algorithm would have to be modified to account for uncertainties with regard to the multicopter's position. The modifications were discussed in Chapter 5.

To implement the algorithm on a multicopter suitable hardware and software were selected. A ROS network was used to connect the different hardware components and each of the components was discussed in Chapter 6.

Finally the prototype obstacle avoidance system was implemented on a multicopter and tested in a test environment and the results were discussed in Chapter 7. The obstacle avoidance system was able to successfully guide the multicopter from its starting position to the goal for all of the test cases and thus the over-arching aim of the study was achieved.

This project contributed towards a framework for future work on this topic

by STERG and other researchers. The modular design of the prototype obstacle avoidance system allows the optimisation of individual parts rather than creating a whole new system.

Appendices

Appendix A

How Multicopters Work

Multicopters, or multirotors as they are also known, have multiple sets of rotors. A flight control unit (FCU) is used along with motor controllers to adjust the speed of each individual rotor to control a multicopter's movement, orientation and elevation. The FCU processes information from all of the multicopter's sensors before adjusting the speed of each rotor. A multicopter's elevation is determined by measuring the atmospheric pressure using a barometer. Additionally, a global positioning system (GPS) is used to determine the multicopter's position in space and accelerometers to measure changes in the multicopter's velocity. A gyroscope and magnetometer (compass) are used to estimate the multicopter's orientation.

A quadcopter (multicopter with two sets of rotors) is the simplest version of a multicopter and is used to illustrate the key concepts of how multicopters work. Figure A.1 shows the basic layout of a quadcopter; it has four rotors; two opposing rotors turn clockwise (rotor 1 and 4) while the other two turn counter-clockwise (rotor 2 and 3). When all four the rotors turn at the same speed the angular momentum of the rotors are in equilibrium which results in the multicopter maintaining its orientation and elevation.

By increasing the speed of all four rotors the quadcopter's lift force is increased as shown in Figure A.2a. If the lift force generated is greater than the gravitational force on the quadcopter the quadcopter will increase its elevation, while if it is less than the gravitational force it will decrease in elevation as shown in Figure A.2b. The quadcopter hovers at a constant elevation when lift and gravitational forces are in equilibrium .

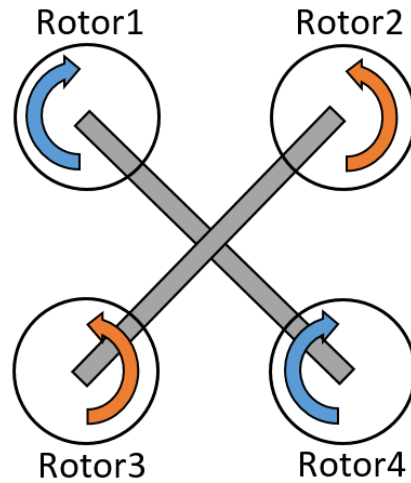


Figure A.1: Primary polar histogram constructed around robot.

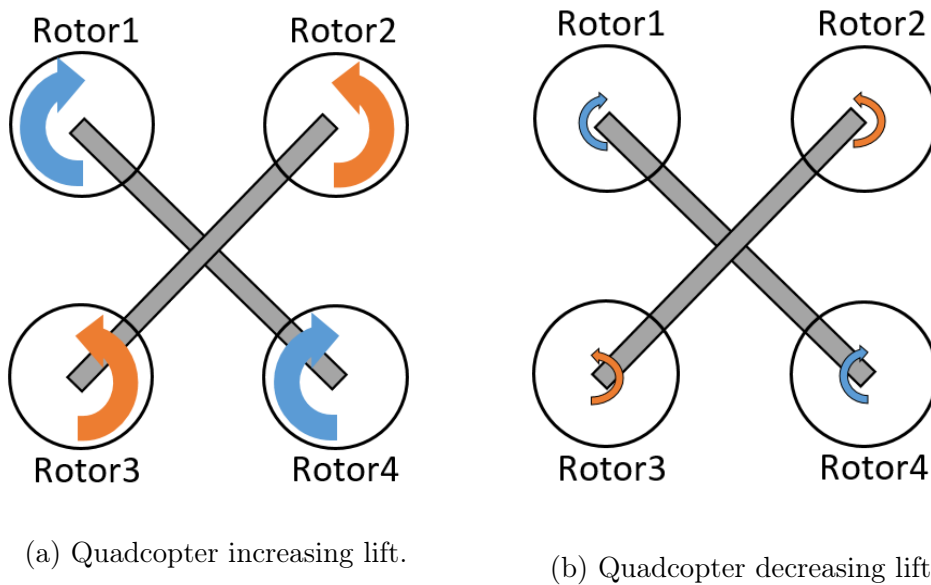


Figure A.2: Quadcopter lift manoeuvres.

The quadcopter moves about and changes its orientation using a combination of pitch, roll and yaw manoeuvres. To move forward the robot increases its pitch by increasing the speed of rotors 3 and 4 relative to rotors 1 and 2 as illustrated in Figure A.3a. Conversely, to move backwards the speed of rotors 1 and 2 are increased relative to rotors 3 and 4 as illustrated in Figure A.3b.

The quadcopter moves left or right by performing roll manoeuvres. To roll to the right the quadcopter increases the speed of rotors 1 and 3 relative to rotors 2 and 4 as illustrated in Figure A.4a. Conversely, to roll to the left the

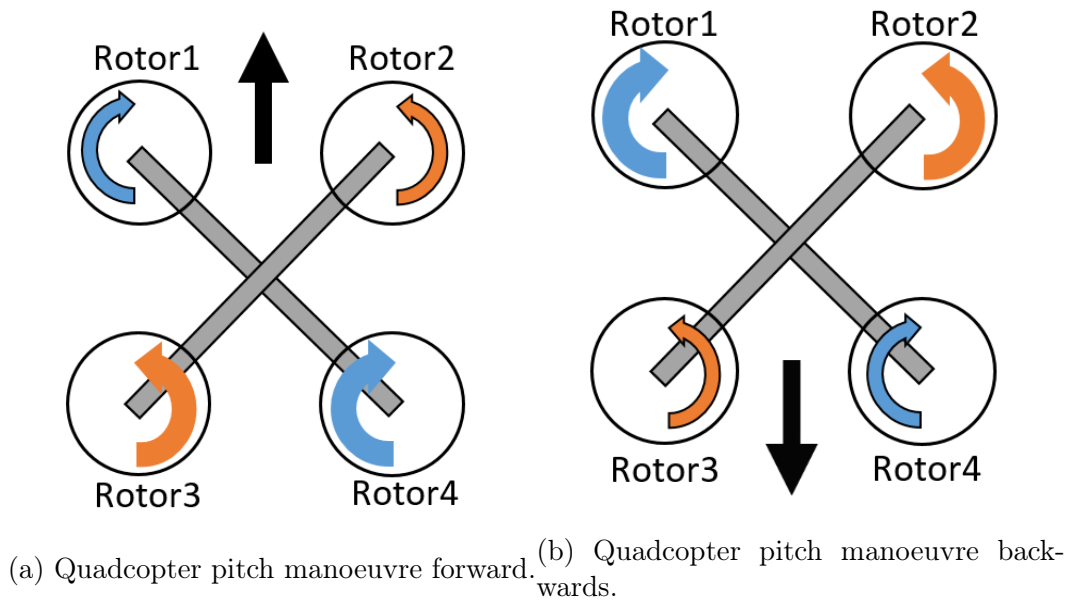


Figure A.3: Quadcopter performing pitch manoeuvres.

quadcopter increases the speed of rotors 2 and 4 relative to rotors 1 and 3 as illustrated in Figure A.4b.

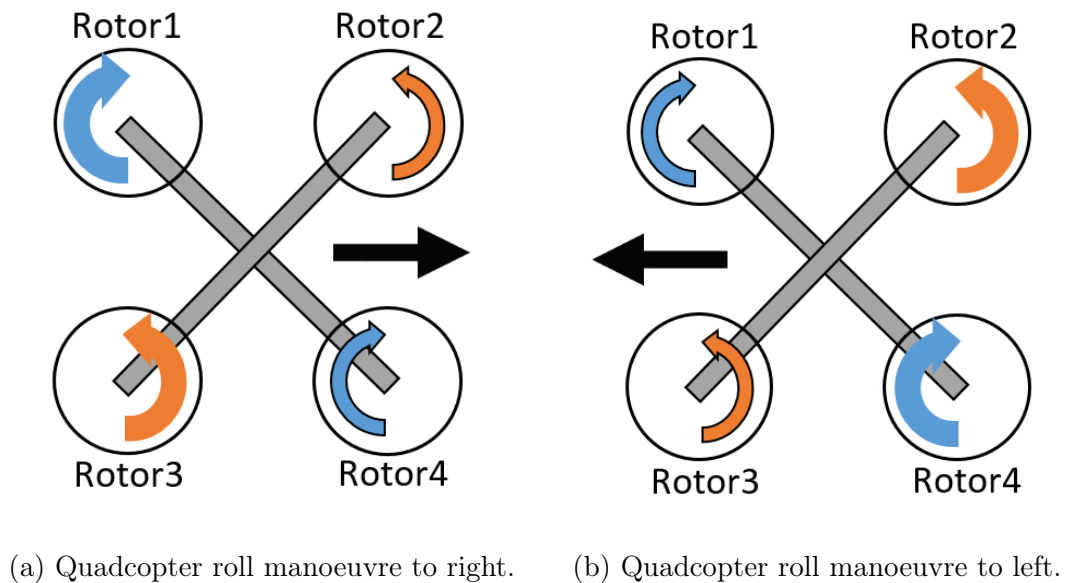
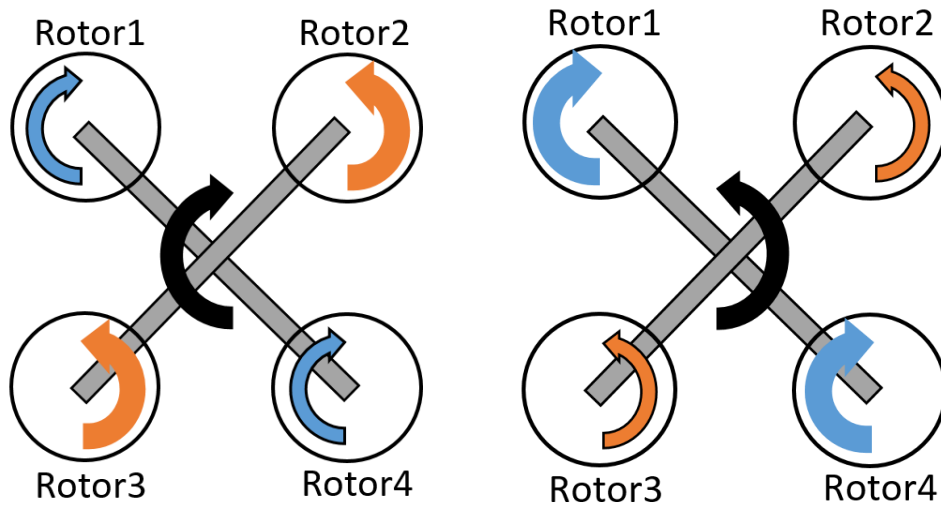


Figure A.4: Quadcopter performing roll manoeuvres.

A yaw manoeuvre is performed by increasing the speed of either the clockwise or counter-clockwise turning rotors. Due to the conservation of angular momentum of the system, the multicopter will rotate in the opposite direction

than what the faster rotors are turning. For example, to yaw clockwise the speed of rotors 2 and 3 are increased relative to rotor 1 and 4 as illustrated in Figure A.5a. To yaw counter-clockwise the speed of rotors 1 and 4 are increased relative to rotor 2 and 3 as illustrated in Figure A.5b.



(a) Quadcopter clockwise yaw manoeuvre. (b) Quadcopter counter-clockwise yaw manoeuvre.

Figure A.5: Quadcopter performing yaw manoeuvres.

Appendix B

A* Search Algorithm

The A* search algorithm is a global path planning algorithm. It uses existing map information to determine paths to the goal and selects one based on a cost function f_n . The algorithm works by expanding nodes, starting at the start node n_{start} . Subsequent nodes are each assigned a cost f_{n_i} based on the path cost $g(n_i)$ to reach that node n_i and a heuristic $h(n_i)$, which is an estimate of the cost to reach the goal from that node. A priority queue is used to keep track of which nodes have been expanded and which still need to be expanded with the nodes with lowest cost being expanded first. Nodes are expanded until the goal node n_{goal} is reached. The search ends when either all nodes have been expanded or no path exists with a lower cost than the one that has already been found.

To compare search algorithms the term efficient is used as a measure of the search, i.e. the number of nodes visited to determine the path, while optimal refers to a measure of the path quality itself (Howie Choset *et al.*, 2005). The efficiency of the search and quality of the path is influenced by the selection of the heuristic $h(n)$ used for estimating the estimated cost $f(n)$.

The A* search algorithm differs from other graph search methods, such as the depth-first and breadth-first methods that arbitrarily search all nodes until the goal node is found. The A* search uses prior knowledge of the goal node's position when expanding its search to more efficiently determine a path to the goal (Howie Choset *et al.*, 2005). Although the breadth-first and depth-first searches are not as efficient as the A* search the breadth-first search can find the optimal path since it selects the path with the shortest path length. The heuristic function is described in more detail in the section 3.6.2 and is followed by an example of the A* search algorithm at work in section B.2.

B.1 Heuristic

A heuristic is used to hypothesise the expected cost $h(n)$ of reaching the goal node from the current node n_i . Various heuristics exist, for example, a robot

might use the shortest Euclidean distance to the goal as heuristic to decide which node to traverse to next. However, there is no guarantee that this heuristic will lead to the selection of the node with the shortest path to the goal (Howie Choset *et al.*, 2005). This is due to the fact that the other factors, such as the robot's dynamics for example, might influence the robot's path. Thus, a heuristic is used to facilitate an educated guess of the optimal path based on information available on the robot and its environment.

If a heuristic is "good" the search will be efficient. However, if the heuristic is "bad" the search will most likely take longer and the path will likely be sub-optimal (Howie Choset *et al.*, 2005). For the A* search algorithm to produce an optimal path its heuristic has to be admissible.

An heuristic is admissible if it never overestimates the cost to reach the goal node (Ulrich & Borenstein, 2000). For example the Euclidean distance to the goal, as was mentioned above, can be used as a heuristic. This is illustrated in Figure B.1 where the hypotenuse of the triangle is the shortest Euclidean distance to the goal, while the actual path the robot would follow is indicated as the other two sides of the right triangle. In this case the heuristic would always predict a value less than or equal to the shortest path making it admissible (Howie Choset *et al.*, 2005).

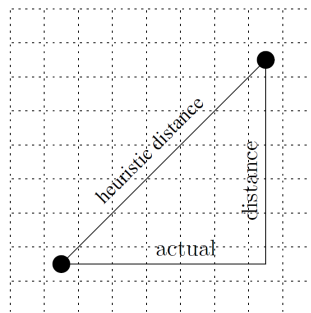


Figure B.1: Example of heuristic: Euclidean distance to the goal (Howie Choset *et al.*, 2005).

A heuristic does not always have to find the optimal path based on the shortest path length to a goal. Many other metrics exist that can be used either individually or in combination. For example, one might wish to find the optimal path with respect to some other metric such as transferability, energy, time, safety, etc. (Howie Choset *et al.*, 2005).

B.2 A* Search Algorithm Example

The A* search algorithm uses a priority queue to determine the optimal path to the goal node. Nodes are sorted by priority, i.e. a node with a lower cost $f(n)$ will have higher priority. Each node n has a cost defined as $f(n) = g(n) + h(n)$

where $g(c)$ is the path cost from the start node to node n and the heuristic $h(n)$ is the expected cost of reaching the goal node from node n (Howie Choset *et al.*, 2005).

The process followed to expand the search and find the optimal path to the goal is discussed using Figure B.2. Initially, the first node is put into the priority queue. To expand the search the start node is popped (removed from priority queue) and its adjacent nodes are added to the priority queue. The node with the highest priority, i.e. lowest cost $f(n)$, is popped next and removed from the priority queue. Node B has the highest priority and is thus popped and thus the priority queue is expanded with nodes G,H,I. Next, node H is popped since it has the highest priority. However, H has no neighbouring nodes and thus no new nodes are added to the priority queue. Node A is popped next since it has the highest priority, consequently its adjacent nodes are added to the priority queue.

When node E is popped it results in a path to the goal with a path cost $f(n)$ of 5. The total cost is equal to the path cost $g(n)$ because the goal has been reached and thus $h(n) = 0$. Some nodes in priority queue have higher costs than E and are thus discarded since they cannot produce more optimal paths.

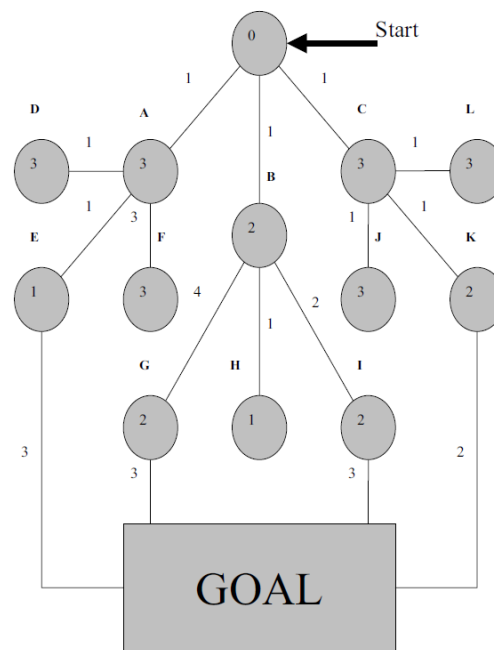


Figure B.2: Example to illustrate how the A* search algorithm determines the optimal path to the goal node (Howie Choset *et al.*, 2005).

The explicit path from goal to start node can be found by means of a series of back-pointers. This is illustrated in Figure B.3 where the priority queue

is shown at different stages. Initially A,B and C were the only nodes in the priority queue and therefore point back to the start node. After B was popped H,I, and G were added to the queue and point back to B. Thirdly, H could not be expanded on, while popping A resulted in D and F being added to the priority queue. When E was popped a path to the goal was found. Thus the path to the goal can be found using each node's back-pointer, i.e. goal, E ,A , start node .

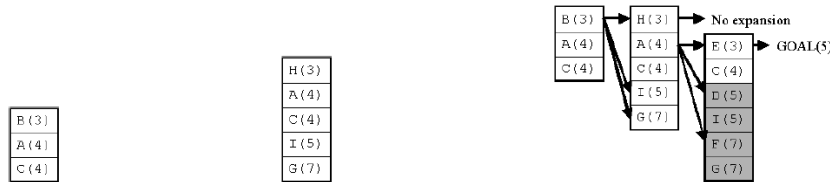


Figure B.3: Priority queue expansion (Howie Choset *et al.*, 2005).

The A* search algorithm will continue expanding nodes even if a path to the goal has been found since this path is not necessarily the optimal path; other nodes exist in the priority queue that have lower costs. This is illustrated in Figure B.4 where node C has been popped as the node with the highest priority and K,L, and J have been added to the priority queue. Subsequently, K has the highest priority and leads to the goal. Since no node on the priority queue has a lower cost to the goal node than the current path, the path is selected as the optimal one.

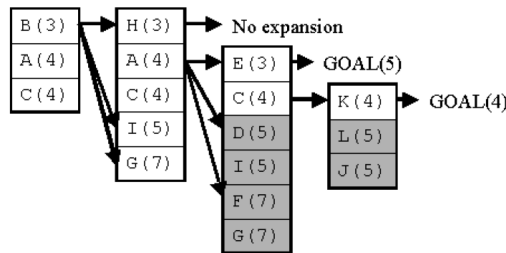


Figure B.4: Determining the optimal path from priority queue(Howie Choset *et al.*, 2005).

Appendix C

Intel Edison Datasheet

Product Brief
Intel® Edison



Intel® Edison Development Platform

Introduction

The Intel® Edison development platform is designed to lower the barriers to entry for a range of inventors, entrepreneurs, and consumer product designers to rapidly prototype and produce "Internet of Things" (IoT) and wearable computing products.

Intel® Edison Board for Arduino*

Supports Arduino Sketch, Linux, Wi-Fi, and Bluetooth.

Board I/O: Compatible with Arduino Uno (except 4 PWM instead of 6 PWM):

- 20 digital input/output pins, including 4 pins as PWM outputs.
- 6 analog inputs.
- 1 UART (Rx/Tx).
- 1 I²C.
- 1 ICSP 6-pin header (SPI).
- Micro USB device connector OR (via mechanical switch) dedicated standard size USB host Type-A connector.
- Micro USB device (connected to UART).
- SD card connector.
- DC power jack (7 to 15 VDC input).

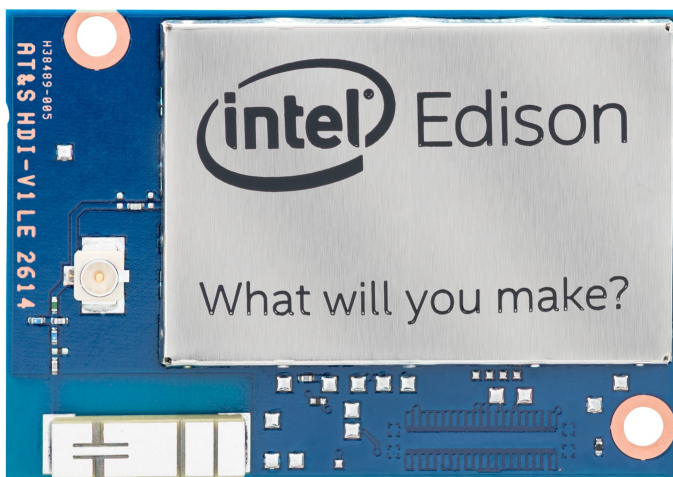
Intel® Edison Breakout Board

Slightly larger than the Intel® Edison module, the Intel® Edison Breakout Board has a minimal set of features:

- Exposes native 1.8 V I/O of the Edison module.
- 0.1 inch grid I/O array of through-hole solder points.
- USB OTG with USB Micro Type-AB connector.
- USB OTG power switch.
- Battery charger.
- USB to device UART bridge with USB micro Type-B connector.
- DC power supply jack (7 to 15 VDC input).

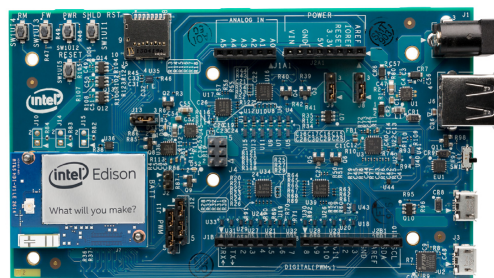
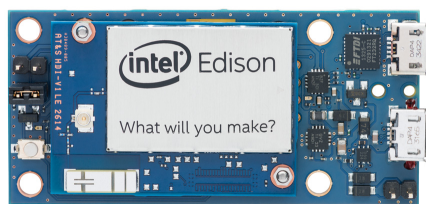
Intel® IoT Analytics Platform

- Provides seamless Device-to-Device and Device-to-Cloud communication.
- Ability to run rules on your data stream that trigger alerts based on advanced analytics.
- Foundational tools for collecting, storing, and processing data in the cloud.
- Free for limited and noncommercial use.



Intel® Edison Development Platform

PHYSICAL	
Form factor	Board with 70-pin connector
Dimensions	35.5 × 25.0 × 3.9 mm (1.4 × 1.0 × 0.15 inches) max
C/M/F	Blue PCB with shields / No enclosure
Connector	Hirose DF40 Series (1.5, 2.0, or 3.0 mm stack height)
Operating temperature	32 to 104°F (0 to 40°C)
EXTERNAL INTERFACES	
Total of 40 GPIOs, which can be configured as:	
SD card	1 interface
UART	2 controllers (1 full flow control, 1 Rx/Tx)
I2C	2 controllers
SPI	1 controller with 2 chip selects
I2S	1 controller
GPIO	Additional 12 (with 4 capable of PWM)
USB 2.0	1 OTG controller
Clock output	32 kHz, 19.2 MHz
MAJOR EDISON COMPONENTS	
SoC	22 nm Intel® SoC that includes a dual-core, dual-threaded Intel® Atom™ CPU at 500 MHz and a 32-bit Intel® Quark™ microcontroller at 100 MHz
RAM	1 GB LPDDR3 POP memory (2 channel 32bits @ 800MT/sec)
Flash storage	4 GB eMMC (v4.51 spec)
WiFi	Broadcom® 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) Onboard antenna or external antenna (SKU configurations)
Bluetooth	Bluetooth 4.0
POWER	
Input	3.3 to 4.5 V
Output	100 ma @3.3 V and 100 ma @ 1.8 V
Power	Standby (No radios): 13 mW Standby (Bluetooth 4.0): 21.5 mW (BTLE in Q4-14) Standby (Wi-Fi): 35 mW
FIRMWARE + SOFTWARE	
CPU OS	Yocto Linux® v1.6
Development environments	Arduino® IDE Eclipse supporting: C, C++, and Python Intel XDK supporting: Node.JS and HTML5
MCU OS	RTOS
Development environments	MCU SDK and IDE



Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com/design/literature.htm>.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

Intel, the Intel logo, Atom, Pentium, Quark, and Xeon are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2014 Intel Corporation. All rights reserved.

♻️ Please Recycle

331179-001



Appendix D

Lidar-Lite Datasheet



PulsedLight, Inc.

A NEW BENCHMARK IN OPTICAL SENSOR TECHNOLOGY



LIDAR-LITE SPECIFICATIONS

Dimensions

21 X 48.3 X 35.5 mm
PCB 44.5mm X 16.5mm

Performance

Range: 0-20m LED Emitter
Range: 0-60m Laser Emitter
Accuracy: +/- 0.025m
Power: 5vdc, <100ma
Acquisition Time: < 0.02 sec
Rep Rate: 1-100Hz

Configurations

- LED/PIN Diode, No Optics
- LED/PIN Diode, 12mm Optics
- Laser/PIN Diode 14mm Optics (Class I Laser Product)

Interface

- I2C
- PWM

US Patent: 8,125,620
Additional Patents Pending

Overview

PulsedLight targets the need for high performance, very compact optical distance measurement sensors for applications such as robotics and UAV's where a very small, low-power, high performance, reduced cost optical ranging sensor is desired.

Our single chip processing solution in combination with minimal supporting hardware enables a new class of optical distance measurement sensors.

PulsedLight's goal is to make our technology available in an easily configurable sensor module that can be used as the basic building block for sensor applications in robotics, UAV and Maker projects.

Technology

Our single board implementation uses as its standard emitter an 850nm LED. The design also supports the substitution of a variety of other optical sources such as VCSEL's (Vertical-Cavity Surface-Emitting Lasers) or edge emitting lasers. The laser version of LIDAR-Lite uses an edge emitting, 905nm, single stripe laser. This Laser Product is designated as Class I during all procedures of operation, however operating the sensor without it's optics or housing or making modifications to the housing can result in direct exposure to laser radiation and the risk of permanent eye damage.

The standard detector is based on a Si PIN diode, but optionally, could support a Si Avalanche Photo-Diode (APD) for greater sensitivity and range. Use of an APD would require external power and temperature compensation circuitry and provisions have been made to allow for access to the detector bias input circuit..

Technology innovations

- The use of a signature matching technique (known as signal correlation) that estimates time delay by electronically sliding a stored transmit reference over the received signal in order to find the best match.
- Operation of the infrared LED or laser in short bursts allowing a 100:1 advantage in peak output power over measurement systems using a continuous beam.
- A novel current driver technology with nanosecond signal transition times at high peak currents to produce high power transmit burst sequences.
- A signal processing approach implementable in a single programmable logic chip.

Other Innovations

While not implemented in LIDAR-Lite, other innovations to be released in future products include;

- Detector switching technology allowing multiple detectors to be processed by a single signal-processing channel. Enabling compact multichannel systems.
- Multiple digital processing cores implementable in a single programmable logic chip enable use of our technology in high resolution machine vision or scanning systems.



Signal/Power Interfaces	Specifications
Power	4.7 - 5.5V DC Nominal, Maximum 6V DC
Weight	PCB 4.5 grams, Module 16 grams with optics and housing
Size	PCB 44.5 X 16.5mm, Housing 21 X 48.3 X 35.5mm
Current Consumption	<100ma continuous operation, <2ma @ 1Hz (power off between acquisitions)
Max Operating Temp.	70° C
External Trigger	3.3V logic, high-low edge triggered
PWM Range Output	PWM Signal proportional to range, 1msec/meter, 10µsec step size
I2C Machine Interface	100Kb - Fixed, 0xC4 slave address. Internal register access & control
Supported I2C Commands	Single Distance Measurement, Velocity, Signal Strength
Mode Control	Busy status using I2C, External trigger input PWM Outputs

System Parameters	LED/Pin	LED/Pin wi Optics	Laser/Pin ⁽¹⁾ Class I Laser Product
Transmitter	850nm, 5mm Plastic LED 6° divergence	850nm, 5mm Plastic LED 6° divergence	905nm, 75µm, 1watt, 4mrad, 14mm optic
Receiver	5mm Plastic Si PIN 30° FOV	5mm Plastic Si PIN 10° FOV wi 12mm optics	Surface mount PIN, 3° FOV wi 14mm optics
Detector Gain	1X	1X	1X
Max Range @ 1Hz 30% Target	3 meters	10 meters	30 Meters
Max Range @ 1Hz 90% target	5 meters	20 meters	60 Meters
Accuracy	+/- 0.025 meter	+/- 0.025 meter	+/- 0.025 meter
Acquisition Time	<0.02 sec	<0.02 sec	<0.02 sec
Max Rep Rate	100Hz ⁽²⁾	100Hz ⁽²⁾	100Hz ⁽²⁾

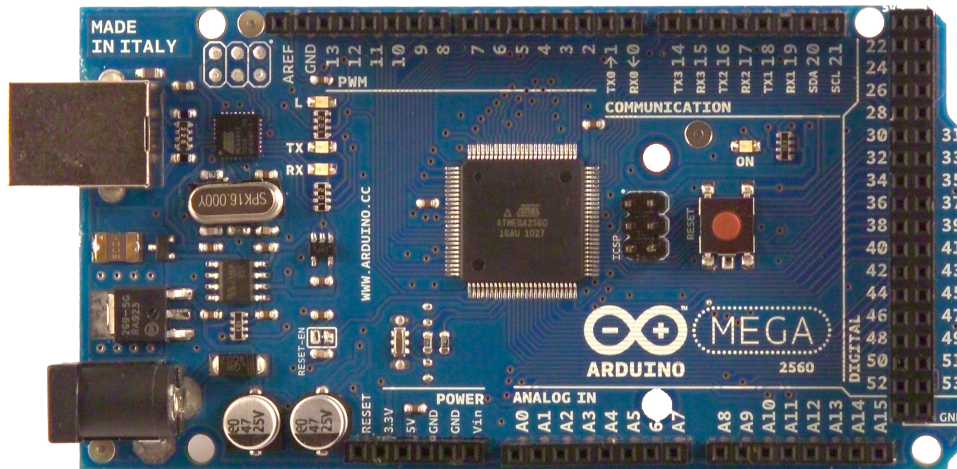
NOTES:

1. CLASS I LASER PRODUCT CLASSIFIED EN/IEC 60825-1 2007. Complies with US FDA performance standards for laser products except for deviations pursuant to Laser Notice No. 50, dated June 24, 2007. System contains no user serviceable components. Repair or service of the system is only to be handled by factory-trained technicians. No Service by the user is allowed.
2. Higher Rep Rates have an impact on maximum range. 1Hz to 10Hz there is no change, from 10Hz to 100Hz max range will decrease until it is approximately 50% at 100Hz. Rep Rate can be dynamically configured.
3. All Operating Specifications are Preliminary.

Appendix E

Arduino Mega Datasheet

Arduino MEGA 2560



Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 ([datasheet](#)). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

Technical Specifications	Page 2
How to use Arduino Programming Enviroment, Basic Tutorials	Page 6
Terms & Conditions	Page 7
Enviromental Policies half sqm of green via Impatto Zero®	Page 7



radiospares

RADIONICS



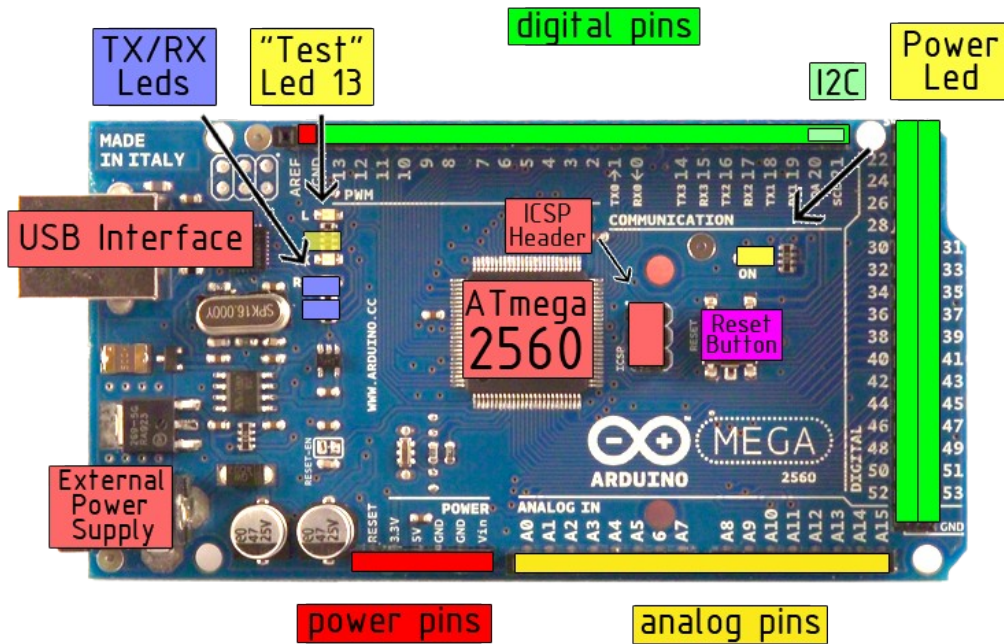
Technical Specification

EAGLE files: [arduino-mega2560-reference-design.zip](#) Schematic: [arduino-mega2560-schematic.pdf](#)

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



radiospares

RADIONICS



Appendix F

Pixhawk Datasheet



PX4FMU – Flight Management Unit

QUICK START – HARDWARE VERSION 1.6

Description

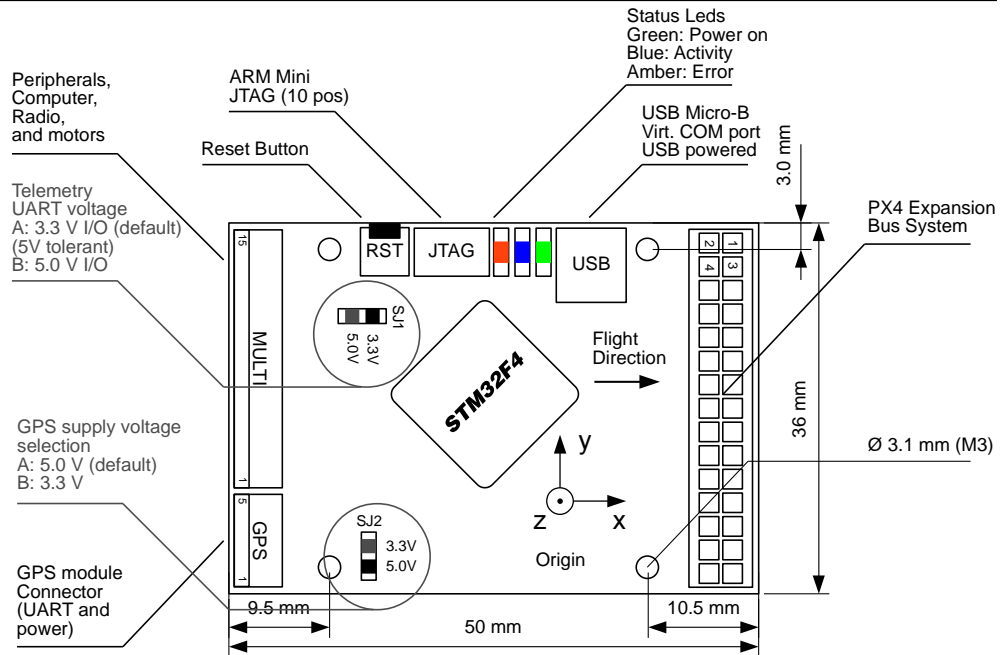
PX4FMU is an onboard management unit for micro air vehicles. It combines an autopilot and inertial measurement unit and enables the control of an aircraft using a single-board solution. Additional I/O can be easily connected via the 30-pin expansion bus.

<http://pixhawk.ethz.ch/px4/>

Features

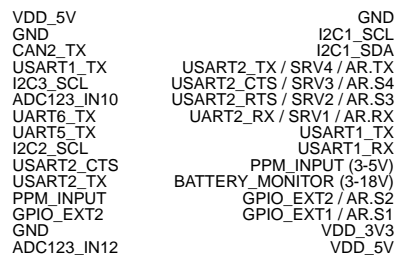
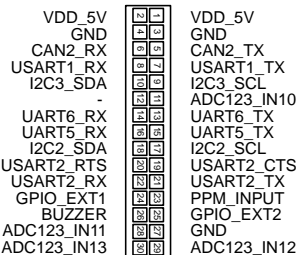
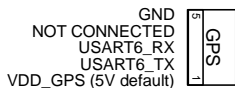
- 168 Mhz Cortex-M4 CPU (196 KB RAM, 1 MB Flash)
- 250 mW typical power consumption
- Reverse polarity protection on all power inputs
- 3D gyro, accelerometer and magnetometer, pressure sensors
- I2C, 3x UART, PPM, analog, GPS, 2x 5V GPIO, 4x PWM / Servo
- MicroSD card slot
- Expansion bus: CAN, 2x I2C, SPI, 4x analog, 2x UART, GPIOs
- USB Serial Port (Virtual COM Port / VCP) and bootloader
- 50 x 36 x 6 mm (1.38x1.97x0.24"), 8g, 30x30 mm mounting holes
- 4.5-6 V wide supply input range (incl. USB power)
- Selectable 3.3 V or 5 V IO for UART2 and GPS ports

Connectors, Jumpers and Dimensions



Pinout and absolute maximum Ratings

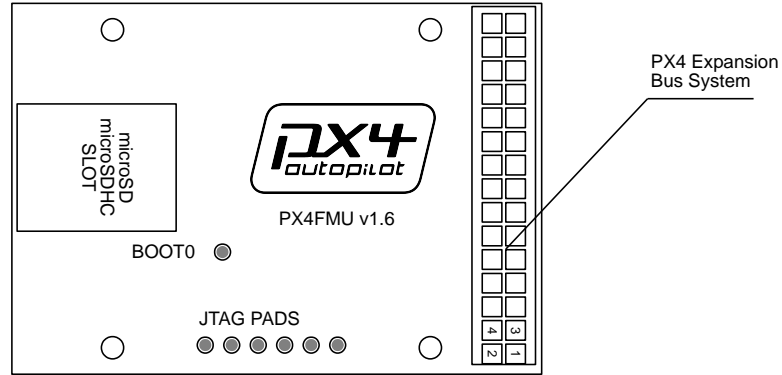
- Input: 4.3-6 V (VDD_5V), 20 mA onboard use, max. 800 mA for max peripheral load. Reverse-polarity protected.
- Output: 3.3 V (VDD_3V3), fuse-limited 500 mA EXT, 3.3 V, fuse-limited 200 mA GPS



Mates housing: Hirose DF13 "DF13-5S-1.25C", contacts: "DF13-2630SCF", AWG 26-30 Mates 2 mm header: 3M "951230-2520-AR-PR" Mates housing: Hirose DF13 "DF13-15S-1.25C", contacts: "DF13-2630SCF", AWG 26-30

Additional connectors (bottom side)

The footprints on the bottom side of the connector can be used by advanced users to interface additional boards or sensors.



Software Tools / Getting Started

Please follow the steps below to get started with PX4FMU.

- Download the GCS GUI (Windows / Linux / Mac) from <http://www.example.com>
- Install the application
- Connect PX4FMU with an USB-A to Micro USB-B cable to your computer (cellphone usb data cable type)
- Your operating system might display a message indicating that new hardware was found
- Start GCS from your application menu
- Go to Communication > Add new Link
- Leave the default settings, except for these values:
Baud rate: 115200 baud, data bits: 8 bits, stop bits: 1 bit, no parity, no hardware flow control
- GCS will display the heartbeat of MAV001. The displayed attitude will change if you move PX4FMU.

Upgrading Firmware / Developing Custom Code

After the steps in the getting started guide have been completed, follow these instructions to upgrade your firmware:

- Start GCS, select from the "Widget" menu the item "PX2 Firmware"
- In the PX4Firmware widget, click on "Check for Updates"
- Select the firmware revision to flash – usually the newest one at the top of the list, but the tool also allows to downgrade to older versions.

To develop custom code, please follow the developer instructions at: http://www.example.com/developers_guide

Open Hardware License

PX4FMU is an open hardware design, following the OSHW 1.1 definition licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) license. PX4FMU uses the BSD-licensed NuttX operating system as base for the PX4 software stack (<http://nuttx.sourceforge.net>).

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

<http://creativecommons.org/licenses/by-sa/3.0/>

Appendix G

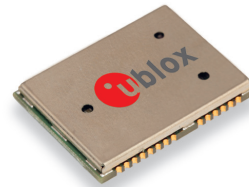
GPS Datasheet

LEA-6 series

u-blox 6 GPS, QZSS, GLONASS and Galileo modules

Highlights

- Multi-constellation variants:
 - GPS (LEA-6A/S)
 - GPS, Galileo ready (LEA-6H)
 - GPS/QZSS, GLONASS (LEA-6N)
- Optimized mode for low power and maximum sensitivity
- UART, USB and DDC (I²C compliant) interfaces
- Integrated antenna supervisor



LEA-6:
17.0 x 22.4 x 2.4 mm

Features

- u-blox 6 position engine:
 - Navigate down to -162 dBm and -148 dBm coldstart
 - Faster acquisition with AssistNow Autonomous
 - Configurable power management
 - Hybrid GPS/SBAS engine (WAAS, EGNOS, MSAS)
 - Anti-jamming technology
- Simple integration with u-blox wireless modules
- A-GPS: AssistNow Online and AssistNow Offline services, OMA SUPL compliant
- Backward compatible (hardware and firmware); easy migration from LEA-5 or LEA-4 families
- LCC package for reliable and cost effective manufacturing
- Compatible with u-blox GPS Solution for Android
- Based on GNSS chips qualified according to AEC-Q100
- Manufactured in ISO/TS 16949 certified production sites
- Qualified according to ISO 16750

Product description

LEA-6 modules bring the high performance u-blox 6 position engine to the industry standard LEA form factor. u-blox 6 has been designed for low power consumption and low costs, independent of which satellite constellation is used (e.g. GLONASS, Galileo). Intelligent power management is a breakthrough for low-power applications. The versatile, standalone LEA-6 receivers combine an extensive array of features with flexible connectivity options. Their ease of integration results in fast time-to-market for a wide range of automotive and industrial applications.

LEA-6 modules work with all available satellite positioning systems: LEA-6H is ready to support the European Galileo system via a simple firmware upgrade; LEA-6N combines full feature GPS performance with the QZSS regional satellite system. LEA-6N also targets the Russian market, featuring the lowest power GLONASS functionality in the industry and is designed for ERA-GLONASS.

All LEA-6 modules are manufactured in ISO/TS 16949 certified sites. Each module is tested and inspected during production. The modules are qualified according to ISO 16750 - Environmental conditions and electrical testing for electrical and electronic equipment for road vehicles.

Product selector

Model	Type	Supply	Interfaces	Features
	Standalone GPS Standalone GLONASS Standalone Galileo QZSS Timing & Raw Data Dead Reckoning	1.75 V - 2.0 V 2.7 V - 3.6 V	UART USB SPI DDC (I ² C compliant)	Programmable (Flash) FW update Oscillator RTC crystal Antenna supply and supervisor Configuration pins Timepulse External interrupt / Wakeup
LEA-6N	• • R •	•	• • •	• T O • 1 •
LEA-6H	• R R	•	• • •	• T O • 1 •
LEA-6S	•	•	• • •	T O • 1 1 •
LEA-6A	•	•	• • •	C O • 1 1 •

R = HW ready, firmware upgrade required.
O = Onboard RTC crystal for faster warm and hot starts.

C = Crystal / T = TCXO

Receiver performance data

Receiver type	50-channel u-blox 6 engine GPS/QZSS L1 C/A code GLONASS L1 FDMA Galileo L1 open service (with upgrade) SBAS: WAAS, EGNOS, MSAS		
Navigation update rate	up to 5 Hz (ROM version), 2 Hz (Flash)		
Accuracy ¹	GPS	GLONASS	
	LEA-6H/6S/6N/6A	LEA-6A	LEA-6N
Position	2.5 m CEP	4 m CEP	
SBAS	2.0 m CEP	n.a.	
Acquisition ¹	LEA-6H/6S/6N	LEA-6A	LEA-6N
	Cold starts:	26 s	27 s
Aided starts ² :	1 s	3 s	n.a.
Hot starts:	1 s	1 s	3 s
Sensitivity ³	LEA-6H/6S/6N	LEA-6A	LEA-6N
	Tracking:	-162 dBm	-162 dBm
Cold starts:	-148 dBm	-147 dBm	-138 dBm
Hot starts:	-157 dBm	-156 dBm	-153 dBm

¹ All SV @ -130 dBm

² Demonstrated with a good active antenna

³ Dependent on aiding data connection speed and latency

Electrical data

Power supply	2.7 V – 3.6 V		
Power consumption	GPS	GLONASS	
	LEA-6H/6S/6N	LEA-6A	LEA-6N
Continuous ⁴	121 mW	114 mW	121 mW
Power Save Mode ^{4,5}	36 mW	33 mW	n.a.
Backup power	1.4 V – 3.6 V, 22 µA		
Antenna power	External or internal VCC_RF		
Supported antennas	Active and passive		
Antenna supervision	Integrated short-circuit detection and antenna shutdown, open circuit detection with minimal external circuitry		

⁴ @ 3.0 V.

⁵ PSM @ 1 Hz.

Interfaces

Serial interfaces	1 UART	
	1 USB V2.0 full speed 12 Mbit/s	
	1 DDC (I ² C compliant)	
Digital I/O	Configurable timepulse	
	1 EXTINT input for Wakeup	
Serial and I/O	1 reset	
	Voltages	2.7 V – 3.6 V
Timepulse	Configurable	0.25 Hz to 1 kHz
Protocols	NMEA, UBX binary, RTCM	

Legal Notice

u-blox reserves all rights to this document and the information contained herein. Products, names, logos and designs described herein may in whole or in part be subject to intellectual property rights. Reproduction, use, modification or disclosure to third parties of this document or any part thereof without the express permission of u-blox is strictly prohibited.

The information contained herein is provided "as is". No warranty of any kind, either express or implied, is made in relation to the accuracy, reliability, fitness for a particular purpose or content of this document. This document may be revised by u-blox at any time. For most recent documents, please visit www.u-blox.com.

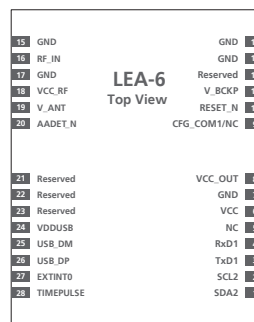
Copyright © 2012, u-blox AG

Specification applies to FW 7 and GLONASS FW 1.00
LEA-6N values: Objective Specification

Package

28 pin LCC (Leadless Chip Carrier): 17.0 x 22.4 x 2.4 mm, 2.1 g

Pinout



Environmental data, quality & reliability

Operating temp. -40° C to 85° C

Storage temp. -40° C to 85° C

RoHS compliant (lead-free)

Qualification according to ISO 16750

Manufactured in ISO/TS 16949 certified production sites

Support products

u-blox 6 Evaluation Kits:

EVK-6N:	u-blox 6 Evaluation Kit GPS/GLONASS/QZSS with TCXO, suitable for LEA-6N
EVK-6H:	u-blox 6 Evaluation Kit with TCXO, suitable for LEA-6H, LEA-6S
EVK-6P:	u-blox 6 Evaluation Kit with Crystal, suitable for LEA-6A

Ordering information

LEA-6N-0	u-blox 6 GPS/GLONASS/QZSS Module, TCXO, Flash, 17 x 22mm, 250 pcs/reel
LEA-6H-0	u-blox 6 GPS Module, TCXO, Flash, 17 x 22mm, 250 pcs/reel
LEA-6S-0	u-blox 6 GPS Module, TCXO, 17x22mm, 250 pcs/reel
LEA-6A-0	u-blox 6 GPS Module, 17x22mm, 250 pcs/reel

Available as samples and tape on reel (250 pieces)

Contact us

HQ Switzerland +41 44 722 7444 info@u-blox.com	China +86 10 68 133 545 info_cn@u-blox.com
EMEA +41 44 722 7444 info@u-blox.com	Japan +81 3 5775 3850 info_jp@u-blox.com
Americas +1 703 483 3180 info_us@u-blox.com	Korea +82 2 542 0861 info_kr@u-blox.com
APAC – Singapore +65 6734 3811 info_ap@u-blox.com	Taiwan +886 2 2657 1090 info_tw@u-blox.com

Appendix H

Piksi GPS Datasheet



Piksi Datasheet

Flexible, high-performance GPS receiver platform running open-source software

Features

- Centimeter-accurate relative positioning (Carrier phase RTK)
- 10 Hz position/velocity/time solutions
- Open-source software and board design
- Low power consumption - 500mW typical
- Small form factor - 53x53mm
- USB and dual UART connectivity
- External antenna input
- Full-rate raw sample pass-through over USB

Applications

- Autonomous Vehicle Guidance
- GPS/GNSS Research
- Surveying Systems
- Precision Agriculture
- Unmanned Aerial Vehicles
- Robotics
- Space Applications

Overview

Piksi™ is a low-cost, high-performance GPS receiver with Real Time Kinematics (RTK) functionality for centimeter-level relative positioning accuracy.

Its small form factor, fast position solution update rate and low power consumption make Piksi ideal for integration into autonomous vehicles and portable surveying equipment.

Piksi's open source firmware allows it to be easily customized to the particular demands of end users' applications, easing system integration and reducing host system overhead.

In addition, Piksi's use of the same open source GNSS libraries as Peregrine, Swift Navigation's GNSS post-processing software, make the combination of the two a powerful toolset for GNSS research, experimentation and prototyping at every level from raw samples to position solutions.

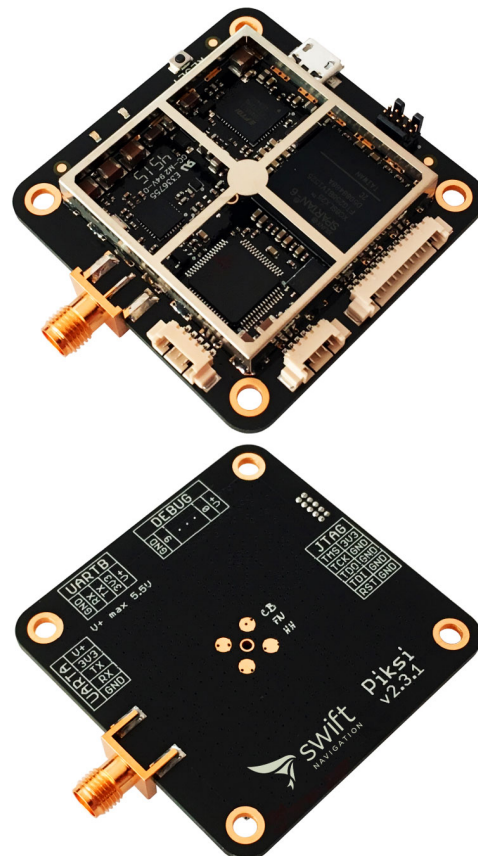


Figure 1: Piksi front and back view

With these tools, developers can quickly move from prototyping software on a desktop to running it standalone on the Piksi hardware.

A high-performance DSP on-board and our flexible Swift-NAP correlation accelerator provide Piksi with ample computing resources with which advanced receiver techniques, such as multipath mitigation, spoofing detection and carrier phase tracking can be implemented.

System Architecture

The Piksi receiver architecture consists of three main components. The RF front-end downconverts and digitizes the radio frequency signal from the antenna. The digitized signal is passed into the SwiftNAP which performs basic filtering and correlation operations on the signal stream. The SwiftNAP is controlled by a microcontroller which programs the correlation operations, collects the results and processes them all the way to position/velocity/time (PVT) solutions.

firmware. The SwiftNAP contains correlators specialized for satellite signal tracking and acquisition. The correlators are flexible and fully programmable via a high-speed SPI register interface and are used as simple building blocks for implementing tracking loops and acquisition algorithms on the microcontroller.

While the SwiftNAP HDL is not open-source at this time, the Piksi has no restrictions against loading one's own firmware onto the on-board Spartan-6 FPGA.

Front-end

The RF front-end consists of a Maxim MAX2769 integrated down-converter and 3-bit analog-to-digital converter operating at 16.368 MS/s. This front-end is capable of covering the L1 GPS signal bands.

Microcontroller

The on-board microcontroller is a STM32F4 with an ARM Cortex-M4 DSP core running at up to 168 MHz. This powerful processor performs all functions above the correlator level including tracking loop filters, acquisition management and navigation processing and is able to calculate PVT solutions at over 10 Hz in our default software configuration. All software running on the microcontroller is supplied open-source.

SwiftNAP

The SwiftNAP consists of a Xilinx Spartan-6 FPGA that comes pre-programmed with Swift Navigation's SwiftNAP

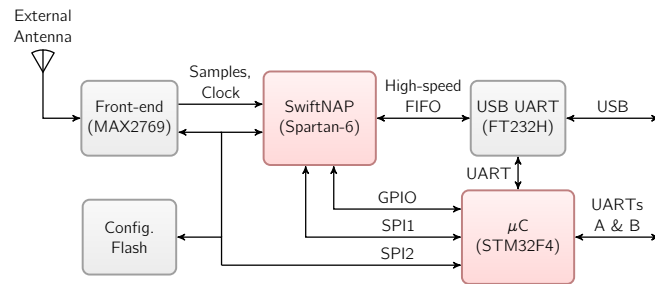


Figure 2: Piksi Block Diagram

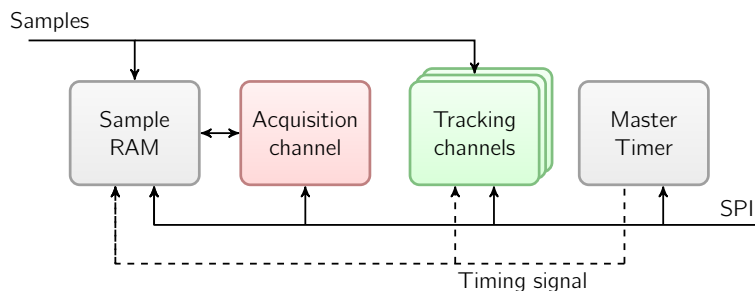


Figure 3: SwiftNAP Block Diagram

List of References

- Apostolopoulos, D.S. 2000. Technology and Field Demonstration of Robotic Search for Antarctic Meteorites. *The International Journal of Robotics Research*, 19(11):1015–1032.
- Bischoff, R. 1999. Advances in the development of the humanoid service robot HERMES. *Second International Conference on Field and Service Robotics.*, (August).
- Borenstein, J. & Koren, Y. 1989. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187.
- Borenstein, J. & Koren, Y. 1991a. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 7(4):535–539.
- Borenstein, J. & Koren, Y. 1991b. The vector field histogram—Fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288.
- Brock, O. & Khatib, O. 1999. High-speed navigation using the global dynamic window approach. *Proceedings 1999 IEEE International Conference on Robotics and Automation*, 1(May):341–346.
- E.Prassler, Scholz, J. & Fiorini, P. 1999. A Robotic Wheelchair Roaming in a Railway Station. *1999 IEEE International Conference on Field and Service Robotics*.
- Foessel, A., Foessel, A., Chheda, S., Chheda, S., Apostolopoulos, D. & Apostolopoulos, D. 1998. Short-Range Millimeter-Wave Radar Perception in a Polar Environment. *Carnegie Mellon University Research Showcase: Robotics Institute*.
- Fröhlich, C. & Mettenleiter, M. 2004. Terrestrial laser scanning: new perspectives in 3D surveying. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(8):7–13.
- Fröhlich, C., Mettenleiter, M., Härtl, F. & Langer, D. 2000. Imaging laser radar for 3-D modelling of real world environments. *International Conference on OPTO / IRS2 / MTT*, 20(4):273–282.
- Howie Choset, K.M.L., Seth Hutchinson, G.K., Wolfram Burgard, L.E.K. & Thrun, S. 2005. *Principles of Robot Motion: Theory Algorithms and Implementations*. Massachusetts Institute of Technology.

- Mathworks. 2016. Robot operating system (ros) support from robotics system toolbox.
Available at: <http://www.mathworks.com/hardware-support.html>
- Matsumoto, Y., Ikeda, K., Inaba, M. & Inoue, H. 1999. Visual navigation using omnidirectional view sequence. *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 317–322.
- Matsumoto, Y., Inaba, M. & Inoue, H. 2003. View-based navigation using an omniview sequence in a corridor environment. *Machine Vision and Applications*, 14(2):121–128.
- Navarro-Serment, L.E., Paredis, C.J. & Khosla, P.K. 1999. A beacon system for the localization of distributed robotic teams. *International Conference on Field and Service Robotics*, 6:2–7.
- ROS. 2016. About ros.
Available at: <http://www.ros.org/about-ros/>
- ROS-Wiki. 2016. Ros concepts.
Available at: <http://wiki.ros.org/ROS/Concepts>
- Santosh, D., Achar, S. & Jawahar, C.V. 2008. Autonomous image-based exploration for mobile robot navigation. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2717–2722.
- Simmons, R. 1996. The curvature-velocity method for local obstacle avoidance. *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, 4(April):3375–3382.
- Soto, A., Saptharishi, M., Trebi-ollennu, A., Dolan, J.M. & Khosla, P. 1999. Cyber-ATVs: Dynamic and Distributed Reconnaissance and Surveillance Using All-Terrain UGVs. *Carnegie Mellon University Research Showcase: Institute for Software Research*.
- Swinton & Campbell. 1920. *Power from the Sun*, volume 104.
- Thrun, S., Bennewitz, M., Burgard, W., Cremers, A.B., Dallaert, F., Fox, D., Haehnel, D., Lakemeyer, G., Rosenberg, C., Roy, N., Schulte, N., Schulte, J. & Steiner, W. 1999. Experiences with two Deployed Interactive Tour-Guide Robots. *International Conference on Field and Service Robotics*.
- Torresol. 2016. Gemasolar.
Available at: <http://www.torresolenergy.com/TORRESOL/gemasolar-plant/en>
- Ulrich, I. & Borenstein, J. 1998. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. *Proceedings of the 1998 IEEE International Conference on Robotics and Automation.*, pages 1572–1577.

- Ulrich, I. & Borenstein, J. 2000. VFH*: local obstacle avoidance with look-ahead verification. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, 3:2505–2511.
- Warren, C. 1989. Global path planning using artificial potential fields. 1:316–321.
- Ye, C.Y.C. & Borenstein, J. 2002. Characterization of a 2D laser scanner for mobile robot obstacle negotiation. *Proceedings 2002 IEEE International Conference on Robotics and Automation*, 3(May):2512–2518.
- Zohaib, M., Pasha, M., Riaz, R.a., Javaid, N., Ilahi, M. & Khan, R.D. 2013. Control Strategies for Mobile Robot With Obstacle Avoidance. pages 1027–1036.