2016

Autonomous navigation using image processing

Sivanesu, Ajantharojan

Autonomous Navigation Using Image Processing

Prepared by
Ajantharojan Sivanesu

Department of Electrical and Computer Engineering
Lakehead University

Thunder Bay, Ontario, Canada
October 24, 2015

Abstract

In the recent years, the pursuit intelligent and self-operated machines have increased. The human user is to be completely eliminated or minimized as much as possible. It is also popular now to observe machines that are able to do variety of tasks, instead of just one.

A division of intelligent robotics is autonomous navigation. Google, Tesla, Honda, and many other large corporations are trying to master this field, since the search for a self-driving vehicle is profitable as much as it is difficult. The research presented in this thesis is autonomous navigation for mobile robots in an indoor environment, but not limited to.

Some explored algorithms focus on navigating in environment that has been already explored and mapped. Algorithms such as modified A-Star and goal-based navigational vector field were tested for how effectively a path is planned from one point to another. The algorithms were compared and analyzed for how well the robot avoided obstacles and the length of the path taken. Other algorithms were also developed and tested for navigation without a map.

The navigational algorithms are simulated on different artificial environments as well as on real environments. Machine learning is used to learn and adapt to the robot's motion behaviours, which enables the robot to perform movements as intended by the implemented navigational algorithms. Referred to in this thesis as the intelligence engine, a feed-forward artificial neural network was created to predict power delivery to the motors. Back-propagation algorithm is used alongside the neural network to enable supervised learning.

Similar to human vision, the algorithm relies mainly on image processing to obtain data about the surrounding environment. The data human eyes provide helps one perceive and understand the surroundings. Similarly, a Kinect sensor is used in this thesis to get 2-dimensional colour data as well as depth data.

A program was implemented to process and understand this arbitrary sequential array of numbers in terms of quantifiable values. The robot in return is capable of understanding target, obstructions, and is capable of navigation. All external data are gathered from one optical sensor. Many different algorithms were implemented and tested to efficiently detect and track a target. The idea is to make an artificial robot perceive its' surrounding using 3-Dimentional image data and intelligently navigate the local surroundings.

Acknowledgements

I would like to thank my supervisor, Dr. X. Liu, for all the help and freedom he had given me with my research. Thank you for listening to my ideas and allowing me pursue my solutions. Thank you for coming in on the weekends to help me with any problems I may have.

I would like to thank my parents for encouraging me and supporting me throughout this endeavour. Aiming high and pursuing my goals is a trait nurtured in me by my mother. Thinking practically and applying my self is what I learned from my father. Without these qualities, I would have never accomplished what I have thus far. These qualities will always be a part of me forever and I am forever grateful.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
OF	Optical Flow
LK	Lucas Kanade
IED	Improvised Explosive Devices
AI	Artificial Intelligence
RGB	Red, Green, Blue
BGR	Blue, Green, Red
3D	Three-Dimension
IR	Infrared
NUI	Natural User Interface
SDK	Software Development Kit
FPS	Frames per second
FR	Frame Rate
PSS	Prime Sense Sensor
ICC	Intel's Creative Camera
CMOS	Complementary Metal-Oxide Semiconductor
DSP	Digital Signal Processor
CPU	Central Processing Unit
RAM	Random Access Memory
GPU	Graphics Processing Unit
CPC	Central Processing Computer
ASCII	American Standard Code for Information
PWM	Pulse Width Modulation
CRC	Cyclic Redundancy Check
GUI	Graphical User Interface
SSE	Steady-State-Error
PID	Proportional-Integral-Derivative
FIE	Fuzzy Inference Engine
COA	Center of Average
MKSDK	Microsoft Kinect Software Development Kit
API	Application Program Interface
ICR	Instantaneous Center of Rotation
GBNVF	Goal-Based Navigational Vector Field
MOBD	Manhattan Obstacle-Based Distance
ANN	Artificial Neural Network
FF	Feed-Forward
FFANN	Feed-Forward Artificial Neural Network
BP	Back-Propagation
GPGPU	General Purpose Graphic Processing Unit

Chapter 1
Introduction

1.1 Idea and Motivation

Autonomous navigation is one of the major research fields in robotics of this decade. It is the idea of self-operated machine that can intelligently navigate a given environment. Majority of the machines used today rely on multiple sensors and feed-back designs. Autonomous navigation can be achieved through simple proximity sensors, which return a value of high if an object is in the range and low if no object is detected. However, a more robust design would use highly accurate sensors that can better detect and understand the surrounding environment. For example, the navigation can be guided by precise satellite information about the surrounding, or multiple high precision sensors attached on the robot that can communicate with external sensors from the surroundings. Each such intelligent machine has their unique problems and challenges. Usually a machine that is able to perform at higher velocities requires expensive, precise, sensors and processors as well as an efficient algorithm.

In 2012, Sergey Brin, founder of Google, announced that the self-driving car will be available in 2017 for the general public. Elon Musk, chief executive of Tesla Motors, announced that an auto-pilot mode will be available in their cars by the summer of 2015. Many other major automotive corporations are taking large steps towards autonomously driving vehicles. Autonomous driving is yet another step towards intelligent technology that is able to adapt and perform an array of tasks that ensure a successful navigation from one point to another. On a smaller scale, there are devices for warehouses and private homes that are meant to better ease one's everyday life. One such technology is the Roomba vacuum from iRobot; it is a vacuum that is self-operated to clean the floor. It will navigate around an environment such as a room in a house and clean dirt and dust off the floor using sensors that can detect collision such as proximity sensors.

Another area this type of technology is used is in the military. The military has been pursuing autonomous technology for some time now. icasualities.org, stated that in 2010, the United States military took 368 fatal casualties just from IED. To deliver supplies, transportation, bomb disarming, armed warfare, and many other purposes; the military is in dire need for improvements in autonomous navigation. The idea is to go into areas and environments that are too dangerous for humans to enter and/or perform tasks. Now Imagine an unmanned, intelligent robots doing such tasks as part of a regular routine with no supervision. This will save many soldiers' lives and it will be much cheaper than human soldiers for the military to maintain. Many of this already exists in the military, but there is always room for improvements and new novel ideas that can add a new layer of efficiency to the idea.

This thesis is intended to investigate topics for a small scale, indoor, mobile robot but, not limited to. The mobile robot is intended to autonomously navigate an environment using mainly image processing achieved by the optical sensor data. Having one sensor instead of multiple sensors is a cost and energy efficient solution to autonomous navigation. However, such a design will need complex algorithms to properly function since one sensor is responsible for all necessary data that can be utilized. To achieve this, the final product will integrate multiple aspects from different fields of research. The topics consisted of choosing the correct optical sensor, tracking and detecting, hardware design, controller design, navigation algorithms, and artificial intelligence.

The topics explored can be implementable on large scale projects with additional configuration changes. Each topic can be explored individually for better efficiency; however, this thesis focused on the integration of the different topic as a whole to achieve autonomous navigation. The overall motivation and idea is to create a pet like robot that will perform simple tasks required by the user. The overall algorithm is designed to replicate simple form of human like behaviour and logic.

1.2 Objectives

The objective of this thesis was to design a robot capable of the following:

A. To detect the environment as a visual representation.

B. To track objects in three dimensional space.

C. To navigate an unknown environment, unsupervised while avoiding collision.

D. To navigate to a target in an environment that is already mapped.

E. To follow or navigate to a target in an un-mapped environment.

F. To have an intelligent system that is capable of learning robot behaviour over-time.

G. To have a user interface that is simple and informative.

H. To have practical and efficient controller designs.

1.3 Organization

First, an introduction on the scope of this thesis was given. Next, background and literature review will be discussed for three major topics involved. The core theory of each topic will be provided at first with mathematical models and/or pseudo code. The theory will be reinforced with novel ideas published by other authors. Chapter 3 will attempt to summarize the final design of the robot. It will also provide information about the setup used to perform the experiments. Chapter 4 consists of all the experimental results and conclusions drawn. There are five Appendix chapters presented in this thesis. Optical sensor, history of robot assembly and design, and controller design and results are all the topics explored in the Appendix chapters. It is important to note that all results were performed and obtained from the same computer unless otherwise stated; the specifications can be found on Chapter 3. Finally, a conclusion will summarize the important findings of the entire thesis, difficulties encountered, and any future work to be done in Chapter 5.

Chapter 2
Background and Literature Review

This chapter will present the background information of three important topics explored in this thesis: optical tracking, navigation, and artificial neural network. Related literature topics will also be analyzed in this chapter.

2.1 Optical Tracking

Tracking is the displacement of focus in the 3-Dimentional (3D) space as well as the displacement in the dimension of time; for if there is no time, there is no motion. Humans and many animals have the ability to follow objects or a region of space. This is no simple task, yet humans do it so effortlessly. Historically, this was a very important ability to have since majority, if not all the primitive hunting relied upon it. To follow animals or a fish and calculate their future trajectory to strike is a skill refined over thousands of years. This skill still plays a major role in everyday life. The ability to track and predict the position of a car on a parallel lane and change lane accordingly, the ability to catch a ball flying through the air, the ability to simply focus on any object in the three dimension, are all examples of human tracking used daily.

Tracking is absolutely necessary when navigating. It is a way to keep reference in the 3D space while in motion. For example an arbitrary table is observed and then the robot has moved in a given direction. If the table appears smaller, the robot has moved away, and if the table appears larger, then the robot has moved closer. If this relation happens when the robot has not moved, it can be concluded that the table has moved instead. Also, if the table is the target that the robot needs to navigate towards, it is crucial to have the ability to "track/follow" that target.

To follow a given target, there are two different ways. One is tracking and the other is actually detecting. Tracking and detecting are not the same thing, even though they may appear to be the same to an observer. Simply stated, tracking is following a region without identifying what that region may be while detecting is the ability to recognize that region as something. For example, the robot will understand that it is not simply following a region

that is pointed towards a table. It will understand that it is following an object previously stored in its reference as a table. However, it typically needs more memory and computation to actually detect than to track. Section 2.1.1 will explore topics which try to replicate the human tracking ability using 2-Dimensional (2D) frames captured in sequence, otherwise known as video tracking. Detection as a form of tracking will be explored in term of Hue, Saturation, Value (HSV) tracking, colour tracking, and feature detection based tracking. Then, motion tracking will be explored in terms of tracking based on change and optical flow.

2.1.1 Colour and HSV Tracking

Colour detection is one of the simpler means of tracking. It is a filtering process where a band-pass filter will only allow certain colour through and block the rest. Therefore, this colour can be tracked and redetected through any sequential or non-sequential set of frames. That is the basic principle behind colour tracking.

An image frame consist of three, unsigned integer, 8-bit frames; each for blue, green, red (BGR). The 8-bit allows each cell to take up a value ranging from 0 to 255. The BGR matrices may be looked at like a tinted window and the ratio of 0 to 255 will be a percentage of the tint in a given area. The combination of the three tints can be seen as an image. Figure 2-1 shows the breakdown of an arbitrary colour frame with a resolution of 25 pixels (5x5). It is an example to illustrate the illusion of a BGR frame from the three separate frames. Each box/cell represents a pixel of the image. With this information, it is easy to see that there will need to be three separate filters for each separate matrix and the combination of the filters will result in one final filter. However, a more suitable method is using HSV segmentation. It is very similar to BGR, but it has three matrices of hue, saturation, and value, instead of blue, green, and red.

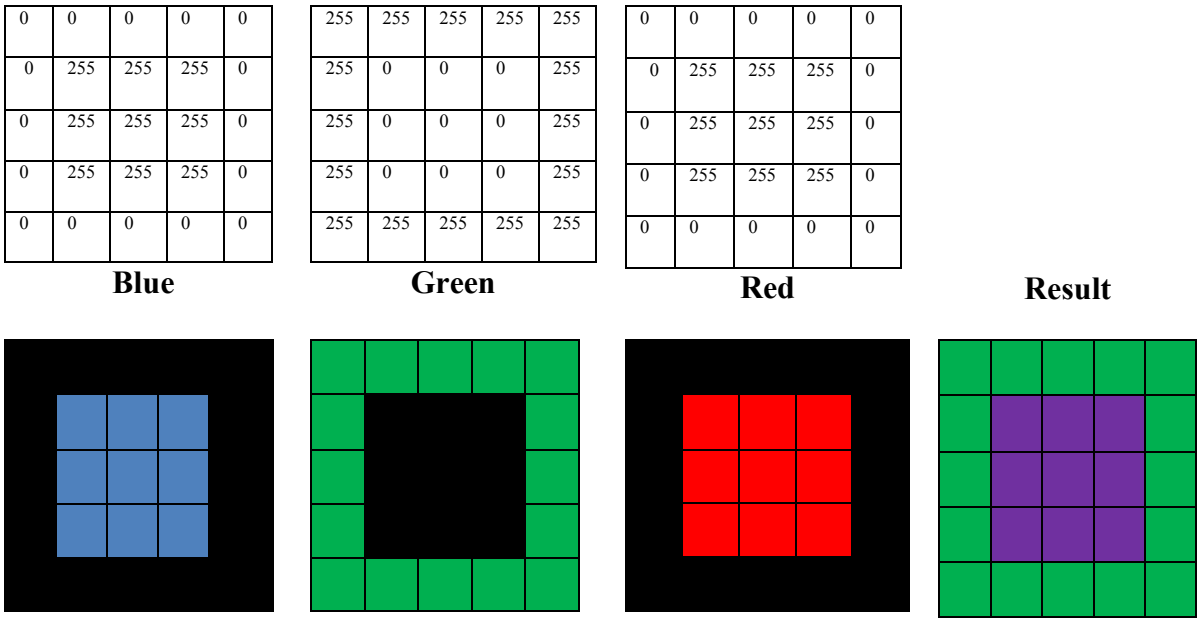


Figure 2-1: Break-down of an arbitrary BGR image.

Hue represents the colour, which ranges from 0 – 179. Saturation represents amount of white that is mixed with that colour, which ranges from 0 – 255. Value represents the amount of black that is mixed with that colour, which ranges from 0 – 255. Hue colours range as follows: orange is from 0-22, yellow is from 22 – 38, green is from 38 -75, blue is from 75 – 130, violet is from 130 – 160, and red is from 160 -179. With the help of EMGUCV library, this process is can be coded as shown on Figure 2-2. Note that there were two sliders placed for each filter (H, S, V). This is done to have an upper limit and a lower limit for each category, like a band pass filter. “imgProcessed” is the final image where the displayed image will show the target in white and all other region in black; refer to Section 3.1.1.

```
public void HSVimage(Image<Bgr, byte> IMAGE)
{
      Image<Hsv, Byte> HSVImage = IMAGE.Convert<Hsv, Byte>();
      hsvImage = HSVImage;

      Hsv hsv_min = new Hsv(hueSliderMin.Value, satSliderMin.Value,
                    valSliderMin.Value);

      Hsv hsv_max = new Hsv(hueSlider.Value, satSlider.Value, valSlider.Value);

      rangeImage = HSVImage.InRange(hsv_min, hsv_max);
}
```

Figure 2-2: C# code of HSV filtering process.

2.1.2 Feature Detection Based Tracking

Feature detection tracking is essentially a template matching algorithm. It can be done by cropping a targeted area from the previous frame or having a previously stored image and matching that to the current frame and future frames to come. This is no simple task since to a computer, each frame is a set of numbers and one cannot simply try to match a portion of numbers in every frame. Even the smallest angle change or lighting change will offset the numbers drastically and the template will no longer match; even if it is looking at the exact same image from a slightly different view point. This is where "features" play their role.

There is no universal definition for a feature. In this context, features are points in an image that appear to be the unique part of the image and less likely to change. Features are a starting point for many different computer vision applications. It can be said that the effectiveness of detection heavily relies on the algorithm's ability to extract "good" points as features.

There are many different feature detectors proposed by a number of authors. Some of the more popular ones use at least one of the following three classifications; edge, corner, and/or blog. Edge detectors such as Canny, and Sobel detect a set of points which have a strong gradient magnitude [30]. Corner detectors such as Harris and Stephens, Shi and Tomasi, are modified edge detectors. They identify edge in an image and find rapid changes

on that edge to find corners [15]. Blog detection algorithms focus more on identifying local maximums in an image, which detect area in an image that are too smooth to detect by other detectors. Most of the more accurate detectors use a combination of methods.

When the features are detected, they will be stored as feature vectors via feature extraction or dimensionality reduction. It can also be called descriptors since they describe the detected features in a unique form that is easier to identify later. Matching is essentially comparing detected feature descriptors on the current frame with that of a stored model. Therefore, once the target is selected by the user, features must be detected and stored as a descriptor. Every frame thereafter will have its features detected and described in a vector. The two descriptors are then compared via matching, which will yield if the target is in the current frame or not. Note that there are many good detectors and descriptors purposed in the relevant past for accurate detection. Detectors such as FAST [21], and SURF [26] and descriptors such as SIFT [18], SURF [26], BRIEF [43], ORB [22], BRISK [51], and FREAK [1]. SURF showed the most accurate performance in different experiments done to test accuracy, stability, and compatibility [43, 47, 40].

2.1.3 Tracking Based on Change

Tracking based on change is applicable when there is change in the content through consecutive frames. It is essentially a way of tracking motion using sequential frames. The algorithm looks at two consecutive frames and identifies any differences found between them. Pseudo code of the algorithm is illustrated on Figure 2-3. Note there is a blur applied for noise elimination. Blurring methods and how it is done will be explored later in the chapter. Near the end of the algorithm, there will be a black and white image. Black pixels represent all the area that has no detected motion and white pixels represent all the area that has motion. Even with noise compression, there will always be some false detection with dust and/or other noise that has not been filtered. For this reason, it is important to only detect the largest motion as the accepted motion. To find the largest motion, one must find the largest contour around white blobs as illustrated from step 4b, in Figure 2-3. Contour is the edge or outline of an area. A rectangle is then drawn around the largest contour to show the user the largest motion detected in consecutive frames.

```
1. Get the current frame.

2. Convert to gray frame.
      a. For i=0; i < every pixel multiplied by a factor of
         three(frame width x frame height x 3):
            i.  n=0.
           ii.  Pixel (i) + pixel (i+1) + pixel (i+2) / 3.
          iii.  Store that value as a single gray pixel(n).
           iv.  n=n+1.
            v.  i=i+3.

3. If it is the first frame, store as previous frame and return.

4. Else store as current frame.
      a. Eliminate noise using a blur convolution.
            i.  For every pixel, apply blur mask.

      b. Find the absolute difference between the current frame and
         previous frame and store as a new Gray Image.
            i.  For every gray pixel, i=0:
                  1. Absolute difference frame(i) = Current frame(i) –
                     previous frame(i).
                  2. If absolute deference frame(i) < 0.
                        a. Absolute deference frame(i) * -1.

      c. Convert the image to black and white with a threshold value.
            i.  For every pixel:
                  1. If gray pixel < threshold, pixel = 0.
                  2. Else gray pixel > threshold, pixel = 255.

      d. Find the largest white, blob or contour.
            i.  Find the contour of the first blob. Store its' area and
                store it as the largest contour.
           ii.  Find the next contour and store its area.
                  1. If area of the new contour is larger, store it as
                     the largest contour.
          iii.  Repeat until all contours are analyzed.

      e. Draw a rectangle around the largest contour to show the
         largest motion detected.

      f. Store current frame as previous frame and return.
```

Figure 2-3: Pseudo code for motion tracking algorithm.

2.1.4 Optical Flow

Optical flow is the distribution of apparent velocities produced by the change in motion of pixel intensity. It can arise from relative motion with respect to object and the viewer [13]. The original concept of optical flow was introduced by a psychologist James J. Gibson in 1940s as a means to describe the visual stimuli provided to the animals in motion [34]. In the recent past, optical flow has been more involved in robotics applications and computer vision applications. It is used in image processing and control of navigation, such as motion detection, object segmentations, luminance, and motion compensated encoding [40, 56]. In robotic application, it will be an estimation rate of pixel flow through sequential frames or video. Due to this, the rate of flow can be represented as an instantaneous velocity or discrete image displacement [56]. The focus of this thesis will be isolated more towards optical flow as a tracking algorithm. The goal is to track a rectangular area from frame to frame.

Even though it is possible to track every pixel in the frame, it will require an abundance of unnecessary computation. Next best option is to track every single pixel in the targeted area. However, even this may not be necessary. Tracking a few good pixels within the targeted area will be sufficient. How does one determine a good pixel/point compared to another? Good feature points must be detected from the targeted area using one of the methods suggested in Section 2.1.2. Unlike the tracking based on detection algorithm, the features only need to be detected once. Once detected, the optical flow algorithm will track those points through consecutive frames.

2.1.5 Good Feature Detection using Harris Corner Detection

Harris corner detector is another name for the original algorithm proposed by Chris Harris and Mike Stephens in their paper, “A Combined Corner and Edge Detector” [15]. This is a good feature detector to choose for optical flow since the two algorithms have similar calculations. The Harris Corner Detection algorithm finds any change in intensity for all directional displacement as shown in Equation (2-1). Where I is intensity, w is the window function, w(x,y) is 1 if pixel exist inside the image window and 0 otherwise, x and y

are the pixel coordinates, *v* and *u* are shifted displacement in orthogonal direction, and *E(u,v)* is the change produced by the shift [15].

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,\ y+v) - I(x,y)]^2 \tag{2-1}$$

A visual representation of the patch shift is illustrated in Figure 2-4.

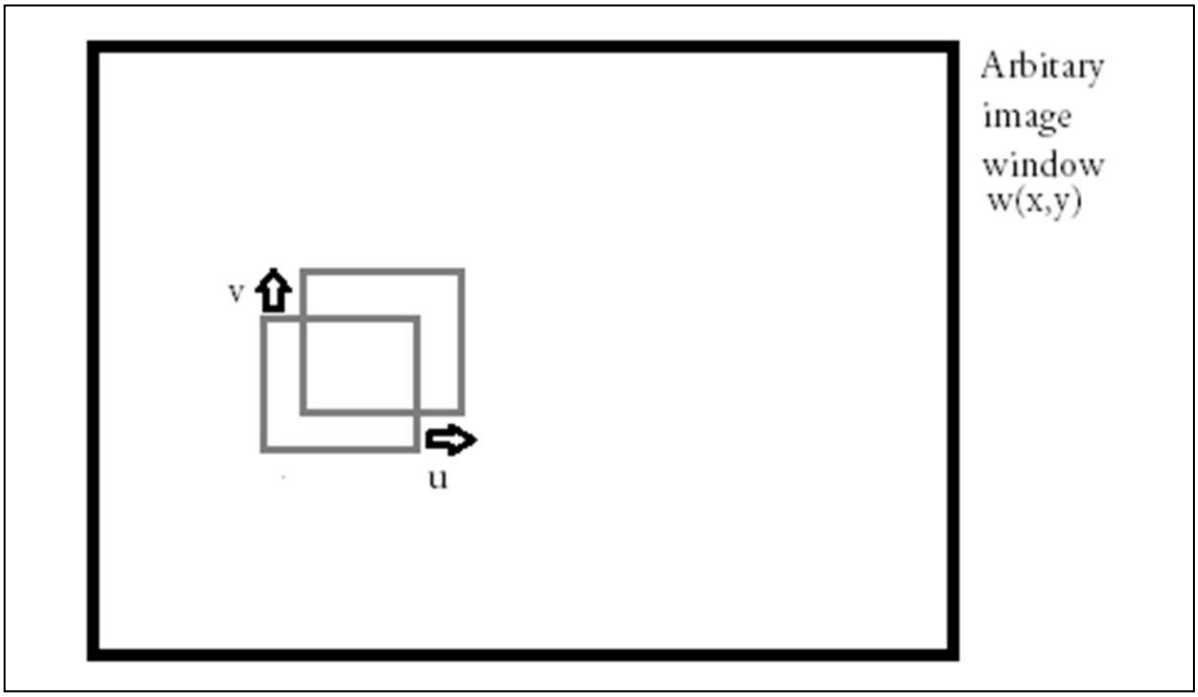


Figure 2-4: Patch shift to check if corner exists (small square checks change in intensity, E).

Expanding the equation using Taylor series expansion will be an approximation for

$$E(u,v) = [u\ \ v]M\begin{bmatrix}u\\v\end{bmatrix} \tag{2-2}$$

Where

$$M = \sum w(x,y)\begin{vmatrix}I_xI_x & I_xI_y\\ I_xI_y & I_yI_y\end{vmatrix}, \tag{2-3}$$

and I_x is the derivative of the image in the x direction and I_y is the derivative of the image in y direction. To find the directional derivate, a Sobel mask is used in a convolution process of the image frame as shown in Equation (2-4).

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} for\ Y-gradient, \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} for\ X-gradient \tag{2-4}$$

Finally, the equation for determining if a window can contain a corner was modeled based on completing the above calculations. For this equation, traditionally the eigenvalues must be found for M: λ_1 and λ_2. Equation (2-5) presents the Harris corner equation that is used to determine if a corner exists depending on the value R. k is the Harris parameter constant set by the user, which determines the quality of the corner.

$$R = \lambda_1 * \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{2-5}$$

If the absolute value of R is small, λ_1 and λ_2 are small, therefore the region is smoother. If R is smaller than zero, then one of the λ values are much larger than the other. This means that the region is an edge. When R value is very large, both λ values must also be large, which means the region is a corner. However, the determinant and the trace can be found without calculating eigenvalues by modifying the already calculated results as shown in Equation (2-6) [16]. Pseudo code of this algorithm is listed in Figure 2-5.

$$R = (I_x I_x * I_y I_y - I_x I_y * I_x I_y) - k(I_x I_x + I_y I_y) \tag{2-6}$$

1. Acquire color frame.

2. Convert the colour frame to gray frame (refer to Figure 2-3, step 2)

3. Compute the first order derivatives of x and y at every pixel (I_x and I_y) using the gradient Sobel masks.
   a. Convolution of y-gradient mask of the gray image and store as I_y.
   b. Convolution of x-gradient mask of the gray image and store as I_x.

4. Compute the products of the derivatives at every pixel. ($I_{xx} = I_xI_x$, $I_{yy} = I_yI_y$, $I_{xy} = I_xI_y$)
   a. For 0 to total number of pixels image : i.
      i. $I_{xx}[i] = I_x[i]*I_x[i]$.
      ii. $I_{yy}[i] = I_y[i]*I_y[i]$.
      iii. $I_{xy}[i] = I_x[i]*I_y[i]$.

5. Use Gaussian blur to smooth-out the three matrixes from the previous step.
   a. Use convolution with a 5 by 5 Gaussian mask.

6. Calculate the Harris response function R for each pixel.
   a. For 0 to total number of pixels image : i.
      i. $R[i] = (I_xI_x[i] * I_yI_y[i] - I_xI_y[i] * I_xI_y[i]) - k(I_xI_x[i] + I_yI_y[i])$
         1. K → Harris parameter constant

7. Find the maximum Harris value and the minimum.
   a. For every R value : i. (initial Rmax = 0, Rmin =0).
      i. If(Rmax < R[i]) Rmax = R[i].
      ii. If(Rmin > R[i]) Rmin = R[i].

8. Use quality of corner formula to pick good corners as the feature points.
   a. For every R value : i.
      i. If(R[i] > Rmin + (Rmax - Rmin)*qualityLevel/100.
         1. It is an acceptable corner.

Figure 2-5: Pseudo code Harris corner detection.

There are a lot of convolutions involved in the many of the image processing and signal processing algorithms. Convolution is essentiality multiplying all the surrounding pixels with the corresponding mask value and adding all of them together; which will represent the respective pixel. An algorithm for 2-dimensioanl convolution using a 3x3 mask matrix is provided in Figure 2-6 [63].

```
For i through every row of image.
      For j through every columns.
            For y through every mask rows.
                  For x through every mask columns.
  1.  iBoundry = i + y – half of mask columns.
      jBoundry = j + x – half of mask rows.


  2. if(iBoundry >= 0 and iBoundry < rows and jBoundry >= 0 && jBoundry
     < cols )

      convImage (i,j) += originalImage (iBoundry, jBoundry) * maskMatrix
                         (maskRows – y - 1, maskCol – x - 1).
```

Figure 2-6: Convolution algorithm.

The Gaussian mask is also used in Harris corner detection. The 3x3 Gaussian mask used is illustrated in Equation (2-6) [48].

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix} \tag{2-6}$$

2.1.6 Optical Flow Model

Once good features from the targeted area are detected, it can be tracked using optical flow. In optical flow, the common assumptions are that the pixel intensities are translated from each frame and that the movement from each frame is small [17]. Since optical flow

calculates relative motion in consecutive frames at a given position; the position of each consecutive frame can be labeled as t, and t+Δt. The intensity translation in 2-D can be represented as x+$u(x)$, and y+$v(y)$. An equation can be modeled to represent the intensity (I) translation in space and time with the stated assumptions, as shown in Equation (2-7).

$$I(x, y, t) = I\big(x + u(x),\ y + v(y), t + \Delta t\big) \tag{2-7}$$

Since Taylor series of f(x + a) = f(x) + a$\frac{d}{dx}$f(x) + higher order terms. Taylor series can be applied to Equation (2-7), and the higher order terms can be ignored: Equation (2-8).

$$I(x\ + u(x), y + v(y), t + \Delta t) = I(x, y, t) + \frac{dI}{dx}u(x) + \frac{dI}{dy}v(y) + \frac{dI}{dt}\Delta t \tag{2-8}$$

Substitution of Equation (2-7) into Equation (2-8) yields the optical flow equation as illustrated below in Equation (2-10).

$$0 = \frac{dI}{dx}u(x) + \frac{dI}{dy}v(y) + \frac{dI}{dt}\Delta t \tag{2-9}$$

$$0 = \frac{dI}{dx}\frac{u(x)}{\Delta t} + \frac{dI}{dy}\frac{v(y)}{\Delta t} + \frac{dI}{dt} \tag{2-10}$$

The derivative of position with respect to time reveals an equation of relative velocity. It is also appropriate to represent it as a derivative of time since it is an estimation of the future. This equation can then be represented as shown in Equation (2-12); where I_t is $\frac{dI}{dt}\Delta t$, $I_x u$ is $\frac{dI}{dx}u(x)$, and $I_y v$ is $\frac{dI}{dy}v(y)$.

$$0 = I_t + I_x u + I_y v \tag{2-11}$$

$$0 = I_t + \nabla I \cdot [u\ \ v]^T \tag{2-12}$$

The optical flow equation shown in Equation (2-12) cannot be solved at such a state. There are two unknowns, u and v, and only one equation. Dealing with visual motion, such a challenge is called aperture problem.

2.1.7 Solving the Aperture Problem

Motion perception is a process of understanding speed and direction of elements in a scene based on visual inputs [17, 19, 16]. Aperture problem is explained as if the scene is viewed through a small window or an aperture. However, the observed motion direction of a contour is ambiguous. Imagine a scene with parallel black lines being viewed through a small window. Now imagine the scene moving, and the lines are in motion. It is possible to recognize that there is motion, as the lines are appearing and disappearing from the view of the window. However, it is impossible to predict which direction the scene is actually moving. To solve such a problem, many different authors proposed different methods [23, 13, 17, 2]. Lucas Kanade [9] method is a popular method that is simple and efficient.

2.1.8 Lucas-Kanade Method

Bruce D. Lucas and Takeo Kanade proposed an idea for stereo image processing in 1981 and it is still being used in other image processing applications today. It is mostly used today as part of optical flow algorithms [9]. The idea is to assume that all local neighbouring pixels behave the same way and therefore all flow in a local neighbourhood is constant [17,2,9,10]. This method is less sensitive to noise and can solve ambiguity problems like the aperture problem.

The assumption made for this specific case is that for every feature point detected, all neighbouring twenty-five pixels behave the same way. Therefore the optical flow equation will go from having one equation and two unknowns to having twenty-five equation and two unknowns. This over determined problem can be solved by least squares principle. Equation (2-13), (2-14), and (2-15) illustrates how this method is mathematically implemented.

$$\begin{bmatrix} I_x(p1) & I_y(p1) \\ \vdots & \vdots \\ I_x(p25) & I_y(p25) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p1) \\ \vdots \\ I_t(p25) \end{bmatrix} \tag{2-13}$$

Transpose of A is multiplied on both sides to achieve a square matrix.

$$A^T A\,[\,u\;\;v]^T = A^T I_t \tag{2-14}$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{2-15}$$

One can now solve for the two velocities u and v as illustrated in Equation (2-16) and (2-17).

$$u = \frac{-\sum I_y I_y \sum I_x I_t + \sum I_x I_y \sum I_y I_t}{\sum I_x I_x \sum I_y I_y - [\sum I_x I_y]^2} \tag{2-16}$$

$$v = \frac{\sum I_x I_t \sum I_x I_y - \sum I_x I_x \sum I_y I_t}{\sum I_x I_x \sum I_y I_y - [\sum I_x I_y]^2} \tag{2-17}$$

These are the final optical flow equations obtained using Lucas Kanade approach. This can now be programed to track a region by estimating the chosen pixel velocities. The algorithm for this approach is demonstrated on Figure 2-7.

1. Get the current frame.

2. Convert to gray frame.

3. If it is the first frame.
   a. Extract the Region of interest (ROI) or targeted area.
   b. Find interesting features (Figure 2-5).
   c. Plot the feature and draw a target rectangle around the target to identify the target.

4. If it is not the first frame.
   a. Get a 5x5 window of pixels from the current frame around one feature from the previous frame.
   b. Compute the first order derivatives of x and y at those pixels (I_x and I_y) using the gradient masks (Figure 2-5, step 3).
   c. Compute the products of the derivatives ($I_{xx} = I_xI_x$, $I_{yy} = I_yI_y$, $I_{xy} = I_xI_y$) (Figure 2-5, step 4).
   d. Sum up all the products calculated on the previous steps.
   e. Apply the summed terms to optical flow Equations (2-16) and (2-17) and find the velocities u and v.
   f. Multiply the velocity by the difference in time between the two frames to get displacement of that feature.
   g. Store new feature location and repeat from step 4,a for the next feature.
   h. Plot the translated features and draw a rectangle around all of them to show the target has shifted.

Figure 2-7: Algorithm for optical flow-Lucas Kanade.

2.2 Navigation

Navigation is the ability to manoeuvre around a given environment, from one point in space to another. This is one of the most important abilities to possess for any autonomous vehicle. Avoiding non-ideal situations such as obstacles, bad terrains, and bad illumination, are just some of the circumstances that will measure the effectiveness of the robot's navigation ability. Many times, the task of translating its position to a destination position in 3D space is the real test of this ability.

Humans perform this complicated task simply throughout their everyday life. For humans, a goal location or destination is set in mind. Assuming that one knows where this location is with respect to their current location; one will start to make their way towards that target. If the human has never navigated the environment in between him/her and the destination, it can be assumed that they may encounter some dead ends and incorrect paths. Through this process, the human is making an internal map of the local area from their perspective. When the human finally arrives at the correct destination, he or she will possess at least a partial map of the local area. The next time the human wishes to navigate to an area within this partial map, they will plan a path internally to arrive at the destination based on navigation time, physical exertion, and other applicable factors. Regardless of the planned path or arbitrarily locating, one will avoid obstacles, such as chairs, tables, and any other objects that are hindering the path to the destination.

The upcoming portion of this chapter will attempt to recreate the navigational logic process explained above. The navigation is split into two major sections; one will focus on navigating on an area that is already mapped and the other will focus on navigating in an un-mapped area. Different algorithms will be tested for each scenario in simulation on the real robot. The process of mapping itself when navigating a non-mapped area is not explored in this thesis as it is a very large topic that should be explored individually.

For the map-based navigational algorithms explored in this thesis, the entire map of the local area was provided and assumed to be accessible at all times. Given the map, starting position, and final destination; it becomes a path planning problem. Path planning problem can be viewed from many different perspectives. From an artificial intelligence

perspective, path planning is the logical actions that transform an initial state to desired state. This can be obtained by genetic algorithms [62], machine learning algorithms [3, 67], and others of the sort. From a control theory perspective, it becomes more of stability [65], feedback, and optimality problem. From video gaming and some robotics perspective, it is focused more on motion planning [57, 24, 59, 27], where useful motions are generated based on geometric model to navigate from one point to the other. The implemented algorithms focus towards a motion model with respect to regular and irregular environments. For the purpose of simulation, the motion of the robot also needed to be modeled.

2.2.1 Modeling the Differential Steering Robot

The actual robot uses two separate tank threaded wheels, controlled by two separate motors, and motor controllers. The wheels are exactly the same size and are mounted mirroring each other. Therefore, it can be stated that the robot is using a differential steering system. A differential steering robot is a robot that uses two independently controlled wheels mounted on the same axis for motion [52, 57, 61]; also referred to by some as differential drive system. However, vehicles driven by different set of wheels and steered by another set can also be referred to as differential drive (rear wheel drive vehicles). To avoid confusion in this thesis, it will be referred to as differential steering system. All simulated robot motions are based on the differential steering kinematic model derived in the following paragraphs.

Based on Figure 2-8, it is evident that by varying the ratio of each wheels' velocity, the trajectory can be modeled. The trajectory can be taken with respect to the left wheel, right wheel, or the center of the axis [52, 57]. The point of rotation is called instantaneous centre of rotation (ICR) and the current center position of the robot is considered as the reference point in x and y space.

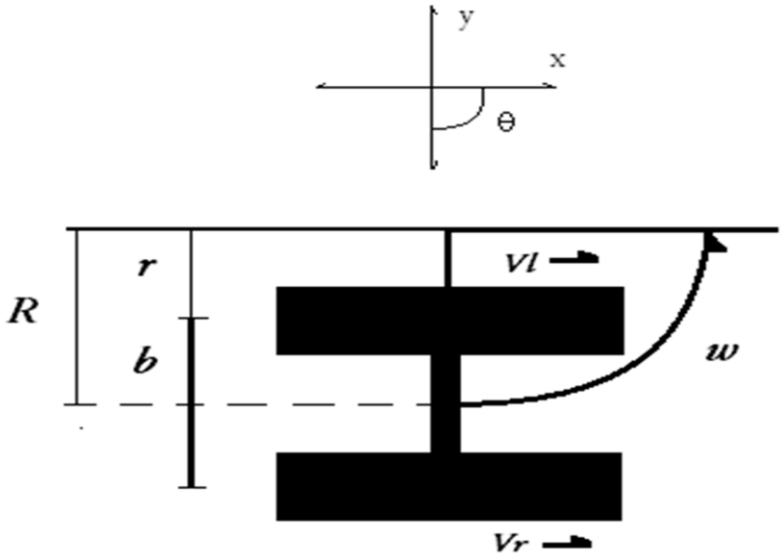


Figure 2-8: Differential steering system trajectory.

By varying the velocities of both wheels, the trajectory can be modeled, since the rate of rotation, ω about the ICR, must remain constant for both wheels, as shown in Equation (2-18) and Equation (2-19).

$$Vr = \omega(R + b/2) \tag{2-18}$$

$$Vl = \omega(R - b/2) \tag{2-19}$$

Where b is the distance between the two wheels on the same axis, R is the distance from the center of the robot to ICR, Vr and Vl are the velocities of the left and right wheels, respectively. The trajectory of the turn with respect to "Vr" can be found by substituting R with $(b + r)$ to the above equations. Similarly, by substituting R with r, the trajectory with respect to "Vl" can be found. For this model, the center of the robot is considered as the best choice since that is the reference point. With Equations (2-18) and (2-19), two new equations can be calculated to solve for R and ω at any instant; refer to Equation (2-20) and Equation (2-21).

$$R = \frac{b(Vr + Vl)}{2(Vr - Vl)} \tag{2-20}$$

$$\omega = \frac{Vr - Vl}{b} \tag{2-21}$$

Both Equations (2-20) and (2-21) translates the motion of the robot to three unique situations that a differential steering robot can be in. First, when $Vr = Vl$, R will become infinite and ωwill equal zero. In this scenario, the robot is moving either forward or backward in a linear path. Second, when $Vr = -Vl$, R will equal zero and the rate of rotation will double. In this scenario, the robot will rotate in the same spot and there will be no translation in x and y space. The third scenario is when one wheel rotates and the other does not. This scenario, ICR will become the position of the stopped wheel and R will equal half the robot's width, $\frac{b}{2}$. Achieving the three unique situations however is not always practical due to traction, friction, power delivered to the motors, and in-accuracies in the sensor data. Many times, a situation very close to the three unique situations are achieved. For the purposes of a navigational simulation, it was assumed that the unique situations are achievable.

Now that the robot behaviour is modeled based on each wheel's velocity, there is a need for a motion model based on displacement in the x, y plane. Translation in orientation can then be described by the angle Θ. Assume sampling time is dt, x_0 and y_0 are the current position, Θ_0 is the current orientation, and R is the displacement to ICR from the center of the robot. Equation (2-22) describes the new position and orientation at t + dt using the rotation matrix.

$$\begin{bmatrix} x \\ y \\ \Theta \end{bmatrix} = \begin{bmatrix} cos(\omega dt) & -sin(\omega dt) & 0 \\ sin(\omega dt) & cos(\omega dt) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Rsin(\Theta_0) \\ -Rcos(\Theta_0) \\ \Theta_0 \end{bmatrix} + \begin{bmatrix} x_0 - Rsin(\Theta_0) \\ y_0 + Rcos(\Theta_0) \\ \omega dt \end{bmatrix} \tag{2-22}$$

Assuming that the velocities of each wheel are constant, Equations (2-23), (2-24), and (2-25) illustrate the new position and orientation as a function of each wheels' velocity;

$$\Theta = \omega dt + \Theta_0$$

$$\Theta = \frac{Vr - Vl}{b} dt + \Theta_0 \tag{2-23}$$

$$x = R[\sin(\Theta) - \sin(\Theta_0)] + x_0$$

$$x = \frac{b(Vr + Vl)}{2(Vr - Vl)} [sin(\frac{Vr - Vl}{b} dt + \Theta_0) - sin(\Theta_0)] + x_0 \tag{2-24}$$

$$y = -R[cos(\Theta) - cos(\Theta_0)] + y_0$$

$$y = \frac{b(Vr + Vl)}{2(Vr - Vl)} [cos(\frac{Vr - Vl}{b} dt + \Theta_0) - cos(\Theta_0)] + y_0 \tag{2-25}$$

Since the actual robot is relatively small, not heavy, and the sampling time is very small, acceleration and inertia are ignored for the simulation. Also, the small velocities are considered instantaneous. The equations are coded into a function where the program will provide the current position and orientation as well as the speed of each wheel based on the navigational algorithm. The function will then update the current position and orientation based on the velocities of each wheel.

2.2.2 Modified A-Star (A*)

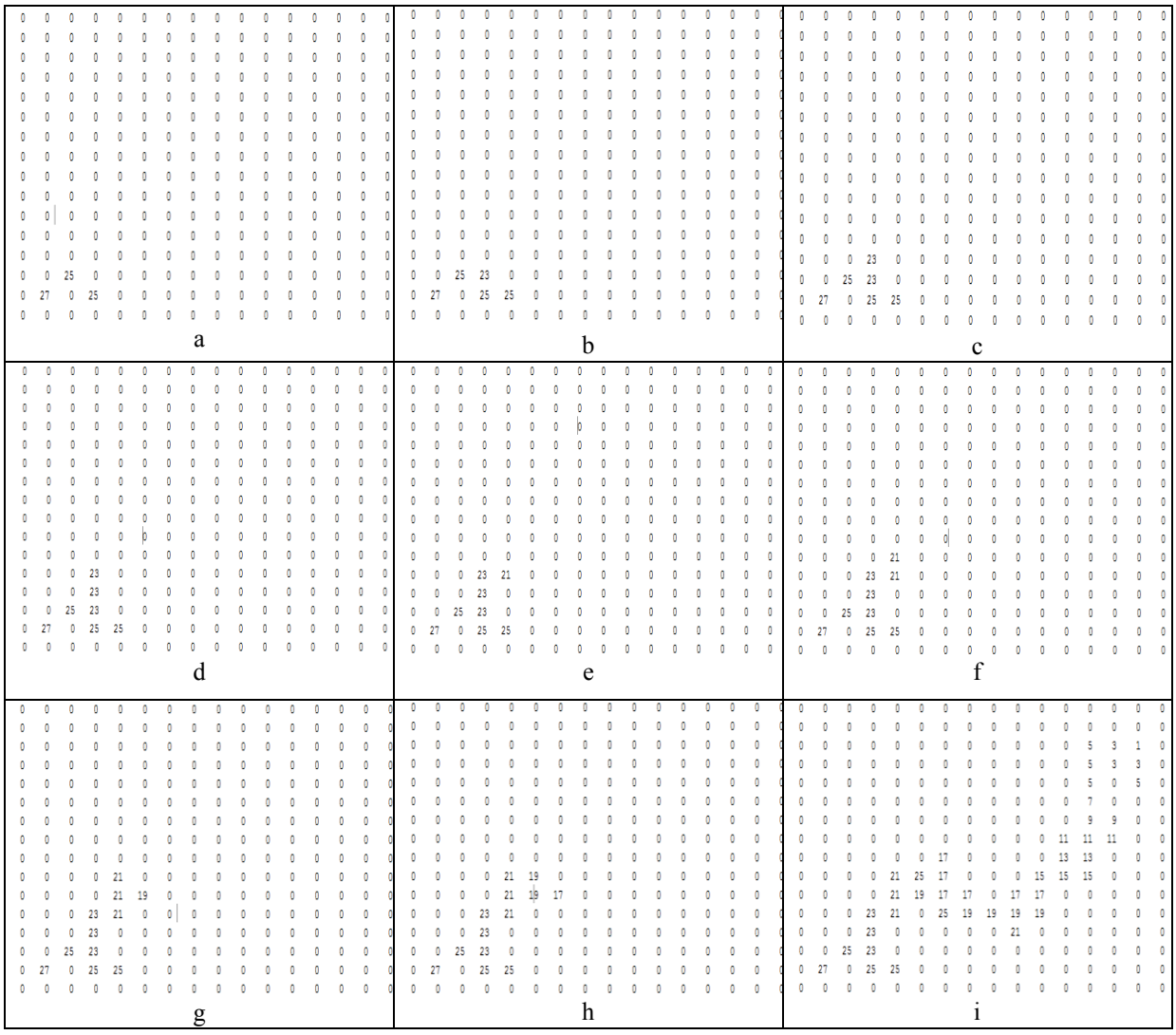
A-Star [24, 59, 27, 49] is a well-known path planning algorithm that can be applied to geometric space, but not limited to. It is used to find a path from a starting point to a destination point. This algorithm will start at the robot's starting location and expand outwards until the destination is reached. Then, it will map the path from the destination to the start.

The simulation is done in grid-block like map. This is further explained in the experimental results chapter, Chapter 3. The implemented algorithm starts at the grid-block with the robot's current position, or starting position. This block is considered as the "parent block" for the moment. A "parent block" is a block that is currently being analyzed. From the parent block, all adjacent blocks, except the diagonal blocks, need to calculate three factors, G(gx,gy), H(gx,gy), and F(gx,gy); gx and gy are coordinates of the grid-block location. If the adjacent block being analyzed has "non-path" status, that block will be ignored. G(gx,gy) is

equal to the distance traveled from the original starting block, also referred to as movement cost. H(gx,gy) is the estimated distance to the destination block. It is an estimate since there may be obstructions in the path that is not yet calculated. F(gx,gy) is the sum of both G(gx,gy) and H(gx,gy) as shown in Equation (2-26).

$$F(gx, gy) = G(gx, gy) + H(gx, gy) \tag{2-26}$$

The distances are calculated using "Manhattan method" instead of Euclidean. Manhattan distance is the sum of blocks in the x-direction and y-direction. This will save computation and it is an estimate regardless. Also, for every analyzed adjacent block, a directional vector point towards the parent block must be provided. This is used for tracing back the original start position at the end of the algorithm. Once all the calculations are done for the applicable diagonal blocks, the parent block will need to be added to the "closed-blocks" list. Closed-list has all the blocks that have been analyzed as the parent block, ensuring no repetition. Now a new parent block will need to be designated and the block with lowest F(gx,gy) will be assigned as the parent block. This process will continue until, the destination block is analysed. Figure 2-9 shows the process of propagating the "Fgrid" for the first eight iterations and the final iteration. "Fgrid" is the 2D matrix that stores all the F(gx,gy) block values.



3

Figure 2-9: First eight iterations and the final iteration of Fgrid propagation for the map "Tree-road" (Appendix C). Start's at "a" and finishes at "i".

Now, destination block will need to trace its way back from the destination to the starting block by following the direction from every block to its respective parent block. As the path is traced back to the original first parent block, each block will be given a number higher than the previously occupied block by one.

2.2.3 Goal Based Navigational Vector Field (GBNVF)

The algorithm proposed here is similar to the A-Star navigation. However, unlike A-Star, GBNVF focuses on the destination with respect to the entire map, regardless of where the robot may be. Such an algorithm can also be used for multiple robots, or any dynamic obstacles that are trying to navigate to the same goal [55, 54, 31]. Similar algorithms are implemented in the real-time strategy (RTS) video gaming physics; where a "crowd" of units need to get to a single destination [31].

The idea is to have a vector field, for each block of the grid-map, pointing in the direction towards the closest open space that will eventually lead to the target. It is important to note that GBNVF is unlike generic potential fields created based on target and obstacles. Generic potential fields have an attractive force towards the target and repulsive forces from the obstacles [32, 38]. A field such as this will only show the vector difference of the attraction force and repulsive force. This type of field can cause the robot to experience local minimums as illustrated in Figure 2-10.

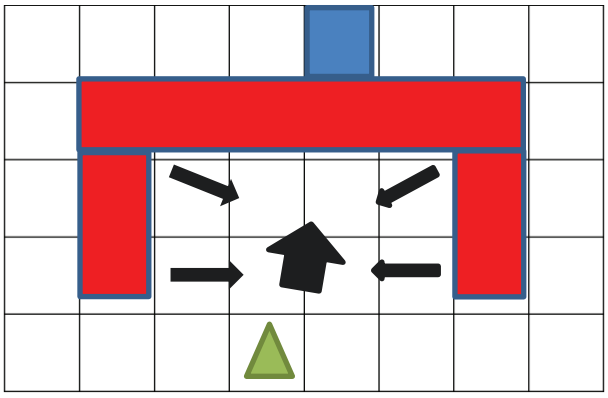


Figure 2-10: Arbitrary local minimum encountered with potential field. The robot is the green triangle, red rectangles are the obstacles, and blue square is the target.

GBNVF is an algorithm that is essentially building a path from every grid-block to the final destination. The path distance is calculated using Manhattan method (blocks), but unlike A-Star, the distance is reflected based on obstacles. Figure 2-11 shows the three different types of distance from an arbitrary point represented by the robot to the destination. For GBNVF algorithm, Manhattan obstacle-based distance (MOBD) was used.

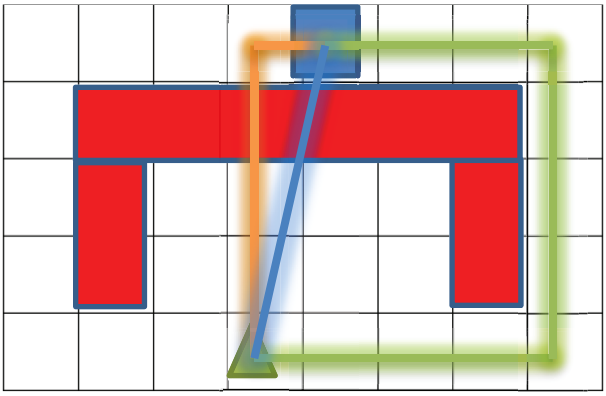


Figure 2-11: The three different types of distance calculations to a destination. Blue line represents the Euclidean distance (4.12 blocks), orange line displays the Manhattan distance (5 blocks), and the green line represents the MOBD (11 blocks).

MOBD will be calculated for every block starting at the destination with a value of zero and propagating outwards until all navigational blocks are reached. This process is referred to as a "wavefront algorithm". Once a distance is established for all the applicable blocks, a vector field must be generated for each block. X component of the vector is calculated by taking the difference from the block that is to the left and right. Y component of the vector is calculated by taking the difference from the block above and below. If a block is a non-navigational block, then that block is given the sum of the current block value and an obstacle constant. The source code of this algorithm is provided in Appendix C.

2.3 Artificial Intelligence based on Neural Networks

What is intelligence? One could argue that intelligence is the ability to make the best possible choice based on past experiences; it is the ability to learn. Part of human intelligence is due to the rate in which humans can learn and recognize pattern and reason based on past experiences. This ability allows humans and many animals to avoid making similar mistakes in repetition. However, it is still a slow process for most. Even "common sense" of a ten years old child is based on ten years of constant learning through experience and literature. Regardless, without the ability to learn and make predictions, there will be no intelligence.

What enabled humans to learn has been a fascinating subject for many centuries until recently. The milestone discovery of the neuron was not accepted until the late nineteenth century; it could be argued as the birth of modern neuroscience. How the human brain works is a complex topic that is still a mystery in many ways. For the purpose of this thesis, only a simple summary of the biological neural network concept will be explored.

Neurons are made of three major parts; refer to Figure 2-12 for an illustration of a biological neuron. The image in Figure 2-12 is taken from Wikipedia.

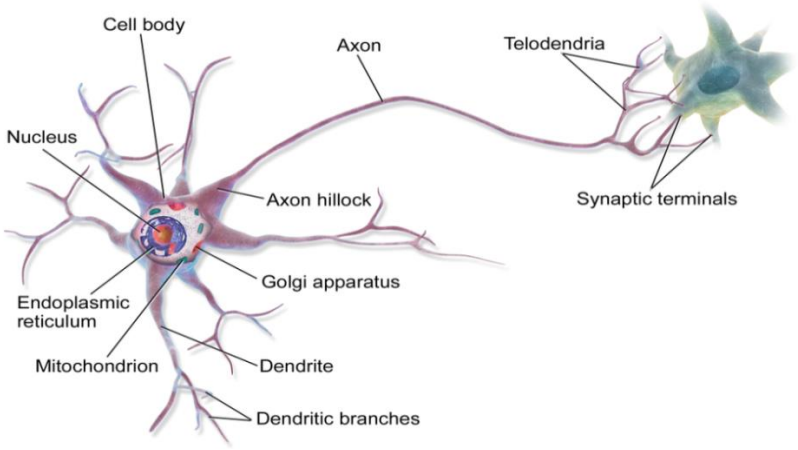


Figure 2-12: Labeled image of a neuron.

The dendrites are branches of fibre that connects to other neurons and relays messages to the cell body. Cell body controls and directs all activities within the neuron. The axon connects to dendrites of other neurons and transmits signals from the cell body. To communicate a message, the neuron releases a neurotransmitter into the synapses. Billions of neurons are connected together in a very complex network topology to create consciousness. These networks formed inside the brain are responsible for learning, idea, emotions, personality, and so much more.

In an attempt to replicate partial human reasoning and learning behaviour, artificial neural networks (ANN) were created. It is first introduced by W. McCulloch and W. Pitts in 1943 as a mathematical algorithm. It is essentially a statistical learning method based on biological neural network to estimate and approximate [42].

ANN was introduced as a solution to solve challenges encountered with the navigation algorithms on the mobile robot. The ANN was responsible for learning the motion

behaviours of the robot with respect to the power delivered to each wheel. The ANN used is a very simple form of a large division that is AI.

2.3.1 Artificial Neural Network

Typical feed-forward (FF) artificial neural network (FFANN) are structured with neurons or nodes connected to at least another node in a unique or general topology of network; where each connection has a certain weight associated with it. Such networks are used for signal prediction, where the signal is time variant and the desired output is a prediction at a certain time [53]. This topology can also be used for signal classification such as speech recognition, signal production, and optimization [53, 5].

Each node/neuron has an internal state, which is known as activity level. The activation is transmitted by one neuron to another. Usually, each neuron only has one output as it only has one activation function, which can then be broadcasted to many more if necessary. This creates a product sum network for each neuron that is not in the input layer, as shown in Equations (2-27) and (2-28) [42]. "i" is a value associated with the layer, starting at 0 at the input layer. The neuron in a given one layer is arranged from 0 to "k" where "k" is a real number, k∈R. "w" is the weight associated to each respective connection.

$$neuronIN_{(i+1)n} = bias_k + \sum_k neuronOUT_{ik} * w_{ik} \tag{2-27}$$

Where,

$$nueronOUT \quad = activationFunction\ (nueronIN) \tag{2-28}$$

Equation (2-27) is illustrated in Figure 2-13.

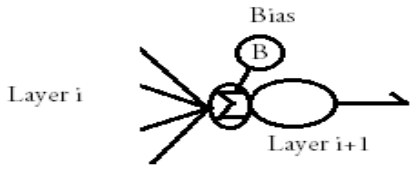


Figure 2-13: Neuron model

There are many different activation functions [50, 42]. For the purpose of this thesis, only one activation function was used called the sigmoid or bipolar sigmoid function as shown in Equation (2-29). This choice is explained later on this chapter.

$$a_k = \frac{(1- e^{-k})}{(1+ e^{-k})} \tag{2-29}$$

The network considered for this thesis is static, having no internal time delays and generating output immediately for an input signal. The network of neurons is usually arranged in layers: input layer, hidden layer, and output layer [50] as illustrated in Figure 2-14. The hidden layer can have multiple layers if necessary, creating a deep-hidden layer.

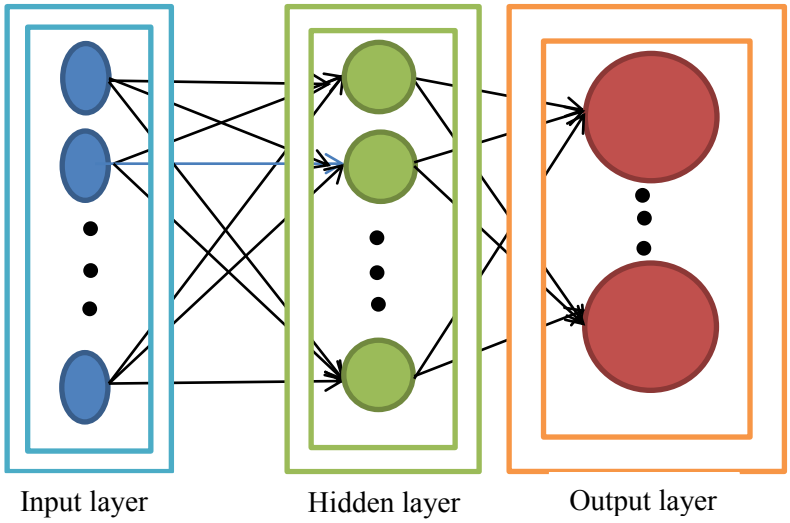


Figure 2-14: FFANN layer setup.

Once a network is established, it must be trained with "experience" or datasets. When in training, the weights of each connection to the neurons will be adjusted to behave in a desired way. There are two approaches to training, supervised training and unsupervised training. For this thesis, only supervised training is explored.

In supervised training, a dataset is provided with the desired output. The system is trained in repetition to that dataset by comparing the actual output with the desired output. The weights are adjusted through back-propagation algorithm.

2.3.2 Back-Propagation (BP)

The back-propagation (BP) algorithm is used to adjust the weights of each connection [42, 50, 53]. The idea is a forward propagating network, which propagates the error at output backwards throughout the network. Starting at the input layer, signal from the input node(s) is/are multiplied by the respective weights associated with the connection and then summed together with a bias. This is then supplied as the input to the connecting node in the hidden layer as shown in Equation (2-27). The output signal is the activation signal for a given neuron, which is multiplied by the connection weight and summed up with all other connection activation functions and bias. This will be the input for the next layer, another hidden layer or the output layer. Based on the principle w = w + Δw, the training process can be identified as a process that will minimize the error by altering the weights and bias as shown in Equation (2-30).

$$\Delta w \propto -\frac{dE}{dw} \tag{2-30}$$

Where E is the error and w is the associated weight. The error can be viewed as a gradient decent on sum squared, where the total in the network can be modeled by the Equation (2-31).

$$E = 1/2 \sum_{k} (d_k - a_k)^2 \tag{2-31}$$

Where d_k is the associated desired value at the output of the neuron and a_k is the actual output by the activation function at that neuron. The activation function considered for the ANN used in this thesis is the bipolar sigmoid function, refer to Equation (2-29). This function creates horizontal hyperbolas at y = 1 and y = -1. The function at infinity will only get close to the 1 and never equal 1, and at negativity infinity will only get close to -1 and never equal -1. This function is plotted between -10 and 10 on Figure 2-15, image acquired from wolframAlpha.

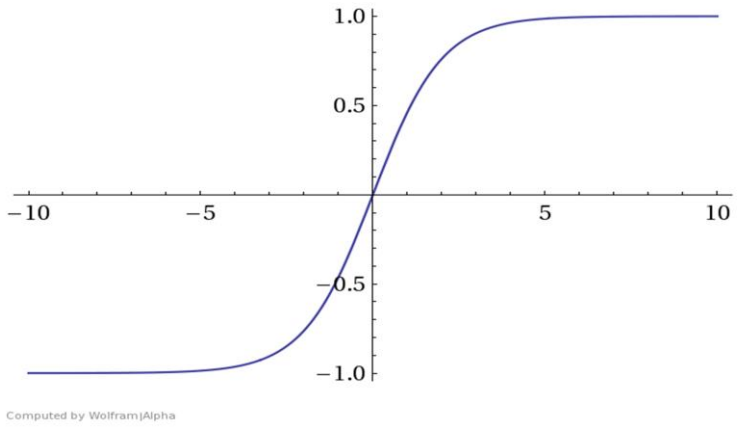


Figure 2-15: Bipolar-sigmoid function.

The back propagation starts initially at the output layer, where the change in weight can be represented by Equation (2-32); the notation "o" represents the output layer and "io" represents the input at the output layer.

$$\Delta w_o \propto -\frac{dE}{dw_o}$$

$$\Delta w_o \propto -\frac{dE}{da_o}\frac{da_o}{d(io)}\frac{d(io)}{dw_o} \tag{2-32}$$

The chain rule expansion can be further broken down into the individual derivatives as illustrated in Equations (2-33), (2-34), and (2-35).

$$\frac{dE}{da_o} = \frac{d(\frac{1}{2}(d - a_o)^2)}{da_o} = -(d - a_o) \tag{2-33}$$

$$\frac{da_o}{d(io)} = \frac{d((1 - e^{-(io)}) * (1 + e^{-(io)})^{-1})}{d(io)} = \frac{e^{-(io)}}{2(1 + e^{-(io)})}\frac{-e^{-(io)}}{(1 + e^{-(io)})}$$

$$\frac{da_o}{d(io)} = \frac{1}{2}(1 + a_o)(1 - a_o) \tag{2-34}$$

$$\frac{d(io)}{dw_o} = \frac{d(w_o * a_h)}{dw_o} = a_h \tag{2-35}$$

Therefore, the change in weight at the output layer can be written as Equation (2-36) where the equation can be simplified with the constant δ_o to simplify calculations at the hidden layer.

$$\Delta w_o = \frac{1}{2}(d - a_o)(1 + a_o)(1 - a_o)a_h$$

$$\Delta w_o = \delta_o * a_h \tag{2-36}$$

At the hidden layer, the change in weight can be represented by Equation (2-37). The hidden layer is identified by the notation "h".

$$\Delta w_h \propto -\frac{dE}{dw_h}$$

$$\Delta w_h \propto -[\sum \frac{dE}{da_o}\frac{da_o}{d(io)}\frac{d(io)}{da_h}]\frac{da_h}{da_i}\frac{d(i_h)}{dw_h} \tag{2-37}$$

The derivative of error with respect to the activation at the output layer and the derivative of the activation at the output layer with respect to the input at the output layer are already calculated from Equations (2-33) and (2-34). With the Equation (2-36), derivative of input at the output layer, and activation at the hidden layer; the Equation (2-37) can be simplified and written as Equation (2-39).

$$\frac{d(io)}{da_h} = \frac{d(w_o * a_h)}{da_h} = w_o \tag{2-38}$$

$$\Delta w_h = [\sum(\delta_o * w_o)]\frac{da_h}{da_i}\frac{d(i_h)}{dw_h} \tag{2-39}$$

The derivative of the input at the hidden layer with respect to the weight at the hidden layer will yield the original input itself as demonstrated in Equation (2-40).

$$\frac{di_h}{dw_h} = \frac{d(w_h * a_i)}{dw_h} = a_i = input \tag{2-40}$$

Similar to Equation (2-36), the activation at the hidden layer with respect to the input can be equated as shown in Equation (2-41).

$$\frac{da_h}{da_i} = \frac{1}{2}(1 + a_h)(1 - a_h) \tag{2-41}$$

The entire equation for change in weight at the hidden layer can be written as shown in Equation (2-42).

$$\Delta w_h = [\sum(\delta_o * w_o)]\frac{1}{2}(1 + a_h)(1 - a_h)a_i \tag{2-42}$$

2.3.2 Modeled Network for Supervised Learning

The problem was that the robot did not behave the way the modeled simulation intended it to. As a result, the implemented solution is supposed to learn the navigational behaviour of the robot based on data sets measured from the actual robot. This process is considered supervised learning, where a data set is trained using an ANN for a certain number of iterations. Once trained, it should be able to predict the desired output for a given input. The idea is that the ANN will be provided with the positional displacement and orientation change intended by the simulated algorithm. The FF network will then predict the correct output power to the wheels based on what the FFANN has learned from the trained data set and past experience.

It is evident that there needs to be three input nodes at the input layer; change in angle, change in orthogonal displacement, and change in normal displacement. There should be two output nodes at the output layer; each representing the PWM duty ratio that should be sent to the microcontroller. With that stated, many different FFANN topologies were tested and the one that demonstrated the best result is presented in Figure 2-16.

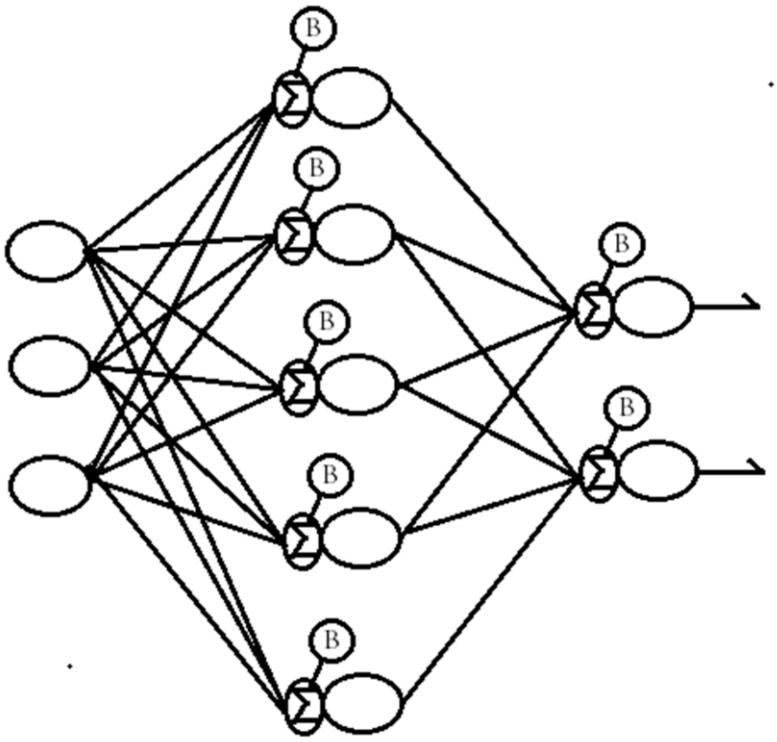


Figure 2-16: Generated feed-forward artificial neural network for the intelligence engine.

2.4 Literature Review

Novel ideas from published authors on the two major topics explored above will be explored in the following paragraphs.

N. Nourani-Vatani et al. [46] investigated feature extraction algorithms for optical flow tracking. It was suggested that sparse optical flow algorithms, such as Lucas-Kanade[9] approach, provide more robustness to noise than dense optical flow algorithms. Spares optical flow will estimate the displacement for a selected few pixels instead of an entire region. To select these pixels, "good features" (Harris corners[15], Shi-Tomasi, SIFT[18], and SURF[26]) were used. The results were analysed and evaluated based on processing time, the number of features tracked, and performance in indoor and outdoor environments. The conclusion suggested that in ideal conditions, there is very little performance difference. However, given abnormal environments, there was a considerable change in performance.

The paper showed that simpler feature extraction methods, such as canny edge or Harris corner feature extraction, provided the best results. Also, feature extraction methods only showed slight improvements in error elimination compared to random feature seeding. Random feature seeding is selecting random pixels as one of the tracked pixel.

In this thesis, only indoor environment was tested. Harris corner was used as the feature extraction method alongside a spares optical flow process. However, the accuracy of each pixel tracked was not measured. Instead a median-mean filtering solution was implemented in Section 3.1.5 to reduce the noise detected.

E. Patel and D. Shukla [23] compared different optical flow methods to determine the speed of a moving object in sequence of a video frame. Optical flow Lucas-Kanade[9] and Horn-Schunck was used to track an object by plotting a rectangle around the tracked region. The speed was then calculated by the displacement of the object with respect to frame rate. Lucas-Kanade algorithm performed very well when tested with noise, but failed to produce as many density flow vectors as Horn-Schunck. Lucas-Kanade had an average angular error of 4.3 degrees in comparison to that of Horn-Schunck which was 9.8 degrees. The author also tested a combined algorithm alongside Lucas-Kanade and Horn-Schunck algorithm. The combined algorithm combined the local and global flow, yielding a slightly smaller average angular error of 4.2. The author proposed pyramids as a future implementation to reduce the error.

In this thesis, the sequence of frames will be noisy due to the motion of the mobile robot. With that said, the Lucas-Kanade was used for the purpose of this thesis. It was more resistant to noise challenges and required less computation than the combined algorithm. The combined algorithm would be more ideal for a dense flow tracking, however, a sparse approach was used in this thesis.

A study was done to track large motion in objects with irregular shapes using contours by J. W. Choi et al. [35]. The idea was to track contour by having an active contour model. The featured points (“snake points”) on the next frame will be defined by the change in curvature of the object in the current frame. The optical flow is then calculated at that location. To filter noise and any inaccuracies, two sequential edge frames were compared and

morphology was created; which then works with optical flow to find the activation contour. The experiments showed that it was accurate in tracking fast moving, irregular objects. However, further research needs to be done to limit motion blur at high speeds to increase accuracy.

This approach is similar to the tracking based on motion used in this thesis. However, examined literature above [35] compares the two consecutive edge frames to eliminate noise from non-motion objects in the frame. It then uses dilation to increase the area of the leftover contour. Optical flow was then used to track the points placed over the resulting contour image. This thesis did not focus on tracking complex objects based on contours, instead, the target was identified as a rectangular shape and majority of the motion of the "good feature points" was translated on to all the feature points; refer to Section 3.1.5. This approach is also capable of tracking irregular shaped objects as well as normal geometric objects. Many of the irregular shaped objects produced better results because they had drastic change in pixel intensity.

C. Cheng and H. Li [16] also pursued in a search to find the most accurate feature-based optical flow computation, analysing scatter brightness, edge acquisition, and feature orientation. The results were based on six test images, all of which are drastically different. The results published showed that image patches with significant brightness variation were the most accurate. It was proposed that scatter brightness and/or edge acquisition can be used to improve accuracy on feature extraction.

For this thesis, the target may be any arbitrary object or region in space. However, if the target is chosen by the user for effective tracking, a target with diversity of intensity should be chosen. Scatter brightness was not used as part of the algorithm in this thesis. Instead of edge acquisition, corners were used as the feature points in this thesis for even better tracking results.

Optical flow was used in a video surveillance application system for tracking moving objects from a sequence of video frame [19]. This paper experimented on "abrupt change video" as well as "gradual change video" to explore the optical flow method, Horn-Schunck, with region filtering. It seemed from the results obtained by this paper that dynamic

threshold with exponential decaying as part of the filter worked best; yielding the most amounts of detections (60) as well as precision (0.91). It was no surprise that the abrupt change video had half as much (35) detections as the gradual change video (60) as optical flow works best in small change scenarios.

Research done in Lingaya's university [45] showed a different approach to tracking. They used "Sum of Absolute Difference" (SAD) algorithm for motion detection and tracking. They were using a fixed camera to detect moving objects, and its location with the help of background subtraction, edge detection, and segmentation. It was evident that this algorithm needed to have a fixed background as well as a camera that did not move. From there, the sequential frame was subtracted to find the new object in the new frame. This object was then tracked throughout the frames until it was no longer in the current frame. Working with a static camera, displacement was easily calculated.

A similar approach was taken in "tracking based on change" portion of this thesis. Unfortunately, the application is only applicable when the mobile robot is stationary. When the mobile robot is in motion, the entire frame is detected as motion.

Fragments-based similarity measurement as a tracking tool was analyzed by Jun Shang [37]. The target and the reference were divided into several fragments of the same size. Then, the average intensity of the patches was compared with respect to the overlapped smaller patches. Color and spatial information were encoded to track non-ridge object under complex background by comparing global similarity and local similarity. This algorithm was found to be sensitive to the size of the patches, the number of the searched rectangles, and the initial actual position of the object. The best results found in this paper came from a soccer sequence of frames with 0.11 seconds as the average tracking time per frame at a scale of 40x40. The worst results also came from the soccer sequence of frames with 1.33 seconds as the average tracking time per frame at a scale of 8x8.

Particle swarm optimization is also a great candidate for visual tracking. X. Cheng et al. [69] proposed an approach called visual tracking based on particle swarm optimization framework using SIFT[18] features. Also using, multiple fragments in a candidate target region to cope with the problems of particle occlusions, illumination changes, and large

motion changes of the tracked region. The experiments results showed that this algorithm was capable of performing in complex environments with occlusions. How the results were measured was confusing to understand. However, it was evident that the higher the number of swarm particles, the lower the error.

Fragmentation and swarm optimization was not used in this thesis. However, it is a good topic to explore as future work to be done on this project.

S. Avidan used an interesting approach to tracking. Ensemble tracking [58] views tracking as a binary problem where a collection of "weak" classifiers are trained online to distinguish between background and object. In return, a product of one "strong" classifier is created. Furthermore, week classifiers can be added or removed at any time to compensate for any new information or change in the tracked object. A classifier in this context can be viewed as a threshold definition that fits a corresponding object. It estimates the next possible location based on a confidence map and average shift. The results published by the paper shows a lot of promise in this area of research as it is reliable in variety of scenarios. However, this process does require access to faster computational tools and an online network. It is also one of the more complicated algorithms to solve a simple tracking problem. It is more suited as a detecting and learning algorithm than a tracking algorithm applicable for this thesis. For this reason, this approach was ignored for the purpose of this thesis at this time. However, if the mobile robot is controlled by a powerful external processing unit, it may be an applicable topic to pursue, falling under possible future work category.

A paper was published by H. Joshi, demonstrating path planning for autonomous robot using image processing [27]. The paper focused on navigation in space exploration type environmental scenario using A-star algorithm. Obstacles were detected based on a form of contour detection algorithm. This paper only focused on planning the path; no simulations were done for actual navigation. A gridded approach was taken to plan the path, yielding better computation.

A slightly modified A-Star algorithm was explored in a science direct journal by F. Duchon et al. [24]. Similar to [27], a gridded method was used for the development of the

algorithm. Modified algorithms in this literature includes, Theta-Star, Phi-Star, A-Star-Rectangular Symmetry Reduction (RSR), and A-Star-Jump Point Search (JPS). The algorithm was judged based on computational time and path optimality. It is important to note that this literature only explored the path planning aspect and not navigation and SLAM. Basic Theta-Star is a modification to A-star algorithm. It ignores the next position to be navigated to, if a more future position is visible from the current location. Phi-Star algorithm is an extension to Theta-Star. This algorithm records the local predecessor as well as the angles to each block. RSR algorithm is a pre-processing step which eliminates symmetries in the generic A-Star algorithm. It was concluded in the journal [24] that A-Star-JPS was the best algorithm for finding the shortest path. JPS is a method of cropping the neighbourhood blocks that are already evaluated and then choosing a block that is accessible to all cropped blocks. However, with higher computational time provided, it was stated that Theta-star algorithm may be more superior [24].

This thesis also explores the possibility of using a modified version of A-star algorithm as the path planning algorithm for map-based navigation scenario. A gridded approach was also used in this thesis similar to the previous literature [24, 27]. JPS and RSR are both good modifications. However, this thesis used a slightly different modification that evaluated the surrounding, applicable navigational blocks. The modification applied chooses the diagonal option that can combine two separate movements into one diagonal motion; refer to Section 3.2.2.

Navigational algorithms are not always intended for mobile robots; it is also used in graphical simulation such as video games. One such approach explained it in terms of directing a crowd using navigational fields, by P. Gweosdek et al. [54]. The idea of this approach is to guide multiple objects in a path using a guidance field; it is a local collision-avoidance method for navigation with a goal. The algorithm generates the navigational field and uses a variant of Dijkstra's to propagate navigation cost values; similar to the cost factor from the A-Star navigation implemented in this thesis. Many simulation results were shown, in which the method was deemed effective. One of the best contributions of this method is that it may be used as a global model where many objects can be guided with one method.

J. Hagelback proposed another similar method to [54], where a potential field is used for navigation, from the very popular game StarCraft [31]. It is a strategy game based on building massive army and dominating your opponent. Dealing with many objects trying to navigate to different positions simultaneously; it is evident that global type navigation was needed. It stated that A-Star algorithm did not cope well in dynamic environments provided in the game. Therefore, a combined A-star and potential field algorithm was implemented. The algorithm can be viewed as a 3D fabric of space, where obstacles are hills and empty paths are valleys and gravity is the guiding force that is exerted on the navigating object. Therefore, the object is guided through the empty space towards the goal.

J. Vascak proposed an idea that combined potential field, neural network, and fuzzy logic [32]. The experiment was approached as a parking problem; where a robot must park itself in an empty parking slot. The approaches are based on numerical values that are derived from metaphors existing in the real world. The paper presented a lot of areas to be improved as part of future work.

In this thesis, a global potential field algorithm that is similar to the previously examined literature [31,32,54] was tested. GBNVF resembles derivation of the algorithm proposed in [54]. A part of J. Vascak's resembles a version of the approach taken for the non-mapped navigational option. However, instead of using a fuzzy-neural system to control the turning of the robot, another algorithm was developed and tested based on obstacles at different angles with respect to the robot. ANN was used as the intelligence engine that is capable of learning the navigation motion behaviours of the actual robot and sending the correct power to each wheel, ensuring the turning angle is the desired angle intended by the navigational algorithm. Unlike [31], no attempts to combine modified A-Star and GBNVF was made in this thesis as the implemented algorithm was sufficient for the task at hand.

An article published in Neurocomputing by A. Prieto et al. suggested a method of classifying the motion behaviour from groups of robots [5]. They attempted to create a cognitive model that understands local events surrounding the robot using visual information as the input to an artificial neural network . The objective was to classify the type of motion a group of agents perform. The hybrid method implemented was called automatic neural-based

pattern classifier (ANPAC). The idea was to model a system that varied the size of the ANN based on the learning results of that network and an advisor module that adjusted the pre-processing parameters. The motion is analyzed through camera on the robot, which is then analyzed to classify motion patterns. ANPAC is an un-supervised learning algorithm which uses many pre-processing steps before the ANN step to constantly update the parameters at different layers proposed.

Un-supervised learning proposed in the literature above is effective and novel. ANN learning is only a small portion of the entire algorithm. The proposed algorithm was effective in learning any external motion patterns. In this thesis, the focus was on learning the motion behaviour of the mobile robot with respect to power delivered to each motor driver. Such a challenge can be solved with un-supervised learning. However, it is unnecessarily complex. In this thesis, the learning was done through supervised learning to achieve satisfactory results. Since the behaviour of only one robot needs to be learned, the algorithm can adjust to the motion data-set obtained from the mobile robot. Therefore, only the ANN step was used in FF and BP was used as a means to update weights. However, an un-supervised learning algorithm can be used in conjunction to the tracking algorithm to predict the motion of the target for future work.

S. P. Day and M. R. Davenport proposed a continuous-time approach to a neural network (NN) that learned through back-propagation and used adaptable time delays to improve performance [53]. The authors proposed that the implemented technique can be used in signal prediction, signal production, and spatio-temporal pattern recognition and is also possible to parallelized through hardware and multi-dimensional training signal. The idea was to predict the future input of a Mackey-Glass signal by providing the current input of the signal. Mackey-Glass signal is a non-linear time delay signal, making it difficult to predict. Results showed a successful algorithm amongst usefulness of other things such as momentum and network configurations. Momentum is a method applied usually to speed up the learning rate. It acts like a controller that will either speed up or slow down the change in weight in a given direction.

In this thesis, BP is a major part of the learning algorithm implemented. No variable time delay between connections was programed however. The network topology proposed in the literature above is large. Instead in this thesis, a smaller network topology was used with some hidden nodes only connected one of the output node. The results observed from the network can be observed in Section 4.3.

Chapter 3
Experimental Setup

This chapter will illustrate the final design of the robot and the setup used to obtain experimental data.

3.1 Final Robot Design

The robot construction can be viewed as a rectangular prism of dimensions 0.43m by 0.15m by 0.29m. The wheels are extended a little past the robot body, so that the robot will be able to climb over any small terrain with ease. The overall design is inspired from an armoured, tank. The body is constructed with aluminum. The optical sensor was attached to the back of the robot, on an elevated plane. The optical sensor is erected 0.61m above the floor to help with the fact that Kinect cannot measure depth under 0.8m. By the concept of Pythagorean Theorem, the visual floor in front of the robot is 0.75m. This enables the robot to identify obstacles closer to the robot. Also, having the Kinect at the back will ensure that, when in forward motion, the first point of contact will be the tank wheels. This makes it easier to manure over terrain and small obstacles.

The optical sensor base was implemented with two degrees of rotary freedom. Figure 3-1, (b) displays the vertical rotary joint and Figure 3-1, (c) displays the horizontal rotary joint. The two motors used were Maxon 343100. These motors are small in size and are able to produce power of 22 watts. Figure 3-1, (a) displays the final version of the robot.

(a) (b) (c)

Figure 3-1: Robot, version two. (a) View of the entire model. (b) Vertical rotary joint with a potentiometer. (c) Horizontal rotary joint with a potentiometer.

Many different GUIs were used throughout the building process. New layout and options were added or deleted with every edition. The final version of the GUI built and used is demonstrated in Figure 3-2.

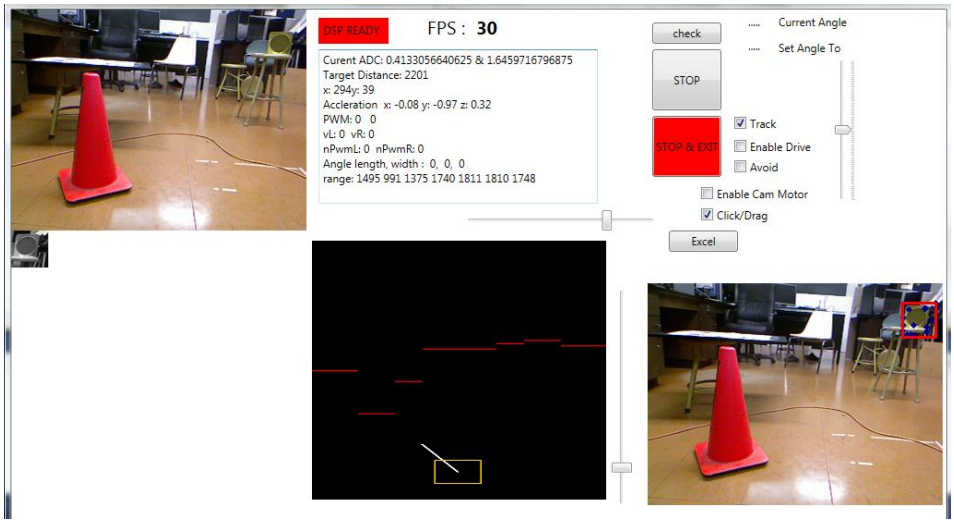


Figure 3-2: GUI for the robot.

3.2 Parts Used for the Final Version of the Robot

All the parts used in the final version of the robot are listed in Table 3-1. The block diagram of the robot is shown in Figure 3-3.

Table 3-1 Parts used in the robot

Part Name	Description	Voltage (V)
Maxon Motor T-05	This motor was used to control horizontal rotation of the Kinect.	12
2-Maxon 343100 Motors	More efficient motors from the previous motors. They are used to control both wheels.	12
Maxon 343100 Motor	This motor is used to control the vertical rotation of the Kinect.	12
2-Potentiometer	To estimate the directional angle that the Kinect is facing (horizontal and vertical).	3.3
TI-Digital signal processor (DSP)	The microcontroller that controls the motor driver and the communication from the on board computer.	3.3
Kinect sensor with ARM DSP	Main sensor used for colour, depth, and sound information,	12

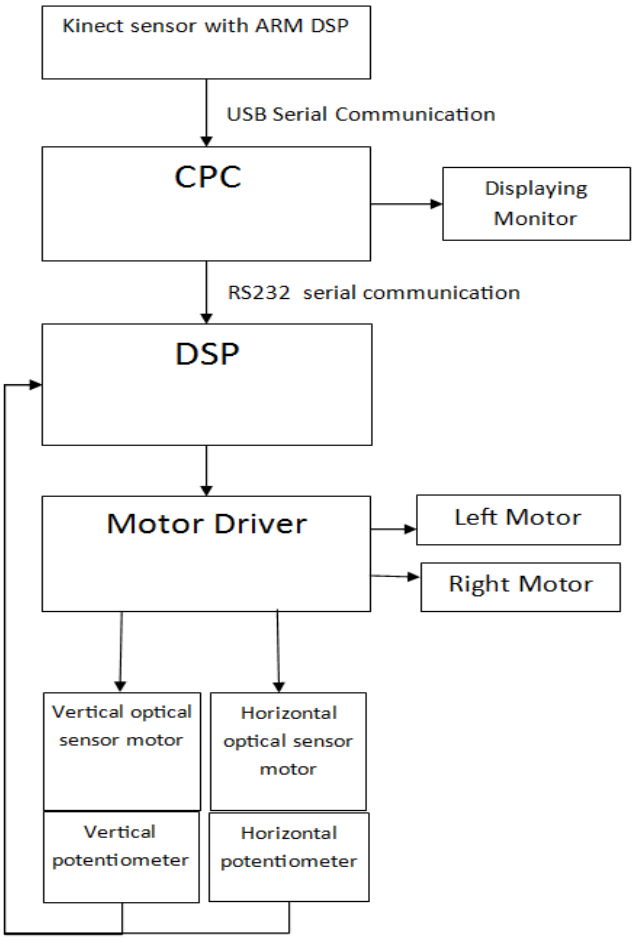


Figure 3-3: Block diagram of the robot.

3.3 Central Processing System

Lenovo Y570 laptop was used as the main computing and decision making system. The system specifications are listed below. Note that this is the system that all simulations and tests are performed on unless otherwise stated.

- Central processing unit (CPU) → Intel Core i7-2670QM CPU @ 2.20GHz Boost @ 2.99GHz.
- Random access memory (RAM) → 8.00 GB.
- Graphics processing unit (GPU) → NVIDIA GeForce GT 555M.
- 64-bit operating system, windows 7.

3.4 Simulation Setup

For all optical tracking algorithms explored, the experimental setup used one Microsoft Kinect sensor and the CPC.

Environments were created for simulating the navigation algorithm. The environments modeled are often referred to as maps in this thesis. The maps are created as a picture (.PNG) file with arbitrary space and obstacles. Some are created with large open areas for easy navigation and some are created with very narrow areas. All the maps experimented on are provided in Appendix C. Note that all graphical simulations are done using MATLAB R2013.

Before the implementation of the algorithms, the maps must be set up in a way that is understandable by code. This was achieved by converting the maps to gray scale, where gray (0-254) will be areas that are un-navigable and whites (255) navigable, open space.

The robot's size is programed to have equal x, y dimensions based on the robot's largest side. This will ensure that if the robot must rotate on its center, the extra room programed will give the necessary space to avoid collision. It is almost like creating a bubble around the robot that is slightly bigger than the robot and using that bubble as the safety region. A line is drawn from the center of the robot outwards to demonstrate the current orientation of the robot. Figure 3-4 shows the robot in its starting position and orientation, in the map "Tree-road", where the destination is highlighted with a "red star". From here, the robot must find its way to the destination; this is where path planning algorithms are utilized.

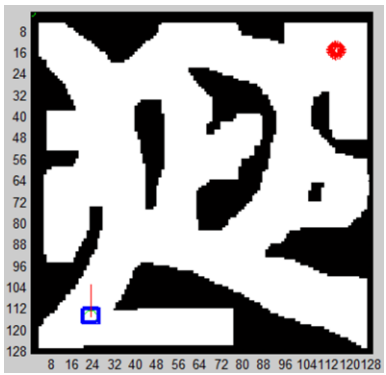


Figure 3-4: Illustration of one of the maps with the robot at a starting location and orientation, and the destination is shown as a "red-star".

As shown in Figure 3-4, the robot has a size that is larger than a pixel. Therefore, it is unnecessary, and sometimes incorrect to calculate the path for every pixel. With the consideration of the robot's size and to minimize computation, the map should be gridded into sub-portions. Keeping each grid size slightly larger than that of the robot, the map was divided into a 16 by 16 grid map. For each grid, 8 by 8 pixels were given one value which is the average value of all the pixel intensities within the grid. This value was then divided by 64 and rounded to normalize the integer value to 0,1,2,3, and 4; where 4 is almost completely white pixels, and 0 is completely black pixels. Equation (3-1) describes this grid calculation; where A is the average value, x and y are the pixels inside the grid, gx and gy are the grid coordinate, i is the intensity value of the pixel, and N is the number of pixels in one row or column. Since it is an integer, any decimal value will be cut-off, hence the addition of 0.5 to round up any number that is 0.5 higher than its whole value (3.5 = 4, while 3.4 =3).

$$A(gx, gy) = 0.5 + \frac{1}{N * N}\sum_{x,y}^{N} i(x, y) \tag{3-1}$$

Figure 3-5 displays the same map as Figure 3-4 with the new gridded values displayed on each grid.

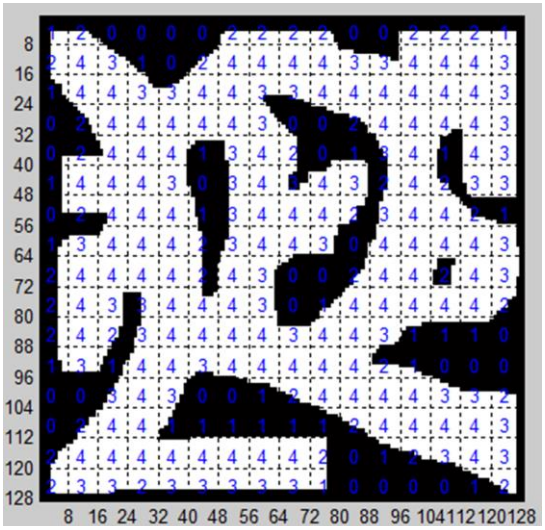


Figure 3-5: Illustration of one of the maps that is gridded and given a value from 0 to 4.

It is evident from Figure 3-5 that blocks with a value of 4 are the ideal “stepping” or “driving” regions; all other regions have a higher chance of collision. Implemented algorithms considered only blocks with a value 4 as navigable space.

For navigation algorithm tested on the actual robot, the robot was placed on a smooth floor with obstacles. Obstacles are any 3D objects that hinder the motion of the robot. They may vary in size, weight, and shape.

The intelligence engine was setup as a software addition to the implemented navigation algorithm. No additional hardware or environment setup was required to test the ANN.

The simplified version of the entire algorithm implemented on the robot is presented in Figure 3-6 as a flow chart.

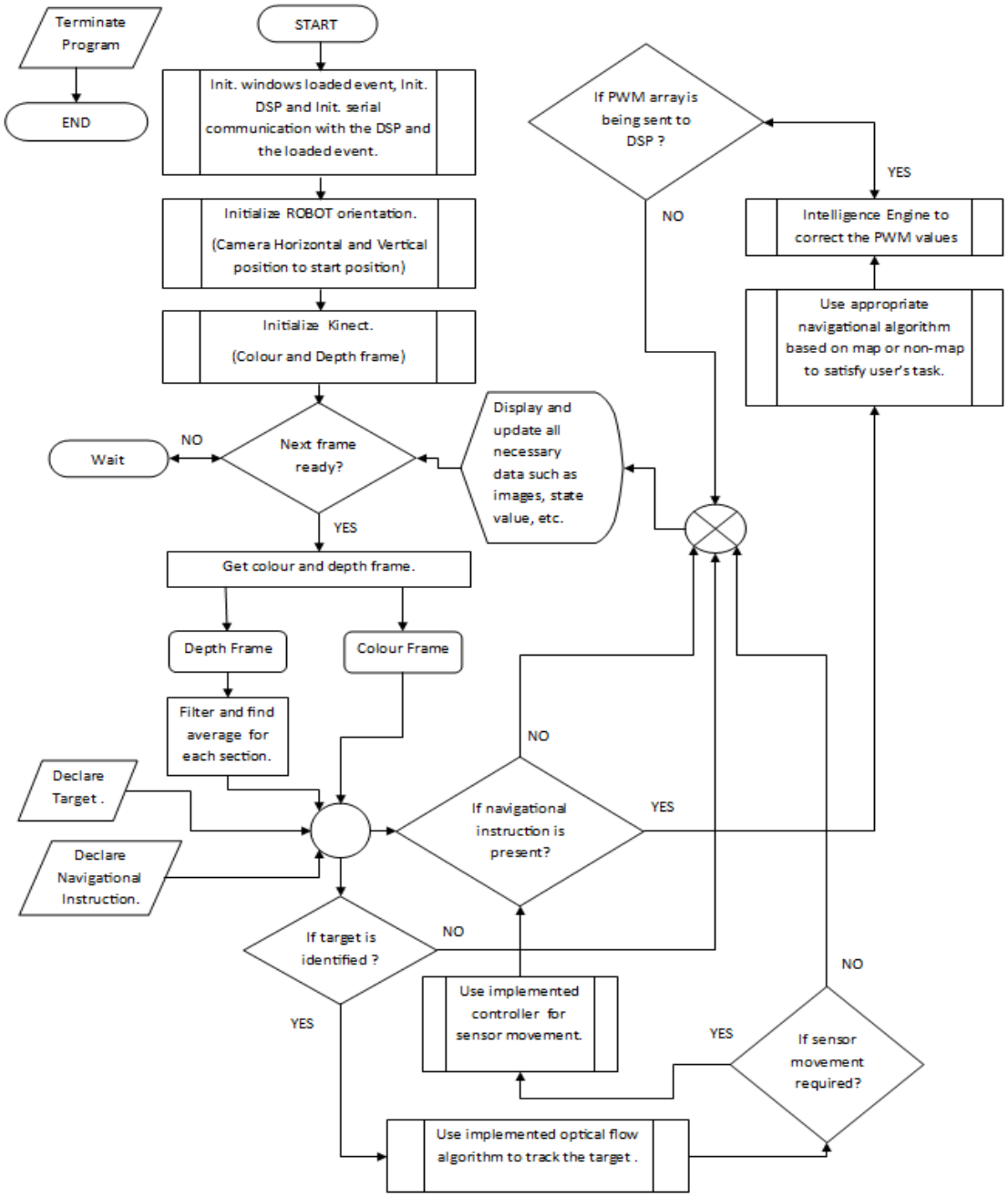


Figure 3-6: Simplified flow chart of the software algorithm implemented on the robot.

3.5 Software Libraries

There are two software libraries used for this project, EMGUCV and Kinect SDK. EMGUCV is an open source, computer vision, library that is a cross platform .NET wrapper for OpenCV. This library is great in assisting with image processing tools such as real-time image drawing, generic image classes, automatic garbage collection, XML serialization, and pixel operation.

The second library used was the Microsoft Kinect SDK (MKSDK). MKSDK provides the tools and application program interface (API) that is native and updated to develop application with Kinect. It is also a key tool in acquiring colour, depth, sound, and other embedded sensor data from the Kinect sensor.

Chapter 4
Experimental Results

4.1 Optical Tracking

In this chapter, the experimental results of the topics explored in Chapter 2 will be presented. A conclusion will be drawn from the observed results for the different topics explored.

4.1.1 Colour and HSV Tracking

The HSV tracking algorithm was implemented as stated in Section 2.1.1. The results are illustrated in Figure 4-1.

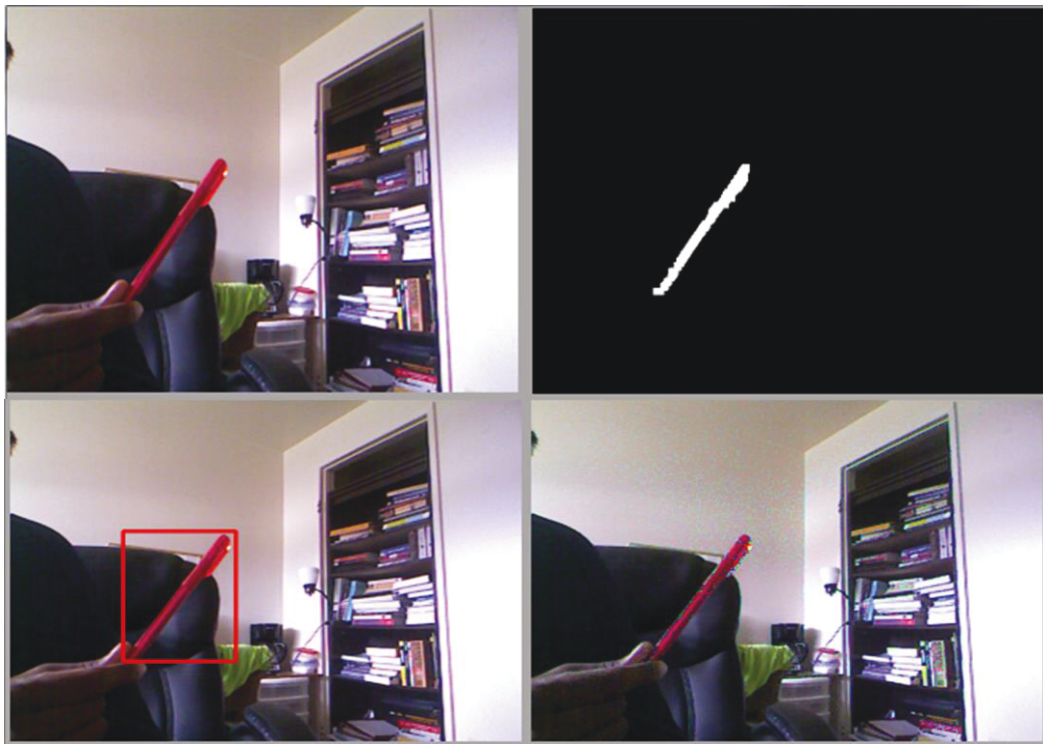


Figure 4-1: Filtered HSV image and targeting. Top-Left: original image. Bottom-Right: HSV image. Top-Right: filtered image. Bottom-Left: target drawn around the filtered area.

Tracking through HSV filtered detection is fast and effective. However, it is very limited in the sense that the target has to be primarily one colour. Also, the background cannot have anything of the same colour or there will be a false target. With lighting change or change in the angle the light hits the target, the observed colour of the target changes and this process will no longer accurately track or detect. This makes HSV approach useful in a very narrow range of situations.

4.1.2 Feature Detection Based Tracking

The SURF algorithm was implemented on the central processing computer (CPC) (refer to Section 3.3) and tested for performance time and accuracy. The set of experiments were performed on mostly modeled images that were cropped out of the original background images. The results are shown in Table 4-1. The images of the actual test can be found in Appendix A.

Table 4-1: SURF performance on models cropped from the loaded image

Image set (appendix A)	Accuracy	Time (milliseconds)	Resolution
Image set 1.0	accurate	343	640x480
Image set 2.0	accurate	1287	720x540
Image set 3.0	accurate	119	260x188
Image set 4.0	accurate	937	800x530
Image set 5.0	accurate	9416	2560x1600

As expected, these experiments proved that SURF detections were accurate. When the same test was performed on a video stream presenting an object in the real world, the results were not good. Even though there was almost no false detection, many times, there was no detection. Many of the detection were only possible when the object was very close to the camera, approximately taking up 50% of the image frame. The object must have a flat surface facing the camera and the texture of the object must be “non-smooth” for better detection. The SURF algorithm implemented was also not fast enough on the CPC to have 30

Frames per Seconds (FPS) frame rate (FR). To achieve 30 FPS, the object must be detected within 33 milliseconds at the latest. As shown in Table 4-1, the detection times were longer than 33 milliseconds; even for identical models. Therefore, feature based detection as a tracking algorithm was not ideal. However, it can be used as a means to identify a target once the mobile robot is very close to target.

4.1.3 Tracking Based on Change

Tracking based on change algorithm explored in Section 2.1.3 was implemented and tested as shown in Figure 4-2.

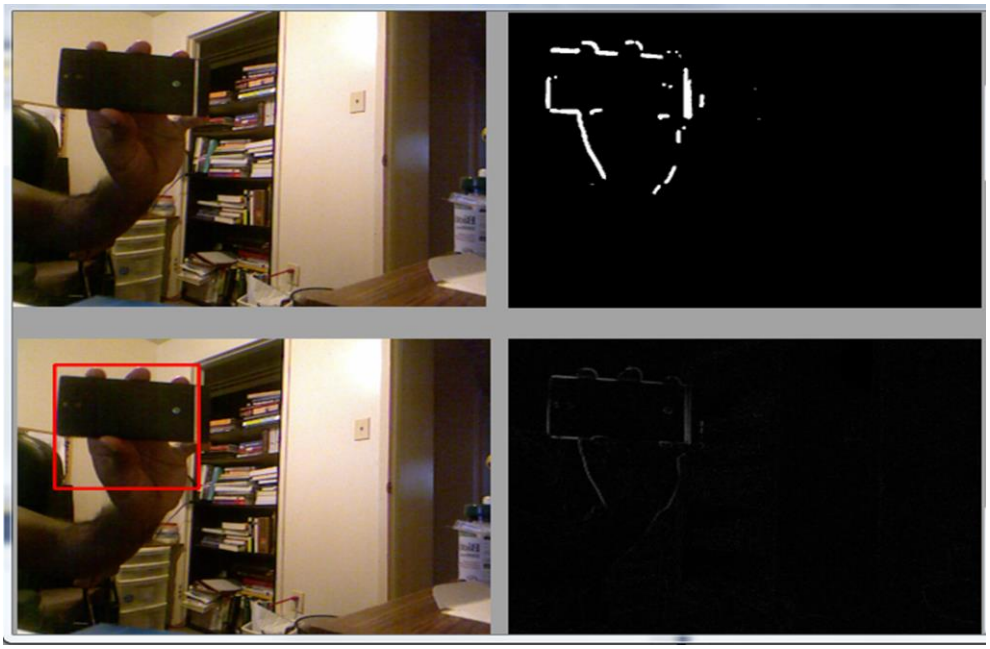


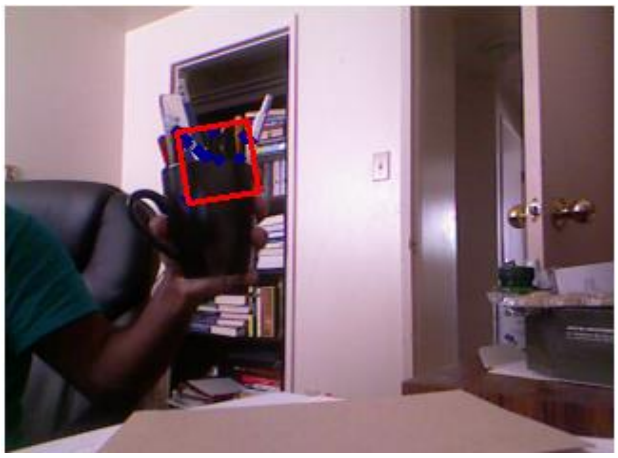
Figure 4-2: Image of GUI with detected motion. For this demonstration, a phone was held and shaken by hand. The top left image is the original image. The bottom right is the gray scale image. Top right image is the blurred, black and white image. The bottom left is the final image with the detected motion.

This method of tracking is ideal for detecting motion in the environment. However, one of the biggest problems with this method is that it only detects and tracks motion. Therefore, it will not work on a moving platform. To detect motion, the camera must be

placed in a stationary platform so that all other motion in the environment can be detected. If the camera is on a moving platform, such as a robot, the entire frame will be perceived as motion due to change relative to the motion of the robot. Therefore, the algorithm was used as a method to obtain the original target, when the robot is stationary. For example, in “find me mode”, the user can shake an object, which will become the target. Once the target has been initialized; other tracking methods like optical flow can track the target thereafter.

4.1.4 Optical Flow

At first, this method was not working as predicted. Even though Lucas Kanade [9] approach reduces noise levels, it was still far too noisy for accurate tracking. Also, one or more features will not behave like the majority of the features within the target. These uncertainties made the new targeted area increase in size over time and not accurately follow the target. An illustration of this is shown in Figure 4-3.



Original target.

4 seconds after detection.

2 seconds after detection.

3 seconds after detection.

Figure 4-3: Inaccuracies in the implemented optical flow tracking.

The blue dots are the features detected then tracked. A red rectangle is drawn around all the features to represent the target as a whole. It is evident that the algorithm is not yet complete.

4.1.5 Implemented Solution to the Optical Flow Tracking Error

To solve this problem, one must filter out the features that are moving inaccurately and while keeping the features that are functioning well. Another way is to detect the

features that are moving accurately and superimpose those translations on the features that are not performing accurately. This can be achieved by identifying the translation from the majority of the features and imposing those on every feature.

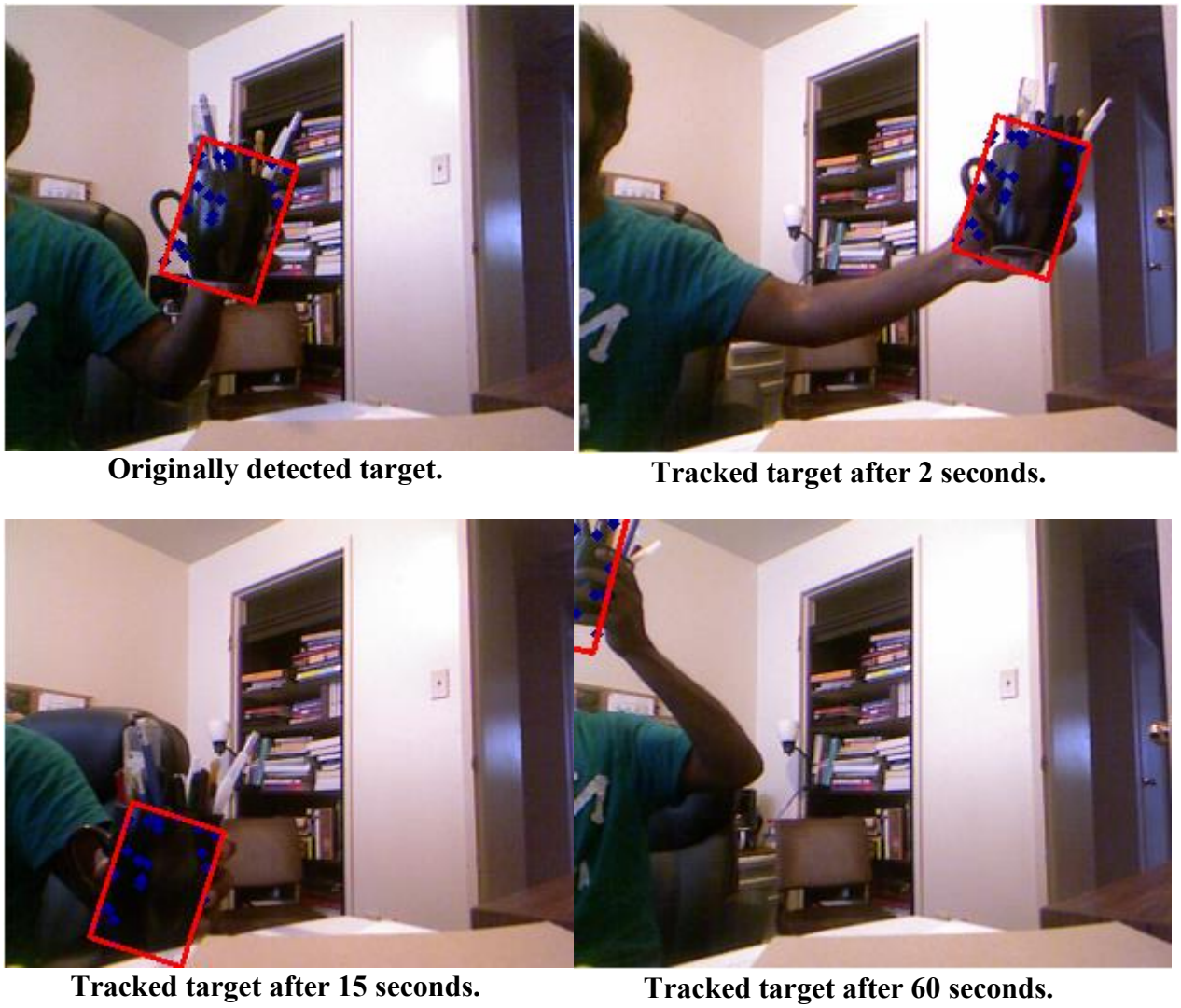
One cannot simply find the average of every detected feature translation and use that as the translation for all features. This will also give false translations over longer period of time, especially if the feature detection count is low. This was fixed by taking the median of all the current translation and averaging a body of features surrounding the median value. Then the translation of all the features was changed to the calculated accurate translation. This resulted in the best possible translation of the target.

To find the median, a sorting algorithm was used. For this thesis, the "bubble sorting" [60] algorithm was used. This algorithm compares adjacent pairs and swaps accordingly. It is looped through until the algorithm is able to go through all the adjacent values without having to swap once. The final addition to the optical flow algorithm is provided in Figure 4-4.

This final algorithm performed extremely well comparatively. It was accurate, fast, and used very little processing time. The recorded FR was still 30 FPS on average. The results from the final optical flow algorithm are provided on Figure 4-5. The target was tracked accurately through long periods of time and near the boarders of the frame. With a compromising note, if the object moves further away, the targeted area will not get smaller with this algorithm and there is more probability for false tracking at that point. However, a robot is more likely to detect a target and move closer to the target. Therefore, the target is more likely to appear bigger on the frame and not smaller. The targets tested did not have to be any given shape or colour, it can be an arbitrary region in the 3D space. With that given, it is recommended that the target occupies at least 10% of the full size of the frame for good tracking.

```
1. For each feature point, find the translation from previous
   frame to the current frame.
      a. deltaX[i] = currentFeature[i] – previousFeature[i]
      b. deltaX[i] = currentFeature[i] – previousFeature[i]

2. Use bubble sort to sort all the translations.
      a.  While swap is true
           i.  For every delta value
                 1. If deltaX[i] > deltaX[i+1], swap values.
                 2. If deltaY[i] > deltaY[i+1], swap values.


3. Find the average of the middle three values for x and y
   translations.
      a. deltaX[center-1]+ deltaX[center]+ deltaX[center+1]/3.
      b. deltaY[center-1]+ deltaY[center]+ deltaY[center+1]/3.


4. Impose that translation on every feature point and draw a
   rectangle around them.
      a. currentFeature[i]= previousFeature[i]+new translation.
```

Figure 4-4: Filtering algorithm used as an addition to the optical flow algorithm for more accurate tracking.



Originally detected target.

Tracked target after 2 seconds.

Tracked target after 15 seconds.

Tracked target after 60 seconds.

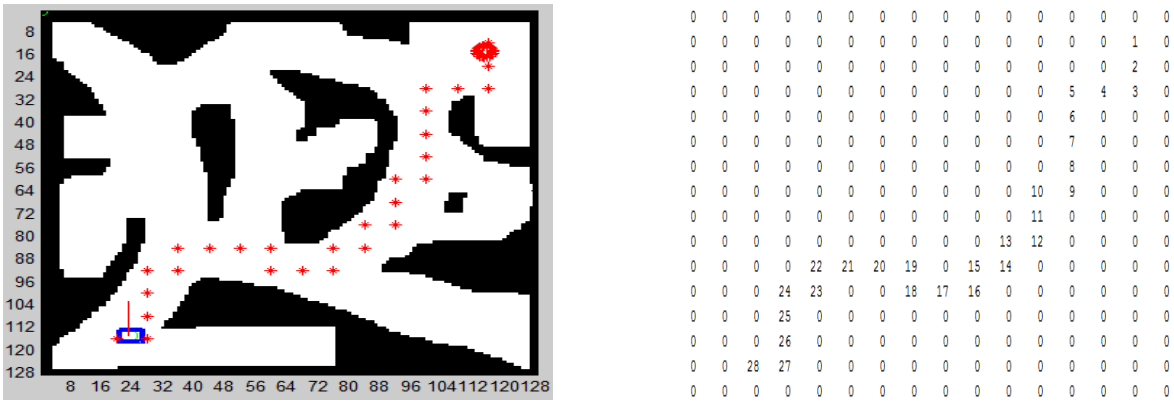
Figure 4-5: Performance of the implemented final optical flow algorithm.

4.2 Navigation

In this portion of the chapter, the experimental results of the topics explored in Section 2.2 will be presented. Experimental setup of the simulated map based navigation is explained in Section 3.4.

4.2.1 Modified A-Star (A*)

Modified A-Star Algorithm, Fgrid propagation and path numbering explained in Section 2.2.2 is displayed on Figure 4-6, (b). The corresponding path plotted on the map is shown in Figure 4-6, (a).



(a) **(b)**

Figure 4-6: (a) A-Star path plotted. (b) The corresponding matrix to the path.

It is evident from Figure 4-6 (a) that there are some points that can be navigated diagonally for an even faster path. To navigate diagonally at these scenarios; the robot checks the 8 blocks around its current block before moving to the next listed position on the path. It then chooses the block with the lowest number. This will ensure that the robot can take advantage of diagonal navigational opportunities, as illustrated on Figure 4-7.

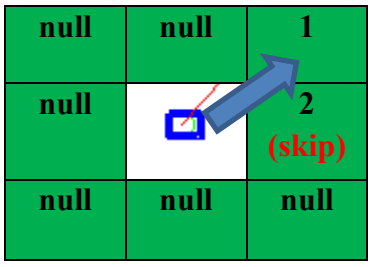


Figure 4-7: Diagonal navigation.

The diagonal navigation source code is provided in Figure 4-8. Note that "rPath" is the planned path matrix from Figure 4-7 (b), "pos" is the current position, and" nxtPos" is the possible next coordinate to move towards.

```
for j=pos(1)-1:pos(1)+1
    for k=pos(2)-1:pos(2)+1
        if (k>0)&&(k<gridDim)&&(j>0)&&(j<gridDim)
            if(rPath(k,j)>0)
                if (rPath(k,j)< nxtPos(1))
                    nxtPos(1)= rPath(k,j);
                    nxtPos(3)=j*8-4;
                    nxtPos(4)=k*8-4;
                end
            end
        end
    end
end
```

Figure 4-8: Diagonal navigation source code.

The velocities of the wheels were adjusted accordingly to first turn towards the next chosen point and navigate to it.

Four different maps were experimented on; each map had different sets of starting and ending points. The maps are named "Tree-road", "Maze", "Simple-line", and "Simple-circle". "Maze" and "Tree-road" are maps that are hard to navigate. They have dead-ends, sharp edges, multiple path options, and narrow areas. "Simple-circle" and "Simple-line" are much easier to navigate comparatively. They have large navigational spaces and very simple obstructions.

The experiments observed computation time for path planning, number of iterations taken to navigate to the target, and navigational time. Navigational results for every map using modified A-Star path planning are provided in Appendix C. Table 4-2 lists all the numerical results for modified A-Star path planning algorithm.

Table 4-2: Modified A-Star results for different maps and navigational coordinates.

Maps	Start position (x,y)	End position (x,y)	Path planning computation time (s)	Number of iterations	Navigational time (s)
Tree-road	23,115	115,15	0.034	287	45.460
	23,115	60,11	0.033	265	38.610
	115,115	20,20	0.039	218	27.178
	75,20	20,115	0.034	308	49.732
Maze	32,100	115,20	0.039	457	102.099
	16,25	115,30	0.042	326	58.651
	115,115	20,20	0.063	409	118.891
	20,20	20,115	0.039	294	47.156
Simple-circle	15,100	115,35	0.034	197	22.923
	50,115	50,20	0.048	231	29.767
	115,60	20,75	0.044	308	55.621
	20,20	115,115	0.040	238	31.636
Simple-line	30,30	115,115	0.055	283	42.467
	30,30	115,75	0.025	121	20.187
	15,100	20,20	0.037	146	24.031
	15,100	115,32	0.038	135	22.680

The implemented algorithm navigated to the target from the starting point on every attempt. The computation time for path planning was very low, which is low enough so that if necessary, path planning can be done every few frames. However, the path planned was not always the shortest path. This is even more evident in the “simple” maps. Also, the algorithm did not always behave well in sharp corners as the robot sometimes collided with some sharp corners. To avoid this in real situations, the safety bubble radius could be increased and/or the grid size could be changed.

4.2.2 Goal Based Navigational Vector Field

GBNVF algorithm explained in Section 2.2.3 is demonstrated as a vector field on the map below, Figure 4-9.

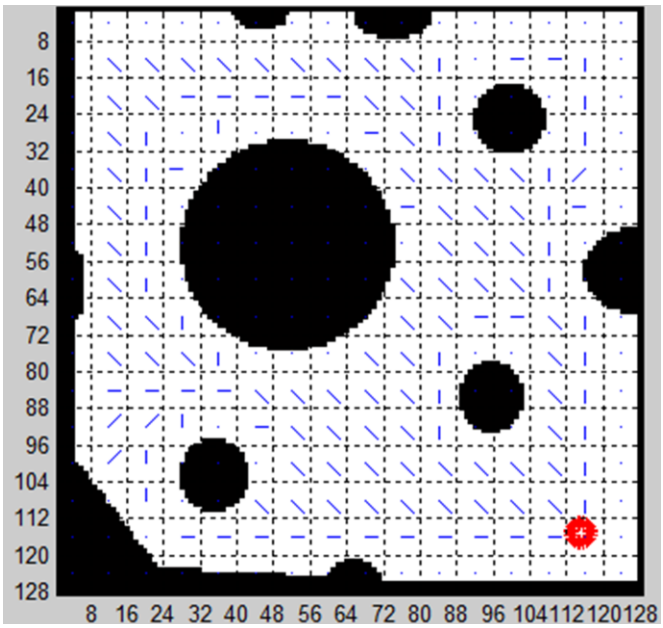


Figure 4-9: Illustration of the vector field created for the map Simple-circle with an arbitrary destination.

For navigation, the vector magnitude is used as a speed controller. The controller output is obtained by multiplying the current wheel speed with the magnitude of the vector that the robot is currently occupying. The vector direction is used to correct the robot's current direction. This in return ensures that the robot is driving slower around smaller areas and faster in open areas.

This path planning algorithm is tested on the same maps as the modified A-Star algorithm, with the same starting positions and destinations. The results are displayed on Table 4-3. The actual images of the navigation itself are provided in Appendix C.

Table 4-3: GBNVF results for different maps and coordinates.

Maps	Start position (x,y)	End position (x,y)	Path planning computation time (s)	Number of iterations	Navigational time (s)
Tree-road	23,115	115,15	0.046	280	134.689
	23,115	60,11	0.046	184	78.204
	115,115	20,20	0.045	243	110.604
	75,20	20,115	0.045	217	95.699
Maze	32,100	115,20	0.043	154	62.620
	16,25	115,30	0.048	158	64.987
	115,115	20,20	0.047	358	160.133
	20,20	20,115	0.033	262	97.594
Simple-circle	15,100	115,35	0.035	145	44.465
	50,115	50,20	0.044	158	54.453
	115,60	20,75	0.046	120	34.972
	20,20	115,115	0.048	187	69.436
Simple-line	30,30	115,115	0.036	163	68.711
	30,30	115,75	0.035	85	23.744
	15,100	22,22	0.046	156	57.419
	15,100	115,32	0.044	129	38.362

The algorithm navigated to the target for every scenario provided. The path planning computational time was very low. There were little to no collision scenario. The algorithm can be improved by having more grid blocks; an additional algorithm can also be added to have vectors at the edge blocks of every obstacle for better performance.

The two algorithms, A-Star and GBNVF, were compared on computation time, navigational time, and number of iterations taken. The graphs are provided on the following figures: Figure 4-10, Figure 4-11, and Figure 4-12. Each series in the graphs represents navigation from one location to a destination per map.

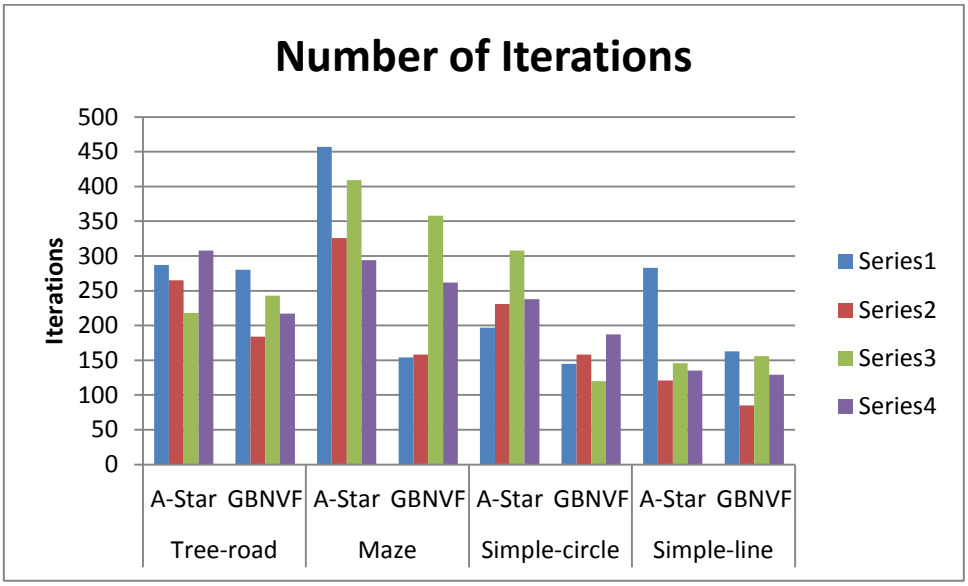


Figure 4-10: Comparison of A-Star and GBNVF by number of iterations taken to navigate.

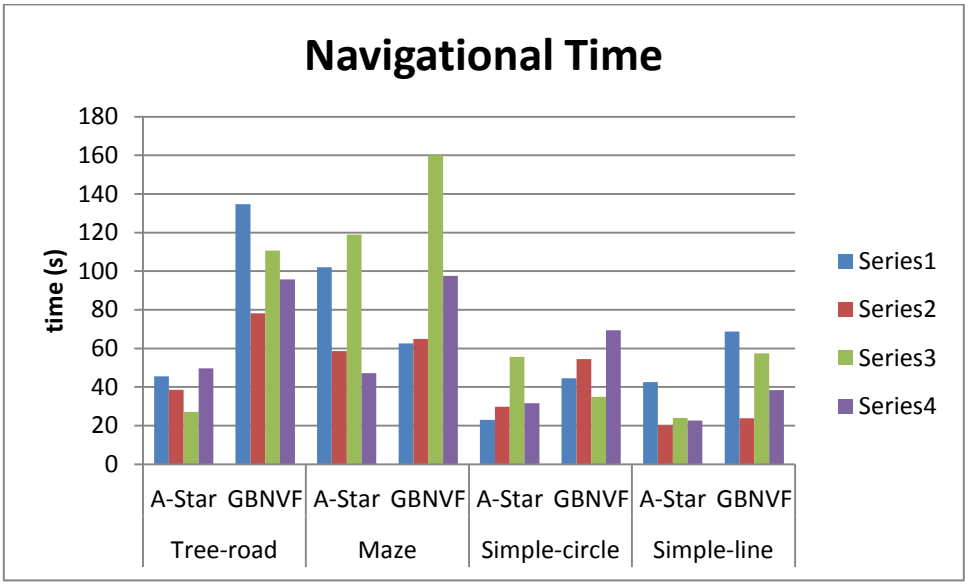


Figure 4-11: Comparison of A-Star and GBNVF by navigational time.

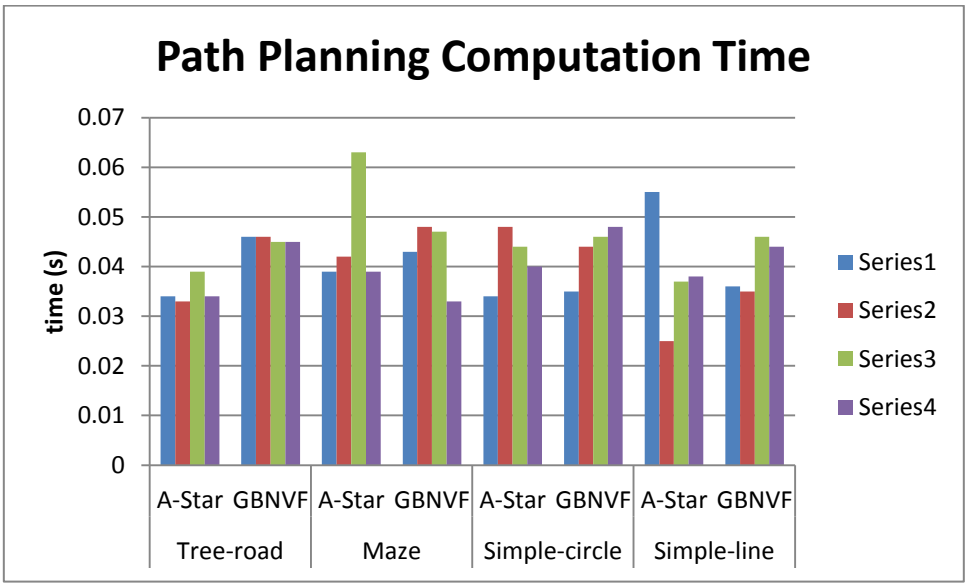


Figure 4-12: Comparison of A-Star and GBNVF by computational time.

From the comparison of the number of iterations taken, it is evident that GBNVF required fewer steps for almost every scenario. When comparing computational time, modified A-Star algorithm performed slightly better. The modified A-Star algorithm also has better navigational time in comparison with the GBNVF for the tested maps. However, when comparing the actual navigations on the maps, the GBNVF algorithm performed more stable while the modified A-Star algorithm seemed aggressive at time. The modified A-Star algorithm also had many collision scenarios near edges. Overall, GBNVF algorithm with some minor modifications can be argued as the better suited path planning algorithm for the real robot, if the local map, position, and orientation were provided.

4.2.3 Non-Map Based Navigation (NMBN)

The algorithms explored and tested in this portion of the thesis are modeled based on the assumption of having no map, or positional data for the robot. For simulation purposes, simple maps are experimented on. However, the robot does not comprehend its surrounding, except for what is directly visible to it. The experimental setup for the simulations on the explored algorithms was the same as the MBN setup, refer to Section 3.4. The simulated algorithm was implemented on the actual robot after.

4.2.4 Follow the Target

The “follow mode” is responsible for following a target. The idea is to navigate quickly and make the displacement between the target and the robot equal the threshold limit. Once the robot is at the threshold distance away from the target, the goal is to maintain that distance. An algorithm based on the horizontal position of the target is implemented to turn the robot towards the target. Once the target is centered, the robot drives forward based on a PID controller. If the displacement between the target and the robot is larger than 2.5 meters the robot will drive at the set max speed. If the displacement is smaller than 2.5 meters, the PID controller will decrease the speed proportionally until the distance is smaller than 1.0 meter. At that point, the robot must stop. The ideal distance to follow the target is therefore between 1.5 meters and 1.0 meters.

4.2.5 Avoid Obstacles

This algorithm is responsible for avoiding all obstacles in the local area when in motion. There is no target destination to be reached and the robot is arbitrarily navigating the local area. An algorithm such as this can be used when the local area is getting mapped.

This algorithm works by measuring the displacements at different regions of the frame. The frame is divided into seven different rectangular regions, each varying in size. The average distance is calculated for each of the region to have one crisp value per region. Figure 4-13 shows a frame with arbitrary obstacles and how the regions are divided. The regions all have the same height. Horizontally, they are 50, 40, 30, 80, 30, 40, and 50 pixels wide from left to right, respectively. Each region is divided to compensate for noise from the depth data and to emphasize the importance of the respective regional data. For example, the depth information directly in front takes priority over other directions.

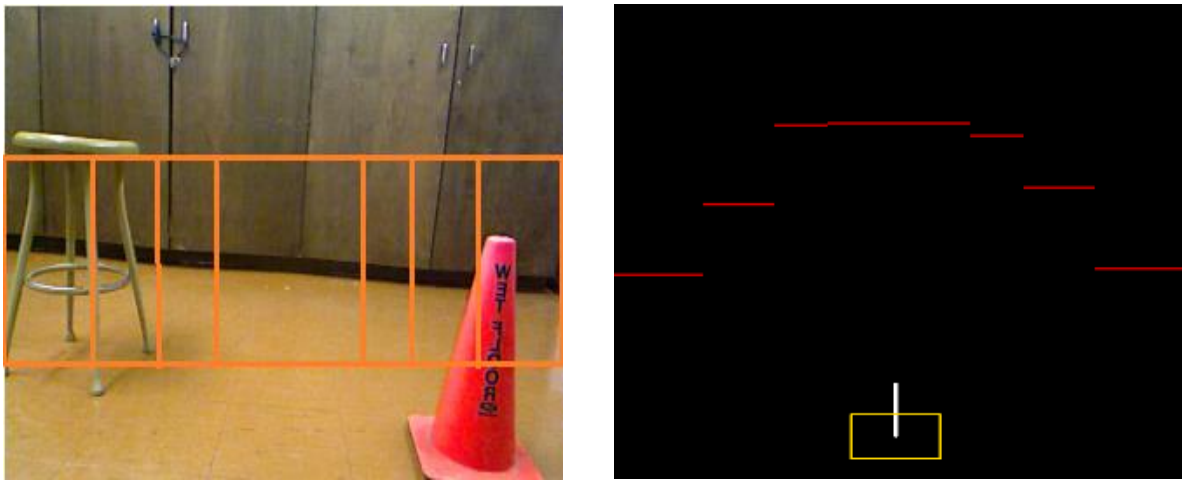


Figure 4-13: Left: Demonstration of how a frame is divided to find a crisp value for each region. Right: GUI representation from the robot's perspective.

Each region can be viewed as a "range finder" of a given area. Using each of the range values, an algorithm can be written to avoid all obstacles. The algorithm is modeled by Equations (4-2) and (4-3). Note that the seven range finders are labeled L1, L2, L3, C, R3, R2, and R1 from left to right respectively. pR and pL are the power given to the right and left motors, respectively. sr is the speed-ratio to control the speed of the robot. I, J, K, L, and M are all tuning variables.

$$\text{Check} = \text{R1 \&\& R2 \&\& R3 \&\& L3 \&\& L2 \&\& L1}$$

$$\text{pR} = \begin{cases} \left(\frac{I}{R1} + \frac{J}{R2} + \frac{K}{R3} - \frac{L}{L1}\right) * C * sr & \text{If check} > \text{threshold} \\ \frac{M}{C} & \text{If check} < \text{threshold} \end{cases} \quad (4\text{-}2)$$

$$\text{pL} = \begin{cases} \left(\frac{I}{L1} + \frac{J}{L2} + \frac{K}{L3} - \frac{L}{R1}\right) * C * sr & \text{If check} > \text{threshold} \\ -\text{pR} & \text{If check} < \text{threshold} \end{cases} \quad (4\text{-}3)$$

On the actual robot, a minimum of 85% PWM to each wheel was necessary to achieve a stationary turn from stop, on a smooth floor.

This algorithm was modeled and simulated on a sample map to observe the robot avoiding collision. The results can be seen in Figure 4-14. Note that the lines are drawn in front of the robot representing the visual range finders.

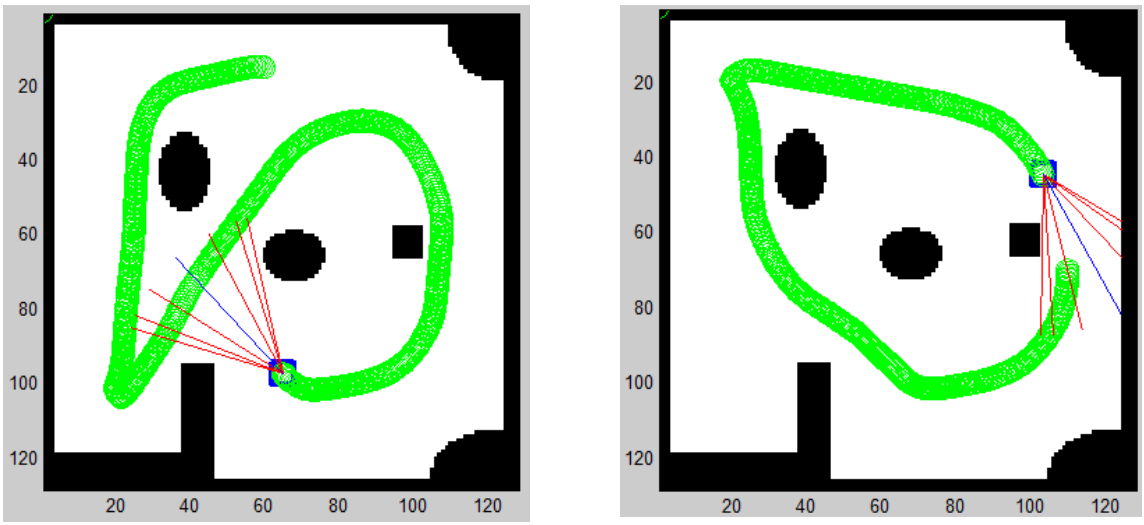


Figure 4-14: Robot using the avoid algorithm to navigate around arbitrary obstacles.

The simulations were successful in avoiding collision in the simulated scenarios. The simulation even performed well in navigating out of a local minimum scenario, as shown on the left, Figure 4-14. The same algorithm was then implemented on the real robot with different values for the tuning parameters.

4.2.6 Navigate to Target

"Simple-navigation mode" is responsible for navigating to a target at a calculable distance away while avoiding basic obstacle(s). This is a combination of "follow target" and "avoid obstacles" algorithms. The main problem is that the robot cannot deviate too far from the target angle when avoiding an obstacle. This is because if the target is lost from the frames, the robot will not be able to redetect. This would mean that the target must be visible at all times and the turning trajectory must be small. Due to this, the avoid algorithm is modified by an auto-correcting factor that finds the difference in the target angle and current orientation, as illustrated in Equation (4-4). Where θ is the current angle and Φ is the target angle, ar is the angle correcting ratio, and overall speed-ratio, sr, is controlled by the

displacement between the robot and the target similar to the "follow the target" algorithm. Range finders are labeled L1, L2, L3, C, R3, R2, and R1 from left to right respectively. pR and pL are the power given to the right and left motors, respectively.

$$\begin{aligned} \text{pR} &= \left(\frac{I}{R1} + \frac{J}{R2} + \frac{K}{R3} - \frac{L}{L1}\right) * C * sr - (\Theta - \Phi) * ar \\ \text{pL} &= \left(\frac{I}{L1} + \frac{J}{L2} + \frac{K}{L3} - \frac{L}{R1}\right) * C * sr + (\Theta - \Phi) * ar \end{aligned} \tag{4-4}$$

This algorithm was first simulated with one or multiple obstacles. As limited as it may be on changing trajectory, the algorithm performed well under simulated circumstances. The results are shown in Figure 4-15.

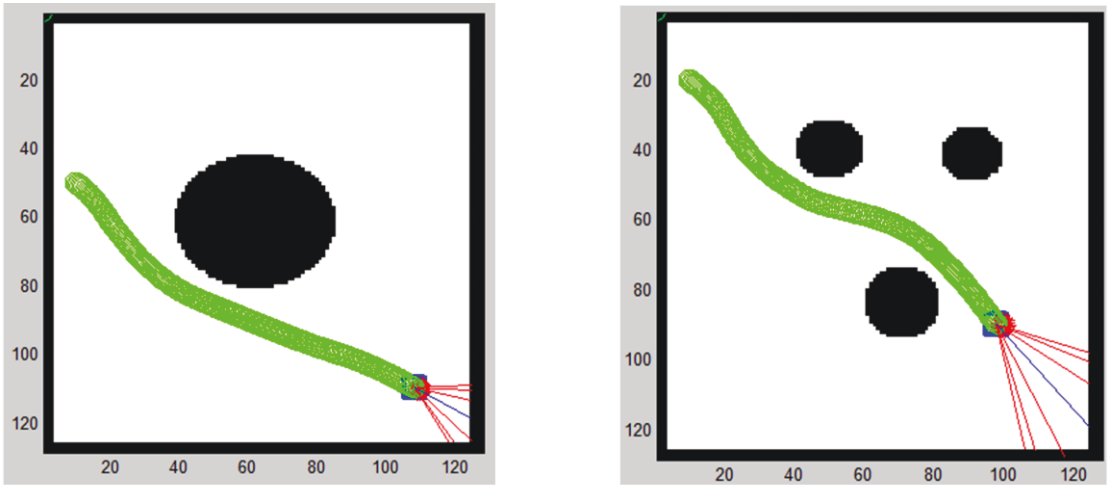


Figure 4-15: Robot navigating to a target while avoiding obstacles.

The idea is to change trajectory slightly while the obstacle is still a significant distance away. Both avoid-obstacles and navigate-to-target algorithms were retuned and implemented on the robot. However, the robot did not behave in a similar way when applied on the floor. There was some wheel slip due to traction but, this did not affect the robot's ability to navigate much. The major issue was that when a PWM of zero was applied to only one of the motor, that wheel did not stay neutral in motion. The gear ratio between the motor and the wheel did not effectively translate the rotational power provided to the wheels. As a result, if one wheel was in motion, the other will also rotate automatically causing a linear motion. Any small angular deviation applied by the navigational algorithm is considered

ineffective. Also, the power delivered to each wheel is not an accurate representation of the power sent.

A solution was implemented to fix this problem without changing every navigational algorithm simulated. The implemented solution is an intelligence engine that learns the behaviour of the robot and adjusts the PWM applied to the motor accordingly to produce positional and orientation translation desired by the algorithm.

4.3 Intelligence Engine (IE)

The training dataset was measured by applying a certain PWM value to each wheel and measuring the normal displacement, orthogonal displacement, and the change in angle. The change in time was set to a constant. These values were then normalized to have a value between -1 and 1. The same PWM values were tested for both wheels separately. Figure 4-16 shows the normalized PWM values applied to both wheels and Figure 4-17 shows the normalized displacement values for the corresponding iteration. The x-axis presents the corresponding iteration set for both graphs.

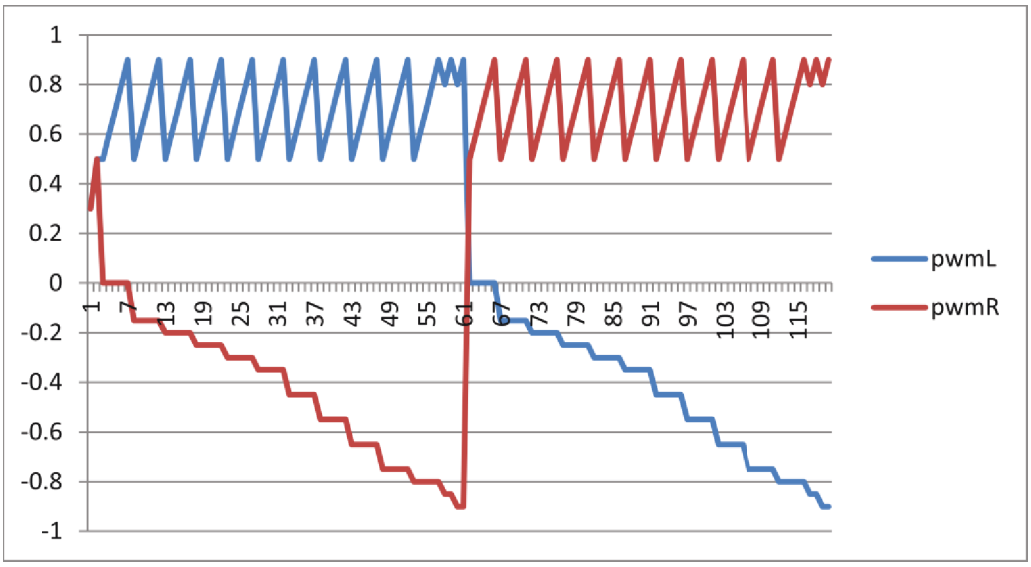


Figure 4-16: PWM values tested on each wheel to acquire robot's motion behaviour. First half of the data set included PWM for the left motor positive and the right motor negative. The other half, PWM values were flipped.

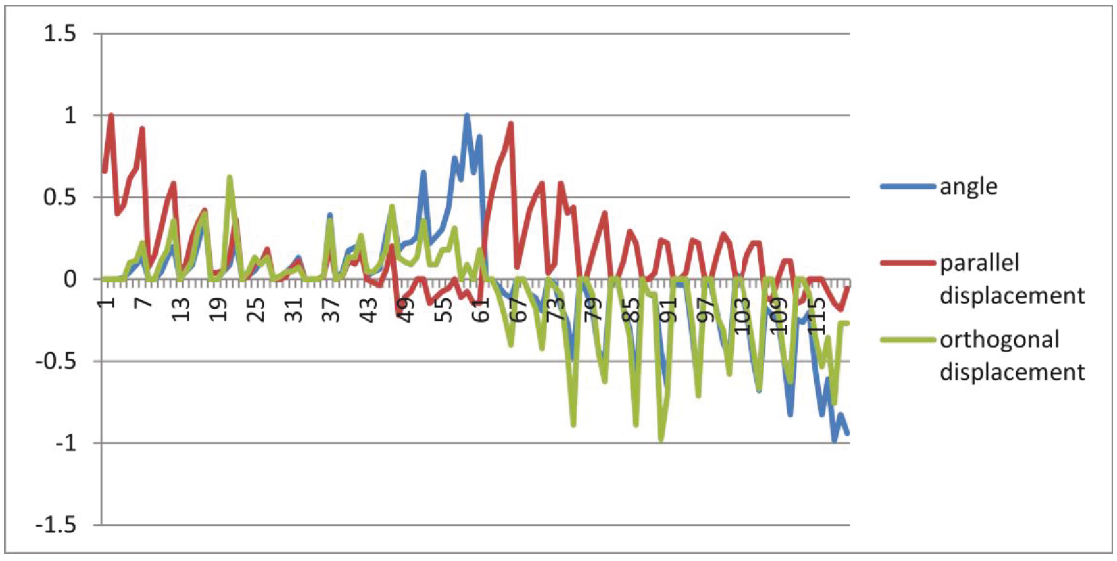


Figure 4-17: The corresponding displacement and change in orientation to the applied PWM values.

As it is evident from Figure 4-17, the robot does not behave in a linear fashion. Now to train the FFANN more accurately, the dataset was divided into two halves and trained separately. The weights for each connection were stored separately and are called upon,

based on the directional element of the intended angle. To test the accuracy of the FFANN, the same values for angle, parallel displacement, and orthogonal displacement from the dataset were provided as the input to the FFANN. The output PWM values are displayed on Figure 4-18.

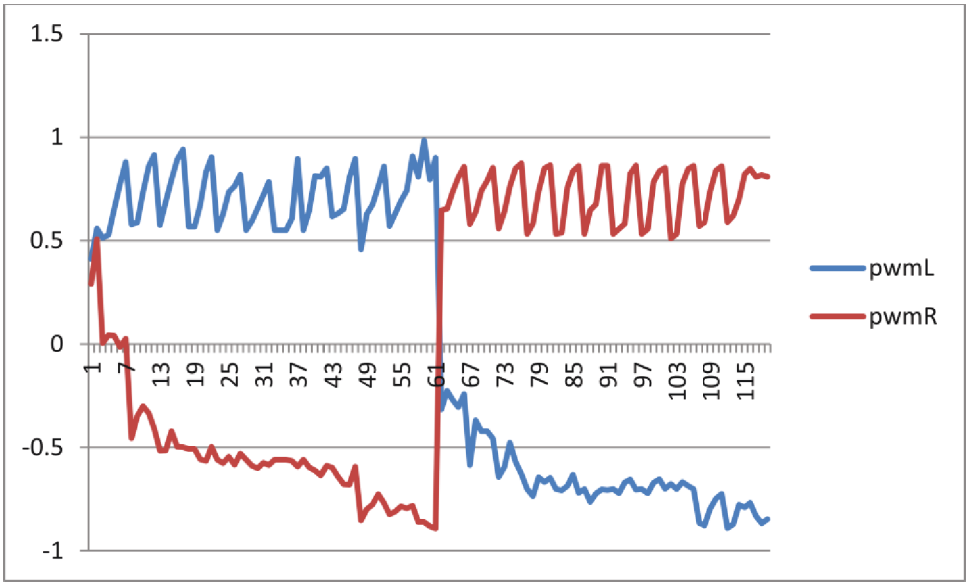


Figure 4-18: The PWM values provided by the FFANN from the original trained data-set.

The output provided by the FFANN is acceptable for the non-linear system trained. This system was stored as the intelligent engine that provided the correct PWM values based on the desired navigational behaviour. The dataset was measured manually at this time. However, if the robot was fitted with a gyroscope and accelerometer, the intelligence engine can constantly adapt to the robot through past and current navigational behaviour; including terrain changes.

With the addition of the intelligence engine, the robot behaved more accurately for the few tested scenarios. After the initial training, the addition of the IE requires negligible amount of computation time. Therefore, the FR and the speed of the entire system are not affected by the additional computation.

4.4 Conclusion

In this chapter, many different means of optically detecting and tracking methods explored in Chapter 2 were tested. Many different applicable algorithms were demonstrated and tested. First, re-detection on every frame as a primary means of tracking a target was tested. However, there were many drawbacks including higher computation, limited targeting, and inaccuracies. They may be integrated with the main algorithm for a more refined flexible algorithm.

HSV and colour detection was a fast and efficient method of tracking a single target. However, the colour of the target must consist of primarily one colour. The background cannot contain the same colour as the target or there will be a false target. Also, if there is a change in lighting in the local area or change in the angle of light reflection, the target's colour was perceived inaccurately. Since this was the targeting solution implemented by the previous student, it was important to explore the possibilities of this algorithm. With many drawbacks however, HSV tracking was not deemed to be effective as the main tracking method.

Feature detection was another tracking method explored. SURF was chosen to be the main algorithm tested in this area. Though SURF showed great accuracy at closer ranges; it failed to produce the same results for targets at far. It also required an abundance of computation time using a single processor. Due to the large computation time required by SURF, it was not possible to achieve, 30 FPS FR. It was evident that SURF as a tracking algorithm was not the most optimal option as the main tracking method.

Non-detection means of tracking was also tested on this chapter. These types of methods tested in this chapter used more satisfactory amount of computation.

Tracking based on motion was a good algorithm that detected the largest motion in the frame. However, it only tracked the largest motion, making any large motion the only possible target. It also required a stationary base to detect motion. If the base were to move, the entire frame was in motion from the camera's perspective. This causes some major inaccuracies. One of the major drawbacks is that, it will not be able to track a target that is stationary. For user friendly interfacing, this algorithm is used as a pre-processing algorithm

for identifying the target. When the robot is stationary, the user can shake an object and the robot will perceive it as the target. Optical flow algorithm is used to further track that object.

A more optimal method tested was optical flow. However, it was a combination of different algorithms that made this method very successful. First, by not tracking the optical flow of the entire frame, large amounts of computations were saved. Use of Harris corner algorithm to detect good feature points in a given targeted area enabled the optical flow algorithm to only track a few tens of points instead of hundreds or even thousands of points. Then Lucas-Kanade method helped solve the aperture, ambiguity problem. Finally, an algorithm was implemented to eliminate noise and inaccurate tracking by following the majority of the flow direction.

It produced very accurate results when tracking many arbitrary targets. There were minimal restrictions to choosing the target. As long as it was an appropriate sized target (original target size is at least 10% of the frame) and not completely smooth in texture, it can be tracked. It can be concluded that Harris corner-Optical flow-Lucas Kanade with median-mean motion filtering algorithm is the best suitable main tracking algorithm for this thesis. At closer ranges, SURF algorithm can be used for target identification.

For MBN, two different algorithms were tested; modified A-Star algorithm and goal-based navigational vector field simulations were done on the same maps with identical starting and destination positions. The results showed that both algorithms performed well enough to plan a path that will lead that robot to the final destination. Modified A-Star algorithm had better navigational times as well as computational time in comparison to the GBNVF. However, modified A-star algorithm sometimes collides with edges and didn't necessarily find the shortest path given a simple navigational area. GBNVF algorithm performed slightly faster in when comparing the number of iterations. GBNVF also required a slightly heavier computation load due to calculating the entire map. However, when comparing the navigation simulated on the maps, GBNVF performed more stable and appeared more reliable. GBNVF can also be used as an environment for multiple robots navigating in the same environments. Analyzing the two algorithms, it can be argued that GBNVF is a slightly better approach to navigation for the purposes of this thesis.

The NMBN algorithms explored worked well in simulation. When applied to the actual robot, they did not perform as predicted due to the dynamics of the robot.

A FFANN topology was proposed as a suitable solution and trained on a dataset measured from the robot. The dataset contained change in orientation angle, displacement in parallel direction, and displacement in the perpendicular direction for a set of different PWM inputted. The obtained data proved that the robot behaved in a non-linear fashion and non-symmetrical.

The created FFANN was programed along with the BP algorithm. In fact, many different models were tested and the one with the best results was presented. For more accurate results, the dataset was split in half and trained separately. The weights of the connections from each training set were stored separately. Based on the directional value from the change in angle at the input node, the corresponding set of connection weights was chosen.

The intelligence engine was tested for accuracy by providing the change in angle, parallel displacement, and perpendicular displacement at the input layer. The IE behaved in an acceptable way for the tested scenarios only. It is still not clear if the ANN will behave as intended for other situation due to lack of data from the results. It should be further expanded and tested for accuracy in other situations, as part of future work.

Chapter 5
Conclusion

5.1 Conclusion

Different methods of optically detecting and tracking were tested. Many different applicable algorithms were demonstrated and tested. Results from detection methods showed that they were not the most appropriate methods for the task at hand.

HSV and colour detection was considered ineffective to be the main tracking algorithm. Feature detection based SURF algorithm was proved to be accurate in 2D scenarios at close ranges. When dealing with 3D objects in 3D space, it did not perform accurately. It also required more computation power, which required more than 33 milliseconds to process. This means the next frame obtained will have to wait to be processed, dropping the FR below 30 FPS.

Non-detection methods of tracking performed well with respect to computation required. Motion tracking algorithm tested produced good results. However, it was only able to tracking anything in motion. It was also incapable of functioning when the mobile robot is in motion. Therefore the algorithm was used as a pre-step to tracking a given object. An object can be shaken in front of the robot and it will identify that object as the target, if the robot is in “find me mode”. The detected target is then taken over by the implemented optical flow algorithm.

A combination of Harris corner detector, Lucas-Kanade, optical flow with median-mean filtering algorithm produced the best results for tracking. The algorithm was capable of tracking any objects in 3D space. There were minimal restrictions to choosing the target. As long as it was an appropriate sized target and not completely smooth in texture, it can be tracked. In addition, different controllers for stable and fast tracking were tuned and tested to help the tracking algorithm maintain a visual reference pass the original borders.

The navigational ability of the robot in different environments was also tested. For MBN, two different algorithms were tested: modified A-star algorithm and GBNVF. A-Star algorithm sometimes collided with edges and didn’t always find the shortest path. The

GBNVF algorithm had a higher computational time since it calculated paths to the destination from every position of the map. GBNVF was also slightly faster based on the iterative steps. In comparison to the simulated modified A-Star algorithm, the GBVF algorithm can be argued as the better option for this thesis.

Navigate to the target and avoid obstacles algorithms performed well in simulation. When the algorithms were retuned and tested on the actual robot, it did not perform as expected. A solution was to learn the robot's behaviour to perform motion as desired by any simulated algorithm. As a solution, a FFANN was created using BP algorithm to update the weight. Once this network was trained, it was implemented as the intelligence engine that provided the correct PWM values. The performance of the intelligence engine was satisfactory for the few scenarios tested. More testes need to be done to show that the intelligence engine is capable of adapting and performing in different environments and situation.

There were difficulties and challenges encountered throughout this thesis. Some of which were due to the structure and design of the physical robot. Sometimes, one motor will receive much more power than the other, causing one wheel to not move at times. The screws became unsecured faster than expected. The robot body was designed in a way that was not easy to assemble and disassemble.

This thesis required topics from a variety of fields. One must be familiar with electrical engineering when dealing with the circuitry and embedded system design. The programing aspects, including tracking and navigation, can be considered as software engineering and computer science. This thesis also considered designing, building, and tuning controllers such as fuzzy, PID, and ANN. At some point, parallel computing was considered to optimize the implemented algorithms. However, it was abandoned due to time restrictions. Overall, this thesis required learning an array of topics and integrating them together.

5.2 Future Work

The research and experiments done in this thesis covered many arrays of topics. Each topic can be focused on separately to optimize the implemented solutions or even find alternative solutions that may be more efficient for the task at hand.

This thesis is intended for a mobile robot. Therefore, the entire algorithm should be transferred onto a more compact processing system or a micro-processing system. Mobile phones are capable of having up to eight cores, processing at more than 2.0 GHz each. Transferring the algorithm in a capable compact system will save a lot of power and the difference in weight.

An additional algorithm can be implemented to map the local area as the mobile robot navigates around.

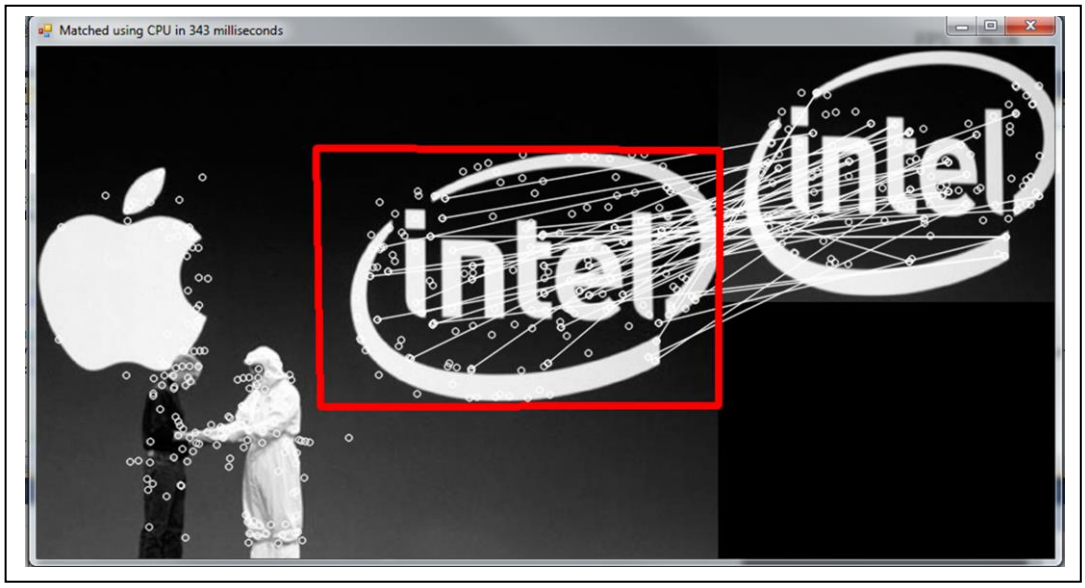
Fragmentation and swarm optimization are good areas to look into for optimizing the optical tracking algorithm.

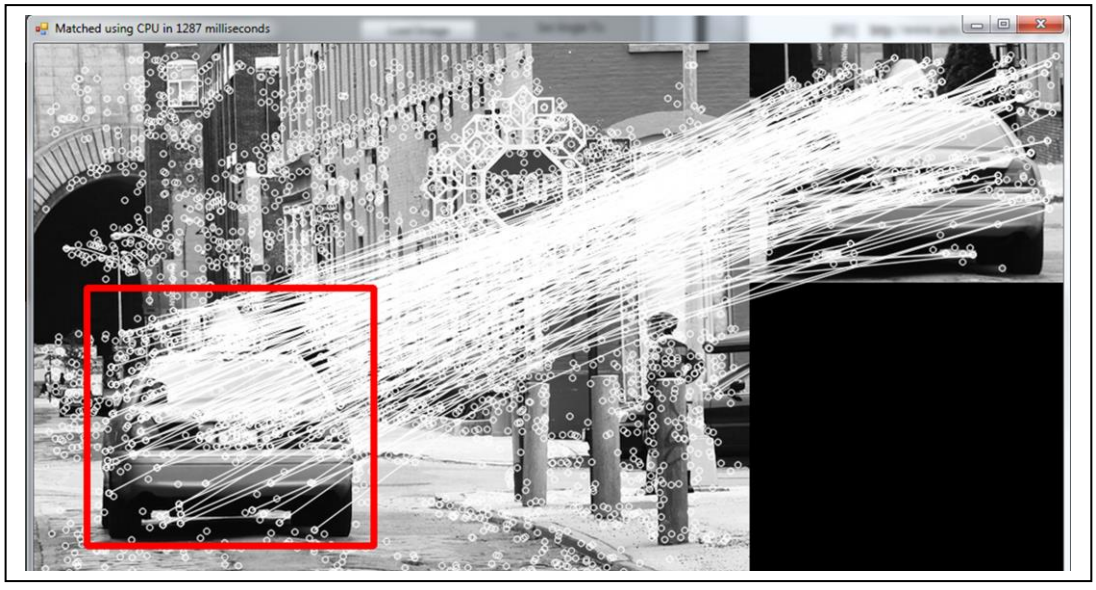
Another area for improvement is parallelizing the entire algorithm. An algorithm capable of parallel computation can exploit the different processing cores available in the CPU or the General Purpose Graphical Processing Unit (GPGPU).

Appendix A

Images Used for Optical Tracking

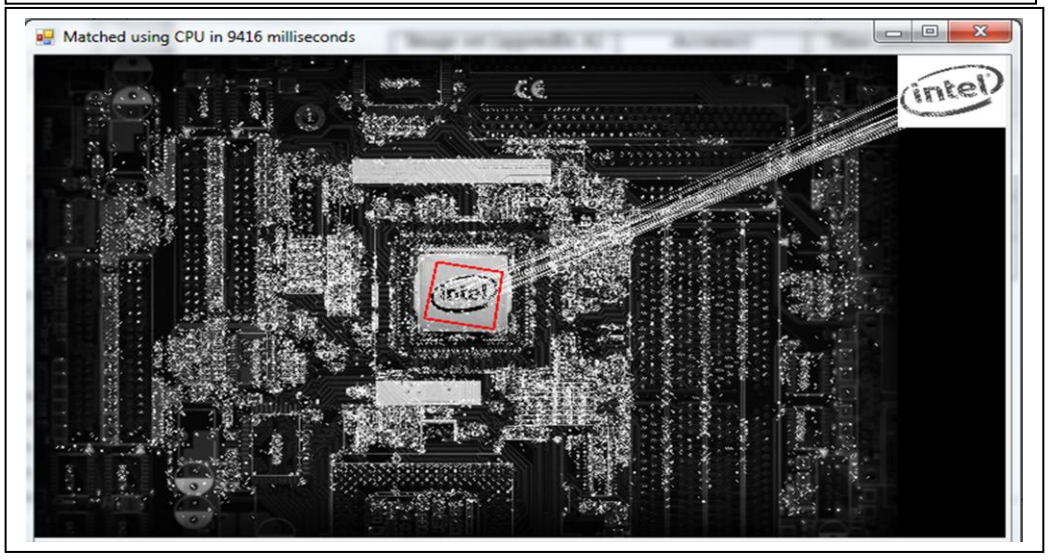
Images Used for Testing SURF Performance:

1. 

2. 

3. 
Matched using CPU in 119 milliseconds
STOP

4. 
Matched using CPU in 937 milliseconds

5. 
Matched using CPU in 9416 milliseconds
intel
intel

Appendix Chapter B
Controller Design and Results

B.1 Introduction

The isolated closed-loop systems from the robot were perceived to be non-time dependant and linear. Therefore, the closed-loop system can be modeled in the Laplace domain as illustrated in Equation (B-1) [29].

$$H(s) = \frac{P(s)C(s)}{1 + P(s)C(s)F(s)} \tag{B-1}$$

Where H(s) is the closed-loop transfer function (output/reference), C(s) is the controller transfer function, P(s) is the transfer function of the plant to be controlled, and F(s) is the feedback transfer function from the output sensor. The closed-loop block diagram is displayed on Figure B-1.

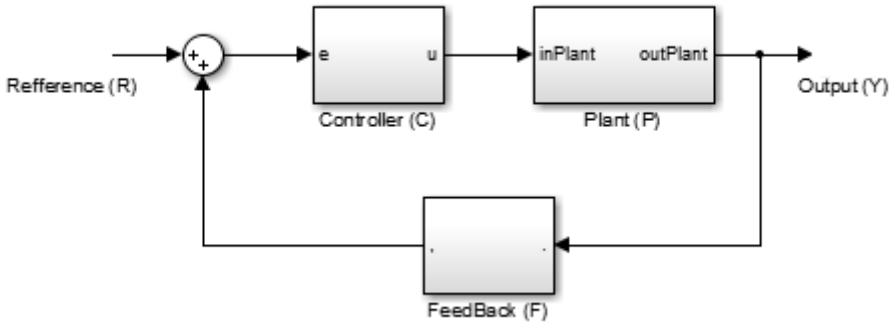


Figure B-1: Closed-loop system block diagram.

B.2 Version One

For version one of the robot, controllers were only tested for optical sensor tracking control for horizontal displacements. First, a proportional (P) controller was implemented, followed by a Proportional-Integral (PI) controller, and then a Proportional-Integral-Derivative (PID) controller was implemented. Finally, a fuzzy controller was implemented. The object of the controller is to take the tracked/detected target and center it in the image

frame. A faster performing controller will aid the tracking algorithm by "cushioning" the velocity of the target's motion. However, with an unstable controller, it will hinder the tracking performance drastically, so stability is emphasized. The feedback used is the observed location of the target from the optical sensor. The controller modeling and results obtained can be seen in the following section. It is also important to note that all tuning were done, except for the fuzzy controller, based on Table B-1 [33].

Table B-1: Effects based on gains from the different parameters

Parameters	Rise time	Over-shoot	Settling time	SSE	Stability
Kp	Decrease	Increase	Small change	Decrease	Decrease
Ki	Decrease	Increase	Increase	Eliminate	Decrease
Kd	Small Change	Decrease	Decrease	No change	improvable

B.2.1 Proportional (P) Controller

Proportional controller is the simplest controller implemented. It takes the feedback from the optical sensor and compares to the reference location (160 pixels). The feedback is the horizontal center of the tracked target. The difference from the reference and the observed will then be multiplied by a proportional gain (Kp). Proportional term is modeled in Equation (B-2) and Equation (B-3).

$$u(t) = K_p e(t) \tag{B-2}$$

$$U(s) = K_p E(s) \tag{B-3}$$

With that given, an algorithm can be implemented in code to replicate this controller, refer to Figure B-2. The output will be the PWM sent out to the DSP.

```
public short Pcontroller(double refference, double feedBack, double Kp)
{
    double feedBackError = refference - feedBack;
    short outPut = (short)(feedBackError * Kp);

    if (outPut > 100) { outPut = 100; }
    else if (outPut < -100) { outPut = -100; }

    return outPut;
}
```

Figure B-2: C# code of proportional controller.

The value for Kp was experimented with to obtain smooth tracking. The best results were with Kp =0.5. The results obtained can be seen on the graph below, Figure B-3. The steady-state error was 5-10 pixels.

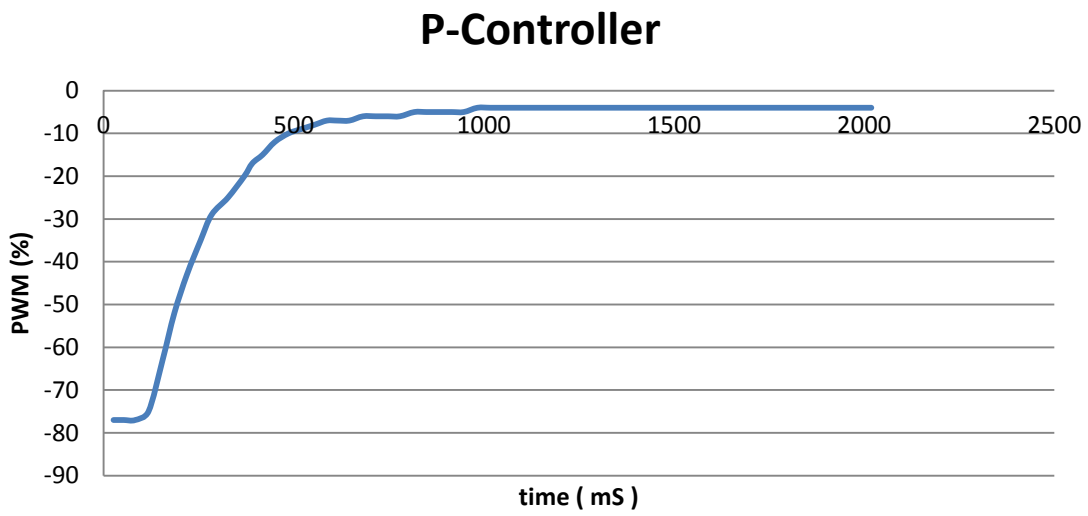


Figure B-3: Proportional controller response for horizontal tracking.

B.2.2 Proportional-Integral (PI) Controller

In addition to the proportional term, an integral term was added in the hopes of eliminating the steady-state-error (SSE). The integral term here is the sum of the

instantaneous error over time, multiplied by the integral gain. The new controller equation is illustrated in Equation (B-4) and Equation (B-5).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)dt \tag{F-4}$$

$$U(s) = K_p E(s) + \frac{K_i E(s)}{s} \tag{F-5}$$

This controller was implemented through code; refer to Figure B-4. The output is sent to the DSP as the respective PWM value.

```
public short PIcontroller(double refference, double feedBack, double Kp,
double Ki)
{
        float feedBackError = (float)(refference - feedBack);
        double P = feedBackError * Kp;

        I += feedBackError;

        double outPut = P + I*Ki;

        if (outPut > 100) { outPut = 100; }
        else if (outPut < -100) { outPut = -100; }

        return (short)outPut;
}
```

Figure B-4: C# code of proportional-integral Controller.

Values for Kp, and Ki were experimented with; Kp=1.0 and Ki=0.01 provided the best possible result for this controller implementation. The results are illustrated in Figure B-5. The SSE disappeared and the settling time was acceptable but, there was some overshoot.

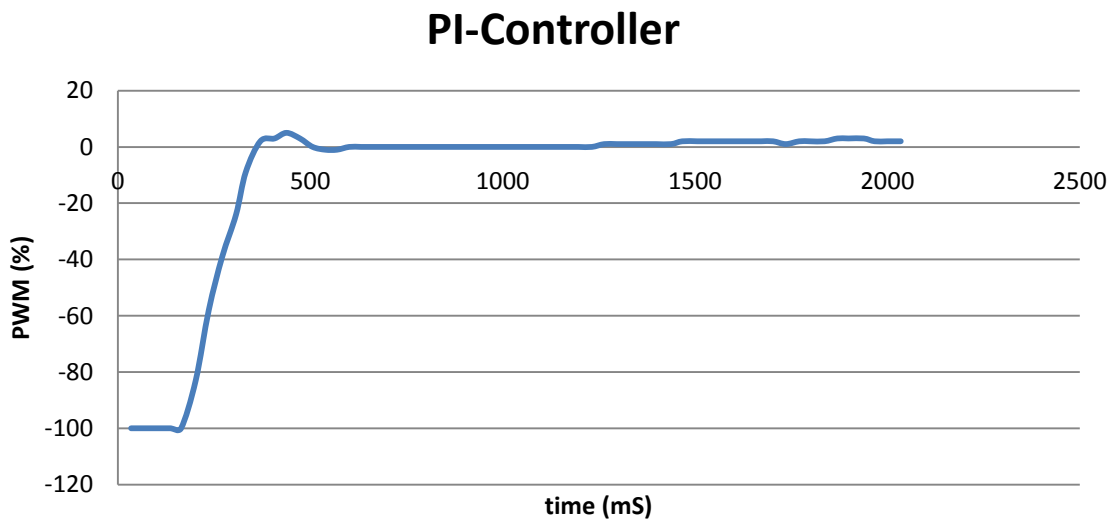


Figure B-5: Proportional-integral controller response for horizontal tracking.

B.2.3 Proportional-Integral-Derivative (PID) Controller

The PI controller results were not ideal due to the over-shoot, so a derivative component was added to the controller to have a more stable convergence. The idea is to calculate the derivative of the error over time and multiply it by the derivative gain (Kd). The derivative component is added to the previous controller equation; refer to Equation (B-6) and Equation (B-7).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)dt + K_d \frac{d}{dt} e(t) \tag{B-6}$$

$$U(s) = K_p E(s) + \frac{K_i E(s)}{s} + sK_d E(s) \tag{B-7}$$

The new PID equation is implemented in code as shown in Figure B-6.

```
public short PIDwithTime(double refference, double actual, double Kp, double Ki,
double Kd, double dt)
{

    errorPID = refference - actual;
    double P = errorPID * Kp;

    iPID += errorPID*dt;

    double D = (errorPID - preError)/dt;


    short outPut = (short)(P + iPID * Ki + D * Kd);

    if (outPut > 100) { outPut = 100; }
    else if (outPut < -100) { outPut = -100; }

    preError = errorPID;

    return outPut;
}
```

Figure B-6: C# code of PID controller.

From the most appropriate setting, Kp = 1.0, Ki = 0.01, and Kd = 0.05. The results are shown in Figure B-7.

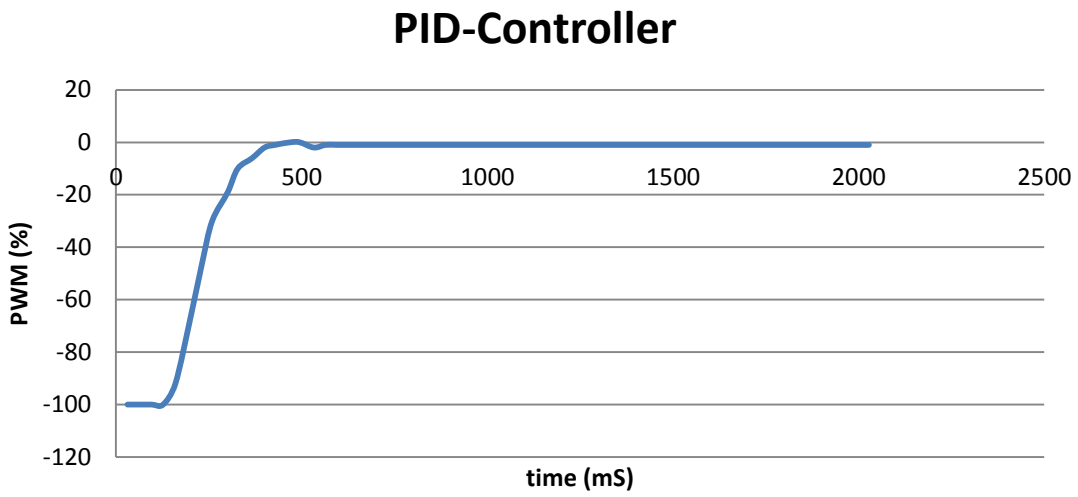


Figure B-7: Proportional-integral-derivative controller response for horizontal tracking.

Compared to the other two controllers (P and PI), PID produced the best results for this isolated system. There was no noticeable SSE, and the convergence was fast and stable. One more controller that is unlike the last three was implemented to check if the system reacts any better to that.

B.2.4 Fuzzy Logic Controller

Fuzzy logic was first proposed in 1965 by Lotfi A. Zedeh from the University of California. In his paper, he proposed the idea of paradigm, which is rules and regulations that define the boundaries for successful problem solving. The first simulated fuzzy system was implemented in Japan in 1985 to control a rail way system. Unlike the classical controllers explored above, fuzzy logic controllers analyze analog input in terms of logical variables between 0 and 1 [14]. These logic values are based on "human language" rules which are converted into mathematical equivalent. Fuzzy logic controllers are very flexible in the sense that they can work with problems that have incomplete or inaccurate data.

A fuzzy controller consists of three major processes; fuzzifier, fuzzy inference engine (FIE), and defuzzifier. Figure B-8 illustrates a block diagram of this process.

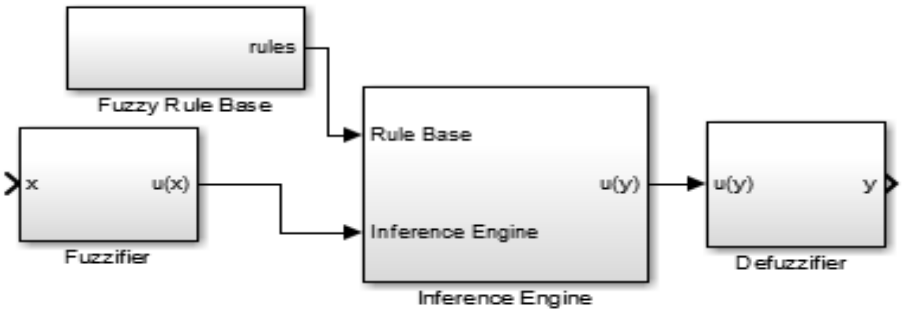


Figure B-8: Block diagram of a fuzzy control system.

Fuzzifier is responsible for taking a crisp input and converting it to a fuzzy input so that the fuzzy inference engine can process it. Fuzzifiers map a crisp point to a fuzzy set as illustrated in Figure B-9.

$$u \in x \subset R^n \text{ to a fuzzy set } U \subset x$$

Figure B-9: Fuzzifiers process.

There are different fuzzifiers that can be used to determine how the input is converted. For this controller, a singleton fuzzifier was used, which is defined in Equation (B-8).

$$U(x) = \begin{cases} 1 & at\ x^* \\ 0 & at\ \overline{x^*} \end{cases} \tag{F-8}$$

The input membership function created for this controller using a singleton fuzzifier is shown in Figure B-10.

- NB exists from -1 to -0.2.
- NS exists from -1 to 0.
- Z exists from -0.2 to 0.2.
- PS exists from 0 to 1.
- PB exists from 0.2 to 1.

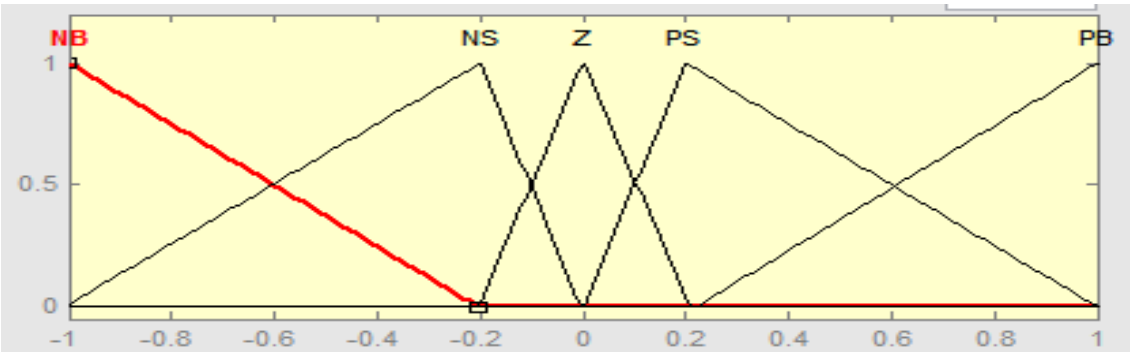


Figure B-10: Input membership function, version one.

The fuzzy input can now be sent to the fuzzy inference engine. FIE is a computing framework based on fuzzy set, and reasoning rule base [51]. Fuzzy rule base is a set of rules that are modelled as "If, then" layout. For example, if input1 is NB and input2 is PB, then output is PS. The FIE then combines all the rules using product, min, or max operations to predict the strength of each rule. This generates an output membership function accordingly as shown in Figure B-11.

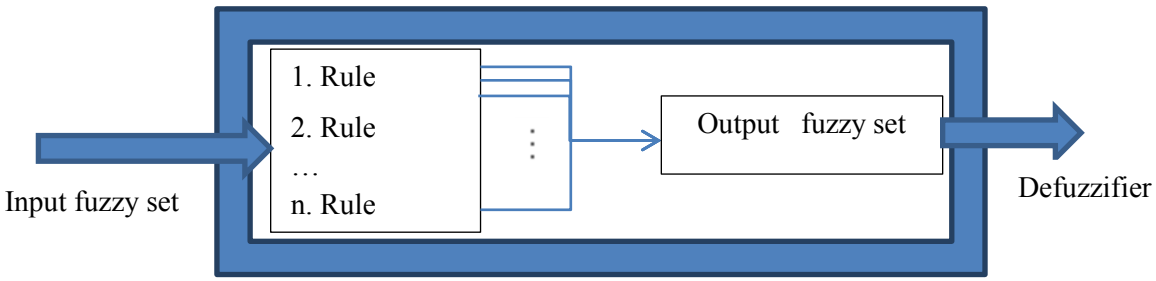


Figure B-11: Fuzzy inference engine block diagram.

For this controller, twenty five rules were applied based on two inputs, error (e) and change in error (ce), as shown in Table B-2. Based on the rules, an output membership function is created as shown in Figure B-12. Appendix B displays graph and performance of the inference engine based on the given rule base.

Table B-2: Rule base for fuzzy controller, version one.

e/ce	NB	NS	Z	PS	PB
NB	N	N	N	Z	P
NS	N	N	N	Z	P
Z	N	Z	Z	Z	P
PS	N	Z	P	P	P
PB	N	Z	P	P	P

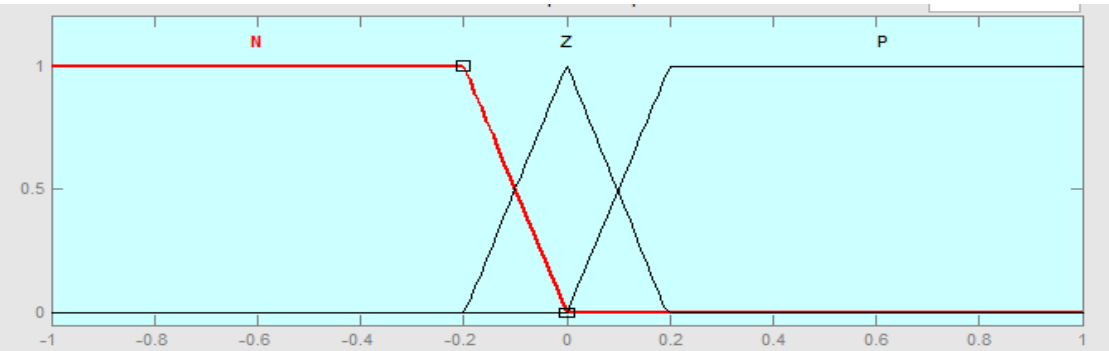


Figure B-12: Output membership function, version one.

Finally, a defuzzifier must be used to take the output fuzzy set and convert it to a crisp set. The defuzzifier used for this controller is center of average (COA) defuzzifier. It finds the center of all the areas and finds the average based on the weight of each area. It is also called center of area defuzzifier.

The entire controller can be modeled by Equation (B-9).

$$R^k = IF\ x1\ is\ A^k\ and.....and\ x_n\ is\ A_n^k\ THEN\ y\ is\ B^k,\ k = 1, \dots, n.$$

$$Y = \frac{\sum_{k=1}^{N} \overline{y^k} \left(\prod_{i=1}^{n} u_{A_i^k}(xi) \right)}{\sum_{k=1}^{N} \left(\prod_{i=1}^{n} u_{A_i^k}(xi) \right)} \qquad \text{(F-9)}$$

Where,

$$B^k = u_{A_i^k}(xi) \qquad \text{(F-10)}$$

This controller is modeled in code (source code for this controller is provided at the end of this chapter) and tested for performance. The results were acceptable. The settling time was very fast and there were no overshoots, but the target did not center perfectly in the frame. There was a small SSE about 3 to 8 pixels. Figure B-13 displays the system response of the targeting system when fuzzy controller is used.

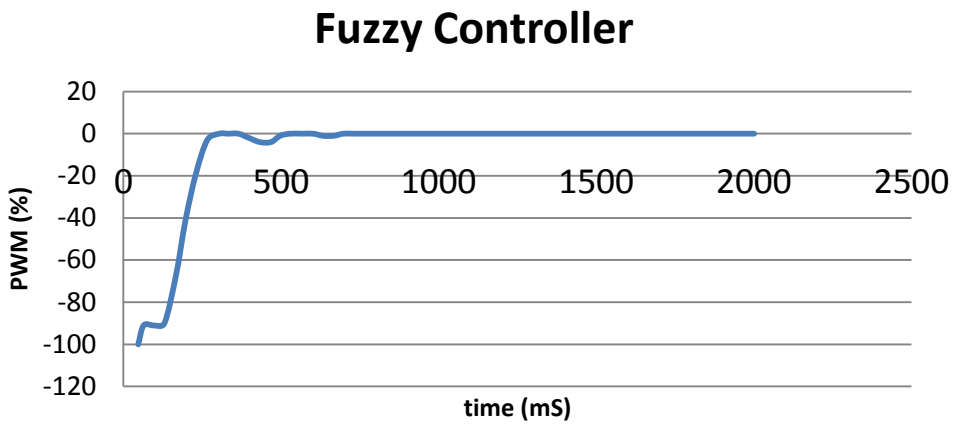


Figure B-13: Fuzzy controller response of horizontal tracking, version one.

Results from horizontal position of the target (pixels) and the PWM (%) response over time are illustrated in Figure B-14.

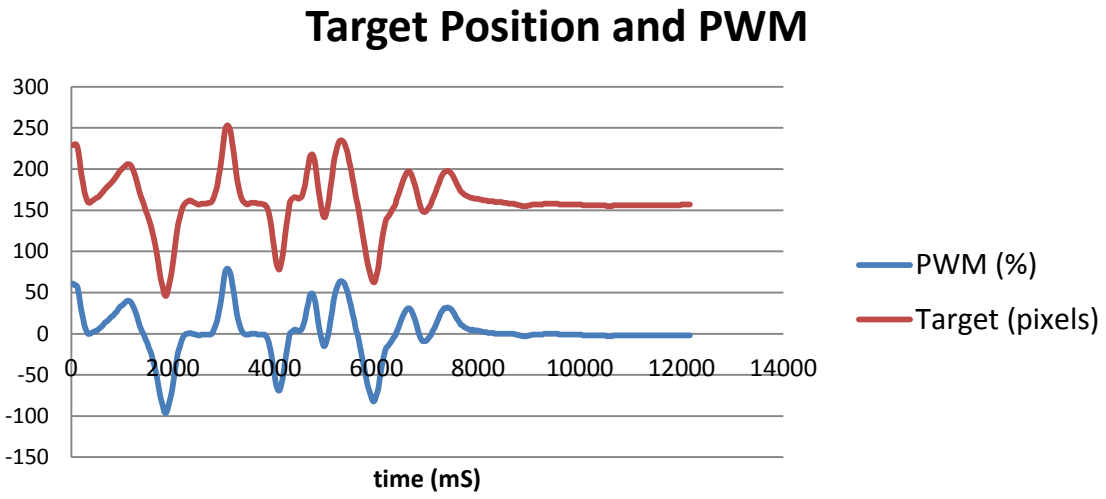


Figure B-14: Horizontal position and PWM over time using fuzzy controller, version one.

B.3 Version Two

Version two of the robot will need to control the horizontal and vertical position of the target. Based on the performance from version one, it can be concluded that either a PID or a fuzzy controller should be ideally implemented. Given the completely different physical design, new parameters should be entered for satisfactory performance of the controllers.

B.3.1 Horizontal Target Position Controller, Version Two

The PID controller was re-tuned for the new horizontal-camera controller. The new design has a higher load for the horizontal controller and due to the added length of the "camera stand height"; there were some vibrational noise at higher speeds. By experimenting with different parameters, satisfactory values were found for Kp, Ki, and Kd; Kp = 0.85, Ki = 0.085, and Kd = 0.08. The step response of the controller is illustrated in Figure B-15. There was a small SSE of 3-5 pixels on average, which is acceptable for this implementation.

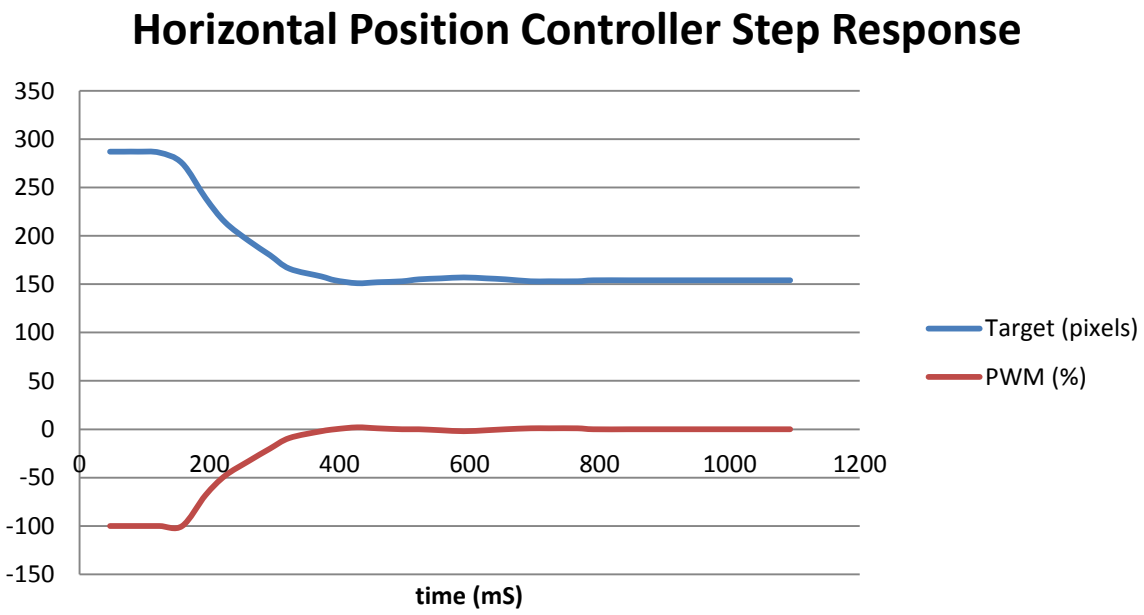


Figure B-15: Horizontal position and PWM over time using PID controller, version two.

B.3.2 Vertical Target Position Controller, Version Two

Unique to this design, there is vertical position controller. This vertical freedom was controlled by a motor mounted vertically, with a worm gear. Though this design ensured that the vertical base is fixed when there is no power, it also required higher revolution speeds to create small displacement in the vertical position. Similar to the horizontal position controller, the PID controller was retuned until satisfactory results were obtained. The results for the vertical position controller is illustrated in Figure B-16 with Kp = 1.5, Ki = 0.01, and Kd = 0.07.

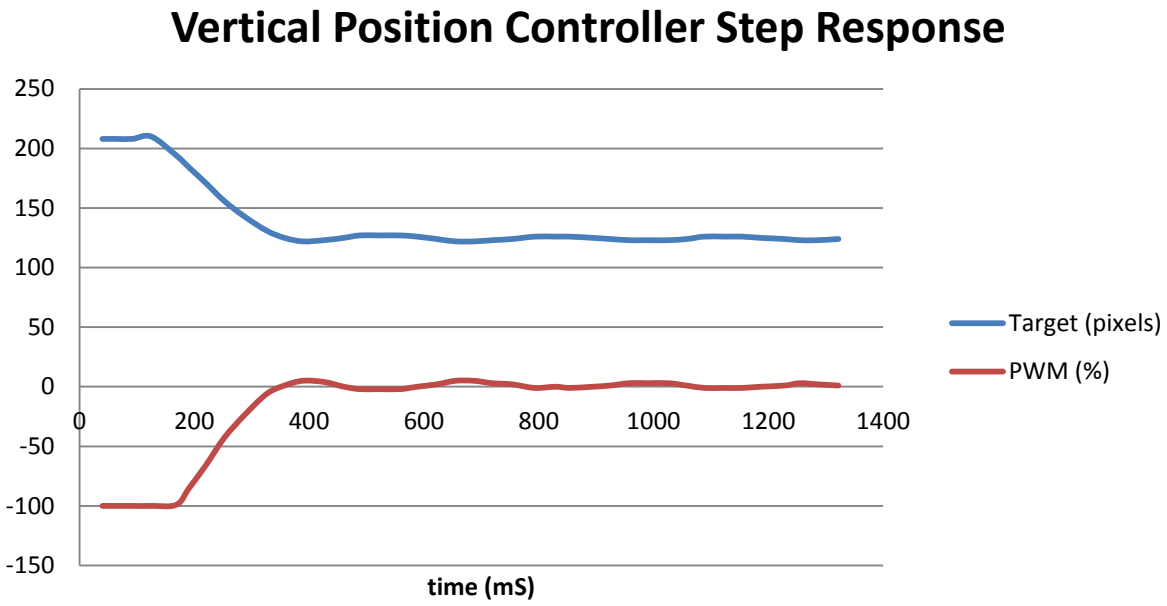


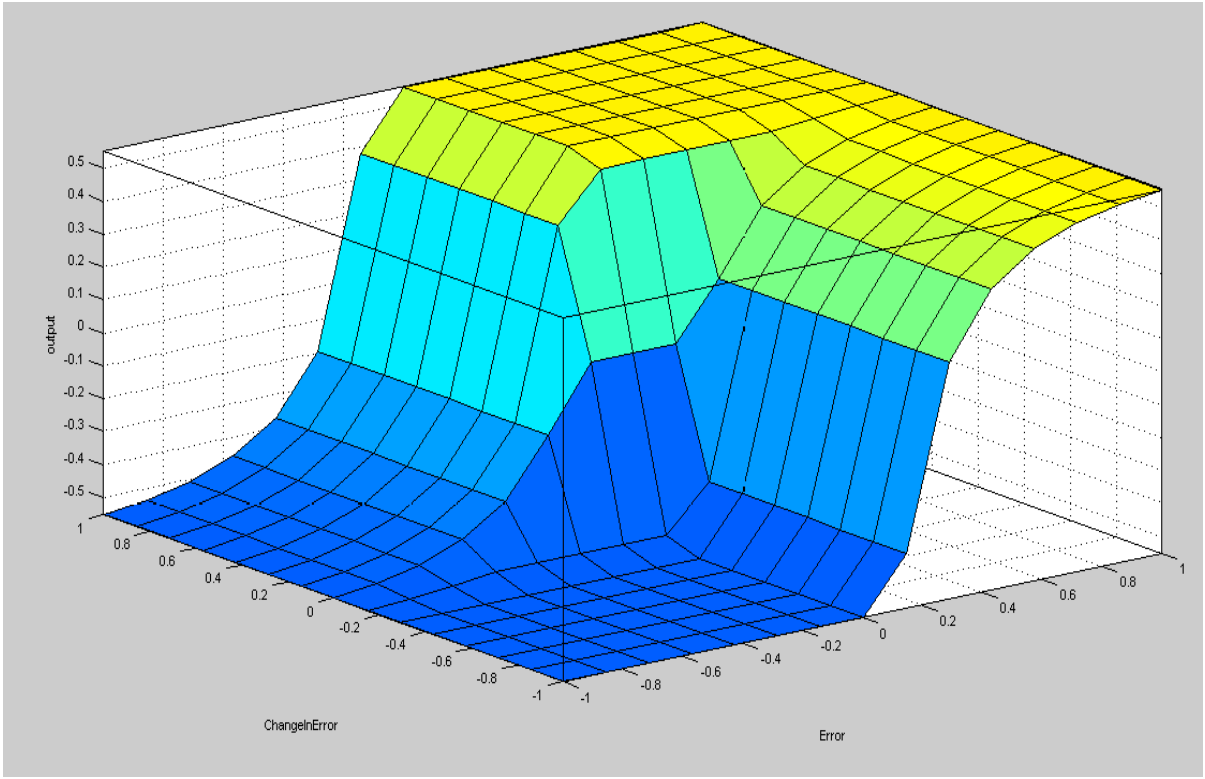
Figure B-16: Vertical position and PWM over time using PID controller, version two.

The position used as the reference was the vertical midway point; 120 pixels. There was an acceptable SSE of 1-3 pixels. With the combination of the two controllers, the overall target centering performed well at high and low speeds. At high speeds, there were small vibrational noises causing minor oscillation. At low speeds, the position controller was very stable and acceptable.

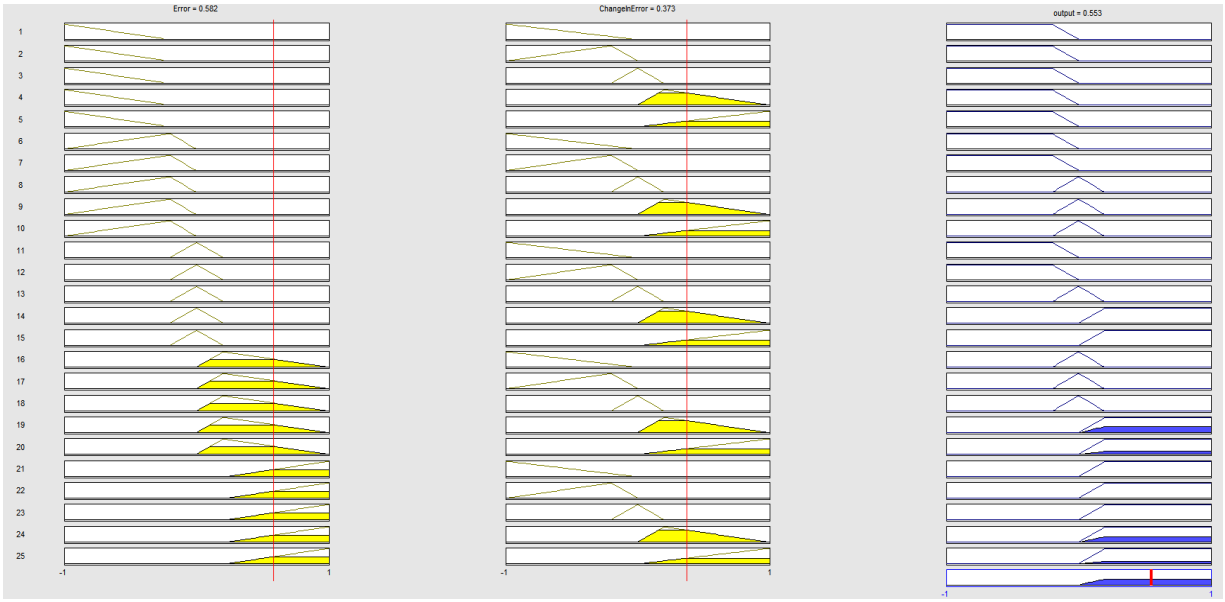
The speed controller was also a PID controller. It adjusts the maximum power delivered to the motor based on how far the robot is from an obstacle.

B.4 Additional Information (graphs and code for Fuzzy controller)

Fuzzy Rule Based Output Graph (version one):



Fuzzy Inference Engine Process (version one):



Fuzzy Controller Source Code (version one):

```
double[] inputCenters = new double[5] { -100, -4, 0, 4, 100 };
double[] outputCenters = new double[3] { -20, 0, 20 };

double[] outputLimmitNB = new double[3] { -100, 20, 0 };
double[] outputLimmitZ = new double[3] { -20, 0, 20 };
double[] outputLimmitPB = new double[3] { 0, 20, 100 };

double prevValue = 0;
public short fuzzyController(double currentValue, double reference)
{
    double outputValue = (currentValue-reference)/1.6;


    double iNB = inputNB(outputValue);
    double iNS = inputNS(outputValue);
    double iZ = inputZero(outputValue);
    double iPS = inputPS(outputValue);
    double iPB = inputPB(outputValue);

    double[] array1 = { iNB, iNS, iZ, iPS, iPB };

    double changeInValue = (outputValue - prevValue) / 2;
    prevValue = outputValue;
    double max1 = 0;
    double max2 = 0;

    for (int i = 0; i < 5; i++)
```

```
    {
        if (max1 < array1[i]) max1 = array1[i];
    }

    double ciNB = inputNB(changeInValue);
    double ciNS = inputNS(changeInValue);
    double ciZ = inputZero(changeInValue);
    double ciPS = inputPS(changeInValue);
    double ciPB = inputPB(changeInValue);

    double[] array2 = { ciNB, ciNS, ciZ, ciPS, ciPB };


    for (int i = 0; i < 5; i++)
    {
        if (max2 < array2[i]) max2 = array2[i];
    }


    double[] centers = new double[5] { -1, -0.5, 0, 0.5, 1 };
    double[] cuCenter = new double[25] { -20, -20, -20, 0, 20, -20, -20, -20, 0, 20, -
    20, 0, 0, 0, 20, -20, 0, 20, 20, 20, -20, 0, 20, 20, 20 };
    double yStarN = 0;
    double yStarD = 0;
    short yStar = 0;

    for (int i = 0; i < 5; i++)
    {
        for (int x = 0; x < 5; x++)
        {
            yStarN += cuCenter[x + 5 * i] * (array1[i] * array2[x]);
            yStarD += (array1[i] * array2[x]);
        }
    }

    yStar = (short)(-5 * (yStarN / yStarD));


    return yStar;
}

//Fuzzy membership set for 5-membership system INPUT
double inputNB(double value)
{
    if (value >= inputCenters[1]) return 0;
    else return -(value + -inputCenters[1]) / (inputCenters[1] - inputCenters[0]);
}
double inputNS(double value)
{
    if (value >= inputCenters[2]) return 0;
    else if (value < inputCenters[2] && value > inputCenters[1]) return -(value) /
(inputCenters[2] - inputCenters[1]);
    else return (value + -inputCenters[0]) / (inputCenters[1] - inputCenters[0]);
}
double inputZero(double value)
```

```
{
    if (value <= inputCenters[1] || value >= inputCenters[3]) return 0;
    else if (value < inputCenters[2] && value > inputCenters[1]) return
((inputCenters[2] - inputCenters[1]) + value) / (inputCenters[2] - inputCenters[1]);
    else return (inputCenters[3] - value) / (inputCenters[3] - inputCenters[2]);
}
double inputPS(double value)
{
    if (value <= inputCenters[2]) return 0;
    else if (value < inputCenters[3] && value > inputCenters[2]) return value /
(inputCenters[3] - inputCenters[2]);
    else return (inputCenters[4] - value) / (inputCenters[4] - inputCenters[3]);
}
double inputPB(double value)
{
    if (value <= inputCenters[3]) return 0;
    else return (value - inputCenters[3]) / (inputCenters[4] - inputCenters[3]);
}



//Fuzzy membership set for system OUTPUT
double outNB(double value)
{
    if (value >= outputLimmitNB[2]) return 0;
    else if (value >= outputLimmitNB[1] && value < outputLimmitNB[2]) return -(value /
(outputLimmitNB[2] - outputLimmitNB[1]));
    else return 1;
}

double outZero(double value)
{
    if (value <= outputLimmitZ[0] || value >= outputLimmitZ[2]) return 0;
    else if (value < outputLimmitZ[1] && value > outputLimmitZ[0]) return (value +
outputLimmitZ[1] - outputLimmitZ[0]) / (outputLimmitZ[1] - outputLimmitZ[0]);
    else return (value) / (outputLimmitZ[2] - outputLimmitZ[1]);
}

double outPB(double value)
{
    if (value <= outputLimmitPB[0]) return 0;
    else if (value <= outputLimmitPB[1] && value > outputLimmitPB[0]) return (value /
(outputLimmitPB[2] - outputLimmitPB[1]));
    else return 1;
}
```

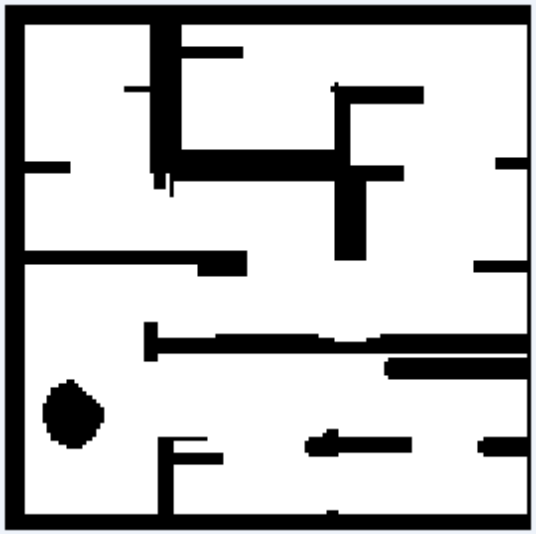
Appendix C

Navigation Maps and Code

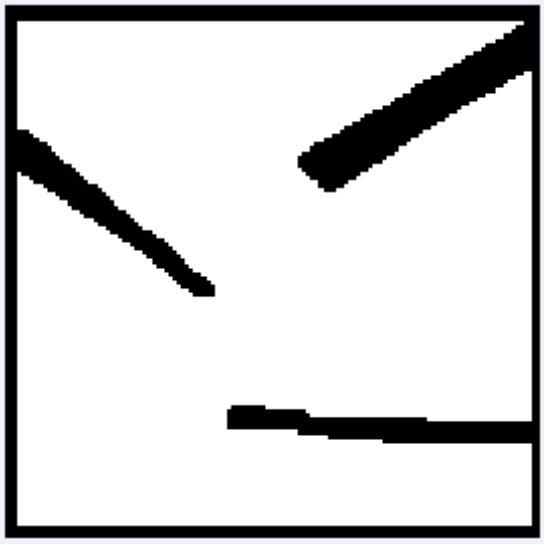
Original Maps:

Tree-road

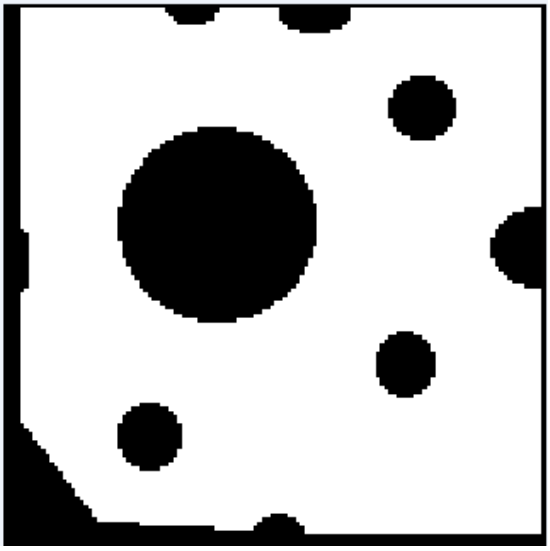
Maze



Simple-line



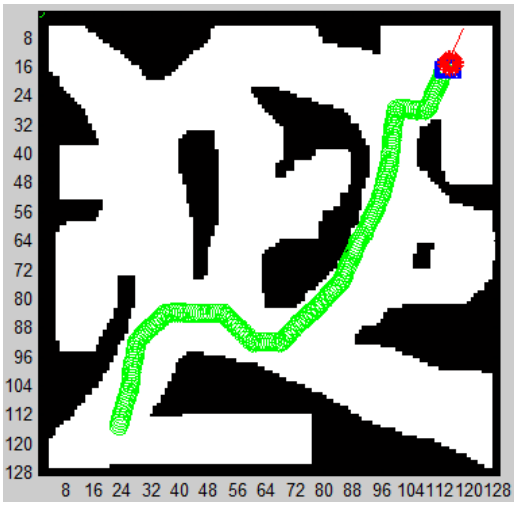
Simple-circle



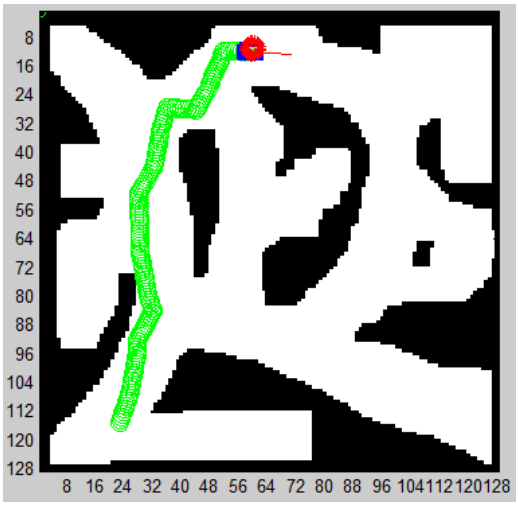
Modified A-Star Navigational paths from Table 6-1:

Tree Road:

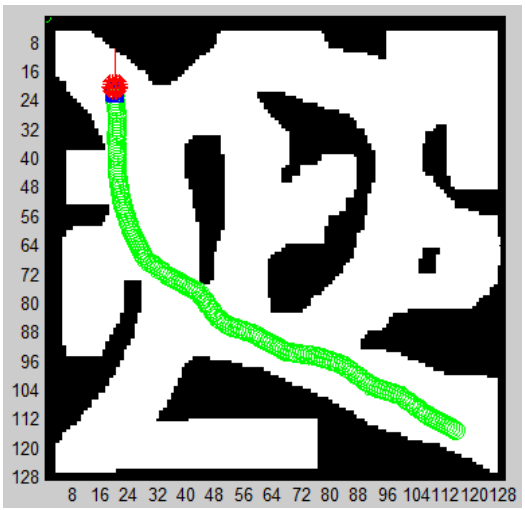
Start (23,115), Destination (115,15)



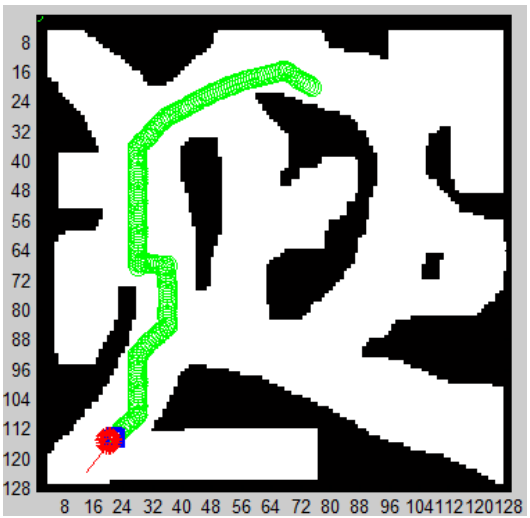
Start (23,115), Destination (60,11)



Start (115,115), Destination (20,20)

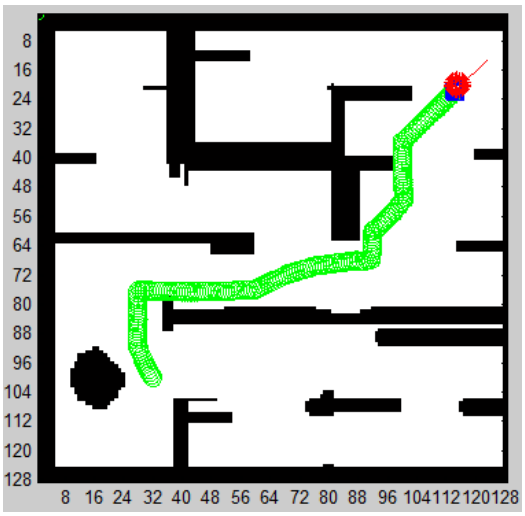


Start (75,20), Destination (20,115)

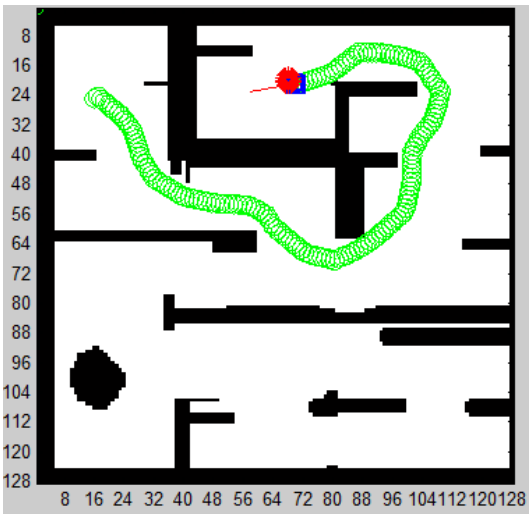


Maze:

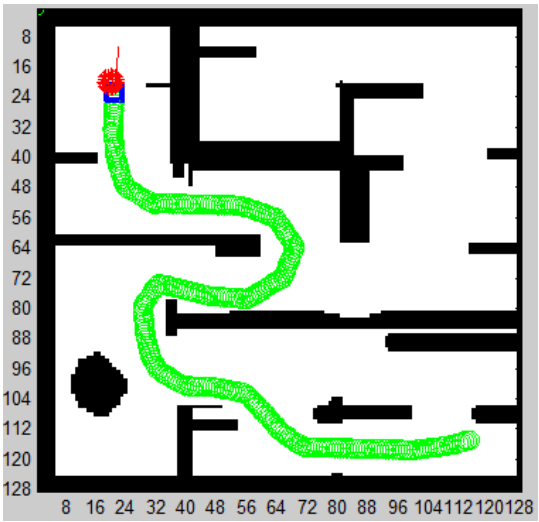
Start (32,100), Destination (115,20)



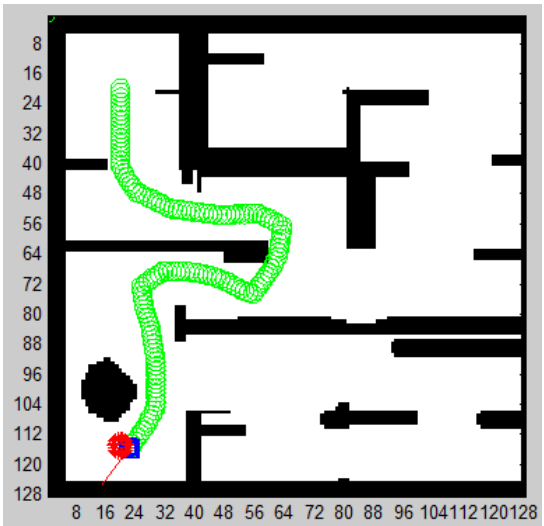
Start (23,115), Destination (60,11)



Start (115,115), Destination (20,20)

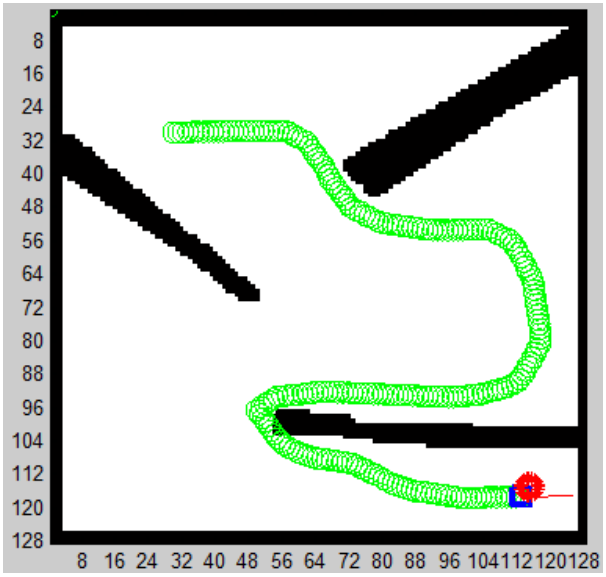


Start (20,20), Destination (20,115)

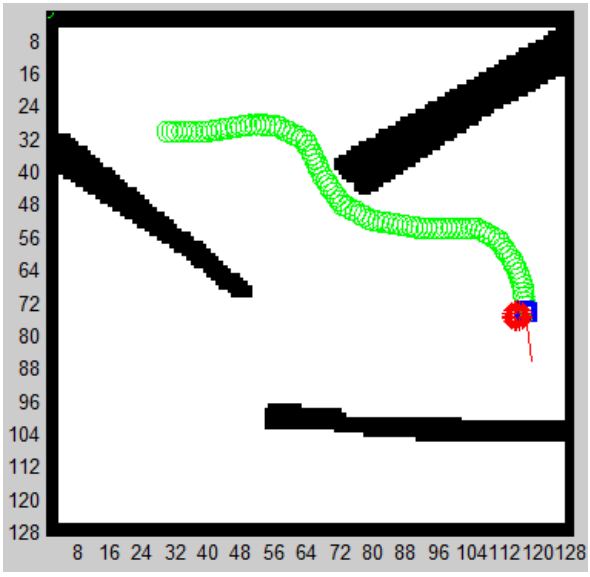


Simple Line:

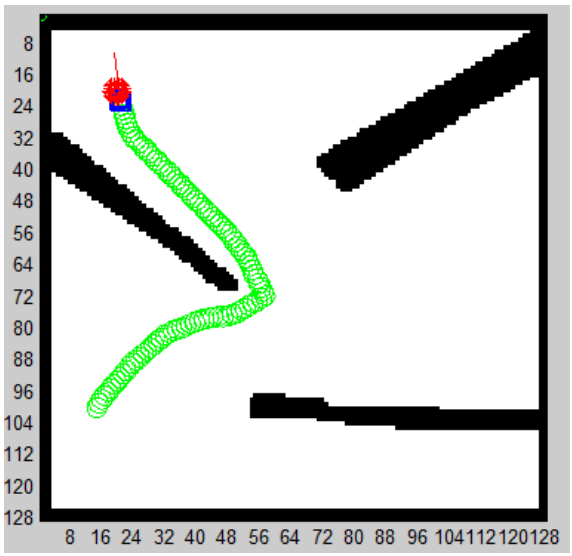
Start (30,30), Destination (115,115)



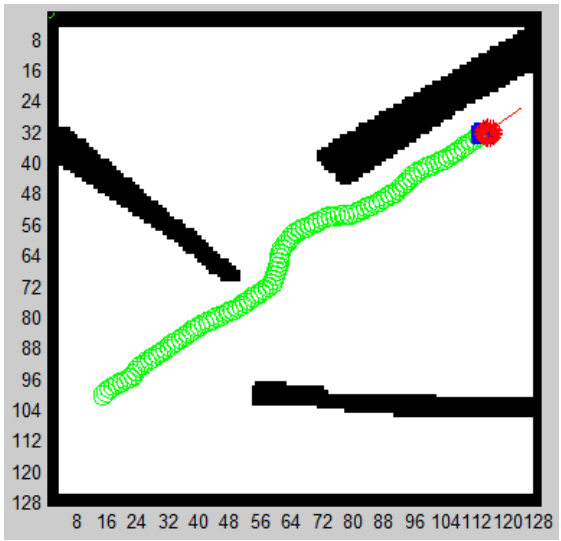
Start (30,30), Destination (115,75)



Start (15,100), Destination (20,20)

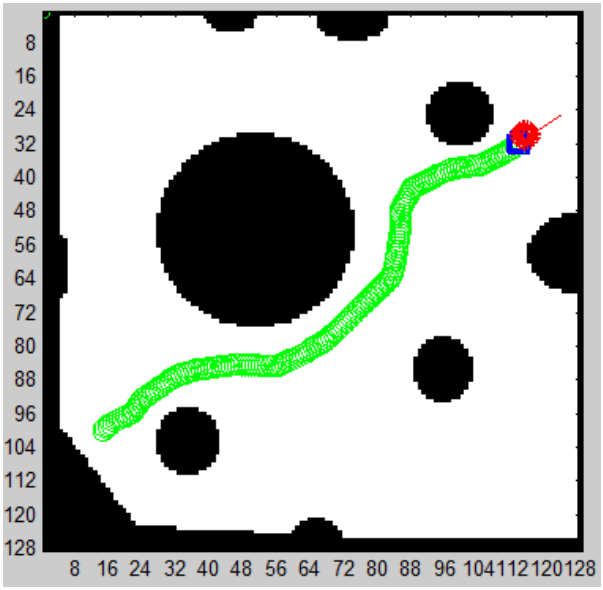


Start (15,100), Destination (115,32)

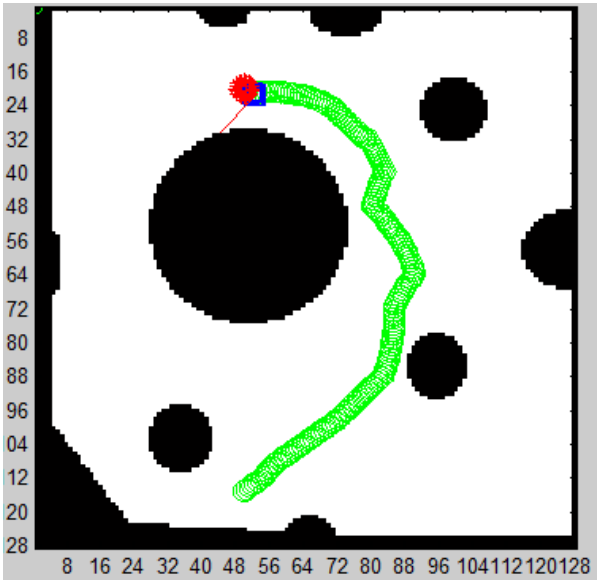


Simple Circle:

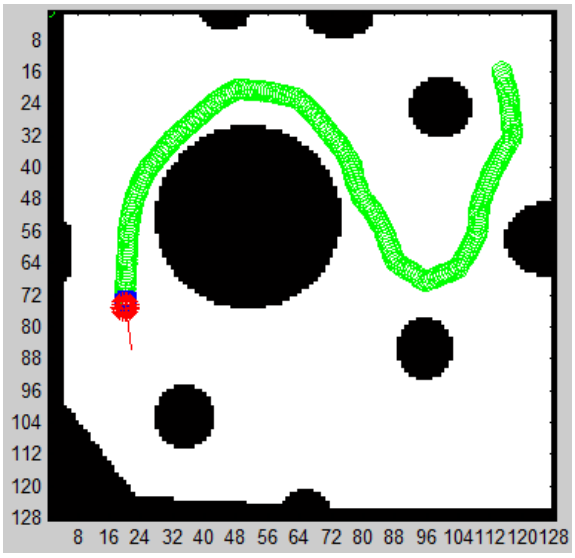
Start (15,100), Destination (115,35)



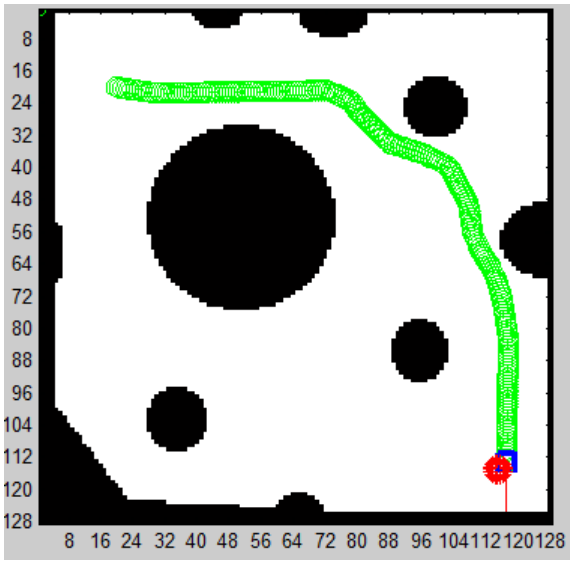
Start (50,115), Destination (50,20)



Start (115,16), Destination (20,75)



Start (20,20), Destination (115,115)



GBNVF source code:

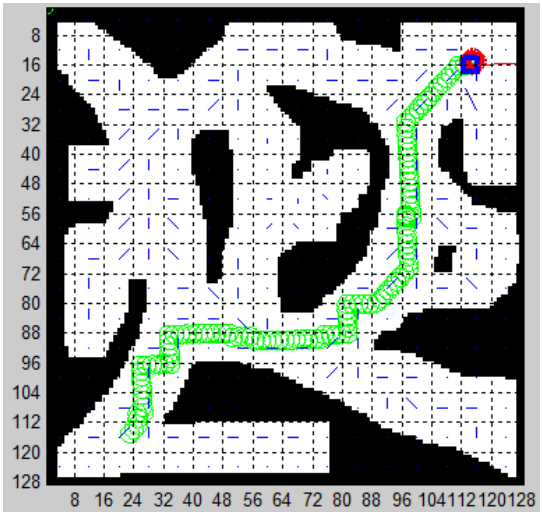
```
function path = GApotentialField(pos,des,gridDim)
global Roi;
disFromTarget = zeros(gridDim, gridDim, 'int8');
selected = zeros(gridDim, gridDim, 'int8');
dirPath = zeros(gridDim, gridDim, 'int8');

xd=des(2);
yd=des(1);

%calcluate navigational steps from destination
disFromTarget(xd,yd)=1;
disFromTarget(xd-1,yd)=2;
disFromTarget(xd+1,yd)=2;
disFromTarget(xd,yd-1)=2;
disFromTarget(xd,yd+1)=2;
disFromTarget(xd-1,yd+1)=3;
disFromTarget(xd+1,yd+1)=3;
disFromTarget(xd-1,yd-1)=3;
disFromTarget(xd+1,yd-1)=3;

selected(des(2),des(1))=1;

i=1;
j=1;
count=0;
while(1)
 if(Roi(i,j)==4)
    if(selected(i,j)~=1)
        if(disFromTarget(i,j)~=0)
          selected(i,j)=1;
          for y = 1:3
            for x = 1:3
                if(disFromTarget(i+x-2,j+y-2)==0)
                    if(y==2)&&(x~=2)
                        disFromTarget(i+x-2,j+y-2)=disFromTarget(i,j)+1;
                    elseif(y~=2)&&(x==2)
                        disFromTarget(i+x-2,j+y-2)=disFromTarget(i,j)+1;
                    end
                end
            end;
          end

        end
    end
 else
     selected(i,j)=1;
     disFromTarget(i,j)=100;
 end

   if(i==gridDim)
       if(j==gridDim)
           j=1;
```

```
        end
        i=1;
        j=j+1;
    else
        i=i+1;
    end
    count=count+1;
    if(count==(6000))
       break;
    end
end
%end calc. navigation

%generate vector field
global vecField;
global rPath;
vecField = zeros(gridDim, gridDim,2,'double');
for j=1:gridDim
    for k=1:gridDim
        if(disFromTarget(k,j)~=100)
            xL=double(disFromTarget(k-1,j));
            if(xL==100.0)xL=double(disFromTarget(k,j)+1.1);end
            xR=disFromTarget(k+1,j);
            if(xR==100.0)xR=double(disFromTarget(k,j)+1.1);end
            hor=double(xL-xR);

            xU=disFromTarget(k,j-1);
            if(xU==100.0)xU=double(disFromTarget(k,j)+1.0);end
            xD=disFromTarget(k,j+1);
            if(xD==100.0)xD=double(disFromTarget(k,j)+1.0);end
            ver=double(xU-xD);

            vecField(k,j,1) = (atan2(ver,hor));
            vecField(k,j,2) = hypot(hor,ver);

        end
    end
end
%end vector field.

rPath=disFromTarget;
return ;
```

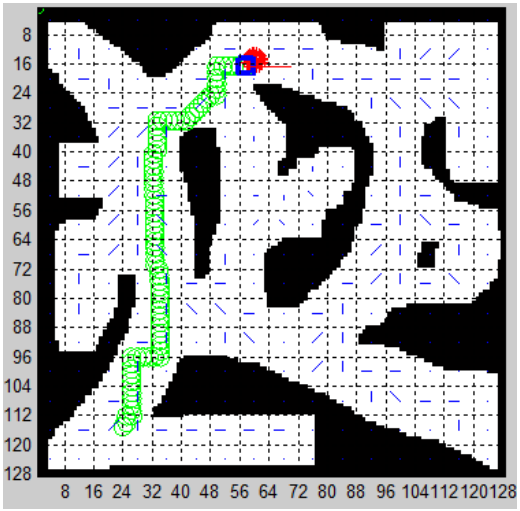
GBNVF Navigational paths:

Tree Road:

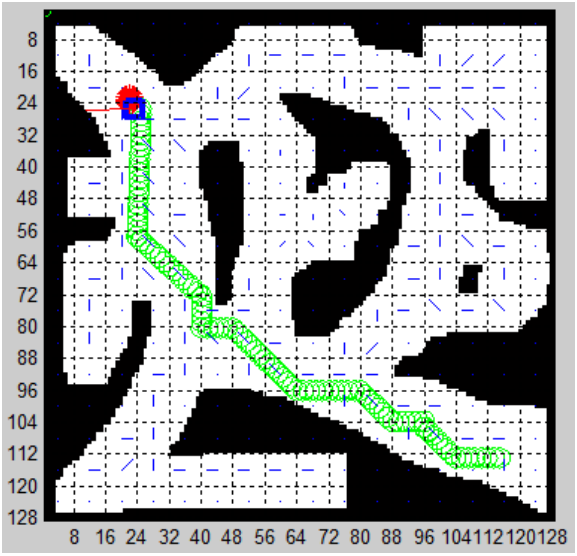
Start (23,115), Destination (115,15)



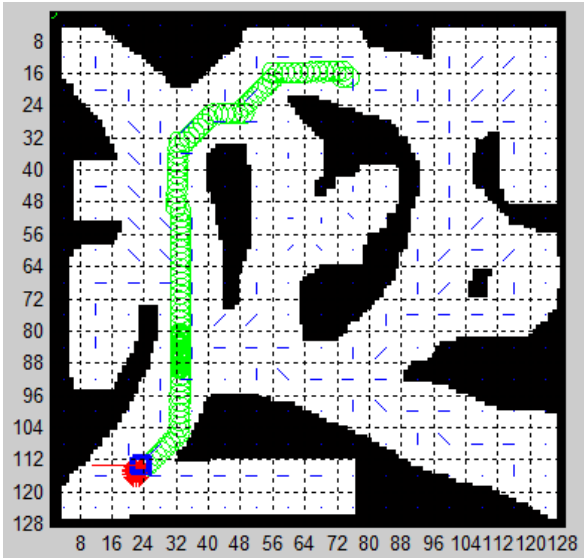
Start (23,115), Destination (60,11)



Start (115,115), Destination (20,20)

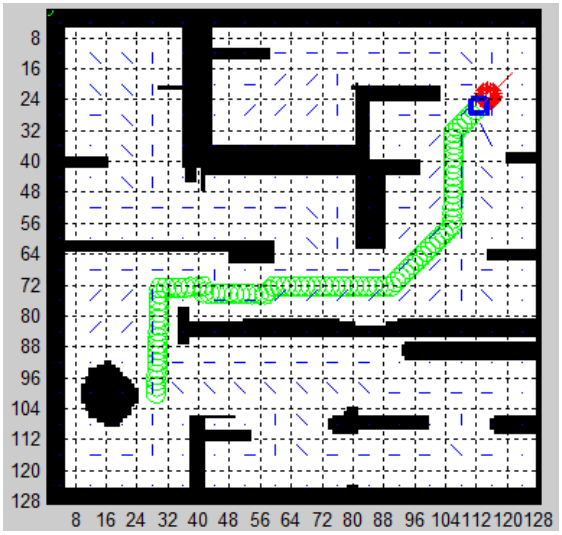


Start (75,20), Destination (20,115)

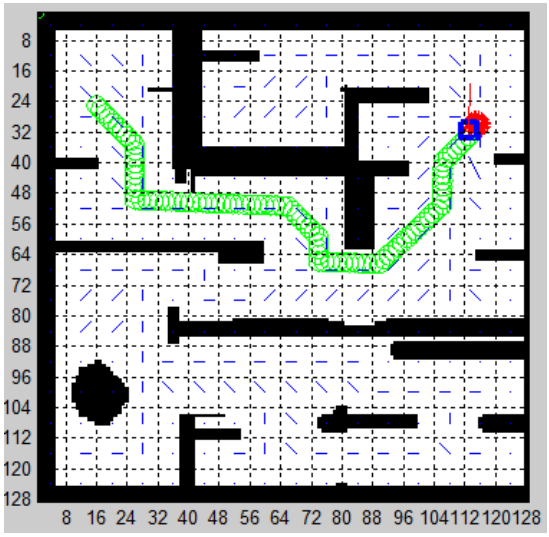


Maze:

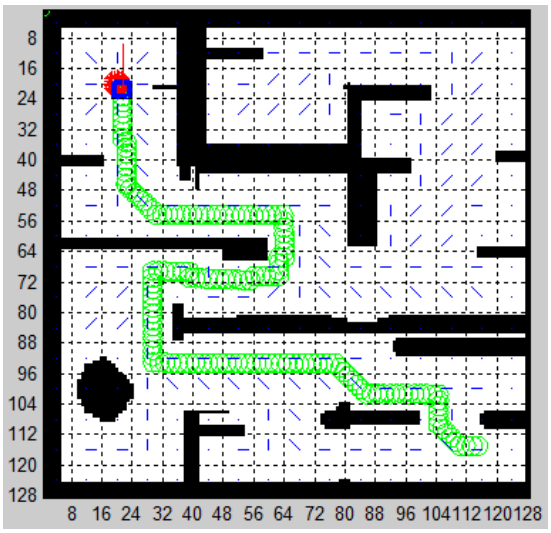
Start (32,100), Destination (115,20)



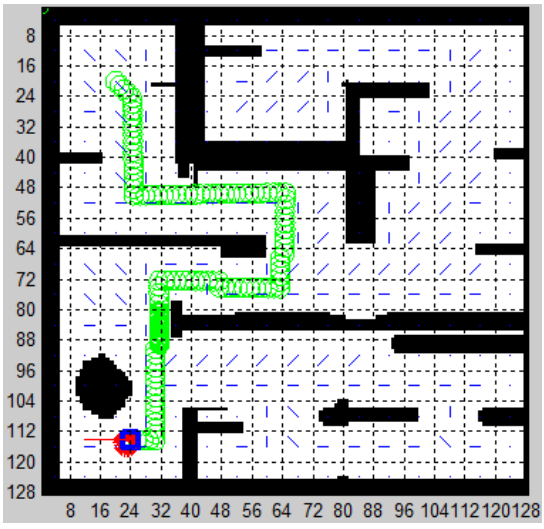
Start (23,115), Destination (60,11)



Start (115,115), Destination (20,20)

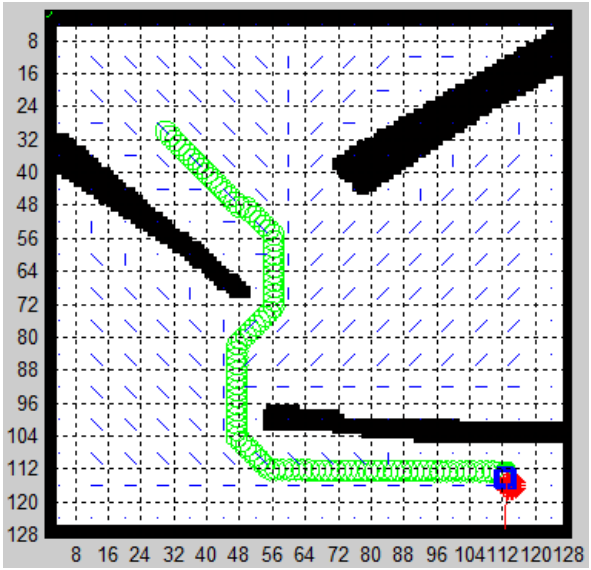


Start (20,20), Destination (20,115)

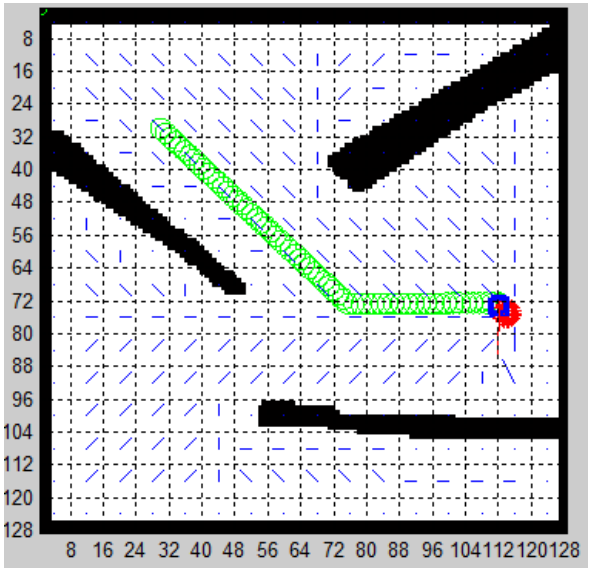


Simple Line:

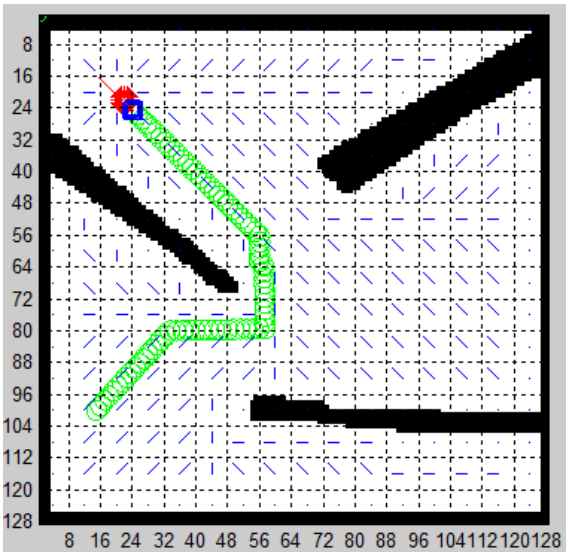
Start (30,30), Destination (115,115)



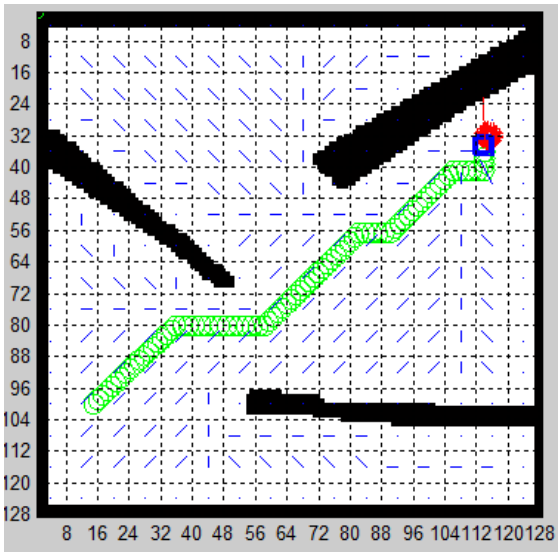
Start (30,30), Destination (115,75)



Start (15,100), Destination (20,20)

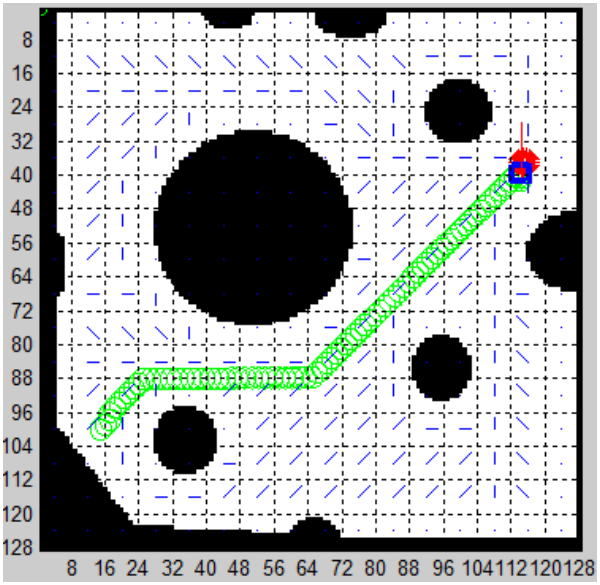


Start (15,100), Destination (115,32)

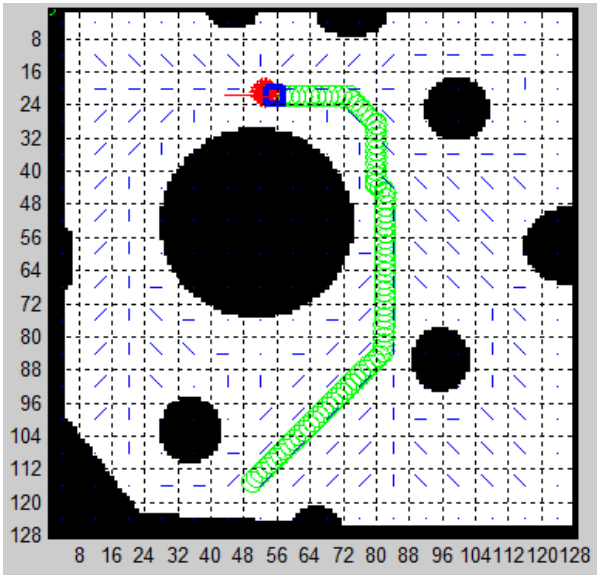


Simple Circle:

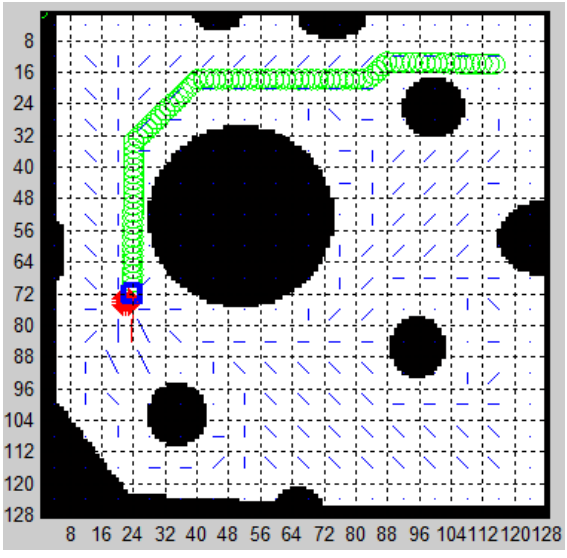
Start (15,100), Destination (115,35)



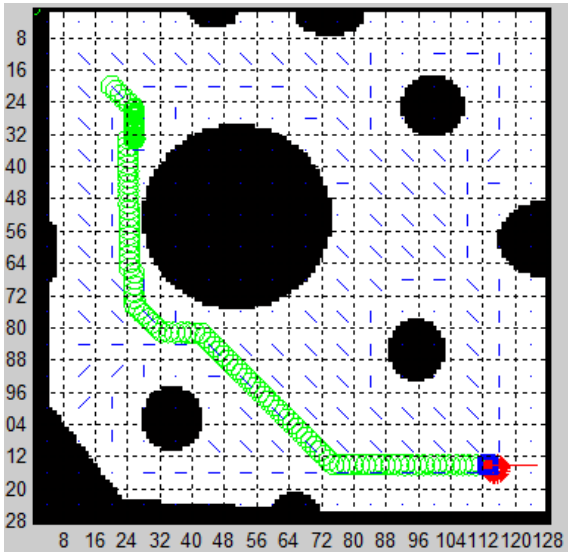
Start (50,115), Destination (50,20)



Start (115,16), Destination (20,75)



Start (20,20), Destination (115,115)



Appendix Chapter D
Optical Sensor

D.1 Introduction

The optical sensor is the major sensor involved in this project. It is intended that the robot visually observes the world around it and understands it similar to how humans perceive their respective surroundings. Humans perceive combinations of the three primary colours; red, yellow, and blue. The ability of color vision is possible from the cells in the eyes called cones. There are three types of cones and each one activated by different wavelengths of light. When a spectrum of visual light hits one's eyes, the three cones relay a message to the brain which then combines the signals into one perception. This perception is the sense humans refer to as color vision. Each cone is able to detect approximately one hundred shades of the color, thus enabling humans to see one million colours. Some scientists argue that humans can see more shades but most agree that the visual wavelength is from 700 nanometers (nm) to 400 nm.

A simple RGB camera will output an array of three values per pixel; the intensity of red, green, and blue in every pixel. This will solve the problem of perceiving colours of the world like any humans. However, humans observe the world in three dimensions and depth perception is absolutely important when navigating. This chapter explores the possibilities in choosing an appropriate sensor that is capable of capturing colour and relative depth information from the surrounding. Topics explored in the chapter consist of stereo-vision and infrared cameras.

D.2 Stereo Vision

Stereo vision is the process of recovering depth from camera images by comparing two or more views of the same scene. Binocular stereo is most appropriate for this case. It uses two images of the same scene that are separated horizontally. It then calculates the displacement of each pixel with respect to the displacement of the two cameras, which will output a depth value for that pixel. Although this is a good method to obtaining 3-D point cloud data and it is relatively cheap to implement, it will have to be calibrated almost every

time there is a reset and environment change. There is also processing overhead from the two frames of the same scene. Another solution is using infrared (IR) enabled cameras.

D.3 Infrared Enabled Cameras

Most Infrared enabled cameras (IRC) are able to detect infrared portion of the wavelength. They are able to convert infrared energy into electronic signal which can then be processed and viewed. Though infrared cameras are generally used to identify the level of heat present in an area, in the recent past, IRCs have been modified and used as a device that is capable of measuring distance/displacement. Many are able to detect depth for every pixel with very little processing and calibration needed. There are three IRC that seemed suitable and most applicable for the task at hand. Prime Sense Sensor (PSS), Intel's Creative Camera (ICC), and Microsoft's Kinect Sensor, each of which has user friendly software support and developmental kits intended for entertainment and research purposes. These three sensors are compared in the Table D-1 with system specifications and available Natural User Interface (NUI) libraries [44,8].

Table D-1: Optical infrared sensor comparison

DEVICES	PRIME SENSE SENSOR	INTEL'S CREATIVE CAMERA	KINECT SENSOR
Frame rate	60 frames per second (FPS) maximum.	30 frames per second (FPS) maximum.	30 frames per second (FPS) maximum.
Colour camera Resolution	640x480 pixels at 30 FPS.	1280x720 pixels at 30 FPS.	640x480 pixels at 30 FPS or 1280x960 pixels at 12 FPS.
Depth Camera Resolution	640x480 pixels at 30 FPS.	320x240 pixels at 30 FPS.	320x240 pixels at 30 FPS.
Depth range	0.8m – 3.5m	0.15m – 0.99m	0.8m - 4m
Horizontal angle range	57.5 degrees (°)	73 degrees (°) diagonal field view	57 degrees (°)
Vertical angle range	45 degrees (°)	-	43 degrees (°)
Supported Operating systems	Windows, Linux.	Windows.	Windows
Available major SDK libraries	OpenNI	OpenNI	Microsoft Kinect SDK, and OpenNI.
Other Comments	Two microphones. Low power consumption.	Two microphones. Smaller in size comparably.	Four 24-bit audio microphones. Accelerometer. Vertical tilt motor.

Each device has its own advantages and disadvantages, but with the comparison of the three, it is evident that PSS and Kinect Sensor will be more applicable for a mobile robot. PSS is impressive with its more accurate 640x480 resolution depth image at 30 FPS and low power consumption. Having accurate 3D sensory as well as low energy consumption are very important factors for a mobile robot. Kinect has a built in accelerometer as well as a vertical tilt motor. It also has 4 effectively placed microphones which can be used for better sound triangulation. At this stage, the choice would be the prime sense sensor. However, the biggest difference in hardware comes from the software support available. Microsoft continuously

had great driver support and a growing library dedicated for the Kinect Software Development Kit (SDK). Do to all the support provided by Microsoft, the Kinect SDK is easier to implement when compared the PSS. Therefore the device used for this project was a Microsoft Kinect sensor.

D.4 Microsoft Kinect Sensor

D.4.1 Introduction to Kinect

The Microsoft Kinect was first available for consumers in November 2010 as an accessory to Xbox360, a video game console. In an E3 conference, 2010, Microsoft announced and advertised the Kinect as hands-free controller that will revolutionize the gaming industry. Soon after its release, an online community reverse engineered the USB data stream and released open-source software enabling Kinect to be used on any computer[8]. In June 2011, Microsoft released a software development kit for the Kinect, which then became a tool that anyone can use. Over the years to follow, the Kinect SDK became well supported and user friendly; giving births to many creations that Microsoft never imagined. The Kinect has multiple sensors embedded with in it; IR emitter, colour sensor, IR detector, four microphones, tilt motor, and a 32-bit ARM Microprocessor. Figure D-1 illustrates the placement of these sensors, image from Microsoft website.

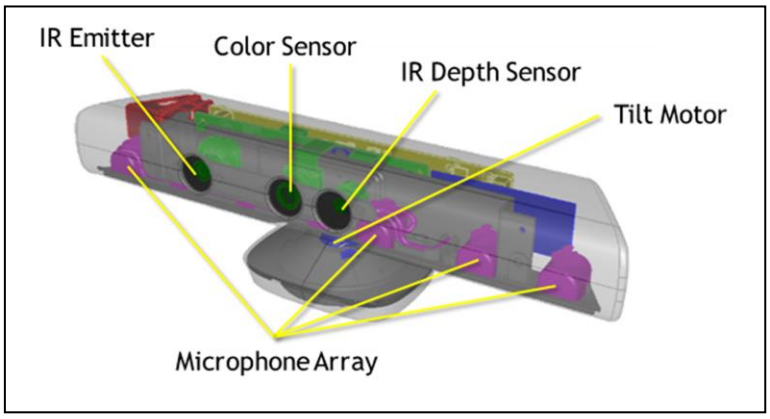
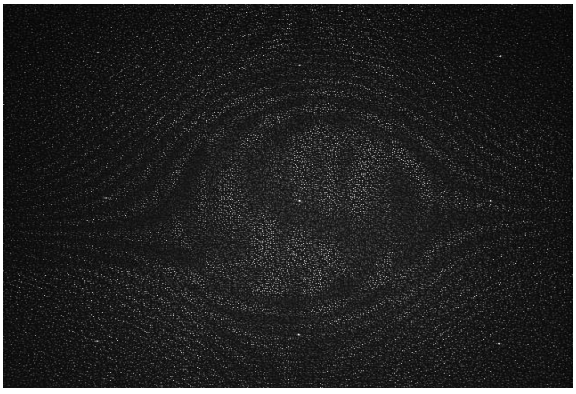


Figure D-1: Sensors within the Kinect.

D.4.2 Retrieving Depth Information from the Sensors

To achieve depth, combinations of sensors work in synchronous. The IR emitter radiates a 2D plane of IR pattern. The CMOS camera attached is fitted with an IR-pass filter to simultaneously capture that IR image. The pattern is shown in Figure D-2(a) and D-2(b) [44]. The embedded ARM-processor then calculates the distance by comparing the pattern with a pre-stored, reference pattern at a known distance. If a projected IR speckle's distance that is smaller or larger than the reference plane, the position of the IR speckle will be shifted in the direction of the perspective center in between the emitter and the IR-pass camera [8]. These shifts are measured for all speckles by image correlation procedure. For each pixel, the disparity corresponding will retrieve the displacement as shown in Figure D-3 [11].



(a)

(b)

Figure D-2: (a) Shows the speckle IR patterns captured by the camera inside. (b) Shows a close-up of the IR pattern emitted by the sensor.

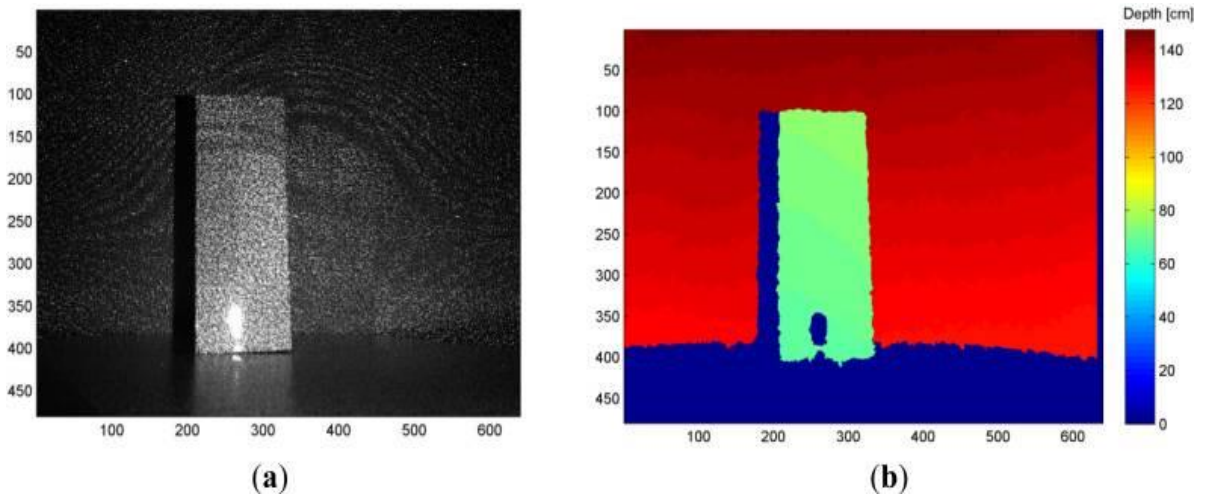


Figure D-3: (a) IR pattern is projected a scene with an object where the pattern on the object is slightly shifted. (b) Output disparity image used to calculate displacement.

The displacement outputted by the sensor is not the actual distance away from the sensor; it is the estimated displacement from the sensor plane as illustrated in Figure D-4 [18]. This allows the 3D point cloud to not be perspective-centric; instead, the data is organized in a 3D plane. This also ensures that the objects in the captured 3D space in not obscured due to perspective like a panoramic image.

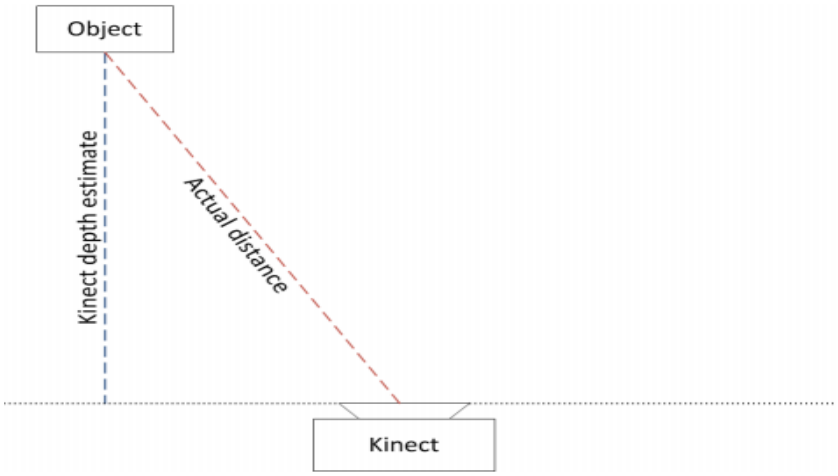
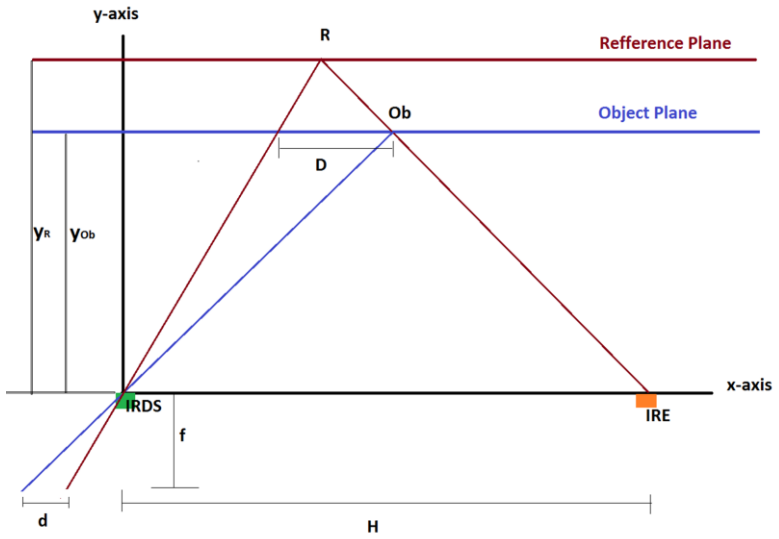


Figure D-4: The interpretation of how the Kinect sensor perceives its displacement from an object. The depth value obtained is that of the displacement from the sensor plane and not the actual distance from the sensor.

D.4.3 Mathematical Model of Estimating Depth

The mathematical model can be illustrated and derived from Figure D-5. All the variables used in the derivations are also labeled in Figure D-5. The arbitrary object on the object plane is labeled **Ob** and the measured disparity by the sensor is labeled **d**. For simplicity of explanation of the concept, this illustration is presented in 2-dimentions, ignoring the height dimension or z-axis. The x-axis is parallel to the sensor plane and the y-axis orthogonal to sensor plane.



Ob → Point on Object on the object plane.
R → Point on reference plane corresponding to the object.
IRDS → Infrared depth sensor.
IRE → Infrared emitter.
H → Distance between the IR emitter and the IR detector.
d → Observed disparity in image space.
D → Displacement of Ob with respect to R.
y_{Ob} → Distance to Ob on object plane.
y_R → Distance to R on reference plane.
f → Focal length of the IR camera.

Figure D-5: A model created to help illustrate how Kinect calculates depth for an arbitrary object point.

Using triangulation, the following two Equations; Equation (D-1) and Equation (D-2) can be obtained from Figure D-5.

$$\frac{D}{H} = \frac{y_r - y_{Ob}}{y_r} \tag{D-1}$$

$$\frac{d}{f} = \frac{D}{y_{Ob}} \tag{D-2}$$

Substituting Equation (D-1) into (D-2) and solving for y_{Ob} will yield the Equation (D-3).

$$y_{Ob} = \frac{y_R}{1+\frac{y_R d}{fH}} \quad \text{(D-3)}$$

y_{Ob} is the estimated distance of the object plane from the camera plane. This can be done simply by the embedded processor for every pixel and output a 3D point cloud to the USB stream. Now, let's take a look at how this information is received and processed on the main processor side.

2.4.4 Processing the Information from the Kinect

Kinect can be set-up to be synced with the computer by C/C++ or C# coding languages when using Microsoft Kinect SDK. This project was coded in C# and the compiler used was Microsoft Visual Studio.

When windows loaded event is activated (program is started), Kinect must be started and the different functionalities that is to be used must be enabled as shown in Figure D-6. Note that the "try, catch" is used to close the program if a connection between the computer and the Kinect is not achieved. Also, the "AllFramesReady" event handler is used at the end to go to function "kinect_ AllFramesReady" when frames are available from all the active streams of the Kinect. This will loop until the program is closed.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{

    try
    {
        kinectSensor = KinectSensor.KinectSensors[0];
        kinectSensor.Start();
    }
    catch (System.IO.IOException)
    {
        return;
    }

kinectSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480
Fps30);

kinectSensor.DepthStream.Enable(DepthImageFormat.Resolution320x240Fps
30);

kinectSensor.AllFramesReady += new
EventHandler<AllFramesReadyEventArgs>(kinect_AllFrameReady);

}
```

Figure D-6: Initial code to activate and sync the Kinect sensor and its active streams.

"kinect_AllFramesReady" function is in charge of capturing the colour frame and the depth frame; refer to Figure D-7 and Figure D-8 for the actual code. Note that "using" statement is used for colour frame and depth frame to dispose of the object once finished. Also, both sub-functions check to see if the there is a colour frame or a depth frame to be processed.

To obtain the colour frame, first an empty array of bytes are created and filled with pixels. A stride function is used to scale accordingly since each pixel has 4 bytes; blue, green, red, and transparency. Using the information such as frame width, height, pixel format, stride, and actual array of pixels, a "bitmap-source" can be created. It is then converted to simple bitmap, since bitmaps are much easier to use for image processing and is compatible with "EMGUCV" (image processing library). The image is stored as a BGR, byte, Image data.

Note that “Image” (Image<Bgr, byte>) is a class in EMGUCV library, which is used for image processing. It is a wrapper of OpenCV library to be used in C# and other .NET compatible languages.

```
using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
{
    if (colorFrame == null)
    {
        return;
    }

    byte[] cPixels = new byte[colorFrame.PixelDataLength];
    colorFrame.CopyPixelDataTo(cPixels);

    int stride = colorFrame.Width * 4

    image1.Source = BitmapSource.Create(colorFrame.Width, colorFrame.Height, 96,
    96, PixelFormats.Bgr32, null, cPixels, stride


    Image<Bgr, byte> colorImage = new Image<Bgr,
    byte>(BitmapFromSource(BitmapSource.Create(colorFrame.Width, colorFrame.Height,
    96, 96, PixelFormats.Bgr32, null, cPixels, stride)));

}
```

Figure D-7: Obtaining and storing the image frame as a usable colour bitmap.

To obtain the depth frame and creating a meaningful depth image, a little bit more processing needs to be done. First the empty byte array is filled with raw depth data received from the Kinect sensor and a colour byte is generated. To generate the colour bytes corresponding to distance, a colour index is created. The depth and colour index is looped through one at a time to obtain the depth for every pixel and colour code it to a meaningful value. Figure D-8 shows an example where pixels are coloured red or green depending on their range. Similar to color image, a bitmap is created of these pixels and stored as BGR, byte, Image data. Now both, the colour and depth data from the Kinect is available for further processing. Note that Figure D-8 is just a sample code of how to obtain and manipulate the depth pixel data.

```
using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
{
    if (depthFrame == null)
    {
        return;
    }
    depthFrame.CopyPixelDataTo(rawDepthData);
    byte[] dPixels = GenerateColoredBytes(rawDepthData);

    int stride = 320 * 4;
    Image<Bgr, byte> depthImage = new Image<Bgr,
        byte>(BitmapFromSource(BitmapSource.Create(320, 240, 96, 96,
        PixelFormats.Bgr32, null, dPixels, stride)));
        depthBox.Image = depthImage;
     }
 }

 private byte[] GenerateColoredBytes(short[] rawDepthData)
 {
     Byte[] pixels = new byte[240 * 320 * 4];

     const int blueindex = 0;
     const int greenindex = 1;
     const int redindex = 2;
     for (int depthIndex = 0, colorIndex = 0;
         depthIndex < rawDepthData.Length && colorIndex < pixels.Length;
         depthIndex++, colorIndex += 4)
        {

         int depth = rawDepthData[depthIndex] >>
         DepthImageFrame.PlayerIndexBitmaskWidth;


         if (depth > minDepthDistance && depth < maxDepthDistance)
         {
             pixels[colorIndex + blueindex] = 0;
             pixels[colorIndex + greenindex] = 255;
             pixels[colorIndex + redindex] = 0;
         }

         else if (depth > maxDepthDistance)
         {
             pixels[colorIndex + blueindex] = 0;
             pixels[colorIndex + greenindex] = 0;
             pixels[colorIndex + redindex] = 255;
         }}
     return pixels;
 }
```

Figure D-8: Obtaining and storing the depth frame as a usable colour bitmap.

D.5 Comments

In this chapter, different solutions were explored. Stereo and non-stereo solutions were analyzed to find the best suitable solution for this project. It was evident that an infrared optical sensor may be better suited for this project. Then some suitable infrared devices were compared. Of the three analyzed sensors, both the Microsoft Kinect and the prime sensor seemed to be the obvious choice, since Intel's Creative Camera can only process depth for up to 0.99 meters.

The Microsoft Kinect was the best possible solution available since it had great driver and SDK support. The Microsoft Kinect was then examined through hardware and sensor placement. From the placements of the embedded sensors, a theoretical understanding was created on how the Kinect retrieves depth information. This theory was further reinforced with a mathematical model and was illustrated by a diagram with an arbitrary object. The attention was shifted towards the main processing end. Code examples were given to show a step by step process of how to activate Kinect using C# as well as to illustrate a more detailed understanding of how the color image and the depth image is received and processed into useable information to be further processed in the future.

Appendix Chapter E
History of Robot Design and Assembly

E.1 Introduction

This chapter consists of information as to how the robot was constructed; hardware analysis, sensor placements, processors used, power management, communication, and reasoning behind over-all assembly. It is important to have a well-constructed machine that is designed to perform flexible tasks. It should be easy to make an upgrade and troubleshoot when necessary. This will ensure that the machine is constantly adaptable and up to date with hardware.

Majority of the robot was assembled by the previous student working on this project, Allan Pan. Many additional modifications were made to ensure better performance, and have additional degrees of freedom. Completely different algorithms were implemented and tested with the new modifications.

E.2 Robot Frame

The main frame is constructed with aluminum. The belt connecting the front wheel to the back wheel is constructed with rubber for grip and flexibility. The overall shape of the robot is rectangular prism. A better design would have been a cylindrical frame as it is less likely to get stuck in corners. It is also easier to manoeuvre around obstacles and jagged areas. However, modifications to the overall frame were not made at this time. The frame can be seen on Figure 3-1, version one of the robot. The dimensions of the frame are 0.43m by 0.15m by 0.29m. The wheels are extended a little past the front so that the robot will be able to climb over any small terrain with ease. The overall design is inspired from an armoured, tank.

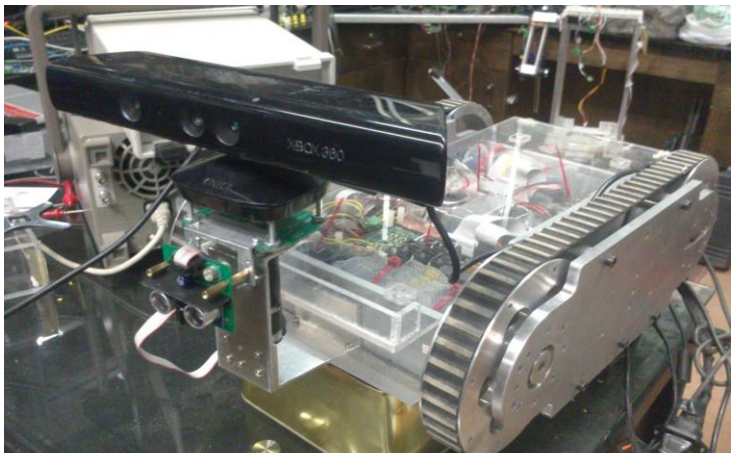


Figure E-1: Robot, version one.

E.3 Motors and Sensors

E.3.1 Version One

All the parts used in version one of the robot are listed in Table E-1.

Table E-2 Parts used in version one.

Part Name	Description	Voltage (V)
Maxon Motor T-05	This motor was used to control horizontal rotation of the Kinect.	12
2-Stock motors (no name)	These motors were received with the original frame. They are used to control both wheels.	24
Potentiometer	To estimate the direction that the Kinect is facing.	3.3
TI-Digital signal processor (DSP)	The microcontroller that controls the motor driver and the communication from the on board computer.	3.3
Kinect sensor with ARM DSP	Main sensor used for colour, depth, and sound information,	12
Sonar sensor	To identify obstacles at close ranges,	3.3

E.3.2 Version Two

All the parts used in version two of the robot are listed in Table E-2.

Table E-3 Parts used in version one.

Part Name	Description	Voltage (V)
Maxon Motor T-05	This motor was used to control horizontal rotation of the Kinect.	12
2-Maxon 343100 Motors	More efficient motors from the previous motors. They are used to control both wheels.	12
Maxon 343100 Motor	This motor is used to control the vertical rotation of the Kinect.	12
2-Potentiometer	To estimate the directional angle that the Kinect is facing (horizontal and vertical).	3.3
TI-Digital signal processor (DSP)	The microcontroller that controls the motor driver and the communication from the on board computer.	3.3
Kinect sensor with ARM DSP	Main sensor used for colour, depth, and sound information,	12

E.4 Operating Computer System

For both versions of the robot, one Lenovo Y570 laptop was used as the main computing and decision making system. The system specifications are listed below. Note that this is the system that all simulations and tests are performed on unless otherwise stated.

- Central processing unit (CPU) → Intel Core i7-2670QM CPU @ 2.20GHz Boost @ 2.99GHz.
- Random access memory (RAM) → 8.00 GB.
- Graphics processing unit (GPU) → NVIDIA GeForce GT 555M.
- 64-bit operating system, windows 7.

E.5 Assembly Analysis and Reasoning

E.5.1 Version One

Version one was derived from the original version zero, constructed by Allan Pan. Version zero had a stationary Kinect camera attached towards the front of the robot. It had an on board, small factor desktop as the main processing unit. The desktop had an ITX motherboard with Intel Core 2 Quad mobile CPU. It also had a CUDA enabled GeForce 530 GPU.

One of the main changes from version zero to version one was the placement of the Kinect sensor and a base that is able to rotate horizontally. Figure E-1 shows the sensor placement. This additional degree of freedom enabled the robot to be able to track targets without having to turn the entire frame. Not having to turn the entire frame saves power and makes it less likely to make contact with an obstacle while trying to track a target. The performance of the horizontal-rotary tracking system can be found on Chapter F: Controllers.

Since the Kinect sensor can only detect depth beyond 0.8m; a sonar sensor was mounted to the front of the robot. The sonar sensor was used as a backup sensor for obstacles within close proximity. The original desktop computer was removed and replaced with a laptop (refer to Section E.5). The laptop had better overall performance comparatively through CPU, GPU, and RAM upgrades. Most importantly, the laptop had a portable battery attached. This eliminates the wire that must be connected to the desktop for power.

E.5.2 Version Two

The major modification from version one was the placement of the optical sensor. It was moved to the back of the robot and at a much higher plane. This will help with the fact that Kinect cannot measure depth under 0.8m. The robot will be able to identify obstacles at closer distance, eliminating the use for the sonar sensor. Also, having the Kinect at the back will ensure that, when in forward motion, the first point of contact will be the tank wheels. This makes it easier to manure over terrain and small obstacles.

The horizontal rotary joint was re-designed and attached. Initially, it was designed to be closer to the Kinect, near the top. That design was re-engineered to fit near the bottom as this gave the Kinect more stability and less oscillation. In addition, a vertical rotary base was attached to give the robot an additional freedom. The design focused on implementing the motor in a vertical alignment, in the center, instead of to the side. This is done to have better balance and stability in motion while having less vibration. Figure E-2, (b) display's the vertical rotary joint and Figure E-2, (c) display's the new version of the horizontal rotary joint. All of the power was rewired to one power supply; a battery is used at this connection when navigation testing done.

The original motors were replaced with more efficient motors, Maxon 343100. These motors are smaller in size comparatively and able to produce a listed 22 watts of power. Figure E-2, (a) display's the second version of the robot.

(b) (b) (c)

Figure E-2: Robot, version two. (a) View of the entire model. (b) Vertical rotary joint with a potentiometer. (c) Horizontal rotary joint with a potentiometer.

E.6 Communication

There are two stages of communication implemented. One is the communication between the central processing computer (CPC) and the DSP, and the communication between the DSP and the motor drivers. Communication between CPC and DSP is achieved serially via RS232 connector. Commands from the CPC are initially converted internally to a 16-byte array and sent asynchronously to the DSP; which is then decoded accordingly by the DSP to perform a certain task.

The First byte of every transmission is an ASCII character. The characters are either U: upper limit, L: lower limit, S: start, P: pause and R: run. The second and third bytes are for pulse width modulation zero (PWM0), upper and lower byte respectively. The fourth and fifth bytes are for PWM1 upper and lower byte respectively. This order follows until PWM5 at eleventh and twelfth byte. Thirteenth byte is designated for Timeout. Finally, fourteenth and fifteenth bytes are assigned to cyclic redundancy check (CRC) upper and lower bytes. Initially a command with “U” with 0xFFFF and “L” with 0x0000 should be sent to the DSP to set the upper and lower limits. An array of bytes is sent with the command code “S” to start the internal timer. Now the DSP is initialized and set to receive commands that will control the robot.

When an array with “R” command is received; the PWM values and direction of each PWM values are sent to the relay that controls the motors. When the command “P” is sent, internal timer is stopped and the relay circuit is powered off.

Bibliography:

[1] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint", In Proc. *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 510-517.

[2] A. Bruhn, J. Weichert, and C. Schnorr, "Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods", *International Journal of Computer Vision*, Vol. 61, No. 3, pp. 211–231, 2005.

[3] A. Konar, G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot", *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* Vol. 43, No. 5, pp. 1141-1153, September 2013.

[4] A. Patel, D. R. Kasat, S. Jain, and V. M. Thakare, "Performance Analysis of Various Feature Detector and Descriptor for Real-Time Video based Face Tracking", *International Journal of Computer Applications*, Vol. 93, No. 1, pp. 0975 –8887, 2014.

[5] A. Prieto, F. Bellas, P. Caamano, and R. J. Duro, "Automatic Neural-Based Pattern Classification of Motion Behaviours in Autonomous Robots", *Neurocomputing*, Vol. 75, No. 1, pp. 146-155, January 2012.

[6] A. Ulusoy, S. L. Smith, X. C. Ding, and C. Belta, "Robust Multi-Robot Optimal Path Planning with Temporal Logic Constraints". In Proc. *IEEE International Conference and Robotics and Automation (ICRA), ,* 2012, pp. 4693-4698

[7] A. Kessler, "Elon Musk Says Self-Driving Tesla Cars Will Be in the U.S. by Summer", *The New York Times*, pp. B1, March 20, 2015.

[8] A. Shingade and A. Ghotkar, "Animation of 3D Human Model Using Markerless Motion Capture Applied To Sports", *International Journal of Computer Graphics & Animation (IJCGA),* Vol. 4, No. 1, pp. 27-39, 2014.

[9] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", In Proc. *Workshop on Imaging Understanding*, 1981, pp. 121-130.

[10] B. D. Lucas, "Generalized Image Matching by the Method of Differences", Department of Computer Science, Carnegie-Mellon University, Pittsburgh, USA, 1984.

[11] B. Freedman., A. Shpunt, M. Machline, and Y. Arieli, "Depth Mapping Using Projected Patterns". *http://www.ncbi.nlm.nih.gov*. May 13, 2010.

[12] B. Langmann, K. Hartmann, and O. Loffeld, "Depth Camera Technology Comparison and Performance Evaluation". In Proc. *1st International Conference on Pattern Recognition Applications and Methods*, 2012, pp. 438-444.

[13] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow", *Artificial Intelligence*, Vol. 17, No. 1, pp. 185–203, 1981.

[14] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller II", *IEEE Trans Systems, Man, and Cybernetics*, Vol. 20, No. 2, pp. 419-435,1990.

[15] C. Harris and M. J. Stephens, "A Combined Corner and Edge Detector", In Proc. *Conference on Alvey Vision*, 1988, pp. 147–152.

[16] C. Cheng and H. Li, "Feature-Based Optical Flow Computation". *International Journal of Information Technology*, Vol. 12, No. 7, pp.82-90, 2006.

[17] D. J. Fleet and Y. Weiss, "Mathematical Models in Computer Vision: The Handbook," N. Paragios, Y. Chen, and O. Faugeras (Eds.), Chapter 15, Springer, 2005, pp. 239-258

[18] D. Lowe, "Object Recognition from Local Scale-Invariant Features", In Proc. *International Conference on Computer Vision ICCV*, *Corfu*, 1999, pp. 1150–1157.

[19] D. Shukla and S. Biswas, "Region Filter and Optical Flow based Video Surveillance System", *International Journal of Computer Applications*, Vol. 63, No. 6, pp.5-12, 2013.

[20] E. H. Mamdani, "Application of Fuzzy Algorithms for the Control of a Simple Dynamic Plant", In Proc. *IEEE*, 1974, pp. 121-159.

[21] E. Rosten and T. Drummond, "Machine learning for Highspeed Corner Detection", In Proc. *European Conference on Computer Vision*, 2006, pp. 430–443.

[22] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: An Efficient Alternative to SIFT or SURF", In *Proc. International Conference on Computer Vision (ICCV),* 2011, pp. 2564–2571.

[23] E. Patel and D. Shukla, "Comparison of Optical Flow Algorithms for Speed Determination of Moving Objects". *International Journal of Computer Applications,* Vol. 63, No. 5, pp. 32-36, 2013.

[24] F. Duchon, A. Babinec, M. Kajan, P. Beno, M. Florek, T. Fico, and L. Jurisica, "Path Planning with Modified A-star Algorithm for Mobile Robot", *Science Direct, Precedia Engineering*, Vol. 96, No. 1, pp. 59-69, 2014.

[25] G. Somanath, S. Cohen, B. Price, and C. Kambhamettu. "Stereo+Kinect for High Resolution Stereo Correspondences", In Proc. *IEEE International Conference, 3D Vision,* pp. 9-16, 2013.

[26] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features", *Computer Vision and Image Understanding*, Vol. 110, No. 3, 2008, pp. 346–359.

[27] H. N. Joshi, "An Image Based Path Planning Using A – Star Algorithm", *International Journal of Emerging Research in Management &Technology,* Vol. 3, No. 5, pp. 127-131, 2014.

[28] I. J. Eyoh and U.A. Umoh, "A Comparative Analysis of Fuzzy Inference Engines in Context of Profitability Control". Department of Computer Science, University of Uyo, Nigeria, 2013.

[29] J. C. Basilio and S. R. Matos, "Design of PI and PID Controllers with Transient Performance Specification", *IEEE Transactions on Education*, Vol. 45, No. 4, pp. 364-370, November 2002.

[30] J. Canny**, "***A Computational Approach to Edge Detection"*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, pp. 679–698, 1986.

[31] J. Hagelback, "Potential-Field Based Navigation in Starcraft," In Proc. *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 388-393.

[32] J. Vascak, "Navigation of Mobile Robots Using Potential Fields and Computational Intelligence Means", *Acta Polytechnica Hungarica*, Vol. 4, No. 1, 2007, pp. 63-74.

[33] J. Zhong, "PID Controller Tuning: A Short Tutorial", 2006.

[34] J. J. Gibson, *The Perception of the Visual World*, Houghton Mifflin, 1950.

[35] J. W. Choi, T. K. Whangbo, and C. G. Kim. "A Contour Tracking Method of Large Motion Object Using Optical Flow and Active Contour Model". *Multimedia Tools and Applications*, Vol. 74, No. 1 pp.199-210, 2013.

[36] J. Li, and J. Wang, "Robust Object Tracking Algorithm based on Sparse Eigenbasis", *IET Computer Vision*, Vol. 8, No. 6, pp. 601-610, 2014.

[37] J. Shang, "A Novel Fragments-based Similarity Measurement Algorithm for Visual Tracking", *Journal of Computer*, Vol. 9, No. 9, pp. 2167-2172, 2014.

[38] K. F. Uyanik, "A Study on Artificial Potential Fields", Department of Computer Engineering, Middle East Techinical University, Ankara, Turky.

[39] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications", *Sensors (Basel,Switzerland)*, 2012, Vol. 12, No. 2, pp. 1437-1454.

[40] K. R. T. Aires, A. M. Santana, and A. A. D. Medeiros, "Optical Flow Using Color Information". ACM New York, NY, USA., 2008.

[41] L. Juan and O. Gwun, "A Comparison of SIFT, PCA-SIFT and SURF", *International Journal of Image Processing (IJIP)*, Vol. 3, No. 4, pp. 143-150, 2009.

[42] L. V. Fausett, "Fundamentals of Neural Networks: Architectures, Algorithms, and Applications", Prentice-Hall Englewood Cliffs, NJ, 1994.

[43] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features", In Proc. *ECCV,* 2010, pp. 778–792.

[44] M. R. Andersen, T. Jensen, P. Lisouski, A. K. Mortensen, M. K. Hansen, T. Gregersen, and P. Ahrendt, "Kinect Depth Sensor Evaluation for Computer Vision Applications", Aarhus University. pp. 1-34, 2012.

[45] M. Bhat, P. Kapoor, and B .L .Raina, "Application of SAD Algorithm in Image Processing for Motion Detection and Simulink Blocksets for Object Tracking", *International Journal of Engineering Science and Advanced Technology*, Vol. 2, No. 3, pp.731–736, 2012.

[46] N. Nourani-Vatani, P. V. K. Borges, and J. M. Roberts. "A Study of Feature Extraction Algorithms for Optical Flow Tracking". In Proc. *Australasian Conference on Robotics and Automation*, 2012, pp. 1-7.

[47] P. M. Panchal, S. R. Panchal, and S. K. Shah, "A Comparison of SIFT and SURF", *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 1, No. 2, April 2013, pp. 2320 – 9801.

[48] P. Gwosdek, S. Grewenig, A. Bruhn, and J. Weickert, "Theoretical Foundations of Gaussian Convolution by Extended Box Filtering", Mathematical Image Analysis Group,

Department of Mathematics and Computer Science, Campus E1.1, Saarland University, Germany.

[49] S. G. Cui, H. Wang, and L. Yang, "A Simulation Study of A-star Algorithm for Robot Path Planning", In Proc. *16th International Conference on Mechatronics Technology*, 2012, pp. 506-510.

[50] S. H. Dezfoulian, "A Generalized Neural Network Approach to Mobile Robot Navigation and Obstacle Avoidance", Department of computer science, University of Windsor, Windsor, Canada, 2011.

[51] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary Robust Invariant Scalable Keypoints", In Proc. *International Conference of Computer Vision*, 2011, pp. 2548–2555.

[52] S. Malu, and J. Majumadar, "Kinematics, Localization and Control of Differential Drive Mobile Robot", In Proc. *Global Journal of Researches in Engineering: Robotics & Nano-Tech*, Vol. 14, No. 1, pp. 1-7, 2014.

[53] S. P. Day, and M. R. Davenport, "Continuous-Time Temporal Back-Propagation with Adaptive Time Delays", *IEEE Transactions on Neural Networks*, Vol. 4, No. 2, 1993, pp. 348-353.

[54] S. Patil, J. Van Den Berg, S. Curtis, M. C. Lin, and D. Manocha, "Directing Crowd Simulations Using Navigation Fields", *IEEE Trans. Visualization and Computer Graphics,* Vol. 17, No. 2, pp. 244-254, February 2011.

[55] S. R. Lindemann, and S. M. Lavalle, "Smoothly Blending Vector Fields for Global Robot Navigation", Department of Computer Science, University of Illinois Urbana, USA.

[56] S. S. Beauchemin and J. L. Barron, "The Computation of Optical Flow", ACM New York, USA.

[57] S. M. LaValle, "Planning algorithms", Cambridge University Press, New York, USA, 2006.

[58] S. Avidan, "Ensemble Tracking", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29, No.2, pp. 261-271, 2007.

[59] T. Cain, "Practical optimizations for A* path generation", AI Game Programming Wisdom. Charles River Media, pp. 146-152, 2002.

[60] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, “Introduction to Algorithms”, Second Edition. MIT Press and McGraw-Hill, 2001.

[61] T. M. Ravikumar, R. Saravanan, and N. Nirmal, “Modeling and Optimization of Odometry Error in a Two Wheeled Differential Drive Robot”, In Proc. *International Journal of Scientific and Research Publications*, Vol. 3, No. 12, pp. 1-7, December 2013.

[62] T. W. Manikas, K. Ashenayi, and R. L. Wainwright, "Genetic Algorithms for Autonomous Robot Navigation", *IEEE Instrumentation and Measurement Magazine*, Vol. 10, No. 6, pp. 26 -31, 2007.

[63] V. Podlozhnyuk. “Image Convolution with CUDA”, nVidia, pp. 1-21, 2007.

[64] V. B. Le, A. T. Nguyen, and Y. Zhu, “Hand Detecting and Positioning Based on Depth Image of Kinect Sensor”, *International Journal of Information and Electronics Engineering*, Vol. 4, No. 3, pp. 176 -179, May 2014.

[65] W. H. Al-Sabban, L. F. Gonzalez, E. N. Smith, G. F. Wyeth, “Wind-Energy Based Path Planning for Electric Unmanned Aerial Vehicles using Markov Decision Processes”, In Proc. of *the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[66] W. Zeng, “Microsoft Kinect Sensor and Its Effect”, *IEEE Computer Society*, Pp. 4-10, 2012.

[67] X. Song, H. Fang, X. Jiao, and Y. Wang, "Autonomous Mobile Robot Navigation Using Machine Learning", In Proc. *Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on,* 2012, pp. 135 – 140

[68] X. Chai, G. Li, Y. Lin, Z. Xu, Y. Tang, and X. Chen, “Sign Language Recognition and Translation with Kinect”, *Microsoft Research Asia,* pp. 1- 2.

[69] X. Cheng, N. Li, S. Zhang, and Z. Wu, “Robust Visual Tracking with SIFT Features and Fragments Based on Particle Swarm Optimization”, *Circuit, Systems, and Signal Processing*, Vol. 33, No. 5, pp. 1507–1526, 2013.