

University of Southern Queensland

Faculty of Health, Engineering and Sciences

**AUTONOMOUS NAVIGATION OF
A PARTIALLY UNKNOWN
ENVIRONMENT**

A dissertation submitted by

Jason Pont

in fulfillment of the requirements of

ENG4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Honours) (Mechatronic)

Submitted October, 2019

Abstract

The aim of this project was to investigate navigation methods for supporting autonomous operations on Mars. To date there have been several robotic rover missions to Mars for the purpose of scientific exploration. These missions have relied heavily on human input for navigation due to the limited confidence in computer decision-making and the difficulty in localising an unknown environment with limited supporting infrastructure, such as satellite navigation. By increasing the confidence in the performance of an autonomous rover on Mars, this project will contribute to increasing the efficiency of future missions by reducing or removing humans from the control loop.

Due to the signal propagation delay between Earth and Mars, a certain level of autonomy is required to ensure a rover can continue operating while awaiting instructions from a human on Earth. However, due to the level of risk in relying solely on automation, there is still considerable human intervention. This can result in significant downtime when awaiting a decision by a human operator on Earth. While acceptable for scientific missions, greater autonomy will be required for routine Mars operations.

The project reviewed systems and sensors that have been used on previous robotic missions to Mars and other experiments on Earth. The most appropriate systems were assembled into a simulated test environment consisting of a small rover, an overhead camera that might be carried by a drone or balloon and wireless communications between the systems. A machine vision algorithm was developed to test the concept of an overhead camera mounted on a drone or balloon, while evaluating different path-planning algorithms for speed in navigating a previously unknown environment. An experimental system was built consisting of a rover, fixed overhead camera and communications between them. The machine vision algorithm was used to send instructions to the rover which could then follow a path through a test environment with different obstacle densities. Two different path-finding algorithms were tested with the system.

The key outcomes of the project were the construction and testing of the system. The rover could navigate, rotate towards and travel to a target location, after receiving instructions via serial radio communications. The rover could also detect obstacles using an ultrasonic sensor and send this information back to the machine vision algorithm. The algorithm would then update the path with the new information received on obstacle locations and the rover would then follow the new path to the target location. By successfully testing the concept, the project showed that this system could be used to support future scientific missions, resource gathering and preparation for human exploration of Mars.

University of Southern Queensland
University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111/ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Jason Pont

Student Number: XXXXXXXXXX

Acknowledgements

I gratefully acknowledge the assistance of my supervisor Craig Lobsey who was a fantastic help throughout the duration of this project. He was always available for consultation and was of immense assistance in technical matters such as the use of radio modems, programming in Python and machine vision using OpenCV.

On a personal note, I would also like to acknowledge the patience of Kerri-Lee, Mahala, Caitlin, Allira and Ivy, while the living room was turned into a Mars research and testing centre and while I was absorbed in trying to make a small rover travel two metres by itself.

Table of Contents

ABSTRACT	I
CERTIFICATION.....	III
ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS.....	V
LIST OF FIGURES.....	VII
1 INTRODUCTION	1
1.1 AIM OF RESEARCH	2
2 LITERATURE REVIEW	4
2.1 INTRODUCTION	4
2.1.1 <i>Historical use and design of Mars Rovers</i>	4
2.1.2 <i>Determining Location</i>	6
2.1.3 <i>Path-Finding Algorithms</i>	8
2.1.4 <i>Most Common Obstacle Avoidance Algorithms</i>	9
2.2 SENSORS FOR OBSTACLE DETECTION AND NAVIGATION	11
2.3 FUTURE DIRECTIONS	13
2.4 CONCLUSION.....	14
2.5 PROJECT AIMS.....	15
3 METHODOLOGY.....	17
3.1 ROVER.....	18
3.1.1 <i>Overview</i>	18
3.1.2 <i>Motion Control</i>	20
3.1.3 <i>PID Controller</i>	22
3.1.4 <i>Communications</i>	23
3.1.5 <i>Obstacle Detection</i>	24
3.1.6 <i>Rover Testing</i>	25
3.2 LOCALISATION	25
3.3 PATH-FINDING ALGORITHMS	30
3.4 SYSTEM.....	31

4	RESULTS	33
4.1	TESTS PERFORMED PRIOR TO MACHINE VISION	33
4.2	ESTABLISHING MACHINE VISION PARAMETERS.....	34
4.3	MOTION CONTROL REFINEMENT	35
5	DISCUSSION	43
5.1	ROVER CONTROL AND COMMUNICATIONS PROTOCOLS	43
5.1.1	<i>PID Controller</i>	44
5.1.2	<i>Rover Speed Measurement</i>	44
5.1.3	<i>Communications</i>	45
5.2	LOCALISATION USING MACHINE VISION	45
5.2.1	<i>Susceptibility to Light Conditions</i>	45
5.2.2	<i>Camera Resolution</i>	46
5.2.3	<i>Camera Support Platform Motion</i>	47
5.3	PATH-PLANNING AND CONTROL	47
5.4	OBSTACLE DETECTION AND AVOIDANCE	48
5.5	HARDWARE ISSUES	51
5.5.1	<i>Radio Modem</i>	51
5.5.2	<i>Gyroscope</i>	52
5.5.3	<i>Troubleshooting</i>	53
5.6	CONCLUSION.....	54
6	CONCLUSIONS	55
6.1	FURTHER WORK	57
7	REFERENCES	58
	APPENDIX A - PROJECT SPECIFICATION	62
	APPENDIX B - ARDUINO CODE	64
	APPENDIX C - MACHINE VISION CODE – PYTHON	76

List of Figures

FIGURE 1.1 – THE CONCEPTUAL SYSTEM TO BE USED AS A BASIS FOR THE RESEARCH PROJECT	2
FIGURE 3.1- SYSTEM SET UP – ROVER COMMUNICATES WITH LAPTOP TO RECEIVE DIRECTION INSTRUCTIONS AND TRANSMIT OBSTACLE INFORMATION FROM THE ENVIRONMENT. OVERHEAD CAMERA IS USED TO DETERMINE POSITION OF ROVER IN RELATION TO TARGET AND CALCULATED PATH.....	17
FIGURE 3.2 - THE EXPERIMENTAL ROVER SHOWING THE LOCATION OF THE MAIN SYSTEMS ON BOARD THE ROVER AND THE LOCATION OF THE COLOUR TARGETS.....	18
FIGURE 3.3 - ROVER CIRCUIT DIAGRAM LAYOUT. THE MPU6050 ACCELEROMETER/GYROSCOPE ULTRASONIC SENSOR MOUNTED ON SERVO MOTOR AND RADIO MODEM WERE ALL CONNECTED DIRECTLY TO THE MICROCONTROLLER. THE MOTORS ARE CONNECTED THROUGH THE MOTOR CONTROLLER TO PROVIDE THE HIGHER NECESSARY CURRENT.....	20
FIGURE 3.4 - WHEN THE REFERENCE ORIENTATION IS RELATIVE TO THE CURRENT ROVER ORIENTATION, THE ROTATION IS CALCULATED AS 130°, FROM 0° TO -140°. WHEN THE REFERENCE ORIENTATION IS FIXED TO THE INITIAL ORIENTATION, IT IS POSSIBLE THAT THE COMPLEMENTARY ANGLE WILL BE CALCULATED, AS IN THIS CASE, THE SAME ROTATION IS CALCULATED AS -60° TO 170° FOR A TOTAL OF 230° ROTATION.....	22
FIGURE 3.5 - ULTRASOUND SENSOR ROTATED 45° FROM THE CENTRE AXIS SHOWING SENSOR FIELD. THE SERVO ROTATES 15° PER ITERATION UNTIL A FULL SWEEP OF THE AREA IN FRONT OF THE ROVER IS COMPLETE.....	23
FIGURE 3.6 - UNPROCESSED IMAGE CAPTURED BY THE CAMERA. THE “TARGET”, “ROVER BACK” AND “ROVER FRONT” TARGET COLOURS WERE DELIBERATELY CHOSEN TO CONTRAST WITH THE BACKGROUND AND BE LARGE ENOUGH TO BE IDENTIFIED USING THE OVERHEAD CAMERA.	26
FIGURE 3.7 - THE ORIGINAL IMAGE IS CAPTURED (A) AND THE IMPORTANT COLOURS ARE IDENTIFIED AND SPLIT INTO SEPARATE IMAGES (B) WITH THE TARGET, ROVER FRONT AND ROVER BACK COLOURS IN EACH IMAGE. THE CENTROID OF EACH COLOUR IS CALCULATED FROM THESE IMAGES. THESE IMAGES ARE THEN COMBINED AND THE FINAL IMAGE (C) IS USED TO CALCULATE ROVER HEADING AND TARGET HEADING.....	27
FIGURE 3.8 - ANGLE CONVENTION USED BY THE MACHINE VISION ALGORITHM, BASED ON THE ROTATION CONVENTION OF THE GYROSCOPE. 0° WAS LOCATED AT THE TOP OF THE CAPTURED IMAGE.....	28
FIGURE 3.9 - THE HEADING ERROR IS CALCULATED FROM THE DIFFERENCE BETWEEN THE ROVER HEADING AND THE HEADING TO THE TARGET. THE TARGET HEADING IS CALCULATED USING THE FRONT AND BACK ROVER MARKERS. USING THE POSITIONS OF THESE MARKERS, THE CENTRE OF THE ROVER COULD BE CALCULATED. THE HEADING FROM THE CENTRE OF THE ROVER TO THE TARGET COULD THEN BE CALCULATED AND THE HEADING ERROR WAS THE DIFFERENCE BETWEEN THESE TWO DIRECTIONS.....	30
FIGURE 3.10 – THE SYSTEM FLOWCHART. INITIALISE ROVER: THE ROVER IS POWERED ON AND INPUT/OUTPUT PINS SET, GYROSCOPE INITIALISATION CONDUCTED TO REMOVE GYRO DRIFT. ONCE THE INITIALISATION IS COMPLETE, THE POSITION OF THE ROVER IS DETERMINED USING THE CAPTURE FROM THE OVERHEAD CAMERA. THE PATH-PLANNING ALGORITHM THEN CALCULATES THE PATH AND THE ROVER IS SENT THE REQUIRED INFORMATION TO TRAVEL TO THE NEXT WAYPOINT. IF THE ROVER ARRIVES AT THE WAYPOINT, THE PATH-PLANNING ALGORITHM WILL SEND THE ROVER THE DATA REQUIRED TO TRAVEL TO THE NEXT WAYPOINT. IN THE EVENT OF AN OBSTACLE BEING DETECTED, THE PATH WILL BE RECALCULATED AND THE ROVER WILL THEN TRAVEL TO THE NEXT WAYPOINT ON THE NEW PATH. ONCE THE FINAL WAYPOINT IS REACHED, THE SYSTEM STOPS.....	32

FIGURE 4.1 - THE RESULTS OF APPLYING MORPHOLOGICAL OPERATIONS TO THE CAPTURED IMAGE, IN THE ORIGINAL IMAGE, A, THE ORANGE PIXELS ARE IDENTIFIED AND ISOLATED AS SHOWN IN B. UNWANTED PIXELS ARE REMOVED USING MORPHOLOGICAL OPERATIONS TO PRODUCE THE FINAL IMAGE OF THE ORANGE TARGET, C 35

FIGURE 4.2 - ROVER ATTEMPTS TO ORIENT USING OVERHEAD IMAGES TO OBTAIN HEADING ERROR. 1. HEADING ERROR AT 2.17s IS -33.9°. 2. HEADING ERROR AT 2.73s IS 23.7°. 3. HEADING ERROR AT 3.30s IS -27.1°. 4. HEADING ERROR AT 3.67s IS 36.9° 36

FIGURE 4.3 - ROVER SETTLES AFTER A FEW SECONDS WHEN USING GYROSCOPE TO OBTAIN HEADING ERROR. 1. HEADING ERROR AT 14.40s IS 65.6°. 2. HEADING ERROR AT 14.70s IS 45.3°. 3. HEADING ERROR AT 15.00s IS -2.8°. 4. HEADING ERROR AT 16.47s IS -0.8° 37

FIGURE 4.4 - MODIFIED HEADING ERROR COMPARED WITH ORIGINAL HEADING ERROR. 38

FIGURE 4.5 - ROVER OVERSHOOTS TARGET HEADING WHEN ROTATING, AT 22.33s THE HEADING ERROR IS -65.5°, THEN AT 23.70s THE HEADING ERROR HAS OVERSHOT BY A MAXIMUM OF 64.5°. IN THIS SCENARIO A CODED OBSTACLE WAS USED BUT IN A REAL SCENARIO THE ROVER WOULD HAVE COLLIDED WITH THE OBSTACLE..... 39

FIGURE 4.6 - THE SYSTEM USING THE POTENTIAL FIELD ALGORITHM TO NAVIGATE THROUGH A COURSE OF SMALLER OBSTACLES. FOUR FRAMES ARE SHOWN AT TIMES 0.00s, 32.30s, 1:08.94s AND 2:04.36s 40

FIGURE 4.7 - THE INDICATED POSITION OF THE OBSTACLES MEASURED WERE OUTSIDE THE LIMITS OF THE ACTUAL OBSTACLE. 41

FIGURE 4.8 - THE FINAL TEST USING OBSTACLE DETECTION AND THE A* ALGORITHM TO NAVIGATE AN ENVIRONMENT WITH A REAL OBSTACLE. AT THE START (0.00s), THE PATH IS CALCULATED AS IF NO OBSTACLES ARE PRESENT. AT TIME 39.15s THE ROVER APPROACHES THE OBSTACLE WHILE PERIODICALLY CHECKING FOR OBSTACLES. AT TIME 42.00s THE OBSTACLE IS DETECTED AND A NEW PATH CALCULATED. THE ROVER THEN FOLLOWS THE NEW PATH (1:14.13s AND 1:23.38s) UNTIL THE TARGET LOCATION IS REACHED (2:36.29s) 42

FIGURE 5.1 - THE EFFECT OF ROUNDING THE POSITION OF COLOUR TARGETS RESULTS IN VARIABLE HEADING ERRORS EVEN THOUGH THE ROVER AND TARGET DO NOT MOVE. IN THE LEFT FRAME THE HEADING ERROR IS -3.8. IN THE NEXT FRAME, ON THE RIGHT, THE HEADING ERROR HAS CHANGED TO -1.3 EVEN THOUGH NEITHER THE ROVER NOR TARGET HAVE PHYSICALLY MOVED. 47

FIGURE 5.2 - THE EFFECT OF INCREASING THE ROBOT RADIUS USED IN THE PATH PLANNING ALGORITHM. THE ROBOT RADIUS DEFINES THE AREA EXCLUDED FROM PATH PLANNING CALCULATIONS AROUND EACH OBSTACLE. A LARGER ROBOT RADIUS WILL PROVIDE A GREATER MARGIN FOR ERROR BY THE ROVER FOLLOWING THE CALCULATED PATH BUT MAY INCREASE THE LENGTH OF THE PATH CALCULATED. 49

FIGURE 5.3 - POSSIBLE CIRCUIT ARRANGEMENT FOR THE RADIO MODEM USING A DARLINGTON PAIR TRANSISTOR ARRANGEMENT TO AMPLIFY THE POWER SOURCE FOR THE RADIO MODEM. 52

1 Introduction

The exploration of the surface of Mars has largely been conducted by autonomous rovers since the early part of the century. The signal delay between Earth and Mars of up to 24 minutes, makes some level of autonomy essential for any machine operating on Mars. The only rover currently operating on Mars is Curiosity, from the Mars Science Laboratory mission. To operate this rover, instructions are sent each morning and the rover is left to follow the instructions for the rest of the day. At the end of the day, information is received from Curiosity relating to its current position, so it is possible to plan the next day. However, this means that every second day is lost because it takes a day to plan the next stage of the mission (Grotzinger et al. 2012).

If the autonomy of the rover could be increased, so that every second day were not lost in planning, the operation would make better use of resources. While this may not be essential for a scientific mission, future operations on Mars such as resource gathering to support a future human crewed mission will require greater autonomy. Greater autonomy for future Mars missions is the main research goal for this project.

A key limitation of all Mars rovers to date is that they are limited by what they can see. A plan view of the location, showing areas the rover cannot see, would allow the rover to travel autonomously over greater distances. The idea of flight on Mars has been proposed by other authors, either in the form of a helicopter (Fantino et al. 2017) or a balloon (Kerzhanovich et al. 2004). A platform such as this, with a camera to observe the rover, would provide a view of the rover in its surroundings beyond what the rover itself could see. This concept is illustrated in Figure 1.1. The concept of an overhead view of the rover was the basis of this research project.

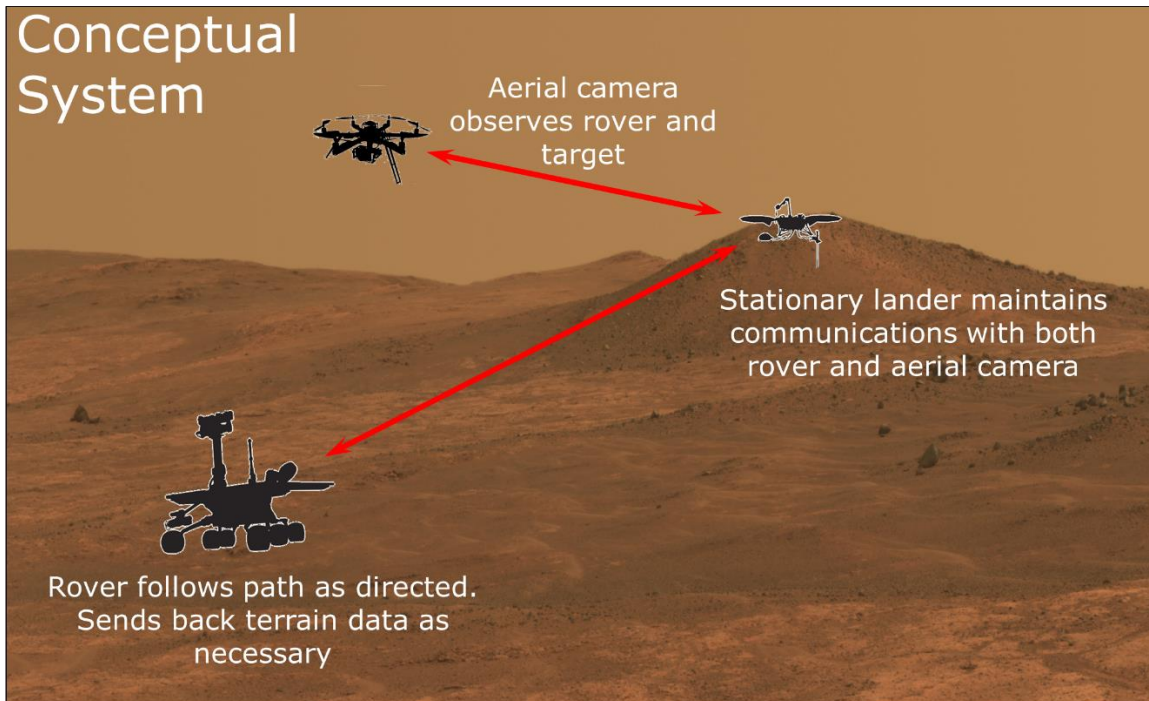


Figure 1.1 – The conceptual system to be used as a basis for the research project.

1.1 Aim of research

This research project will examine the use of path-finding algorithms and obstacle avoidance algorithms to complete a repetitive journey to and from a base location, a similar task expected for mining operations on Mars. To examine this scenario on Earth, an autonomous rover will be used to travel to a specific location with one or more objects between the start location and the target location. The rover will then return to the start location by the most direct route, using the knowledge of obstacles it has gained on the outward journey. For this task, the rover will be equipped with a gyroscope for determining direction and an ultrasonic obstacle detector which can be rotated to a defined orientation to determine if the path is clear. A microcontroller will operate two wheels on either side of the rover, which will control direction and speed.

With the sensors and actuators on board, the rover can find its way to the sample location and return, using dead reckoning only. Given the increasing inaccuracy of the dead reckoning with distance travelled, the method for localisation will be with a tracking camera above the rover. The tracking software will provide the location of the rover relative to the fixed points of the origin and target locations. Such a method for localisation has not yet been tested on Mars or any other terrestrial body. To date, it has not been possible to obtain an overhead image of the rover directly. Artificial satellites orbiting Mars are too high to observe a rover directly and the orbit of the satellites means

that even if they were able to observe the rover directly, the satellite would remain overhead for only a short period of time.

In order to investigate this idea, an experimental system will be set up in a controlled research environment. The system will use an autonomous rover with path finding and obstacle avoidance algorithms, with an overhead camera for tracking and localisation. The combination of an autonomous rover with overhead tracking will be tested for feasibility. By building this system it is hoped that an understanding of its potential for mining on Mars will be understood and the research can act as a building block for further innovation.

2 Literature Review

2.1 Introduction

The exploration of Mars has to date been a venture concerned with gathering scientific data, which has been solely carried out by robotic orbiters, landers and rovers. Examples include the Mariner and Viking missions in the 1970's and the Mars Science Laboratory (MSL) Curiosity rover, which is still operating at present. There are several man-made satellites currently orbiting the planet, which have a dual purpose of relaying signals to and from surface landers and rovers and completing scientific objectives such as surface mapping and taking atmospheric measurements. The goals of most of the lander missions in the past has been to examine the geology of the planet, look for possible signs of life and the presence of water.

Szocik et al. (2017) has outlined reasons why a manned mission to Mars instead of other nearby celestial bodies is worth considering. These reasons include its proximity to Earth and the presence of an atmosphere and water, making the logistics of a manned mission a feasible prospect. In comparison, Venus which is closer to Earth at perihelion than Mars, present challenges that are difficult to overcome for a manned mission. These challenges include extreme temperatures and a crushing atmosphere equivalent to 93 Earth atmospheres. However, Mars still presents significant challenges to human presence, such as the lack of a breathable atmosphere and high levels of cosmic radiation. Szocik et al. (2017) argues that the assistance of robots will reduce the exposure of humans to these hazards and is a more feasible option for future exploration of celestial bodies.

It is recognised that the objectives of rover missions to Mars and other celestial bodies will evolve and change moving into the future. This literature review aims to summarise the position of current technological advances while highlighting areas where further research is required or current research improved.

2.1.1 Historical use and design of Mars Rovers

The use of autonomy has benefitted robotic explorers on Mars by reducing the time required for decision making due to the signal delay to and from Earth (Francis et al. 2019). By allowing the robotic explorers to make simple decisions without confirmation from Earth, a signal delay of up to 24 minutes can be removed from the decision-making process for simple decisions. Autonomous rovers have been used on Mars for gathering scientific data and performing experiments. Rovers are an advance

in technology from earlier robotic landers such as Viking 1 and 2, which could only explore their immediate vicinity upon landing. The rover concept allows for closer exploration of features that would otherwise be unattainable by a fixed lander.

There have been four successful rover operations on Mars (Brown et al. 2013), with the earliest use of an autonomous rover on the planet being the Mars Pathfinder mission in 1997 (Matijevic & Shirley 1997). The rover, named Sojourner was a “micro-rover” designed to operate for seven ‘sols’ or Martian days. Sojourner operated within the vicinity of the lander, so that an exact path could be computed with the aid of cameras mounted on the lander. There was a once per sol opportunity to receive data from the rover and transmit instructions from earth back to the micro-rover. The rover would then perform the task allocated autonomously until the next transmission window.

The Sojourner rover required some autonomous hazard avoidance to operate successfully while not under the supervision of a human operator. The hazard avoidance system was operated every 10 seconds with the rover stopped. A laser stripe illuminated an area in front of the rover, which was detected by an on-board camera. Any gaps in the laser stripe were interpreted as holes or cliffs, while any high points were interpreted as obstacles that needed to be avoided. Additionally, tilt sensors determined whether the rover should stop and take another path to the ultimate destination, which was determined by human operators on a daily basis.

The second rover mission to Mars was the Mars Exploration Rovers (MER) mission. The mission deployed two rovers in different landing locations, Spirit and Opportunity (Cook 2005). Unlike the previous Mars Pathfinder Mission, the Mars Exploration Rovers had no fixed lander to observe progress and were controlled using only the aid of sensors and cameras onboard the rovers. These rovers used a more advanced method of obstacle avoidance than the Sojourner rover, which allowed greater autonomy on the surface while still requiring a significant input from human operators on Earth. Obstacle avoidance on the Mars Exploration Rovers was carried out with the use of several onboard cameras which could interpret stereo images to determine the distance and direction of an obstacle. The use of a human operator was still required but the autonomy allowed the rovers to operate while out of contact with Earth.

The most recent rover mission to Mars was the Mars Science Laboratory in 2011, using the Curiosity rover. The Curiosity rover operates with similar autonomous navigation capabilities as its predecessors in the Mars Exploration Rover mission. The autonomous operation of the Curiosity rover can extend to three days, where staff are not present over the weekend (Francis et al. 2019). The Curiosity rover

also has many autonomous operations available to its scientific instruments but contact with the rock or sample to be examined requires confirmation by a human operator.

A key function of rover autonomy is to plan a path to a given objective and avoid obstacles along the way. A path planning algorithm is required to determine this path, given information about the environment. Additional information about obstacles may be gathered by on-board sensors along the way, meaning the path-planning algorithm must recalculate the path once new information is gathered about the environment. The rover determines its position in the environment using localisation. On Earth this can be achieved using the global positioning system (GPS) however on Mars, where no GPS exists, this poses a further challenge.

2.1.2 Determining Location

Localisation is the ability of a robot to determine its location in an environment. In the early days of Mars exploration, the position of a robot was determined largely relative to estimates of landing site location. Data gathered by Mars orbiters has enabled later landing missions to establish location relative to a global grid, with increasing accuracy as technology improves (Tao et al. 2016). Once the position of a rover is known, this information can then be used as an input to the path planning algorithm. Rover localisation is also an important part of the scientific data gathered by existing rovers, as the geological data can be interpreted in different ways depending on its exact location.

The method used by the first rover mission, Mars Pathfinder, was to determine location based on correlation between images taken from the lander and the rover combined with dead-reckoning from the rover. This could also be extended to navigating based on images taken by the rover alone in the rover travelled “over the horizon” or out of the field of view of the lander (Shirley 1995).

This method of localisation provided an estimate of location that was acceptable for the goals of the Pathfinder mission. Later rover missions that travelled far beyond the original landing location and did not have an associated lander required a more accurate method of localisation. This was also vision based, used data from orbiting satellites to determine location on a global Mars grid (Tao et al. 2016). An extension of this method by Tao et al. (2016) was used to achieve even greater accuracy and reduce errors accumulated during the traverse. This method, however relies on prominent features that can be identified from orbit to determine location. These features are then compared with images obtained with the rover’s cameras to determine the location of the rover in relation to the feature identified from orbit. Only features that are large enough to be clearly imaged can be

used for this process, which is a restriction in largely featureless environments. At present, localisation methods will rely on image data to determine location, commonly known as visual odometry, however other methods have also been proposed to improve localisation.

Guan et al (2014) presents a solution to the position determining problem using star positions to accurately determine the position and attitude of a Mars Rover. This solution significantly reduces the accumulated error in position created when using inertial navigation or odometry. The solution allows for autonomous navigation without relying on a signal from Earth or a planetary orbiter (Guan et al. 2014). The method is expanded upon by Ning et al. (2016) and Lu et al. (2017) who use celestial navigation in combination with inertial navigation and visual navigation, to determine the path of a lunar rover. This method results in greater accuracy than dead-reckoning methods. However, the method can only be used at night when the rover is stationary. This may prove to be unfeasible for solar-powered rovers.

(Dabrowski & Banaszkiwicz 2008) describe a method to determine the location of a rover as long as it is part of a group of rovers within radio contact of each other. The method is similar to GPS in that it relies on an external clock signal broadcast to all rovers in the group to synchronise positions. The method was tested using a computer simulation and the results showed that in theory the method would be successful. This method of localisation would reduce reliance on satellites orbiting Mars, but would still require at least an initial position calculation from orbiter data.

There are also other options for tracking the rover directly which have not been explored in detail by previous authors. Satellites in a low orbit could view the rover directly but the height of the orbit would mean that the satellites would be moving quickly. This would require a large number of satellites to have a constant image of a ground rover. Small nanosatellites have been used in Low Earth Orbit to provide high resolution images, however a constant image provided by a constellation of such satellites has not yet been obtained.

Another possibility is the use of an aerial drone to provide a direct image above the operating area of the surface rover. An aerial drone for exploring the Martian surface independent of any ground presence has been described by Fantino et al. (2017). A third possibility is the use of a balloon floating in the Martian atmosphere to provide a platform for a camera looking down at the area of operation. Such a system, which could carry a payload of 2kg, has been proposed by Rand and Phillips (2004). A balloon similar to this was tested on Earth in the upper atmosphere as described by Kerzhanovich et al. (2004).

2.1.3 Path-Finding Algorithms

Path planning algorithms are used to define a path from a start point to an objective through a series of way-points. This can be as simple as a straight line from start to finish but more often than not requires way-points to avoid known obstacles. Path planning algorithms have been used by all Mars rovers to some extent, with increasing use as more autonomy is allowed.

When moving around on the surface of Mars, human control is limited by the signal delay from Earth, between four and 24 minutes depending on relative locations of Earth and Mars. The first rover to operate on Mars, Sojourner, moved around on the surface using a method that required high levels of daily human interaction even though there were some autonomous functions (Matijevic & Shirley 1997). The actual path was plotted daily with only minimal decision making required from Sojourner to avoid obstacles or other hazards. The later Spirit, Opportunity and Curiosity rovers had higher degrees of autonomy, but still required human interaction on a daily basis to determine the next waypoint within visual range (Erickson et al. 2007; Brown et al. 2013). The path of these rovers was plotted based on images received from the rover indicating its current position and a new location determined within the range of sight of the rover's cameras. The next step for autonomous pathfinding for a Mars rover is to autonomously travel to a point beyond visual range.

Miller et al. (1989) describe a path-planning algorithm using a Bayesian model of the sensor uncertainty. In this theoretical algorithm, local high-resolution maps sourced from the rover's cameras are used to plan a path towards a target that is beyond the range of the rover's cameras. The target is determined using a low-resolution map sourced from an orbiter, which lacks sufficient detail for a rover to safely traverse. The high-resolution map may only be 10 metres long but enables obstacle identification that is not possible using the low-resolution map from the orbiter. The algorithm was tested on two existing robots, a six-wheeled rover similar to future Mars rovers and a K2A industrial mobile robot. The results of the tests were not outlined in the paper but the rover described bears significant similarity to the Spirit and Opportunity rovers used as part of the Mars Exploration Rover mission 14 years later. (Miller et al. 1989)

The above methodology of Miller et al. (1989) was put into practice and built upon further by Post et al (2016). Post et al. (2016) evaluates an algorithm that uses a statistical model of infra-red sensors with Bayesian methods to map a small outdoor test area with several obstacles. The presence of an obstacle within the range of the sensors, triggers the obstacle avoidance program. This moves the rover in such a way that the path of the rover does not intersect the obstacle. The rover used in testing

had three infra-red sensors at the front, one pointing straight ahead and the others either side, which were positioned out 10° from the axis of advance. The algorithm was tested on a small, four wheeled rover in an outdoor test area and the results showed that the obstacles were all detected and mapped successfully. This method does not rely on a target point, instead using a mapping algorithm to direct the rover to any area within a given radius of the start point that hasn't been mapped. Once the entire area has been mapped the rover will stop (Post et al. 2016).

Tarokh (2008) describes a method of path planning using genetic algorithms for global path planning and heuristic algorithms for dealing with unexpected obstacles. The global path planning algorithm operates on a visual image of the local terrain obtained by the rover at the start of the traverse. If an unexpected obstacle is encountered during the traverse, the local obstacle avoidance heuristics are activated. The algorithm was tested using Matlab and Simulink and using a simulated Mars rover with an existing snapshot of a Martian landscape. Random obstacles were placed in the rover's path, after the global path planner had calculated the optimal path to simulate an unexpected obstacle. The simulation showed that the algorithm was effective at finding a path through a Martian landscape (Tarokh 2008).

Muñoz et al (2017) describe a path finding algorithm based on previous path finding algorithms such as A* and Theta*, which they called 3D accurate navigation algorithm or 3Dana. The path finding algorithm assumes that a global map of the area in question is available to base the path on. One of the main differences between this and other path-finding algorithms is that the height of a potential path is considered as well as the terrain cost of crossing the path. The authors compare the new algorithm with existing algorithms in terms of length, terrain cost, number of turns and computational time. Results showed that the 3Dana algorithm obtained a safer path to a target point than the A*, Field D* and Theta* algorithms, however this was at the cost of runtime. The study was theoretical, with the algorithms being run on several actual Martian digital terrain maps (DTMs) obtained from the Mars Reconnaissance Orbiter. (Muñoz et al. 2017) The Theta* algorithm is described by Daniel et al (2010) (Daniel et al. 2010)

2.1.4 Most Common Obstacle Avoidance Algorithms

For the purposes of this research, a distinction has been made between a path-finding algorithm and an obstacle avoidance algorithm. In the first case, the path-finding algorithm is based on a known environment, where obstacles have already been identified and the shortest path determined. In the second case, an obstacle avoidance algorithm will alter the path once an obstacle has been detected

to find a new shortest path to the target location. A path-finding algorithm can be considered to be more of a global approach to the problem while an obstacle avoidance algorithm will consider the local environment as determined by sensors onboard the rover. All the Mars rovers so far have used some form of obstacle avoidance.

The field D* obstacle avoidance and path-finding algorithm was created by Ferguson and Stentz (2005) who also tested the algorithm on several robotic platforms. Hayati et al (2007) describes the implementation of the Field D* algorithm on the Mars Exploration Rover mission while the rovers were on the surface of Mars. The existing obstacle avoidance algorithm, GESTALT had worked well for small isolated obstacles but tended to become stuck when it encountered wider obstacles or a large number of scattered obstacles. The algorithm was tested on Earth before it was transmitted to the rovers on Mars as a software update. Testing on Mars confirmed the successful operation of the algorithm, after a significant obstacle was overcome that would have caused the GESTALT algorithm to become stuck (Hayati et al. 2007).

Mutlu and Uyar (2012) describe an obstacle avoidance algorithm implemented on a tracked robot. The robot uses a laser measurement system on board, similar to LIDAR to define its environment. The environment surrounding the robot was redefined each time the laser measurement system was used. Like many other obstacle avoidance algorithms, the one described by Mutlu and Uyar uses a grid to describe the surrounding environment and has limited turning angles similar to the A* algorithm. The robot was built and tested successfully in a laboratory environment. (Mutlu & Uyar 2012)

Zeno et al. (2017) describe a prototype robot based on rodent neuro-physiology for navigation and sensing. The robot created named "ratbot" used the A* algorithm for navigation as it is similar to the way rodents and other animals navigate. The robot was tested on a simple indoor training area by instructing it to follow a simple course then find its way back to its start point. There were five ultrasonic sensors at the front for obstacle detection, a MEMS gyroscope and accelerometer for navigation and an Arduino microcontroller for controlling the ratbot's movement. The ratbot worked well in the small, controlled environment of the lab but the authors stated that more work was required to test the ratbot in a more complex environment (Zeno et al. 2017).

Given known obstacles based on sensor information, a robot can find its way to a target object using the "path of least resistance," as calculated using the potential field. This approach was described by Sasiadek and Green (1997), where the target point has an attractive force while any obstacles have a repelling force. Arutselvan and Vijayakumari (2015) propose a modification to this algorithm which

includes a solution to the “local minima” problem (where a robot gets stuck). A “virtual force ” will pull the robot towards free space so it can have another attempt at negotiating the obstacle. (Arutselvan & Vijayakumari 2015). While the algorithm was successful, the authors noted significant drawbacks such as the random motion of the robot and the fact that the robot does not take the shortest path to the target destination. There is potential for further work such as altering the algorithm in such a way that it finds the shortest path to the destination and finding a better way to avoid local minima, as the virtual work method was not always successful

2.2 Sensors for Obstacle Detection and Navigation

Whichever obstacle avoidance or path-finding algorithm is chosen, it can only perform successfully if an accurate description of the surrounding environment is obtained and the rover can be localised within this environment. Several types of sensors have been used to identify obstacles in the immediate vicinity of the rover as well as the movement of the rover itself.

One of the earliest and simplest sensors used to determine the position of the rover was using wheel encoders to determine the distance travelled based on the number of rotations of a wheel. The Mars Pathfinder mission used this method on the Sojourner rover to determine the distance travelled (Matijevic & Shirley 1997). (Pedraza et al. 1998) also used wheel encoders to determine distance travelled, although not in the context of a Mars mission as the data was correlated with satellite navigation information. More recently, Barfoot et al. (2011) describe the testing of a prototype rover in Utah that could be used to drill in exploration for ice at the Martian poles. Wheel odometry was used in this prototype also, to determine distance travelled. While many rovers and rover concepts use wheel odometry to estimate distance travelled, this can be prone to error due to wheel slippage. It also doesn't give any indication of the direction of travel, which must be provided by another sensor such as a gyroscope. Bertrand and Winnendael (2000) also describe a different method of overcoming the direction problem that involves the lander craft directing a laser at a target determined by human operators. Wheel odometry was then used to determine the distance travelled along a path directly towards the laser reflection. In this way, a point to point traverse could be accurately performed but only within site of the lander craft.

The direction of travel can be calculated using a gyroscope, as was the case for the Mars Pathfinder mission (Matijevic & Shirley 1997). By combining distance and direction with an initial start point, the final position of the rover can be calculated using dead reckoning. This method has an accumulated error, however, so a periodic update of position is required, such as the example described by Fox et

al. (1998). In the case of the Mars Pathfinder mission, this could be achieved while the rover, Sojourner, was in view of the lander craft. Once out of the field of view of a known landmark, however, the problem of localisation becomes more complex.

Stereo ego-motion, more commonly known as visual odometry, uses two cameras to determine the distance of the rover from a particular object. The difference in the images taken by each of the cameras is compared automatically and the distance from the object can be calculated using the known distance between the cameras and the heading of the cameras (Miller et al. 1989). There is still an error associated with this method of localisation, which has been reduced by the addition of data from other sensors, such as a gyroscope, compass or sun sensor, and wheel encoder (Olson et al. 2003). Many authors (Kubota et al. 2010; Neveu et al. 2010; Tao et al. 2016) have worked on increasing the accuracy of visual odometry by combining the data obtained from stereo cameras with that obtained from other sensors. Using cameras as sensors for navigation has almost become industry standard with many prototypes on Earth and all rovers on Mars being equipped with stereo cameras for some level of navigation assistance (Cook 2005; Hayati et al. 2007; Brown et al. 2013; Balme et al. 2019).

Stereo cameras can also be used for identifying obstacles in the rover's path in the same manner that they are used for visual odometry, with the difference being that in visual odometry the target landscape feature is usually much further away (Spiteri et al. 2017). Hazard avoidance was performed using stereo cameras on all three Mars rover missions (Matijevic & Shirley 1997; Hayati et al. 2007; Brown et al. 2013). Another method of identifying obstacles in the path of the rover are LiDAR (Barfoot et al. 2011) where an entire area can be scanned for obstacles prior to planning a path through it. A laser strip was used to detect obstacles on the Mars Pathfinder mission (Matijevic & Shirley 1997), where gaps in the reflected laser strip were interpreted as holes and peaks were interpreted as obstacles. The most basic level of obstacle detection uses contact switches (Matijevic & Shirley 1997; Bertrand & Winnendael 2000), although the obstacle would not be avoided if these were closed.

Ultrasonic sensors have also been used for obstacle detection in various applications. Elrayes et al. (2019) describe the building of a small rover used to detect foreign object debris (FOD) on an airport runway. This rover used a combination of LiDAR for detection of obstacles on the runway and ultrasonic sensors for detection of obstacles in the path of the rover. Mishra et al. (2016) describe an array of ultrasound sensors used to map an unknown environment. There were 24 sensors arranged in a circular array. The sensors were fired using a specific sequence to avoid crosstalk between sensors.

Gonzalez et al. (2019) use an experimental approach to test different machine learning algorithms on detecting wheel slip on a test lunar rover. The rover was tested using two different wheel types, two different surfaces and 11 different machine learning algorithms. Inertial measurement units (IMUs) were used to determine the amount of wheel slip and the best machine learning algorithm was found after a number of tests.

2.3 Future Directions

Currently space exploration using rovers is largely human controlled and operated, which means there is considerable potential for a more autonomous approach. The high cost of space exploration missions and the risks of failure of the autonomous systems, has resulted in a slow adoption of autonomy in space exploration (Post et al. 2016). However, as mission goals become more ambitious, an increased level of autonomy will be required to carry out the mission goals in the allowable timeframe (Pilles et al. 2018).

Multiple authors have speculated on more advanced robotic missions to Mars. O'Neil and Cazaux (2000) describe a concept study into a sample return mission, where rock and soil samples from Mars would be returned to Earth for analysis. This would allow for greater analysis due to the resources of terrestrial laboratories rather than the limitations of in-situ labs and processing equipment. Such a mission would require the search for an appropriate sampling location, along the lines of previous exploration missions. The second stage would be the autonomous return to the original location with the sample, for return to Earth.

An essential element for a manned mission to Mars would be the adequate supply of water for human consumption and hygiene, plants and other uses (Ralphs et al. 2015). Shishko et al. (2017) have briefly described the technology requirements of mining water or ice on Mars using an autonomous rover and economic analysis indicates that this would be profitable for a future Mars colony. The study assumes a mining operation on Mars would use traditional "truck and shovel" methods for mining, like what is used on Earth.

A different approach to water extraction on Mars is described by Ralphs et al. (2015). The regolith or loose unconsolidated solid material on the surface, contains water in the form of hydrated minerals. An autonomous rover could be used to scoop up this material and heat it to a temperature where the water would be liberated from the material. As the processing of raw materials would be performed on-board the rover, this could take place during the return to a central collection facility. Such a

process would be extremely slow, so would likely be established well before a manned mission to Mars.

Research by Kading and Straub (2015) also suggests another use for autonomous mining rovers and machines; where in-situ materials on planets are mined and 3D printing operations are used to build a base structure prior to human arrival. The source of raw material for this process would be basalt rocks collected from the immediate vicinity and taken back to a central hopper. In this scenario, the base structure would be completed before the arrival of humans so would require a largely autonomous process.

The common factor in all these mining scenarios, is the autonomous operation of a rover to collect raw materials and bring them to a central location. Mining applications such as these, would require a different type of autonomous rover operation than the normal scientific exploration that has been used in the past. There would be repeated journeys to a collection location some distance from the start point followed by a return to an initial location for deposit. It is this type of scenario that is addressed by this research project.

2.4 Conclusion

Throughout the history of Mars exploration, the use of rovers has required varying levels of autonomy to reduce the idle time spent waiting for human input and the signal delay between Earth and Mars. Originally, the Mars Pathfinder mission used a rover that did not stray far beyond the line of sight of the lander and required a specific set of instructions from Earth before an action was carried out. The degree of autonomy given to robotic explorers has increased over time, with the latest rover Curiosity, performing pre-programmed tasks unsupervised overnight and at weekends to reduce human staffing requirements.

As research into rover autonomy has continued, the focus has been on localisation and path-finding abilities for exploration rovers. Future missions are expected to focus on tasks such as sampling missions within the next decade and in the longer term, mining for water or minerals in support of a future manned mission to Mars. To date, there has been some research into how a sampling mission could be achieved but minimal research into a more complex mission such as mining on Mars. A summary of the sampling mission research states that this could be achieved with some human intervention but most authors agree that a mining mission would be largely autonomous. At present there has been little research into how this would be performed.

2.5 Project Aims

This research project will examine the use of path-finding algorithms and obstacle avoidance techniques to complete a journey to a target location to simulate a rover operating autonomously on Mars. The research summarised by this review shows that any future mining mission on planets such as Mars, would need to operate autonomously to make many trips to and from a central location feasible.

Another vital aspect to achieving the aim of this research project is addressing the localisation problem. A review of the literature summarises how this issue has been addressed previously such as using visual odometry combined with satellite images of the Martian surface. Other methods involved using star shots at night or images from a lander craft. These methods would be impractical in a mining scenario, where more frequent updates of position would be required.

This project aims to address the localisation issue by having a low altitude, high resolution image tracking the rover in relation to the origin and target locations. Such an image could be provided by low-orbit satellites, aerial drones or balloons. This information could be used to provide an updated position to the autonomous rover to increase localisation accuracy beyond that achieved by dead reckoning alone for future missions.

In previous work of this nature there have been two distinct approaches taken. One is to use simulation to examine the effects of using path-finding algorithms. The other approach is to build a prototype and test it in a controlled situation. For this project, the aim is to test a system rather than the performance of a computer algorithm. In order to simulate the system accurately, the exact operating parameters would need to be known, for example force applied by the motors, mass of the rover, how data is handled between the two systems. In order to measure these properties, it is often necessary to build a prototype first and perform tests.

For this project the aim was achieved by building a prototype system consisting of an overhead camera, machine vision software, a test rover and communications between the rover and machine vision system. When considering whether to build the physical system or simulate using a model, the complexity of the system with many different parts from different manufacturers had to be taken into account. If the system were to be simulated, the system would need to be modelled using data provided by manufacturers based on testing under controlled conditions. When the system is tested under different conditions, errors may be introduced in the modelled assumptions. When many

different components are acting together, these errors can be compounded so that the modelled result could be very different from the actual result. By building this system its feasibility and performance could be tested with the confidence of knowing the results were accurate, rather than using a simulation with modelled assumptions which may be incorrect.

3 Methodology

To investigate and test a system that could be used for a sample return mission or mining operation on Mars, a scaled down model that simulates an unknown, obstacle-filled environment was built and tested in a terrestrial setting. The system was designed to use an overhead camera and an autonomous rover that communicates with a laptop. The laptop will compute the optimal path and any course corrections required to get the rover through an obstacle course via the testing of two path finding algorithms. Figure 3.1 shows a schematic diagram of the system setup.

The test area was indoors and tests were conducted at roughly the same time of day every day to ensure similar lighting conditions for each test run. To ensure the rover starting position was always within the test area, the test area under observation by the camera was marked with tape. This also ensured that the exact same area was used for each test. The camera position was fixed to a marked position on the ceiling at the start of the trials so that if the camera was ever removed it could be replaced at the exact same spot looking at the same area.

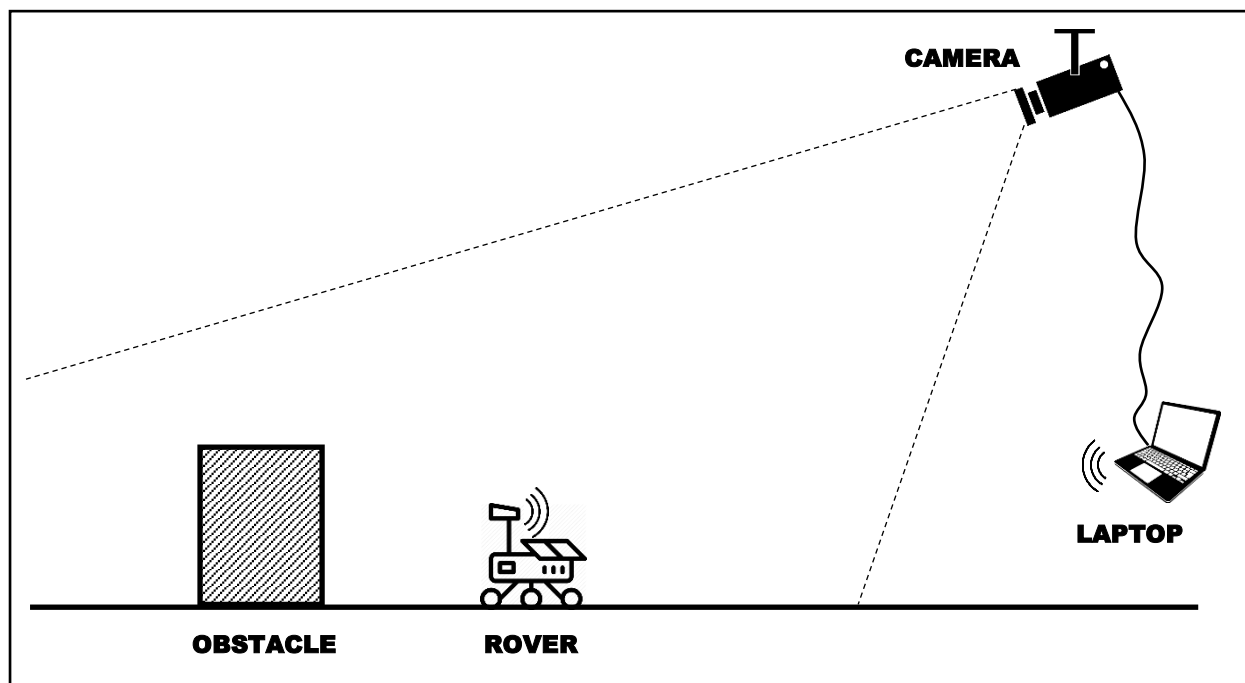


Figure 3.1- System set up – rover communicates with laptop to receive direction instructions and transmit obstacle information from the environment. Overhead camera is used to determine position of rover in relation to target and calculated path.

The system consisted of two projects that could be undertaken independently. The autonomous rover used a microcontroller to operate two motors which controlled the heading and speed of the rover. The localisation and path-planning project used a video feed provided by the overhead camera to

observe the rover and provide instructions via two-way communications between the rover and the laptop.

3.1 Rover

3.1.1 Overview

The rover as shown in Figure 3.2 consisted of an off-the-shelf kit that was modified to meet the requirements of the project. The rover was made up of the following systems:

- Motion control
- Communications
- Obstacle detection

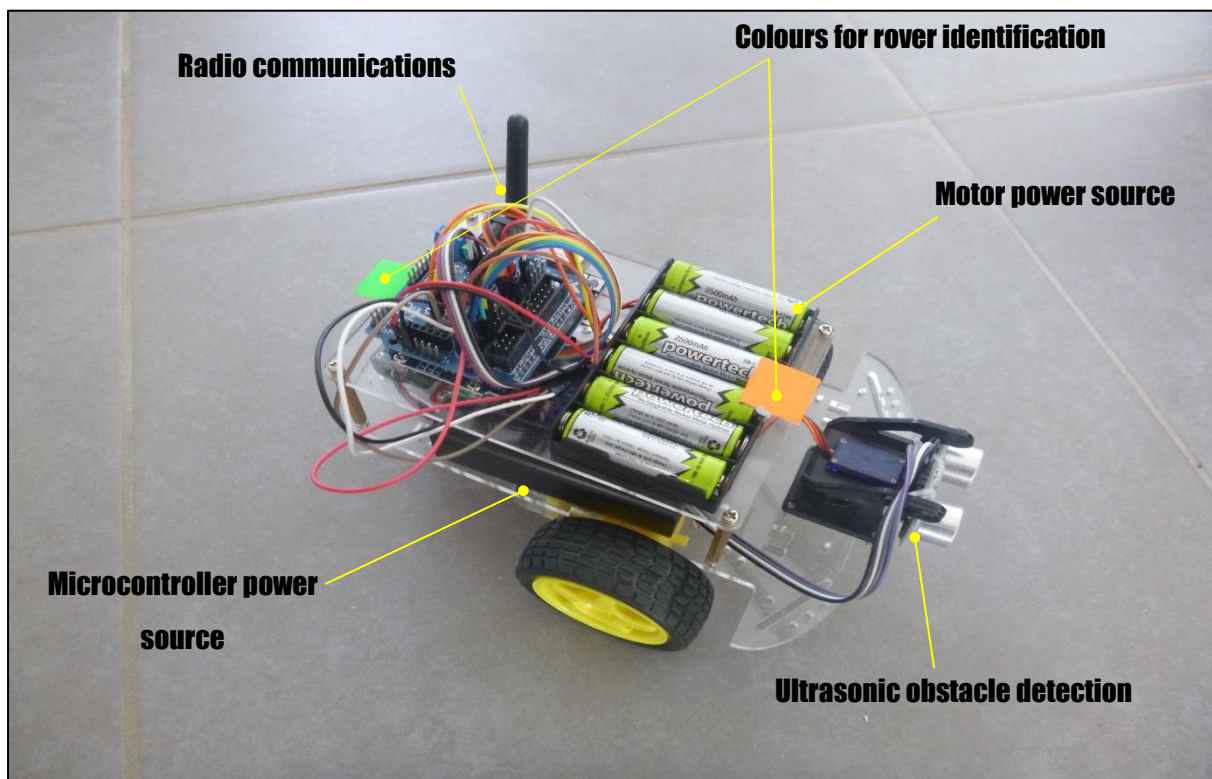


Figure 3.2 - The experimental rover showing the location of the main systems on board the rover and the location of the colour targets

These systems were controlled by an Arduino Uno microcontroller with a separate motor controller to provide the larger current required by the motors. The motors were attached to two wheels either side of the rover with a free spinning castor at the back of the rover added for balance. This allowed for two basic movements, forward and rotate. There was potential to add to this, for example move

on an arc or reverse, however to simplify the problem, the movements were limited to these two basic operations.

Power to the motors was provided by six 1.2V rechargeable AA batteries in series for a 7.2V supply. The microcontroller and attached sensors were powered by a 5V USB battery. It was found that the same power source could not be used for both the motors and the microcontroller as there was too much current drawn by both the motors and the radio modem to reliably operate both. A sensor shield provided additional 5V power pins for any sensors attached to the rover. Basic code was used to test the operation of the rover for moving forward, obstacle detection and rotating left and right. The circuit diagram of the designed rover is shown in Figure 3.3.

For basic obstacle detection, an ultrasonic sensor at the front of the device was attached to a servo motor, so that it could be rotated through an arc of 120° as required. The HC-SR04 sensor operates by sending an ultrasonic pulse at 40kHz and measuring the time taken for a reflected pulse from a solid object. From this, the distance to the object can be calculated based on the speed of sound. Unlike light-based methods, this sensor has a measuring angle of 15° rather than a narrow beam. This means that any obstacle detected could be on an azimuth $\pm 15^\circ$ from the azimuth of the ultrasonic sensor. This was taken into account when considering the location of any obstacle detected by the ultrasonic sensor.

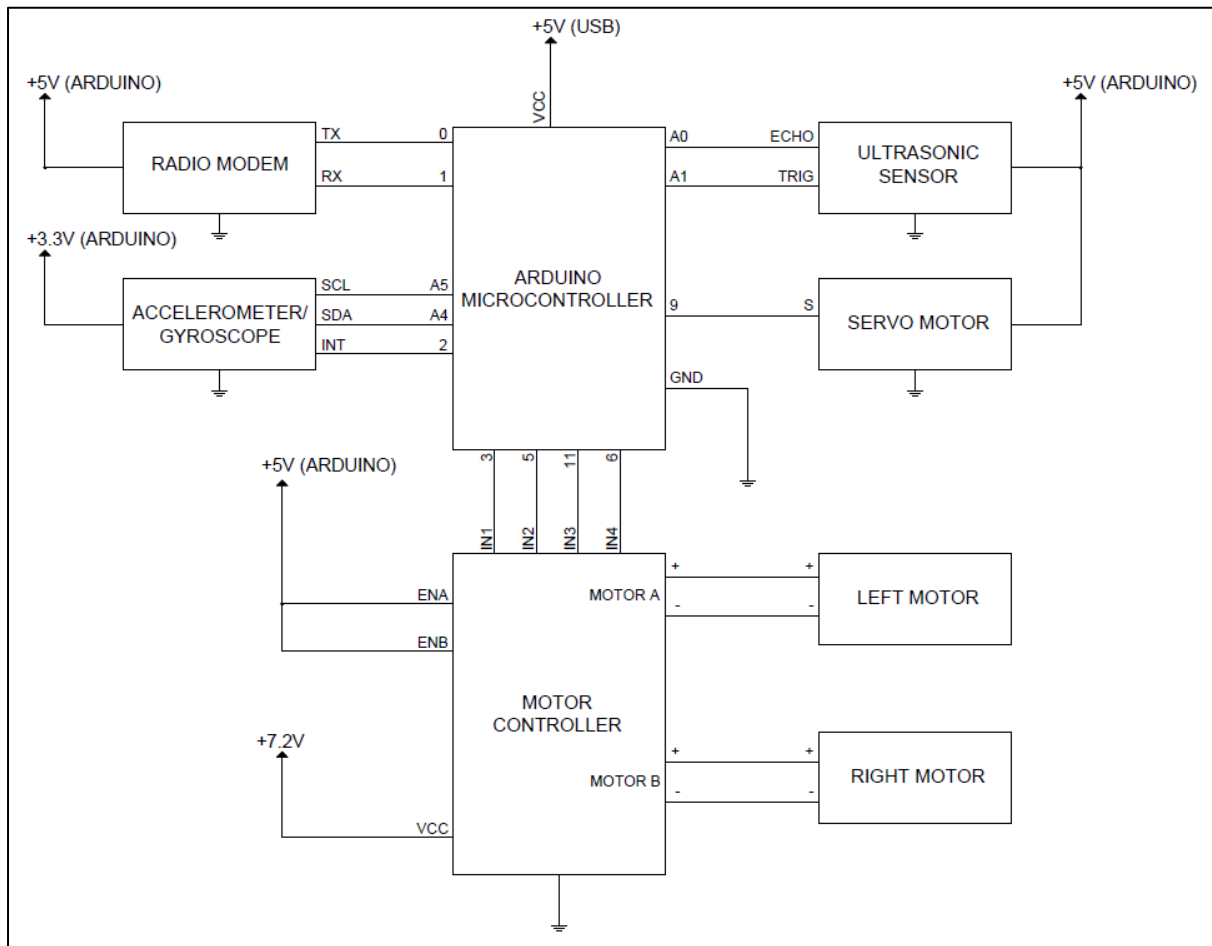


Figure 3.3 - Rover circuit diagram layout. The MPU6050 accelerometer/gyroscope ultrasonic sensor mounted on servo motor and radio modem were all connected directly to the microcontroller. The motors are connected through the motor controller to provide the higher necessary current.

3.1.2 Motion Control

A gyroscope for measuring the orientation of the rover relative to a reference angle was installed at approximately the centre of the rover. An option for reference angle was to use an integrated magnetometer(compass) for reference angle but because Mars does not have a magnetic field and therefore magnetic poles, this was not considered appropriate. Instead, for the initial stages of the project, the reference angle was to be the starting orientation of the rover. For the gyroscope, an MPU6050 was used, which is a commonly used Micro-Electro-Mechanical System (MEMS) accelerometer/gyroscope.

The MPU6050 measures angular velocity(ω) rather than actual orientation(θ). In order to obtain the orientation of the device some processing was required. To obtain orientation from angular velocity, the integral over time(t) is required:

$$\theta(t) = \int \omega dt$$

In the context of a digital controller, the time difference is the loop time as measured by the internal controller timer. However, the MPU6050 includes a Digital Motion Processor (DMP) as part of the package. The DMP stores angular velocity and time, while constantly calculating the orientation of the device. This orientation is stored in the DMP memory, which can trigger an interrupt every time new data is available. By using this feature of the MPU6050, it was possible to determine orientation directly without using the timing of the microcontroller itself. Once an interrupt was triggered, a flag was set within the code so that during the loop, the orientation of the MPU6050 could be retrieved by calling the data retrieval routine which would reset the flag.

When the MPU6050 was powered on, the initial angular velocity drifted for about 20 seconds until it settled to the correct value. When the angular velocity settled, the heading returned by the DMP was unrelated to the heading of the rover. To give the rover a fixed heading to relate to, the initial heading of the rover was set to 0°. Target headings would then be in the range -180° to 180°, making the direction of rotation simple to determine. Once the new heading was attained, this new heading was set to 0°. Using this method, the direction of rotation was always simple to determine, compared with a non-zero initial condition. For example, if the original heading was -60° and the target was 170°, the rotation calculation would be 230° clockwise and not 130° anticlockwise. This point is illustrated in Figure 3.4.

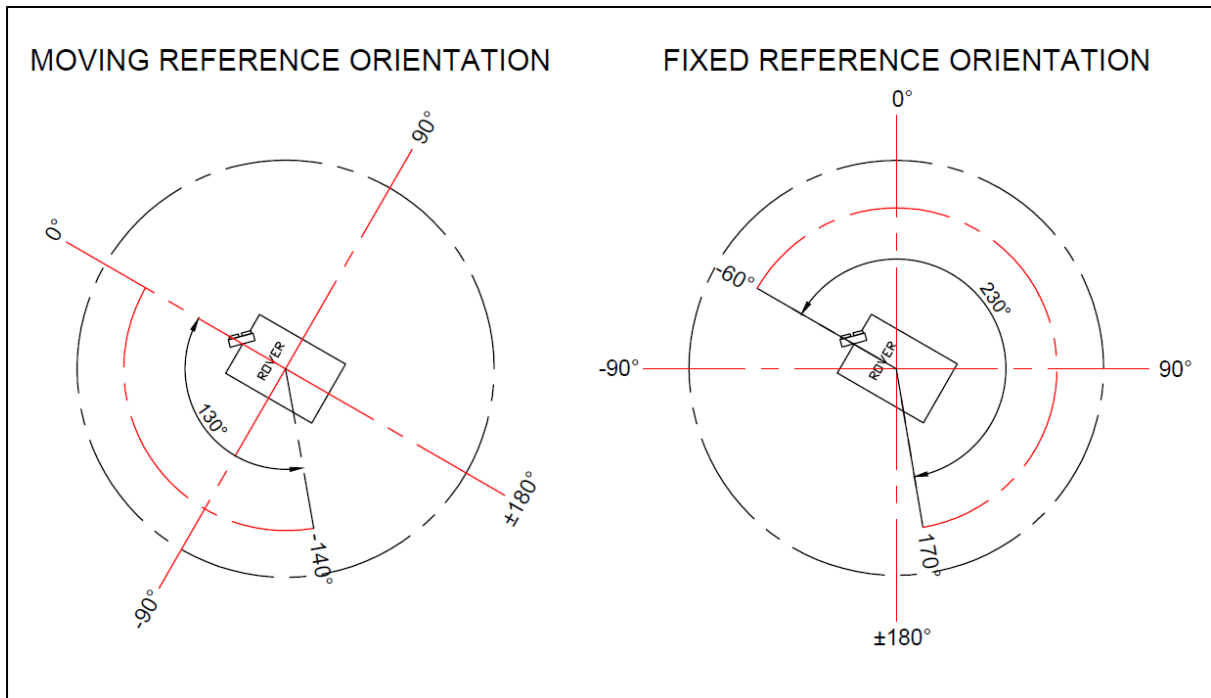


Figure 3.4 - When the reference orientation is relative to the current rover orientation, the rotation is calculated as 130°, from 0° to -140°. When the reference orientation is fixed to the initial orientation, it is possible that the complementary angle will be calculated, as in this case, the same rotation is calculated as -60° to 170° for a total of 230° rotation.

3.1.3 PID Controller

Once the orientation of the MPU6050 could be retrieved using the microcontroller, a PID control program could be written to control the orientation of the rover. The PID controller sent control signals to the motors based on the reading from the MPU6050. In order to tune the PID controller, a short test code was written to rotate the rover to a target angle using the PID controller. The PID controller was tuned using the following method:

1. The proportional gain was adjusted so that the rover oscillated about the target orientation.
2. The derivative gain was increased to stop the oscillation about the target orientation but the steady state error was greater than zero.
3. The integral gain was increased for small errors only so that the steady state error was reduced to zero.

The PID controller was tested using code to drive the rover a short distance along a fixed bearing and then rotating to return to the start point. It was noted that there was some gyro drift because the gyro did not consistently measure the same angle while stationary. In the finished system, this issue was overcome by using the camera to check the location of the rover in relation to the required path and adjust the orientation angle accordingly.

3.1.4 Communications

Once the control system of the rover was established, the communication system was built. For the communication system, a 3DR 433MHz radio modem (3D Robotics, Berkeley, Ca) is connected to the Arduino microcontroller which provides a serial communications link to another 433MHz radio modem connected to the laptop. This enables data to be transmitted to and from the rover from the laptop, sending instructions such as the optimal path to follow and any obstacles that may be encountered and their position from the rover. Data was transmitted at 57600 baud using the 8-N-1 configuration, that is 1 start bit, 8 data bits, no parity bit and 1 stop bit.

Data transmitted from the rover, apart from debugging information used during the building and testing phase, indicated the relative location of any obstacles detected by the ultrasonic sensor. The data was sent as a string, with the first character being "S" to indicate sonar data, then an angle value and a distance value separated by a forward slash "/". The angle was measured in degrees from the centre axis and the distance was measured in metres from the sensor. For example, "S/45/0.23" indicated an obstacle at 45° from the centre axis and 0.23m from the sensor as shown in Figure 3.5. This data was then split by the python algorithm at the receiving end using the forward slash to separate values.

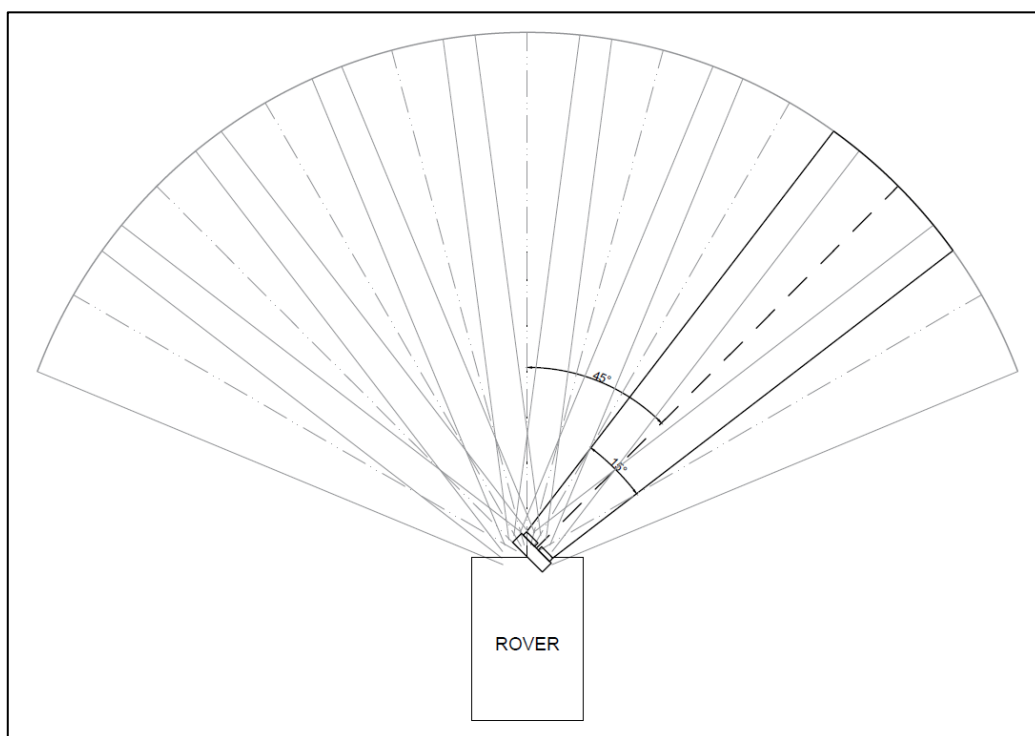


Figure 3.5 - Ultrasound sensor rotated 45° from the centre axis showing sensor field. The servo rotates 15° per iteration until a full sweep of the area in front of the rover is complete.

Data transmitted from the laptop was a heading error and distance error. This data was also sent as ASCII text and was surrounded by “<” and “>” to indicate the start and end of a transmission. This was required as the Arduino code does not automatically separate transmissions using an end of line as Python does. The angle error and distance error within the “<” and “>” were again separated by a forward slash, for example <19.3/20> would indicate that the rover needed to rotate 19.3° clockwise and move forward 20 pixels. A negative value for the rotation would indicate an anti-clockwise rotation was required.

3.1.5 Obstacle Detection

Obstacle detection required actions by both the rover and the machine vision, to identify the location of obstacles relative to the rover. On the rover, the ultrasonic sensor checked for obstacles after a set distance travelled, then transmitted the information to the path-finding algorithm. The heading of the rover, as determined by the machine vision system, was used to determine the location of the obstacle in relation to the rover and target. If an obstacle was detected, the path-finding algorithm was run again and a new path used to transmit instructions to the rover. The system then checked for obstacles again after a specified distance was travelled. The obstacle detection routine operated as follows:

1. Check for obstacles after a set distance travelled, if an obstacle is detected by the rover, transmit the data to the machine vision system.
2. Use the heading of the rover as determined by the machine vision system to place the obstacle in the test area.
3. Recalculate the path to the target location.
4. Direct the rover to the next waypoint.

On the rover, the ultrasonic sensor was rotated on the servo 15° at a time and a check for obstacles was performed after each rotation. This task was performed only when the rover was stationary to remove any error incurred due to measuring from a moving platform. Because the sensor detects any object within a 15° cone, beyond a certain point, echoes may be received from the floor. To remove these false floor readings, only echoes received from 0.4m or less were recorded as obstacles and transmitted to the path-finding algorithm. The check for obstacles was performed every 0.3m, so no obstacles were missed when the rover moved beyond the range of the previous obstacle check.

3.1.6 Rover Testing

At each stage of the process of building the system, testing was performed to determine whether the system was behaving as required. One of the major milestones, was building the rover with all the required systems apart from obstacle detection. To test the gyroscope, serial communications, PID and motor systems working together, a working remote-control rover was programmed. The serial data to be sent was one of either an ASCII 'a', for rotate left, 'w' for move forward, 'd' for rotate right and any other value to stop the rover. The gyroscope was used to measure the rover heading and maintain the direction when the rover was moving forward. Once this was working successfully, the work continued on to using the video feed for path-planning and localisation with machine vision.

3.2 Localisation

The overhead camera used was a HD C920 webcam (Logitech, Newark CA). This was installed in a fixed position on the ceiling, directly over the test area. A two-metre extension USB cable was used to connect the camera via its existing one-metre USB cable directly to a laptop, when used for rover guidance. Machine vision techniques were used to localise the rover and target in the test environment and determine the rover orientation. The techniques used included colour thresholding and morphological operations as described below and were implemented in Python using the OpenCV library.

On top of the rover, were two different colour squares that would contrast with the background floor. A third colour square was used to represent the target location and was placed on the floor as shown in Figure 3.6. For each colour, a range was determined using the HSV (Hue, Saturation, Value) colour system. In this colour system, each colour is represented by a three-column row vector, with each member of the vector being a value from 0 to 255. The first column, hue, represents the colour. The second column, saturation, represents the amount of colour with zero being no colour, or completely white and 255 being fully saturated. The third column, value, represents the amount of shadow in the colour, with 0 representing completely shaded, or black and 255 being not shaded at all. The objective of the machine vision algorithm was to determine the location of each colour within the test area.

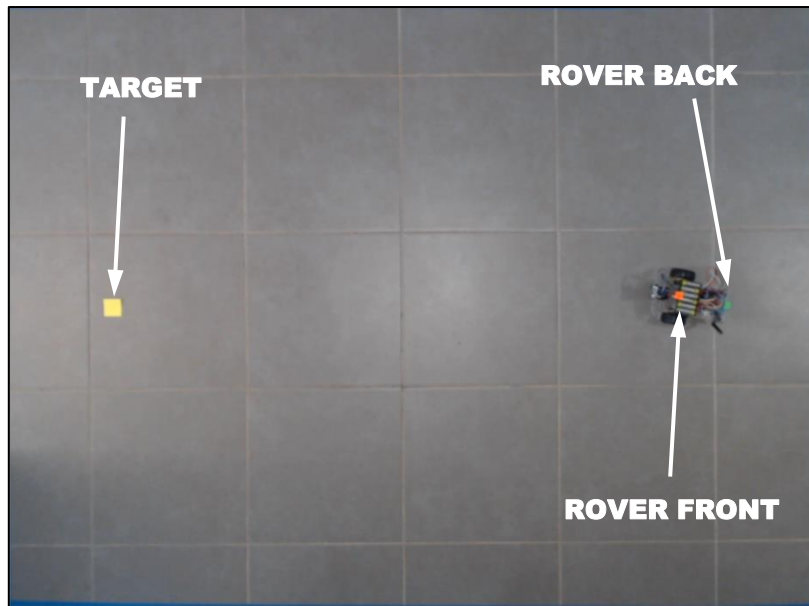


Figure 3.6 - Unprocessed image captured by the camera. The “target”, “rover back” and “rover front” target colours were deliberately chosen to contrast with the background and be large enough to be identified using the overhead camera.

The first step for the python algorithm was to find the colours of interest within the test area. The colours were defined using an upper limit and lower limit of the HSV values for each colour. For the hue values, the range was determined by estimating the value then placing a range of ± 10 on that value. Using this method, the colour could be isolated within the frame. The saturation and value were chosen to be anything from 50 to 255. This would not pick up ‘very light’ or ‘very dark’ colours with the correct hue but would still allow for a wide range of light conditions.

Once the colours were isolated, two morphological operations were performed to reduce noise and fill holes in the image to make the next step of the process easier. The first operation was erosion, which had the effect of reducing the size of the object by a specified amount. This was used to remove very small objects that may fall in that colour range. As the only object of interest was of a substantial size when compared with other objects in the frame, this operation had the effect of completely removing any pixels within the frame that fell within the colour range of the object of interest. The next operation was dilation, which was the opposite of erosion. This expanded the object of interest by a specified amount. The effect of this was to smooth the edges of the object to improve the efficiency of the next step of the process. The original image shown with the processing and final image are shown in Figure 3.7.

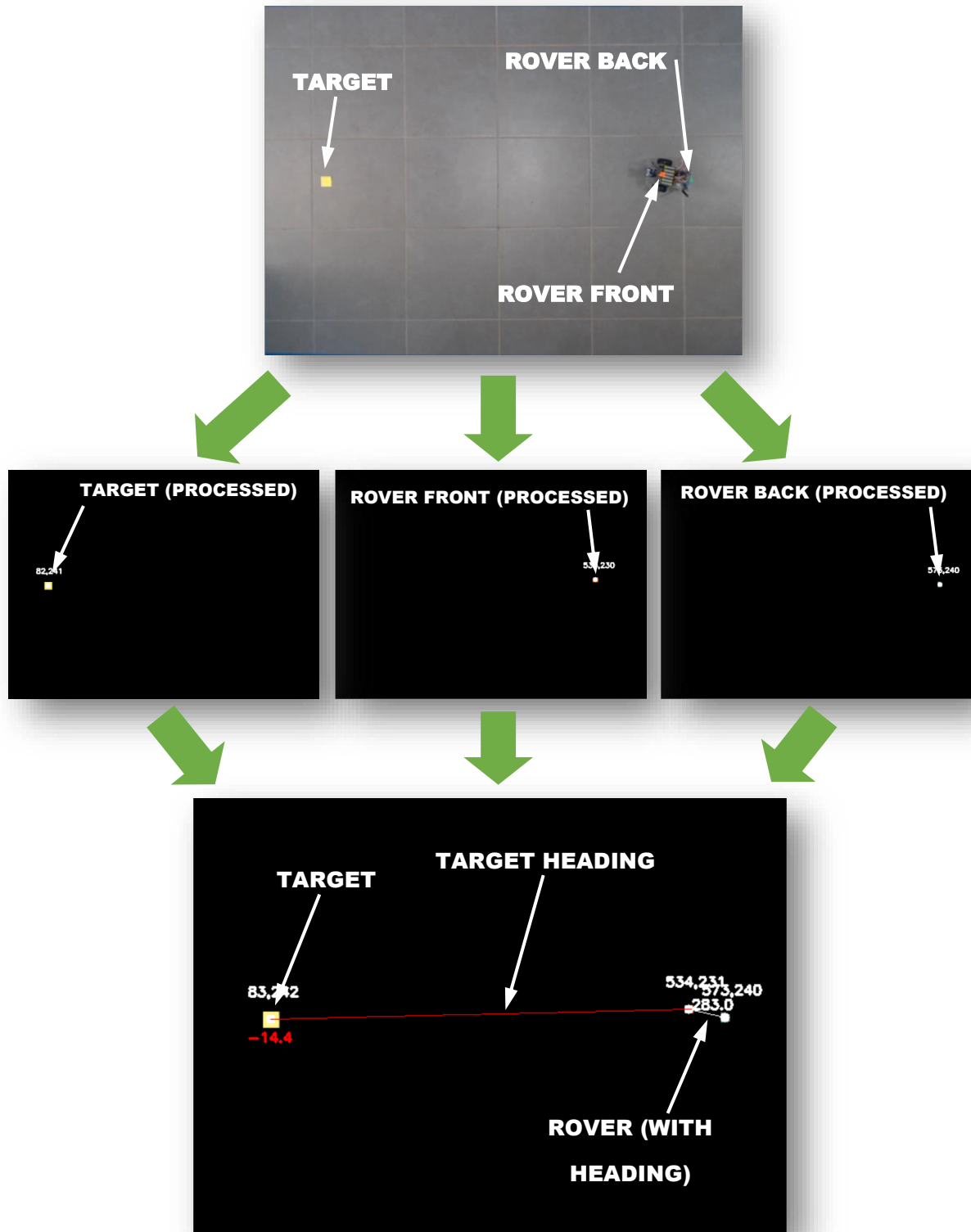


Figure 3.7 - The original image is captured (A) and the important colours are identified and split into separate images (B) with the target, rover front and rover back colours in each image. The centroid of each colour is calculated from these images. These images are then combined and the final image (C) is used to calculate rover heading and target heading.

Once a solid image of each colour without noise was established, the centroid of each colour could be used to determine rover location, rover heading and target location. The “moments” function within OpenCV was used to find the centroid of each colour square, returned as a set of cartesian coordinates representing the location within the frame. After this step, trigonometry and vector algebra was used to establish the centre of the rover, the heading of the rover and the direction to the target. As there can be ambiguity using inverse trigonometric functions, the convention used for the heading was to have 0°/360° at the top of the screen and headings increasing clockwise, following the same rotation convention used by the MPU6050 gyroscope. This convention is shown in Figure 3.8.

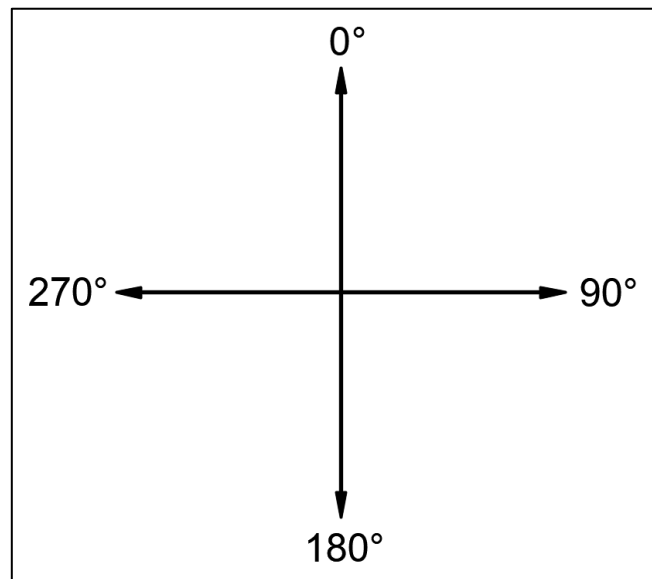


Figure 3.8 - Angle convention used by the machine vision algorithm, based on the rotation convention of the gyroscope. 0° was located at the top of the captured image.

The heading of the rover was calculated using the front and back colour targets:

$$heading = \tan^{-1} \left(\frac{x_{front} - x_{back}}{y_{front} - y_{back}} \right)$$

However, because multiple headings can have the same tan ratio, a series of ‘if’ statements were required to return the correct heading:

If $x_{front} > x_{back}$ and $y_{front} < y_{back}$:

$$heading = \tan^{-1} \left(\frac{x_{front} - x_{back}}{y_{front} - y_{back}} \right)$$

If $x_{front} > x_{back}$ and $y_{front} > y_{back}$:

$$heading = \tan^{-1} \left(\frac{x_{front} - x_{back}}{y_{front} - y_{back}} \right) + 90$$

If $x_{front} < x_{back}$ and $y_{front} > y_{back}$:

$$heading = \tan^{-1} \left(\frac{x_{front} - x_{back}}{y_{front} - y_{back}} \right) + 180$$

If $x_{front} < x_{back}$ and $y_{front} < y_{back}$:

$$heading = \tan^{-1} \left(\frac{x_{front} - x_{back}}{y_{front} - y_{back}} \right) + 270$$

After the rover heading was determined, the heading required to drive to the next target was determined in a similar manner by using the centre point of the rover, measured as a distance between the front and back of the rover and the target. The difference between the target heading and the rover heading was calculated as an error, and it was this error that was sent to the rover. This is illustrated in Figure 3.9.

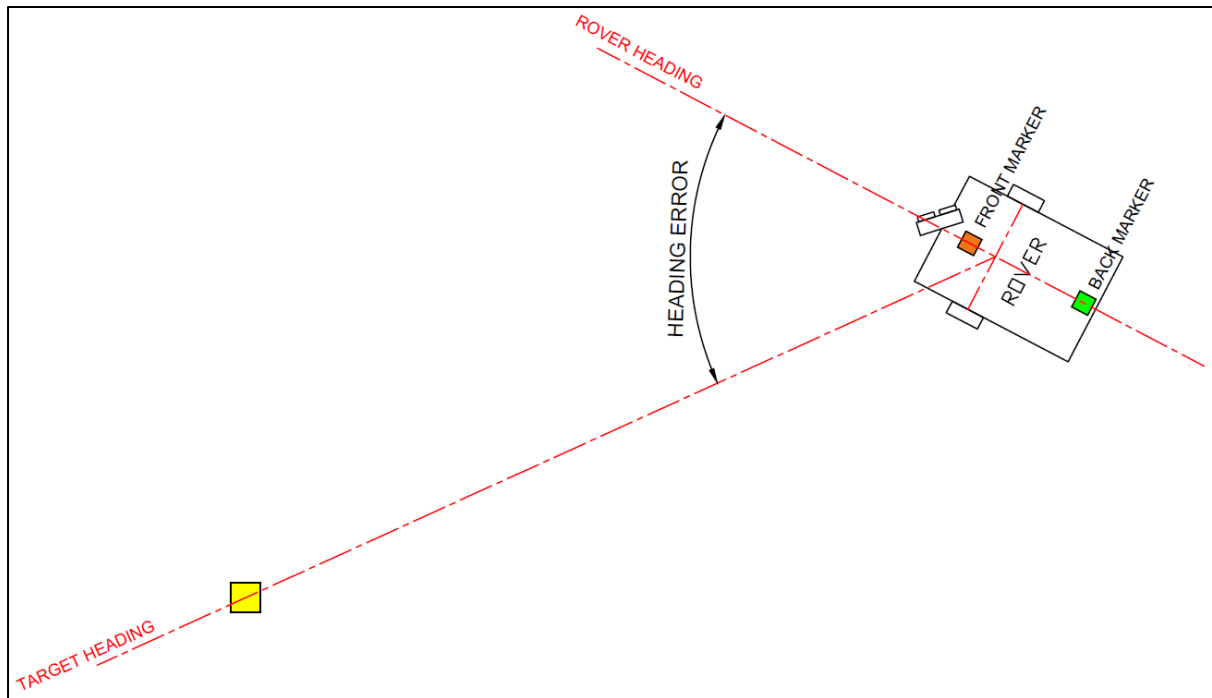


Figure 3.9 - The heading error is calculated from the difference between the rover heading and the heading to the target. The target heading is calculated using the front and back rover markers. Using the positions of these markers, the centre of the rover could be calculated. The heading from the centre of the rover to the target could then be calculated and the heading error was the difference between these two directions.

3.3 Path-Finding Algorithms

The two path-finding algorithms tested were the A* algorithm and the Potential Field algorithm. Both of these algorithms are based on a 2D grid and use heuristic methods to find a path from a start location to a goal location, given a grid with obstacles present. Neither of these algorithms are guaranteed to find the shortest path, however they both provide good approximations to the shortest path. Both of these algorithms calculate the shortest path given a known environment, so each time an obstacle is detected, the path must be recalculated. As such they are not as useful for dynamic obstacle avoidance, where the obstacle may be present at one time but not at another, however this scenario is unlikely to be encountered on Mars at present.

The A* algorithm uses an algorithm to consider each location on the grid and the cost of moving to that location, with the cost of moving to the goal location. In the case of this research the cost is the distance but there can be other parameters that are considered, such as the slope of the terrain or the “traversability,” which is a measure based on the roughness of the terrain. The algorithm looks at every possible combination of paths and finds the path with the lowest cost. The output is a series of waypoints, in cartesian coordinates.

The other path-finding algorithm that was tested was the potential field algorithm. This algorithm calculates a value for each grid location, a “potential” based on the distance from the goal location and the proximity of any obstacles. It is possible to associate a “gain” for each of these, so that an obstacle would have a high negative gain while the goal location would have a positive gain. The algorithm then determines the path to the obstacle by following the highest value from each grid location to the next, until the goal location is attained. Again, the output from the algorithm is a series of waypoints in cartesian coordinates.

Once a series of waypoints has been determined, the next step is to use this information to direct the rover. A short Python routine was used to determine which waypoint was current and direct the rover to the next waypoint in the path. The assumption was made that the rover location did not change between measuring the rover location and heading and calculating the path to the goal location. This is a reasonable assumption for the project because the rover was always stopped while the path was calculated, a task which could take up to a few seconds. The first waypoint would then be the current rover position. Once the rover was within a specified distance of the current waypoint, the Python algorithm would detect this and set the target to the next waypoint. For the current waypoint, a distance error and heading error combination was determined using the current rover position in cartesian coordinates within the overhead camera footage.

A limitation with both of these algorithms is that only limited headings can be used, however these can be increased as necessary. It is also possible to change the grid size and the robot radius to suit the application. Both of these algorithms calculate the path to the goal taking into account known obstacles, so if a new obstacle is encountered, the algorithm needs to be run again to find the path to the goal, taking into account the newly discovered obstacles.

3.4 System

Each part of the system described was required to work together with the other parts of the system, to perform the task of travelling to a target while avoiding any obstacles along the way. This required communication between different systems using different conventions, such as passing serial data from Arduino to the Python algorithm, to determine the location of obstacles. Extensive testing was performed, to determine the exact form data would take when it was received, to ensure the data was handled correctly. A flowchart of the entire system with both Arduino rover tasks and Open CV tasks is shown in Figure 3.10. The final code used by the Arduino microcontroller is listed in Appendix B, and the final Python code used by the machine vision algorithm is listed in Appendix C.

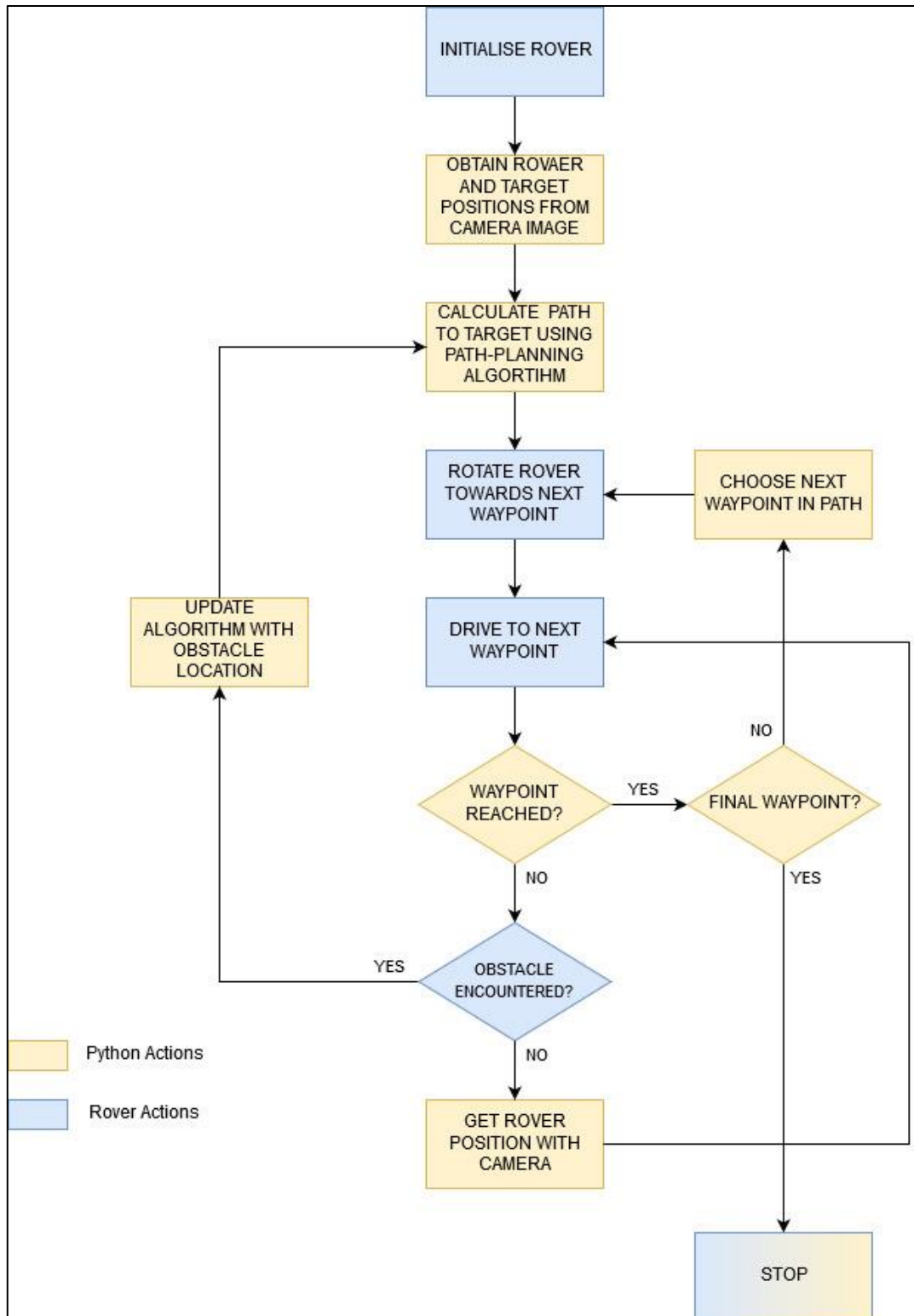


Figure 3.10 – The system flowchart. Initialise rover: the rover is powered on and input/output pins set, gyroscope initialisation conducted to remove gyro drift. Once the initialisation is complete, the position of the rover is determined using the capture from the overhead camera. The path-planning algorithm then calculates the path and the rover is sent the required information to travel to the next waypoint. If the rover arrives at the waypoint, the path-planning algorithm will send the rover the data required to travel to the next waypoint. In the event of an obstacle being detected, the path will be recalculated and the rover will then travel to the next waypoint on the new path. Once the final waypoint is reached, the system stops.

4 Results

In order to test the performance of the system, a series of tests were performed at each stage of the building process. The stages of the building process were:

1. **Rover control and communications protocols:** The rover was tested without any human input on simple forwards and back loop to improve the performance of the turn and drive forwards functions. A radio modem was installed on the rover to allow control of the two functions remotely. At this stage the control was by a human but the later on in the build process the control was performed by a computer.
2. **Localisation (machine vision):** The coloured indicators were installed on the front and back of the rover for this test. There was some trial and error required to achieve the correct HSV values for the coloured squares. In addition, the target was identified at this stage and the heading to the target could be calculated within the Python code.
3. **Path planning and control:** In this stage, the code used the error between the rover heading and the heading to the target to send instructions to the rover to rotate towards the target. Once the rotate command was operating successfully, the “forward” command was added to rotate the rover towards the target and then drive in a straight line to the target. The two path-finding algorithms then used were the A* and potential field algorithms. These two algorithms coded in Python were freely available by Sakai et al. (2018). The algorithms used the starting rover location and the target location. Obstacles were artificially inserted into the algorithms using code. Once a path was calculated the rover could follow it using the “forward” and “rotate” commands.
4. **Obstacle detection and avoidance:** The sonar was tested on the stationary rover to check an obstacle could be detected and located correctly within the test grid. After an obstacle was identified by the traversing rover, a new path was determined and the rover then followed the new path.

4.1 Tests performed prior to Machine Vision

The first two tests were performed before the computer vision system was set up, in order to test the functioning of the rover and set operating parameters. The first test was on the rotation and forward movement functions. To test these two functions, a short code was written to drive the rover forward for a fixed time then rotate 180° and repeat indefinitely. At first, the rover repeated the same track with good accuracy, however over time the difference in heading between the first track and the last




track became greater as the gyro drift error increased. The PID parameters set from this test were $K_p = 11$, $K_i = 0.3$, $K_d = 0.6$. Using these parameters, the rover held the heading while driving and rotated quickly to the new heading at the end of the forward movement.

The second test was to send the “forward” and “operate” commands to the rover using the radio modem and serial communications and observe the performance. Simple ASCII codes were sent to the rover using serial communications and the rover performed as expected, executing the tasks when commanded. These two tests combined, set the rover up for the next stage of testing, which was to have the OpenCV/ path-finding algorithm send the commands rather than a human.

4.2 Establishing Machine Vision Parameters

The machine vision code was first tested to determine if it could locate the colours required to identify the rover and the target. Distinctive colours were used to ensure they would contrast with the background of the testing area. With some trial and error, the HSV values for the colours were determined as shown in Table 4.1.

Table 4.1 - Colour markers with associated HSV values

Location	Colour	Lower Limit (H, S, V)	Upper Limit (H, S, V)
Target		25, 40, 50	35, 255, 255
Rover Front		5, 67, 50	22, 255, 255
Rover Back		55, 50, 50	80, 255, 255

The upper and lower ranges for each colour were determined over several days with different lighting conditions. The testing was performed inside with the lights on but there was still some influence on the colours from the outside light conditions. Once noise was removed from the image and the centre of each colour correctly identified in the image, the data could be used for the next test.

Figure 4.1 shows the effect of the morphological operations on the orange colour target. Once the threshold is applied, there are still some unwanted parts of the image that register as orange. After the morphological operations are applied, the clean image shows only the required target.

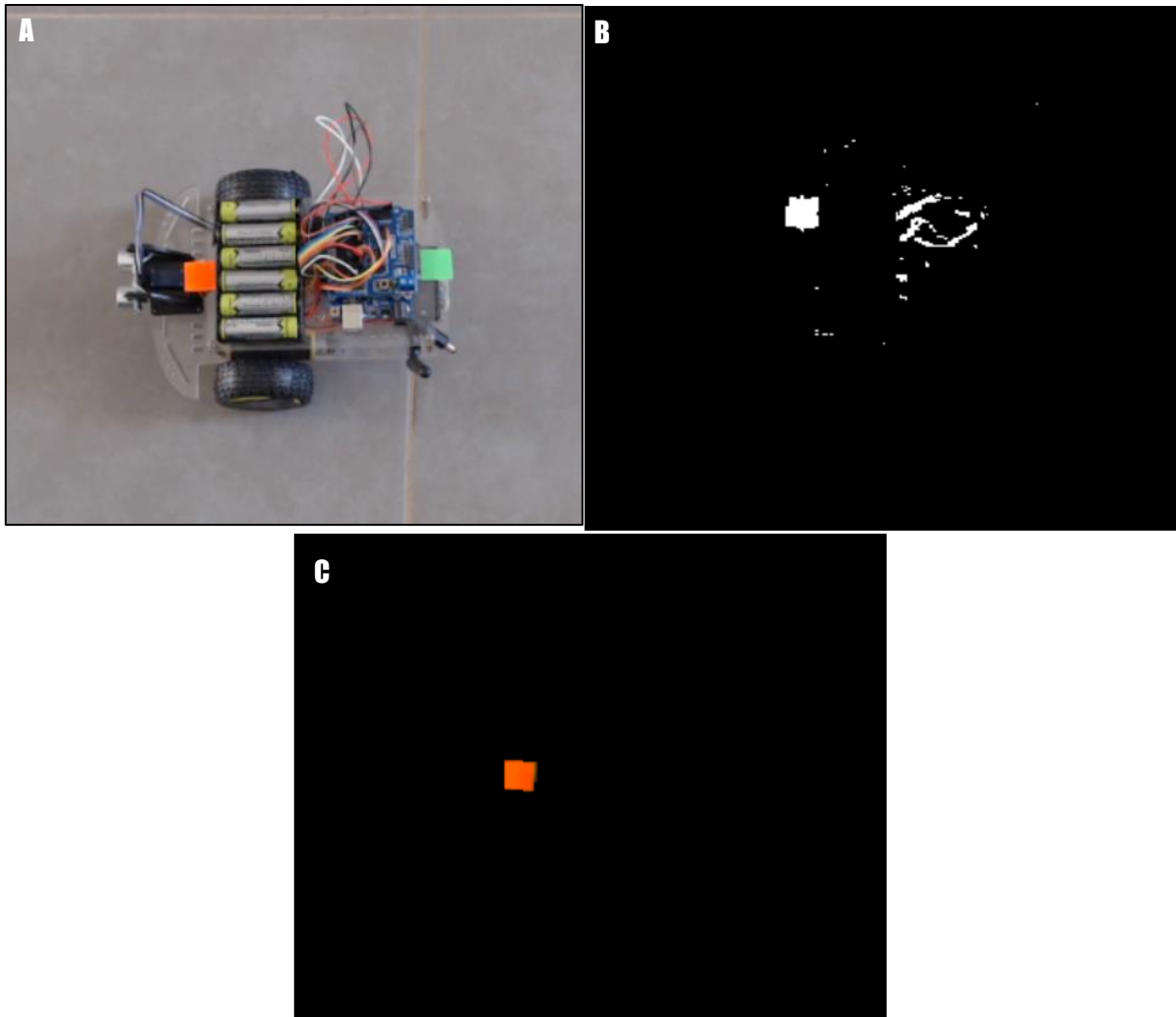


Figure 4.1 - The results of applying morphological operations to the captured image. In the original image (A), the orange pixels are identified and isolated as shown in (B). Unwanted pixels are removed using morphological operations to produce the final image of the orange target (C).

4.3 Motion Control Refinement

The heading error calculated from the previous test was sent to the rover and the rover rotated to the target. There were two tests performed for this stage, in the first test the heading error from the overhead camera was used as the angle input to the PID controller. This resulted in oscillation about the target heading as shown in Figure 4.2. The second test used the heading error as sent by the overhead camera, as well as the heading measurement from the onboard gyroscope, to rotate the rover to the correct orientation. The second test resulted in the rover settling to the target orientation without oscillation within a few seconds, which can be seen in Figure 4.3.



Figure 4.2 - Rover attempts to orient using overhead images to obtain heading error. 1). Heading error at 2.17s is -33.9°. 2). Heading error at 2.73s is 23.7°. 3). Heading error at 3.30s is -27.1°. 4). Heading error at 3.67s is 36.9°.

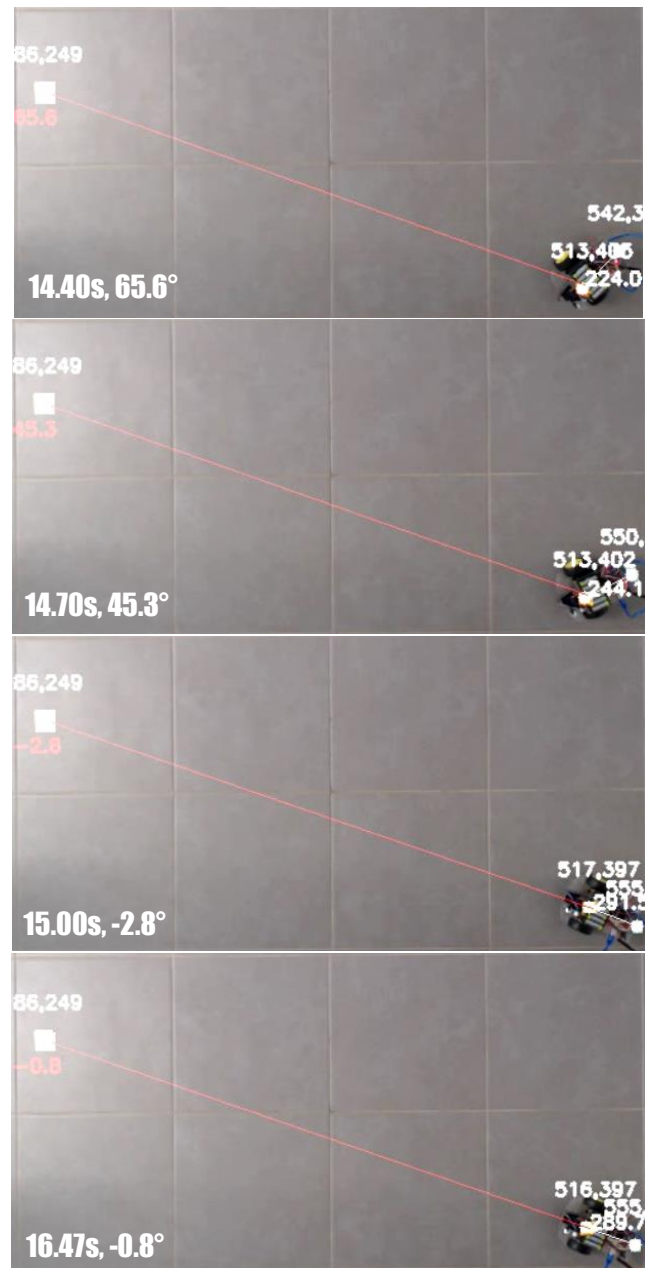


Figure 4.3 - Rover settles after a few seconds when using gyroscope to obtain heading error. 1). Heading error at 14.40s is 65.6°. 2). Heading error at 14.70s is 45.3°. 3). Heading error at 15.00s is -2.8° . 4). Heading error at 16.47s is -0.8° .

Once the rotation could be controlled by the computer alone, the next test introduced the “forward” command to computer control. For this test, the motors were set at a fixed speed until the rover was within a short distance of the target location. Once the rover was within the specified distance of the target, the motors were set to zero. If at any time the rover heading was greater than 10° away from the target, the rover would stop and rotate to a new heading. This test was completed successfully with the rover moving towards the target and stopping more frequently as the distance to the target was reduced. After this test the limits for the target heading were changed so that they were a

function of the distance to the target. If the heading error was θ and the distance from the rover to the target point in pixels was L then:

$$\theta = \tan^{-1} \frac{10}{L}$$

A plan view of this is represented in Figure 4.4, where the distance is measured in pixels as observed by the overhead camera.

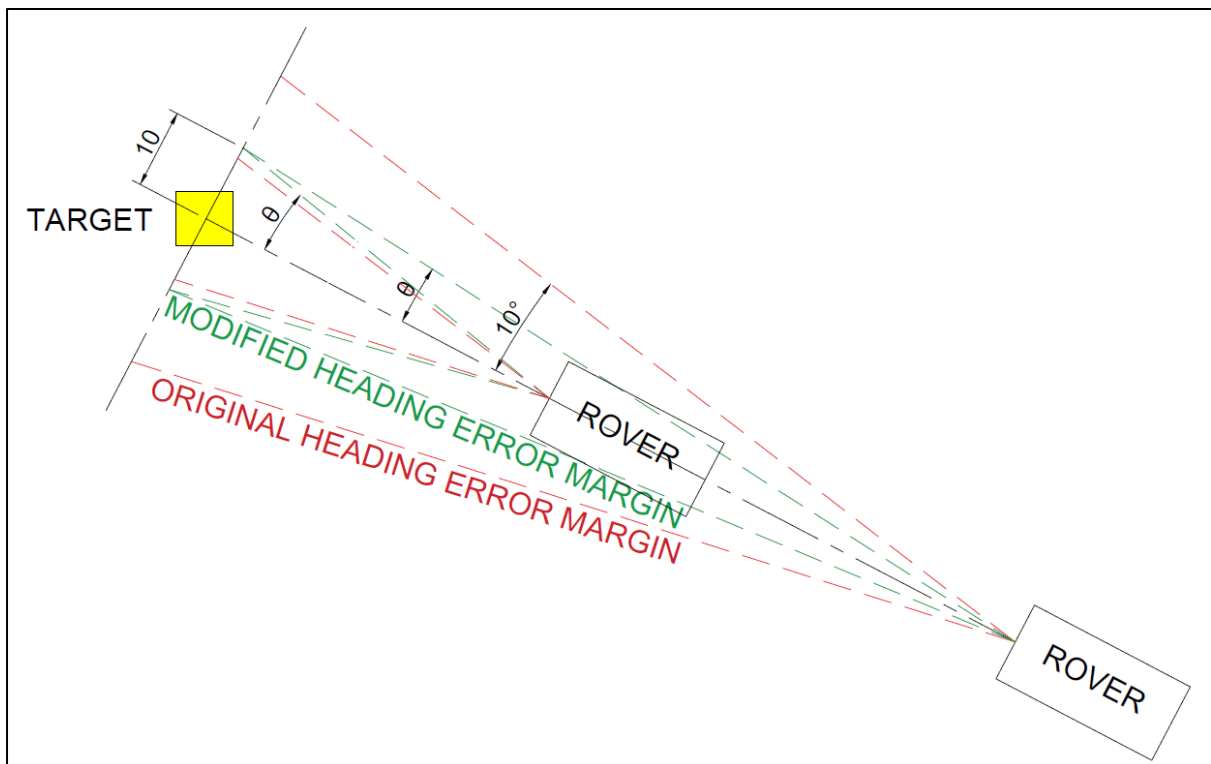


Figure 4.4 - Modified heading error compared with original heading error.

After the rover could be directed to a given target using the data from the overhead camera, two different path-finding algorithms were used to determine a path from the initial rover location to the target location. Obstacles were artificially inserted into the path with code, to test the ability of the rover to follow a path to the target. An initial test was run and it was found that the rover would deviate from the target path due to the limits set on the deviation from the target heading. This deviation was high enough that it would have caused a collision with a physical object as shown in Figure 4.5. It was also found that the overshoot of the “rotation” function was too high when the rover was turning to face a waypoint, with the rover rotating up to 45° past the target heading in some cases. This was especially true when the initial error was large (greater than about 90°). An example of this is shown in Figure 4.5.

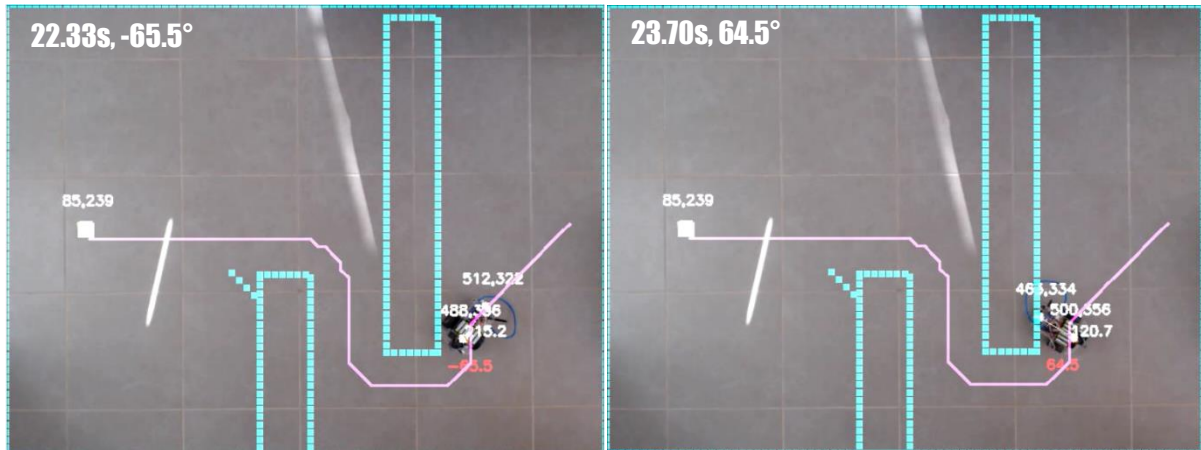


Figure 4.5 - Rover overshoots target heading when rotating, at 22.33s the heading error is -65.5° , then at 23.70s the heading error has overshoot by a maximum of 64.5° . In this scenario a coded obstacle was used but in a real scenario the rover would have collided with the obstacle.

To deal with the deviation from the target heading, the allowed heading deviation as observed by the camera was halved. This resulted in the rover following the calculated path with a maximum deviation from the target of 5 pixels either side of the target. This represented about 18mm compared with the previous tolerance of 10 pixels or 36mm. The new equation for calculating the maximum allowed heading error before the rover stopped and corrected was:

$$\theta = \tan^{-1} \frac{5}{L}$$

To remove the excessive overshoot of the rotation function, the parameters of the PID loop were again altered by trial and error. The new values were $K_p = 30$, $K_i = 3$, $K_d = 5$ which resulted in a much higher damping of the rotation speed due to the higher K_d . This produced a variable rotation speed, which appeared as a vibration when the rover was turning towards a target but the overshoot was greatly reduced. After making these modifications to the code, the rover followed the path towards the target with lower heading deviation than previously encountered.

The two path finding algorithms were tested in two different scenarios. The first scenario consisted of large obstacles placed in the path of the rover as shown in Figure 4.5, while the second scenario was coded with smaller obstacles in the path of the rover. Figure 4.6 shows frames from the machine vision system, with the rover following a path to the target location using the potential field algorithm. The A* algorithm successfully found a path to the target in both cases while the potential field algorithm was only able to find a path to the target in the second scenario. These results are summarised in Table 4.2.



Figure 4.6 - The system using the potential field algorithm to navigate through a course of smaller obstacles. Four frames are shown at times 0.00s, 32.30s, 1:08.94s and 2:04.36s.

Table 4.2 - The results of using different path-finding algorithms.

Algorithm	Test Environment	Result
A*	Large Obstacles	Rover arrived at target
A*	Small Obstacles	Rover arrived at target
Potential Field	Large Obstacles	Failed to find path (local minimum)
Potential Field	Small Obstacles	Rover arrived at target

To test the operation of the sonar system for detecting obstacles, the sonar was first tested with the rover stationary, to determine the object detection capability. The servo operated as programmed to rotate the ultrasonic sensor 15° at a time. This obstacle then appeared on the overhead camera footage in the location detected. There was some discrepancy between the actual location of the

object and its indicated position on the camera footage, which was estimated to be about 0.05m as shown in Figure 4.7.

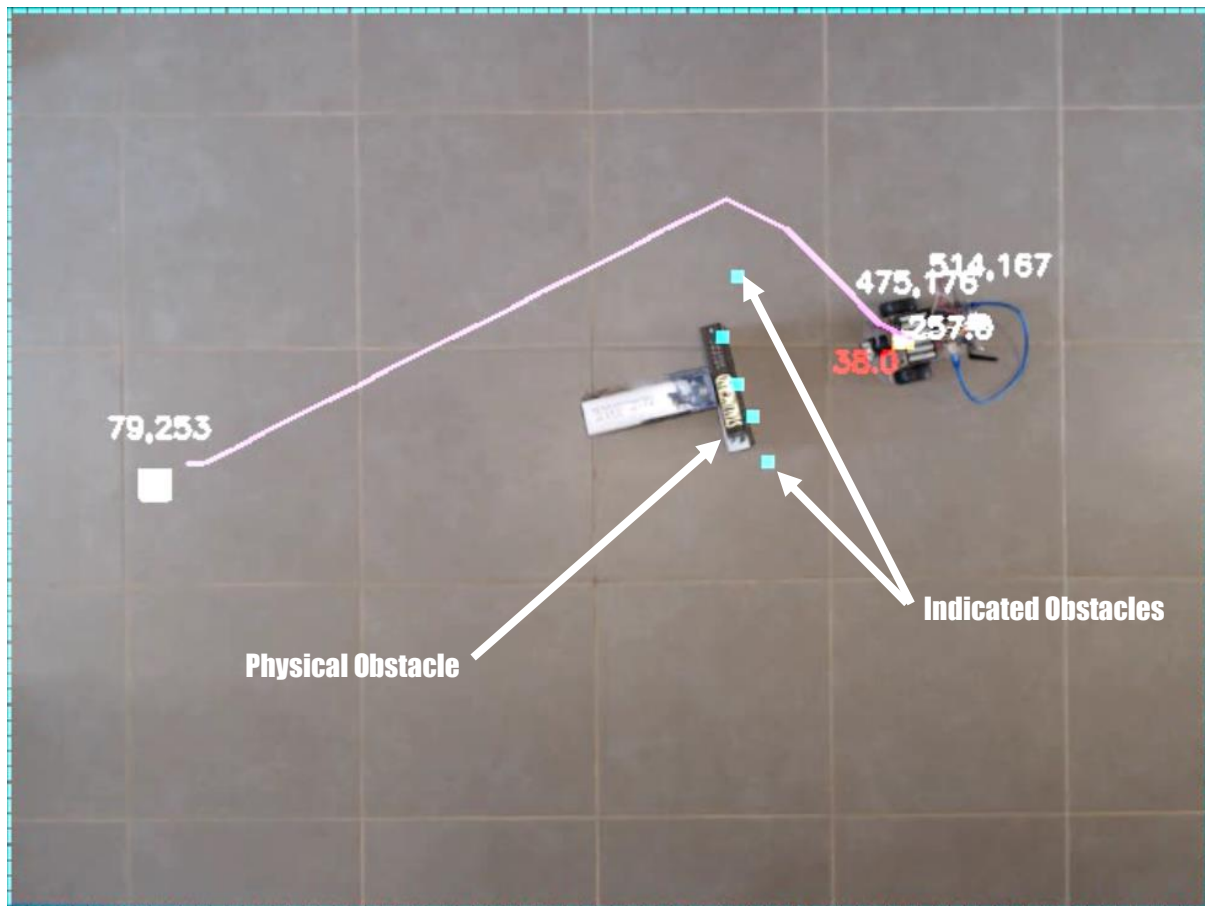


Figure 4.7 - The indicated position of the obstacles measured were outside the limits of the actual obstacle.

Once the sonar was operating on the stationary rover, the code was modified to operate on a moving rover. This involved measuring the distance travelled using the overhead camera and stopping the rover at short intervals to run the sonar routine. Again, this proved successful, the same result was achieved as with the stationary rover, with the obstacles indicated as being slightly outside the boundaries of the actual obstacle. The new path was calculated using the A* algorithm with the additional obstacle present. The rover then followed the new path as with the previous tests, using coded obstacles as shown in Figure 4.8. This was the final test performed.

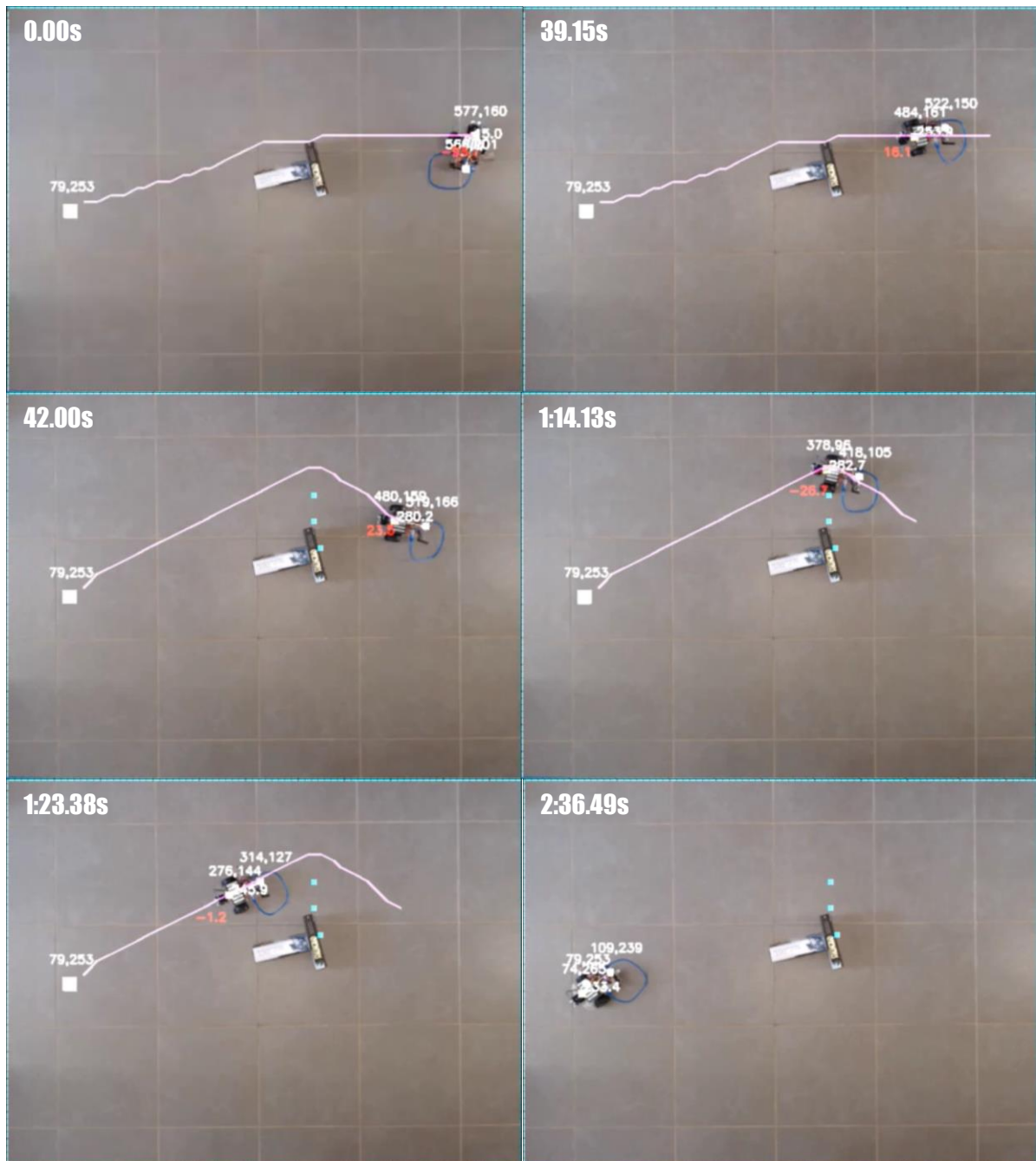


Figure 4.8 – The final test using obstacle detection and the A* algorithm to navigate an environment with a real obstacle. At the start (0.00s), the path is calculated as if no obstacles are present. At time 39.15s the rover approaches the obstacle while periodically checking for obstacles. At time 42.00s the obstacle is detected and a new path calculated. The rover then follows the new path (1:14.13s and 1:23.38s) until the target location is reached (2:36.29s)

5 Discussion

The aim of this project was to investigate localisation and navigation in an unstructured environment such as that found on Mars. For the project it was assumed that there would be no satellite navigation that could be used for localisation and no magnetic field to use as a direction reference, as is the case on Mars. To overcome these challenges, a concept was developed that would use machine vision to both localise and navigate a small rover through a test course. To evaluate the concept, the system was built and tested in a real environment, as opposed to a simulation, to better capture sensing and system integration practicalities.

The results of this project showed the system is a feasible method for autonomous navigation of a partially unknown environment. By simplifying the problem of autonomous navigation on Mars, several assumptions were made that may need to be addressed before a system such as this were tested on a more realistic situation.

- The surface over which the rover was traversing was perfectly flat. This may not be the case on Mars but it could be close enough that it would make little difference to the system.
- The overhead camera remained motionless and was positioned in the centre of the test area. An actual platform for an overhead rover on Mars would almost certainly move whether due to operational requirements or weather conditions. Testing the performance of the vision system while the camera was moving would be part of future studies.
- The overhead camera in this project pointed directly down. If the camera platform in a real Mars situation were moved away from directly over the rover, it may be observed from an angle. The effectiveness of the vision system from various angles would need to be tested from various angles.
- The path to the target is within the field of view of the overhead camera. Even though it is unlikely, the field of view may need to be moved to find a path to the target. If this were the case, an autonomous method of moving the overhead camera to find a path for the rover would need to be developed.

5.1 Rover Control and Communications Protocols

The rover performed well as an experimental platform. It received instructions from the machine vision algorithm and followed the calculated path as required. The instructions to the rover were sent as a direction and distance to the next waypoint. The rover would then rotate towards the waypoint,

if not already facing that direction and drive forwards until the machine vision algorithm sent a message to drive to the next waypoint. A PID controller was programmed to allow rotation control through the use of an onboard gyroscope and it worked well to control the heading of the rover during rotation and also forward movement. The communications protocol also worked well and allowed many instructions to be sent per second. The success of these systems allowed the evaluation of different path planning algorithms. However, there were still some areas of the system that could be improved in the future.

5.1.1 PID Controller

To control the rotation of the rover, a PID controller was used with gain values determined by trial and error. The input to the control loop was the heading error, as supplied by the gyroscope combined with the overhead camera. The output was the voltage supplied to the motors. However, this output only had an indirect influence on the rotation. The voltage supplied to the motors provided a force to rotate them but there were other nonlinearities such as the surface the rover was on and the properties of each individual motor. In some situations, the PID controller produced a jerky response while in other situations the response was smooth. To achieve more efficient and accurate control, further work would be required into the sensors and control techniques needed for fast accurate rotation control. Another aspect to investigate might be the gearing system of the motor to improve the output linearity. The measurement of motor current may be useful for this purpose.

5.1.2 Rover Speed Measurement

The speed of the rover was not measured directly by any sensor. A simple technique of stopping the rover once it was within a defined range of the target was used, which was successful for this project. However, if more precision were required to control the position of the rover, an odometer could be used to measure the speed of the rover. Using speed information with distance information obtained from the overhead camera, would enable more accurate positioning of the rover at the target waypoint.

Two possible options for an odometer are available, either a wheel encoder that measures wheel revolutions or an optical odometer that uses the same principle as a computer mouse to accurately measure speed. A wheel encoder would be the simplest method but it would be subject to error depending on the accuracy of measuring the diameter of the wheel or wheel slippage. An optical odometer measuring actual distance travelled, would be more accurate but may not work as well on

rough terrain or poor light conditions. Considering the constant error correction available from the overhead camera, a wheel encoder would be an appropriate choice for this application.

5.1.3 Communications

The communications protocol was not a major research topic for this project, however some experience was gained that may prove useful to future projects. The 433Mhz radio modems used for communications have a range of around 500m with a data transfer rate of up to 250kbps. This was much more than was required for this project and demonstrates the potential of even a small system such as this one. However, the code on either side of the communications network used different data types and had potential for simplification.

The rover was programmed using the Arduino IDE with the C++ language, while the machine vision was programmed using Python. Due to this, the data was handled slightly differently by each language, which had to be taken into account when processing the data. For example, Python appends a 'b' in front of any data sent and adds a 'b' in front of any data received to denote a byte data type. By using defined characters to represent the start and end of a string, it was a simple matter to then remove any extra characters from the data.

5.2 Localisation using machine vision

The machine vision section of the project gave consistent results. The colour targets could be easily distinguished against the background during day time conditions, however poor light or night time conditions would require a change in the colour thresholds. A number of other factors would need to be considered for future applications, such as the nature of the overhead camera platform and the resolution and position of the camera platform in relation to the rover.

5.2.1 Susceptibility to Light Conditions

Even though the testing was conducted indoors at around the same time every day, there was still some fluctuation in light conditions due to cloud or other factors. When this happened, there were times when the camera would be unable to detect the required colours, causing the system to fail. To mitigate the effects of lighting conditions, a future system could use specific coloured LEDs instead of colour markers. This would have the advantage of being visible at night and during poor lighting conditions.

To allow for some changes in lighting conditions, it was found that there was a large range of HSV values required to detect the necessary colours. As a result of this, there were occasions when other colours would enter the range of the required colours so that the system could not distinguish the target, for example, from a component on the rover. Using specifically coloured LEDs would also solve this related issue by allowing the narrowing of the range defined for each colour.

5.2.2 Camera Resolution

For this project, the camera resolution used was 640 pixels by 480 pixels. When calculating the centroid of a colour target, whether this was on the rover or the target location, the calculated centroid was related to an individual pixel. The coordinates of this centroid would therefore, by definition, be integers. Because the actual location on the ground or the rover was not an integer, there was a rounding error, this rounding error could vary from frame to frame, so even though there was no rover movement there could still be a change in the heading error, as illustrated in Figure 5.1. In most cases this was not a problem, however when the rover was closer to the target or waypoint, this rounding error could cause unnecessary correction in the rotation angle if the fluctuation in the position caused the heading error to move outside the allowed limits.

To minimise this error, a higher resolution image could be captured, which would result in a smaller physical distance represented by each pixel. In this project, the combination of the camera position and the image resolution, resulted in a pixel representing about 3mm by 3mm. With a higher image resolution, one pixel would represent a smaller area, so a target position fluctuating between one pixel and another would represent a smaller change in the heading error.

Another solution would be to have the field of vision narrow as the rover approached the target. In the case of a Mars based operation, this could be achieved by having the camera platform reduce altitude as the rover approached the target or through the use of a variable focal length lens. This would have the same effect as a high-resolution image, by reducing the physical distance represented by each pixel.

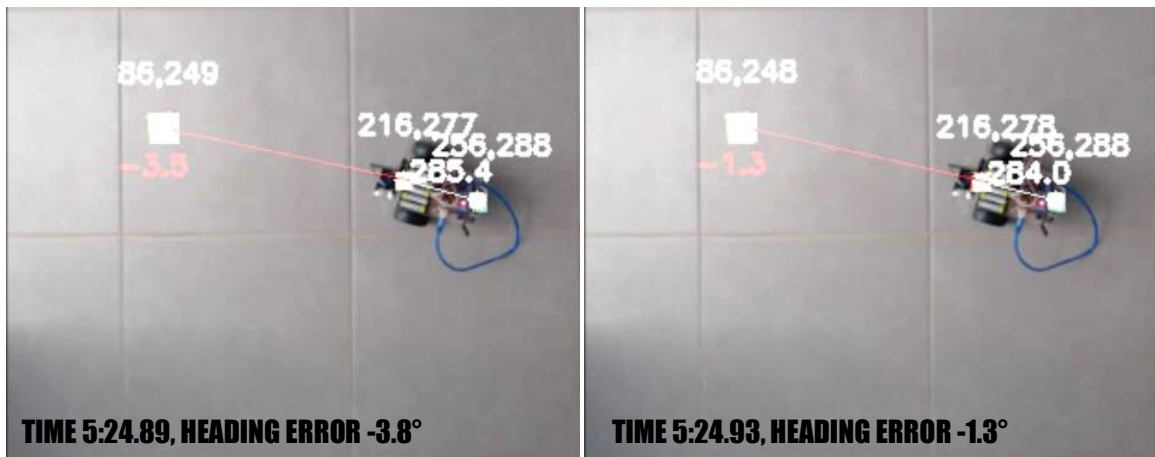


Figure 5.1 - The effect of rounding the position of colour targets results in variable heading errors even though the rover and target do not move. In the left frame the heading error is -3.8 . In the next frame, on the right, the heading error has changed to -1.3 even though neither the rover nor target have physically moved.

5.2.3 Camera Support Platform Motion

A key assumption for this project is that the camera would remain stationary directly above the rover operating area. In the case of a Mars mission, this may not be the case. If the camera were to move due to weather conditions or other requirements of the operation, a gimbaled camera platform would be able to rotate to keep the camera in view. This could be achieved through the use of optical flow as described by Walter et al. (2018). With this method, the terrain features are used in combination with the accelerometer/gyroscope data, to rotate a gimbaled camera to keep the same area in view. Even with this solution, the angle at which the rover was viewed would change, so more work would be required to determine the effects on the machine vision and the localisation and navigation.

5.3 Path-Planning and Control

The path finding algorithms tested for this project were A* and potential field, as made freely available by Sakai et al. (2018). Both of these algorithms limited the possible headings to 45° increments starting at due North (0°). In order to allow more possible paths, the algorithms were modified for this project to include heading increments of about 22.5° . There is no limit to the number of heading increments that may be used but increasing the number of possible headings will increase the number of calculations required by the path-finding algorithm and therefore increase the time taken for the computer to solve the problem. Ultimately it would be a matter of compromise between the number of headings used for the algorithms and the speed required for the calculation of these algorithms. There are also path-finding algorithms available that do not restrict the available headings in any way, such as Field D*, however the use of these algorithms was outside the scope of this project and would need to be the subject of further research.

The choice of path-finding algorithm is another matter for further research. From this limited study of the two path-finding algorithms A* and Potential Field, it is clear that while the A* algorithm requires more computing resources, it consistently finds a solution to the problem. However, there are circumstances where the Potential Field algorithm may provide a quicker solution that is comparable with the A* solution. Where the obstacles are small and there is no potential for a local minimum, the potential field algorithm would find an appropriate solution quickly. Perhaps the best solution is to use a combination of path-finding algorithms or to equip the autonomous unit with a selection of path-finding algorithms and let the software choose the most appropriate one using machine learning. Again, such a topic was outside the scope of this research and would need to be addressed by further work.

Another characteristic of the path-finding algorithms that would require further research would be the size of the grid and the number of waypoints. For this project, every time the rover reached a waypoint it stopped and rotated to the next waypoint, causing a delay. There are several ways to reduce the number of waypoints that could be the subject of further investigation. In the case of waypoints that are in a straight line, these could be removed from the final path with no detrimental effect on the outcome. These waypoints existed only because each iteration of the path-finding algorithms required movement to an adjacent grid-square. Another way to reduce the number of waypoints would be to increase the size of each grid square. In the case of this project, one grid square had a side length of 0.03m, which was related to the smallest grid square that could be used to solve the problem in a reasonable time. Increasing the size of each grid square would be another way to reduce the number of waypoints. The third way to reduce the number of waypoints would be to use another path-finding algorithm such as Field D* that did not require each waypoint to be on adjacent grid squares. Again, perhaps machine learning techniques would be suitable for the selection of the grid-size and the number of waypoints.

5.4 Obstacle Detection and Avoidance

For this project, an ultrasonic sensor (HC-SR04) attached to a servo motor was installed at the front of the rover to be used for obstacle detection. The servo motor rotated 15° at a time for the ultrasonic sensor to take distance readings at each 15° segment of a 135° arc across the front of the rover. As described previously, the 15° corresponded to the width of the ultrasonic sensor cone, as shown in Figure 3.5. While this was adequate for the project, there were several disadvantages that became apparent during the testing phase that would need to be addressed for any future applications.

Due to the nature of the ultrasonic sensor, the 15° cone of detection means that it is impossible to define the nature of an obstacle within the cone of detection. It may be to one side of the cone or in the middle, or it may encompass the entire cone. This effect is more pronounced the further away the obstacle is from the sensor. At 0.4m, the limit of obstacle detection coded into the rover, the width of the cone of detection was 0.11m. For this project, the obstacle was assumed to be present in the centre of the cone of detection, however it is possible that the obstacle only existed on one side or the other. This introduces a potential error of 0.05m between the calculated location of the obstacle and the actual location of the obstacle. To counter the effects of this error, the “robot radius” in the path-finding algorithm was increased by the same amount, so that any path calculated would avoid an obstacle by the actual robot dimensions plus 0.05m. Figure 5.2 illustrates the effect of increasing the robot radius on the calculated path.

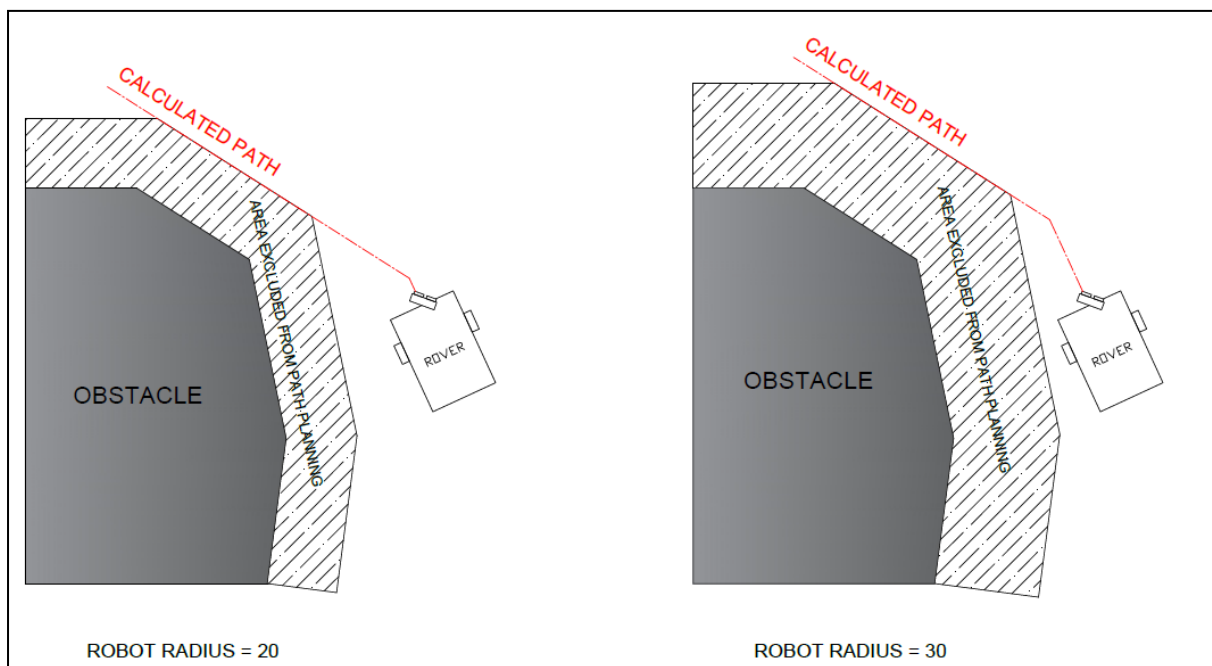


Figure 5.2 – The effect of increasing the robot radius used in the path planning algorithm. The robot radius defines the area excluded from path planning calculations around each obstacle. A larger robot radius will provide a greater margin for error, by the rover following the calculated path but may increase the length of the path calculated.

The range of detection for the HC-SR04 ultrasonic sensor is 4.0m, however in this project the practical range was about 0.65m as beyond this the sensor picked up the floor as an obstacle. Even without this restriction, at 4.0m, the potential error in calculated obstacle position compared with actual obstacle position would be up to 0.52m, which provides little benefit considering the dimensions of the rover are 0.15m wide by 0.21m long. This assumes very high precision of the servo motor in determining the direction of the ultrasonic sensor. If the servo direction was incorrect by 1°, at 4.0m

this possible error increases to 0.6m, making it unsuitable for anything but short-range obstacle detection. In addition to this, due to the very low atmospheric pressure on Mars, it is unlikely an ultrasonic sensor would be effective because it relies on a medium for sound waves to travel through. It is clear that an ultrasonic sensor would be unsuitable for obstacle detection in this application.

The most common method for obstacle detection in existing Mars rovers is visual odometry such as that described by Kubota et al. (2010). The advantage with this method is that the exact location of an obstacle may be determined. Other methods that could determine the exact position of an obstacle are millimetre wave radar, which can determine the position of an object accurate to within a fraction of a millimetre, or LIDAR which uses laser reflections from objects to determine distances. Both of these methods result in a point cloud defining the surrounding environment and result in additional computational cost compared with a simple ultrasonic sensor. Both millimetre wave radar and LIDAR have been successfully tested for use in autonomous vehicles on Earth.

Another type of obstacle that may be present on Mars is the surface over which the rover is driving. If the surface is loose, this may result in the wheels slipping or even completely losing traction. This project did not analyse this possibility as the testing was conducted indoors on a tiled surface, however it may be handled in the same way by the path finding algorithm, only the manner of detection would change. By using a wheel encoder to determine the number of wheel revolutions and comparing this information to the distance travelled, a loose surface could be determined. The distance travelled could be obtained from the overhead image or some other method. In the case of a rougher surface that could also slow progress, onboard accelerometers could measure vibration as the rover travelled over a surface. If the gradient of the surface became steeper than a pre-set limit, accelerometers could be used to sense the slope of the rover.

In any of these cases, the surface may be considered an obstacle even though it is not a physical barrier restricting the movement of the rover. Where it is possible for the rover to traverse a grid square more slowly than another due to the surface conditions, a “traversability” score may be assigned to the grid square. In this case a more sophisticated path-finding algorithm would be applied where the presence of an obstacle is no longer represented by a ‘1’ or ‘0’ but a continuum between these two values, representing the “traversability” of the grid square. Algorithms using a method such as this were described by Hayati et al. (2007) and Muñoz et al. (2017).

5.5 Hardware Issues

During the rover building and testing phase there were some technical issues that needed to be overcome. While these did not directly affect the results of the project, they did prove to be challenging to overcome. They are discussed here as they have implication for future development or modification of the system hardware.

5.5.1 Radio Modem

To communicate between the rover and laptop, a pair of 433MHz radio modems were used. These modems paired automatically when in range and powered, which could be determined by a solid green LED on each modem. Modems that are not paired will have a flashing green LED. Initially when connecting the radio modems to power, one on the Arduino and one at the USB on the laptop, it was found that the radio modems would not pair. The specifications for the radio modems state that the maximum power output is 100mW, which using a 5V supply would require 20mA. The output pin of an Arduino microcontroller can provide a maximum of 20mA at 5V. It was found after some troubleshooting, that the radio modem connected to the Arduino was not receiving enough power to pair with the radio modem connected to the laptop, even though the green LED was flashing, indicating it was attempting to pair.

The solution to this problem was to use two separate power sources for the rover, one to power the motors and one to power the Arduino microcontroller and the attached sensors including the radio modem. The power source for the Arduino and sensors was a 5V USB battery while the six 1.2V batteries in series, provided power to the motors. By separating the high-power users on the rover with different power sources, it was possible for all systems to draw enough current to operate.

Another possible solution that would not require two power sources would be to use a transistor amplifier or Darlington pair arrangement as shown in Figure 5.3. This would have eliminated the need for a separate power source, which would be impractical in most situations. In addition, the use of rechargeable NiMH batteries which only provide 1.2V each for a total of 7.2V for six batteries may have had an effect on the power supplied to each pin. The datasheet for the Arduino microcontroller states that providing less than 7V may result in an unstable voltage, which may have been the case if the batteries were running low. In this case the use of non-rechargeable Zinc-Carbon batteries providing 1.5V each for a total of 9V for six batteries, may have been a more stable power source. This was not considered because the expense of continually providing new batteries for testing was outside the project budget.

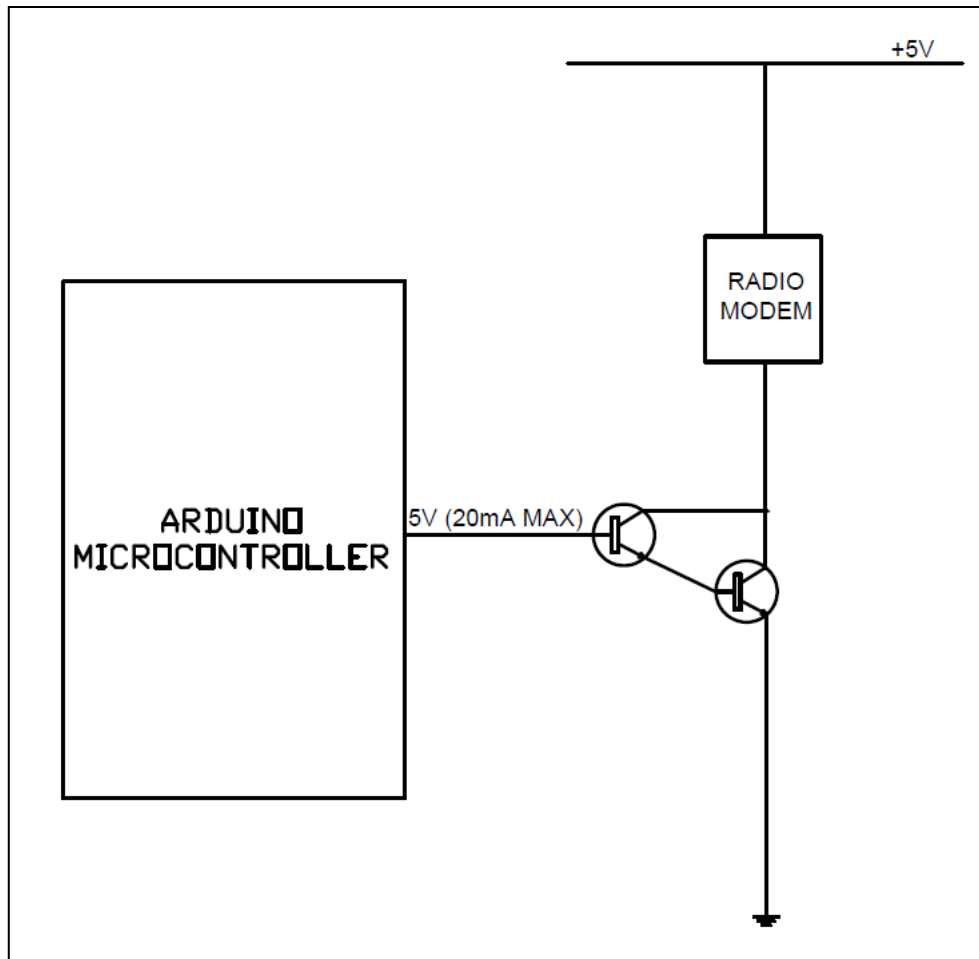


Figure 5.3 – Possible circuit arrangement for the radio modem using a Darlington pair transistor arrangement, to amplify the power source for the radio modem.

5.5.2 Gyroscope

The MPU6050 accelerometer/gyroscope was used to provide very accurate rotations about the z-axis. It was aligned so that the z-axis was vertical, so that all rotation angles were in the horizontal plane. There were two main issues relating to this device that needed to be addressed in the code for it to be effective. Firstly, the drift on the gyro when it is reset is too large for the device to be useful. Secondly, the reference heading is not set, so the initial heading could be any value.

Before data from the gyroscope could be used, the gyroscope had to go through a calibration process, which addressed the gyro drift and the heading. To address the gyro drift, the gyroscope needed to remain stationary for about 20s after resetting. After this time the drift was at a manageable level but still present. The heading could be any value at this time, so the second part of the calibration process was to reset the heading to zero. The effect of this process was to provide the rover with an initial stable heading of 0°.

5.5.3 Troubleshooting

During the rover testing phase, two separate areas were used, one with a carpeted surface and one with a tiled surface. Often preliminary testing was conducted on the carpeted surface before final testing was conducted on the tiled surface. To test the correct functioning of the gyroscope, the testing of the rotation function was conducted on a carpeted surface. Once testing used the overhead camera, a tiled surface was used due to space requirements. However, after travelling only a short distance on the tiled surface, the rover would stop responding to commands and drive until it hit a solid object.

After five weeks of troubleshooting, it was found that the z-axis accelerometer on the MPU6050 was faulty and as soon as any z-axis acceleration was detected the device would crash. One line of code to control the rover used a “while” loop to wait for data from the MPU6050:

```
while (!mpuInterrupt && fifoCount < packetSize);
```

This line of code halted the program until an interrupt was received from the MPU6050 indicating new data was ready. Ordinarily this interrupt would happen many times per second as data was collected by the MPU6050, however if the device crashed no interrupt would be generated. In this case, the rover would remain in its current state indefinitely as the “while” loop prevented the code from advancing. On the carpeted surface, there was no z-axis acceleration so the rover tested proceeded as expected, however on the tiled surface there was a small amount of z-axis acceleration when the rover travelled over a gap between tiles. It was this z-axis acceleration that caused the device to fail on tiles but not on carpet. Once this problem was identified, a new MPU6050 module was purchased to replace the faulty one and the project could proceed.

Three possible causes of this failure were identified. The device was initially powered by a 5V pin from the Arduino microcontroller in error, while the datasheet specifies a power source between 2.38V and 3.46V. After this error was identified the MPU6050 was correctly powered by a 3.3V pin from the Arduino. Another possible cause for the failed unit was rough handling while it was in storage for over a year. The other possible cause was that the device was faulty when shipped and this was not identified. Further work would be required to determine the actual cause of the failure of the device but this was outside the scope of the project.

5.6 Conclusion

Overall the project achieved the main aim, which was to demonstrate a system of autonomous navigation in an unknown environment using images obtained from an overhead camera to guide a test rover. There were some parts of the system that could be improved in future work. From the point of view of the rover control, the PID algorithm could be improved to achieve a more precise response, possibly with the addition of sensors such as a wheel encoder or current measuring from the motors. The machine vision software consistently identified the rover and the target and could accurately determine the heading of the rover. There was some fluctuation in heading calculations due to rounding errors, however this could be reduced by using a higher resolution image. Finally, obstacle detection could be improved by using more appropriate sensors for the environment on Mars such as Lidar or Millimetre Wave Radar. These are all topics that can be considered for future work using this project as a basis.

6 Conclusions

The aim of the project was to build and test a system that could be used for autonomous navigation on Mars. The system was to consist of a rover with images from an overhead camera used to guide it through an environment with impassable obstacles present. The overhead camera in this system would be carried by an airborne platform such as a helicopter or balloon. Operating a robot on Mars has many challenges but for this project the specific challenges that were examined were the signal delay between Earth and Mars (up to 24 minutes), the lack of a magnetic field and the lack of satellite navigation. On Earth, these challenges do not exist because there is a magnetic field to provide a direction reference, a satellite navigation system to provide accurate localisation and if these are not enough to allow autonomous navigation, a person can operate the vehicle either in person or remotely.

The system was built and tested successfully in a controlled environment and with no human input, was able to navigate from any starting position to a target location. The lack of a direction reference, as provided by a magnetic field and satellite navigation, were overcome by using a fixed overhead camera to provide a direction reference and localisation. The direction reference became the orientation of the camera and the localisation was provided by a reference to a target location. By overcoming these challenges, this project shows that fully autonomous operation is possible with the aid of an overhead camera for localisation and navigation.

The rover used a PID controller to follow a path from waypoint to waypoint with an onboard gyroscope used to determine the rotation from a starting point. The overhead camera image was used to determine when the rover had reached the waypoint. It is possible that the control system could be configured to use only the overhead camera for this feedback, rendering the gyroscope redundant. However, a gyroscope would likely remain on the rover for use as a redundant system. Even if the overhead camera was not available, the rover could navigate by dead reckoning temporarily until the camera feed became available again. In this situation the rover would benefit from an odometer or some other method of measuring the distance travelled by the rover. In an actual Mars mission, redundant systems could be used to extend the life of the rover as repair is not possible if one system fails.

The overhead camera used machine vision techniques to identify the rover, its heading and the target. These techniques used colour detection to identify the relevant locations within the camera frame and morphological techniques to reduce the noise from the camera image to identify a point at the

centroid of each colour target. This was a successful method of identifying the important parts of the image, however the testing was run at the same time each day to ensure similar lighting conditions. At times the machine vision algorithm would have trouble identifying the correct colours. A proposed solution to this issue is to use LEDs as identifiers for the rover. In this case, the LEDs would emit the same colour at all times regardless of lighting conditions.

The ultrasonic sensor on the rover used for range finding did perform its function of detecting obstacles, however there were some improvements that could be made to any future project of this nature. The accuracy of the obstacle detection could be improved through the use of different sensors that use a narrow beam for detection rather than the broader cone of the ultrasonic sensor. This would improve the accuracy of the sensors at distances greater than 0.4m, which was the limit of detection set for this project. For this project, the obstacle detection was conducted while the rover was stationary. A further improvement would be to have the rover detect obstacles while moving.

Finally, there were some technical issues encountered during the building and operation of the project. These were related to failed components and unexpected power requirements and while they did not contribute to the overall research goals of the project, these issues did demonstrate the problems that can be encountered when physically building a system for testing rather than simulating the solution to the problem. When simulating the system, it is only possible to assume the model parameters. Therefore, greater confidence in the validity of the results can be achieved by building a system.

The aim of this project was to examine a system for navigation and localisation in an unstructured environment such as that encountered on Mars. A conceptual system consisting of a rover, overhead camera, machine vision algorithm and communications between them was built and tested using two different path-finding algorithms and two different environments. From these tests it was shown that if the system could be applied on Mars it would be a practical solution to the localisation and navigation problem. With the further research that has been identified, the system would be a versatile method of autonomous navigation on Mars which could one day support scientific exploration, resource mining and preparation for human exploration of Mars.

6.1 Further Work

During the course of this project, some opportunities for further work that were outside the scope of the project were identified:

- Improved obstacle detection – The ultrasonic sensors used for this project were suitable for the test environment but would be unsuitable for a practical implementation on Mars. The range and accuracy are limited by the wide cone of detection (15°) and the lack of an atmosphere on Mars would make it impossible to use. Instead a device using a narrow beam such as millimetre wave radar or lidar could provide more accuracy in the location of the obstacle as well as a greater range for obstacle detection. Another aspect of obstacle detection worth investigating is the possibility of obstacle detection while the rover is in motion.
- Path-finding algorithms – The path-finding algorithms tested for this project had mixed results. The A* algorithm was able to consistently calculate a path to the target location if one existed, while the potential field algorithm was unable to calculate a path in the same conditions if it encountered a local minimum. There have been a number of studies conducted previously on path-finding algorithms on Mars. Future research could look at comparing path-finding algorithms using different grid sizes and different obstacle densities, there may be a correlation between the most appropriate conditions for a particular path-finding algorithm. It would be possible to use simulation for this type of study as only the path-finding algorithm would be under test and not the entire system.
- Machine Vision – For this project a fixed camera was used looking directly down on the test area. A more realistic situation on Mars would use a moving camera looking at the area of operation with a range of angles. Further research could examine the same system with a moving overhead camera to be used for localisation and navigation.

7 References

- Arutselvan, K & Vijayakumari, A 2015, 'Assistive Autonomous Ground Vehicles in Smart Grid', *Procedia Technology*, vol. 21, pp. 232-9.
- Balme, MR, Curtis-Rouse, MC, Banham, S, Barnes, D, Barnes, R, Bauer, A, Bedford, CC, Bridges, JC, Butcher, FEG, Caballo-Perucha, P, Caldwell, A, Coates, AJ, Cousins, C, Davis, JM, Dequaire, J, Edwards, P, Fawdon, P, Furuya, K, Gadd, M, Get, P, Griffiths, A, Grindrod, PM, Gunn, M, Gupta, S, Hansen, R, Harris, JK, Hicks, LJ, Holt, J, Huber, B, Huntly, C, Hutchinson, I, Jackson, L, Kay, S, Kyberd, S, Lerman, HN, McHugh, M, McMahon, WJ, Muller, JP, Ortner, T, Osinski, G, Paar, G, Preston, LJ, Schwenzer, SP, Stabbins, R, Tao, Y, Traxler, C, Turner, S, Tyler, L, Venn, S, Walker, H, Wilcox, T, Wright, J & Yeomans, B 2019, 'The 2016 UK Space Agency Mars Utah Rover Field Investigation (MURFI)', *Planetary and Space Science*, vol. 165, pp. 31-56.
- Barfoot, T, Furgale, P, Stenning, B, Carle, P, Thomson, L, Osinski, G, Daly, M & Ghafoor, N 2011, 'Field testing of a rover guidance, navigation, and control architecture to support a ground-ice prospecting mission to Mars', *Robotics and Autonomous Systems*, vol. 59, no. 6, pp. 472-88.
- Bertrand, R & Winnendaal, Mv 2000, 'Mechatronic Aspects of the Nanokhod Micro-Rover for Planetary Surface Exploration', *IFAC Proceedings Volumes*, vol. 33, no. 26, pp. 289-96.
- Brown, D, Cole, S, Webster, G, Agle, DC, Chicoine, RA, Rickman, J, Hoover, R, Mitrofanov, I, Ravine, M, Hassler, D, Cuesta, L, Jones, NN, Barnstorff, K, Faccio, R, Apuzzo, ML & Pagan, VM 2013, 'The Mars science laboratory landing', *World Neurosurgery*, vol. 79, no. 2, pp. 223-42.
- Cook, RA 2005, 'The Mars exploration rover project', *Acta Astronautica*, vol. 57, no. 2-8, pp. 116-20.
- Dabrowski, B & Banaszkiwicz, M 2008, 'Multi-rover navigation on the lunar surface', *Advances in Space Research*, vol. 42, no. 2, pp. 369-78.
- Daniel, K, Nash, A, Koenig, S & Felner, A 2010, 'Theta*: Any-Angle Path Planning on Grids', *Journal of Artificial Intelligence Research*, vol. 39, pp. 533-79.
- Elrayes, A, Ali, MH, Zakaria, A & Ismail, MH 2019, 'Smart airport foreign object debris detection rover using LiDAR technology', *Internet of Things*, vol. 5, pp. 1-11.
- Erickson, JK, Callas, JL & Haldemann, AFC 2007, 'The Mars Exploration Rover Project: 2005 surface operations results', *Acta Astronautica*, vol. 61, no. 7, pp. 699-706.
- Fantino, E, Grassi, M, Pasolini, P, Causa, F, Molfese, C, Aurigemma, R, Cimminiello, N, de la Torre, D, Dell'Aversana, P, Esposito, F, Gramiccia, L, Paudice, F, Punzo, F, Roma, I, Savino, R & Zuppardi, G 2017, 'The Small Mars System', *Acta Astronautica*, vol. 137, pp. 168-81.
- Ferguson, D & Stentz, A 2005, *The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments*, CMU-RI-TR-05-19, Carnegie Mellon University, Pittsburgh, PA.
- Fox, D, Burgard, W & Thrun, S 1998, 'Active Markov localization for mobile robots', *Robotics and Autonomous Systems*, vol. 25, no. 3, pp. 195-207.
- Francis, R, Pilles, E, Osinski, GR, Mclsaac, K, Gaines, D & Kissi, J 2019, 'Utility and applications of rover science autonomy capabilities: Outcomes from a high-fidelity analogue mission simulation', *Planetary and Space Science*.

Gonzalez, R, Chandler, S & Apostolopoulos, D 2019, 'Characterization of machine learning algorithms for slippage estimation in planetary exploration rovers', *Journal of Terramechanics*, vol. 82, pp. 23-34.

Grotzinger, JP, Crisp, J, Vasavada, AR, Anderson, RC, Baker, CJ, Barry, R, Blake, DF, Conrad, P, Edgett, KS, Ferdowski, B, Gellert, R, Gilbert, JB, Golombek, M, Gómez-Elvira, J, Hassler, DM, Jandura, L, Litvak, M, Mahaffy, P, Maki, J, Meyer, M, Malin, MC, Mitrofanov, I, Simmonds, JJ, Vaniman, D, Welch, RV & Wiens, RC 2012, 'Mars Science Laboratory Mission and Science Investigation', *Space Science Reviews*, vol. 170, no. 1, pp. 5-56.

Guan, X, Wang, X, Fang, J & Feng, S 2014, 'An innovative high accuracy autonomous navigation method for the Mars rovers', *Acta Astronautica*, vol. 104, no. 1, pp. 266-75.

Hayati, S, Rankin, A, Kim, W, Leger, P, Castano, R & Ali, K 2007, 'Advanced Robotics Technology Infusion to the Nasa Mars Exploration Rover (Mer) Project', *IFAC Proceedings Volumes*, vol. 40, no. 15, pp. 180-5.

Kading, B & Straub, J 2015, 'Utilizing in-situ resources and 3D printing structures for a manned Mars mission', *Acta Astronautica*, vol. 107, pp. 317-26.

Kerzhanovich, VV, Cutts, JA, Cooper, HW, Hall, JL, McDonald, BA, Pauken, MT, White, CV, Yavrouian, AH, Castano, A, Cathey, HM, Fairbrother, DA, Smith, IS, Shreves, CM, Lachenmeier, T, Rainwater, E & Smith, M 2004, 'Breakthrough in Mars balloon technology', *Advances in Space Research*, vol. 33, no. 10, pp. 1836-41.

Kubota, T, Sato, T & Ejiri, R 2010, 'Experimental Study on Visual Navigation for Exploration Robot', *IFAC Proceedings Volumes*, vol. 43, no. 16, pp. 169-74.

Lu, J, Lei, C, Yang, Y & Liu, M 2017, 'In-motion initial alignment and positioning with INS/CNS/ODO integrated navigation system for lunar rovers', *Advances in Space Research*, vol. 59, no. 12, pp. 3070-9.

Matijevic, J & Shirley, D 1997, 'The mission and operation of the mars pathfinder microrover', *Control Engineering Practice*, vol. 5, no. 6, pp. 827-35.

Miller, DP, Atkinson, DJ, Wilcox, BH & Mishkin, AH 1989, 'Autonomous Navigation and Control of a Mars Rover', *IFAC Proceedings Volumes*, vol. 22, no. 7, pp. 111-4.

Mishra, P, Vajjramatti, H, Rai, A, Mangond, K, Anantharajaiah, N & Kishore, JK 2016, 'Computational architectures for sonar array processing in autonomous rovers', *Microprocessors and Microsystems*, vol. 42, pp. 49-69.

Muñoz, P, R-Moreno, MD & Castaño, B 2017, '3Dana: A path planning algorithm for surface robotics', *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 175-92.

Mutlu, L & Uyar, E 2012, 'Control and Navigation of an Autonomous Mobile Robot with Dynamic Obstacle Detection and Adaptive Path Finding Algorithm', *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 355-60.

Neveu, D, Bilodeau, VS, Alger, M, Moffat, B, Hamel, JF & Lafontaine, Jd 2010, 'Simulation Infrastructure for Autonomous Vision-Based Navigation Technologies', *IFAC Proceedings Volumes*, vol. 43, no. 15, pp. 279-86.

- Ning, X, Gui, M, Zhang, J & Fang, J 2016, 'A dimension reduced INS/VNS integrated navigation method for planetary rovers', *Chinese Journal of Aeronautics*, vol. 29, no. 6, pp. 1695-708.
- O'Neil, WJ & Cazaux, C 2000, 'The mars sample return project', *Acta Astronautica*, vol. 47, no. 2, pp. 453-65.
- Olson, CF, Matthies, LH, Schoppers, M & Maimone, MW 2003, 'Rover navigation using stereo ego-motion', *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 215-29.
- Pedraza, S, Pozo-Ruz, A, Roth, H & Schilling, K 1998, 'Outdoor Navigation of Micro-Rovers', *IFAC Proceedings Volumes*, vol. 31, no. 2, pp. 223-7.
- Pilles, EA, Cross, M, Caudill, CM, Francis, R, Osinski, GR, Newman, J, Battler, M, Bourassa, M, Haltigin, T, Hipkin, V, Kerrigan, M, McLennan, S, Silber, EA & Williford, K 2018, 'Exploring new models for improving planetary rover operations efficiency through the 2016 CanMars Mars Sample Return (MSR) analogue deployment', *Planetary and Space Science*.
- Post, MA, Li, J & Quine, BM 2016, 'Planetary micro-rover operations on Mars using a Bayesian framework for inference and control', *Acta Astronautica*, vol. 120, pp. 295-314.
- Ralphs, M, Franz, B, Baker, T & Howe, S 2015, 'Water extraction on Mars for an expanding human colony', *Life Sciences in Space Research*, vol. 7, pp. 57-60.
- Rand, JL & Phillips, ML 2004, 'An analysis of the deployment of a pumpkin balloon at Mars', *Advances in Space Research*, vol. 33, no. 10, pp. 1812-8.
- Sakai, A, Ingram, D, Dinius, J, Chawla, K, Raffin, A & Paques, A 2018, 'PythonRobotics: a Python code collection of robotics algorithms', *CoRR*, vol. abs/1808.10703.
- Sasiadek, JZ & Green, DN 1997, 'Guidance and Control of Autonomous Planetary Rover', *IFAC Proceedings Volumes*, vol. 30, no. 20, pp. 241-6.
- Shirley, DL 1995, 'Mars Pathfinder Microrover Experiment - A paradigm for very low cost spacecraft', *Acta Astronautica*, vol. 35, pp. 355-65.
- Shishko, R, Fradet, R, Do, S, Saydam, S, Tapia-Cortez, C, Dempster, AG & Coulton, J 2017, 'Mars Colony in situ resource utilization: An integrated architecture and economics model', *Acta Astronautica*, vol. 138, pp. 53-67.
- Spiteri, C, Shaukat, A & Gao, Y 2017, 'Structure augmented monocular saliency for planetary rovers', *Robotics and Autonomous Systems*, vol. 88, pp. 1-10.
- Tao, Y, Muller, J-P & Poole, W 2016, 'Automated localisation of Mars rovers using co-registered HiRISE-CTX-HRSC orthorectified images and wide baseline Navcam orthorectified mosaics', *Icarus*, vol. 280, pp. 139-57.
- Tarokh, M 2008, 'Hybrid intelligent path planning for articulated rovers in rough terrain', *Fuzzy Sets and Systems*, vol. 159, no. 21, pp. 2927-37.
- Walter, V, Novák, T & Saska, M 2018, 'Self-localization of Unmanned Aerial Vehicles Based on Optical Flow in Onboard Camera Images', Springer International Publishing, Cham, pp. 106-32.

Zeno, PJ, Patel, S & Sobh, TM 2017, 'A novel neurophysiological based navigation system', *Biologically Inspired Cognitive Architectures*, vol. 22, pp. 67-81.

Appendix A - Project Specification

ENG4111/ENG4112 Research Project

Project Specification

For: Jason Pont
Title: Autonomous Navigation of a Partially Unknown Environment
Major: Mechatronic Engineering
Supervisor: Craig Lobsey
Enrolment: ENG4111 – EXT S1 2019
ENG4112 – EXT S2 2019

Project Aim: To investigate methods for navigating a structured, partially unknown environment using an autonomous rover in a test scenario.

Programme: Version 1, 16th March 2019

1. Review navigation and path planning algorithms suitable for autonomous rover operation in remote and unknown environments (e.g. on Mars where the environment is unknown and signal delay makes autonomous navigation a necessity).
2. Review and Investigate the use of sensors to navigate around obstacles and localisation in an environment where GPS is unavailable.
3. Investigate methods of externally tracking the rover through the environment.
4. Investigate the use of wireless/Bluetooth/RF or another method of remotely communicating with an autonomous vehicle.
5. Design and build a test rover using the most appropriate combination of navigation methods, sensors, communication and tracking methods.
6. Operate the autonomous rover in at least two test environments with varying levels of obstacle density.

If time and resources permit:

7. Investigate the performance of the test rover in returning to the starting location, analogous to a Mars sample return mission.
8. Investigate different navigation methods to find the most efficient way of navigating to a target location.
9. Investigate the use of different sensors for obstacle detection.

Appendix B - Arduino Code

```

// Code to run the autonomous rover for research project 2019
// Jason Pont 29/4/2019

/* Uses data from the video to set a target direction then rotate
to that direction using the
* gyro. The rover will then drive to the target and stop at the
target. Includes obstacle detection, i.e if
* an obstacle is detected, the location will be transmitted to
the CPU and a new path calculated
*/

// Include NewPing library for operation of ultrasound module

#include <NewPing.h>

// Include Servo library for operating servo

#include <Servo.h>

// include the I2Cdev and MPU6050_6Axis_MotionApps20 header files
// for communication with the gyroscope

#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"

// Define the pins used

#define pinLB 3 // define pin3 as left motor back
connect with IN1
#define pinLF 5 // define pin5 as left motor forward
connect with IN2
#define pinRB 6 // define pin6 as right motor back
connect with IN3
#define pinRF 11 // define pin11 as right motor
forward connect with IN4
#define ECHO A0 // define ultrasonic receive pin
(Echo)
#define TRIG A1 // define ultrasonic send pin(Trig)
#define INTERRUPT_PIN 2 // pin used for interrupts
(gyroscope)
#define LED_PIN 13

// Define state variables

#define ROTATION 0
#define DRIVE 1
#define SCAN 2
#define WAIT 3

// Set up ultrasound sensor and servo with a maximum range of 100cm

NewPing Sonar(TRIG, ECHO, 100);
int Sonar_ahead = 78; //Servo angle when sonar is
pointing directly ahead

```

```

float Sonar_Range = 0;           //Distance to object picked up
by sonar
int Sonar_angle = 78;           //Direction to object, angle of
sonar
long sonar_time = 0;            // used to make sure no sonar
ping is sent within 60ms of the previous one
bool Sonar_read = false;        //Flag set when sonar reading
taken
Servo myservo;                  // new myservo

// MPU variables

MPU6050 mpu;
Quaternion q;                   // [w, x, y, z]           quaternion container
VectorFloat gravity;            // [x, y, z]           gravity vector
float ypr[3];                   // [yaw, pitch, roll] yaw/pitch/roll
container and gravity vector
float yaw = 0;
float yaw_calibration = 0; //resets the yaw to zero after gyro
stabilises
uint16_t fifoCount;             // count of all bytes currently in FIFO
uint16_t packetSize;           // expected DMP packet size (default is 42
bytes)
uint8_t fifoBuffer[64];        // FIFO storage buffer
bool dmpReady = false;         // set true if DMP init was successful
bool setup_complete = false;   // set true if the initial setup is
complete

uint8_t mpuIntStatus;          // holds actual interrupt status byte from
MPU

volatile bool mpuInterrupt = false; // indicates whether MPU
interrupt pin has gone high

// PID variables

float error = 0;
float previous_error = 0;
float pid_p = 0;
float pid_i = 0;
float pid_d = 0;
float PID = 0;
float constrained_PID = 0;
float kp = 30;
float ki = 3;
float kd = 5;
float start_time = 0;
float loop_time = 0;
float loop_previous = 0;
float right_bias = 0.8; //variable to take account of the higher
power of the right motor
float PIDL = 0;
float PIDR = 0;
float q0 = 0;
float q1 = 0;
float q2 = 0;

```

```

long timer = 0;

bool direction_set = false;
long current_time = 0;

String data = "\n"; //end of line for serial data
int count = 0;

// set up a buffer to receive serial data
const byte numChars = 32;
char RxBuffer[numChars]= {'0'};
float angleError = 0;
int distanceError = 0;
boolean newData = false;
char *distanceRX = "";
int previousDistance = 0;

// state variable
int state = WAIT;

// set up the delimiter
char delim[] = "/";

////////////////////////////////////
//      SETUP ROUTINE      //
////////////////////////////////////

void setup() {
  digitalWrite(LED_PIN,HIGH);
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(57600);
  myservo.attach(9) ; // define the servo pin(PWM)
  myservo.write(Sonar_angle);
  Gyro_initialise();

  // Set pin inputs and outputs for motors and ultrasonic range
finder
  pinMode(pinLB,OUTPUT);
  pinMode(pinLF,OUTPUT);
  pinMode(pinRB,OUTPUT);
  pinMode(pinRF,OUTPUT);
  pinMode(ECHO, INPUT);
  pinMode(TRIG, OUTPUT);

  // allow two minutes for gyro stabilisation and obtain stable
orientation for calibration
  for (int x =0; x < 2000; x++){
    while (!mpuInterrupt && fifoCount < packetSize);
    yaw = get_direction();
    delay(1);
    if (x % 100 == 0){
      Serial.println(x/100);

```

```

    }
}

yaw_calibration = yaw;
setup_complete = true;
Serial.print("yaw calibration value = ");
Serial.println(yaw_calibration);
digitalWrite(LED_PIN,LOW);
timer = micros();

}

////////////////////////////////////
//      MAIN LOOP ROUTINE      //
////////////////////////////////////

void loop() {
  recvWithStartEndMarkers();

  switch (state) {
    // When the rover is rotating, constantly retrieve the reference
    // angle from machine vision software
    case ROTATION:
      if (!direction_set){
        if (newData){
          if (atoi(RxBuffer) == 500){
            rover_stop();
            Sonar_angle = Sonar_ahead - 60;
            myservo.write(Sonar_angle);
            sonar_time = millis();
            state = SCAN;
          } else if (atoi(RxBuffer) == 600){
            state = WAIT;
          } else
            yaw_calibration += angleError;
        }
        if (yaw_calibration >= 360){
          yaw_calibration -= 360;
        }
        if (yaw_calibration <= -360){
          yaw_calibration += 360;
        }
      }
      direction_set = true;
    } else {
      rotate();
      if (abs(error) <= 1.0){
        direction_set = false;
        rover_stop();
        state = DRIVE;
      }
    }
  }

  break;
}

```

```

    // When in DRIVE state, rover will correct course using gyro
case DRIVE:
    forward(70);
    if (newData){
        if (atoi(RxBuffer) == 500){
            rover_stop();
            Sonar_angle = Sonar_ahead - 60;
            myservo.write(Sonar_angle);
            sonar_time = millis();
            state = SCAN;
        } else if (atoi(RxBuffer) == 600){
            state = WAIT;
        } else if (distanceError <=10){
            rover_stop();
            //state = ROTATION;
        } else if (tan(abs(angleError)*PI/180) >=
5.0/distanceError){
            rover_stop();
            state = ROTATION;
        } else if (previousDistance < distanceError){
            rover_stop();
            state = ROTATION;
        }
    }

    previousDistance = distanceError;
    break;
    // When in SCAN state rover will stop and scan using ultrasonic
sensor
case SCAN:
    rover_stop();
    if (Sonar_angle <= Sonar_ahead + 60){

        if (millis() > sonar_time + 500 && !Sonar_read){
            Sonar_Range = (Sonar.ping_cm())/100.0;
            Sonar_read = true;
            if (Sonar_Range > 0.0 && Sonar_Range <=0.4){
                Serial.print("/");
                Serial.print("S");
                Serial.print("/");
                Serial.print(Sonar_angle - Sonar_ahead);
                Serial.print("/");
                Serial.print(Sonar_Range);
                Serial.println("/");
            }

        } else if (millis() > sonar_time + 1000 && Sonar_read){
            sonar_time = millis();
            Sonar_angle +=15;
            myservo.write(Sonar_angle);
            Sonar_read = false;
        }

    } else{
        Sonar_angle = Sonar_ahead;
    }

```

```

        Sonar_Range = 0;
        myservo.write(Sonar_angle);
        state = ROTATION;
    }
    break;
    // When set to WAIT, rover will stop and wait for further
instructions
    case WAIT:
        rover_stop();
        if (newData){
            state = ROTATION;
        }
        break;
    default:

        break;
}

ShowNewData();
// loop timer for PID controller
current_time = millis();
loop_previous = start_time;
start_time = micros();
loop_time = (start_time - loop_previous) / 1000000.0;

while (!mpuInterrupt && fifoCount < packetSize); //get rover
rotation
yaw = get_direction();

// Adjust yaw values so that only values between -180 and 180 are
obtained
if (yaw < -180) {
    yaw = yaw + 360;
}
if (yaw > 180) {
    yaw = yaw - 360;
}

// Calculate PID values -ve for turning left and +ve for turning
right
error = yaw;
pid_p = kp * error;
pid_i = pid_i + ki * (error-(pid_i/200))*loop_time;
pid_d = kd * ((error - previous_error)/loop_time);
PID = pid_p + pid_i + pid_d;
PID = constrain(PID, -130, 130);

previous_error = error;
}

////////////////////////////////////
//      SETUP THE GYROSCOPE      //
////////////////////////////////////
// Code largely taken from the I2CDev library with slight
modifications

```

```

void Gyro_initialise(void){

    uint8_t devStatus;        // return status after each device
operation (0 = success, !=0 = error)

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection
successful") : F("MPU6050 connection failed"));

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(112);
    mpu.setYGyroOffset(36);
    mpu.setZGyroOffset(28);
    mpu.setZAccelOffset(1468);

    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows
it's okay to use it
        Serial.println(F("DMP ready! Waiting for first
interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPageSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }
}
}

```



```

////////////////////////////////////
//   GET THE DIRECTION (HEADING) OF THE GYROSCOPE   //
////////////////////////////////////

// Code largely taken from the I2CDev library with slight
modifications
float get_direction(){

    mpuInterrupt = false;
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // reset interrupt flag and get INT_STATUS byte

    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();
    if (fifoCount % packetSize !=0){ // check for corrupt packet
        mpu.resetFIFO();
        Serial.println(F("Corrupt Packet!"));
        return yaw;
    }
    else {
        // check for overflow (this should never happen unless our
code is too inefficient)
        if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
            // reset so we can continue cleanly
            mpu.resetFIFO();
            Serial.println(F("FIFO overflow!"));

            // otherwise, check for DMP data ready interrupt (this
should happen frequently)
            } else if (mpuIntStatus & 0x02) {
                // wait for correct available data length, should be a
VERY short wait
                while (fifoCount < packetSize) fifoCount =
mpu.getFIFOCount();

                // read a packet from FIFO
                mpu.getFIFOBytes(fifoBuffer, packetSize);
                // reset FIFO
                mpu.resetFIFO();

                // track FIFO count here in case there is > 1 packet
available
                // (this lets us immediately read more without waiting
for an interrupt)
                fifoCount -= packetSize;

                // retrieve the yaw, pitch, roll values from the DMP
                mpu.dmpGetQuaternion(&q, fifoBuffer);
                mpu.dmpGetGravity(&gravity, &q);
                mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
                //Serial.print("ypr\t");
                //Serial.print(ypr[0] * 180/M_PI);

```

```

        //Serial.print("\t");
        //Serial.print(ypr[1] * 180/M_PI);
        //Serial.print("\t");
        //Serial.println(ypr[2] * 180/M_PI);
        if (setup_complete){
            return (ypr[0] * 180/M_PI) - yaw_calibration;
        } else
            return (ypr[0] * 180/M_PI);
    }
}

////////////////////////////////////
//      FORWARD MOVEMENT      //
////////////////////////////////////

void forward(int rate) {
    int forward_left = 0; //rate for forward left movement
    int forward_right = 0; //rate for forward right movement

    PIDL = constrain(rate - constrained_PID,-255,255);
    PIDR = constrain(rate + constrained_PID * right_bias,-255,255);
    if (PIDL < 0){
        PIDL = -PIDL;
        analogWrite(pinLB,PIDL);
        digitalWrite(pinLF,LOW);
    } else {
        digitalWrite(pinLB,LOW);
        analogWrite(pinLF,PIDL);
    }

    if (PIDR < 0){
        PIDR = -PIDR;
        analogWrite(pinRB,PIDR);
        digitalWrite(pinRF,LOW);
    } else {
        digitalWrite(pinRB,LOW);
        analogWrite(pinRF,PIDR);
    }
}

////////////////////////////////////
//      INTERRUPT DETECTION ROUTINE      //
////////////////////////////////////

void dmpDataReady() {
    mpuInterrupt = true;
}

////////////////////////////////////
//      STOP THE ROVER      //
////////////////////////////////////

void rover_stop() {
    digitalWrite(pinRF,LOW);
}

```

```

digitalWrite(pinRB, LOW);
digitalWrite(pinLF, LOW);
digitalWrite(pinLB, LOW);

}

////////////////////////////////////
//      ROTATE      //
////////////////////////////////////
void rotate(){

    if (PID<0){
        analogWrite(pinLF, -PID);
        digitalWrite(pinLB, LOW);
        digitalWrite(pinRF, LOW);
        analogWrite(pinRB, -PID);
    }
    else {
        analogWrite(pinRF, PID);
        digitalWrite(pinRB, LOW);
        digitalWrite(pinLF, LOW);
        analogWrite(pinLB, PID);
    }
}

////////////////////////////////////
//      RECEIVE DATA FROM PC      //
////////////////////////////////////

void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                RxBuffer[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                RxBuffer[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
                if (atoi(RxBuffer) < 500 ){
                    angleError = atof(strtok(RxBuffer, delim));
                    distanceError = atoi(strtok(NULL, delim));
                }
            }
        }
    }
}

```

```

        }
    }
}

else if (rc == startMarker) {
    recvInProgress = true;
}
}

}
////////////////////////////////////
//      DISPLAY RECEIVED SERIAL DATA      //
////////////////////////////////////

// Largely used for debugging

void ShowNewData() {
    count++;
    if (newData == true) {
        Serial.print("Rover received ... ");
        Serial.println(RxBuffer);

        newData = false;
    }
    if (count == 10){
        Serial.print("  Error= ");
        Serial.print(error);
        Serial.print("  PID Integral= ");
        Serial.print(pid_i);
        Serial.print("  Yaw= ");
        Serial.print(yaw);
        Serial.print("  Yaw_calibration= ");
        Serial.print(yaw_calibration);
        Serial.print("  STATE= ");
        Serial.println(state);
        count = 0;
    }
}
}

```

Appendix C - Machine Vision Code – Python

```

# Machine vision code for research project
# Jason Pont 2019

import cv2
import math
import numpy as np
import serial
import queue
import threading
import matplotlib.pyplot as plt

show_animation = False
path_set = False

serial_queue = queue.Queue(10000)

# serial read function (read serial data and place in queue)
def serial_read_thread(s):
    while True:
        line = s.readline() # read from serial
        serial_queue.put(line) # place in queue

ser = serial.Serial('COM4', 57600) # select correct port and baud

# start the serial thread
thread_serial = threading.Thread(target=serial_read_thread,
args=(ser,)).start()

cap = cv2.VideoCapture(0)
kernel = np.ones((2,2),np.uint8)
kernel2 = np.ones((3,3),np.uint8)
kernel3 = np.ones((5,5),np.uint8)
kernel4 = np.ones((10,10),np.uint8)

# Default resolutions of the frame are obtained.The default
resolutions are system dependent.
# We convert the resolutions from float to integer.
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
test_area_width = 2.36
test_area_height = 1.75

# Define the codec and create VideoWriter object.The output is
stored in 'outpy.avi' file.
out =
cv2.VideoWriter('rover_test1.avi',cv2.VideoWriter_fourcc('M','J','P'
,'G'), 25, (frame_width,frame_height))

# Set the colour centroids to 0,0 to start with
yellow_cX, yellow_cY = 0 ,0
orange_cX, orange_cY = 0, 0
green_cX, green_cY = 0, 0

```

```

# A star path-finding algorithm as made available by Sakai et al.
(2018)
class AStarPlanner:

    def __init__(self, ox, oy, reso, rr):
        """
        Initialize grid map for a star planning

        ox: x position list of Obstacles [m]
        oy: y position list of Obstacles [m]
        reso: grid resolution [m]
        rr: robot radius[m]
        """

        self.reso = reso
        self.rr = rr
        self.calc_obstacle_map(ox, oy)
        self.motion = self.get_motion_model()

    class Node:
        def __init__(self, x, y, cost, pind):
            self.x = x # index of grid
            self.y = y # index of grid
            self.cost = cost
            self.pind = pind

        def __str__(self):
            return str(self.x) + "," + str(self.y) + "," +
str(self.cost) + "," + str(self.pind)

    def planning(self, sx, sy, gx, gy):
        """
        A star path search

        input:
            sx: start x position [m]
            sy: start y position [m]
            gx: goal x position [m]
            gy: goal y position [m]

        output:
            rx: x position list of the final path
            ry: y position list of the final path
        """

        nstart = self.Node(self.calc_xyindex(sx, self.minx),
                           self.calc_xyindex(sy, self.miny), 0.0, -1)
1) ngoal = self.Node(self.calc_xyindex(gx, self.maxx),
                     self.calc_xyindex(gy, self.maxy), 0.0, -1)

        open_set, closed_set = dict(), dict()
        open_set[self.calc_grid_index(nstart)] = nstart

        while 1:
            if len(open_set) == 0:

```

```

        print("Open set is empty..")
        break

        c_id = min(
            open_set, key=lambda o: open_set[o].cost +
self.calc_heuristic(ngoal, open_set[o]))
        current = open_set[c_id]

        # show graph
        if show_animation: # pragma: no cover
            plt.plot(self.calc_grid_position(current.x,
self.minx),
                    self.calc_grid_position(current.y,
self.miny), "xc")
            #if len(closed_set.keys()) % 10 == 0:
            #    plt.pause(0.0001)

        if current.x == ngoal.x and current.y == ngoal.y:
            print("Find goal")
            ngoal.pind = current.pind
            ngoal.cost = current.cost
            break

        # Remove the item from the open set
        del open_set[c_id]

        # Add it to the closed set
        closed_set[c_id] = current

        # expand_grid search grid based on motion model
        for i, _ in enumerate(self.motion):
            node = self.Node(current.x + self.motion[i][0],
                            current.y + self.motion[i][1],
                            current.cost + self.motion[i][2],
c_id)

            n_id = self.calc_grid_index(node)

            # If the node is not safe, do nothing
            if not self.verify_node(node):
                continue

            if n_id in closed_set:
                continue

            if n_id not in open_set:
                open_set[n_id] = node # discovered a new node
            else:
                if open_set[n_id].cost > node.cost:
                    # This path is the best until now. record it
                    open_set[n_id] = node

        rx, ry = self.calc_final_path(ngoal, closed_set)

        return rx, ry

```



```

def calc_final_path(self, ngoal, closedset):
    # generate final course
    rx, ry = [self.calc_grid_position(ngoal.x, self.minx)], [
        self.calc_grid_position(ngoal.y, self.miny)]
    pind = ngoal.pind
    while pind != -1:
        n = closedset[pind]
        rx.append(self.calc_grid_position(n.x, self.minx))
        ry.append(self.calc_grid_position(n.y, self.miny))
        pind = n.pind

    return rx, ry

@staticmethod
def calc_heuristic(n1, n2):
    w = 1.0 # weight of heuristic
    d = w * math.sqrt((n1.x - n2.x) ** 2 + (n1.y - n2.y) ** 2)
    return d

def calc_grid_position(self, index, minp):
    """
    calc grid position

    :param index:
    :param minp:
    :return:
    """
    pos = index * self.reso + minp
    return pos

def calc_xyindex(self, position, min_pos):
    return round((position - min_pos) / self.reso)

def calc_grid_index(self, node):
    return (node.y - self.miny) * self.xwidth + (node.x -
self.minx)

def verify_node(self, node):
    px = self.calc_grid_position(node.x, self.minx)
    py = self.calc_grid_position(node.y, self.miny)

    if px < self.minx:
        return False
    elif py < self.miny:
        return False
    elif px >= self.maxx:
        return False
    elif py >= self.maxy:
        return False

    # collision check
    if self.obmap[node.x][node.y]:
        return False

    return True

```

```

def calc_obstacle_map(self, ox, oy):

    self.minx = round((min(ox)-self.reso)/self.reso)*self.reso
    self.miny = round((min(oy)-self.reso)/self.reso)*self.reso
    self.maxx = round((max(ox)+self.reso)/self.reso)*self.reso
    self.maxy = round((max(oy)+self.reso)/self.reso)*self.reso
    print("minx: %0.2f" %(self.minx))
    print("miny: %0.2f" %(self.miny))
    print("maxx: %0.2f" %(self.maxx))
    print("maxy: %0.2f" %(self.maxy))

    self.xwidth = round((self.maxx - self.minx) / self.reso)
    self.ywidth = round((self.maxy - self.miny) / self.reso)
    print("xwidth:", self.xwidth)
    print("ywidth:", self.ywidth)

    # obstacle map generation
    self.obmap = [[False for i in range(self.ywidth)]
                  for i in range(self.xwidth)]
    for ix in range(self.xwidth):
        x = self.calc_grid_position(ix, self.minx)
        for iy in range(self.ywidth):
            y = self.calc_grid_position(iy, self.miny)
            for iox, ioy in zip(ox, oy):
                d = math.sqrt((iox - x) ** 2 + (ioy - y) ** 2)
                if d <= self.rr:
                    self.obmap[ix][iy] = True
                    break

    @staticmethod
    def get_motion_model():
        # dx, dy, cost
        motion = [[1, 0, 1],
                  [0, 1, 1],
                  [-1, 0, 1],
                  [0, -1, 1],
                  [-1, -1, math.sqrt(2)],
                  [-1, 1, math.sqrt(2)],
                  [1, -1, math.sqrt(2)],
                  [1, 1, math.sqrt(2)],
                  [1,2, math.sqrt(5)],
                  [2,1, math.sqrt(5)],
                  [2,-1, math.sqrt(5)],
                  [1,-2, math.sqrt(5)],
                  [-1,-2, math.sqrt(5)],
                  [-2,-1, math.sqrt(5)],
                  [-2,1, math.sqrt(5)],
                  [-1,2, math.sqrt(5)]]

        return motion

def frange(start, stop, step):
    i = start
    while i < stop:
        yield i
        i += step

```

```

print(__file__ + " start!!")

grid_size = 0.03 # [m]
robot_radius = 0.12 # [m]

# set obstacle positions around the outside of the test area
ox, oy = [], []
for i in frange(0, 2.36, grid_size):
    ox.append(i)
    oy.append(0)
    ox.append(i)
    oy.append(1.75)
for i in frange(0, 1.75, grid_size):
    ox.append(0)
    oy.append(i)
    ox.append(2.36)
    oy.append(i)

sonar_angle = 45
sonar_distance = 0.64

if show_animation: # pragma: no cover
    plt.plot(ox, oy, ".k")
    plt.plot(sx, sy, "og")
    plt.plot(gx, gy, "xb")
    plt.grid(True)
    plt.axis("equal")

current_wp = 0
waypoint_x, waypoint_y = [], []
ox_display, oy_display = [], []

while(True):
    #print("current waypoint ", current_wp)
    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of colours in HSV
    lower_yellow = np.array([25,40,50])
    upper_yellow = np.array([35,255,255])
    lower_orange = np.array([5,65,50])
    upper_orange = np.array([24,255,255])
    lower_green = np.array([60,50,50])
    upper_green = np.array([80,255,255])

    # Threshold the HSV image to get only yellow, orange, green
    colors
    mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
    mask_orange = cv2.inRange(hsv, lower_orange, upper_orange)

```

```

mask_green = cv2.inRange(hsv, lower_green, upper_green)

# Bitwise-AND mask and original image
res_yellow = cv2.bitwise_and(frame,frame, mask= mask_yellow)
res_orange = cv2.bitwise_and(frame,frame, mask= mask_orange)
res_green = cv2.bitwise_and(frame,frame, mask= mask_green)

# Use morphology to clean up the three images
opening_yellow = cv2.morphologyEx(res_yellow, cv2.MORPH_OPEN,
kernel4)
yellow_final = cv2.dilate(opening_yellow, kernel2, iterations = 2)
opening_orange = cv2.morphologyEx(res_orange, cv2.MORPH_OPEN,
kernel3)
orange_final = cv2.dilate(opening_orange, kernel2, iterations = 2)
opening_green = cv2.morphologyEx(res_green, cv2.MORPH_OPEN,
kernel2)
green_final = cv2.dilate(opening_green, kernel2, iterations = 2)

# GREEN PROCESSING
# convert green image to binary image
gray_image = cv2.cvtColor(green_final, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray_image, 63, 255, 0)

# calculate moments of binary green image
M = cv2.moments(thresh)

# calculate x,y coordinate of center of green marker
if M["m00"] != 0:
    green_cX = int(M["m10"] / M["m00"])
    green_cY = int(M["m01"] / M["m00"])

# put text and highlight the center
cv2.circle(green_final, (green_cX, green_cY), 5, (255, 255,
255), -1)
cv2.putText(green_final, str(green_cX) + "," + str(green_cY),
(green_cX - 25, green_cY - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 2)

# ORANGE PROCESSING
# convert orange image to binary image
gray_image = cv2.cvtColor(orange_final, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray_image, 63, 255, 0)

# calculate moments of binary orange image
M = cv2.moments(thresh)

# calculate x,y coordinate of center of orange marker
if M["m00"] != 0:
    orange_cX = int(M["m10"] / M["m00"])
    orange_cY = int(M["m01"] / M["m00"])

# put text and highlight the center
cv2.circle(orange_final, (orange_cX, orange_cY), 5, (255, 255,
255), -1)

```

```

    cv2.putText(orange_final, str(orange_cX) + "," + str(orange_cY)
, (orange_cX - 25, orange_cY - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255, 255), 2)

# YELLOW PROCESSING
# convert yellow image to binary image
gray_image = cv2.cvtColor(yellow_final, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray_image, 63, 255, 0)

# calculate moments of binary yellow image
M = cv2.moments(thresh)

# calculate x,y coordinate of center of yellow marker
if M["m00"] != 0:
    yellow_cX = int(M["m10"] / M["m00"])
    yellow_cY = int(M["m01"] / M["m00"])

# put text and highlight the center
cv2.circle(yellow_final, (yellow_cX, yellow_cY), 5, (255, 255,
255), -1)
cv2.putText(yellow_final, str(yellow_cX) + "," +
str(yellow_cY), (yellow_cX - 25, yellow_cY -
25), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

# Add the three images together
res = cv2.add(yellow_final, orange_final)
res = cv2.add(res, green_final)

# Draw line on image representing orientation of the rover
cv2.line(res, (orange_cX, orange_cY), (green_cX, green_cY), (255,
255, 255), 1)

# Calculate rover centroid
rover_cX = int(green_cX + 0.92*(orange_cX - green_cX))
rover_cY = int(green_cY + 0.92*(orange_cY - green_cY))

# Use A* path finding algorithm to plot path
if (path_set == False or len(rx) <= 1):
    sx = rover_cX * test_area_width / frame_width
    sy = (frame_height - rover_cY) * test_area_height /
frame_height
    gx = yellow_cX * test_area_width / frame_width
    gy = (frame_height - yellow_cY) * test_area_height /
frame_height
    a_star = AStarPlanner(ox, oy, grid_size, robot_radius)
    rx, ry = a_star.planning(sx, sy, gx, gy)
    if show_animation: # pragma: no cover
        plt.plot(rx, ry, "-r")
        plt.show()

    rx.reverse()
    ry.reverse()
    print (rx, ry)

waypoint_x, waypoint_y = [], []

```

```

        for i in range(0, len(rx)-1):
            # convert waypoints into pixel locations for calculating
            angles and distances
            waypoint_x.append (int(rx[i]*frame_width /2.36))
            waypoint_y.append (frame_height - int(ry[i]*
            frame_height / 1.75))
            print (waypoint_x[i], waypoint_y[i])

        for i in range(0, len(ox)-1):
            # convert obstacles into pixel locations for display on
            screen
            ox_display.append (int(ox[i]*frame_width /2.36))
            oy_display.append (frame_height - int(oy[i]*
            frame_height / 1.75))

        path_set = True
        current_wp = 0

        if (current_wp > len(waypoint_y)-1):
            path_set = False

        # Calculate angle between two centroids on the rover for the
        rover heading
        rover_heading = 0
        if (orange_cY - green_cY) != 0:

            if (orange_cX >= green_cX and orange_cY < green_cY):
                rover_heading = math.degrees(math.atan((orange_cX -
                green_cX)/(green_cY-orange_cY)))

            if (orange_cX <= green_cX and orange_cY > green_cY):
                rover_heading = 180 + math.degrees(math.atan((green_cX -
                orange_cX)/(orange_cY-green_cY)))

            if (orange_cX - green_cX) != 0:
                if (orange_cX > green_cX and orange_cY >= green_cY):
                    rover_heading = 90 + math.degrees(math.atan((orange_cY -
                    green_cY)/(orange_cX-green_cX)))

                if (orange_cX < green_cX and orange_cY <= green_cY):
                    rover_heading = 270 + math.degrees(math.atan((orange_cY
                    - green_cY)/(orange_cX-green_cX)))

            heading_text = "%4.1f" % rover_heading
            cv2.putText(res, heading_text, (rover_cX,
            rover_cY),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

        # Calculate angle between rover centroid and target
        target_heading = 0
        if (current_wp <= len(waypoint_y)-1):

            if (waypoint_y[current_wp] - rover_cY) != 0:

                if (waypoint_x[current_wp] >= rover_cX and
                waypoint_y[current_wp] < rover_cY):

```

```

        target_heading =
math.degrees(math.atan((waypoint_x[current_wp] -
rover_cX)/(rover_cY-waypoint_y[current_wp])))

        if (waypoint_x[current_wp] <= rover_cX and
waypoint_y[current_wp] > rover_cY):
            target_heading = 180 +
math.degrees(math.atan((rover_cX -
waypoint_x[current_wp])/(waypoint_y[current_wp]-rover_cY)))

        if (waypoint_x[current_wp] - rover_cX) != 0:
            if (waypoint_x[current_wp] > rover_cX and
waypoint_y[current_wp] >= rover_cY):
                target_heading = 90 +
math.degrees(math.atan((waypoint_y[current_wp] -
rover_cY)/(waypoint_x[current_wp]-rover_cX)))

            if (waypoint_x[current_wp] < rover_cX and
waypoint_y[current_wp] <= rover_cY):
                target_heading = 270 +
math.degrees(math.atan((waypoint_y[current_wp] -
rover_cY)/(waypoint_x[current_wp]-rover_cX)))

        # Calculate heading error for transmission to rover
heading_error = target_heading - rover_heading
if (heading_error > 180):
    heading_error = heading_error - 360
if (heading_error < -180):
    heading_error = heading_error + 360

        # Calculate distance error for transmission to rover, distance
is calculated as pixels on screen
distance_error = 0
if (current_wp <= len(waypoint_y)-1):
    #print (rover_cX, ", ", waypoint_x[current_wp], ", ",
rover_cY, ", ", waypoint_y[current_wp])
    distance_error = int(math.sqrt((rover_cX -
waypoint_x[current_wp])** 2 + (rover_cY - waypoint_y[current_wp])
** 2))

        # Send angle and distance error to rover
rover_data = "<%3.1f/%3d>" % (heading_error, distance_error)
print(rover_data.encode("ascii"))
ser.write(rover_data.encode("ascii"))

        # is there anything in the queue? If so process all messages
waiting
while not serial_queue.empty():
    try:
        linedata = str(serial_queue.get(True, 1))
        print(linedata)
        splitline = linedata.split('/')
        #process this data
        if str(splitline[1]) == "S":
            print(linedata)

```

```

        sonar_angle = float(splitline[2])
        print(sonar_angle)
        sonar_distance = float(splitline[3])
        print(sonar_distance)

    except: print('Queue Empty or Error')

# Move on to next waypoint
if (distance_error < 10):
    if (current_wp < len(rx)):
        current_wp = current_wp+1

# Display heading error and a line from rover to target
heading_text = "%4.1f" % heading_error
if (current_wp <= len(waypoint_y)-1):
    cv2.putText(res, heading_text, (waypoint_x[current_wp]-25,
waypoint_y[current_wp]+25),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
255), 2)
    cv2.line(res, (rover_cX, rover_cY), (waypoint_x[current_wp],
waypoint_y[current_wp]), (0, 0, 255),1)

    # Plot path of rover
    for i in range(0, len(rx)-2):
        cv2.line(res, (waypoint_x[i], waypoint_y[i]),
(waypoint_x[i+1], waypoint_y[i+1]), (219,70,255), 2)

# Plot obstacles on final image
for i in range(0, len(ox)-1):
    cv2.rectangle(res, (ox_display[i]+3, oy_display[i]+3),
(ox_display[i]-3, oy_display[i]-3), (255, 255, 0), -1)

# Write the frame into the file 'rover_testx.avi'
complete = cv2.add(res,frame)
out.write(complete)

cv2.imshow('frame',frame)
cv2.imshow('yellow',yellow_final)
cv2.imshow('orange',orange_final)
cv2.imshow('green',green_final)
cv2.imshow('res',res)
k = cv2.waitKey(5) & 0xFF
if k == 27:
    break

cap.release()
out.release()
cv2.destroyAllWindows()
ser.close

```