2016

Motion Planning and Control of Differential Drive Robot

Kaamesh Kothandaraman
Wright State University

MOTION PLANNING AND CONTROL OF DIFFERENTIAL DRIVE MOBILE ROBOT

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Engineering

By

KAAMESH KOTHANDARAMAN

B.E, RTMN University, Nagpur, Maharashtra, India, 2012

2016

Wright State University

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

December 2, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Kaamesh Kothandaraman ENTITLED Motion Planning and Control of Differential Drive Robot BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

Kuldip S. Rattan, Ph.D.
Thesis Director

Brian Rigling, Ph.D.

Chair
Department of Electrical Engineering
College of Engineering and
Computer Science

Committee on
Final Examination

Kuldip S. Rattan, Ph.D.

Marian K. Kazimierczuk, Ph.D.

Xiaodong Zhang, Ph.D.

Robert E. W. Fyffe, Ph.D.
Vice President for Research and
Dean of the Graduate School

Abstract

Kothandaraman, Kaamesh, M.S.Egr. Dept of Electrical Engineering, Wright State University, 2016. Motion Planning and Control of Differential Drive Mobile Robot.

Motion planning and control of a differential drive robot in a supervised environment is presented in this thesis. Differential drive robot is a mobile robot with two driving wheels in which the overall velocity is split between left and right wheels. Kinematic equations are derived and implemented in Simulink to observe the theoretical working principle of the robot. A proportional controller is designed to control the motion of the robot, which is later implemented on a physical robot. Combination of linear velocity and orientation generates individual wheel velocities which are sent to the robot by wireless communication for its motion. Point to point motion and shapes, such as square and circle, are performed to implement and test the robustness of the controller designed. Others tasks such as leader-follower, pursuer-evader and obstacle detection and avoidance are carried out using multiple robots. For flexibility of the robot, a GUI software is developed for waypoint assignment for the robots to follow through. Different systems such as Vicon Camera, Matlab-Simulink, Xbees, QUARC and robots were integrated via software and hardware in the UAV lab at Wright State University. The camera system used throughout this thesis helps to understand the functioning of a GPS. QUARC, Matlab-Simulink, XBees are used in processing and communication of data from cameras to the robot.

Contents

List of Figures

Acknowledgments

I thank my advisor, Dr.Kuldip Rattan for helping me understand and giving me directional advice for completion of this thesis. I also acknowledge the help from Mohsen Khalili, Andrew Szabo, Neel Dalwadi and Arun Raman for helping in getting information about the system setup and robots. I specially thank Dr.Xiaodong Zhang for letting me use the UAV Lab at Wright State University for performing the experiments.

Dedicated to my family and friends.

Introduction

Robotics is the branch of engineering and computer science that deals with the design, construction, operation, and application of robots [10]. Robots are mainly used for automation of a process to increase efficiency. There are different types of robot such as ground, industrial robot, UAV, nature inspired and nano-robots. In general, the design of the robot changes with it's application. In this thesis, the robot under consideration is a differential drive robot with two driving wheels in which the velocity of the robot is split between left and right wheel. A castor wheel is sometimes added to avoid tipping of the robot. Robots are one of the technological advancement that humans are working on for many years. All these years of scientific study and research on robots have shown almost infinite possible application of robotic systems. In this thesis, the study of differential drive robot is undertaken and experimental tasks are performed to test the applications of such robots.

A great amount of information about the concept of differential drive robot was gained while studying the thesis work of Cory Snyder in *An Intuitive approach to formation Control of Differential Drive Robots* [1]. The system of robot and their kinematic equations written in his thesis were well understood before starting the work on this thesis. The real motivation behind this thesis is to realize the kinematics equations to a physical robot and design and test the controller. This study is extended to include experimental tasks to increase the flexibility and application of mobile robots.

Another approach for the robot model is explained in a work by Gregory Dudek and Michael Jenkin, *Computational Principles of Mobile Robotics* [2]. In this study, the robot kinematics is explained with the concept of *Instantaneous Center of Curvature* (ICC). This approach was found very useful in the formation of circular motion. The objective of this thesis is to study and implement the kinematics equations, design and test the controller for the robot, and plan paths for the robot. Many task have been designed to test the controller and application of the robot. The outline of this thesis is as follows.

Chapter 1 discusses the derivation of the mathematical model, i.e kinematic equations of the robot and its implementation in Simulink. Chapter 2 explains the system setup used in this project. All the systems such as camera system, the robot(s) and the communication involved in this thesis are explained in detail. In chapter 3, the controller is tested with the help of basic motion. Point to point and various other shapes are explained and related results are shown. Chapter 4 discusses the obstacle detection and avoidance along with the associated path planning for avoiding the obstacle is also discussed. Chapter 5 introduces how multiple robots are used for task such as leader-follower and persuer-evader and the results are explained. To increase the flexibility of the application of the robots, a GUI- Graphical User Interface is created in chapter 6. This software interface lets user provide the way-point for the robot in its path. The summary of the thesis is given in the chapter 7 and the conclusion is also drawn in this chapter.

Chapter 1

Mathematical Model

A differential drive robot is a wheeled robot with two controllable wheels as shown in figure 1.0.1. To maneuver any differential drive robot in a plane, the robot needs a linear velocity V and a heading θ. By controlling the velocity and orientation, the path of the robot can be planned.

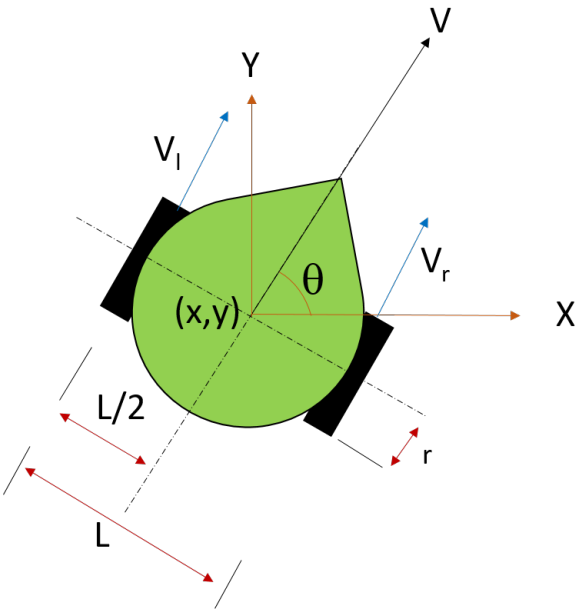


Figure 1.0.1: Differential Drive Mobile robot parameters.

Figure 1.0.1 shows the differential drive robot layout, where V_r is the velocity of

the right wheel, V_l is the velocity of left wheel, L is the distance between the center of the wheels, θ is the directional angle of the robot with respect to the reference coordinate system and r is the radius of the wheel. The velocity of the robot is the average of the individual wheel velocities and is given by

$$V = \frac{V_r + V_l}{2} \tag{1.0.1}$$

While modeling the robot,the following assumptions are made.

1. Robot is moving with constant velocity.
2. The duration of the robot motion is within a small period of time and so the velocities V_r and V_l would be constant in that duration.
3. The wheels of the robots do not slip and the surface for robot motion is flat.

These assumptions are made because the individual velocities are time variant components and they change with the passage of time. So within time t_1 to t_2, these velocities can be considered constant.

1.1 System setup

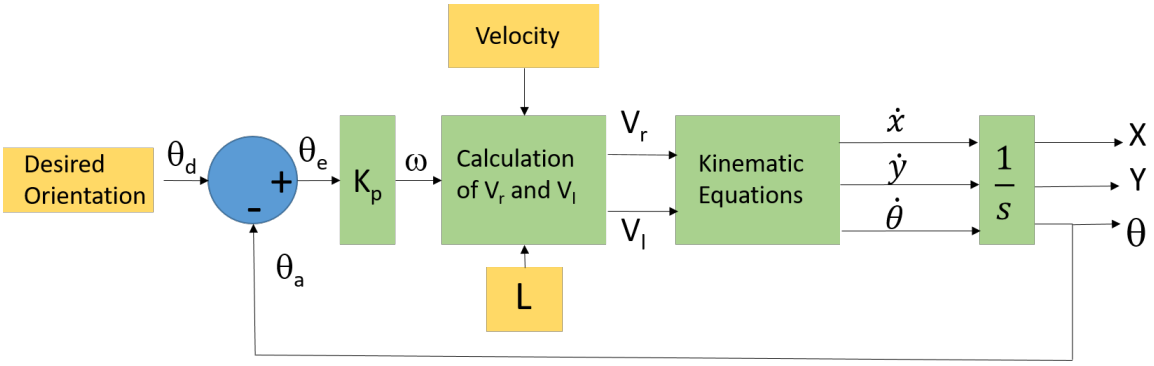


Figure 1.1.1: Block diagram of robot system.

Figure 1.1.1 shows the complete block diagram of the system. It is desired to move the robot in a certain direction θ and with constant velocity V. From these parameters, the velocity for left wheel (V_l) and right wheel (V_r) and are sent to the kinematic equations to calculate the linear and angular velocities of the robot. After integrating the angular velocity, resultant orientation is then fed back as input to make the system a closed loop system, increasing its stability. The different blocks of these systems are explained in details below.

1.1.1 Orientation error

The heading of the robot is controlled to keep the robot on the planned track by keeping the orientation error to minimum. In this block, the error in the orientation is calculated and then this error is fed to the next block to calculate the individual wheel velocities V_l and V_r. Normal mathematical subtraction of the angles produce a result which may not be desired in terms of its sign and value. In order to avoid this, the angle of rotation error is restricted between 0 to 180 degrees and 0 to -180 degrees. For precise calculation and to avoid erroneous orientation, the function of $atan2$ is used. Matlab code used for θ correction is

```
1 theta_difference = theta_desired-theta_actual;
2 theta_error= ...
    atan2d(sin(theta_difference),cos(theta_difference));
```

The ***atan2*** function [9] is a four quadrant function, which produces the angle within the four quadrants of the plane. It helps to determine the sign and the right quadrant placement of the resultant angle, within the range of $(-\pi, \pi]$. The counter-clockwise angles are considered positive and clockwise angles are considered negative.

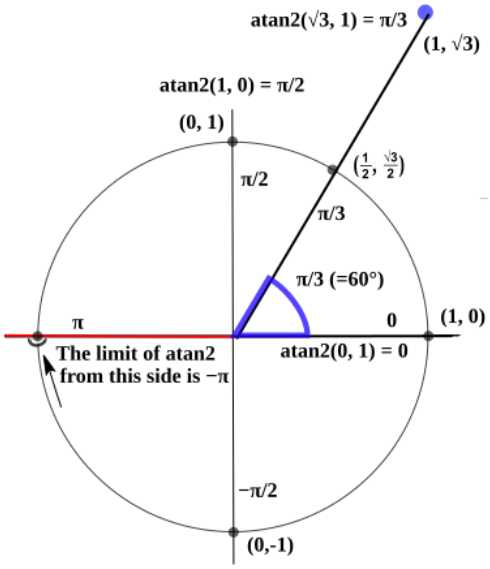


Figure 1.1.2: Angles for atan2. *ImagefromWikipedia*.

To help understand the *atan2* function, let's consider the four quadrant shown in figure 1.1.2 [11]. The angle from the origin to the point(1,1) using *atan* function is 45° and angle to point (-1,-1) using *atan* function is also 45°, which is clearly inaccurate. Performing the same task using *atan*2*d* would produce 45° and −135° for the points. The *atan*2*d* function considers the individual sign of both the components and places the angle in right quadrant, thus giving the correct resultant angle. In another example, the *atan* function to calculate 90° atan($\frac{1}{0}$) would produce error because of presence of 0 in denominator, whereas with *atan*2*d* the angle produced would be 90°. Hence throughout the thesis the function of *atan*2*d* is used extensively, where *d* is for angle in degrees.

1.1.2 Individual wheel velocity calculation

Kinematic equations of a mobile robot

Kinematics is the branch of classical mechanics which describes the motion of a point (object's center) without consideration to the mass of the objects or the forces

that may have caused the motion [4]. The kinematic equations are used to transform the motion from polar coordinates (r, θ) to rectangle coordinate (x, y) system.

As mentioned earlier, the inputs required for the motion of a mobile robot are the linear velocity (V) and the orientation θ. The rate of change of position of robot in x-direction is $\dot{x}$ and that in y-direction is $\dot{y}$ are given by

$$\dot{x} = V cos(\theta) \tag{1.1.1}$$

$$\dot{y} = V sin(\theta) \tag{1.1.2}$$

and the angular velocity of the robot is given by

$$\dot{\theta} = \omega = \frac{(V_r - V_l)}{L} \tag{1.1.3}$$

Substituting the linear velocity V from equation (1.0.1), in equations (1.1.1) and (1.1.2), modifies $\dot{x}$ and $\dot{y}$ as,

$$\dot{x} = \frac{(V_l + V_r)}{2} cos(\theta) \tag{1.1.4}$$

$$\dot{y} = \frac{(V_l + V_r)}{2} sin(\theta) \tag{1.1.5}$$

The velocity V of the robot in a fixed reference coordinate system is therefore given by

$$V = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{1.1.6}$$

From equations (1.1.4) and (1.1.5),

$$V = \sqrt{(\frac{(V_l + V_r)}{2} cos(\theta)) + (\frac{(V_l + V_r)}{2} sin(\theta))} = \frac{V_r + V_l}{2} \tag{1.1.7}$$

The individual velocities, V_r and V_l, can now be calculated using the equations

(1.0.1) and (1.1.3)

$$V_r = (V + \frac{L}{2}\omega) \tag{1.1.8}$$

$$V_l = (V - \frac{L}{2}\omega) \tag{1.1.9}$$

The outputs, V_r and V_l can now be used to generate the output $\dot{x}, \dot{y}$ and ω using equations (1.1.4), (1.1.5), (1.1.3). Integrating these values results in the actual position x, y and the actual orientation θ. The actual orientation θ is fed back for error calculation, making this system a closed loop system.

1.1.3 Controller Gain

By taking Laplace transform of equation (1.1.3) gives,

$$s.\Theta(s) = \Omega(s) \tag{1.1.10}$$

$$\therefore G(s) = \frac{\omega(s)}{\theta(s)} = \frac{1}{s} \tag{1.1.11}$$

The transfer function $G(s)$ is a type-1 system, therefore, the closed loop system will have zero steady-state error with proportional gain. The transfer function of the controller for closed loop system as shown in the figure 1.1.3 is,

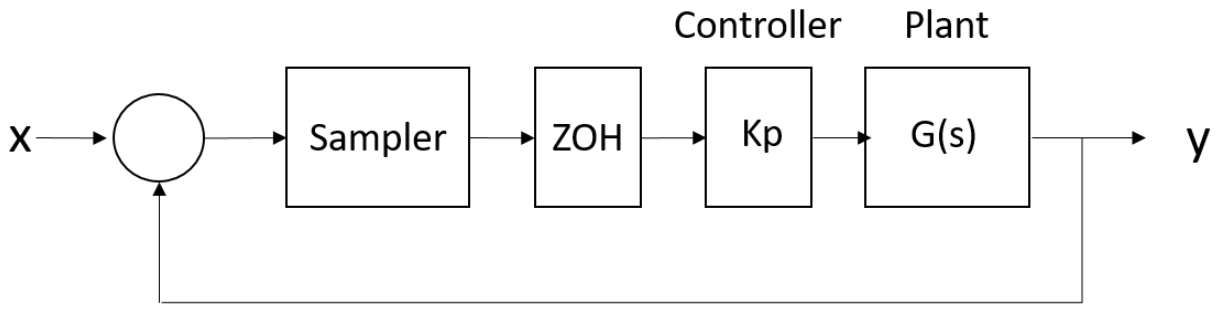


Figure 1.1.3: Closed Loop system.

$$G_c(s) = \frac{K_p G(s)}{1 + K_p G(s)} = \frac{K_p}{s + Kp} \tag{1.1.12}$$

From equation (1.1.12), it can be observed that there is only one pole at $s = -K_p$. The sampling time for the thesis work was set as 0.05 sec or 50 ms. As a thumb rule the smallest time constant of the system should be 10 times the controller's time constant, i.e 500 ms. So the gain for the position controller comes to be 2.

1.1.4 Complete block

All the blocks and the equations discussed above are merged together to form a complete robot control model.

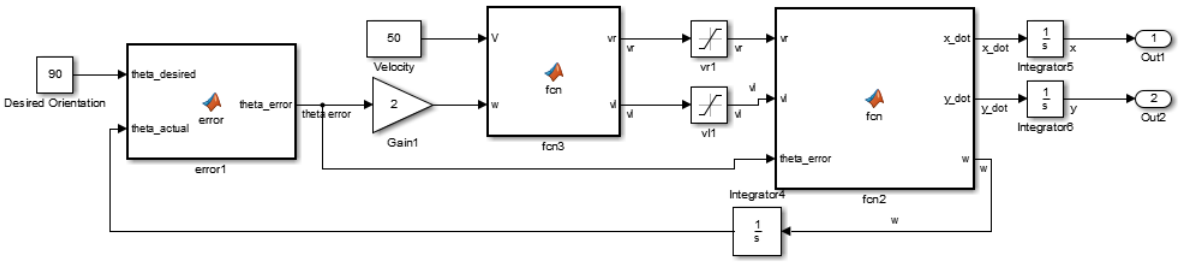


Figure 1.1.4: Complete control block.

Figure 1.1.4 shows the closed loop system implemented based on the block diagram shown in figure 1.1.1 and the mathematical equations (1.1.4), (1.1.5) and (1.1.3). The inputs are the desired orientation and velocity is set as constant which provides the output as the position (x, y) of the robot and the orientation θ. Controlling the θ, a better orientation can be expected and hence the position of the robot is controlled with more precision.

1.1.5 Simulation Results

For simulation, the linear velocity for the robot was set as 50. The V_r and V_l are limited between 500 and -500 as constant values.

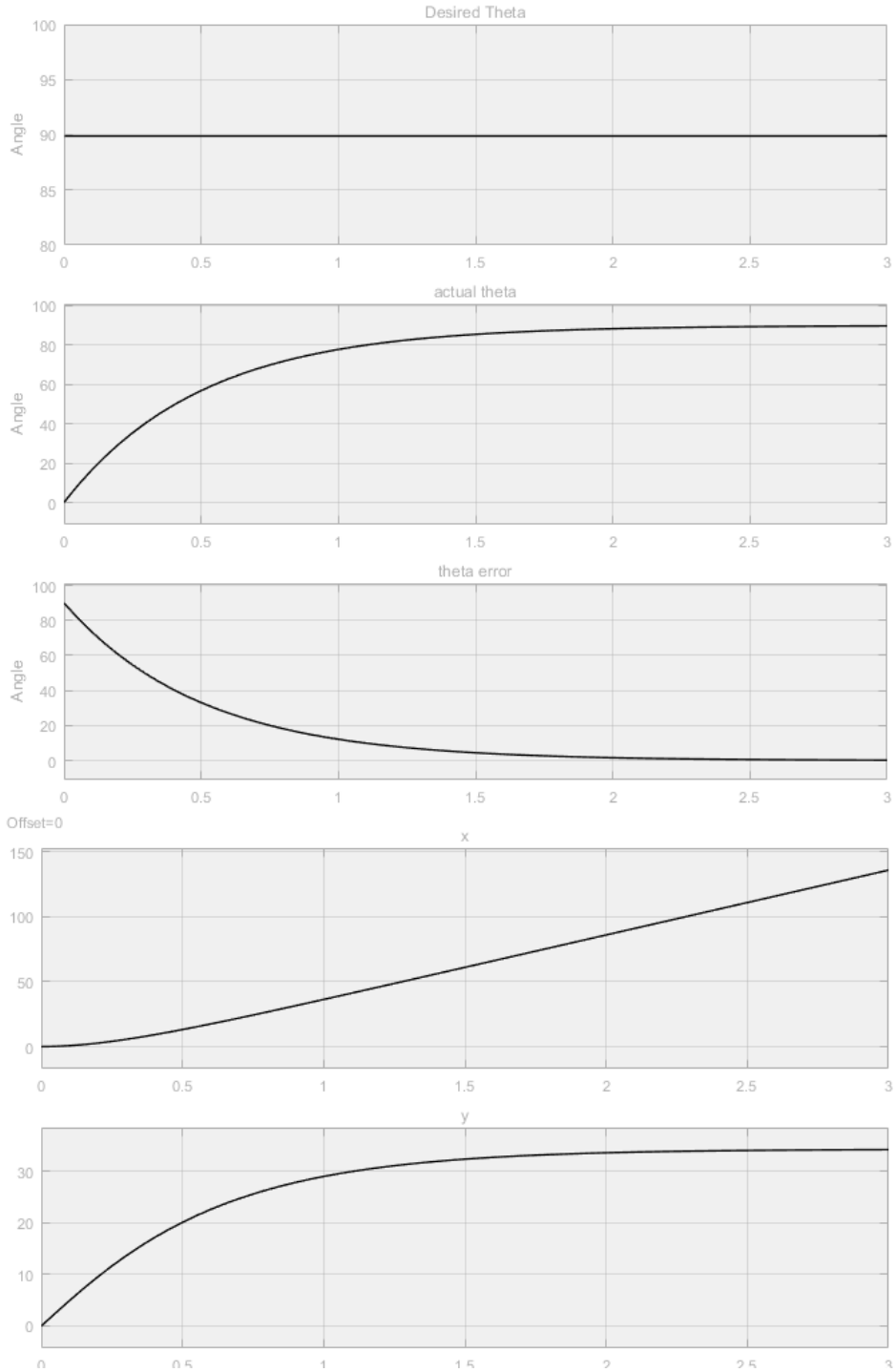


Figure 1.1.5: Simulation Result.

Figure 1.1.5 shows the simulation results. The first plot shows the desired orientation angle θ, which was considered as 90 degrees. Second plot shows the actual

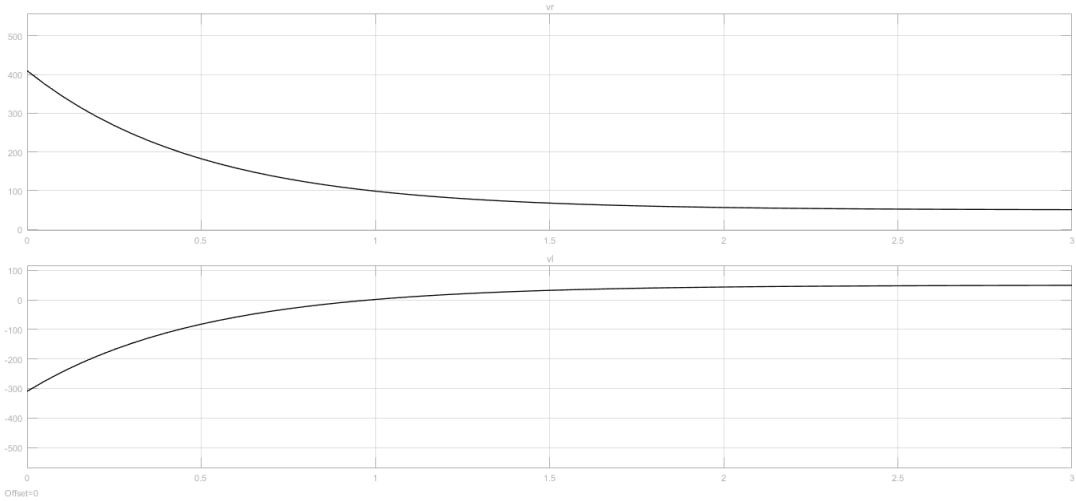


Figure 1.1.6: Simulation Result for Vr and Vl.

orientation of the robot tending towards the desired angle. Hence, the theta error in the third plot can be observed to reach zero. The later plots shows the current position of robot in x and y coordinates. From the plot it can be seen that, initially the values of x and y are zero, this shows that the robot is moving in the same spot and begins to move as it's orientation reaches near to the desired angle. This behavior of the robot can be observed from the figure 1.1.6 where the velocities V_r and V_l are initially constant with max velocity denoting motion is same place. Later they tend to 50 as soon as the robot orientation is near the desired angle. After reaching the desired direction, the robot moves straight in that direction.

Chapter 2

Experimental Setup

To accomplish any task with a robot, integration of software and hardware systems is required. These systems include Vicon camera system, Quanser's QUARC- real-time controller, Xbee and the robot(s). All these systems have to communicate with each other and must have minimum latency. The integration of all the systems was previously done in the UAV Lab at Wright State University.

2.1 System overview

The motion capture of the robot's position (x,y) and orientation (θ) is carried by the Vicon camera system. These positions are transmitted to Matlab-Simulink via Ethernet port and then processed by the control logic blocks within simulink in real-time. The data are then transmitted to the robot with the help of QUARC and a pair of Xbees. Figure 2.1.1 shows the overall system setup.

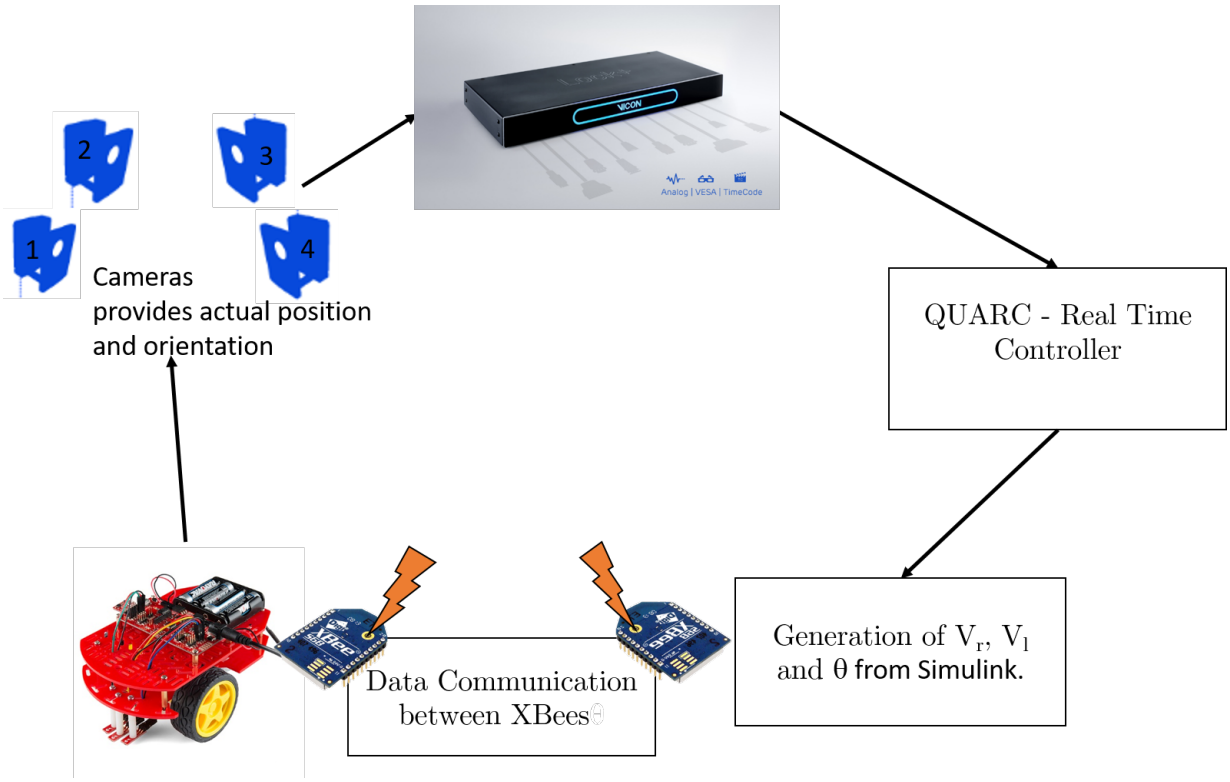


Figure 2.1.1: System Setup.

2.2 Motion Capture

To capture the motion of the robot(s), Vicon camera system [5] was used. Vicon is a motion capture company that specializes in gathering precise, reliable data for movement analysis application. To obtain the position and orientation of any object, atleast four cameras are needed. This is equivalent to the GPS where 4 satellites are required for precise positioning. The software of Vicon cameras maps the whole working space, as seen by the cameras, and sets up the reference co-ordinate system for the robots and other objects. In the UAV lab at Wright State University, four cameras are installed accurately on walls, equidistant to each other, for precise coordinate setup.

The coordinates of the robot is in 3-Dimensional space, i.e, the positions related

to the object are in X, Y, Z axes and Euler angles, Roll (ψ), Pitch (ϕ) and Yaw (θ). Because the robot in consideration is a ground vehicle, it moves in a two dimensional plane, the Z directional data and the angles ψ and ϕ are ignored.

Setting the camera system for use includes making sure that the cameras are in coordination with each other. In the event of disagreement between the cameras, the object may not be visible. The camera provides the X, Y, Z distances in meters and the Euler angles, Roll, Pitch and Yaw, in radians. Note that the Euler angles of the object are set as 0 irrespective of the orientation of the robot. It means, before creating an object, the object must be oriented properly. The positions and angles are calculated with respect to the reference coordinate system provided by the Vicon.

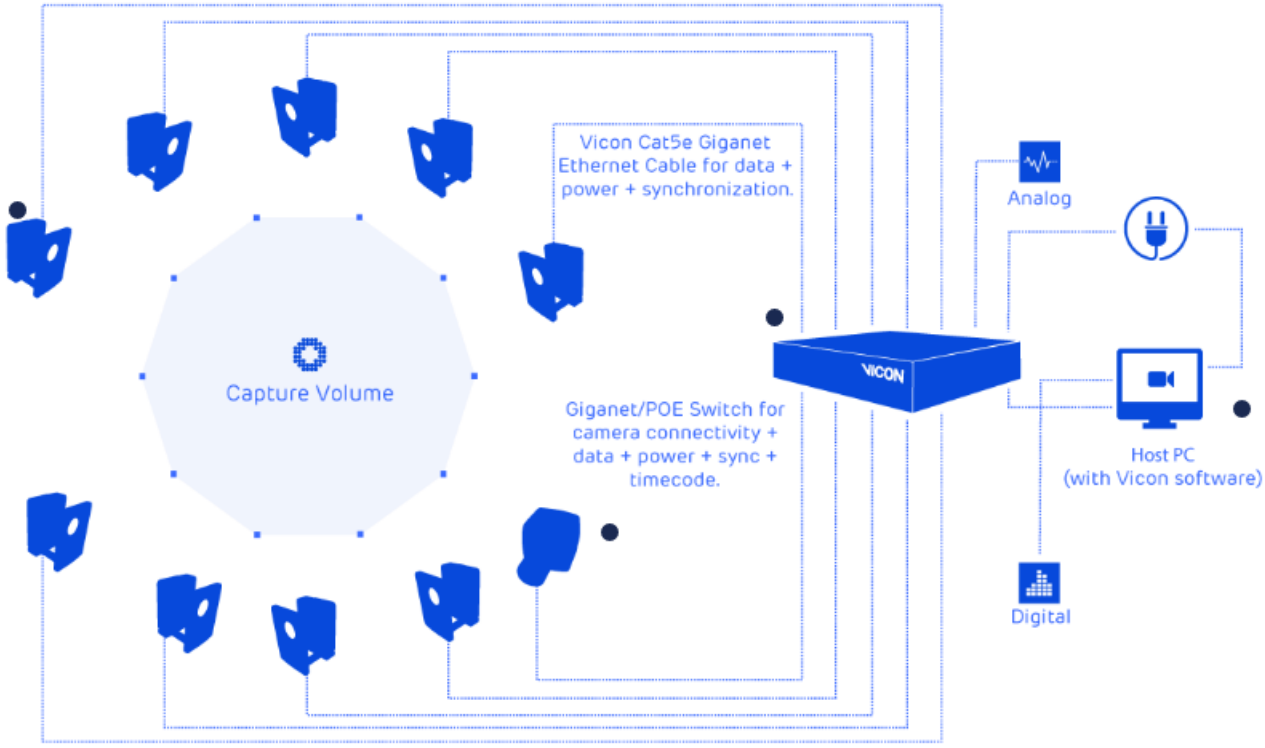


Figure 2.2.1: Vicon Communication. *Image courtesy of Vicon website* [5].

Figure 2.2.1 shows the communication flow of the Vicon system. This includes cameras, Sync box and PC. Sync box provides a single communication point between

the cameras and PC. Interface software called Tracker is used to create objects, change its parameters and observe the results.

2.2.1 Creation of an Object

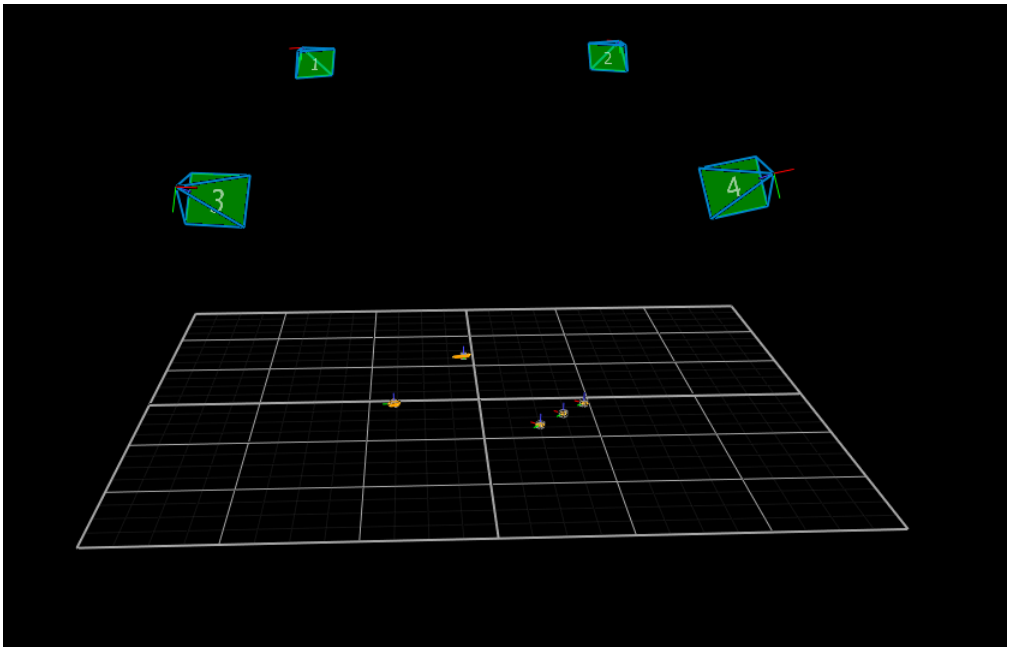


Figure 2.2.2: Perspective view of object from cameras.

Figure 2.2.2 shows the perspective view of the entire work space as seen by the Vicon cameras. To create an object, the robot is tagged with retro-reflective markers. Atleast four markers are required for tagging and these are arranged around the center of the axis going through the wheels to identify its *center*. Thus camera provides the position and orientation of this *center*. The pattern of markers' arrangement must be irregular, to distinguish between various other robots and objects in the workspace.

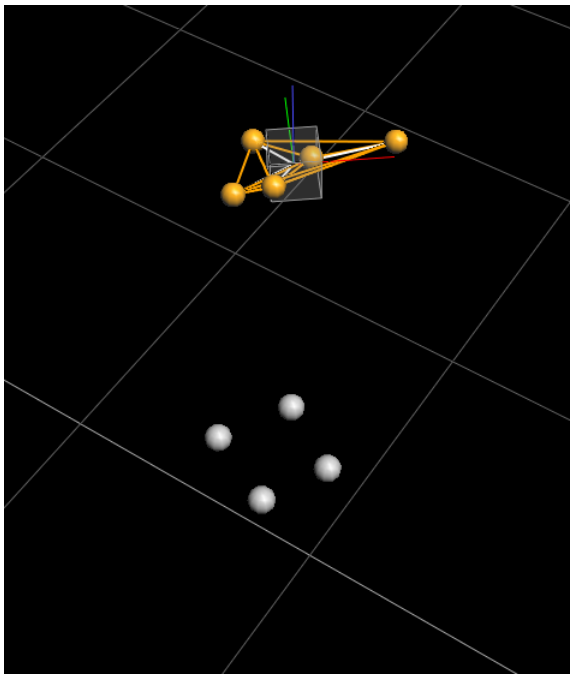


Figure 2.2.3: Object vs no object.

Figure 2.2.3 shows the creation of an object from the marked pattern. The lines between the markers confirms that the object has been created. The markers without lines shows that the object is yet to be created.

After establishing communication between Vicon system and Simulink via Ethernet, the cameras are calibrated for synchronization and detection. This is done with the help of a calibration wand. Following steps are carried out to create an object :

1. Select all the markers used for tagging the object. These marker can be seen as dots on screen as shown in figure 2.2.3. Selecting them together gives an option to create and name the object. The Vicon system provides the position of the center of robot with respect to the reference coordinate system.

2. Add this object in the simulink. While adding , it is important to make sure that the name matches with the created one. The name of the object in simulilnk is set as: *Object name.Object name.*

3. After creating the object, the position and orientation of the object is available.

Whenever this object is placed in the vicinity of the cameras, it's position and orientation can be obtained in simulink block.

2.3 Real-Time Controller

For the robot to react to the updated velocities produced by simulink, the communication between all the systems must be in real-time. To make sure that the response time of the robot is minimum, a software called QUARC is used. QUARC is real-time control system software developed by a company called Quanser. The positions received from the Vicon system is processed in simulink for producing required velocities for the robot. Integration of software-program on any hardware in real-time is called *Hardware in Loop* (HIL).

The velocities produced from simulink is sent to the robot by QUARC with a set baud rate over USB port. For the thesis purpose, the baud rate was set as 115200. This data is sent serially with the help of a pair of XBees. One of the Xbee is connected to the USB port and other one is installed on the robot.To transmit the data serially, it is encrypted in 8-bit format packed between checksum bits for secure transmission. The rest of the data communication between the computer and the robot is taken care by the Xbee devices and Zigbee Protocol.

2.4 XBee

In-order to send the processed velocities to robot, Xbee devices were used. Xbees are the devices which are used for setting up wireless communication. These devices works on Zigbee® wireless protocol. Zigbee was created and validated by an alliance of more than 300 leading semiconductor companies. According to its website [6] , it is based on IEEE 802.15.4 standard for wireless communication. This protocol is

mainly developed for simple connectivity, lower battery consumption and provides minimum latency. On the contrary, IEEE 802.11 protocol (Wifi) has a high power consumption. For Xbees, the communication occurs within 868-868.8 MHz, 902-928 MHz or 2.400- 2.4835 GHz Industrial Scientific and Medical (ISM) bands. Of these, the 2.4 Ghz band range is the most popular and widely used.

Another advantage of Zigbee protocol is that the Xbees can be designed to form a mesh network. Using mesh network, communication between Xbees which are located farther than their range can communicate easily. They can also be set up as point to multi-point network topology. Image 2.4.1 shows a Xbee device.

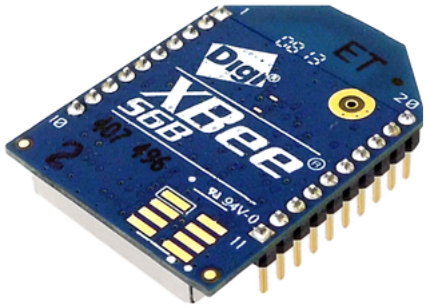


Figure 2.4.1: Xbee Module.

2.5 Robot

The Sparkfun Redbot used in this thesis is based on *Arduino*. Using basic *Arduino* commands, the robot can be moved in a straight line(front and reverse) and in curved path. The onboard arduino board is equipped with Xbee device for receiving the data. This Xbee module is paired with the module on the computer. Figure 2.5.2 shows the *Arduino* controller installed on the chassis of the Redbot used for this thesis work.

Upon receiving the velocities from computer, the robot, using in-built program commands, makes corresponding motions. The velocities generated from the simulink is provided to the wheels of the robot.

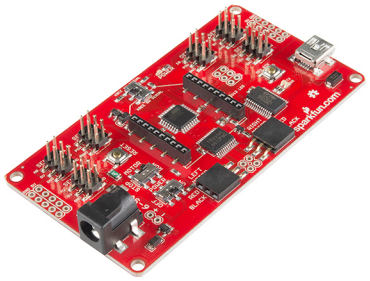


Figure 2.5.1: Redbot Controller.

There are other sensors available for this robot but they have not been used for this thesis. In future applications they can be used for better control and flexibility.

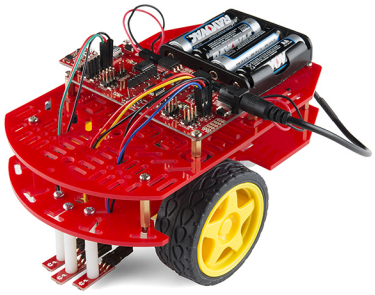


Figure 2.5.2: Redbot.

2.5.1 Programming the Robot

For programming Redbot, *Arduino* controller board is used. Few library commands for basic motion include

1. motors.drive(*num*) to drive forward.
2. motors.stop() to stop.
3. motors.brake() for braking.
4. motors.leftMotor(*num*) to turn left.
5. motors.rightMotor(*num*) to turn right.

The *num* is a numeric values for the PWM to drive the robot at particular speed. The range of *num* is from -255 to 255. The positive values drive the robot in forward direction and the negative values drive the robot in reverse direction. Higher the value of PWM, higher is the velocity. So 255 would give the maximum velocity and -255 would drive the robot with maximum velocity in the reverse direction. For experimental purposes, the PWM values are doubled while sending it to the robot. That means, when a value of 50 is generated from simulink, the actual value sent to the robot is 100. That's why the velocities is limited to -127 to 127.

2.6 Summary

Different systems such as Vicon camera system, QUARC - (real-time controller), XBees and robots are used throughout the experimentation. Cameras help to setup the reference coordinate system for the robots and other objects. QUARC helps to process the data in real-time and maintains the communication between the systems in real time. Xbees are used for wireless communication between the simulink (PC) and the robots. The cameras captures the real-time position of the robot, which is then processed by simulink. The desired orientation and related velocities produced from the simulink is sent to the robot via Xbees. The robot then set its motion with the updated data from PC, completing the desired operation.

Chapter 3

Test of Robot Motion

3.1 Point to point Motion

To begin the maneuverability test of the mobile robot, the first task is moving the robot from point to point (P2P). As it has been discussed earlier, there are two factors which are required for motion of the robot: direction and driving velocity. In addition, the distance between the robot and the destination point is required as shown in figure 3.1.1, where (x_1, y_1) is the starting point, (x_2, y_2) is the destination point and θ is the inclination of the destination point with respect to starting point. For the purpose of the thesis, the gain K_p was set as 100 after a lot of tweaking, which provides enough initial throttle for the robot to move in the surface with friction and other motion related factors. Figure 3.1.2 shows the resultant motion of the robot.

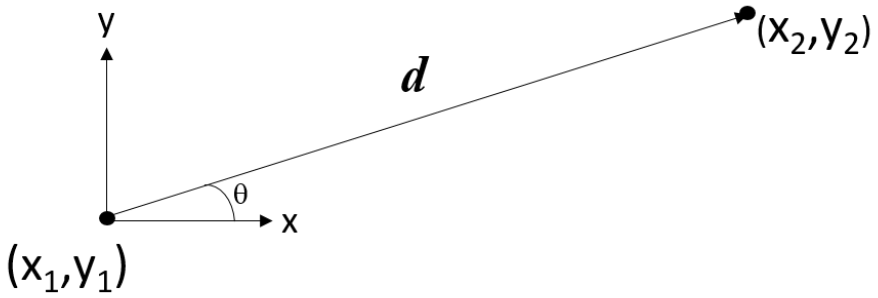


Figure 3.1.1: Point to point.

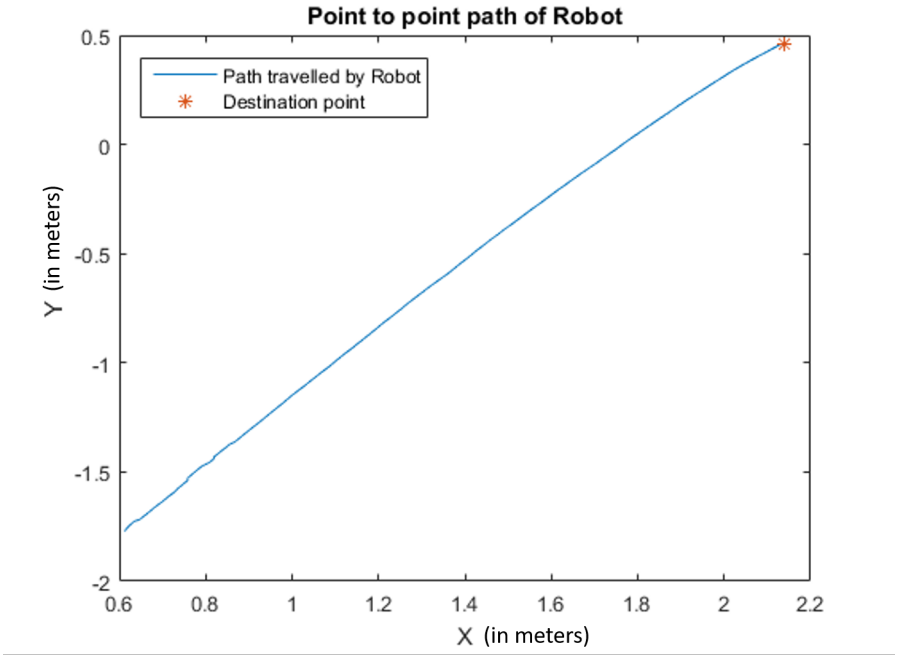


Figure 3.1.2: Robot path for point to point.

3.1.1 Control Logic

To perform a P2P task, the robot needs a constant velocity and the desired direction (θ) pointing towards the destination point. The direction is calculated as,

$$\theta = atan2d((y_2 - y_1), (x_2 - x_1)); \tag{3.1.1}$$

By making use of this function the orientation to any point in the four quadrant of the plane can be determined. The distance to the destination point is calculated using the distance formula.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.1.2}$$

With help of this formula, the velocity of the robot is decided. So long as the distance is greater than 0, velocity is set to high and as the robot is at a distance of less than 10 cm from the destination, the velocity is lowered to zero to avoid overshoot. In actual case, the maximum value of velocity with which the robot will not show any motion is 35, so any velocity less than 35 would stop the robot. Throughout the experiment, velocity of 50-70 was used to perform all the tasks. For collision avoidance with any object or another robot, a safe distance of 35 cm was maintained between them.

In practical situation, it is difficult to reach the destination point with zero error, i.e the robot would not reach the exact destination in general. To avoid overshoot, the speed of the robot is reduced as it reaches its goal. During experimentation, it was found that the closest point the robot will reach destination without overshooting is about 5 cm. If this distance is reduced, the robot loops around the destination point. This distance can be reduced to 0 by properly tuning the PID or by using any intelligent controller.

Additional shapes such as of square, polygon and the tasks such as pursue-evader, leader-follower are performed for the motion test of the robot. For the study purpose some tasks such as obstacle avoidance had fixed destination and obstacle. Other tasks such as, pursuer-evader have varying destination point.

3.2 Basic Shapes

Shape formation was used to test the maneuverability of the robot and to check the robustness of the controller. Basic shapes such as square, polygon and circle have edges and curves which are the motion which the robot undergoes in any situation.

3.2.1 Square and Polygon

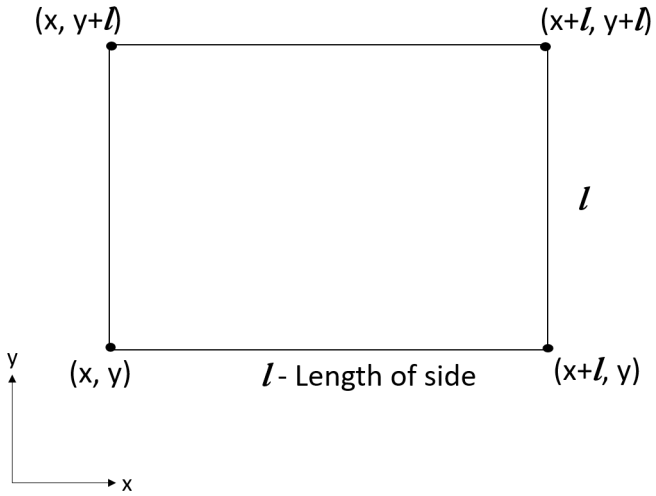
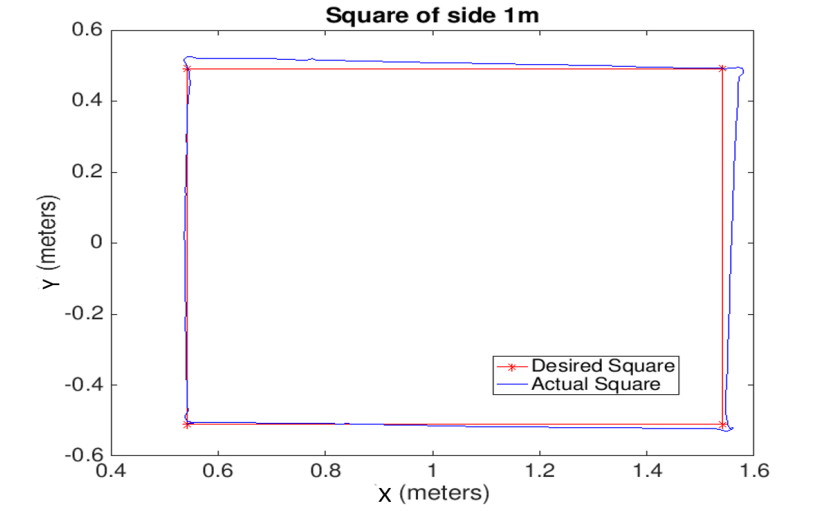


Figure 3.2.1: Motion parameters for drawing a square.

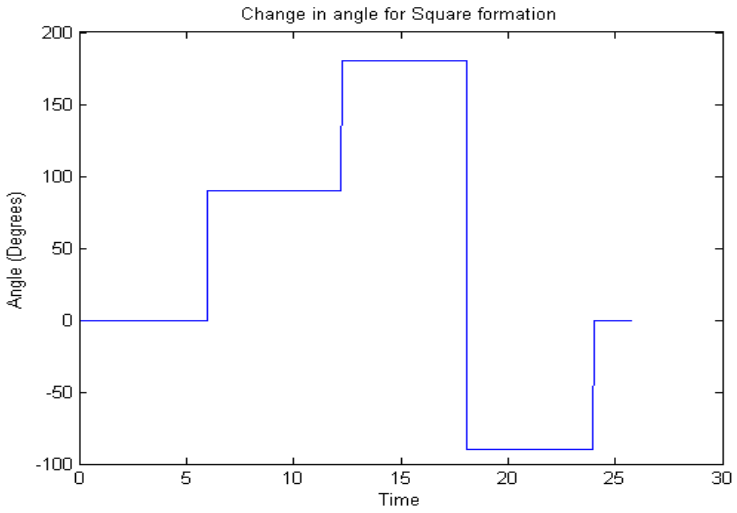
To form a simple square of any size, the length of sides and the starting position are required. Using this information, all the coordinates of vertices are calculated and stored in a matrix form. The coordinates of the vertices of the square is shown in the figure 3.2.1.

For the robot to form a square, it has to move through all the vertices in either counter-clockwise or clockwise direction. This can be done with the help of the P2P motion. When the robot reaches a vertex of the square, the coordinates of the next vertex in the queue is passed from the matrix created earlier. Repeating the sequence four times brings the robot back to a point near its starting position as shown in the figure 3.2.2a. It can be seen from the figure 3.2.2a, there are two squares. One

is the desired square (in red) calculated from the robot's current position, which is superimposed on the square formed (in blue) by the robot. This comparison shows the error in the controller while forming the shape. Figure 3.2.2 shows the resultant square formed by the robot along with the corresponding change in angles.



(a) Square formed by robot.



(b) Change in angles.

Figure 3.2.2: Sqaure formation and corresponding angle change.

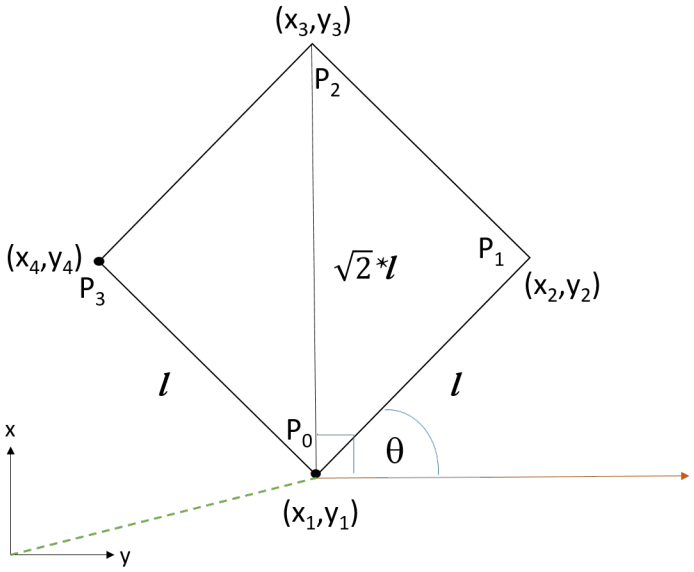
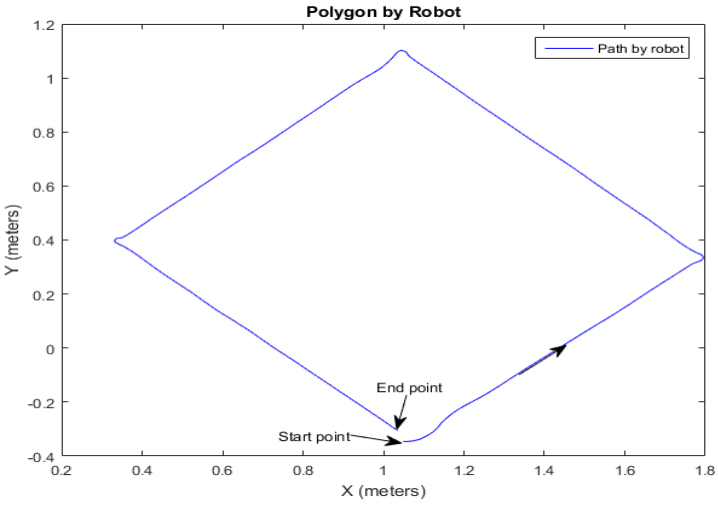


Figure 3.2.3: Motion parameters for drawing a polygon.

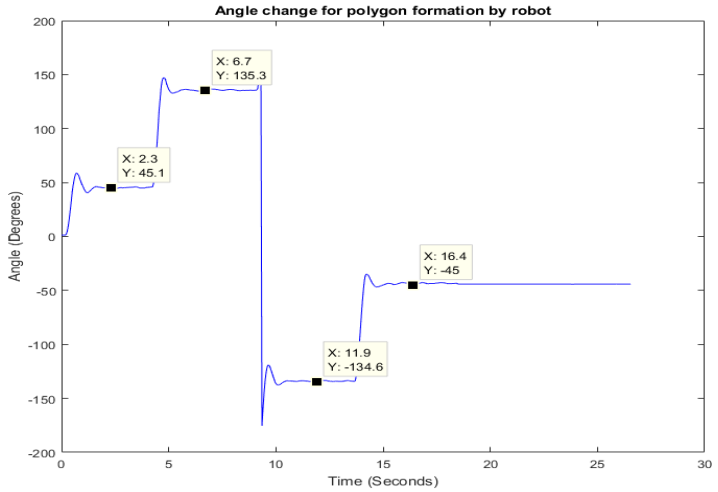
Figure 3.2.3 shows an polygon along with its angle of inclination with respect to the reference co-ordinate system. Robot first takes this orientation of θ and then continue forming the rest of the shape, moving through the matrix of vertices created beforehand. The vertices of this square are calculated using equation (3.2.1).

$$P_1(x, y) = (x_1 + d\cos(\theta), y_1 + d\sin(\theta)) \tag{3.2.1}$$

The equation (3.2.1) calculates the coordinates of any point located at a distance of d from the reference point (x_1, y_1) and at an angle of θ. From figure 3.2.3, we can see the vertex points as P_1, P_2 and P_3 with reference/ starting point of robot as P_0. Points P_1 and P_3 are distanced by length l from the reference point P_0, where as P_2 is at a distance of $\sqrt{2}l$. The side vertices are inclined at an angle of θ and $(90 + \theta)$ with respect to the starting point. The inclination of all the points can be seen in the figure 3.2.3. The polygon is formed in the same way as that of the simple straight square. Figure 3.2.4 shows the resultant polygon formed by robot with corresponding change in angles.



(a) Polygon formed by robot.



(b) Change in Angles.

Figure 3.2.4: Polygon formation and corresponding angle change.

3.2.2 Circle

Circle is another basic shape which was used to test the control of the robot. Unlike square, which has corners, circle represents working with turns and curves. For the robot to form circles, the work by Dudek and Jenkin about ICC -*Instantaneous Center of Curvature* [2] was used.

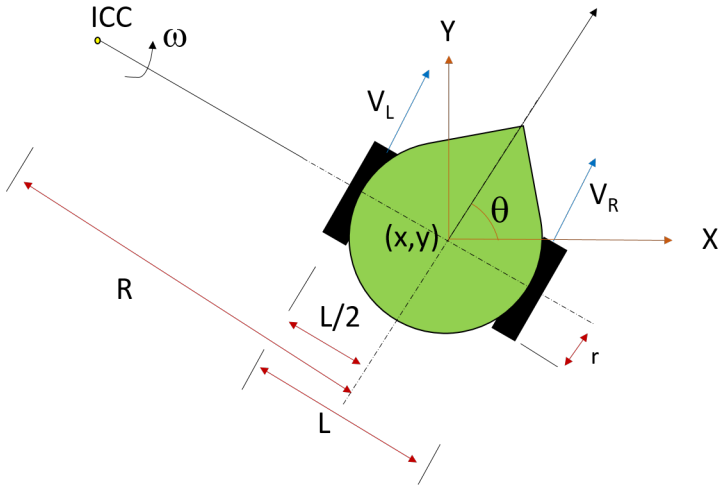


Figure 3.2.5: Concept of ICC.

The robot model shown in figure 3.2.5 is based on the work by Dudek and Jenkin [2] which focuses on ICC - *Instantaneous Center of Curvature*. ICC is the point around which the robot turns or it can be considered as the center around which the robot moves to form a circle. This point changes with the motion of the robot, hence called as an instantaneous point, true to that specific moment of time. Figure **??** shows the concept of ICC. From the figure, R is the distance of ICC from the center of the robot. ω is the angular velocity. When the robot is moving in straight line, R will be infinite and ω will be zero . These parameters will be defined when the robot takes any curvy path, that is, when the left and right wheels have different velocities. Equations 3.2.2 and 3.2.3 explains the calculation for V_r and V_l in terms of angular velocity and radius.

There are four cases explaining the relation between velocities and direction.

1. If $V_r = V_l$, robot will move straight in linear direction. R is infinite and ω will be zero.

2. If V_r = -V_l, robot will rotate in the same place along its axis about its center

point. So R is zero.

3. If $V_l = 0$, robot will turn about left wheel with the radius of $R = \frac{l}{2}$.

4. If $V_r = 0$, robot will turn about right wheel with a radius of $R = \frac{l}{2}$.

Kinematic equations for the mobile robot as per the ICC concept are,

$$V_r = \omega(R + \frac{L}{2}) \tag{3.2.2}$$

$$V_l = \omega(R - \frac{L}{2}) \tag{3.2.3}$$

Inversely,

$$\omega = \frac{V_r - V_l}{L} \tag{3.2.4}$$

$$R = \frac{L}{2}\frac{V_r + V_l}{V_r - V_l} \tag{3.2.5}$$

Using equations (3.2.5), (3.2.4), (3.2.2), (3.2.3) the individual wheel velocities to the robot, based on the required radius and angular velocity, were calculated. By controlling the angle of the robot, any arc or semi-circle can also be drawn,in the reference plane.

Figures 3.2.6 and 3.2.7 shows the resultant circle of 40 cm radius and corresponding angle change. The change in angle can be observed going from current angle of robot to 180 degrees and then from -180 degrees to 0 degrees.

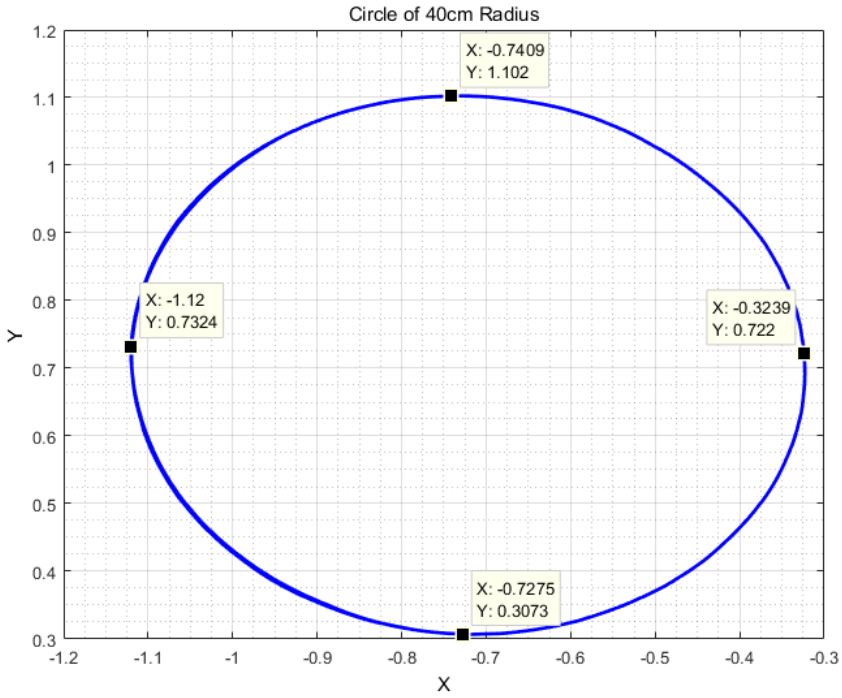


Figure 3.2.6: Circle drawn by the robot.

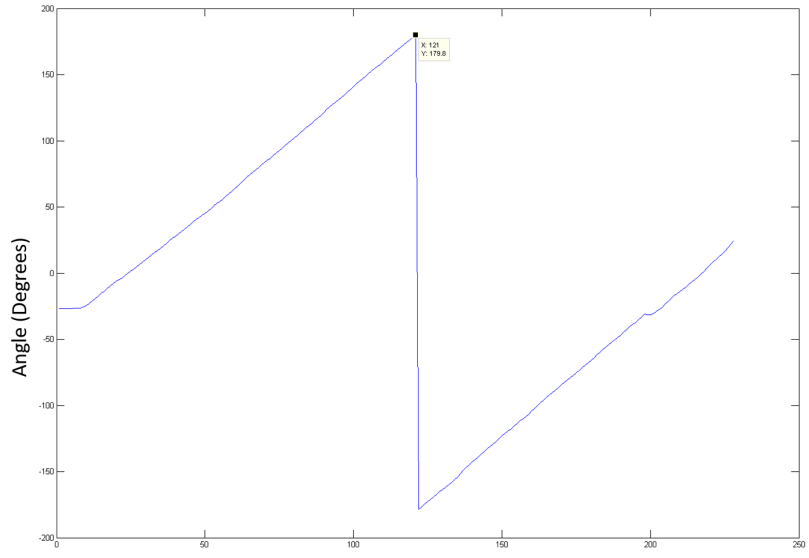


Figure 3.2.7: Angle variation for Circle

3.3 Summary

To test the maneuverability of the mobile robot and to check its controller, basic shapes of square, polygon and circle are fine test objects. Forming squares and polygon has its own challenge that they have edges with a specific angles. The position controller with gain of 100 was used to achieve the specific angles. For forming circle, different concept of Instantaneous Center of Curvature of implemented. This concept can also be used for the entire study of mobile robot.

Chapter 4

Obstacle Detection and Avoidance

In real world, robots may encounter many obstacles in its path. In this chapter, obstacle detection and path planning for avoiding them are discussed.

4.1 Setup

For experimental purpose, a plank of wood was used as an obstacle. The edges of this obstacle is tagged with markers and is created as two separate objects to be recognized by the cameras, namely Edge-1 and Edge-2. The position of the edges are known from the Vicon cameras, so the robot can detect them. Figure 4.1.1 shows the obstacle and different paths to be followed by robot.

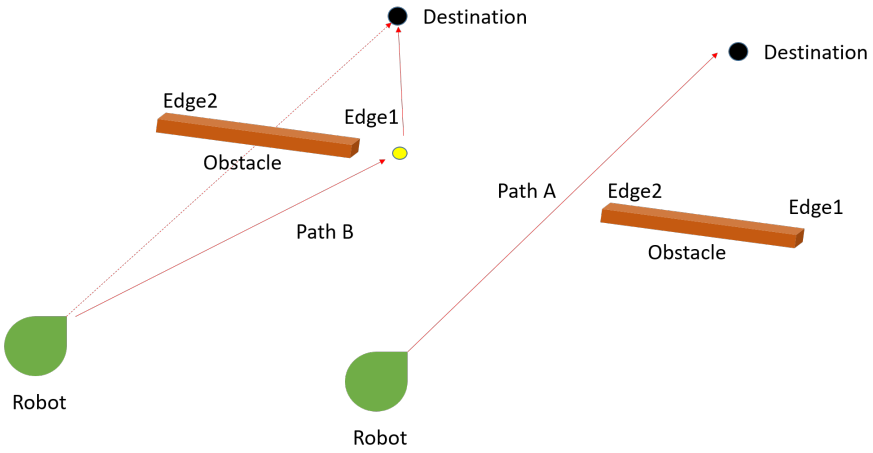


Figure 4.1.1: Path planning to avoid obstacle.

4.2 Path Planning

The position of all the objects are known from the cameras. The position of the obstacle is determined with respect to the robot and the destination point. If the obstacle lies in the path between robot and destination, the path for the robot is modified to avoid the obstacle. It can be seen from figure 4.1.1, that Path-A is a the direct path to the destination, i.e between robot and the destination, and Path-B is an alternate route avoiding the obstacle edges. In this thesis, the obstacle and destination are considered to be stationary. If the obstacle is detected in the Path-A, the robot will take Path-B to avoid the obstacle or else, it will continue with Path-A.

To check whether the obstacle is in the direct path of the robot, the orientation of the obstacle and destination with respect to the robot are calculated using *atan2* function, as shown in the figure 4.2.1. .

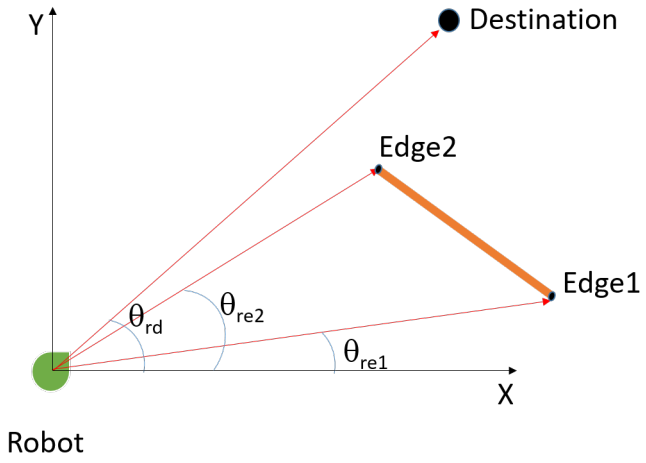


Figure 4.2.1: Orientation calculation.

To determine the exact path requirement, the angles between Robot-Edge1, Robot-Edge2 and Robot-Destination are calculated as θ_{re1}, θ_{re2} and θ_{rd} simultaneously, as shown in figure 4.2.1. To avoid the obstacle, the path to be taken by the robot is planned as described below.

1. If angle θ_{rd} is either less than or greater than both θ_{re1} and θ_{re2}, the obstacle is not in the direct path of robot to the destination, hence the the robot will take the route of Path-A as shown in figure4.2.2.

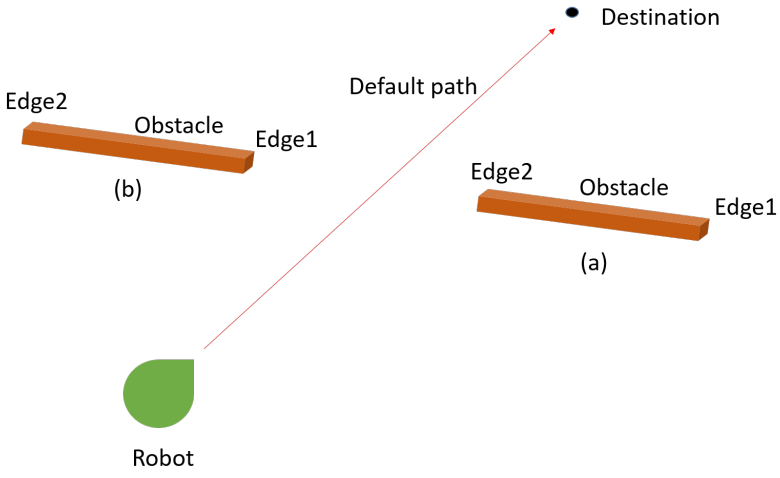


Figure 4.2.2: Default Path selection.

2. If angle θ_{rd} lies in between θ_{re1} and θ_{re2}, the obstacle lies in the direct path to the destination, i.e on Path-A. In this case, the robot takes Path-B to avoid the obstacle which can be seen in the figure 4.2.3. Here Path-B consists of paths 1 or path 2 via point P1 or P2, respectively.

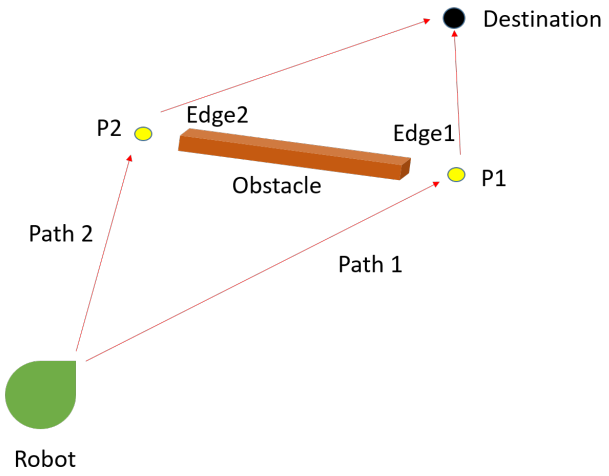


Figure 4.2.3: Path selection with obstacle.

There are two possible paths for the robot to reach its destination. Robot can either take Path-1 around Edge1 or it can take the Path-2 around Edge2. The algorithm for selection of the path between Path-1 and Path-2 looks for the shortest distance from the robot to the edges. After selecting the appropriate path , the algorithm for path planning would not do any more orientation check, i.e the path to be followed gets locked and the robot will follow the same path. This is equivalent to the GPS map system where alternate route is provided as soon as any obstruction is detected on the road, miles ahead.

For the robot to reach its destination while avoiding the obstacle edges, it goes through two sets of paths. First half is the path between robot and a point on the outer side of the edge, called a *Via points*. And the next half of the path is between this Via point and the destination. After deciding the path between Path-1 and Path-2, the robot would go to the respective Via point. The *Via points* are the points

falling in the equation of the line formed by Edge1 and Edge2, lying on the exterior of the obstacle. Point P1 and P2 in figure 4.2.3 are shown as the via points which are the approach points for the robot to go without colliding with the obstacle. The coordinates of the away points are calculated as

$$P_1(x, y) = (x_1 + d * \cos(\theta), y_1 + d * \sin(\theta)) \tag{4.2.1}$$

$$P_2(x, y) = (x_2 + d * \cos(180 + \theta), y_2 + d * \sin(180 + \theta)) \tag{4.2.2}$$

where, θ is the inclination of the line formed by Edge1 and Edge2, with respect to the reference coordinate system, (x_1, y_1) are the coordinates of Edge1 and (x_2, y_2) are the coordinates of Edge2. $P_1(x, y)$ and $P_2(x, y)$ are the respective via points of Edge1 and Edge2. d is the distance of P_1 or P_2 from their respective edge.

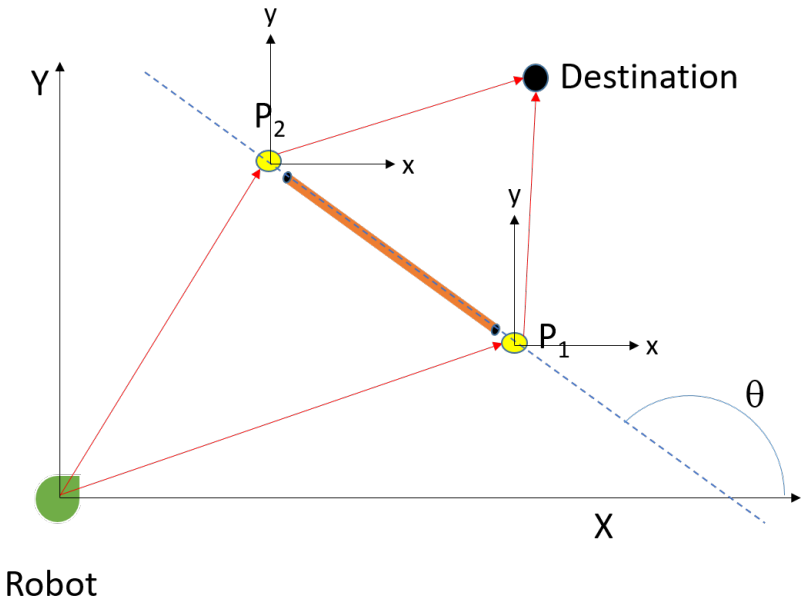


Figure 4.2.4: Calculation for via points.

Figure 4.2.4 shows the orientation of the via points for their coordinate calculation.

4.3 Result

Figure 4.3.1 shows the motion of the robot when there is no obstacle present between the destination and the robot. Whereas figure 4.3.2 shows the obstacle with the obstacle introduced in between the path of the robot. This figure also shows that the robot taking shortest path to destination while avoiding the obstacle.

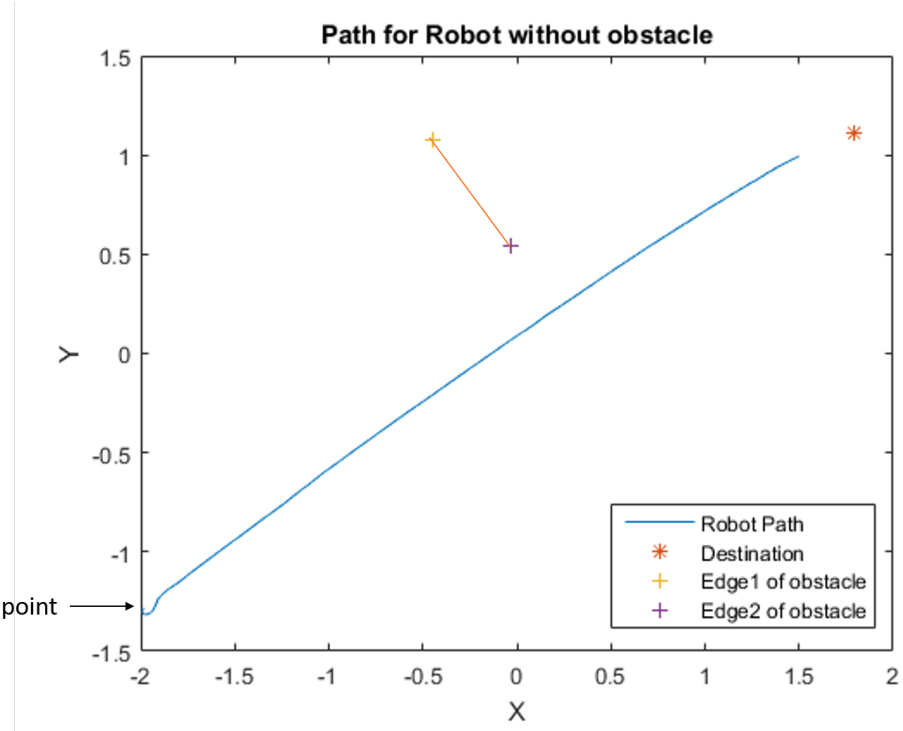


Figure 4.3.1: Path with no obstacle.

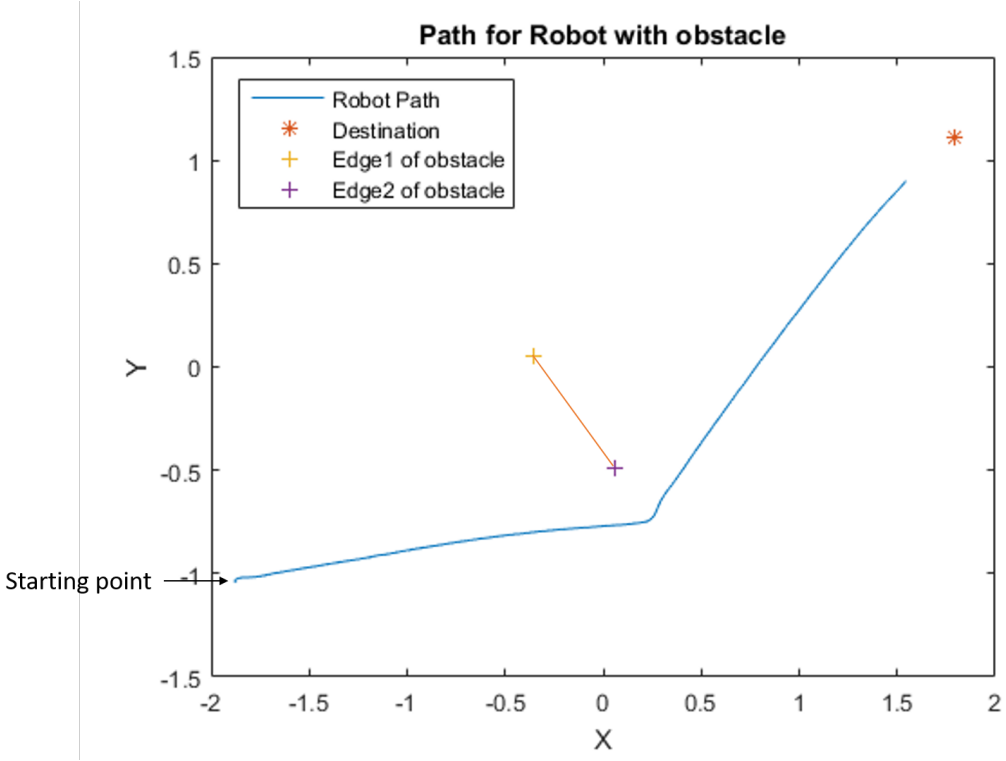


Figure 4.3.2: Path to avoid obstacle.

It can be seen from figures 4.3.1 and 4.3.2 the robot is not completely going to the destination because the destination was represented by another robot. To avoid collision, a fixed distance of 35 cm is maintained between both the robots. Refer appendix 7.C.

4.4 Summary

While path planning, the detection of the obstacles are always considered to avoid any collision. For path planning of the robot in such scenario, first, the obstacle has to be detected in the default path of the robot. This is done with the help of checking the orientation of the edges of the obstacles and the destination with respect to the robot. After the obstacle detected, the path with the shortest distance to the destination, while avoiding the edges, is calculated and then followed. This way the

robot detects the obstacle and selects a path to avoid it.

Chapter 5

Control of Multiple Robot

In this chapter, multiple robots are being introduced to undergo different scenarios such as pursuer-evader and leader-follower.

5.1 Pursuer-Evader

Pursuer-Evader is a classic example of airplane warfare where there is an evading plane and one pursuing plane. The pursuing fighter plane shoots a torpedo missile which follows it's target till it hits. In this scenario, there are two robots where one is pursuer and other has to evade. The objective of the pursuer robot is to catch up with the evading robot. The evader-robot is in continuous motion with changing position and following a path with a constant velocity. For the experiment, pursuer robot is traveling at a "higher" constant velocity than the evader, this is to complete the task within the workspace.

5.1.1 Setup and control

Both the robots are kept at different locations in the vicinity of the cameras with significant distance between them. The pursuer-robot is controlled in such a way that

it will follow the evader-robot wherever it may be within the workspace. So long as it is getting the positions of the evader-robot, the orientation of the pursuer would point in the direction of the evader. Using *atan2* function the orientation of the evader with respect to the pursuer can be calculated. This orientation is fed to pursuer-robot and it follows the evader-robot using this orientation. With the evader-robot in motion, the orientation between both the robots keeps on updating, providing new orientation to the pursuer. Thus, the pursuer will reach its destination (evader-robot), since it is moving at a higher velocity. The evader robot would stop when there is minimum distance (25 cm) between the both the robots.

5.1.2 Result

Figure 5.1.1 shows the path of both the robots moving in the workspace. The pursuer is starting from its initial position and it is getting updated orientations for the change in the position of the evader. To avoid collision, the pursuer robot was made to maintain a specific distance of 25 cm from the evader. Figure 5.1.2 shows the change in angle of the pursuer with respect to the evader. In this example, the evader is given a constant orientation of 45 degrees.

It can be seen from the figure that the orientation of the pursuer is changing and is tending towards that of the evader. Both the robots start from their initial orientation and achieve the desired direction. The constant orientation at the end shows the end of pursue and both the robots have stopped. The final orientation for both the robots may not be same as the approach direction of the pursuer can vary and the robots would stop at the moment when the distance between them is less than or equal to 25 cm. Refer appendix 7.D.

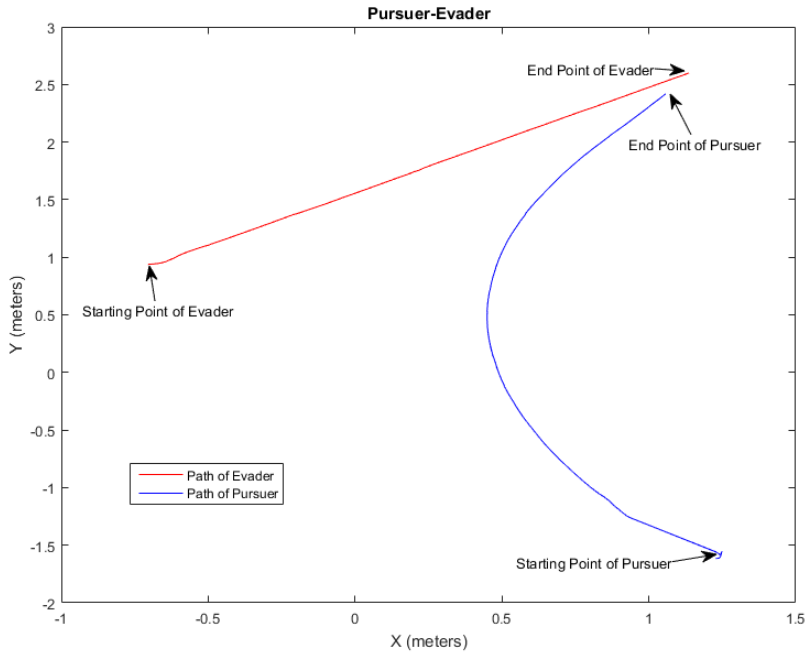


Figure 5.1.1: Positions of the Pursuer and Evader robot.

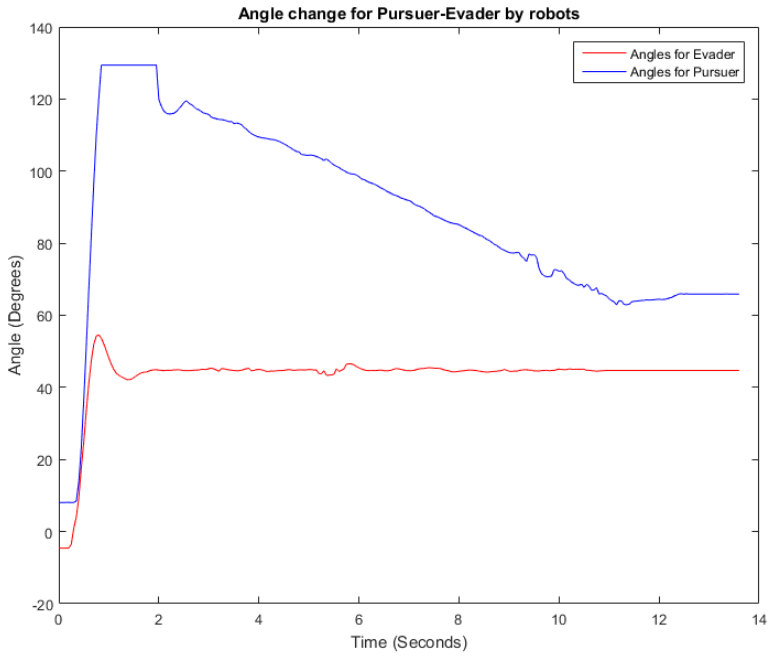


Figure 5.1.2: Angle for Pursuer and Evader robot.

5.2 Leader Follower

Formation control is another task for the multiple robots considered in this thesis. Such scenarios may consists of robot performing a similar task together or moving in a specific formation like any fleet of planes. In such situations, there is always a controlled leader and the rest of the robots are following or replicating the path of the leader with maintaining a specific distance from it. Three different types of leader-follower scenarios were performed in this task as explained below.

5.2.1 Case 1:

In this case the follower replicates the path of the leader from its current location. Consider that the leader and the follower are positioned away from each other. With any controlled path implemented on the leader, the follower would perform the same task by replicating the leader, but from its current position. The velocity and the orientation produced for the leader-robot is fed to the follower-robot. Figure 5.2.1 shows the data processed for leader-robot being replayed to follower-robot. Refer appendix 7.E.

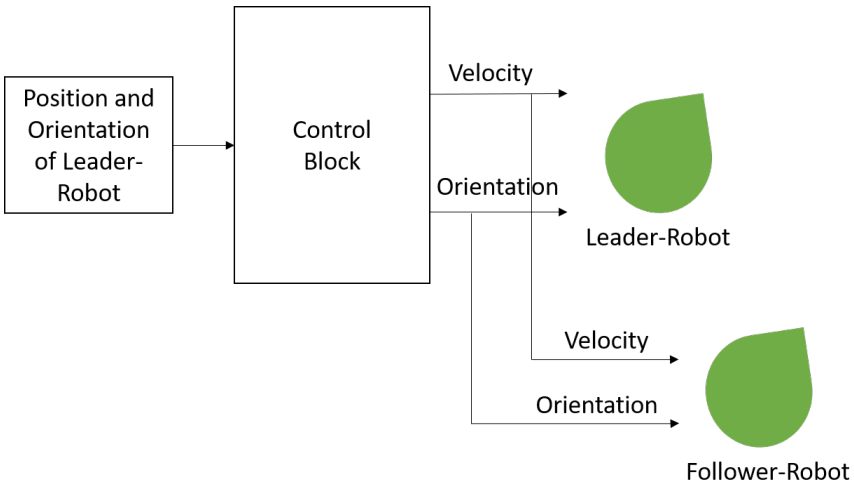


Figure 5.2.1: Flow of data.

The followers moves with the same velocity and orientation till it gets any new information on velocity and orientation. The real life application of this task can be to draw patterns or designs at different location of the workspace. Figure 5.2.2 shows the general idea about this scenario where it can be seen that both the robots are located at different locations but they perform the same task.

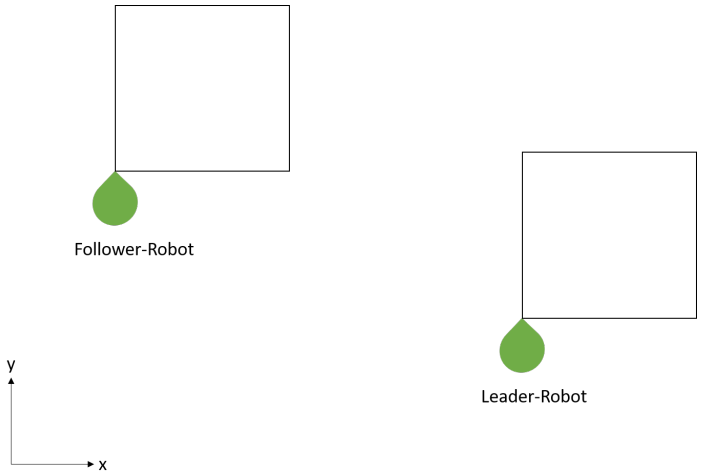


Figure 5.2.2: Leader-Follower case 1 scenario.

Figure 5.2.3 shows the leader forming a basic shape of square and the follower copying it. The leader is programmed to make a square, whereas the follower receives the required controlled velocity and orientation programmed for the leader. In this case the follower performs the task from it's current position.

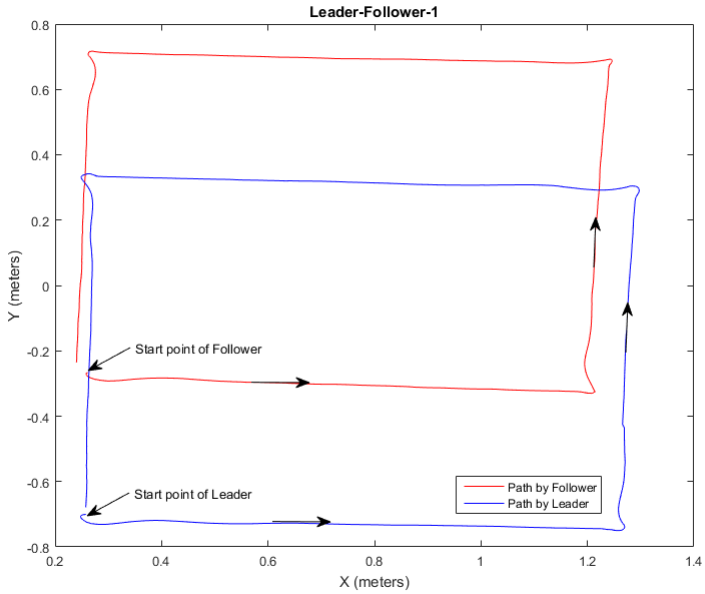


Figure 5.2.3: Leader-Follower without any maintained distance.

5.2.2 Case 2:

In this example, the follower would approach to a location near its leader (1 meter from the leader), before following the path of the leader. This case can be applied to a scenario where the robots are dispersed to begin with, and are required to regroup around the leader before doing any formation or following the path of leader. The initial velocity of the follower-robot is higher for approaching the leader. As soon as it reaches a specific distance from the leader, it moves with the same velocity and orientation as that of the leader. In this task the velocity and orientation of the leader is broadcast to all the follower-robots as soon as they reach near the leader. So irrespective of the motion status (stopped or moving) of the leader, the follower continues with the same velocity and orientation till it gets a new velocity and orientation. The case 2 scenario is shown in the figure 5.2.4. Refer appendix 7.F.

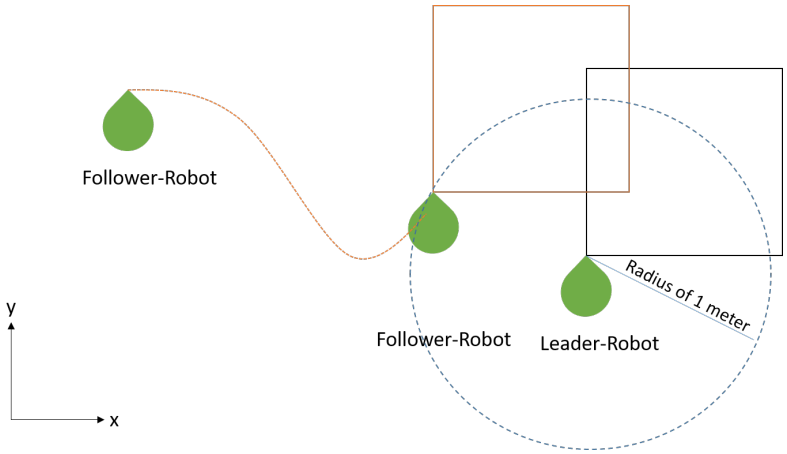


Figure 5.2.4: Leader-Follower case 2 scenario.

Figure 5.2.5 shows that the case where the follower has to maintain a specific distance of 1 meter from the leader. So, whatever may be the position of follower-robot in the plane, it will first reach a point at the specific distance away from the leader. After reaching that point, the follower then continues the path performed by the leader. Meanwhile the leader robot continues on its path.

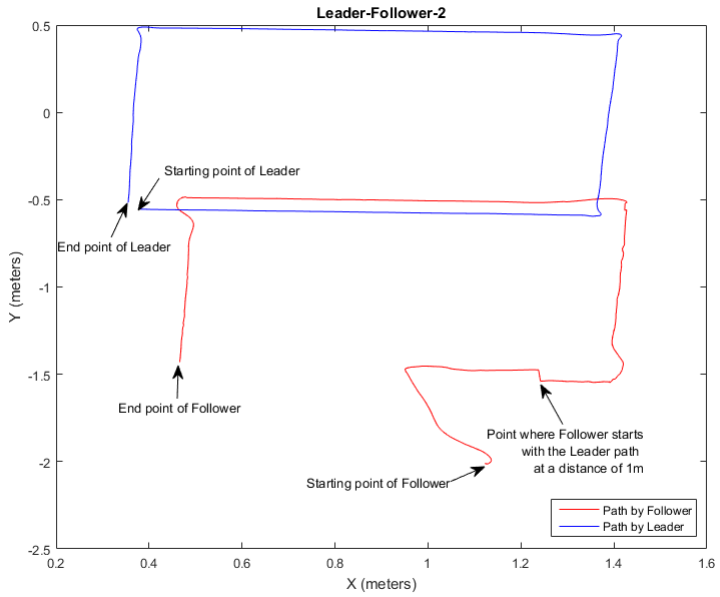


Figure 5.2.5: Leader-Follower with a maintained distance.

5.2.3 Case 3:

In case 3, the follower reaches a position near the leader. But this approach point for the follower-robot is an exact position near the leader, at a distance of 35 cm. The initial velocity for the follower-robot is set higher till it reaches the approach point. After reaching this point, the follower-robot moves with the same velocity as that of the leader and tries to maintain the position throughout the planned path for the leader. This means that the follower-robot follows the position of the leader. The major application of such case can be performing some specific formations, where multiple robots can be programmed to reach a specific position around their leader. Here, if the leader-robot stops, the follower-robot will also stop and the follower-robot moves only when the leader moves. Figure 5.2.6 shows the different possible approach points around leader for a definite formation. Refer appendix 7.G.

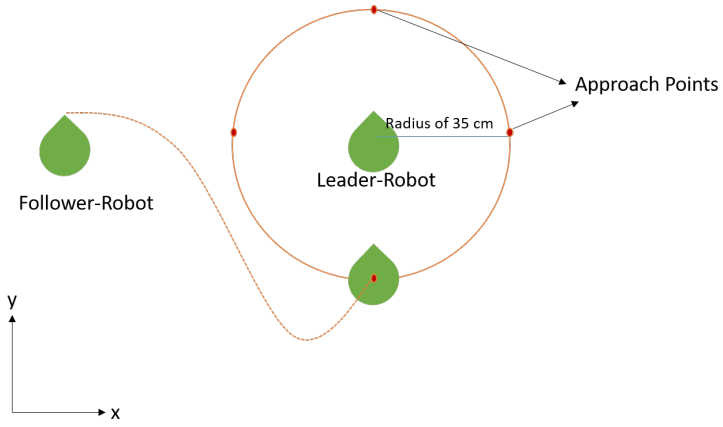


Figure 5.2.6: Leader-Follower case 3 scenario.

Figure 5.2.7 shows the case where the follower robot follows behind the leader at a distance of 35 cm. The leader forms a square but the follower doesn't make the square but just follows the leader's position. It can be observed that the follower robot is not taking sharp turns (or making square) as the leader does, this is because the follower is going to the position it is programmed to which is behind the leader.

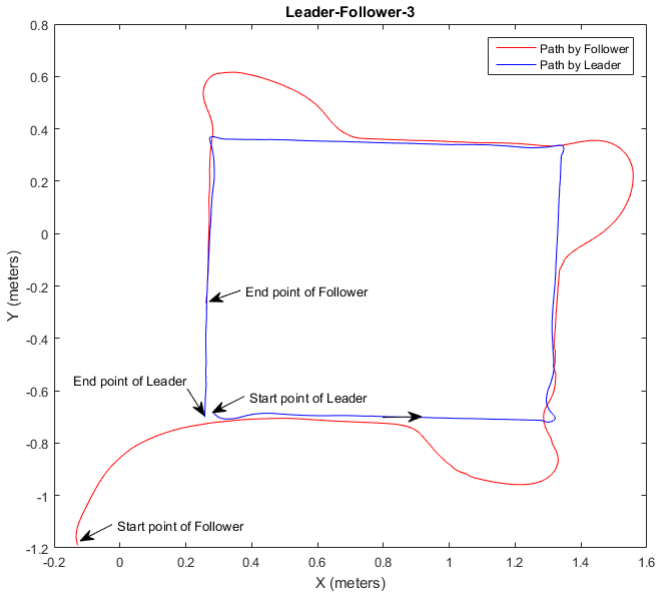


Figure 5.2.7: Leader-Follower with a maintained position.

5.3 Summary

This chapter deals with the control of multiple robots performing tasks such as Pursue-Evader and Leader-Follower. In the task of Pursuer-Evader, the objective of the purser robot is to catch up with the evader-robot. A specific path is designed for the evader and the pursuer-robot follows the position of the evading robot by changing its orientation with respect to the evader-robot. Both the robots stop only when the pursuer reaches a position near to the evader hence catching up with the evader-robot.

In the scenario of Leader-Follower, there are three different cases. In Case 1, the follower-robot replicates the planned path of the leader-robot from its current location. In Case 2 the follower-robot approaches a point near the leader and then continues with the path planned for the leader. In Case 3, the follower-robot reaches an exact point near the leader and follows the position of the leader. All the tasks were performed with two robots and the results are shown.

Chapter 6

GUI Controlled Robot

6.1 GUI Concept

GUI, *Graphical User Interface*, is a front end software within Matlab which gives user the option to pass data or parameters to the program. It is a convenient way to change the input without editing the program when dealing with a long and complex Matlab code. In this thesis, the robot motion is controlled for various tasks from making a point to point motion to Leader-Follower. To make the path planning of robot more flexible, a GUI in Matlab was designed for user to select a point or points in the workspace. The reuslts from the GUI is later sent to Simulink for further data processing. Major application of this would be to draw logos or create design using robot.

6.2 Control Logic

The integration of different systems and their work flow is shown in figure 6.2.1.

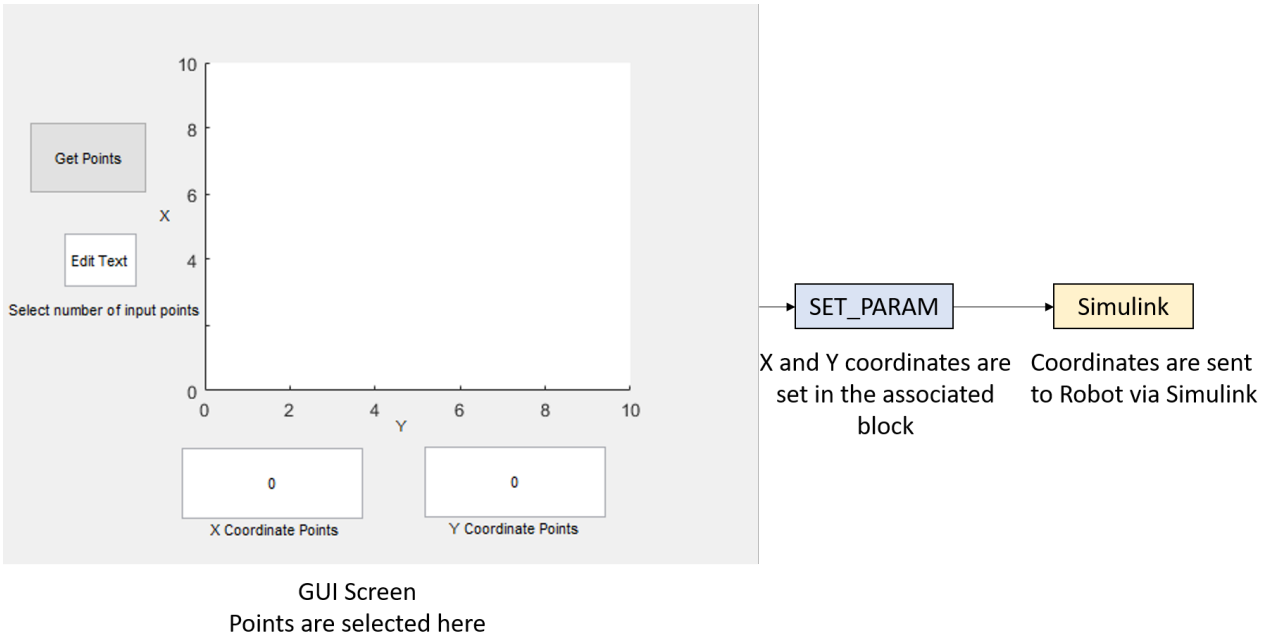


Figure 6.2.1: GUI flow for data communication.

6.2.1 Selecting Points

Using a predefined function of GUI in Matlab, any parameter required can be accessed, passed and processed. All the action blocks such as push button and the display screen generate a call back function which handles the data for that particular action block. The application of each action button is defined in this call-back section. Figure 6.2.1 shows the GUI screen designed for this thesis.

When the "Get Points' button is pressed, the cross lines marker appear on the screen. This marker can now be used to select any point on the X-Y plane. The "Display Screen' below the axes shows the selected points. For multiple points, the editable screen below "Get Points' button is used to input the number of points required. All the points selected are displayed in the "Display Screen', where they are displayed and saved as x and y coordinates separately in an array. To make sure that the points selected are lying in the workspace, the axes limit was set to select any point within the surface area of 2 x 2 meters of the plane.

6.2.2 Data handling

In Matlab, there is a function called *SET-PARAM* which sets the parameter of any block within simulink or program. Thus, any parameter value of the associated block can be changed as per requirement. While experimenting with the robot, two blocks with constant values were used in simulink, each for x and y corordinate. With the help of *SET-PARAM*, the output points of the GUI screen were written inside the associated constant block.

The coordinates transmitted to simulink are then processed according to the function to be performed and sent to the robot to perform the desired task. Multiple coordinates are sent sequentially to simulink, which it processes to move the robot to each point in the order it is received. So, the data for multiple points works in FIFO manner.

6.3 Test and Results

To test the GUI created, random points were selected and can be seen in figure 6.3.1 as dots. The position of these points are registered in reference to the working plane, which is the reference plane, and then are sent to robot. The robot starts from it's current position, goes to the first point, follows through all other points and stops at the last point as shown in figure 6.3.1. With more intelligent coding for handling the points, a whole logo and words can be formed. Multiple robots can also be controlled with GUI to perform complex tasks.

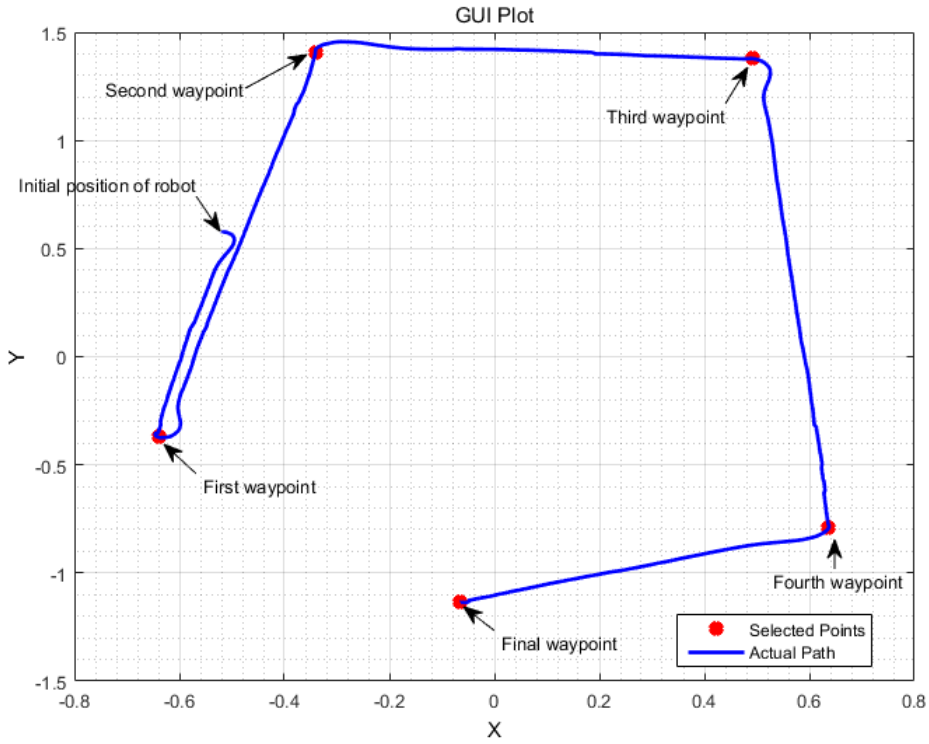


Figure 6.3.1: Gui Result.

6.4 Summary

Graphical User Interface (GUI) for the robot is a unique method for its path planning. By touching the required points on the user screen, specific points or a whole design can be selected, which then can be performed by the robot. This system has a major advantage of providing the flexibility for the control of robot.

Chapter 7

Summary and Conclusion

Path planning and control of a differential drive robot is studied and implemented in this thesis. The model by Snider[1] and the Dudek model[2] on Instantaneous Center of Curvature are also discussed. The simulation of the mathematical model of the mobile robot was carried out in simulink. The design of the controller was also carried out and was later implemented on actual robot and tweaked for better results.

The *atan2* function plays an important role in this thesis. The required orientation of the robot, anywhere in the plane, can be calculated using *atan2* function which gives the direction/orientation towards the destination with the precise quadrant assignment. By controlling the velocity and orientation any position in the workspace can be reached accurately. The *atan2* function was used for driving the robot to a point, drawing shapes and pursuing other moving object.

Many systems such as Vicon camera system, QUARC- real time controller, Xbees and Redbots were used to perform the experimentation. All these systems were configured for real-time data communication. Vicon camera system is similar to a GPS which sends the real-time positions of the robot and other objects in the workspace to the PC, the control block in simulink processes the data and generates new velocity for the robot wheels. These velocities are sent to the robot via pair of

Xbees which, in turn moves the robot to its new location.

Different controllers such as PID were tested, but a simple proportional controller provided as good results as more complicated controllers. Tasks such as point to point motion and shape formation were performed to test the robustness of the controller and the results were used to optimize the controller for better performance. Other tasks such as obstacle avoidance, leader-follower and pursuer-evader were performed to test the controller. Obstacle detection and avoidance task helped to plan the path of the robot under certain scenario. The tasks such as leader-follower, pursuer-evader involved multiple robots and the coordination between them. A GUI software was created to increase the flexibility of path planning and application of the robot. Certain way-points can be assigned for the robot to follow through with the help of GUI.

However, there were errors in the results when taking sharp turns such as in forming a square. In future, more advanced controller can be developed and implemented to obtain better results. Also more complicated tasks can be designed and performed to test the application of the robot. The software with GUI can be developed with more features that gives user a better control of the robot.

Bibliography

[1] Cory. F.Snyder, *An Intuitive approach to formation Control of Differential Drive Robots* . Wright State University, 2012.

[2] Gregory Dudek and Michael Jenkin, *Computational Principles of Mobile Robotics*, *Second Edition* , Cambridge University Press, 2010.

[3] P. Petrov and L. Dimitrov , *Nonlinear path Control for a Differential Drive Mobile Robot*, *Recent*, Vol. 11, no. 1, pp. 41-45, 2010.

[4] Kinematics

`https://en.wikipedia.org/wiki/Kinematics`

[5] Vicon Motion Capture System

`https://www.vicon.com/what-is-motion-capture`

[6] ZIGBEE® WIRELESS STANDARD

`http://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wirele`

[7] QUARC® Real-Time Control Software,

`http://www.quanser.com/products/QUARC`

[8] Sparkfun Redbot,

`https://learn.sparkfun.com/tutorials/getting-started-with-the-redbot`

[9] atan2 function,

https://en.wikipedia.org/wiki/Atan2

[10] Robotics

https://en.wikipedia.org/wiki/Robotics

[11] atan2 angle Image By Dmcq - Own work, CC BY-SA 3.0,

https://commons.wikimedia.org/w/index.php?curid=9381369

Appendix

Appendix 7.A Square

```
1  function [vl1, thetal] = ...
       leader(vl,d,xl,yl,theta,theta_l)
2  %vl = desired constant velocity.
3  %d = length of side of shape.
4  %xl = current x-position of robot.
5  %yl = current y-position of robot.
6  %theta = initial inclination of the shape.
7  %theta_l = current orientation of robot.
8  %thetal = calculated orientation of robot.
9  %vl1 = desired velocity for the robot.
10
11 persistent i  x0  y0   %To store the initial values ...
       of the positions
12 if isempty(i)||isempty(x0)||isempty(y0)
13     i = 1;
14     x0 = xl;
```

```
15      y0 = yl;
16  end
17
18   thetal = t1;
19   vl1 = vl;
20
21  %%%%%%%For Square theta is 0, calculation of the ...
       square vertices
22
23  x1 = x0+d*cosd(theta);
24  y1 = y0+d*sind(theta);
25  x2 = x0+1.4142*d*cosd(45+theta);
26  y2 = y0+1.4142*d*sind(45+theta);
27  x3 = x0+d*cosd(theta+90);
28  y3 = y0+d*sind(theta+90);
29
30  %%%Matrix of coordinates of the vertices
31   x = [x0 x1 x2 x3 x0 x0]
32   y = [y0 y1 y2 y3 y0 y0]
33
34  %%%%%%%% Calculation of angles of the square %%%%%%%%
35
36  ang1 = atan2d(y1-y0,x1-x0);
37  ang2 = atan2d(y2-y1,x2-x1);
38  ang3 = atan2d(y3-y2,x3-x2);
39  ang4 = atan2d(y0-y3,x0-x3);
```

```
%%%Matrix of angles for robot
ang = [ang1 ang2 ang3 ang4 ang4];
 j = i;


if i<5
    if (sqrt((x(j+1)-xl)^2+(y(j+1)-yl)^2))> 0.07    ...
       %Checking for robot within 7cm from the ...
       destination point
        thetal = ang(j);
        vl1 = vl;    %assignment of the velocity
        it = i;
    else
        thetal = ang(j+1);
        vl1 = 0;
         if theta_l ≤ ang(j+1)+ 10 && theta_l ≥ ...
            ang(j+1)- 10 %checking if robot reached ...
            the desired orientation
            i = i+1;
        end

        it = i;
    end
else
    vl1 = 0;
    thetal = ang(j);
```

```
62      it = i;
63  end
64  x_0 = x0;
65  y_0 = y0;
66
67  end
68
69
70  end
```

Appendix 7.B Polygon

```
1  function [vl1, thetal] = ...
       leader(vl,d,xl,yl,theta,theta_l)
2
3  %vl = desired constant velocity.
4  %d = length of side of shape.
5  %xl = current x-position of robot.
6  %yl = current y-position of robot.
7  %theta = initial inclination of the shape.
8  %theta_l = current orientation of robot.
9  %thetal = calculated orientation of robot.
10 %vl1 = desired velocity for the robot.
11
```

```
12 persistent i  x0  y0 %To store the initial values of ...
     the positions
13 if isempty(i)||isempty(x0)||isempty(y0)
14     i = 1;
15     x0 = xl;
16     y0 = yl;
17 end
18
19  thetal = t1;
20  vl1 = vl;
21
22 %%%%%%%For inclined square theta is 45, calculation ...
     of vertices of square
23
24 x1 = x0+d*cosd(theta);
25 y1 = y0+d*sind(theta);
26 x2 = x0+1.4142*d*cosd(45+theta);
27 y2 = y0+1.4142*d*sind(45+theta);
28 x3 = x0+d*cosd(theta+90);
29 y3 = y0+d*sind(theta+90);
30
31 %%%Matrix of coordinates of the vertices
32  x = [x0 x1 x2 x3 x0 x0]
33  y = [y0 y1 y2 y3 y0 y0]
34
35 %%%%%%%% Calculation of angles of the square %%%%%%%%
```

```
36
37 ang1 = atan2d(y1-y0,x1-x0);
38 ang2 = atan2d(y2-y1,x2-x1);
39 ang3 = atan2d(y3-y2,x3-x2);
40 ang4 = atan2d(y0-y3,x0-x3);
41
42 %%%Matrix of angles for robot
43 ang = [ang1 ang2 ang3 ang4 ang4];
44  j = i;
45
46
47 if i<5
48     if (sqrt((x(j+1)-xl)^2+(y(j+1)-yl)^2))> 0.07    ...
       %Checking for robot within 7cm from the ...
       destination point
49         thetal = ang(j);
50         vl1 = vl;
51         it = i;
52     else
53         thetal = ang(j+1);
54         vl1 = 0;
55          if theta_l ≤ ang(j+1)+ 10 && theta_l ≥ ...
              ang(j+1)- 10 %checking if robot reached ...
              the desired orientation
56              i = i+1;
57         end
```

```
58
59          it = i;
60      end
61  else
62      vl1 = 0;
63      thetal = ang(j);
64      it = i;
65  end
66  x_0 = x0;
67  y_0 = y0;
68
69  end
70
71
72  end
```

Appendix 7.C Obstacle Detection and Avoidance

```
1
2  function ...
       [v1,theta1,theta_r_d,theta_r_e1,theta_r_p1,theta_e12, ...
       x, x_p2, y_p2]  = avoidance(xr0,yr0,xd,yd,xe10, ...
       ye10,xe20,ye20,v, theta)
3
4  %xr0 = current x-position of robot.
```

```
5  %vr0 = current y-position of robot.
6  %xd = current x-position of destination.
7  %yd = current y-position of destination.
8  %xe10 = current x-position of edge-1.
9  %ye10 = current y-position of edge-1.
10 %xe20 = current x-position of edge-2.
11 %ye20 = current y-position of edge-2.
12 %v = desired velocity for the robot.
13 %theta = actual orientation of the robot.
14
15
16 %%% orientation of robot towards destination
17 theta_r_d =  (atan2d((yd-yr0),(xd-xr0)));
18
19 %%% orientation of robot towards edge 1
20 theta_r_e1 = (atan2d((ye10-yr0),(xe10-xr0)));
21
22 %%% orientation of edge 1 towards edge2
23 theta_e12 = atan2d((ye20-ye10),(xe20-xe10));
24
25 %%% Point P1 away from edge 1
26 d = 0.35;
27 y_p1 = ye10 + d*sind(180 + theta_e12);
28 x_p1 = xe10 + d*cosd(180 + theta_e12);
29
30 %%% Point P2 away from edge 2
```

```
y_p2 = ye20 + d*sind(theta_e12);
x_p2 = xe20 + d*cosd(theta_e12);

%%% orientation of robot towards P1
theta_r_p1 = (atan2d((y_p1-yr0),(x_p1-xr0)));

%%% orientation of robot towards P2
theta_r_p2 = (atan2d((y_p2-yr0),(x_p2-xr0)));

persistent theta_p1 theta_p2  %To store the initial ...
   values of the positions
if isempty(theta_p1) || isempty(theta_p2)
    theta_p1 = theta_r_p1;
    theta_p2 = theta_r_p2;
end

%Logic for obstacle detection and path determination
if ((theta_r_d>theta_p1)&& (theta_r_d>theta_p2)) || ...
   ((theta_r_d<theta_p1)&& (theta_r_d<theta_p2))
%Determination of the position of obstacle

    if (sqrt((xd-xr0)^2+(yd-yr0)^2))>0.35
        x = 1;
    else
        x = 3;
    end
```

```
55 else                                  %Path B
56     if(sqrt((x_p2-xr0)^2+(y_p2-yr0)^2))<(sqrt((x_p1-xr0)^2+(y_p1-y
57
58         if (sqrt((x_p2-xr0)^2+(y_p2-yr0)^2))>0.05
59             x = 2;
60         else
61             x = 1;
62         end
63     else
64         if (sqrt((x_p1-xr0)^2+(y_p1-yr0)^2))>0.05
65             x = 4;
66         else
67             x = 1;
68         end
69     end
70 end
71
72     switch x                  %Path options for the robot
73         case 1
74             theta1 = theta_r_d;
75             v1 = v;
76         case 2
77             theta1 = theta_p2;
78             v1 = v;
79         case 3
80             v1 = 0;
```

```
81              theta1 = theta_r_d;
82          case 4
83              v1 = v;
84              theta1 = theta_p1;
85          otherwise
86              v1 = 0;
87              theta1 = 0;
88      end
89  end
```

Appendix 7.D Pursuer-Evader

```
1  function [theta_b, va, vb]  = persuer(xa,ya, xb, yb, ...
       va1, vb1)
2
3  %xa = x-position of evader robot
4  %ya = y-position of evader robot.
5  %xb = x-position of pursuer robot
6  %yb = y-position of pursuer robot.
7  %va1 = velocity for evader robot.
8  %vb1 = velocity for pursuer robot.
9
10 if sqrt((ya-yb)^2+(xa-xb)^2) > 0.25     %for ...
       maintaining distance of 25cm between the robots
```

```
11      theta_b = (atan2d((ya-yb),(xa-xb))); ...
            %orientation of evader with respect to pursuer.
12      va = va1;
13      vb = vb1;
14  else
15      theta_b = (atan2d((ya-yb),(xa-xb)));
16      va = 0;
17      vb = 0;
18  end
19
20  end
```

Appendix 7.E Leader-Follower Case1

```
1  function [vl1,vf1,thetaf, thetal] = leader(xf,yf,vl, ...
       vf,d,xl,yl,theta, theta_l)
2
3  %xf = x-position of follower robot.
4  %yf = y-position of follower robot.
5  %xl = x-position of leader robot
6  %yl = y-position of leader robot.
7  %vl = velocity for leader robot.
8  %d = length of square.
9  %vfl = output velocity for follower robot.
10 %theta = inclination of the shape.
```

```
%theta_l = actual orientation of the leader robot.
%thetaf = output orientation for follower robot.
%vl1 = output velocity for leader
thetal = output velocity for leader.

%%%%%%% Logic for leader %%%%

persistent i  x0  y0

if isempty(i)||isempty(x0)||isempty(y0)
    i = 1;
    x0 = xl;%x_2%
    y0 = yl;%y_2%
end

 thetal = t1;
 vl1 = vl;

 %%%%%%%For Square, theta is 0

x1 = x0+d*cosd(theta);
y1 = y0+d*sind(theta);
x2 = x0+1.4142*d*cosd(45+theta);
y2 = y0+1.4142*d*sind(45+theta);
x3 = x0+d*cosd(theta+90);
y3 = y0+d*sind(theta+90);
```

```
%
%
 x = [x0 x1 x2 x3 x0 x0]
 y = [y0 y1 y2 y3 y0 y0]

%%%%%%%% Calculation of angles of the square %%%%%%%%

ang1 = atan2d(y1-y0,x1-x0);
ang2 = atan2d(y2-y1,x2-x1);
ang3 = atan2d(y3-y2,x3-x2);
ang4 = atan2d(y0-y3,x0-x3);

ang = [ang1 ang2 ang3 ang4 ang4];
 j = i;


if i<5
    if (sqrt((x(j+1)-xl)^2+(y(j+1)-yl)^2))> 0.07    ...
       %Checking for robot within 7cm from the ...
       destination point
         thetal = ang(j);
         vl1 = vl;
         it = i;
    else
         thetal = ang(j+1);
         vl1 = 0;
```

```
61            if theta_l ≤ ang(j+1)+ 10 && theta_l ≥ ...
                 ang(j+1)- 10
62                i = i+1;
63            end
64
65            it = i;
66        end
67    else
68        vl1 = 0;
69        thetal = ang(j);
70        it = i;
71    end
72
73    %%%%%%% Logic for Follower %%%%%%%%%%%%%%%%%%
74
75
76    thetaf = ang(j);
77    vf1 = vl1;
78
79    x_0 = x0;
80    y_0 = y0;
81
82    end
```

Appendix 7.F Leader-Follower Case2

```
function [vl1,vf1,thetaf, thetal] = leader(xf,yf,vl, ...
    vf,d,xl,yl,theta, theta_l)

%xf = x-position of follower robot.
%yf = y-position of follower robot.
%xl = x-position of leader robot
%yl = y-position of leader robot.
%vl = velocity for leader robot.
%d = length of square.
%vfl = output velocity for follower robot.
%theta = inclination of the shape.
%theta_l = actual orientation of the leader robot.
%thetaf = output orientation for follower robot.
%vl1 = output velocity for leader
thetal = output velocity for leader.

%%%%%%% Logic for leader %%%%

persistent i  x0  y0

if isempty(i)||isempty(x0)||isempty(y0)
    i = 1;
    x0 = xl;%x_2%
    y0 = yl;%y_2%
end

```

```
 thetal = t1;
 vl1 = vl;

 %%%%%%%For Square, theta is 0

x1 = x0+d*cosd(theta);
y1 = y0+d*sind(theta);
x2 = x0+1.4142*d*cosd(45+theta);
y2 = y0+1.4142*d*sind(45+theta);
x3 = x0+d*cosd(theta+90);
y3 = y0+d*sind(theta+90);
%
%
 x = [x0 x1 x2 x3 x0 x0]
 y = [y0 y1 y2 y3 y0 y0]

%%%%%%%% Calculation of angles of the square %%%%%%%%

ang1 = atan2d(y1-y0,x1-x0);
ang2 = atan2d(y2-y1,x2-x1);
ang3 = atan2d(y3-y2,x3-x2);
ang4 = atan2d(y0-y3,x0-x3);

ang = [ang1 ang2 ang3 ang4 ang4];
 j = i;

```

```
52
53 if i<5
54     if (sqrt((x(j+1)-xl)^2+(y(j+1)-yl)^2))> 0.07    ...
          %Checking for robot within 7cm from the ...
          destination point
55          thetal = ang(j);
56          vl1 = vl;
57          it = i;
58     else
59          thetal = ang(j+1);
60          vl1 = 0;
61           if theta_l ≤ ang(j+1)+ 10 && theta_l ≥ ...
                ang(j+1)- 10
62               i = i+1;
63          end
64
65          it = i;
66     end
67 else
68     vl1 = 0;
69     thetal = ang(j);
70     it = i;
71 end
72
73 %%%%%%% Logic for Follower %%%%%%%%%%%%%%%%%%
74
```

```
75
76 if (sqrt((xl-xf)^2+(yl-yf)^2))> 1 % distance of the ...
      follower would be 1m from leader.
77     thetaf = atan2d((yl-yf),(xl-xf));
78     vf1 = 80;
79 else
80     thetaf = ang(j);
81     vf1 = vl1;
82 end
83 x_0 = x0;
84 y_0 = y0;
85
86 end
87
88
89 end
```

Appendix 7.G Leader-Follower Case3

```
1 function [vl1,vf1,thetaf, thetal] = leader(xf,yf,vl, ...
     vf,d,xl,yl,theta, theta_l)
2
3 %xf = x-position of follower robot.
4 %yf = y-position of follower robot.
5 %xl = x-position of leader robot
```

```
6  %yl = y-position of leader robot.
7  %vl = velocity for leader robot.
8  %d = length of square.
9  %vfl = output velocity for follower robot.
10 %theta = inclination of the shape.
11 %theta_l = actual orientation of the leader robot.
12 %thetaf = output orientation for follower robot.
13 %vl1 = output velocity for leader
14 thetal = output velocity for leader.
15
16
17 %%%%%%% Logic for leader %%%%
18
19 persistent i  x0  y0
20
21 if isempty(i)||isempty(x0)||isempty(y0)
22     i = 1;
23     x0 = xl;%x_2%
24     y0 = yl;%y_2%
25 end
26
27  thetal = t1;
28  vl1 = vl;
29
30  %%%%%%%For Square, theta is 0
31
```

```
x1 = x0+d*cosd(theta);
y1 = y0+d*sind(theta);
x2 = x0+1.4142*d*cosd(45+theta);
y2 = y0+1.4142*d*sind(45+theta);
x3 = x0+d*cosd(theta+90);
y3 = y0+d*sind(theta+90);
%
%
 x = [x0 x1 x2 x3 x0 x0]
 y = [y0 y1 y2 y3 y0 y0]

%%%%%%%% Calculation of angles of the square %%%%%%%%

ang1 = atan2d(y1-y0,x1-x0);
ang2 = atan2d(y2-y1,x2-x1);
ang3 = atan2d(y3-y2,x3-x2);
ang4 = atan2d(y0-y3,x0-x3);

ang = [ang1 ang2 ang3 ang4 ang4];
 j = i;


if i<5
    if (sqrt((x(j+1)-xl)^2+(y(j+1)-yl)^2))> 0.07    ...
       %Checking for robot within 7cm from the ...
       destination point
```

```
56          thetal = ang(j);
57          vl1 = vl;
58          it = i;
59      else
60          thetal = ang(j+1);
61          vl1 = 0;
62           if theta_l ≤ ang(j+1)+ 10 && theta_l ≥ ...
                ang(j+1)- 10
63              i = i+1;
64          end
65
66          it = i;
67      end
68  else
69      vl1 = 0;
70      thetal = ang(j);
71      it = i;
72  end
73
74  %%%%%%% Logic for Follower %%%%%%%%%%%%%%%%%%
75
76  x_lf = (xl-0.35*cosd(thetal));
77  y_lf = (yl-0.35*sind(thetal));
78
79  if (sqrt((xf-x_lf)^2+(yf-y_lf)^2))> 0.1
80      thetaf = atan2d((y_lf-yf),(x_lf-xf));
```

```
81      vf1 = 80;
82  else
83  if (sqrt((xl-xf)^2+(yl-yf)^2))> 1
84      thetaf = atan2d((yl-yf),(xl-xf));
85      vf1 = 80;
86  else
87      thetaf = ang(j);
88      vf1 = vl1;
89  end
90  x_0 = x0;
91  y_0 = y0;
92
93  end
94
95
96  end
```