Real Time Control of a Weed Removal Rover Through Visual Servoing

Tzvi Filler

A Thesis
In the Department of
Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Mechanical Engineering)
at Concordia University
Montréal, Québec, Canada

April 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared,

By: Tzvi Filler

Entitled: Real Time Control of a Weed Removal Rover Through Visual Servoing

And submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Brandon Gordon	Chair
Dr. Glenn Cowan	External Examiner
Dr. Brandon Gordon	Examiner
Dr. Wen-Fang Xie	Thesis Supervisor

Approved by: Sivakumar Narayanswamy
Graduate Program Director

13−*May*−2024
Date

Dr.Mourad Debbabi
Dean,Faculty of Engineering and Computer Science

ABSTRACT

Real Time Control of a Weed Removal Rover Through Visual Servoing

Tzvi Filler

The gardening industry is a multi-billion-dollar enterprise, and currently lacks an autonomous system for removing weeds from an arbitrary garden. The Mobile Weed Remover (MWR) project in the Advanced Mechatronics and Robotics Lab at Concordia seeks to fill this void.

In this thesis, a great amount of re-design work has been carried out to improve the design of both mechanical and electrical systems of the second generation of MWR to make it functional. The mathematical models of MWR are built including the “ideal modeling” and “discrete modeling”. The simulation of the built models is carried out in Matlab/Simulink to investigate the dynamics of the robot. The control algorithm is designed including forward-backward position correction and error direction and handling. The designed controller is integrated with improved MWR robots and the real time control system is implemented in the on-board computer of the robots. The designed robot uses a camera in order to identify, seek out, and destroy weeds, which enables it to operate in an arbitrary garden. In order to target the roots of the weed, the MWR requires a positioning system that can guide the rover to the desired position, which should be able to take the cues from the visual system. The research in this thesis seeks to adapt a control algorithm to the MWR and enable it to approach a target.

Simulation was conducted to validate the effectiveness of the developed control algorithm. A simple mono-chrome target was used to represent the dandelion in the real time control tests. The experimental results further verify that the control system was operational and deliver satisfactory results.

Acknowledgements

This research was conduced in the laboratory of Dr. Wen Fang Xie. Without her guidance and support this research would not have been possible. I would like to thank her for the countless hours she spent guiding me.

Additionally, I would like to thank Ibrahim Babiker and Ehsan Zakari, whose help was most appreciated.

I would also like to thank Taylor Cooney and the staff at the Concordia Library Technology Sandbox, whose aid and advice on matters ranging from 3D printing to network management were indispensable for this research. As well, I would like to thank Elie Saadé and Yoseph (Sefi) Kalmonson of TAV College, who graciously 3D printed several parts for the rover.

Finally, I'd like to thank my wife Esther, without her encouragement this thesis would still be a very short draft.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This thesis aims to integrate a suitable controller into the MWR project. Following is a description of the MWR project, its history and motivation, as well as the motivations of this thesis, its contributions, and the thesis outline.

1.1 Overview and History of the MWR project

The gardening industry is a multi-billion-dollar enterprise, and currently lacks an autonomous system for removing weeds from a home garden. The Mobile Weed Remover (MWR) project in the Advanced Mechatronics and Robotics Lab at Concordia seeks to fill this void. The MWR project aims to build a mobile weed removal robot which would traverse a garden to seek and destroy weeds.

1.1.1 Purpose of MWR Project

The purpose of the MWR Project is to build a robot with weed removal device to autonomously identify and dig out dandelion weeds from a garden. Gardening is a large industry, and it is estimated that it costs between 65 and 120 USD to clear a typical 0.25 Acre lawn, with the US national average being 95 USD [1]. Herbicides used for the purpose of eliminating dandelions typically contain 2,4-Dichlorophenoxyacetic acid, Mecoprop, Dicamba, or Quinclorac [2]. 2,4-Dichlorophenoxyacetic acid has been found in the semen [3] of workers who work with it, and it has been found to alter sperm shape and transfer [4], thereby creating fertility risks. Mecoprop is slightly toxic [5]. Dicamba is considered highly volatile and a danger to non-targeted vegetation due to spray drift [6] and several products containing it have already been declared by US Federal Courts as being illegally approved [7]. Quinclorac does not biodegrade well on its own and its residue is considered an environmental pollutant [8]. Thus, there is a great need for a mechanical system to autonomously remove dandelions to replace manual labor and pesticides.

In industry, there are already rovers for the purposes of removing weeds/dandelions. However, these are very limited in their abilities. The Tertill [9], for example, recognizes weeds based on height and cuts them down, but it does not eliminate the roots. It also requires a very specific shape of garden and assumes that desirable plants are already cultivated to a certain minimal height, and that anything less than that height is an undesirable weed. Other rovers, such as the Acorn [10], work on the assumption that anything that is not in the correct row is a weed which must be destroyed. These too do not destroy the roots, and are not practical for a typical, unordered, garden.

Gokul et al. [11] have suggested a design for removing weeds via a 4 Degree of Freedom robotic arm mounted on a rover. Their design requires the user to "train" the robotic arm to remove weeds, after which the robot can recall the motions and remove weeds on its own, when the weed removal function is called. This robot was only tested on ploughed land, and it is unclear if it would remove the roots of the weed or merely tear out the leaves when operating in an unplowed garden. Furthermore, the movement of the rover and the activation of the removal function still requires a human operator. Thus, while this design potentially reduces the use of pesticide and effort exerted to remove weeds in an agricultural environment, it still has significant human labor costs, and appears to be unable to work in a typical garden environment.

The MWR seeks to provide a rover which destroys dandelions and can work in a generic garden environment. It aims to use a camera to recognize what is a weed and where it is, and a drilling mechanism end effector to destroy the roots of the weed. Due to the size of the drill bit compared to the size of the roots, and the fact that it does not assume that all plants in the garden are arranged in neat rows, it must have a relatively precise positioning control system.

1.1.2 First Generation (2018 Capstone Team)

The first iteration of the MWR was created by a Capstone Team in the 2017-2018 academic year (see Figure 1.1). As can be seen, the drill assembly was placed in the front of the vehicle, which raised safety concerns and required a special assembly to place a camera for the purposes of recognizing weeds near it. It was fairly limited from a control's perspective as it used a simple algorithm wherein it would either rotate or translate (but not both) a fixed amount and then check its position relative to the target. It also used a fairly primitive weed recognition algorithm. It was partially disassembled by a Capstone team in 2020, and as of the time of this writing is not in working condition.

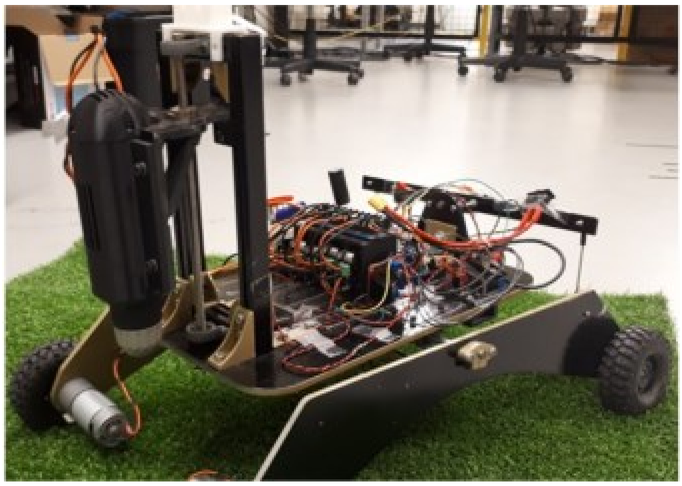


Figure 1.1:The original MWR [12]

1.1.3 Second Generation (2020 Capstone Team)

The second generation of the MWR was create by another Capstone Team in the 2019-2020 Academic year (see Figure 1.2). It placed the drill assembly near the center of the platform, which addressed many of the safety concerns, but raised issues with the field of visions of the camera if it was to be mounted such that it could see the weed during the drilling process. Unfortunately, due to the Covid-19 lockdown, it was left in a state of disrepair, did not have the electronics properly integrated, and the control software was lost. Part of the work of this thesis was to integrate the electronics and rewrite the control firmware.

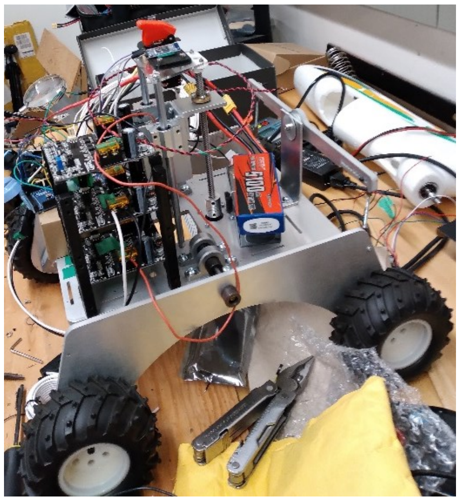


Figure 1.2: The second iteration of the MWR

1.2 Description of system and subsystems

There are several major systems in the current configuration of the MWR, these are summarized below.

1.2.1 The controller (Primary Computer)

Currently, an HP laptop runs Windows 10 on a quad-core/8-thread i5-1035G1 processor. This acts as the controller and will henceforth be referred to as either the laptop or the primary computer in the rest of this thesis. Ideally, the onboard computer should function in this role. However, it is difficult to modify the control parameters that way, so during the design phase, the MWR will use an offboard laptop for the primary computer.

1.2.2 The Observer

The observer currently consists of a Logitech NPU01-2U camera and a Raspberry Pi computer (henceforth referred to as either the Raspberry Pi or as the onboard computer) running Raspbian

OS on a quad core ARM processor. The onboard computer takes the image from the camera, extracts the location of the weed(s), and sends the information to the primary computer over UDP.

1.2.3 The Driving System – Actuators & Drivers

The driving system consists of the DC motors which move the robot, the DC motor which spins the drill bit, the Cytron MD30C driver boards which provide power to them, the stepper motor (for raising and lowering the drill assembly), the stepper controller, and the Arduino Mega which receives commands from the primary computer (via an HC05 Bluetooth module) and controls the various drivers. The motors used by the Capstone seem to be generic 12V DC wormgear gearmotors (see Figure 1.3), which were replaced by 32PG-31ZY Planetary Gearmotors due to their output shafts being too short and the wheels falling off.

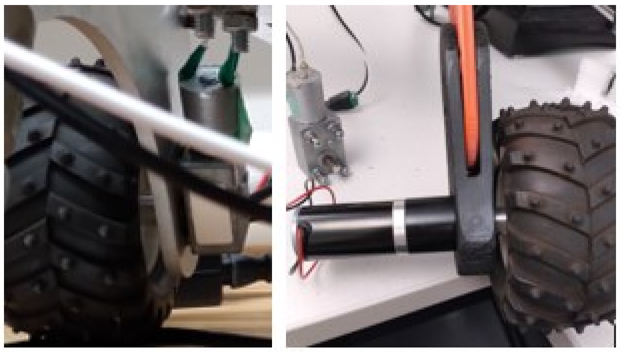


Figure 1.3: The old and new motors

1.3 Motivation

The motivation of this master's thesis is to redesign both mechanical and electrical systems of the second generation of MWR system so that it is able to move and position itself towards the weeds. Also, the mathematical model of the MWR needs to be built and an image-based visual servoing control system will be implemented to position a simple mono-chrome target representing the dandelion weed. Hence, the project aims to improve the design and control algorithm of the MWR systems and to make the second-generation prototype functional and safe.

1.4 Contribution

The contribution of this work is to adapt the control algorithm suggested in [13] to position the MWR over a target. The main contributions are summarized in the following:

- Improve the design of both mechanical and electrical systems of the second generation of MWR to make it functional;
- Build mathematical models of MWR including the "ideal modeling" and the "discrete modeling" and carry out the simulation in Matlab/Simulink to investigate the dynamics of the robot;
- Design the control algorithm of the WMR including forward-backward position correction and error direction and handling;
- Integrate the designed controller with the improved MWR robotic systems and implement the real time control system;
- Carry out both simulation and the experimental tests on the designed integrate systems to position a simple mono-chrome target representing the dandelion and achieve satisfactory positioning results.

1.5 Thesis Outline

Henceforth, the thesis is organized as follows. In Chapter 2, a review of the literature pertaining to the visual servoing control of rovers is conducted. In Chapter 3, a summary of the work done on the development of the rover is presented, including the mechanical modifications to the rover, the electrical modifications, and the development of the simulation model. Chapter 4 gives the suggested control algorithm as well as the simulated results of said algorithm. In Chapter 5, the real-time implementation of said algorithm and its results are presented. Finally, the results are analyzed, the limitations of this research are discussed, and future work is suggested in Chapter 6.

Chapter 2 Literature Review

Visual Servoing is a general description for a control system which responds to visual feedback. Visual Servoing has been used in robotics to increase the robustness of robotic systems, particularly in unstructured environments. In this chapter, a literature review on the classification of visual servoing and the research work on the control algorithms for mobile robots is carried out.

2.1 Classification of Visual Servoing

Visual servoing has been used for workpiece manipulation [14] [15], ensuring the path tracking repeatability of industrial robots [16], improving medical imaging [17], and guidance of observation drones [18]. While there are many visual servoing algorithms in literature, they can generally be classified as one of three types of visual servoing, namely position-based visual servoing (PBVS), image-based visual servoing (IBVS) and Hybrid 2.5D visual servoing. In PBVS, features are extracted from the image and used to deduce a 3D model of the environment. A PBVS controller is then used to guide the rover through the 3D environment. Thus, the controller inputs are based on the true positions of the robot and target structures. In IBVS, the image is directly used to guide the robot, by contrasting the location of the features within the image with their desired position in the image. Thus, the controller inputs are pixel coordinates, rather than "true location". The benefit of IBVS over PBVS is that it is less computationally intense and less susceptible to slight errors in the modeling equations and camera noise. The downside of IBVS is that depending on the skew of the camera with respect to the target the correspondence between 3D distance and pixel value might vary throughout the image, thus giving an inconsistent error signal when compared to the true position error, thereby making it more difficult to create an ideal controller. In hybrid 2.5D visual servoing, both image data and 3D pose information are used for visual servoing of robots, thus it can have the advantages of both.

2.2 Position-based Visual Servoing (PBVS)

In order to implement PBVS, the camera extrinsic parameters must be known. In [19], Sharma et al. suggest a method to automatically estimate the extrinsic parameters during the operation of the rover, through gradient descent. While this method works well for a known target, it does not work well for a generic target of unknown size and shape, as the error is difficult to estimate in such a case. In [20], Zhang et al. recommend a multistage method for implementing PBVS with uncalibrated/unknown camera intrinsic and extrinsic parameters. It first establishes the kinematic model experimentally. Then, it uses an adaptive controller to reduce the angular error to near zero, and in the final step it uses a purely translational controller to bring the robot to its final position. The benefit of this approach is a divide-and-conquer approach which makes the controlling relatively simple despite many unknowns. However, it assumes a purely translational stage in which any angular errors are not accounted for, which is not a good assumption on uneven terrain. In all the previously mentioned literature, there is a predefined “image” of what the final pose should look like. In [21], Li et al. suggest a method for developing a PBVS-like system even without a known final pose image. They suggest recording an initial image, moving to an intermediate pose to record a new image, and extrapolating from there what the final image pose should be. While this technique does allow for a generic target to be targeted during operation, it requires prior knowledge of the camera parameters.

2.3 Image-based Visual Servoing (IBVS)

In order to implement IBVS for mobile wheeled robots, one needs not know the intrinsic parameters of the camera. In [22] , two algorithms to implement IBVS are suggested using a fixed camera which is mounted in a manner that has an image plane parallel to the plane which the wheeled robot is operating within. In [23], an algorithm to implement IBVS with a modification to steer clear of potential singularities is suggested using a similar setup. The benefits of these approaches are that the controllers are fairly straightforward to implement and are agnostic to the camera parameters. The drawback is that they require a fixed camera, fixed directly above the area which the rover will be operating in. In [24], an algorithm to implement IBVS with a camera which is skewed with respect to the plane which the rover will be moving in is suggested. Although this removes the requirement that the camera be mounted directly above the area of operation, it still requires a fixed camera which is mounted independently of the rover. In [25], an IBVS controller is developed in polar coordinates to attain the desired pose of a robot using an onboard camera. This approach does not require an independent fixed camera but does require exact information about the geometry of the target or a prerecorded image of the target in the desired pose.

2.4 Hybrid 2.5D Visual Servoing

In [26], a control strategy is suggested using a combination of computed angular errors with the desired path and pixel location of particular image features as they should appear when the image is captured along the path. While the controller works well, it requires both a sequence of images be previously captured along the desired path of the robot and previous knowledge of the camera intrinsic parameters. In [27], an algorithm is suggested wherein three image features are captured in two different poses, and the relevant coefficients for estimating the pose after that are calculated using the continuous motion constraint in order to allow the robot to be guided to a predetermined position. In [28], an algorithm is suggested to use projection homography to estimate partial camera intrinsic parameters, after which pose and scaled position are estimated using the determined intrinsic parameters, multiple views, and geometric constraints, in order to guide the rover to the predetermined position. While the later two techniques remove the requirements for a prior knowledge of the intrinsic camera parameters, they still require a reference image captured in the desired position.

2.5 Summary

In this chapter, a brief overview of the literature on the visual servoing for mobile robots is presented. The classification of the visual servoing is given. The analysis on the features and implementations of PBVS, IBVS, and Hybrid control schemes is presented.

Chapter 3 Development of the Rover

When developing a mechatronic system, prior to developing a control system, there are three types of work one must do:

1. Mechanical Design
2. Electrical Design
3. Mathematical Modeling

Following is a summary of the work done in these domains during the course of the research.

3.1 Mechanical Additions to the rover

Overall, the mechanical structure of the rover had been implemented by the 2020 Capstone team. However, several additions had to be made in order for the rover to be useable for the purposes of this research.

3.1.1 Stilts and Motor Adaptors

The rover did not initially have enough clearance to mount a camera such that it would be able to have a large enough field of vision to include the location of the end effector. To rectify this, it was decided to place "stilts" between the gearmotors and their mounting points on the rover (See figure 3.1). It was later determined that the gearmotors had to be replaced (note the difference in available protruding shaft to mount the wheels in the first and last images of Figure 3.1), so rather than discarding the stilts an adapter was designed to allow the new gearmotors to be attached to the previous interface.

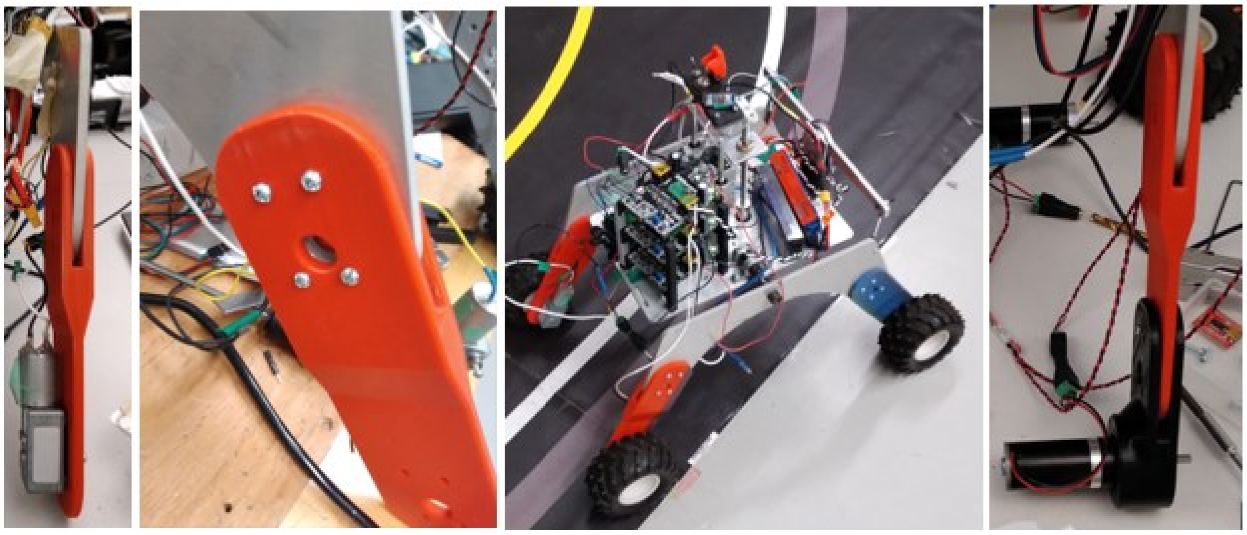


Figure 3.1:The Stilts with the old and new motors

The stilts and adapters were designed in SolidWorks (see Figures 3.2 and 3.3) and 3D printed using FDM 3D printing. The stilt acts to replicate the interface for the motors as found on the chassis such that it is moved by 14.5 cm along the axis of the stilt (see Figure 3.1). The adapter both shifted the location of the mounting of the motor and modified the interface to which the motor is attached (see Figure 3.1). Due to the small length of the mounting screws for the new motors, the mounting surface on the adapter had to fairly thin (about 2.45mm). Due to the thinness of the mounting surface, the adapter was designed to have the motors press fit into them as an additional form of fastening.

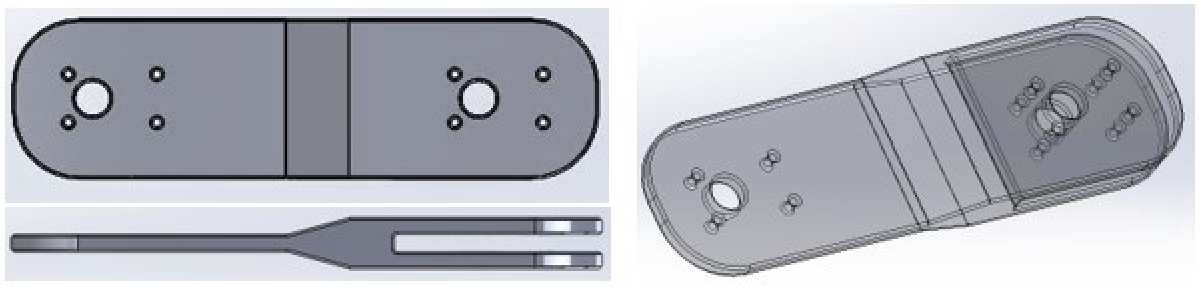


Figure 3.2:The Stilt as Designed in SolidWorks

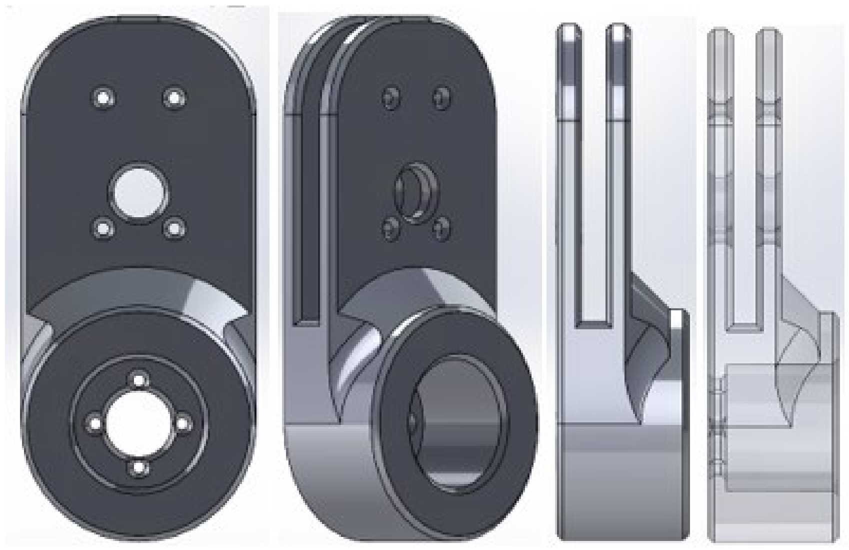


Figure 3.3:The Adapter for the New Motors

3.1.2 Switch Mount

The prototype as received had no ON/OFF switches or emergency stops. This posed both practical and safety challenges. Two toggle switches were added to the circuitry, one simple switch for controlling the power to the Arduino, and a second “emergency stop” switch to power the actuators. The emergency stop is a prominent red cap that would push the switch into the off position if pressed down, which is designed to be easy and intuitive to use in an emergency (see Figure 3.5). To mount the switches to the rover in a manner that they would be easily accessible a switch mount was designed which took advantage of cutouts that already existed in the top of the drilling assembly’s support. The top of the drilling assembly was chosen as the ideal place for the switches as it is the highest point on the rover, and the easiest to reach while the rover is on the ground. It also provided a convenient way to add an upper limit switch to the drilling assembly which was previously not an option. The switch mount consisted of two plates (see Figure 3.4), one above, one below, which were connected via screws through pre-existing cutouts. (see Figure 3.5).

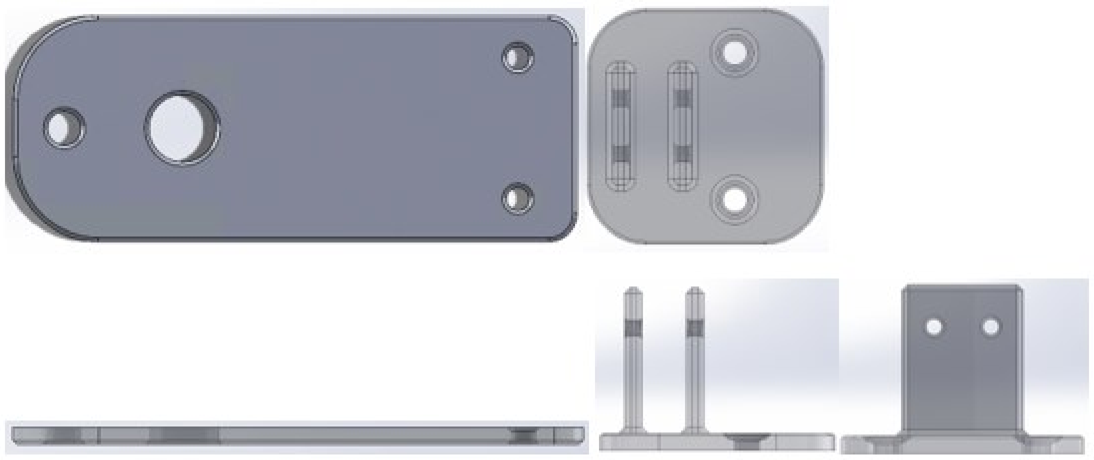


Figure 3.4:Switch Mount CADs

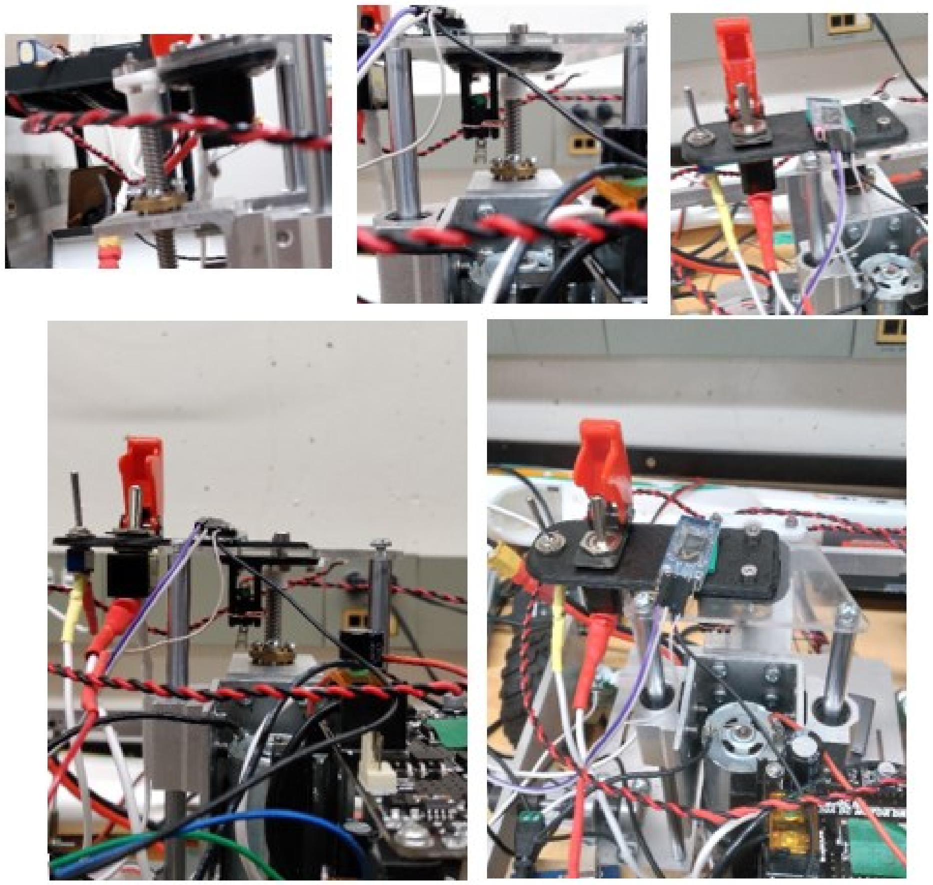


Figure 3.5: Various Views of the Switch Mount

3.1.3 Cytron Board Tower

The rover had several Cytron MD30C boards, which would have taken up precious space on the rover platform and would have run the risk of short circuiting if they touched the metallic platform. To solve these issues, the boards were stacked in a "tower" which prevented them from touching each other or the metallic platform. This design consists of "beams" (see Figure 3.7) and "supporting brackets" (see Figure 3.6), each Cytron board is attached at its four corners to four "supporting brackets", which in turn are mounted to the four "beams" (See Figures 3.8 and 3.9, note that the Cytron Model in Figure 3.8 comes from [29]). As long as two Cytron boards are connected, the assembly is stable. This configuration makes it relatively easy to assemble and disassemble, and has the added benefit that the power input to the Cytron boards can be easily "daisy chained" (as seen in Figure 3.12 in Section 3.2.1)

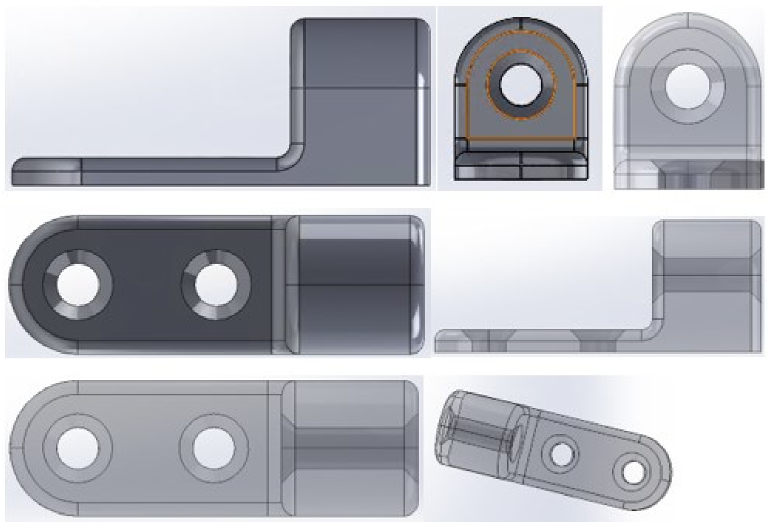


Figure 3.6: Cytron Tower Supporting Bracket CAD

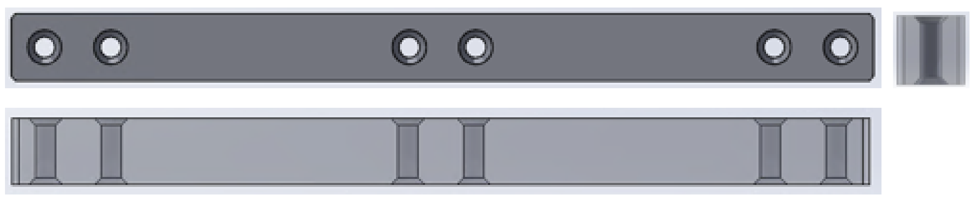


Figure 3.7: Cytron Tower Beams CAD

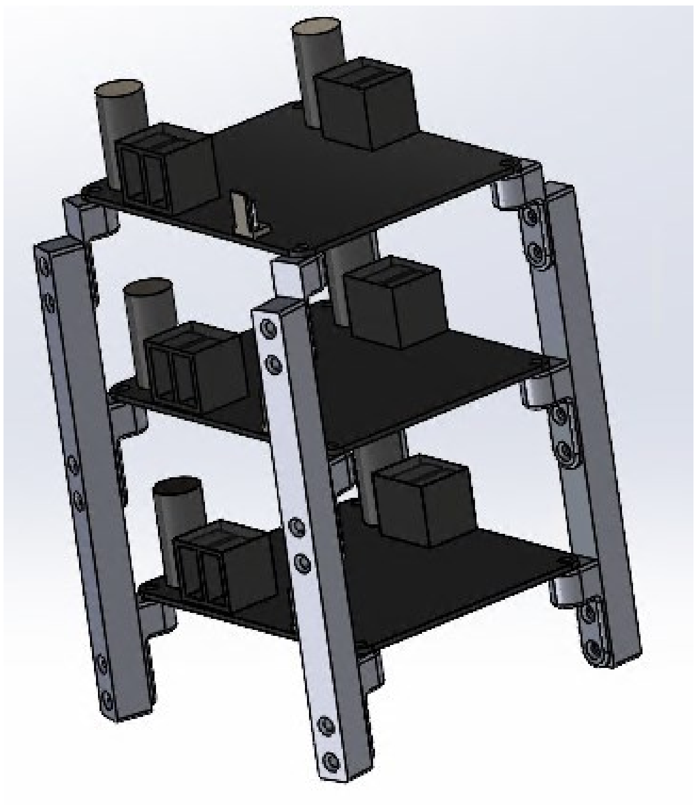


Figure 3.8: Cytron Tower Assembly CAD

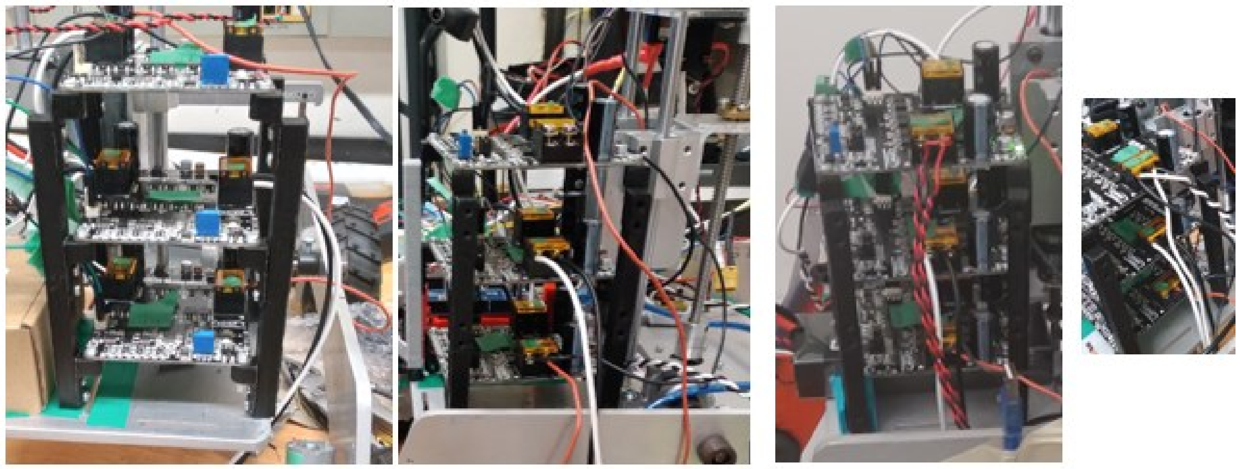


Figure 3.9: The Cytron Tower Assembled

3.1.4 Battery Basket

The Battery Basket was designed to minimize the amount of space on the rover platform taken up by the batteries. The basket had to be designed to be able to hold two LiPo batteries and not to fall off during operation, while also being easy to remove for the purpose of replacing the batteries.

The final iteration included the option of using screws to fasten the basket to the platform and was printed in PLA+ as an added precaution.

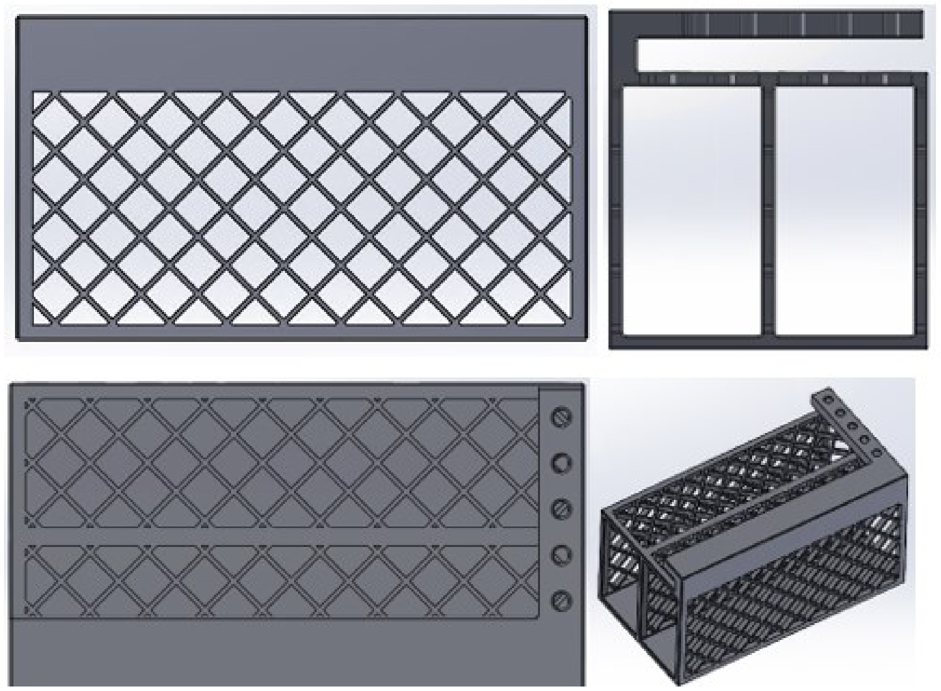


Figure 3.10: Battery Basket CAD

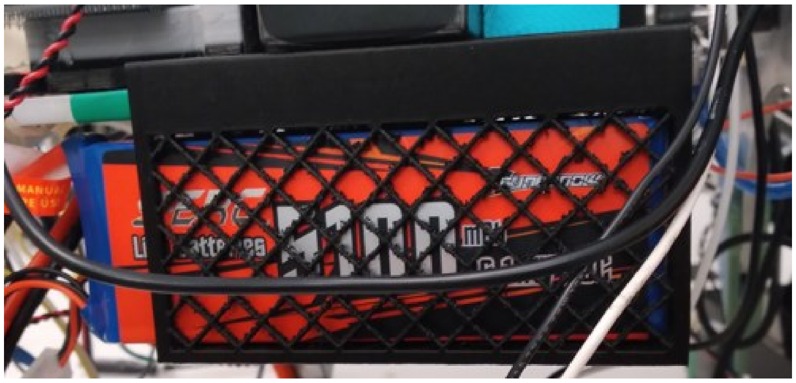


Figure 3.11: Battery Basket in Use

3.2 Electrical Additions to the Rover

The rover, as received from the capstone team, did not have proper wiring, any form of an ON/OFF switch, or an emergency stop. It also did not seem to have anywhere to attach the batteries (it seems that these had been borrowed from one of the labs, and the entire battery circuitry was removed when returning the batteries that the capstone had used for development). Several batteries had to be incorporated into the design, as different devices had different voltage requirements, and to ensure electrical insulation between subsystems.

3.2.1 Actuator Power Circuit

The Cytron boards for controlling the actuators are all powered via one 12V 3S 5100mAh LiPo battery. The Cytron boards are wired in parallel with each other (see Figure 3.12) and in series with the Actuator power kill switch (and the X60 connector which connects to the battery). The actuator power circuit also has a barrel jacket adapter wired in parallel with the Cytron boards, for powering other devices if need be (it was primarily used to power the DC cooling fan for the on-board computer).

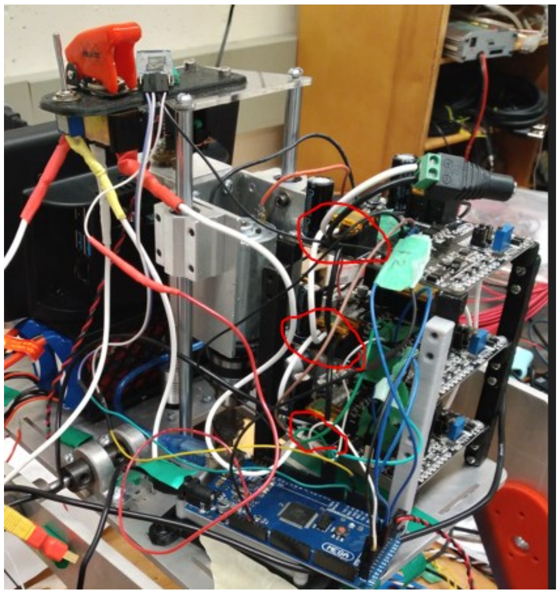


Figure 3.12: Actuator Power Circuit

A total of three Cytron boards were used. One of them is to drive the drill for destroying weeds, one is to drive the left-hand wheels, and one is to drive the right-hand wheels. The output terminals for the wheels were wired in parallel, such that the voltage received by the front right front wheel is the same as the voltage received by the right rear wheel, and the voltage received by the left front wheel is the same as the voltage received by the left rear wheel. Although this does not allow for more complex algorithms such as traction control, at this stage in development this arrangement is satisfactory and allows the reduction of the number of required Cytron boards.

3.2.2. Pi Power Circuitry

The Raspberry pi does not have any on-board voltage regulator, and therefore cannot be directly connected to a LiPo Battery. The Pi requires between 5.25V and 4.75V, and up to 3A. In order to meet these strict requirements, an AMONIDA 6V-32V to 5V DC-DC converter with a USB-A interface was used to provide 5V power. While the AMONIDA DC-DC converter can operate on as little as 6V, it was noted that the alternative DC-DC converters readily available had a minimum input voltage of 7V, therefore as a precaution a 3S LiPo Battery was used to power it, as a 2S LiPo can drop to 6V during normal operation. In order to ensure that a wire did not get loose and potentially start a fire, the input terminal of the DC-DC converter was removed, and a soldered connection was made to an X60 connector (see images below). The Raspberry Pi cable already has an ON/OFF button, and the computers do not generally require emergency stop options, so no switch was required in this circuit.

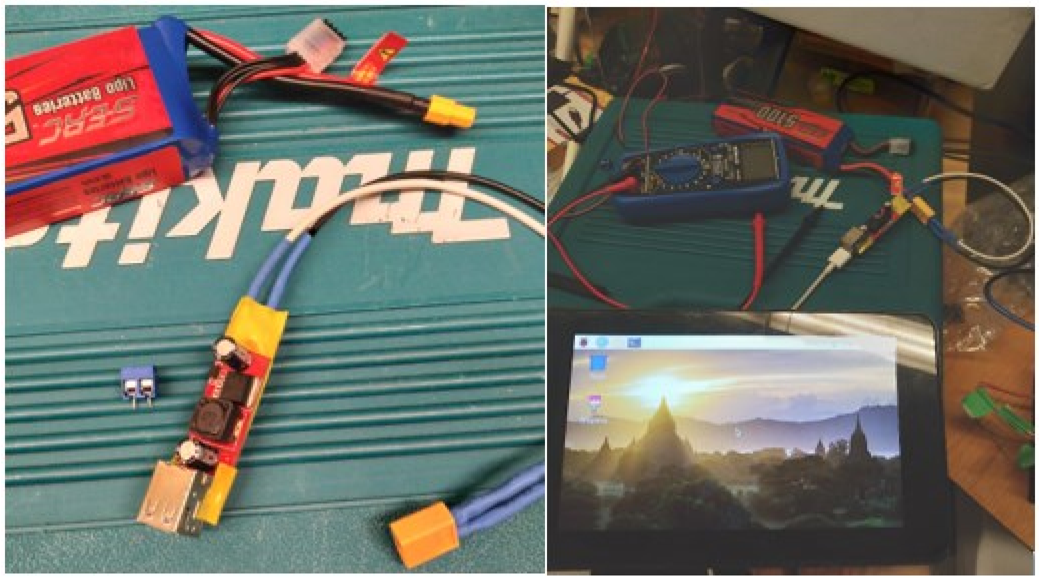


Figure 3.13: The Raspberry Pi and its Power Supply

3.2.3 Arduino Power Circuit and Wiring

The Arduino Mega chip has a built-in voltage regulator and can therefore be powered directly by a LiPo battery either via the vin pin or the barrel jack. The Arduino power circuit consists of an ON/OFF switch, an X60 connector (for the LiPo), and a barrel jack connector (for the Arduino) in series with each other.

The Arduino Mega is placed beside the Cytron tower (see Figures 3.14 and 3.15) such that the Cytron boards can easily be connected to the Arduino Mega. Each Cytron board is connected to a ground pin, a DIO pin for controlling the direction, and a PWM pin for controlling the magnitude. In addition, the Arduino Mega is connected to an HC06 board via Serial3 in order to communicate with the primary computer. A modified version of the Cytron Tower "Beam" is designed with a small platform to mount the Arduino Mega to so that it does not touch the platform and short circuit.

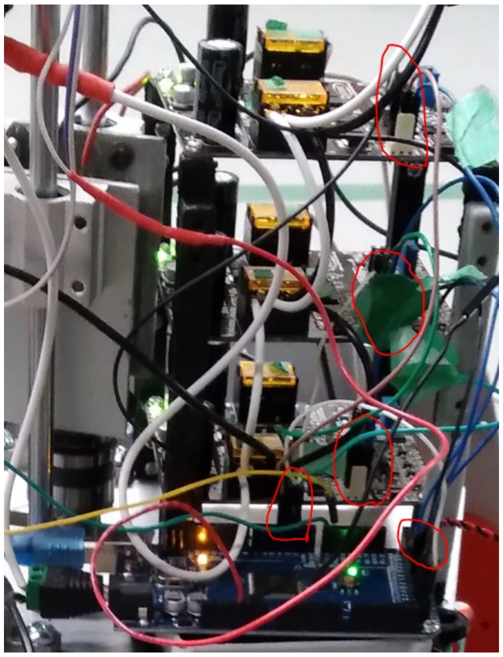


Figure 3.14: The Connections between the MD30Cs and the Arduino Mega

3.3 Mathematical Modelling of MWR

The mathematical modeling of the MWR can be split into two parts, “ideal modelling” and “discrete modelling”. The ideal model is derived from the mathematical formulae, assuming ideal motors and no time delay. The discrete model is built in MATLAB Simulink and seeks to be taken into account the various delays and discretization errors, through the use of discretization blocks, first order holds, and adjusting the rate transfer frequency of the inputs and outputs to the different modules.

3.3.1 The Ideal Model of the MWR

The ideal model is used in order to derive a controller which would work under ideal conditions and will then be adapted to work with real world conditions. At this stage, we assume ideal motors with no transient response delay. Therefore, it is assumed that through the equations found in ref [30] and assuming that the rover is a 2-wheeled robot (although the MWR is using a 4-wheel chassis, we will link the right motor inputs together and the left motor inputs together and treat it as if it were a 2-wheel rover), a desired rotational and translational speed can be instantly achieved. It is also assumed that the sensor acts to update the controller instantaneously without any delay.

We begin by defining our coordinate system, as shown in Figure 3.15. We take x and y as being the coordinates of the rover in the global coordinate system, x_o, y_o as being the coordinates of the target with respect to the world coordinate system, and I, J being the coordinates of the target with respect to the camera, in the camera coordinate system.

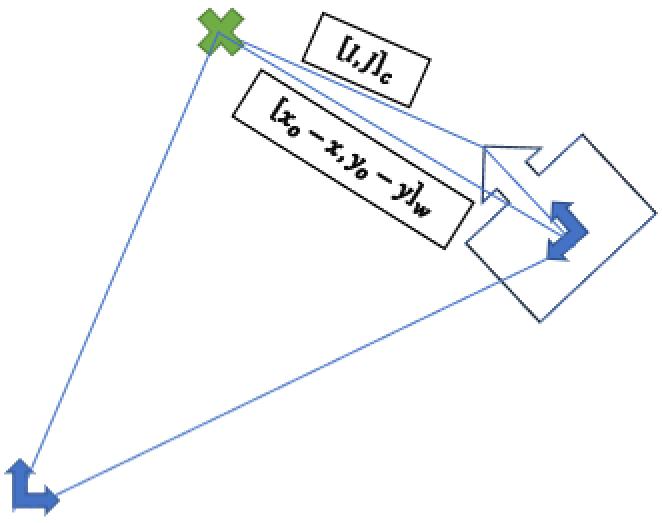


Figure 3.15: Illustration of Coordinate System Used to Derive Ideal Model

The equation for the relationship between I, J and x, y is:

$$\begin{bmatrix} I \\ J \end{bmatrix} = R_{wc}(-\theta)\begin{bmatrix} x_o - x \\ y_o - y \end{bmatrix} - \begin{bmatrix} L \\ 0 \end{bmatrix} \tag{3.1}$$

where R_{wc} is the rotation matrix between the world coordinate system and the rover coordinate system, θ is the angle of the rover with respect to the world coordinate system, and L is the length between the center of the camera and the center of the rover. Within the world coordinate system, the motion of the rover can be described by the following equation:

The equation of motion for a 2 wheel rover in 2D space [31] is given:

$$\dot{\theta} = \omega \tag{3.2}$$

$$\dot{x} = v * \cos(\theta)$$

$$\dot{y} = v * \sin(\theta)$$

We now proceed to find the equivalent equations in the camera coordinate system, by taking equation (3.1) and taking the derivative:

The time derivative of equation (3.1) is

$$\begin{bmatrix} I \\ J \end{bmatrix}' = \left(\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} x_o - x \\ y_o - y \end{bmatrix}\right)' - \begin{bmatrix} L \\ 0 \end{bmatrix}' = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}' \begin{bmatrix} x_o - x \\ y_o - y \end{bmatrix} +$$

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} -x \\ -y \end{bmatrix}' - \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) \\ -\cos(\theta) & -\sin(\theta) \end{bmatrix}\begin{bmatrix} x_o - x \\ y_o - y \end{bmatrix}\omega +$$

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} -v * \cos(\theta) \\ -v * \sin(\theta) \end{bmatrix} \tag{3.3}$$

From equation (3.1), we know that:

$$I = -\cos(\theta) * \mathrm{x} - \sin(\theta) * \mathrm{y} + (\cos(\theta)\,\mathrm{x_o} + \sin(\theta)\,y_o - L) \tag{3.4}$$

$$J = \sin(\theta) * x - \cos(\theta) * y + (-\sin(\theta)\,x_o + \cos(\theta)\,y_o)$$

Rewriting equation 3.3, and back substituting in the two equations above, we have:

$$I' = (\sin(\theta)\,x\omega - \cos(\theta)\,y\omega - \sin(\theta)\,x_o\omega + \cos(\theta)\,y\omega) - (v * \cos^2(\theta) + v * \sin^2(\theta))$$

$$I' = (\sin(\theta)\,x - \cos(\theta)\,y - \sin(\theta)\,x_o + \cos(\theta)\,y)\omega - v \tag{3.5}$$

$$I' = J\omega - v$$

And we have:

$$J' = (\cos(\theta)\,x + \sin(\theta)\,y - \cos(\theta)\,x_o - \sin(\theta)\,y_o)\omega - v(\cos(\theta)\sin(\theta) - \cos(\theta)\sin(\theta))$$

$$J' = (\cos(\theta)\,x + \sin(\theta)\,y - \cos(\theta)\,x_o - \sin(\theta)\,y_o)\omega \tag{3.6}$$

$$J' = -(I + L)\omega$$

This gives us our open-loop model of the system as:

$$I' = J\omega - v \tag{3.7}$$

$$J' = -(I + L)\omega$$

We can construct a simple closed loop model, per [13], as:

$$\omega = k_\omega J \tag{3.8}$$

$$v = k_v I$$

The closed-loop equations for the ideal model of the MWR are

$$I' = J\omega - k_v I = k_\omega J^2 - k_v I \qquad (3.9)$$

$$J' = -(I + L)k_\omega J$$

If I is s greater than -L, we expect J to asymptotically go to 0. If J is asymptotically going to zero, then I will also asymptotically go to 0. Thus, in the ideal model, as long as proper k_ω and k_v are chosen, and the target is not behind the center of the rover with respect to the camera, the rover will converge on its target.

3.3.2 The Discrete Model of the MWR

The discrete model of the MWR seeks to account for the time delays from the sensor to the controller, from the controller to the driver, and the transient response of the motors. As can be seen in Figure 3.16, the model is a Simulink model with four main subsystems, namely the Controller, Observer, Plant, and Recorder. The controller will be described in Section 4.5 and is an implementation of the algorithm designed in Chapter 4. The other three modules will be described in this section. In order to simulate the transfer delays, all signals to the controller are passed through an 8Hz transfer rate block followed by a first order hold, and all signals from the controller to the plant are passed through a 20Hz transfer rate block followed by a first order hold.

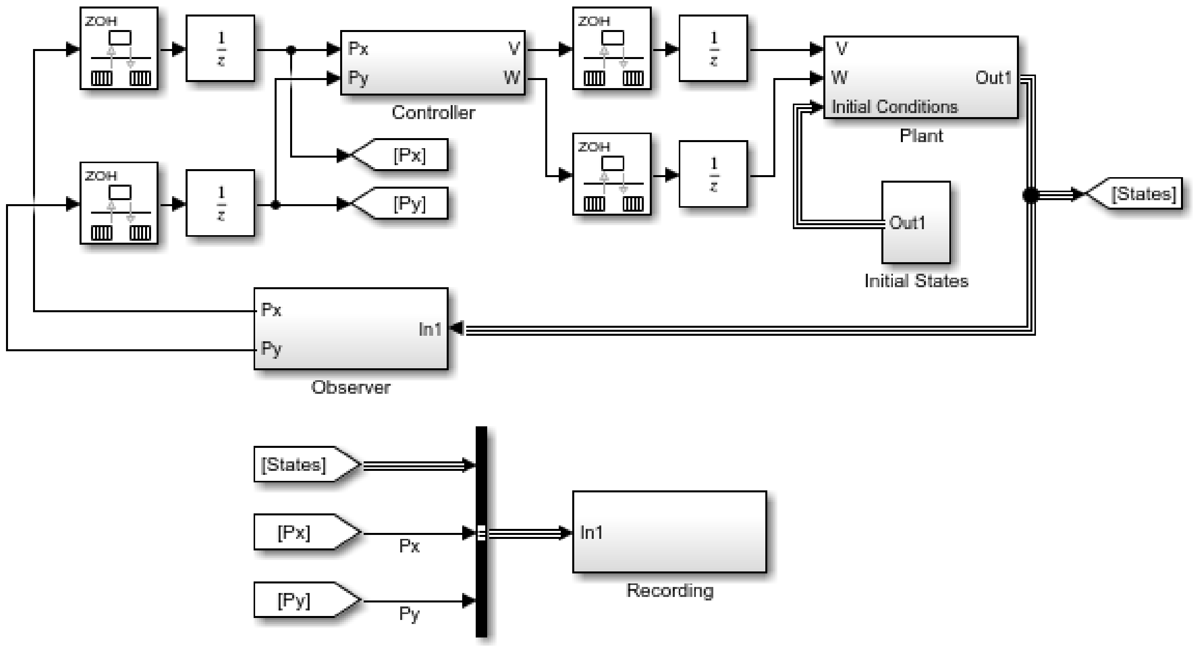


Figure 3.16: General View of Simulink Model

The plant, shown in Figure 3.17, takes *V* and *W* from the controller with a transfer rate of 20ms followed by a first order delay in order to simulate the delay for the controller to calculate the desired *V* and *W* and the Bluetooth communication driver to send it to the Arduino Mega (as described in Section 5.2). It outputs the various states of the rover as a bus. It takes the velocities from the controller and passes them to the motor model block (labeled as *VW* to true *VW*) and receives from it the current velocity and rotational velocity values (as values from -1 to 1), it then multiplies those values by the estimated max translational and rotational velocities of the rover, which gives it the *V* and Omega values. From those it calculates the position and angle of the rover using equations (4.6). The initial condition input is taken from a block with values populated by the initialization script. The *X, Y* coordinate system is chosen such that the target is at the origin.

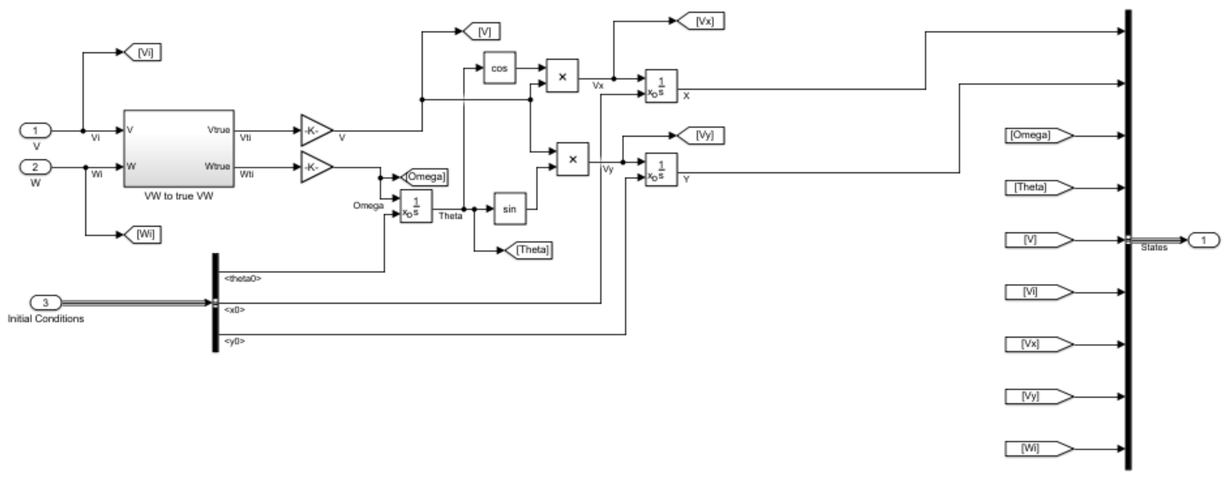


Figure 3.17: The Plant Simulink Model

The motor model block, shown in Figure 3.18, takes the *V* and *W* values from the controller and accounts for the discretization error and the transient response of the motors. It calculates the commanded right and left rotations per equations (4.5) and (4.6), clamps them between -1 and 1 per the procedure described in equation (5.1), adjusts for the discretization effect that results from storing the value as an 8-bit integer as described in Section 5.2.2, and passes it to the motor models, which are assumed to be first order transfer functions. From the motor responses the actual *V* and *W* (as a fraction of max speed) are calculated, and passed back to the plant model, where they are multiplied by gains corresponding to the maximum translational and rotational speeds.

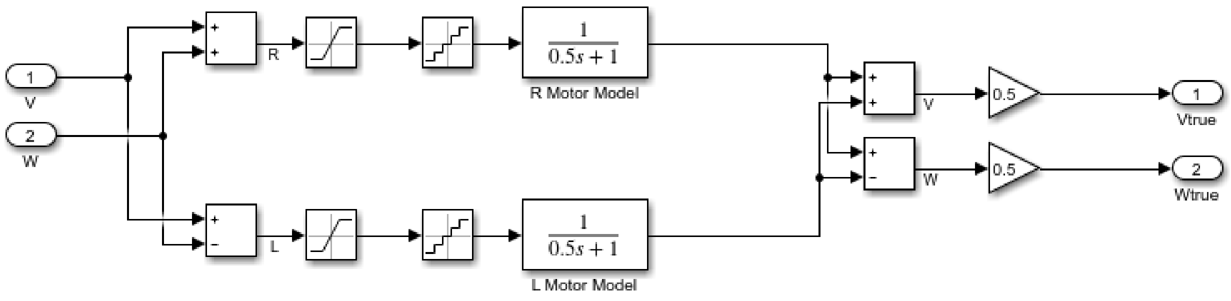
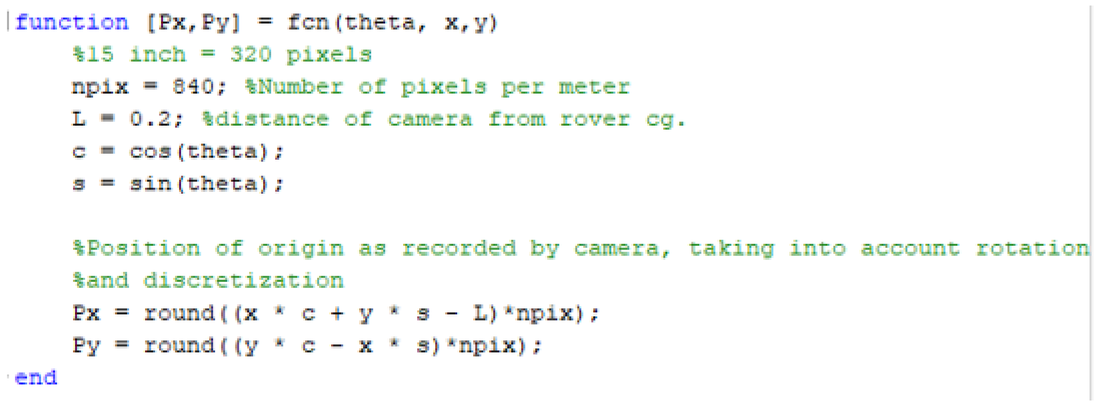


Figure 3.18: The Motor Model Block

The observer block, shown in Figure 3.19, takes the states bus of the rover as input and outputs the pixel coordinates that the target would appear in from the perspective of the onboard camera. The X, Y and Theta values are taken from the rover states bus and passed to the camera model function, which calculates the corresponding pixel values assuming a non-distorted camera that provides an ideal birds-eye view of the target, which would result in a discretized but otherwise linear relationship between the position of the target and the corresponding pixel value [32]. It was experimentally determined that at the height the camera is mounted at on the rover 320 pixels correspond to 15 inches, from which it was derived that there are roughly 840 pixels per meter. For the purposes of the simulation, it was assumed that the camera is ~20cm away from the center of the rover.



```
function [Px,Py] = fcn(theta, x,y)
    %15 inch = 320 pixels
    npix = 840; %Number of pixels per meter
    L = 0.2; %distance of camera from rover cg.
    c = cos(theta);
    s = sin(theta);

    %Position of origin as recorded by camera, taking into account rotation
    %and discretization
    Px = round((x * c + y * s - L)*npix);
    Py = round((y * c - x * s)*npix);
end
```

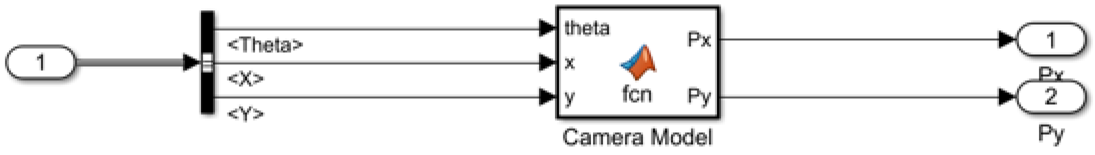


Figure 3.19: The Observer Simulink Model

The recorder block takes the rover states bus and the pixel values, and sends them to two output streams, namely a MATLAB workspace variable and an output file. This allows for analysis of the simulation results in MATLAB.

3.4 Summary

In this chapter, the work done on the hardware of the rover was presented, as well as the preliminary control design and modelling. The designs for the mechanical additions to the rover, namely the stilts, motor adapters, switch mount, cytron board tower, and battery basket were given, as well as pictures of the final product. The electrical work done on the rover, namely the actuator power circuit, the onboard computer power circuit, and the Arduino microcontroller wiring were all presented and explained. In addition, the mathematical model of the MWR was developed, and the discrete Matlab simulation model (with the exception of its controller) was presented.

Chapter 4 Control Algorithm

After the MWR has been re-built to be fully functional, we need to design control algorithm to do the servoing. control algorithm has several components to it, following is a description of them:

4.1 General Control Algorithm

The general control algorithm uses a PI approach to position the end effector over the target region. Two discrete PI controllers are implemented, one for the Right/Left Orientation (Angular Control) and the other for the Forward-Backward Position (Translational Control).

4.1.1 Left-Right Orientation Correction

The left-right orientation of the rover with respect to the target is corrected by applying a PI algorithm to set the rotation command with the following piecewise error function:

The Left-Right Error function:

$$e_x = \left\{ \begin{matrix} 0 & if\ (|I - T_x\ | \leq tol) \\ \frac{(I-T_x-tol)}{C_{cx}} & if\ I > (T_x + tol) \\ \frac{(I-T_x+tol)}{C_{cx}} & if\ I < (T_x - tol) \end{matrix} \right\} \tag{4.1}$$

where e_x is the Left-Right error, T_x is the x coordinate of the desired location for the target in pixel coordinates, I is the x coordinate of the current target location in pixel coordinates, tol is the tolerance for how many pixels away from the desired location is deemed to be close enough, and C_{cx} is a constant which corresponds to the resolution in the x-direction of the image that was captured.

The rotation component of the controller is set as follows:

The commanded rotational velocity:

$$W = k_x e_x + k_{xi} e_{xi} \tag{4.2}$$

where W is the nominal rotation (with CCW being positive) as a fraction of maximum possible rotation (ranging from -1 to 1), e_x is as above (in Eq. 4.1), e_{xi} is a discrete integral of e_x, and k_x & k_{xi} are control coefficients.

4.1.2 Forward-Backward Position Correction

The distance between the rover end effector and the target is corrected by applying a PI algorithm to set the translation command with the following piecewise error function:

The Forward-Backward Error function:

$$e_y = \left\{\begin{matrix} 0 & if\ (\left|J - T_y\right| \leq tol) \\ \frac{(I - T_y - tol)}{C_{cy}} & if\ J > (T_y + tol) \\ \frac{(I - T_y + tol)}{C_{cy}} & if\ J < (T_y - tol) \end{matrix}\right\} \tag{4.3}$$

where e_y is the Forward-backward error, T_y is the y coordinate of the desired location for the target in pixel coordinates, J is the y coordinate of the current target location in pixel coordinates, tol is the tolerance for how many pixels away from the desired location is deemed to be close enough, and C_{cy} is a constant which corresponds to the resolution in the y-direction of the image that was captured.

The translational component of the controller is set as follows:

The commanded translational velocity:

$$V = k_y e_y + k_{yi} e_{yi} \tag{4.4}$$

where V is the nominal translation as a fraction of the maximum possible translation (ranging from -1 to 1), e_y is as above (in Eq. 4.3), e_{yi} is a discrete integral of e_y, and k_y & k_{yi} are control coefficients.

4.1.3 Rotation-Translation to Right-Left Motor Inputs

In order to transfer the commands to inputs that can be used by the rover, the input commands must be changed from V,W (translation, rotation) velocities to (R, L) (right side, left side) velocities. By taking the equations from [30] and modifying it based on our implementation where V and W range from -1 to 1, this can be accomplished via the following formulae:

The right and left commanded wheel velocities:

$$V_R = V + W \tag{4.5}$$

$$V_L = V - W \tag{4.6}$$

where V_R is the commanded power for the right-hand motors as a fraction of maximum available power, V_L is the commanded velocity for the left-hand motors as a fraction of maximum available electrical power, and V & W are as defined in Eqs. 4.2 & 4.4 respectively. Noting that it is possible for V_R and V_L to be beyond the range of [-1, 1], depending on the combination of V and W being used at a given instant, we note that the controller will saturate at the max and minimum values, which will give a slightly skewed result, that should still be representative of the right/left and forward/backward trend that is desired (as explained in Section 5.2.2)

4.2 Error Detection and Handling

Error detection and handling is key to safely operating any robotic system. In the event that the rover receives an erroneous signal from the primary computer, it immediately shuts down the actuators. Similarly, if the on-board computer fails to locate the target in a given image frame, it would set a flag in the status byte of its transmission to the primary computer. When the primary computer receives the flag, it would cause to set V, W, e_{yi}, and e_{xi} all to zero.

4.3 Discrete PI controller Implementation

As noted above, the controller uses an integral component. This is done to overcome dead zones, if there is a region which requires more power to travel through than what would be provided by the proportional controller, such as a slope. The implementation of the discrete PI controller requires a discrete integral of the error function, which is computed by storing variables e_{yi} & e_{xi}(as noted above) and in each iteration incrementing them in accordance with the following equations:

The numerical integrals for the left and right errors:

$$e_{xi} := e_{xi} + e_x * dt \tag{4.7}$$

$$e_{yi} := e_{yi} + e_y * dt \tag{4.8}$$

where e_{xi}, e_{yi}, e_y and $e_x$are as defined above, and dt is the time increment.

4.4 Dead-band Compensation

Due to the internal friction in the gearmotors, the gearmotors have a dead-band at low inputs. To overcome this, the following logic was added to the code:

The sliding mode controller for offsetting the gear breakout force:

$$\begin{gathered} if(W < 0): W \coloneqq W - offset \\ if(W > 0): W \coloneqq W + offset \\ if(V < 0): V \coloneqq V - offset \\ if(V > 0): V \coloneqq V + offset \end{gathered} \tag{4.9}$$

where the offset is a set value in the code, and W & V are as defined in Eqs. (4.2) and (4.4).

4.5 Simulation results

In order to simulate the controller, the controller block shown in Figure 4.1 was used in the Simulink model. As the motor models do not account for friction, the offsets used in Eq. (4.9) were reduced in the model, and a deadband with limits of $\pm tol$ was used to implement Eqs. (4.1) and (4.3). An extra gain Kvw2 was added in order to enable the model to test out the possibility of decoupling the system of equations in Eq. (3.4) by elimination the $k_{\omega}J^2$ component of the derivative of I. As this was not implemented on the rover, this coefficient is set to 0 for the tests being conducted. Once both parameters are within tolerance, the simulation terminates. An extra condition that the time must be greater than 1 second for the simulation to terminate was added to account for the fact that do to the first order transfer delay the error is at zero for the first 125ms, so that the simulation does not immediately terminate on launch. The parameters used for the simulation are recorded in Table 4.1.

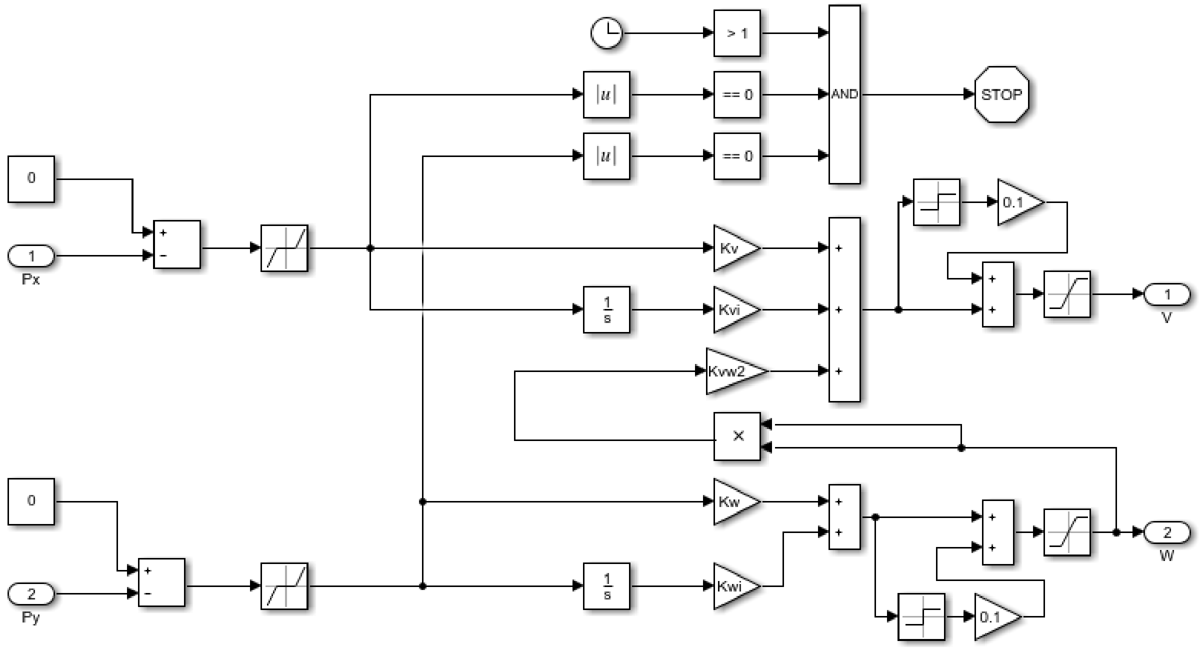


Figure 4.1: The Simulink Controller Model

Three tests were run, with the initial positions varying in each test. One test to test if the control algorithm works with initially only x-error, one test to verify if the control algorithm works with initially only y-error, and one test to see how the control algorithm will react to a combined error input. The results for the first test can be seen in Figures 4.2 and 4.3. As was expected from equation (3.4), the dynamics follow that of a first order system. The results for the second test can be see in Figures 4.4 and 4.5, as expected from equation (3.4), the correction of the y-error induces an x-error. The results from the third simulation can be seen in Figures 4.6 and 4.7. In all cases, the simulations show that the rover can converge the two error functions to zero. Thus, at a simulation level, the algorithm was determined to be satisfactory.

Table 4.1: Parameters Used in Simulation

Parameter Name	Value	Description
L	20 cm	Offset of camera from rover Center of Gravity.
Tau	0.5s	Motor time-constant
Kv	1/160	Control constant for converting from pixels to fraction of V commanded
Kw	-1/140	Control constant for converting from pixels to fraction of W commanded
Kvi	6.2500e-06	Control constant for integral component of V
Kwi	-7.1429e-06	Control constant for integral component of W
Offset	0.1	Offset from equation 4.9 for deadband compensation
Vmax	0.0251 m/s	Max absolute velocity
Wmax	0.3927 rad/s	Max absolute rotation
Tol	5 pixels	Tolerance allowed from target
C_{cx}	160	Coefficient from eq 4.1 for calculating error
C_{cy}	140	Coefficient from eq 4.3 for calculating error

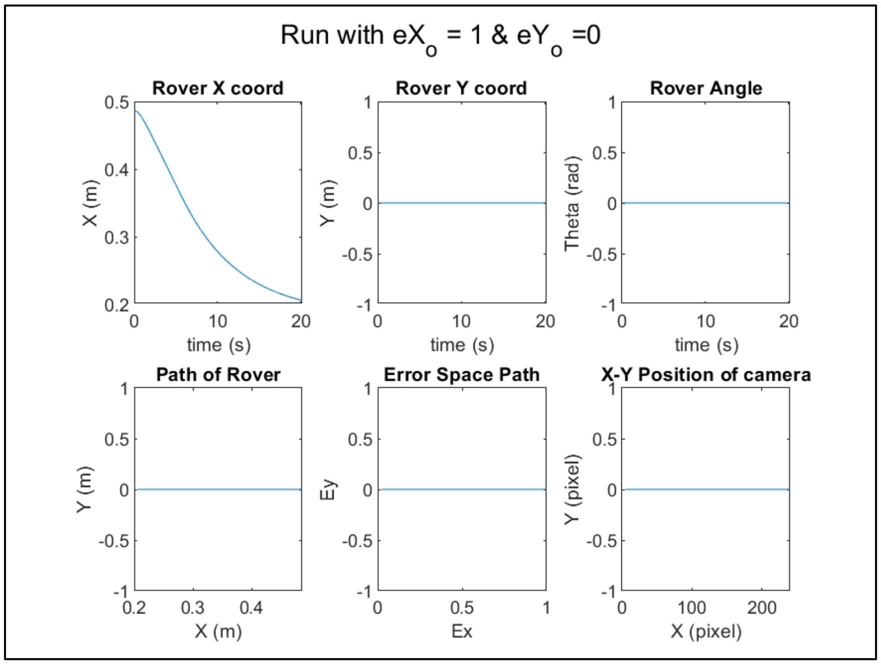


Figure 4.2: Results of first simulation

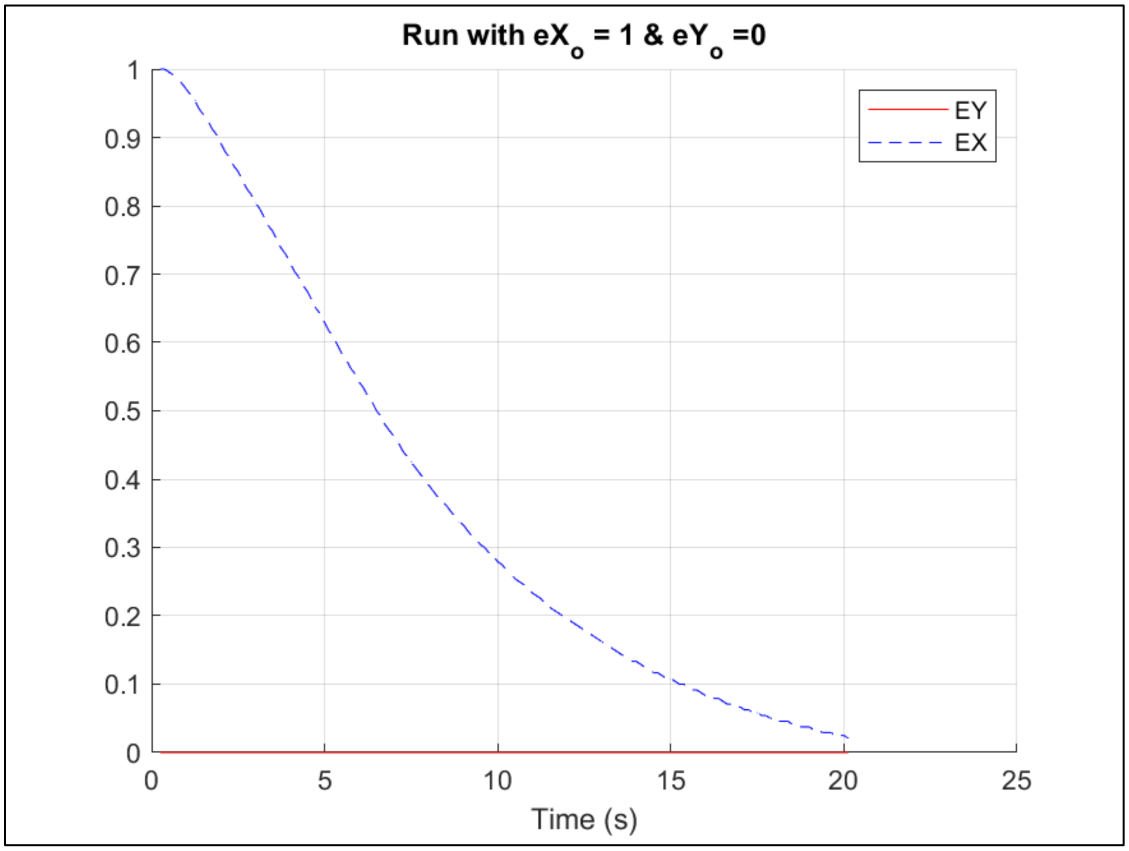


Figure 4.3: Error functions for first simulation

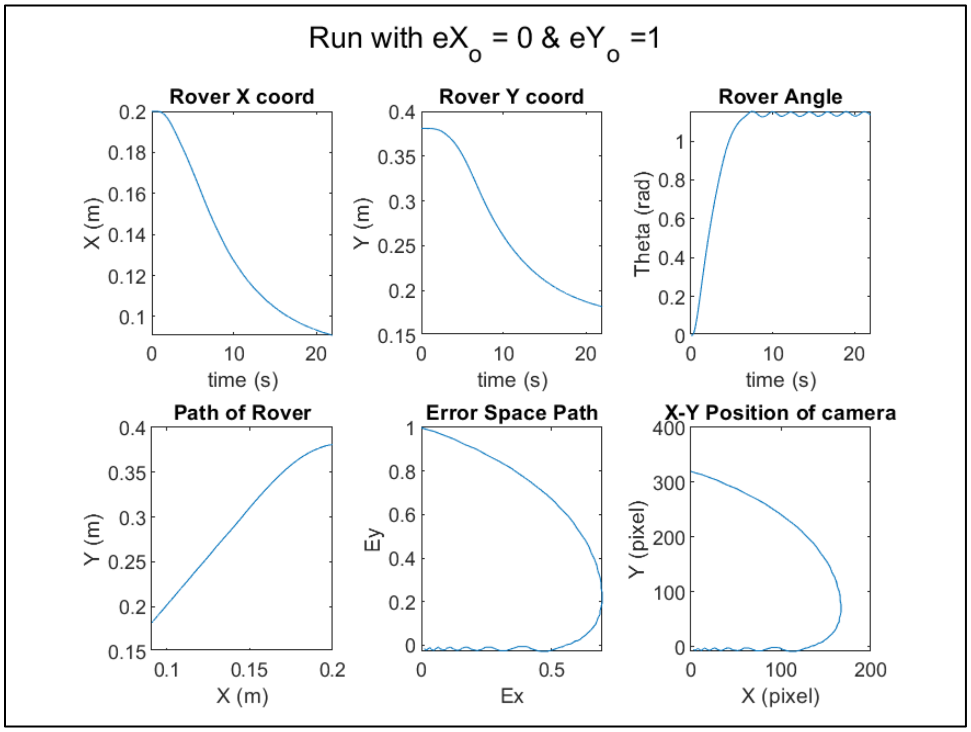


Figure 4.4: Results of Second Simulation

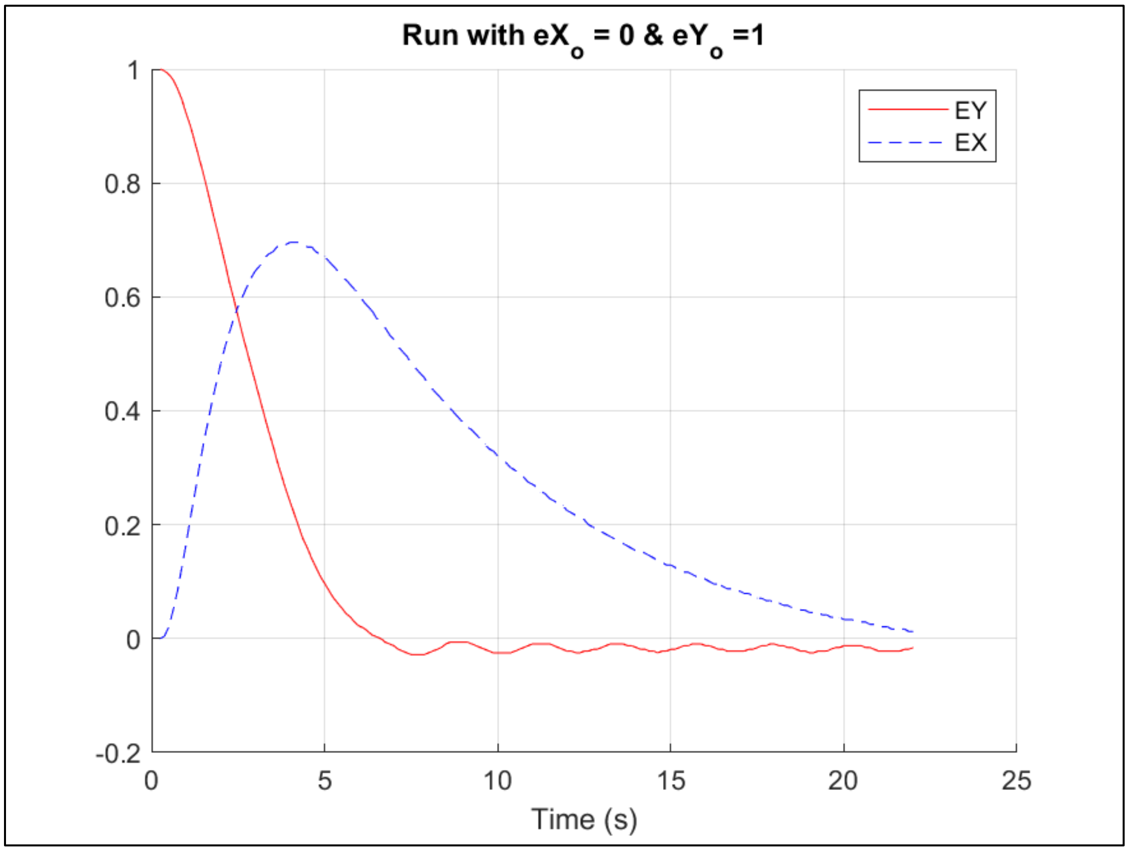


Figure 4.5: Error Functions of Second Simulation

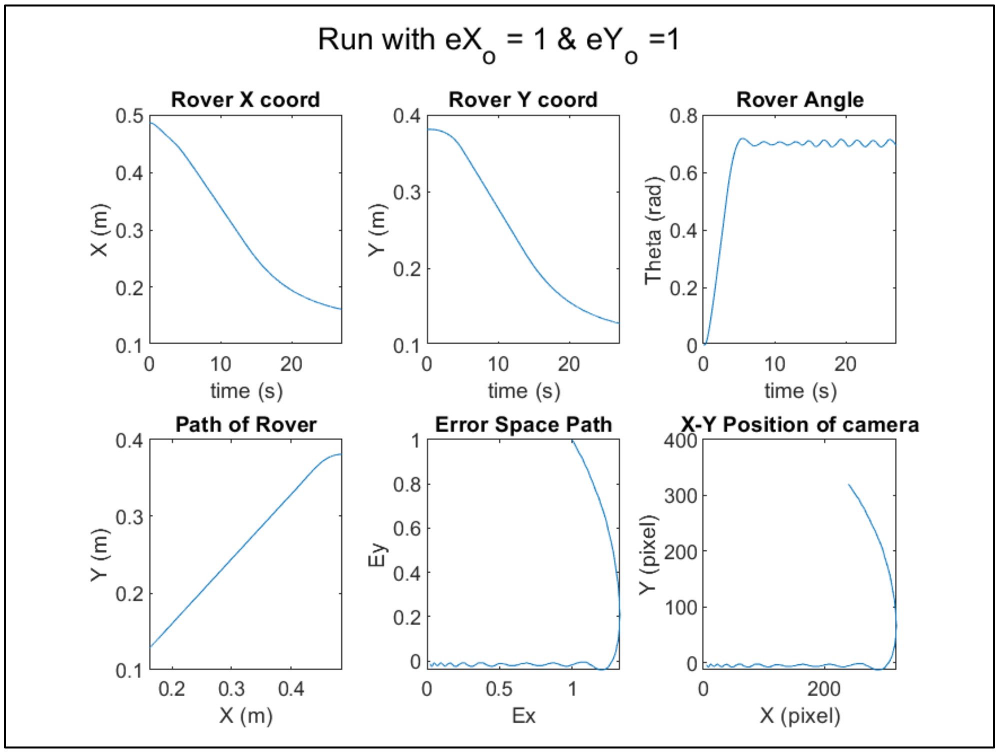


Figure 4.6: The Results of the Third Simulation

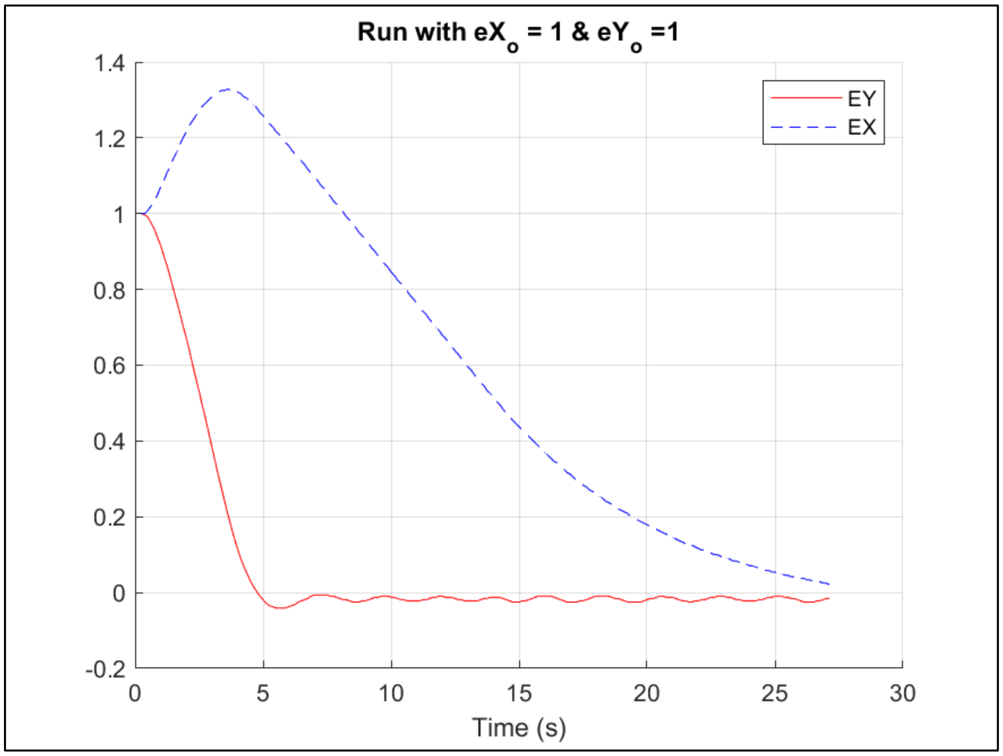


Figure 4.7: The Error Functions for the Third Simulation

4.6 Summary

In this chapter, the control algorithm was explained. In the feedback loop, the following functions such as error functions the negative feedback functions and discrete PI implementation was defined. Also, the error handling of the rover was explained and the deadband compensation was designed. The overall controller is implemented in the MATLA/Simulink simulation. Finally, the simulation results demonstrate that the designed discrete PI controller will deliver the satisfactory results on forward-backward position correction and right-left orientation correction, both independently and when necessary, simultaneously.

Chapter 5 MWR Implementation & Results

The designed controller in Chapter 4 needs to be deployed in MWR in order to guide the robot to the target. In this chapter, a real-time implementation of the designed control system will be done in the on-board computer of MWR. The experimental tests will be carried out to validate the effectiveness of the designed controller.

The overall signal flow diagram of the control system for the MWR is shown in Figure 5.1 and the results obtained in the experiments will be described in this chapter.

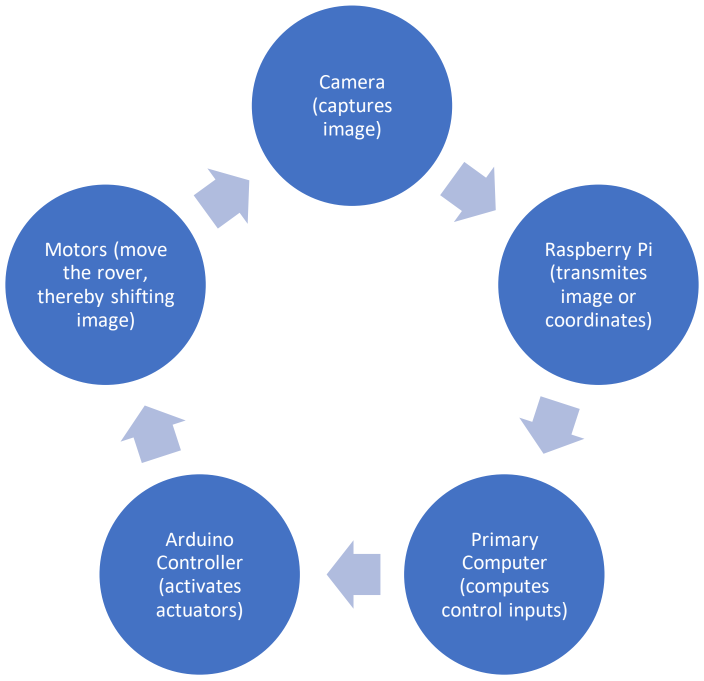

Camera (captures image)
Raspberry Pi (transmites image or coordinates)
Primary Computer (computes control inputs)
Arduino Controller (activates actuators)
Motors (move the rover, thereby shifting image)

Figure 5.1: Rover Feedback Diagram:

5.1 Arduino Microcontroller

The Arduino microcontroller is loaded with a script which implements a finite state machine which changes state based on inputs from the primary computer transmitted over Bluetooth connection to an HC06 and received by the microcontroller over its built-in serial interface. The microcontroller code begins every iteration by checking if there is any information to be received from the serial, if there is it parses it and implements the corresponding changes to the state variables. It then sends signals to the actuators based on the current control loop. There are several possible control loops within the finite state machine, as seen in Figure 5.2, the following is a summary.

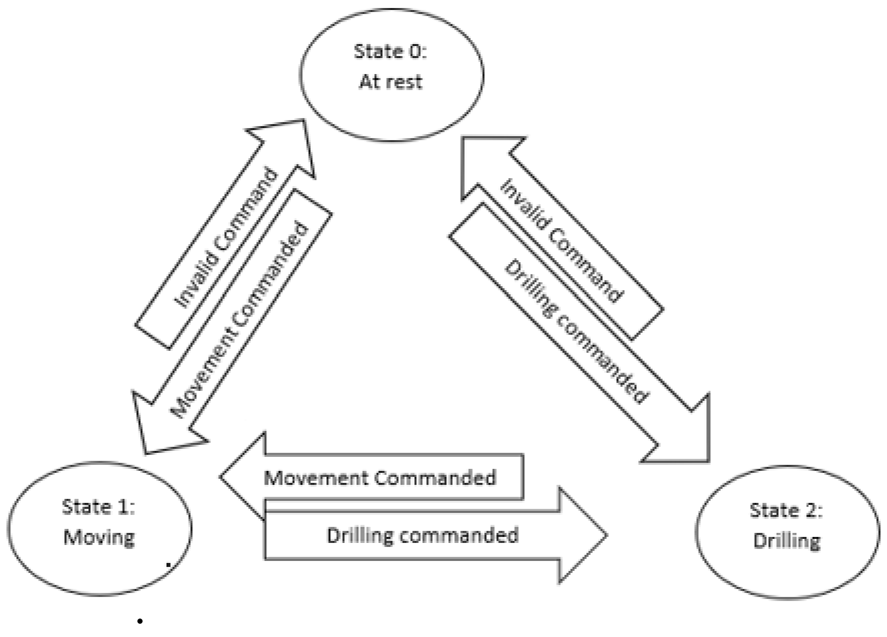


Figure 5.2: The FSM Representation of the Arduino Controller States

5.1.1 Control Loop 0 -At Rest

The default control loop when the microcontroller first boots up is control loop 0. In this control loop, the microcontroller calls a digital write of 0V on all the magnitude pins, thereby disconnecting them from the PWM clock and ensuring they are low. The system leaves this mode whenever it receives a command to activate any of the actuators. This mode is entered whenever an unexpected or incorrect command is received via the serial interface. It also enters this state and waits for 5 seconds before parsing any other input if the rover receives a “pause” command. The pause command was chosen to be the value ten, or the newline character, on the basis that the default settings for the serial terminal include a newline character at the end of every transmission. The rover is meant to receive a continuous stream of commands from the primary computer, but it can also be commanded manually via the serial terminal. However, this is tricky to do in real time and once a velocity is commanded it may be difficult for a user to type the counter command before a collision or other undesirable event occurs, thus adding a pause at the end of every transmission by default seemed to be a good precaution.

5.1.2 Control Loop 1 -Movement

The movement control loop is control loop 1. In this control loop, the drill end effector is set at zero power while the wheels position it above the target. The system enters this state whenever it receives the movement command character, an “m”, over the serial. It then attempts to receive three additional bytes, corresponding to the direction settings of the motors (either both forward, both backward, the left forward and the right backward, or the right forward and the left backward), the magnitude of the righthand motors, and the magnitude of the lefthand motors. If three bytes are not received, or the direction byte is not a valid option, the system rejects the request to enter control loop 1 and enters control loop 0, the resting state, instead. It also reports back an error over the serial interface. If the three bytes are received and the direction byte is an acceptable option, the rover will enter control loop 1 and begin movement, with the direction pins set at the corresponding high or low levels and the magnitude pins for the motors set at the appropriate PWM levels.

5.1.3 Control Loop 2 -Drilling

The drilling control loop is control loop 2. When the control loop is set to two, the power to the wheels will be set at zero. The system enters control loop 2 whenever it receives a request to either set the power to the drill or alter the desired position of the stepper motor for raising or lowering the drill. The request for modifying the drill power is a “d” followed by either a “+” or “-“ which correspond to either CW or CCW and a magnitude byte corresponding to the desired PWM value. If the second byte is not a “+” or “-“ the request is rejected and control loop 0 is entered, with an error message sent back to the primary computer over the serial.

The stepper is controlled by keeping a counter of how many steps should be taken, with negative values corresponding to the opposite direction as positive ones. In every call of the loop function, if the system is in control loop #2 and the value is nonzero, it will either increment or decrement once to make it closer to zero and then (based on its sign) either set the stepper direction pin high or low and toggle the step pin twice. When moving up the system will check if the upper limit limit-switch is pressed before toggling the pin. The stepper commands begin with an “s”, which will put the system into Control Loop #2, and then either increase, decrease, or zero the counter, depending on the second byte. If there is no second byte, it will enter control loop #0.

5.1.4 Other Microcontroller Commands

Two miscellaneous microcontroller commands were also implemented. If the microcontroller receives an “a” over the serial when checking for commands it will send to the primary controller a number corresponding to how may milliseconds have passed since it was last powered on. This is mostly useful for verifying that the Bluetooth connection is working when manually debugging. The second command is an “L”, which causes the microcontroller to report its current control loop and the state of the stepper, this is also used for debugging purposes.

5.2 Microcontroller Communication Driver

The microcontroller communication is responsible for sending the control signals to the microcontroller. Following is a brief summary of its functionality.

5.2.1 Communication with Controller

The Microcontroller Communication Driver communicates with the controller via a shared memory buffer called "shared_memory1". The program will wait for the controller to set the first byte of the buffer to "a" to indicate that it is active and the second byte to "b" to indicate that the buffer contains valid rotation/translation values. The 5th through 12th bytes contain a double precision floating point number corresponding to the desired rotation command (with 0.0 being no rotation, 1.0 being maximum rotation in the CCW direction, and -1.0 being maximum rotation in the CW direction), and the 13th through 20th bytes containing a double precision floating point number corresponding to the desired translation value (with 0.0 being no translation, 1.0 being maximum translation forward, and -1.0 being maximum translation backward). The memory buffer is populated either by the controller program or by a dummy program for debugging purposes.

5.2.2 V-W to R-L

The Microcontroller Communication Driver calculates the right and left wheel PWM velocities in accordance with equations 13 and 14. Noting that a PWM value of 255 is equivalent to 100% of available electrical power, the right and left values are calculated following the following algorithm:

The steps for calculating the movement input to the rover are shown as follows:

$$\begin{gathered} R := 255 * V - 255 * W; \\ L := 255 * V + 255 * W; \\ If(R < 0)\colon R := -R;\ FlagR := true; \\ If(L < 0)\colon L := -L;\ FlagL := true; \\ if(R > 255) R := 255; \\ if\ (L > 255) L \coloneqq 255; \end{gathered} \tag{5.1}$$

where R is the right wheels' Cytron board PWM value (initially calculated as a 32bit integer, but sent as an 8-bit integer), L is the left wheels' Cytron board PWM value, FlagR is a flag for if the right wheels are to be moving backwards, and FlagL is a flag for if the left wheels are to be moving backwards. FlagR and FlagL are used to set the direction byte in accordance with the following table:

Table 5.1: Direction Byte Behaviours

Direction Byte Value	FlagL	FlagR	Instructions when received:
0	True	True	Set both direction pins HIGH
1	True	False	Set left direction pin HIGH, right direction pin LOW
2	False	True	Set left direction pin LOW, right direction pin HIGH
3	False	False	Set both direction pins LOW

Other (4-255)	?	?	Enter resting state, send error message.

5.2.3 Communication with Microcontroller

The microcontroller communication driver keeps track of the keyboard input from the user and what it expects the control loop to be to decide if it wishes to send the movement commands (based on the shared buffer), drilling commands (currently based on the user input), or the miscellaneous commands (also based on user input). It sends the appropriate command bytes over a virtual COM port to the Bluetooth device.

5.3 Raspberry Pi Transmitter

The Raspberry Pi, also referred to as the onboard computer, communicates with the primary computer using UDP over WiFi. In order to allow direct communication over the WiFi hardware, as well as due to Concordia's WiFi firewalls, the primary computer activated a mobile hotspot which the onboard computer connected with. The communication occurred over Port 24, which was found to be available. The controller program spawns a thread which acts as the "listener" for transmissions to port 24, and updates the inputs to it, as will be elaborated on later. Two versions of this program were developed, the first one aimed to transmit the image to the primary computer for processing, the second version aimed to transmit the pixel coordinates of the target to the primary computer.

5.3.1 Image Transmitter Program

The image transmitter program was designed to send the image to the primary computer for processing. It would read the image using OpenCV's drivers and would begin each transmission by reading in the image from the camera 3 times, so that it would read and discard the buffer images used by OpenCV and get to the most recent result from the camera. The image was then down sampled it by a factor of 2 in each direction in order to make it smaller. This was done because the features that the primary computer was using for image processing were not very high frequency, and therefore transmitting the high-resolution version of the image was deemed to be a waste of bandwidth.

The image was RGB, and after being down sampled had a resolution of 320 x 240, for a total of 230400 bytes. The transmission of an image was broken down into 30 steps, each step involving 30 transmissions. Between each step the transmitter would pause for 1.5 milliseconds, to give the primary computer time to process the transmissions in the UDP buffer, and not cause buffer overruns, which would result in lost data. Each transmission consisted of 10 bytes of metadata, and 256 bytes of image data. The first and last byte of metadata were used to ensure that the software versions on both ends (transmitter and receiver) matched. The 4th and 5th bytes were used to record which step and transmission number the block of image data came from, as UDP is an asynchronous protocol and without this information the image can not be properly reconstructed. The remaining bytes of metadata were left for any other information that would need to be transmitted if the need arose.

This approach allowed the primary computer to reconstruct an image of what the camera was seeing, with part of the image being updated as the data became available. While it made it easier to work on the control algorithm, as the image itself was available on the primary computer, it had an image refresh rate of about 3Hz, due to the WiFi hardware limitations of a Raspberry Pi.

5.3.2 Coordinate Transmitter Program

Due to the limitations of the image Transmitter Program, an alternative had to be developed. The alternative did the image processing on the onboard computer and send the pixels where it located the target to the primary computer.

Like the Image Transmitter, the Coordinate Transmitter read in 3 images before transmitting anything. It would then down sample the image to 320 x 240, apply an RGB to HSV transformation, and apply an HSV mask to identify the pixels which matched the color of the target and produce a binary image. It would then dilate the binary image once, and erode it three times, to reduce the impact of noise. It would then take the centroid of the binary image to send to the primary controller. Each transmission to the primary controller consisted of 10 bytes, consisting of 2 bytes for verifying that the programs on each end were using the version intended for each other, 1 byte for fault flags, 2 bytes for the x-coordinate of the pixel location of the target and 2 bytes for the y-coordinate of the pixel location of the target. The remaining 3 bytes were left in case some other data needed to be sent. If the onboard computer detected less than 5 bits in the binary mask that matched the criteria and remained after the erosions, it wouldn't bother updating the centroids and would instead set the first bit of the fault byte to 1, which meant that no target could be acquired.

The coordinate transmitter program increased the refresh rate of the inputs to the primary computer's controller from ~3Hz to ~8Hz. While this did make the controls more stable, it also made it harder to tune the HSV mask parameters when adjusting to different lab lighting. To minimize this "cost", the program used OpenCV sliders to adjust the HSV ranges to allow tuning to different environments without rebuilding the code and would display both the image it captured and the mask results, to facilitate live tuning.

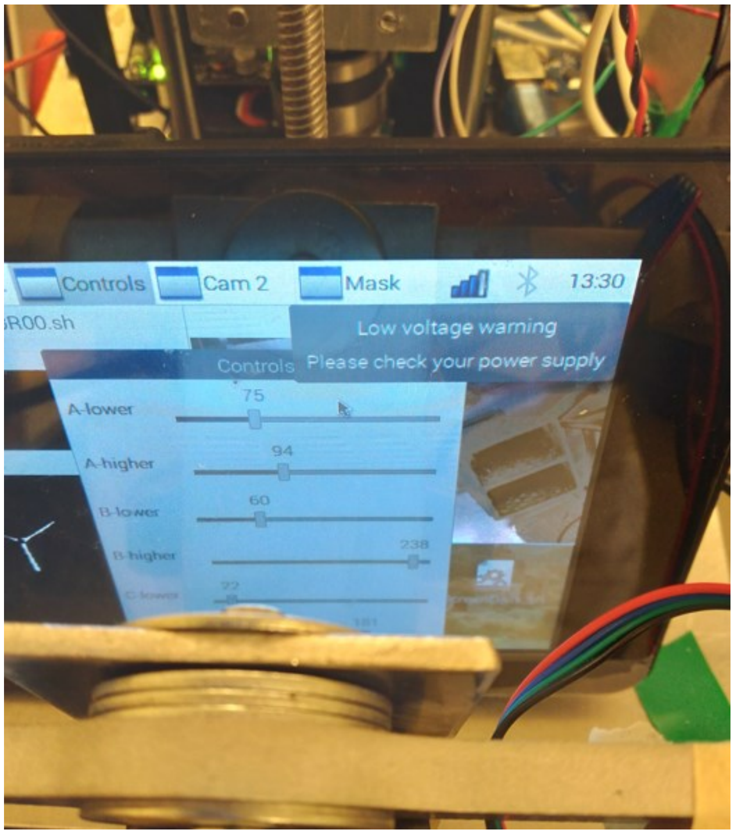


Figure 5.3: The HSV range tuning interface

5.4 Primary Computer Receiver

As noted above, the primary computer control program spawned a CPU thread to listen for transmissions from Port 24 and update its inputs. The receiver thread and the primary controller when designed as complementary to the onboard computer transmitter program, thus for both of the transmitter programs a corresponding receiver and controller was developed.

5.4.1 The Image Receiver

The image receiver program used an OpenCV 240 x 320 RGB image as its input. The main program would begin by creating a shared memory buffer to communicate with the communication driver, and a UDP connection to receive information from the onboard computer. It would then spawn the receiver thread and begin functioning as the controller until it would detect a keyboard input to stop.

When the receiver thread was spawned, it would be passed 3 references to variables within the main program, namely a reference to a boolean variable called “keepRunning” which would be set to false when it was time for it to stop running, a reference to the OpenCV image used as input, and the socket used for communication. As long as the “keepRunning” variable was true, the receiver thread would check if there was any information in the UDP receive buffer, and if there was it would verify from the 10 bytes of metadata if it was data meant for that version of the receiver, and if yes it would calculate where to copy that data within the image based on the step and transmission numbers.

In parallel with the receiver, the main controller program would be running. It would start every iteration by making a copy of the input image. It would then blur the image, apply an RGB to HSV transformation, and then erode the resulting binary map twice. If it would find more than five pixels left in the binary map it would calculate their centroid, otherwise it would set the “wait” flag to true, which would cause the controller to send a command of no motion in either axis to the communication driver. If the wait flag was not set to true, the program would then check if the centroid was more than 20 pixels away from the target location. If yes, it would write a translation and rotation command proportional to the distance from the target location to the communication driver shared memory buffer.

5.4.2 The Coordinate Receiver

The coordinate receiver used the coordinates of the centroid of the target as its input. Like the image receiver, the main program would begin by creating a shared memory buffer to communicate with the communication driver, and a UDP connection to receive information from the onboard computer. It would then spawn the receiver thread and begin functioning as the controller until it would detect a keyboard input to stop.

When the receiver thread was spawned, it would be passed 3 references to variables within the main program, namely a reference to a boolean variable called “keepRunning” which would be set to false when it was time for it to stop running, a pointer to an array of integers, and the socket used for communication. As long as the “keepRunning” variable was true, the receiver thread would check if there was any information in the UDP receive buffer, and if there was it would verify from the first and last of the 10 bytes received if the data was valid and was meant for that version of the receiver. It would then copy the byte for fault flags to the third integer in the array, calculate the horizontal/vertical position of the target from the values transmitted and write them to the first and second values in the array, and record the time that the transmission was received/processed in the fourth value in the array.

In parallel with the receiver, the main controller program would be running. It would start every iteration by copying from the input array the values of the fault flags and the transmitted horizontal/vertical coordinates. If the fault flags indicated that the target was not found, the “wait” flag would be set to true, which would cause the controller to send a command of no motion in either axis to the communication driver. If the wait flag was not set to true, the program would then check if the centroid was more than 20 pixels away from the target location. If yes, it would write a translation and rotation command proportional to the distance from the target location to the communication driver shared memory buffer.

5.4.3 The Recorder

In order to record the test results the coordinate receiver program was modified to add recording capabilities. An array was added to the main function which stored all the parameters to be recorded, which would be overwritten every iteration. A recording function was added which would be called and spawned as a new thread by the recorder controller, which was activated via an OpenCV trackbar.

The recording function would receive a reference to a Boolean, a pointer to an array of doubles, and a reference to a C++ standard output file stream. As long as the Boolean would be true, the recording function would write the first 15 contents of the array to the file output stream, using the CSV format. In order to prevent too many redundant recordings, a five-millisecond delay was added to the recording function.

An OpenCV trackbar was added with a “Record” option ranging from 0 (OFF) to 1 (ON), which when toggled would call a recorder controller function which would either spawn or terminate the recorder thread. When called with a “1”, the recorder controller would set a static Boolean called “dontStop” to true, create a new file containing the file number (a static integer initialized to 0) within its name (which would be stored in a static C++ ofstream), increment the file number, and spawn a recording thread (with a pointer to it stored in a static C++ std thread pointer), which would be passed a pointer to the recorded parameters array, a reference to the “dontStop” Boolean, and a reference to the file output stream. When called with a “0”, the recorder controller would set the “dontStop” flag to false, wait for the recording thread to terminate, and close the filestream. In this way it was possible to turn on and off the recording to isolate specific maneuvers.

5.4.4 Test Results

Several tests were performed using the configuration described above (for videos, click here). In Figure 5.4, the results for a test with a primarily translational response is shown. In Figure 5.5, the results for a combined translational-rotational response are shown. In the Figures, V and W are the commanded velocities and rotations, as percentages of maximum available voltage, and X_error and Y_error are the errors recorded by the controller, calculated using equations (4.1) and (4.3).

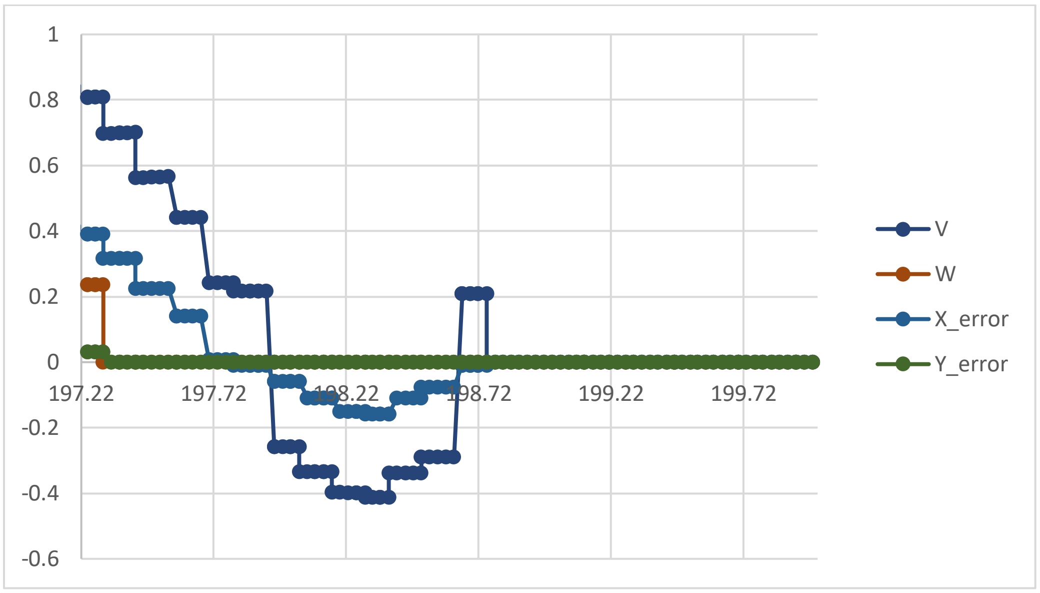


Figure 5.4:Translational Response Test Result of Rover

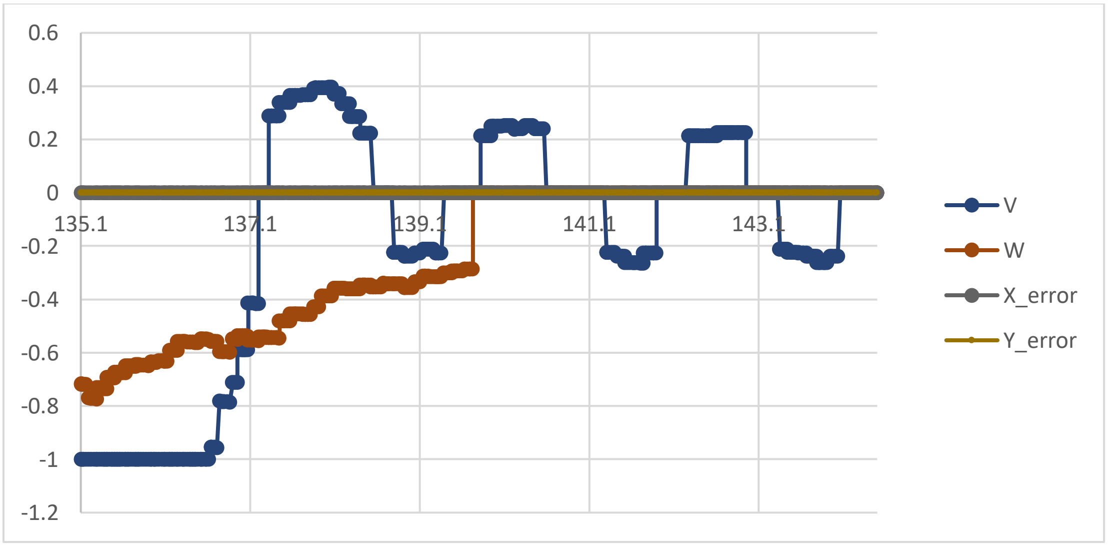


Figure 5.5: Combined Translational-Rotational Response Test Result of Rover

As can be seen in the results, the rover is capable of positioning fairly accurately, though the noise from the camera sometimes destabilizes the result a little. Although it seems like it could use a slight reduction in the offset used in equation (4.9), overall the controller works to converge the rover on the target.

5.5 Summary

In this chapter, the control implementation of the rover was described. This included:

- The microcontroller Finite State Machine and its inputs
- The communication driver from the primary computer to the microcontroller
- The two versions of the Raspberry Pi transmitter
- The two versions of the Primary Computer Receiver
- The data recorder thread
- The test results recorded by the data recorder thread when all the above were working properly.

The videos of the tests, as well as integration testing during various phases of development can be seen here:

https://www.youtube.com/playlist?list=PLbab01V_uTiuBkWz_RR5KL2iC7Dgfbgsm

Chapter 6 Conclusion and Future Work

6. 1 Summary of the research work

In this thesis, an MWR has been re-designed to be functional in terms of traversing a garden to seek and position the weeds for removal. The re-design task including the development and improvement of mechanical, electrical and control systems. The summarization of the research work carried out in this thesis is given as follows:

- Designed mechanical additions to the rover including: stilts, motor adaptors, a switch mount, the Cytron board tower and the battery basket;
- Developed electrical additions to the rover including the actuator power circuit, the onboard computer power circuitry, and the Arduino microcontroller power circuit and wiring;
- Built mathematical models of MWR including an ideal continuous model and a discrete more accurate Simulink model
- Designed the control algorithm of the WMR including forward-backward position correction and error detection and handling;
- Implemented discrete PI controller of WMR and dead-band compensation;
- Carried out simulations of the designed controllers in Matlab/Simulation;
- Integrated the designed controller with the improved MWR robotic systems and implement a real time control system;

- Carried out the experimental tests on the designed integrated system to position the rover above a simple mono-chrome target representing the dandelion and achieve satisfactory positioning results.

In conclusion, it was found that a discrete controller working on correcting both orientation and distance at the same time could be used for a rover to position itself above a target.

6.2 Future work

It is noted that the above-mentioned research restricted its scope merely to the positioning implementation once the weed is found and does not include a control algorithm to search for the weed and map the area to be searched. Additionally, the digging aspect of the rover has not been implemented. The rover will require future work including integrating the controller into the on-board computer, implementing a mapping algorithm for seeking out weeds, and replacing the observer with a neural-network based weed recognition system. Future work could also include making the controller more robust by accounting for cross-input effects, and potentially attempting to reduce the impact of the update lag by switching to a PID control model. As well. The use of an integrated inertial measurement unit (IMU) or gyroscope could allow for some feedback directly from the plant, which could be used to better stabilize the readings from the camera.

References

[1] A. Evans, "How Much Does Weed Control Cost in 2024?," Lawn Starter, 29 November 2023. [Online]. Available: https://www.lawnstarter.com/blog/cost/weed-control-price/. [Accessed 10 February 2024].

[2] T. Kalb, "Managing Dandelions In Your Lawn With Herbicide," North Dakota State University, [Online]. Available: https://www.ndsu.edu/agriculture/extension/extension-topics/gardening-and-horticulture/lawn-and-yard/managing-dandelions-your-lawn. [Accessed 10 February 2024].

[3] T. E. Arbuckle, S. M. S. D. Cole, J. Hall, C. M. Bancej, L. A. Turner and P. Claman, "2,4-Dichlorophenoxyacetic acid residues in semen of Ontario farmers," *Reproductive Toxicology,* vol. 13, no. 6, pp. 421-429, 1999.

[4] National Institute for Occupational Safety and Health., "The effects of workplace hazards on male reproductive health," *DHHS publication,* pp. 96-132, 1996.

[5] S. Bradbury, "Reregistration Eligibility Decision for Mecoprop-p," EPA, 29 August 2009. [Online]. Available: https://www3.epa.gov/pesticides/chem_search/reg_actions/reregistration/red_G-53_29-Aug-07.pdf. [Accessed 18 February 2024].

[6] Minnesota Department of Agriculture, "DICAMBA - DAMAGE & COMPLAINTS," Minnesota Department of Agriculture, [Online]. Available: https://www.mda.state.mn.us/dicamba-damage-complaints. [Accessed 17 February 2024].

[7] *Center for Biological Diversity v. United States Environmental Protection Agency (4:20-cv-00555),* 2024.

[8] Z. Li, T. Shao, H. Min and X. X. Zhenmei Lu, "Stress response of Burkholderia cepacia WZ1 exposed to quinclorac and the biodegradation of quinclorac," *Soil Biology and Biochemistry,* vol. 41, no. 5, pp. 984-990, 2008.

[9] "Tertill.com," Tertill, [Online]. Available: https://tertill.com/blogs/tertillblog/home-gardening-made-easy-with-tertill-the-high-tech-gardening-robot. [Accessed 3 December 2023].

[10] T. (. Fields), "community.twistedfields.com," Twisted Fields, 1 February 2021. [Online]. Available: https://community.twistedfields.com/t/introducing-acorn-a-precision-farming-rover-from-twisted-fields/17. [Accessed 27 December 2023].

[11] S. Gokul, R. Dhiksith, S. A. Sundaresh and M. Gopinath, "Gesture Controlled Wireless Agricultural Weeding Robot," in *5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, Coimbatore, India, 2019.

[12] I. Babiker, "Dandelion Weed Detection and Recognition for a Weed Removal Robot," Concordia University, Montreal, 2020.

[13] Y. F. a. X. Z. Boaquan Li, "Visual Servoing Regulation of Wheeled Mobile Robots With an Uncalibrated Onboard Camera," *IEEE/ASME Transactions On Mechatronics,* vol. 21, no. 5, pp. 2330-2342, 2016.

[14] W.-F. Xie, Z. Li, X.-W. Tu and C. Perron, "Switching Control of Image-Based Visual Servoing With Laser Pointer in Robotic Manufacturing Systems," *IEEE Transactions on Industrial Electronics,* vol. 56, no. 2, pp. 520-529, 2009.

[15] B. Ahmadi, E. Zakeri and W.-F. Xie, "Optimal Image-Based Task-Sequence/Path Planning and Robust Hybrid Vision/Force Control of Industrial Robots," *IEEE Access,* vol. 10, pp. 26347 - 26368, 2022.

[16] T. Shu, S. Gharaaty, W. Xie, A. Joubair and I. A. Bonev, "Dynamic Path Tracking of Industrial Robots With High Accuracy Using Photogrammetry Sensor," *IEEE/ASME Transactions on Mechatronics,* vol. 23, no. 3, pp. 1159 - 1170, 2018.

[17] R. Mebarki, A. Krupa and F. Chaumette, "2-D Ultrasound Probe Complete Guidance by Visual Servoing Using Image Moments," *IEEE Transactions on Robotics,* vol. 26, no. 2, pp. 296 - 306, 2010.

[18] D. Zheng, H. Wang, J. Wang, X. Zhang and W. Chen, "Toward visibility guaranteed visual servoing control of quadrotor UAVs," *IEEE/ASME Transactions on Mechatronics,* vol. 24, no. 3, pp. 1087 - 1095, 2019.

[19] R. S. Sharma, S. Shukla, L. Behera and V. K. Subramanian, "Position-Based Visual Servoing of a Mobile Robot with an Automatic Extrinsic Calibration Scheme," *Robotica,* vol. 38, no. 5, pp. 831 - 844, 2020.

[20] X. Zhang, Y. Fang, B. Li and J. Wang, "Visual Servoing of Nonholonomic Mobile Robots With Uncalibrated Camera-to-Robot Parameters," *IEEE Transactions on Industrial Electronics ,* vol. 64, no. 1, pp. 390 - 400, 2017.

[21] B. Li, X. Zhang, Y. Fang and W. Shi, "Visual Servoing of Wheeled Mobile Robots Without Desired Images," *IEEE Transactions on Cybernetics ,* vol. 49, no. 8, pp. 2835 - 2844, 2018.

[22] W. Fujie, Y. Qin, F. Guo, B. Ren and J. T. Yeow, "Adaptive Visually Servoed Tracking Control for Wheeled Mobile," *Complexity,* pp. 1-13, 2000.

[23] X. Liang, H. Wang, W. Chen, D. Guo and T. Liu, "Adaptive Image-Based Trajectory Tracking Control of Wheeled Mobile Robots With an Uncalibrated Fixed Camera," *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY,* vol. 23, no. 6, pp. 2266-2282, 2015.

[24] X. Liang, H. Wang, Y.-H. Liu, W. Chen and Z. Jing, "Image-Based Position Control of Mobile Robots With a Completely Unknown Fixed Camera," *IEEE Transactions on Automatic Control,* vol. 63, no. 9, pp. 3016-3023, 2018.

[25] X. Zhang, Y. Fang and N. Sun, "Visual servoing of mobile robots for posture stabilization: from theory to experiments," *International Journal of Robust and Nonlinear Control,* vol. 25, pp. 1-15, 2015.

[26] F. Yan, B. Li, W. Shi and D. Wang, "Hybrid Visual Servo Trajectory Tracking of Wheeled Mobile Robots," *IEEE Access,* vol. 6, pp. 24291-24298, 2018.

[27] X. Zhang, Y. Fang and X. Liu, "Motion-Estimation-Based Visual Servoing of Nonholonomic Mobile Robots," *IEEE Transactions on Robotics,* vol. 27, no. 6, pp. 1167-1175, 2011.

[28] K. Zhang, J. Chen, Y. Li and Y. Gao, "Unified Visual Servoing Tracking and Regulation of Wheeled Mobile Robots With an Uncalibrated Camera," *IEEE/ASME Transactions on Mechatronics,* vol. 23, no. 4, pp. 1728-1739, 2018.

[29] A. Martinez, "Grab CAD: Cytron 30A DC Driver," 12 April 2020. [Online]. Available: https://grabcad.com/library/cytron-30a-dc-driver-1. [Accessed 2 January 2024].

[30] J. L. L. F. Q. Z. K. Y. J. W. C. S. L. H. A. Z. W. XINYU GAO, "Review of Wheeled Mobile Robots' Navigation Problems and Application Prospects in Agriculture," *IEEE Access,* vol. 6, pp. 49248 - 49268, 2018.

[31] J. Moreno-Valenzuela, L. Montoya-Villegas, R. Perez-Alcocer and J. Sandoval, "A Family of Saturated Controllers for UWMRs," *ISA Transactions,* vol. 100, pp. 495-509, 2020.

[32] R. C. Gonzalez and R. E. Woods, Digital Image Processing, London: Dorling Kindesley Pearson Education, 2007.

[33] "Robot Shop," ES Motor, [Online]. Available: https://ca.robotshop.com/products/32d-planetary-gearmotor-12v-24-rpm. [Accessed 28 December 2023].

[34] Y. F. X. Z. Baoquan Li, "Visual Servo Regulation of Wheeled Mobile Robots With an Uncalibrated Onboard Camera," *IEEE/ASME TRANSACTIONS ON MECHATRONICS,* vol. 21, no. 5, pp. 2330-2342, 2016.

[35] X. GAO, J. LI, L. FAN, Q. ZHOU, K. YIN, J. WANG, C. SONG, L. HUANG and a. Z. WANG, "Review of Wheeled Mobile Robots' Navigation Problems and Application Prospects," *IEEE Access,* vol. 6, pp. 49248 - 49268, 2018.