Feedback Control and Obstacle Avoidance for Non-Holonomic Differential Drive Robots

by

Dhruv Chopra

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2013 by the
Graduate Supervisory Committee:

Armando Rodriguez, Chair
Konstantinos Tsakalis
Jennie Si

ARIZONA STATE UNIVERSITY

December 2013

ABSTRACT

This thesis discusses control and obstacle avoidance for non-holonomic differential drive mobile vehicles. The two important behaviors for the vehicle can be defined as go to goal and obstacle avoidance behavior. This thesis discusses both behaviors in detail.

Go to goal behavior is the ability of the mobile vehicle to go from one particular co-ordinate to another. Cruise control, cartesian and posture stabilization problems are discussed as the part of this behavior. Control strategies used for the above three problems are explained in the thesis. Matlab simulations are presented to verify these controllers.

Obstacle avoidance behavior ensures that the vehicle doesn't hit object in its path while going towards the goal. Three different techniques for obstacle avoidance which are useful for different kind of obstacles are described in the thesis. Matlab simulations are presented to show and discuss the three techniques.

The controls discussed for the cartesian and posture stabilization were implemented on a low cost miniature vehicle to verify the results practically. The vehicle is described in the thesis in detail. The practical results are compared with the simulations. Hardware and matlab codes have been provided as a reference for the reader.

DEDICATION

To my parents

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Rodriguez for the mentorship he provided me for this thesis. I would also like to thank my roommate Preet Inder Singh who provided me with the moral support during the thesis.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION AND OVERVIEW

Automation in mobile vehicles has been researched a lot in last few decades. Automated mobile vehicles will play a big role in future everyday life. Modern automobiles are being designed to reduce the burden on the drivers. Various kind of automation like vehicle tracking systems, rear view alarm, anti-lock braking system, traction control system have been introduced to make the vehicles easy to operate. Some of the applications for automated ground vehicles are space missions, transportation, military operations, home solutions etc.

1.1 Motivation

Control of ground vehicles is in general a more difficult task than the aerial vehicles. A ground vehicle has to deal with varying degree of friction, slopes and obstacles. Also generally the ground vehicles are non-holonomic in nature and hence their control is not straightforward.

1.1.1 Background

This thesis is part of the Multi-Robot System(M.R.S) project at the control system laboratory at Arizona State University. Multi-Robot System has been conceived as a system consisting of multiple daughter robots having a mother robot controlling them. This swarm of daughter robots would perform tasks assigned to them by the mother vehicle in synchronization with each other.

This thesis is focused on giving these robots their most basic functionality that is the so called "go to goal and obstacle avoidance" behaviors. The go to goal behavior

as the name indicates is the ability of the mobile vehicle to go from one particular co-ordinate to another when instructed to go. The avoid obstacle behavior ensures that the robot does not bump into things while going toward its goal. The thesis also briefly discusses path tracking for the mobile robots.

Control strategies for Cartesian stabilization, posture stabilization and obstacle avoidance are discussed in the thesis. The controls were implemented on a low cost miniature vehicle to verify the results.

1.1.2 Autonomous Ground Vehicles

With enormous progress in the semiconductor technology has resulted in more and more sophisticated and intelligent ground vehicles. Equipped with such powerful processors, the ground vehicles today can perform difficult tasks intelligently in hostile environments.

An autonomous robot would be one in which computer would do everything. The robot would be given an instruction to go to some particular destination and then the computer will take it to the required destination. The computer will decide on the left and right wheel velocity by itself and steer the robot to go to the required destination.

In the process of going to the destination the vehicle has to make sure that it does not hit the obstacles in the way. Thus it may have to travel around obstacles to reach its destination. A crude block diagram for the control is as shown in the Figure 1.1 .

One of the example for such an intelligent ground vehicle is Mars Exploration Rover[18]. There are two such rovers designed as twins. The goal was to make the rovers as a mechanical equivalent of a geologist. These rovers carry panoramic camera, miniature thermal emission spectrometer, Mossbauer spectrometer, alpha particle x-ray spectrometer, magnets, microscopic imagers and rock abrasion tool.

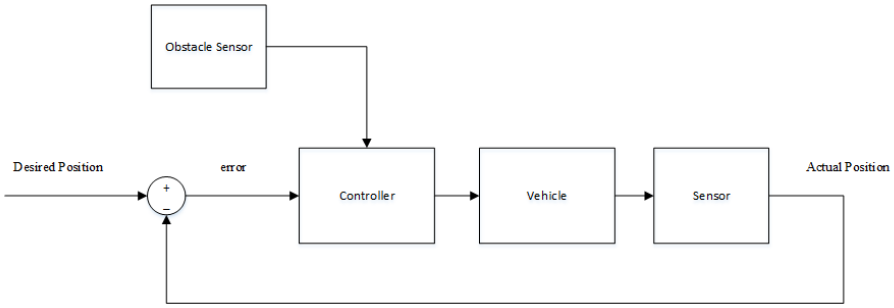

Figure 1.1: Block Diagram

Such an advanced system had been made possible only by the rapid advancement of the semiconductor technology.

1.2 Literature Survey

1.2.1 Modeling

An accurate model of vehicle behavior is very important for the design of the controller for the vehicle. There has been considerate research in developing accurate models for the vehicle. There are two approaches to develop the vehicle model: kinematic and dynamic.

Kinematic model is the most widely used and simpler of the two model for the mobile robots[5]. A kinematic model ignores dynamic properties like mass, inertia, friction etc. It only relies on the non-holonomic constraints of the mobile vehicle. This model is less accurate than the dynamic model. This thesis uses kinematic modeling for designing the position controllers. The model is explained in detail in chapter three.

Dynamic model is much more involved model of the vehicle[6]. It incorporates dynamic properties such as mass, inertia, friction, wheel slippage etc. It gives a much more accurate model of the mobile vehicle. It is difficult to work with this

model since some of the dynamic properties are very hard to measure. [6] explains a dynamic model for mobile vehicles. In [11] a dynamic model which includes torque coupling but neglects actuator dynamics is considered. A controller based on the dynamic model gives a robust controller with respect to the dynamic behavior of the mobile robot. A simple dynamic model has been used in this thesis for the cruise control of the vehicle.

1.2.2 Control

Lot of research has been done to design controllers for the non-holonomic mobile robot. Various linear and non linear controllers have been invented for mobile robots for different kind of tasks.[5] describes nonlinear controllers for the design of controllers for the three different problems in the control of differential drive mobile robots; i.e. trajectory tracking, path following and posture stabilization. D'Andrea-Novel et al. discuss feedback linearization method for tracking problem[13]. Feedback stabilization for car like vehicles has been discussed in [12] and [15]. Some more non-linear controllers are discussed in [3][4]. Viera et al. in [7] discusses a method to use dynamic linear controller for the posture and cartesian stabilization problems. Chapter four gives a detailed explanation of the feedback control for robot.

1.2.3 Obstacle Avoidance

Obstacle avoidance has been given a lot of attention in literarture in last few decades. Borenstein and Koren gives a good account of some of the obstacle avoidance techniques in [9]. Egersted et al details the robot actions in terms of behaviors[8]. In his lecture series at Georgia Tech University Prof. Egersted describe some obstacle avoidance techniques[19]. Chapter five describes the control used for the obstacle avoidance.

1.3 Contributions

The following contributions are made by this thesis .

1. A simple dynamic model and nonlinear kinematic model was used for modeling and analysis of the mobile robot.

2. Cruise control, cartesian stabilization and posture stabilization problems are addressed in the thesis. Matlab simulations are presented to verify the controllers.

3. Three methods of obstacle avoidance are described in the thesis. Simulations are presented to detail the advantages and disadvantages of these approaches.

4. A low cost miniature vehicle was used to implement the control and obstacle avoidance techniques. The results are used to verify the controllers.

5. All the documentation for the hardware and software has been provided in the thesis.

1.4 Organization of Thesis

The remainder of the thesis is organized as follows:

Chapter two describes the hardware used for implementing the control. AAR robot is described in detail in this chapter. The details about the obstacle sensor used are also given in this chapter.

Chapter three addresses the modeling for the differential drive non-holonomic mobile robot. Non-holonomic constraint and differential drive are detailed in this chapter. Dynamic and kinematic model used for the modeling are explained in detail. Finally the localization method used is explained.

Chapter four describes the feedback control for cruise control, cartesian and posture stabilization and path tracking problems. Simulations are presented to verify the controls used.

Chapter five discusses the control for obstacle avoidance for the robot. Three methods are discussed in this chapter. Advantages and shortcomings of the methods are detailed.

Chapter six summarizes the thesis and presents direction for the future research.

Appendix A contains the arduino and matlab codes used. Appendix B includes the circuit diagrams of the hardware used.

1.5 Summary And Conclusions

This chapter gave an overview of the thesis. In Section 1.1 motivation and background of the thesis was discussed. Section 1.2 gives the literature survey. Next the contribution of the thesis were listed. Section 1.4 explained the organization of the rest of the thesis.

Chapter 2

HARDWARE DESCRIPTION

This chapter describes the low cost miniature mobile robot used for in the thesis. The architecture, memory, peripheral features and other features of the robot are discussed in the chapter. The sensor used for the obstacle sensing is also described in the chapter.

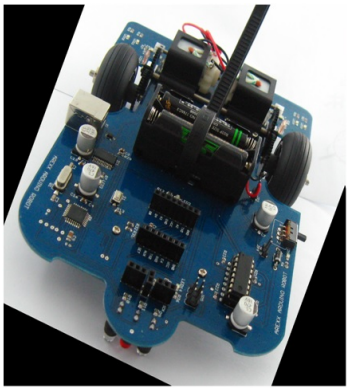

Figure 2.1: AAR

2.1 AAR Robot

This project used AAR robot for implementing the various controls[16]. AAR is a ready to use, low cost differential drive mobile from Global Specialty Inc. AAR has two brushed dc motors along with its drive and rotary encoders. AAR uses Atmega 328p as the micro-controller. AAR can be programmed according to the application required. The system diagram is shown in figure 2.2.

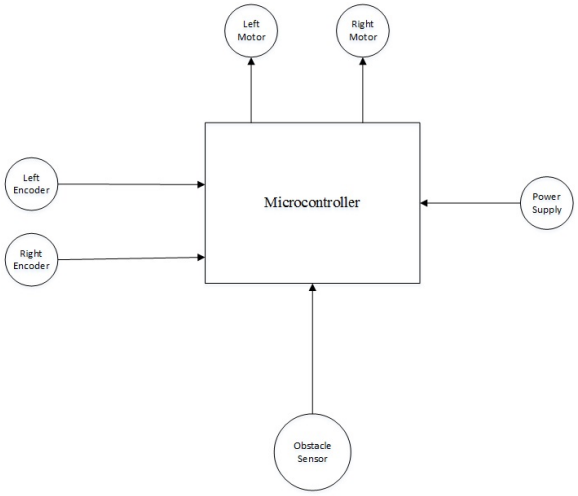

Figure 2.2: System

2.2 Specifications

A specification table is given below. It lists some of the characteristics of the robot.

Table 2.1: Specifications

S.no	Specification	Value
1	Motors	2 dc motors(3V)
2	Microcontroller	ATmega -328p
3	Programming Platform	Arduino
4	Supply Voltage	4.8-6 Volts
5	Supply current	10 to 600 mA

2.3 Microcontroller

The AAR robot has ATmega 328P microcontroller on board. It is high performance, low power RISC architecture 8 bit microcontroller from ATMEL corporation.

2.3.1 RISC Architecture

The controller has a RISC architecture. It has 131 assembly language instructions. The controllers can perform 20 million instructions per second at 20 MHz operating frequency. 32X8 general purpose registers and two on chip multiplier give lot of flexibility to the programmer.

2.3.2 Memory Segment

The controller has 32K bytes of in system self programmable flash memory and 1K bytes of EEPROM. 2K bytes of internal SRAM with in system programming by on chip boot program is featured by the system. The controller promises 20 years of code retention at 85 degree Celsius. It has promises 10000 write/erase cycles for FLASH and 100000 write/erase cycles for EEPROM.

2.3.3 Peripheral Features

With 23 Input/Output line the controller has a good number of lines to support applications. The controller has two 8 bit timers and one 16 bit timers. The 8 bit timers have separate prescaler and compare mode. The 16 bit timer has separate prescaler, compare and capture mode. The controller has a dedicated real time counter and a 6 P.W.M channels. ATMega 328p includes 8 channel 10 bit ADC. The controller also boasts of SPI interface, 2 wire serial interface, watchdog timer and on board analog comparator.

2.3.4 Other Features

The controller has an operating voltage of 1.8 to 5.5 volts . The sink current is 0.2 mA and the working temperature range is -40 to 85 degree celsius.

2.4 Motors

The AAR robot consists of 2 dc motors working at 3 volts. The motors have gears attached to them, the gear ratio is 8:100. Other than this no other information is available on the motors in the AAR documentation.

2.5 Motor Driver IC

The AAR uses L293D IC's to drive the motors. This IC consists of 2 H-Bridge channels and can drive. 6 Amps of current per H-bridge and a peak current of 1.2 Amps. The supply range of IC is from 4.5V to 36V. The IC's circuit is given in the appendix.

2.6 Motor Encoders

The AAR robot includes two optical motor encoders for the two motors. With the gear ration taken into account it converts into 20 pulses per revolution of the robot. With the help of these encoders the robot linear speed, angular speed and positions are estimated. This is called passive localization or dead reckoning. Passive localization introduces some error in the estimated position of the robot. Active localization methods like indoor G.P.S. can be used to increase the accuracy, but it increased the cost of the vehicle system considerably. This thesis uses odometeric localization for position calculation. A The motor encoder circuit is shown in the appendix.

2.7 Obstacle Sensor

To avoid obstacles, it is necessary for the robot to sense the obstacles. There are many different kind of sensors available in the market. Some examples are IR sensor,

SONAR sensors, Ultra Sonic sensors etc. SONAR and Ultrasonic sensors are more precise and costlier than the IR sensors. Most of these sensor give voltage output as an indication of the distance to the obstacle.

In this thesis Sharp GP2Y0A02YK0F I.R Analog Distance Sensor has been used[17]. The maximum range of this sensor is 150cm while the minimum range is 20 cm. The details of the sensor are given in the Appendix B.

2.8 Summary And Conclusions

This chapter gave the details about the hardware used. AAR robot from global inc. has been used for this thesis. AAR has two brushed dc motors along with its drive and rotary encoders. AAR uses Atmega 328p as the micro-controller. The complete details are given in the chapter. The obstacle sensor has also been disccused in the chapter.

Chapter 3

MODELING

This chapter describes the mathematical model of the robotic vehicle. This model is used to derive the feedback control method for cruise control, cartesian stabilization, posture stabilization and obstacle avoidance problems.

In autonomous mobile robots usually two kind of modeling is used - kinematic and dynamics. Kinematic modeling doesn't include mass, torque, inertia etc. It treats the robot as a point object. Dynamic model includes the mass, inertia, slippage etc of the system as well and hence is more precise. This thesis uses kinematic model for the position control, whereas some of the dynamics are included in the motor model which is used for cruise control problem.

3.1 Differential Drive

In a differential drive vehicle both the wheels of the vehicle are controlled independent of each other. The vehicle is steered by the relative rotation of the two wheels. If the two wheels are moving with same speed in same direction, the vehicle will move straight. If the two wheels are rotating in opposite direction with the same speed the vehicle should ideally turn about the central point of axis.

Depending upon the relative speed of rotation of the two vehicles the center of rotation would be along some point on the lone define by the two wheels. The linear and angular velocity of the vehicle is given by the following equations.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{R}{2} & \frac{R}{2} \\ \frac{R}{L} & \frac{-R}{L} \end{pmatrix} \begin{pmatrix} \omega r \\ \omega l \end{pmatrix} \tag{3.1}$$

where ωr is the angular speed of the right wheel, ωl is the left wheel angular speed.v is the linear speed of the robot, ω is the angular speed of the robot.R is radius of the wheels, L is the axle length between the two wheels.

3.2 Non-holonomic Constraints in a Differential Drive Vehicle

A differential drive mobile robot's local movements are restricted but there is no restrictions on the global movements[5]. The robot cannot have a speed in a direction perpendicular to the sagittal axis. But this constraint doesn't result in a constraint on the position of the robot. The robot can still be maneuvered into any position. The situation is shown in the figure 3.1.

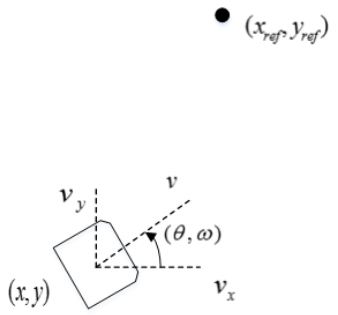

Figure 3.1: Non-holonomic Constraint

It can be better understood from the case of the parallel parking. The driver can not slide the car into the parking space. It has to be maneuvered into the parking space. The figure 3.1 shows the non-holonomic constraint in a differential drive mobile robot. Non- holonomic constraint in the case of mobile robots is given by following equation.

$$\dot{x}sin\theta - \dot{y}cos\theta = 0 \tag{3.2}$$

$$\frac{\dot{y}}{\dot{x}} = \tan\theta = \frac{dy}{dx} \tag{3.3}$$

3.3 Kinematic Model

The kinematic model of the system is a three degree of freedom model. Kinematic model does not includes vehicle dynamics, like mass, inertia, motor parameters, friction. The model is given by following equations. The (x, y) coordinate gives the location of the vehicles center of mass. The cars angle with respect to x axis is theta. For the purpose of modeling, global coordinate are used.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{3.4}$$

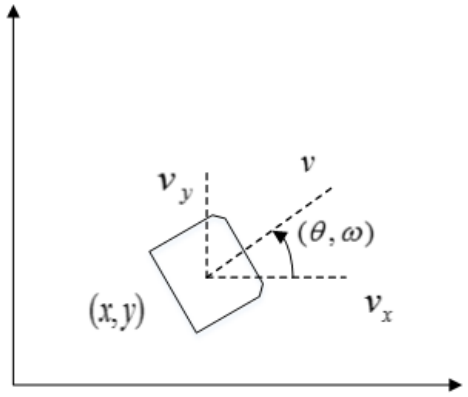

Figure 3.2: Kinematic Model

3.4 Dynamics

The dynamics of the vehicle like moment of inertia, torque and friction are included in the D.C. motor model.The torque T produced in a D.C motor is proportional to the armature current and the magnetic field. Here we assume the magnetic field is constant and hence the torque is proportional only to the armature current I. The back E.M.F Eb is directly proportional to the angular speed ω. The motor is governed by the following equations. The motor dynamics are shown is shown in Figure 3.3.

$$E_a - E_b = La\frac{dI}{dt} + IR_a \tag{3.5}$$

$$T = k_t I \tag{3.6}$$

$$T = J\frac{d\omega}{dt} + B\omega \tag{3.7}$$

$$E_b = k_b\omega \tag{3.8}$$

where

E_a is the applied voltage,E_b is the back E.M.F. L_a is the inductance, I is the current.T is the Torque, J is moment of inertia.B is Friction constant.ω is the angular speed of the motor.k_t is the torque constant of the motor.k_b is the speed constant of the motor.

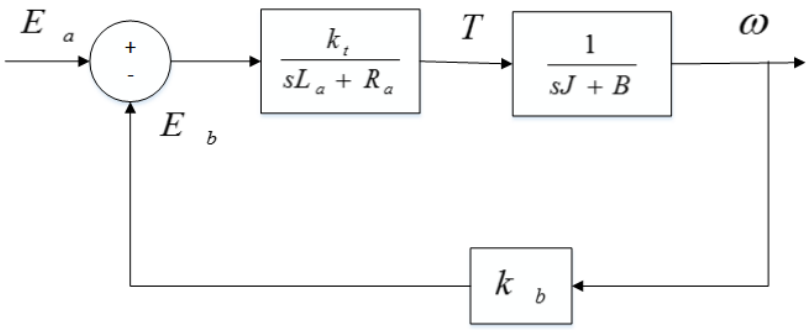

Figure 3.3: Motor Model

In the laplace domain the same equations are given by.

$$I(s) = \frac{E_a(s) - E_b(s)}{sL_a + R_a} \tag{3.9}$$

$$T(s) = k_t I(s) \tag{3.10}$$

$$\omega(s) = \frac{T(s)}{sJ + B} \tag{3.11}$$

$$Eb(s) = k_b\omega(s) \tag{3.12}$$

$$k_t = k_b = k \tag{3.13}$$

The governing equations of the motor dynamics can be expressed in the state space form as follows

$$\begin{pmatrix} \dot{\omega} \\ \dot{I} \end{pmatrix} = \begin{pmatrix} \frac{-B}{J} & \frac{-k}{J} \\ \frac{-k}{L} & \frac{-R_a}{La} \end{pmatrix} \begin{pmatrix} \omega \\ I \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix} Ea \tag{3.14}$$

Solving the above equations we arrive at the open loop the transfer function from the E_a to ω as

$$\frac{\omega(s)}{E_a(s)} = \frac{k}{k^2 + (sL_a + R_a)(sJ + B)} \tag{3.15}$$

Table 3.1: Parameters

Parameter	Description	Nominal Value
k_t	Torque Constant	0.01 Nm/amp
k_b	Speed Constant	0.01 V/(rad/sec)
La	Armature Inductance	0.1 mH
Ra	Armature Resistence	0.1 ohm
J	Moment of Inertia	0.1 kgm^2
B	Air Damping	0.1 Nms
L	Distance Between wheels	9.9 cm
R	Radius of wheels	1.9 cm

3.5 Complete Model

The complete plant is motor dynamics and vehicle kinematics combined together. This is shown in the figure 3.4. The motor model output are the left and right angular

speeds. The matrix M gives a transformation from the motor speeds to linear and angular speed of the vehicle. The kinematic model gives the location of the vehicle in the cartesian space.

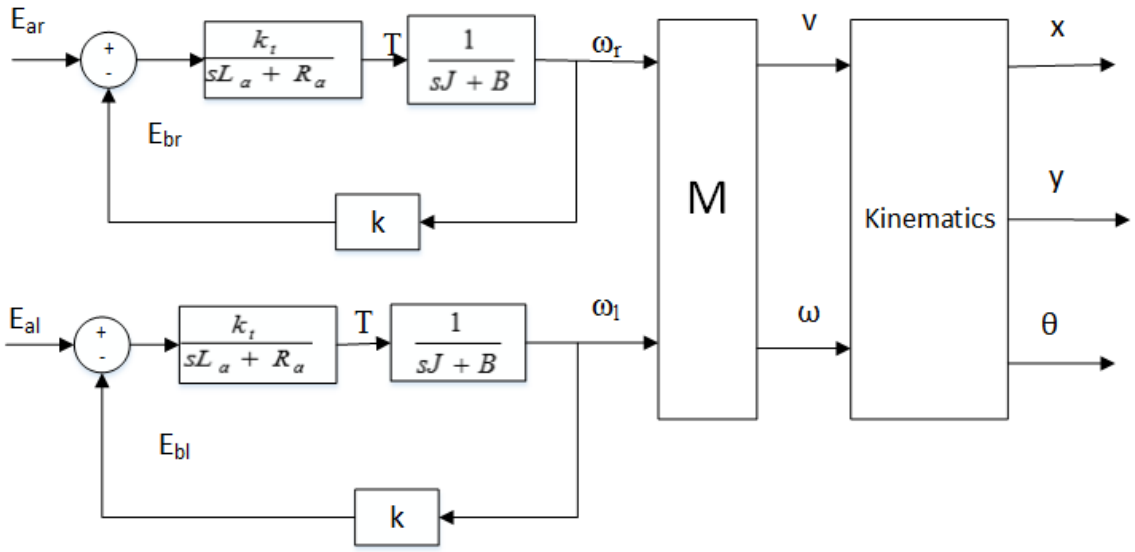

Figure 3.4: Complete Plant

3.6 Odometric Localization

To implement feedback control, robot's posture at each instant is required. AAR robot has incremental encoders that measure the rotations of the wheels. Using this data and combining it with kinematic equation, the robot's posture can be derived.

The assumption made for the calculation is that the linear velocity v and angular velocity ω are constant for the sampling period T_s. Defining that the robot's posture is q_k at time T_k. Then the posture at time T_{k+1} can be derived using forward integration of the kinematic model.

Using Euler method of integration an approximate formula can be used[20].

$$x_{k+1} = x_k + v_k T_s cos\theta_k \tag{3.16}$$

$$y_{k+1} = y_k + v_k T_s sin\theta_k \tag{3.17}$$

$$\theta_{k+1} = \theta_k + \omega_k T_s \tag{3.18}$$

The Euler method introduces an error in x_{k+1}, y_{k+1} since it assumes orientation θ_k is constant throughout T_s. A more exact reconstruction is[20]

$$x_{k+1} = x_k + \frac{v_k}{\omega_k}(sin\theta_{k+1} - sin\theta) \tag{3.19}$$

$$y_{k+1} = y_k - \frac{v_k}{\omega_k}(cos\theta_{k+1} - cos\theta) \tag{3.20}$$

$$\theta_{k+1} = \theta_k + \omega_k T_s \tag{3.21}$$

Equations for x_{k+1} and y_{k+1} still exists for $w_k = 0$, they become equal to the Euler method. In implementation on hardware a conditional statement must be used.

Odometric localization is easy to implement and very inexpensive. At a high sampling rate the positional accuracy is very high. But external factor not considered in kinematic equation produce errors. Wheel slippage and rough surfaces are some of the factors. More expensive methods like G.P.S. or ground beacon system can be used to overcome these errors.

3.7 Summary And Conclusions

This chapter described the mathematical model of the mobile robot used. Non-holonomic constraints in a differential drive mobile robot put a constraint on the vehicle velocity. It has been explained in the chapter. There are two kind of modeling used for mobile robot: Dynamic and kinematic model. Both were discussed in detail in this chapter. The complete model for the robot as the combination of motor dynamic and vehicle kinematics is explained. Odometric localization is used in this project. This method of localization is inexpensive and easy but prone to errors. The errors get accumulated over time. The complete method is discussed in the chapter.

Chapter 4

CONTROL

The control problem of cruise control, cartesian stabilization and posture stabilization have been addressed in this chapter.

Cruise control ensures that the vehicle moves at the commanded speed and the commanded angle. Posture stabilization problem states that a vehicle must reach the reference coordinates $(x_{ref}, y_{ref}, \theta_{ref})$ given its initial position (x_i, y_i, θ_i). Posture stabilization problem has received a lot of attention in research. There are several controllers proposed in the literature for the problem. A. Astolfi gives nonlinear control for the position stabilization problem[2]. Viera et al [7] discusses dynamic linear controller for the same. Cartesian stabilization problem is a subset of the posture stabilization problem and is also discussed in this chapter. Simulations are presented to verify the controllers used for the mentioned problems.

4.1 Cruise Control

Cruise control ensures that the robot moves at the commanded speed along a commanded angle with respect to the global co-ordinate axis. The motor model discussed in the chapter three is used for the control. The model is completely linear and classical control techniques are sufficient to design the control.

To control the linear velocity v and angular velocity ω of the vehicle we need to control the individual angular velocities of the wheel ω_r and ω_l.

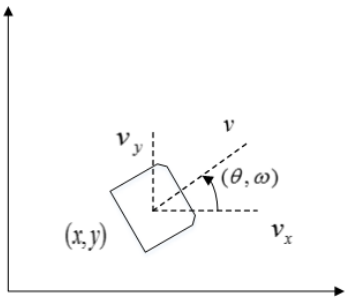

Figure 4.1: Cruise control

4.1.1 Motor Speed Control

The plant transfer function for the motor speed control was discussed in chapter three. The block diagram for the closed loop control is shown in Figure 4.2.

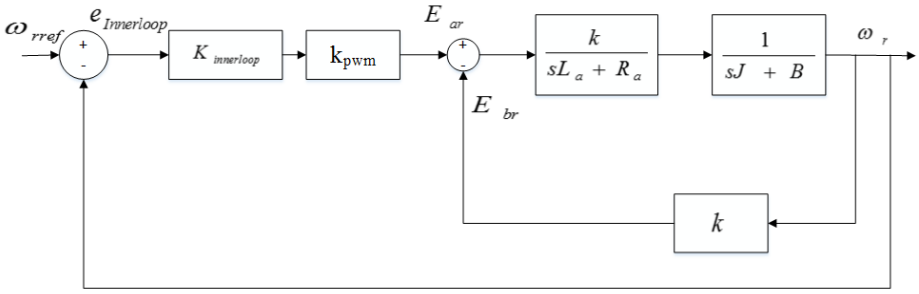

Figure 4.2: Closed loop block diagram for motor speed control

$$P_{innerloop} = \frac{k.kpwm}{k^2 + (sL_a + R_a)(sJ + B)} = \frac{3300}{(s + 1.02)(s + 5000)} \tag{4.1}$$

A proportional integrator controller $K_{innerloop}$ of the form given in equation 4.2 is used to meet the desired specifications.

$$K_{innerloop} = \frac{g(s + z)}{s}[\frac{100}{s + 100}]^2 \tag{4.2}$$

the open loop loop transfer function $G_{innerloop}$ is given by

$$G_{innerloop} = P_{innerloop}K_{innerloop} \tag{4.3}$$

the closed loop loop transfer function $H_{innerloop}$ is given by

$$H_{innerloop} = \frac{G_{innerloop}}{G_{innerloop} + 1} \tag{4.4}$$

three different controllers with different desired specification are shown in the Table.

Table 4.1: g and z designs with specifications

S.no	g	z	settling time(s)	peak overshoot(%)
1	7.5	1	1	0
2	5	5	$\frac{5}{3}$	10
3	9	5.5	1	5

4.1.2 Simulations

Frequency and time responses for the three controllers listed in table 4.1 are given below. The control structure is shown in Figure 4.2.

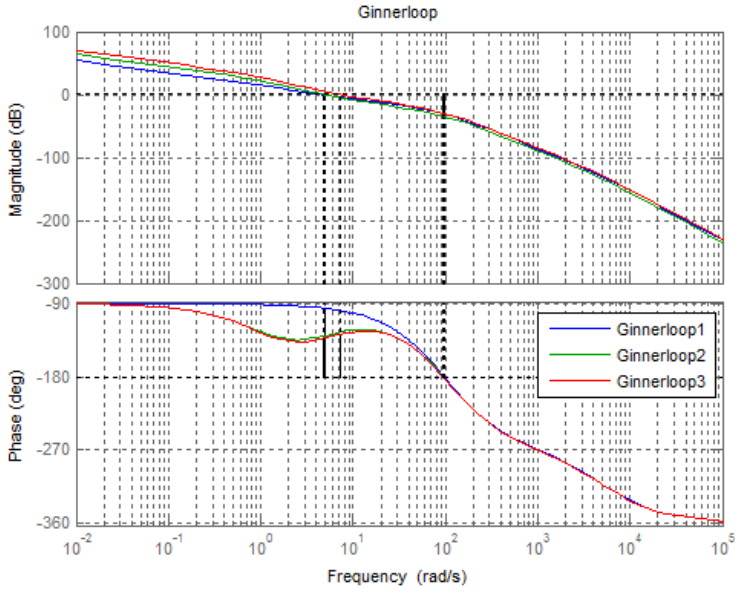

Figure 4.3: $G_{innerloop}$ Bode Plot

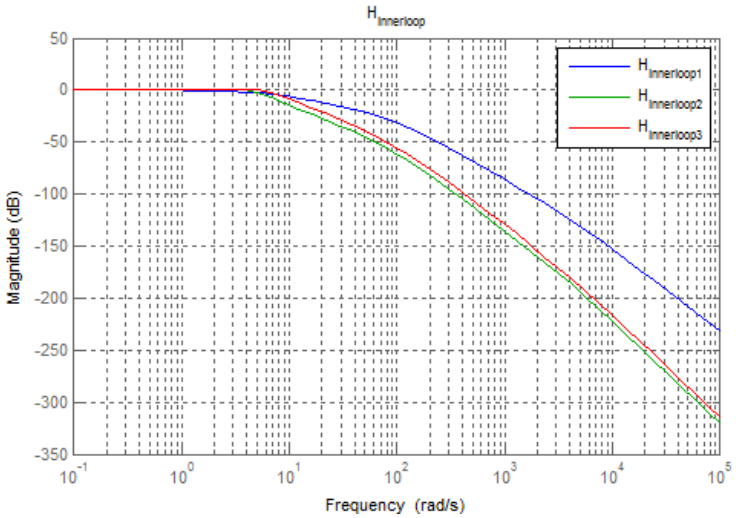

Figure 4.4: $H_{innerloop}$ Bode Magnitude

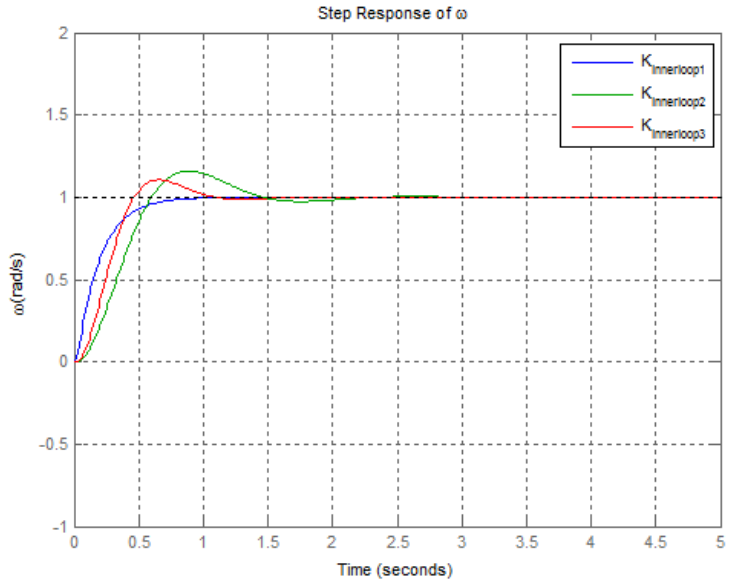

Figure 4.5: Output step response

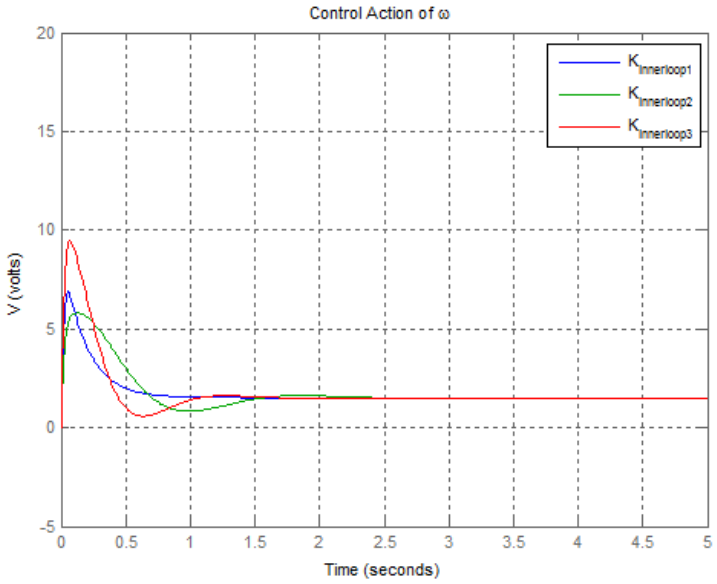

Figure 4.6: Control Action

4.1.3 Angle Control

In the previous section the control design for the individual angular velocities was discussed. Now the complete cruise control control will be discussed. The complete architecture for the cruise control is shown in figure 4.7.

As discussed in the chapter three individual angular velocity references $(\omega_{rref}, \omega_{lref})$ are obtained from the v and ω commands as shown below.

$$\begin{pmatrix} \omega_{rref} \\ \omega_{lref} \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{L}{2R} \\ \frac{1}{R} & \frac{-L}{2R} \end{pmatrix} \begin{pmatrix} v_{ref} \\ \omega_{ref} \end{pmatrix} \tag{4.5}$$

$$M^{-1} = \begin{pmatrix} \frac{1}{R} & \frac{L}{2R} \\ \frac{1}{R} & \frac{-L}{2R} \end{pmatrix} \tag{4.6}$$

where ω_{rref} is the commanded angular speed of the right wheel, ω_{lref} is the commanded left wheel angular speed. v_{ref} is the linear speed of the robot, ω_{ref} is the

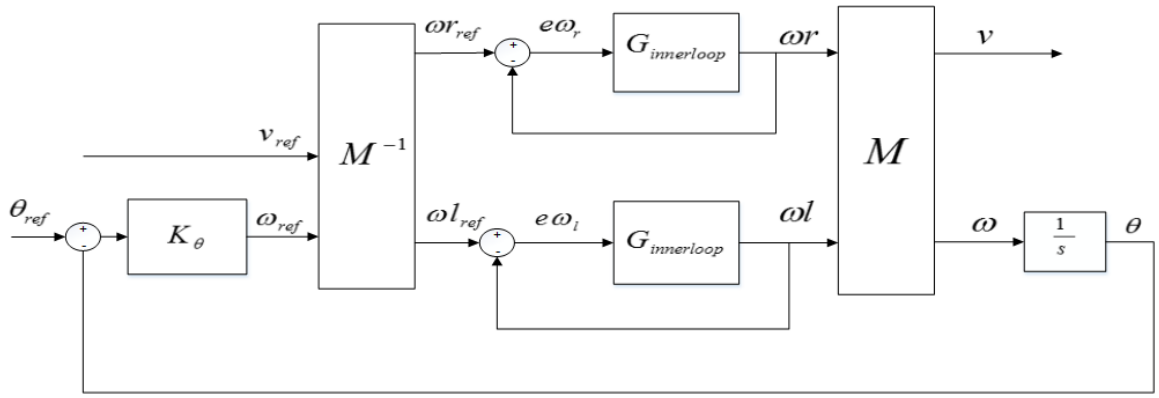

Figure 4.7: Cruise control architecture

angular speed of the robot.R is radius of the wheels, L is the axle length between the two wheels.

It can be seen that the transfer functions from $v_{ref} \rightarrow$ v and $\omega_{ref} \rightarrow \omega$ is equal to $H_{innerloop}$. Hence the $v_{ref} \rightarrow$ v and $\omega_{ref} \rightarrow \omega$ time and frequency responses are same as the motor speed responses.

$$\begin{pmatrix} v(s) \\ \omega(s) \end{pmatrix} = M^{-1} \begin{pmatrix} H_{innerloop}(s) & 0 \\ 0 & H_{innerloop}(s) \end{pmatrix} M \begin{pmatrix} v_{ref}(s) \\ \omega_{ref}(s) \end{pmatrix} \tag{4.7}$$

$$\frac{v(s)}{v_{ref}(s)} = \frac{\omega(s)}{\omega_{ref}(s)} = H_{innerloop}(s) \tag{4.8}$$

where v_{ref} is the velocity command, ω_{ref} is the angular velocity command.

The second part of the cruise control problem is to make the vehicle follow the commanded angle θ_{ref}. As can be seen from the Figure 4.7, the plant for the θ control problem is given by

$$P_\theta = \frac{H_{innerloop}}{s} = \frac{24570}{(s+4995)(s+4.957)s} \tag{4.9}$$

The simplified block diagram is shown in figure below.

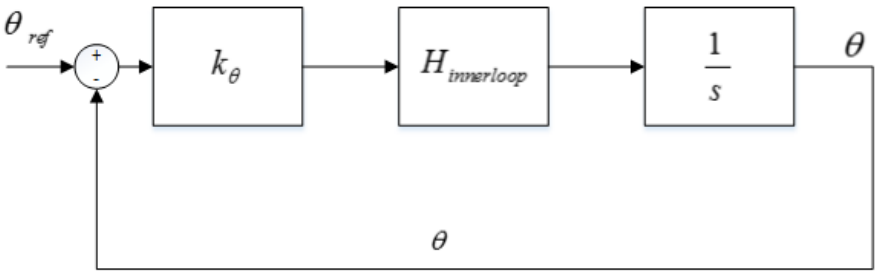

Figure 4.8: Angle Control

A simple proportional controller is used for the control. The controller is of the form.

$$k_\theta = g \tag{4.10}$$

the open loop loop transfer function G_θ is given by

$$G_\theta = P_\theta k_\theta \tag{4.11}$$

the closed loop loop transfer function H_θ is given by

$$H_\theta = \frac{G_\theta}{G_\theta + 1} \tag{4.12}$$

4.1.4 Simulations

Bode magnitude and phase plot of G_θ is shown in figure 4.9. Bode magnitude plot of H_θ is in figure 4.10. Figure 4.11 and 4.12 show step response and controller actions. All plots are plotted for k_θ = g = 0.5,1,2 are given below. The control structure used is shown in Figure 4.8.

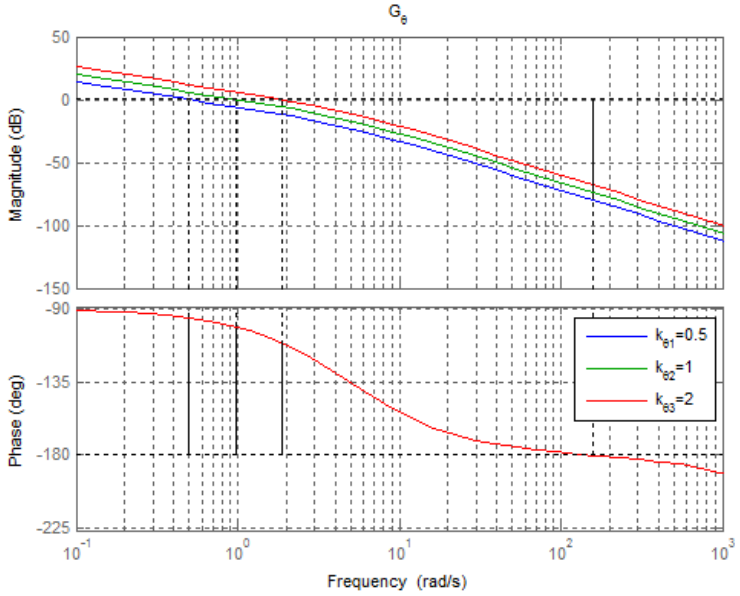

Figure 4.9: G_θ Bode Plot

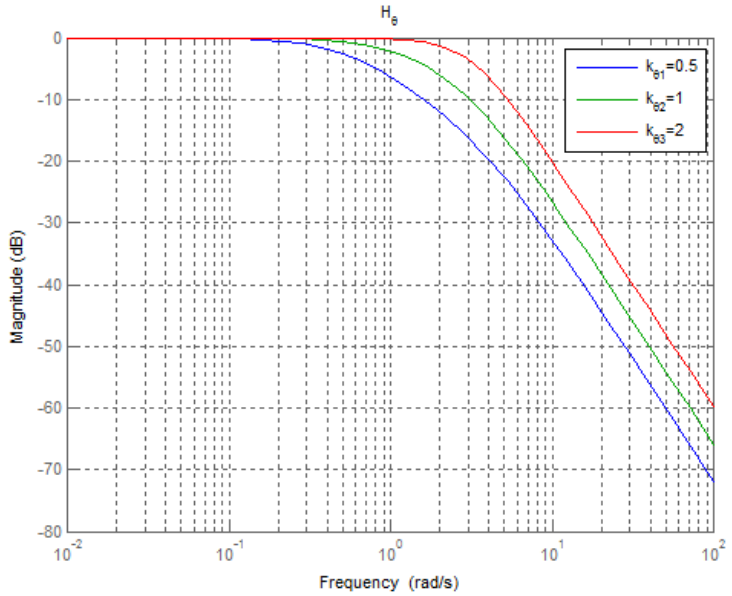

Figure 4.10: H_θ Bode Magnitude

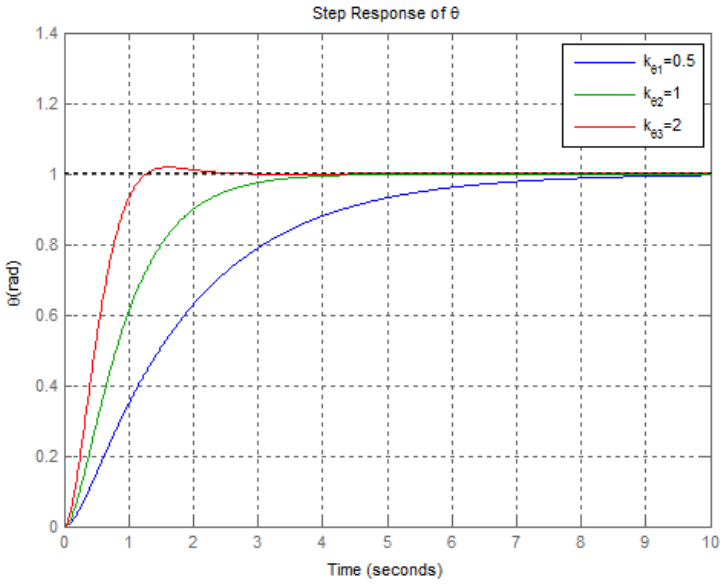

Figure 4.11: θ step response

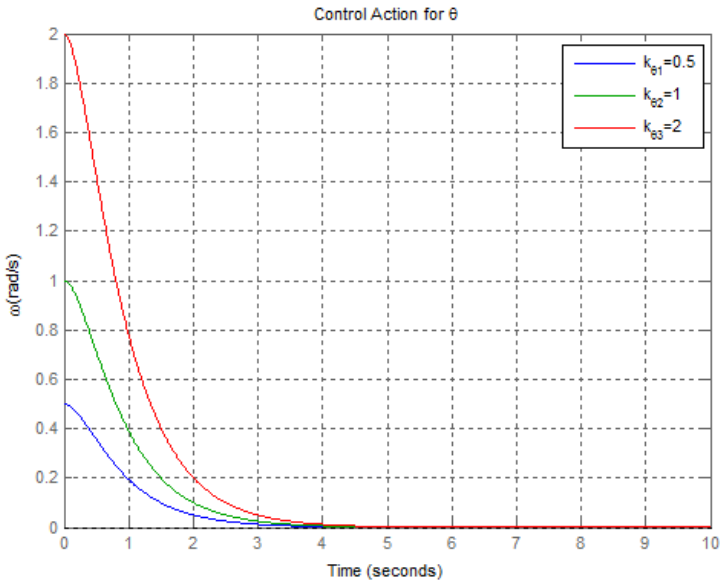

Figure 4.12: Control Action

4.2 Kinematic Model Analysis

4.2.1 Controllability

A system is said to be controllable if there exists a control law u(.) which can transfer the state of the system from any initial state x_0 to any final state x_f within a finite amount of time. Otherwise the system is said to be uncontrollable[10].

The kinematic model is given by the following equations.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.13}$$

This is a non linear system of the form :

$$\dot{p} = \sum_{i=1}^{m} h_i(p) u_i \quad p \epsilon R^n, \quad u \epsilon R^m, \quad m <= n \tag{4.14}$$

The sufficient condition of controllablility is given by

$$rank(h1, h2, [h1, h2]) = rank \begin{pmatrix} cos\theta & 0 & sin\theta \\ sin\theta & 0 & -cos\theta \\ 0 & 1 & 0 \end{pmatrix} = n = 3 \tag{4.15}$$

where

$$[h1, h2] = \frac{\partial h2}{\partial p} h1 - \frac{\partial h1}{\partial p} h2 \tag{4.16}$$

Thus $m = n = 3$ and hence the system is controllable and this confirms the common physical experience that a mobile vehicle can be taken from any point to any other in the physical space.

Linearizing the above model about equilibrium, we get the following equations for a linear system.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.17}$$

The controllibilty matrix of the linearized plant has a rank less than n . Hence this system is uncontrollable . Thus in the process of linearization the controllability of the system is lost.

4.2.2 Brockett's Theorem

Controllability in a nonlinear system is not a sufficient condition for the existence of a static smooth state feedback[5]. Brockett's theorem gives necessary conditions for smooth feedback stabilizability[1]. For the kinematic model, the following corollary as given in [2] is pertinent.

Theorem 1*if* $\dot{q} = g(q)u$ *is a continuously differentiable distribution in neighborhood of* q_0 *with* $g(q_0)u_0 = 0$ *and g(q) being a distribution of constant rank in a neighborhood of* q_0*.*

Then a continuously differentiable control law which make (q_0, u_o) *asymptotically stable exist if and only if dim(q) = dim(u).*

In the case of non-holonomic mobile robots dim(q) =3 and dim(u) = 2. Thus no smooth control law exists which can stabilize the robot about a posture. This result requires new control schemes to be used for the non-holonomic mobile robots.These new schemes include time varying control laws[14] , piece-wise continuous control[5] or model transformation techniques[2].

4.3 Cartesian Stabilization

This control ensures that the robot goes to the reference coordinates (x_{ref}, y_{ref}). The kinematic model is non linear and requires non linear control theory for the control. Astolfi in [2] explains a polar transformation method for the posture stabilization. Same method can be used for the cartesian stabilization problem. A method to use linear control has been shown by Vieara et al in [7].

4.3.1 Control

The kinematic model as given in equation 4.13 can be transformed by writing it in the term of angular and linear displacements $\dot{s} = v$ and $\dot{\theta} = \omega$. Then defining a new state vector ρ as given by

$$\rho = \begin{pmatrix} s \\ \theta \end{pmatrix} \tag{4.18}$$

The state equations can be written as

$$\dot{\rho} = \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.19}$$

We use this transformed system to steer the robot to the goal point. This is shown in the figure 4.13

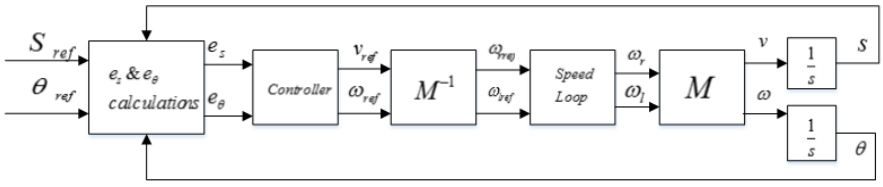

Figure 4.13: Cartesian Stabilization

The problem lies in getting the value of Sref. Sref is meaningless and S is difficult to measure. This problem can be overcome by some manipulations as explained in [7].

Consider a robot at a coordinate (x, y). The goal is to steer the robot to the reference coordinate (x_{ref},y_{ref}). The figure below shows a robot moving in space. Δd is the distance between the (x_{ref},y_{ref}) and (x, y). The control should make sure that Δd goes to zero so that the robot is steered to the target position.

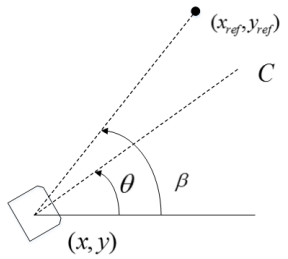

Figure 4.14: Cartesian Space

$$\Delta d = \sqrt{(x_{ref} - x)^2 + (y_{ref} - y)^2} \tag{4.20}$$

A point C is defined in the space. C is the point which is at shortest distance from target point that lies in the orientation line of the robot. The distance to C is ΔS. β is defined as the angle which binds (x_{ref},y_{ref}) and (x, y) and is called pointing angle.

$$\beta = atan(y_{ref} - y, x_{ref} - x) \tag{4.21}$$

$$e_\theta = \beta - \theta \tag{4.22}$$

$$\Delta s = \Delta d cos(e_\theta) \tag{4.23}$$

If the control is designed in such a way that Δs and e_θ goes to zero then the robot will steer to the goal point. It is also very important to note that at the point C only, Sref - S is equal to Δs.

$$e_s = \Delta s \tag{4.24}$$

$$e_\theta = \beta - \theta \tag{4.25}$$

The control law for the problem is defined as

$$v = k_s e_s \tag{4.26}$$

$$\omega = k_\theta e_\theta \tag{4.27}$$

The whole control mechanism is shown in the figure 4.15. The speed loop shown in the block diagram below is same as the used in the cruise control. As can be seen from the block diagram, the system is completely linear from (e_s, e_θ) to (s, θ). The transformations at the input from (x_{ref}, y_{ref}) to (e_s, e_θ) and kinematic transformation at output are nonlinear in nature. Hence overall the closed loop system is nonlinear in nature.

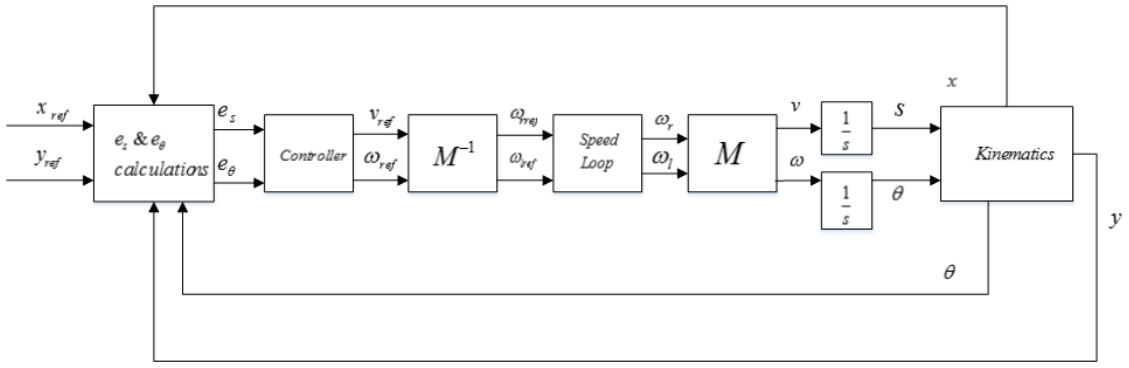

Figure 4.15: Detailed Block Diagram

As can be seen from equation 4.21 and 4.22 as (x, y) approaches (x_{ref},y_{ref}) the control law become undefined. To avoid the situation as (x, y) approaches (x_{ref},y_{ref}), the control law can be defined as

$$v = k_s \sqrt{(x_{ref} - x)^2 + (y_{ref} - y)^2} \tag{4.28}$$

$$\omega = 0 \tag{4.29}$$

4.3.2 Local Stability

Local stability of the closed loop system can be proved by the dynamic error analysis. The error dynamics of the transformed system are given by the following matrix equation.

$$\begin{pmatrix} \dot{e_s} \\ \dot{e_\theta} \end{pmatrix} = \begin{pmatrix} -1 & sin(e_\theta)e_s \\ \dfrac{sin(e_\theta)cos(e_\theta)}{e_s} & -1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.30}$$

The control law is given by the equations

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} k_s e_s \\ k_\theta e_\theta \end{pmatrix} \tag{4.31}$$

The closed loop equation can be found by substituting eq 4.31 into eq 4.30. Linearizing about the equilibrium, the closed loop equations can be written as

$$\begin{pmatrix} \dot{e_s} \\ \dot{e_\theta} \end{pmatrix} = \begin{pmatrix} -k_s & 0 \\ 0 & k_s - k_\theta \end{pmatrix} \begin{pmatrix} e_s \\ e_\theta \end{pmatrix} \tag{4.32}$$

The characteristic polynomial of the matrix has negative real roots if

$$k_s > 0; \quad k_\theta > k_s \tag{4.33}$$

Hence the closed loop system is locally exponentially stable for $k_s > 0$ and $k_\theta > k_s$.

4.3.3 Simulations

The simulations were run in Matlab. The simulation shown below is for $(x_{ref}, y_{ref}) = (100, 100)$ with initial condition $(x_o, y_o) = (0, 0)$. The block diagram in Figure 4.15 is used as the control structure. Eq 4.26-4.29 are used as the control law with $k_s = [0.1, 0.3, 0.5, 0.7], k_\theta = 0.4$.

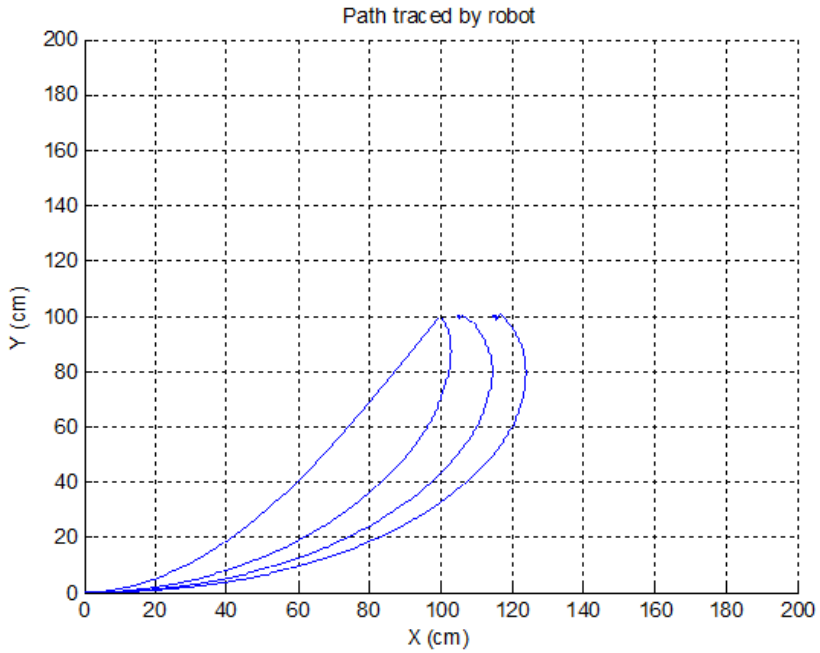

Figure 4.16: xy plot

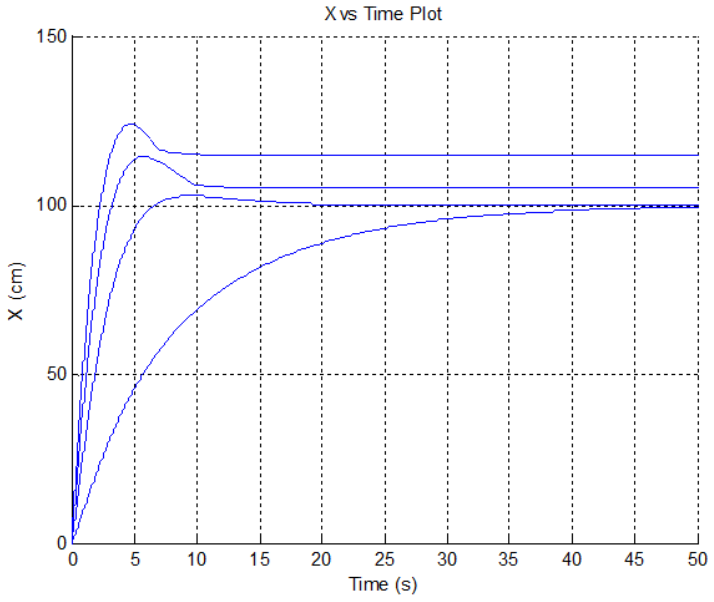

Figure 4.17: x vs time plot

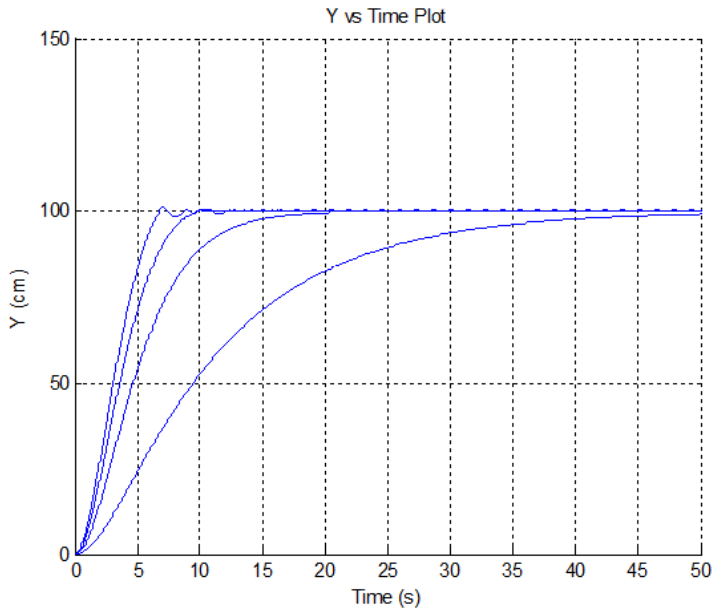

Figure 4.18: y vs time plot

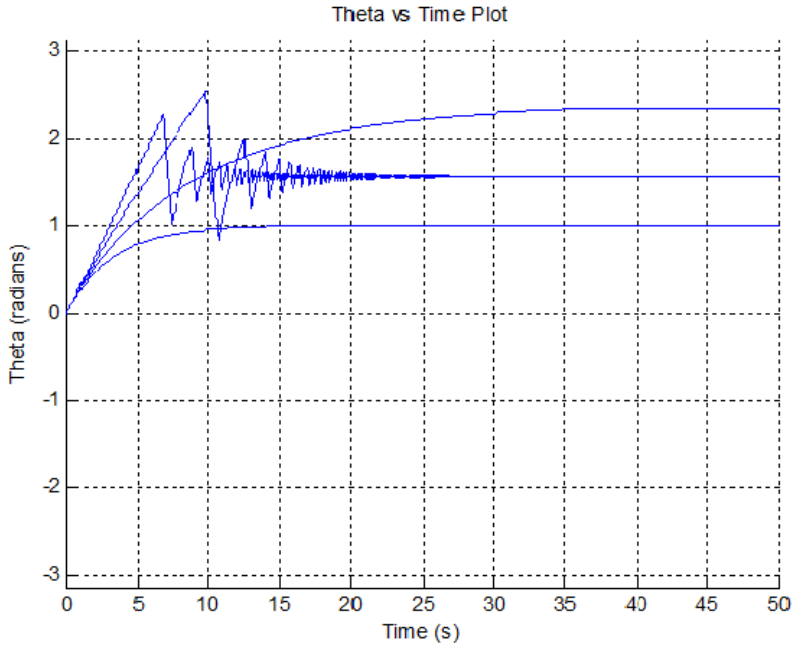

Figure 4.19: θ vs time plot

As can be seen from the above plots that the system is only stable if the local stability conditions is satisfied ; i.e. $k_\theta > k_s > 0$.

4.3.4 Practical Results

The control discussed above was implemented in the AAR robot. The path traced by the robot was compared with simulation. The result shown below is for $(x_{ref},y_{ref}) = (100, 100)$ with initial condition $(x_o, y_o) = (0, 0)$ with $k_s = 0.4 and k_\theta = 0.5$.

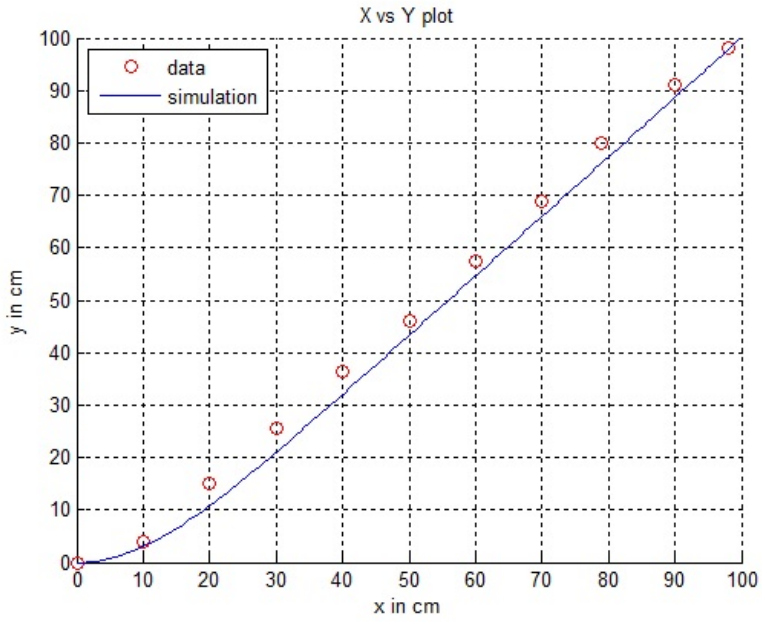

Figure 4.20: practical results

4.4 Posture Stabilization (Parking Problem)

The cartesian stabilization ensured that robot can reach the goal coordinates x_{ref},y_{ref}. Posture stabilization is required when the robot needs to reach the final destination with a specified approaching angle (x_{ref},y_{ref},θ_{ref}). This is like the parking problem where the car needs to approach the spot at a particular angle.

4.4.1 Control

The posture stabilization strategy is illustrated in [7]. In this scheme a running reference generator is used. This is shown in figure 4.21. The references (x_t, y_t) are then fed to cartesian control.

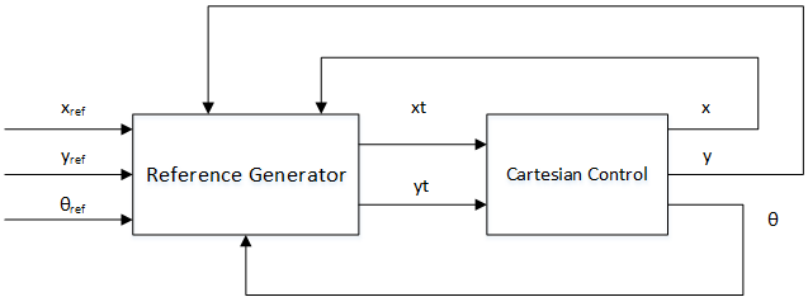

Figure 4.21: Running References

The cartesian space is shown in figure 4.22. Δd is defined as the distance between target position $(x_{ref}$,$y_{ref})$ and the current position (x, y). β is the pointing angle.αd is defined as angle difference of the approaching angle and the pointing angle . α is defined as difference between β and θ.

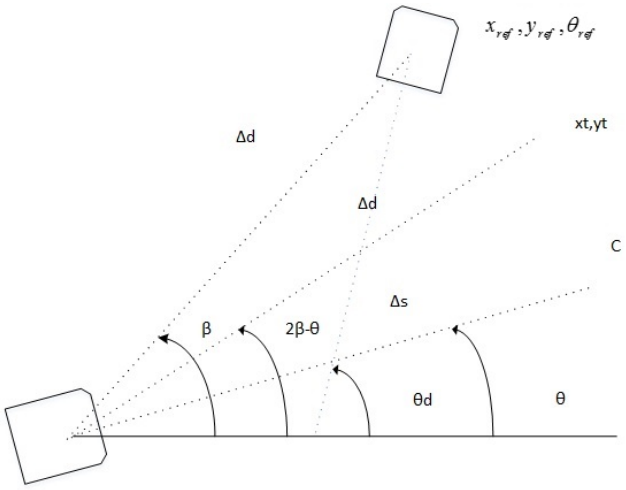

Figure 4.22: Posture Stabilization

$$\Delta d = \sqrt{(x_{ref} - x)^2 + (x_{ref} - x)^2} \tag{4.34}$$

$$\beta = atan(y_{ref} - y, x_{ref} - x) \tag{4.35}$$

$$\alpha = \beta - \theta \tag{4.36}$$

$$\alpha_d = \beta - \theta_{ref} \tag{4.37}$$

the running references are then generated as

$$x_t = x + \Delta d cos(2\beta - \theta_{ref}) \tag{4.38}$$

$$y_t = y + \Delta d sin(2\beta - \theta_{ref}) \tag{4.39}$$

The new references are used to calculate e_s and e_θ in the same way as in the Cartesian Stabilization problem. As can be seen from above equation the θ_{ref} is equal to $2\beta - \theta_{ref}$.

$$e_\theta = atan(y_t - y, x_t - x) - \theta \tag{4.40}$$

$$e_\theta = 2\beta - \theta_{ref} - \theta \tag{4.41}$$

$$e_s = \Delta d cos(e_\theta) \tag{4.42}$$

The control law is given as

$$v = k_s e_s \tag{4.43}$$

$$\omega = k_\theta e_\theta \tag{4.44}$$

The complete block diagram is in figure below

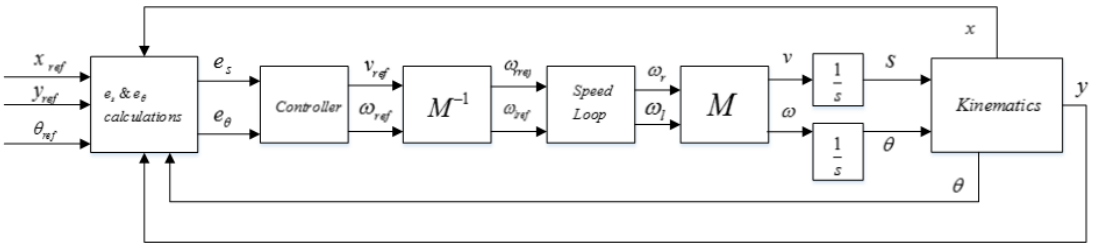

Figure 4.23: Posture stabilization block diagram

As can be seen from eq 4.43 and 4.44 as (x, y, θ) approaches $(x_{ref}, y_{ref}, \theta_{ref})$ the control laws become undefined. To avoid the situation as (x, y) approaches (x_{ref},y_{ref}) the control law can be defined as.

$$v = k_s\sqrt{(x_{ref} - x)^2 + (y_{ref} - y)^2} \tag{4.45}$$

$$\omega = k_\theta(\theta_{ref} - \theta) \tag{4.46}$$

4.4.2 Local Stability

To prove the stability of the above scheme, we divide $e\theta$ into two errors pointing error α and approach error α_d. where :

$$e_\theta = 2\beta - \theta_{ref} - \theta \tag{4.47}$$

$$e_\theta = \beta - \theta_{ref} + \beta - \theta \tag{4.48}$$

$$e_\theta = \alpha + \alpha_d \tag{4.49}$$

If α and α_d go to zero e_θ will go to zero. The error dynamics of s,α,α_d are given by:

$$\begin{pmatrix} \dot{e_s} \\ \dot{\alpha_d} \\ \dot{\alpha} \end{pmatrix} = \begin{pmatrix} -cos(\alpha_d) - sin(\alpha) & -1 \\ \dfrac{sin(\alpha)cos(\alpha + \alpha_d)}{e_s} & 0 \\ \dfrac{sin(\alpha)cos(\alpha + \alpha_d)}{e_s} & -1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.50}$$

Using the control law described by the equations

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} k_s e_s \\ k_\theta\alpha + k_\theta\alpha_d \end{pmatrix} \tag{4.51}$$

The closed loop equation can be found by substituting equation 4.51 into equation 4.52. Linearizing about the equilibrium closed loop equations can be written as

$$\begin{pmatrix} \dot{e_s} \\ \dot{\alpha_d} \\ \dot{\alpha} \end{pmatrix} = \begin{pmatrix} -k_s & -k_\theta & -k_\theta \\ 0 & 0 & k_s \\ 0 & -k_\theta & (k_s - k_{\theta)} \end{pmatrix} \begin{pmatrix} e_s \\ \alpha_d \\ \alpha \end{pmatrix} \tag{4.52}$$

The characteristic polynomial of the matrix has negative real roots if

$$k_s > 0; \quad k_\theta > k_s \tag{4.53}$$

4.4.3 Simulations

The simulations were run in Matlab. The simulation shown below is for $(x_{ref}, y_{ref}, \theta_{ref}) = (100, 100, \frac{pi}{2})$ with initial condition $(x_o, y_o, \theta_o) = (0, 0, 0)$. The block diagram in Figure 4.23 is used as the control structure. Eq 4.43-4.44 are used as the control law with $k_s = [0.1, 0.3, 0.5, 0.7], k_\theta = 0.4$.

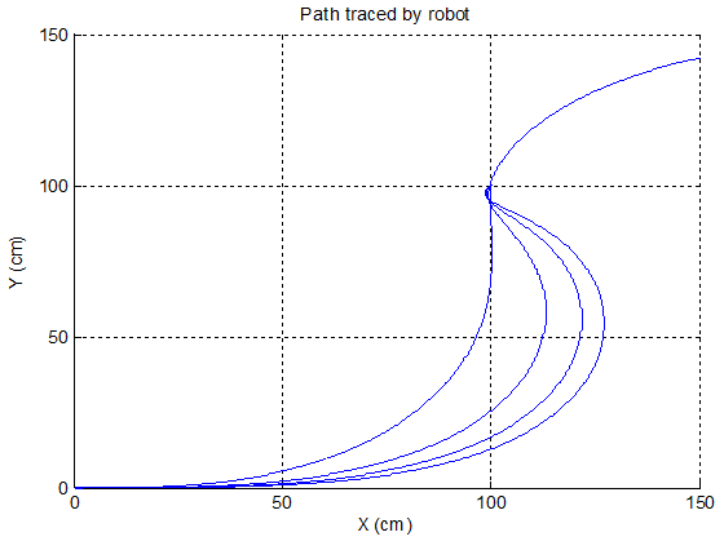

Figure 4.24: x vs y plot

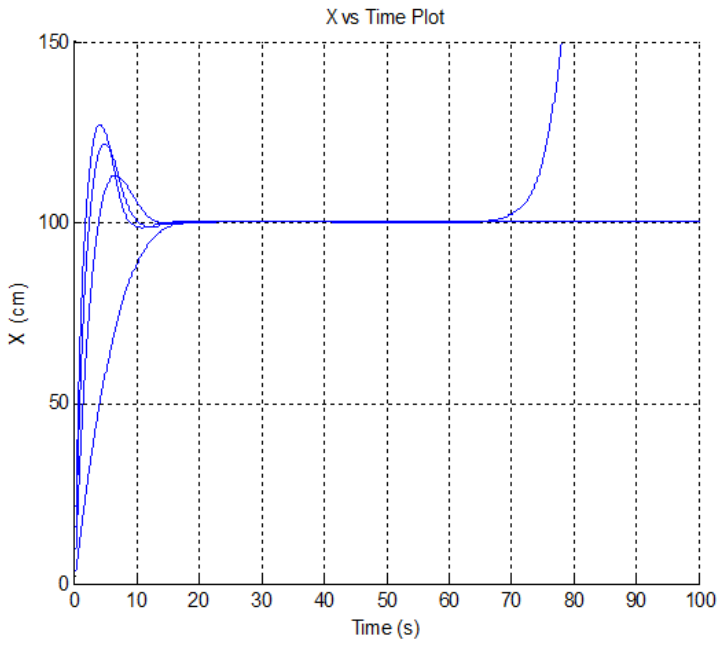

Figure 4.25: x vs time plot

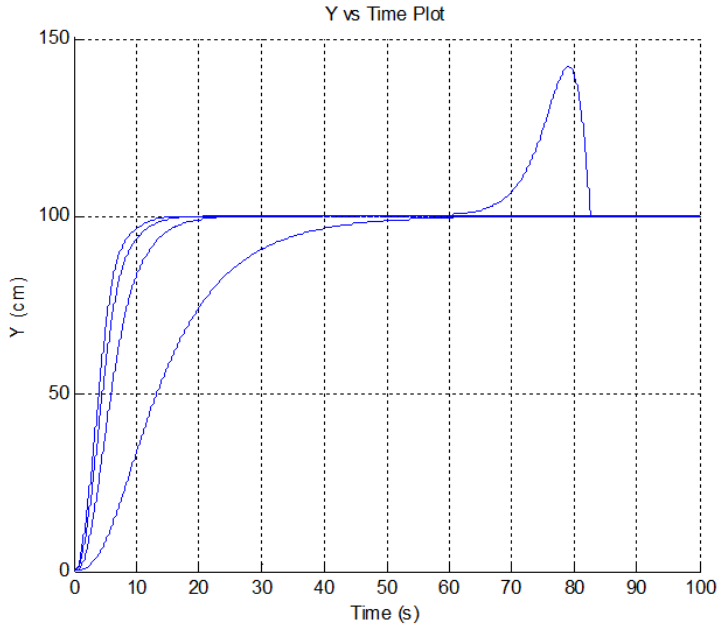

Figure 4.26: y vs x plot

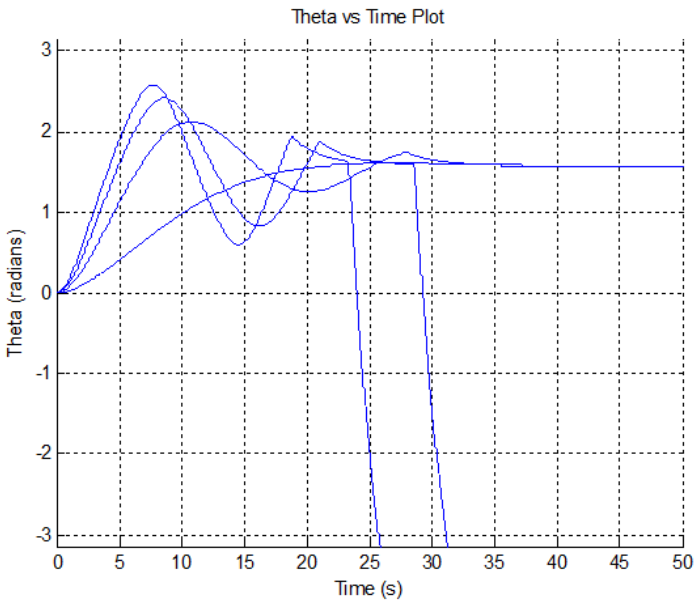

Figure 4.27: θ vs time plot

As can be seen from the above plots that the system is only stable if the local stability conditions are satisfied ; i.e. $k_\theta > k_s > 0$.

4.4.4 Practical Results

The control discussed above was implemented in the AAR robot. The path traced by the robot was compared with simulation. The result shown in figure 4.28 below is for $(x_{ref}y_{ref}, \theta_{ref}) = (100, 100, 0)$ and initial condition $(x_o, y_o, \theta_o) = (0, 0, 0)$ with $k_s = 0.4 and k_\theta = 0.5$.

4.5 Path Tracking

Path tracking is the process of having the robot produce correct linear and angular velocities to trace a path. A path is collection of coordinate in the cartesian space specifying a route.

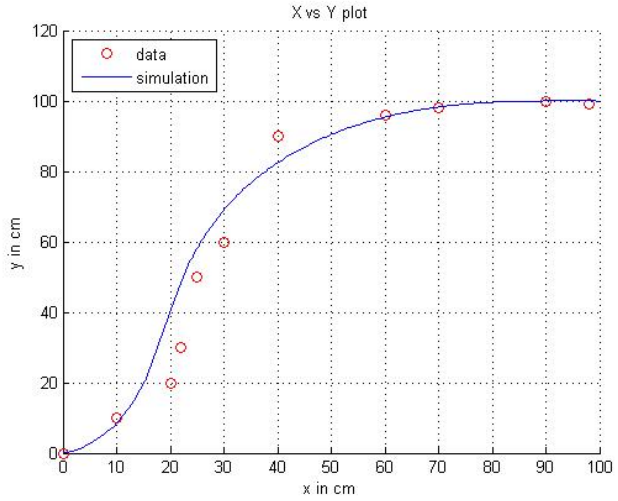

Figure 4.28: Orientation control practical result

4.5.1 Control

In this thesis we consider the path tracking problem to be a subset of the cartesian stabilization problem as in [7]. The cartesian stabilization problem ensures that the vehicle goes to the target point (x_{ref}, y_{ref}). When the vehicle approaches a particular target point in the path the next target point is commanded. Thus the robot can track the specified path.

4.5.2 Simulation

The simulation plot shown in figure 4.29 is for tracking a square of 100X100 units. A look ahead distance of 10 cm is used. $k_s = 0.2$ and $k_\theta = 1$ is used.

4.6 Summary And Conclusions

This chapter described control for cruise control, cartesian and posture stabilization problems. Section 4.1 described cruise control method. Matlab simulation verified the controllers used for cruise control problem. In section 4.2 analysis of kinematic model was done. 4.2 and 4.3 explained the control used for cartesian and

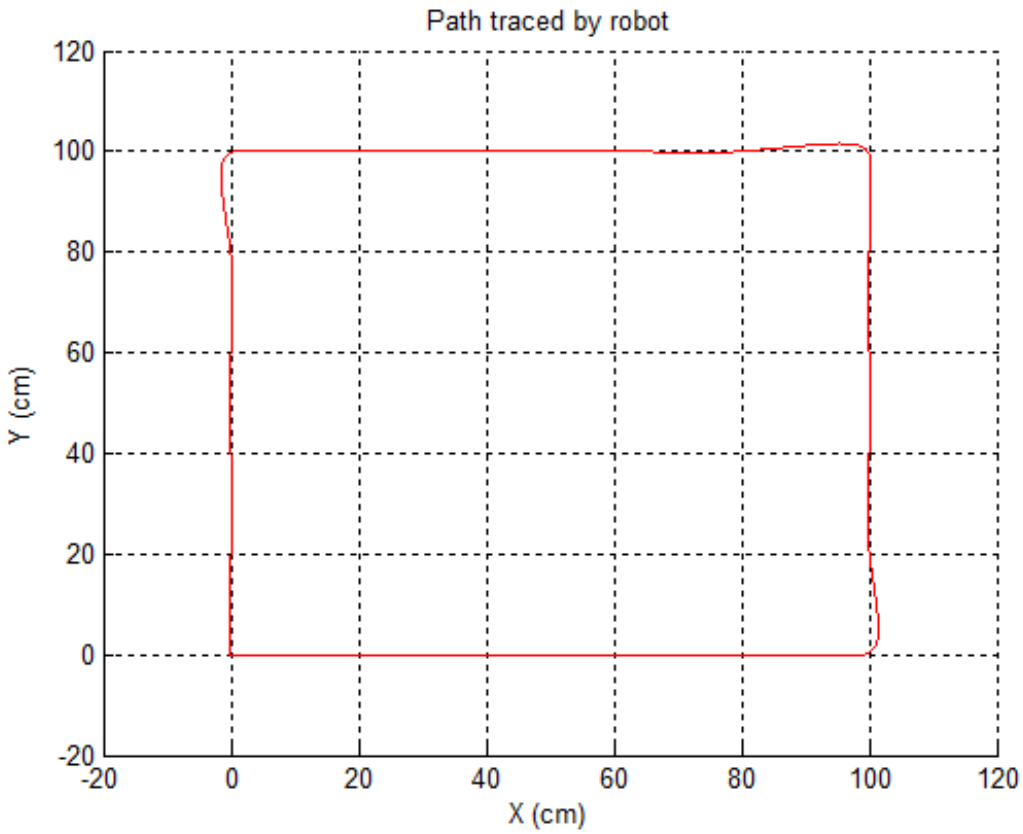

Figure 4.29: Path Tracking

posture stabilization problems. Local stability was proved for both controllers. Hardware results for cartesian and posture stabilization problems were also presented. A brief discussion on path following is done in section 4.5.

Chapter 5

OBSTACLE AVOIDANCE

This chapter discusses obstacle avoidance. One of the most important task for the robot is to reach the destination without hitting obstacles. This chapter looks into the solutions to avoid the obstacle while still going for the goal. This thesis describes some of the easy solutions with simulations to corroborate them.

5.1 Switching Control

Switching control is the most basic control that can be used to avoid obstacle. Switching control method has been discussed by Prof Egersted in his lecture series at Georgia Tech University[19][8]. The cartesian stabilization is used for the go to goal behavior and the controller is switched whenever we want to avoid the obstacle.

In the cartesian control method, e_s and e_θ are obtained from the error between the target position and the current position. This is explained in chapter 4. The control law for the cartesian control method is then defined as

$$v = k_s e_s \tag{5.1}$$

$$\omega = k_\theta e_\theta \tag{5.2}$$

For the go to goal behavior a positive k_s and k_θ will stabilize the system and a θ directing towards the goal will result. If k_θ is negative then the system will be unstable about the target point and the robot will move in a direction opposite to the goal[19].

This idea is used as a method to drive the robot away from the obstacles. The sensors give us the distance from the obstacles or the obstacle coordinates, if this

distance is less than a threshold defined by us, the behavior is switched from go to goal behavior to the avoid obstacle behavior to drive the vehicle away from obstacle. Figure 5.1 below illustrates the situation.

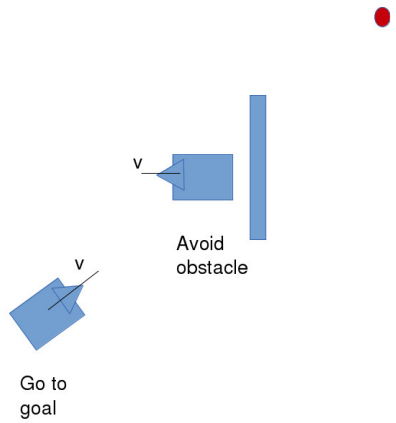

Figure 5.1: Switching control

In case an obstacle is encountered a new β_{ao} is defined which is opposite to the pointing angle to obstacle. Following β_{ao} will take the robot away from the obstacle.

$$\beta_{ao} = -\tan^{-1}(\frac{yo - y}{xo - x}) \tag{5.3}$$

d_o and $e_{\theta o}$ are defined as

$$d_o = \sqrt{(xo - x)^2 + (yo - y)^2} \tag{5.4}$$

$$e_{\theta o} = \beta_{ao} - \theta \tag{5.5}$$

The block diagram is shown in figure 5.2. The control law for the obstacle avoidance behavior is defined as

$$v = k_{d_o} d_o \tag{5.6}$$

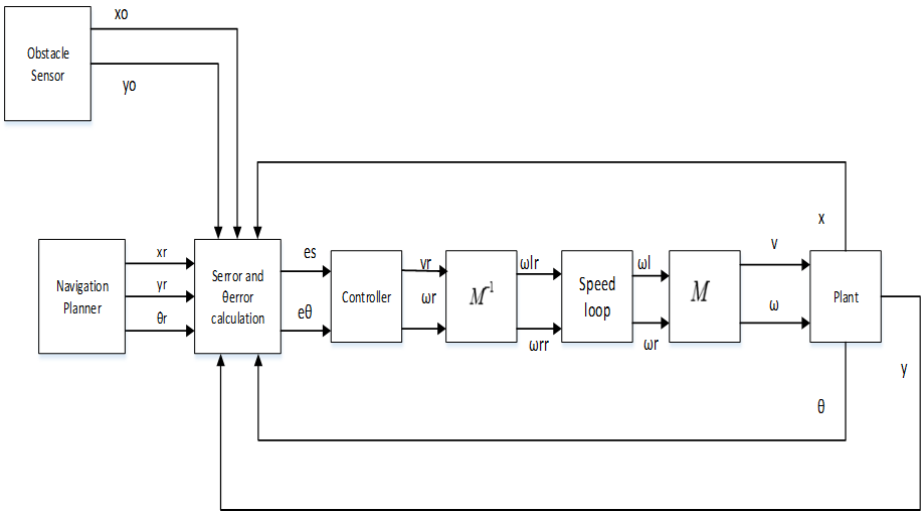

Figure 5.2: Obstacle avoidance block diagram

$$\omega = k_{\theta o} e_{\theta o} \tag{5.7}$$

where x_o and y_o are the nearest obstacle coordinates and d_o is the distance from the obstacle. It can be better understood by the flow chart below.

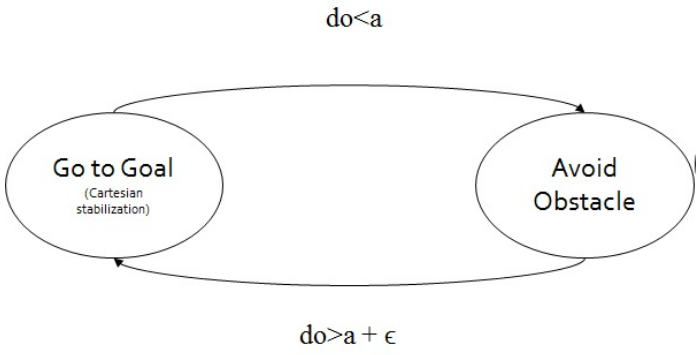

Figure 5.3: Switching Control Flow Chart

5.1.1 Simulation

The control was implemented in the matlab . The obstacle extends from $y = 30 to 70 at x = 50$. Target coordinates are (100,100). The block diagram for the control is as shown figure 5.2. The control law used for the obstacle avoidance behavior is equation 5.5-5.6. The path followed by the robot is shown in plot below.

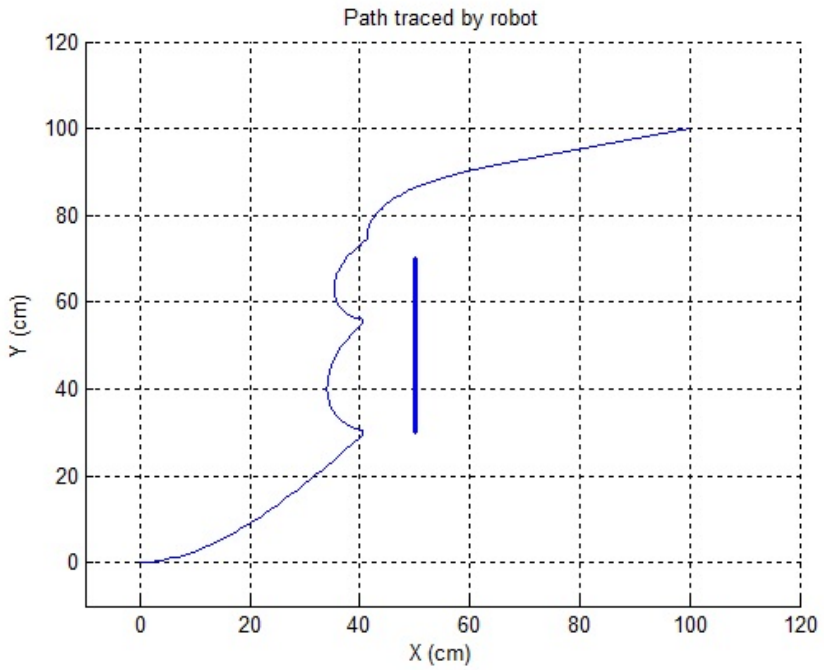

Figure 5.4: Switching Control x vs y Plot

This control works with small obstacles, but produces oscillations in the path. This simple control method fails with concave obstacles.

5.2 Blended Control

This method uses a single control law instead of switched control for the go to goal and avoid obstacle behaviors[19]. As shall be seen this method produces smoother path for the robot.

5.2.1 Control Law

The new control law is defined as the sum of the go to goal behavior and obstacle avoidance behavior[19]. The situation is illustrated in the figure 5.5.The two control laws are added together by multiplying them with a blending function. The blended function itself is a function of the distance from the obstacle.

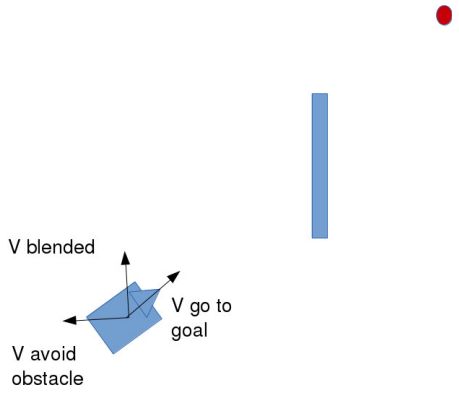

Figure 5.5: Blended Control

We define σ as the blending function. An example of σ is

$$\sigma(d_o) = 1 - \exp(-\gamma d_o) \tag{5.8}$$

where $\sigma(d_o)$ is defined as the blending function. d_o is the distance from obstacle and γ is a user defined value.

It can be seen from the formula of σ that it is an exponential decay, whose decay rate depends upon the value of do(distance from obstacle) and γ.

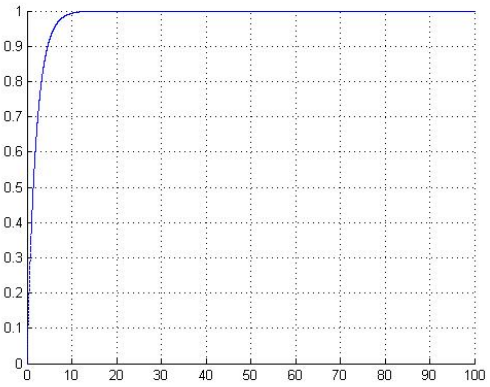

Figure 5.6: σ vs d_o plot at γ =0.5

The control law for the blended obstacle avoidance is define as

$$v = k_s\sigma(do)\sqrt{(xref - x)^2 + (yref - y)^2} + -k_{do}(1 - \sigma(do))\sqrt{(x_o - x)^2 + (y_o - y)^2} \tag{5.9}$$

$$\omega = \sigma(d_o)k_\theta e_\theta + -(1 - \sigma(d_o))k_\theta e_{\theta o} \tag{5.10}$$

5.2.2 Simulation

The control was implemented in the matlab . The obstacle extends from y = 30 to 70 at x = 50. Target coordinates are (100,100). The value of γ used is 0.5. The control law used is given in equation 5.8-5.9. As can be seen this control produces smoother path for the vehicle. However the control doesn't ensures that the vehicle will go to the goal. The method fails in concave obstacles. For avoiding concave obstacles the boundary following algorithm is used which is discussed in the next section.

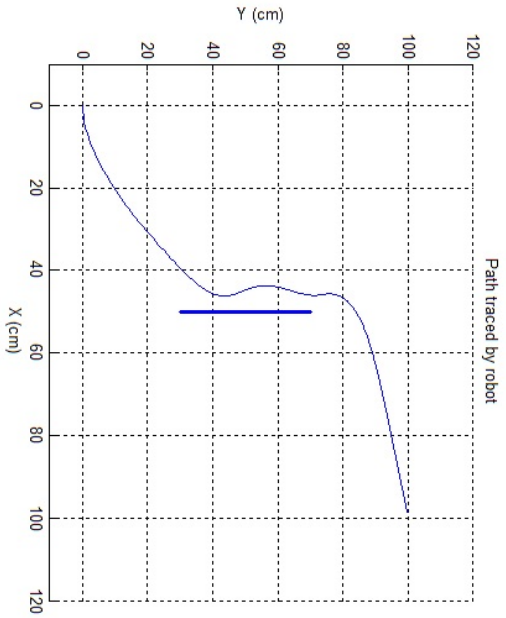

Figure 5.7: Blended Control x vs y plot

5.3 Boundary Following Algorithm

Hard switching and blended control do not work very well with concave and maize type obstacles. Also there is no certainty and there might be oscillations during the robot's path. To overcome these shortcomings the boundary following algorithm is used[19][9].

5.3.1 Algorithm

As the name suggests we want the robot to follow the boundary of the obstacle. The robot then should move towards the goal when it has a clear shot at the goal. When the robot encounters an obstacle it should move in a direction perpendicular to the obstacle coordinate. Thus a new mode is defined which is called boundary following.

If $(d_o > a)$

(go to goal behavior)

else if $(a < d_o < b)$

(follow boundary behavior)

else

(avoid obstacle)

To achieve boundary following behavior the robot need to move perpendicular to the wall. This can be easily accomplished by adding or subtracting pi()/2 from the β_{ao} generated by the avoid obstacle behavior.

$$\beta_{ccw} = \beta_{ao} + pi()/2 \tag{5.11}$$

$$\beta_{cw} = \beta_{ao} - pi()/2 \tag{5.12}$$

+pi()/2 shift achieves a counter clockwise rotation whereas a -pi()/2 shift achieves clockwise rotation. There are no obvious answers to which rotation to make. One

simple method to make this decision is to check whether the goal is above the obstacle or below. If the the goal point is above obstacle make a counter clockwise rotation else make a clock wise rotation. This can be explained in simple language by simple pseudo code line as below.

if atan2$(y_{ref} - y, x_{ref} - x) > 0$
(do a counterclock wise turn)
else
(do a clockwise turn)

The robot can now follow the boundary successfully . But this method has a fallacy. If the obstacle system has some part below and some above the line of goal, then the robot ends up moving up and down along the wall. To overcome this fallacy, a simple change to the above pseudo code is made. The decision whether to go clockwise or anticlockwise is made only once. We stick to the decision made first time. Also once in the boundary following mode, it is important to determine when to leave the mode and go back to go to goal mode. This can be accomplished by defining two new parameters d_t and d_τ.

where d_τ = distance to target at the time of entering wall following mode

d_t = current distance to the target

If d_t is less than d_τ we move back to the go to goal behaviour, else we continue in the boundary following mode. The complete flowchart is shown in the Figure 5.8.

5.3.2 Simulation

The control was implemented in the Matlab. Target coordinates are (100,100). The results are shown in figure 5.9. The algorithm used is given in figure 5.8. As can be seen in the plot this control can avoid maize and concave type obstacles.

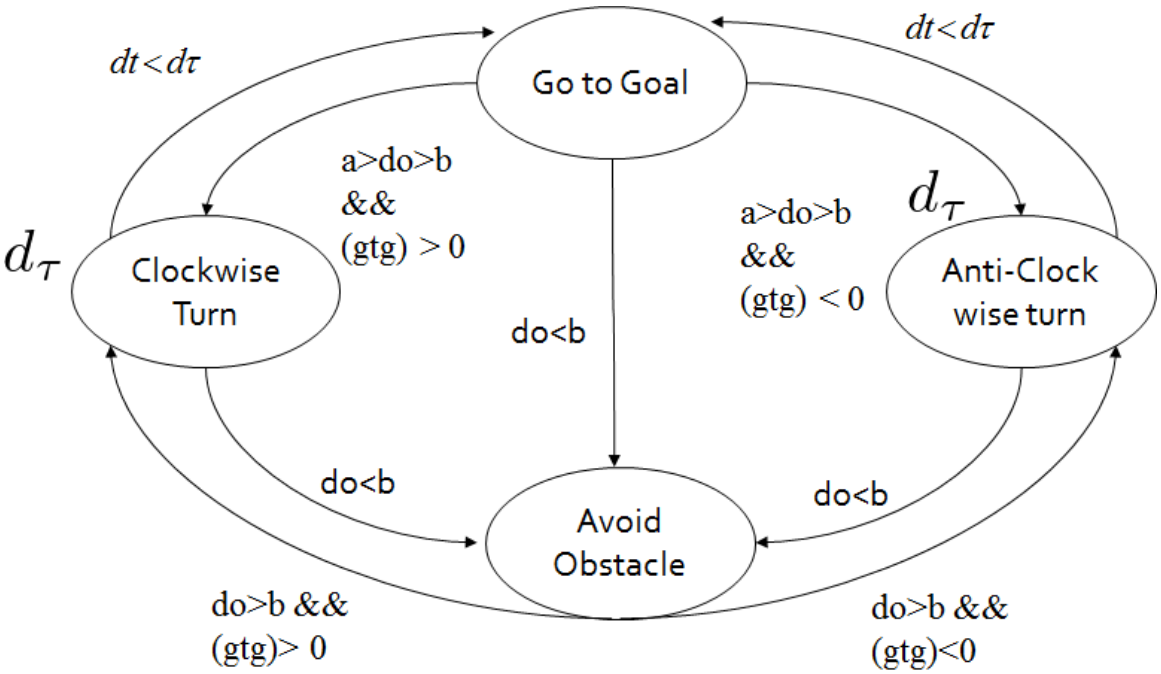

Figure 5.8: Boundary following algorithm

5.4 Summary And Conclusions

This chapter presented three obstacle avoidance methods. Matlab simulations were presented to show the results of the three method. Section 5.1 explained the switched control method. The method only works for small obstacles. This method fails for concave and other complex obstacles. Blended control law was explained in section 5.2. This method produced a smoother path for the robot. But still, blended control also fails for concave and other complex obstacles. Finally boundary following algorithm was detailed in 5.3. Simulation results were presented to show that this algorithm can avoid concave obstacles.

Figure 5.9: Obstacle avoidance using wall following algorithm x vs y plot

Chapter 6

SUMMARY AND DIRECTION FOR FUTURE RESEARCH

6.1 Summary

This thesis discussed control and obstacle avoidance for non-holonomic differential drive mobile vehicles. Go to goal and avoid obstacle are the most basic behaviors for a mobile robot. Both behaviors are discussed in the thesis.

A model of mobile robot consisting of combination of dynamic and kinematic was presented in chapter two. The complete model is given as a combination of kinematics and motor dynamics. This model was used to design controls for cruise control, cartesian and posture stabilization. Matlab simulation were presented to verify the designed controls for these problems.

In chapter five three obstacle avoidance techniques; i.e. switched control, blended control, boundary following were discussed. Switched control and blended control are only suitable for small obstacles. Blended control produces a smoother path. Both switched and blended control method fails for complex obstacles. For more complex obstacles boundary following method was described. Simulations for the three methods were presented in the thesis.

Cartesian and posture stabilization controllers were implemented in AAR robot. These control worked well in the robot. The odometric errors produce an error in the final position. Switched control method for obstacle avoidance was also implemented. This method worked well for small obstacles but fails for bigger and more complex obstacles.

6.2 Directions for Future Research

6.2.1 Localization

The localization method used in the thesis is odometric localization. This method is prone to errors in the position estimation. The error gets accumulated over time. For future work active localization method like G.P.S. or indoor beacons can be used. Modern devices can provide accuracy upto few millimeters. Using indoor localization will increase the cost of the system. But for a highly accurate robot movement active localization is necessary.

6.2.2 Obstacle Avoidance

For obstacle sensing, the current hardware uses a single I.R. sensor. To implement boundary following algorithm multiple ultrasound sensors are needed. Multiple ultrasound sensors will allow the robot to sense the obstacle along all direction. This information can be used to implement boundary following algorithm. More complex obstacle avoidance method's like vector field histogram method shall also be considered for future work.

6.2.3 Additional vehicles

As discussed in chapter one, this thesis is part of Multi Robot System project. The next logical step in to develop more vehicles. These vehicles could be for ground or air operations. These vehicles should be designed so as to work in synchronization with each other. Thus study of swarm behavior for robots is required for future work.

REFERENCES

[1] R. W. Brockett., "Asymptotic stability and feedback stabilization," in R. W. Brockett, R. S. Millman, and H. J. Sussmann, editors,Differential Geometric Control Theory. Birkhauser, Boston, MA, 1983

[2] Astolfi, A., "On the stabilization of nonholonomic systems," Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on , vol.4, no., pp.3481,3486 vol.4, 14-16 Dec 1994

[3] Aicardi, M.; Casalino, G.; Bicchi, A.; Balestrino, A., "Closed loop steering of unicycle like vehicles via Lyapunov techniques," Robotics and Automation Magazine, IEEE , vol.2, no.1, pp.27,35, Mar 1995

[4] De Wit, C.C.; Sordalen, O.J., "Exponential stabilization of mobile robots with nonholonomic constraints," Automatic Control, IEEE Transactions on , vol.37, no.11, pp.1791,1797, Nov 1992

[5] De Wit, C.C.; Khennouf, H.; Samson, C. ; Sordalen, O.J. , "Nonlinear control design for mobile robots," in Recent Trends in Mobile Robots, Y. F. Zheng (Ed.), World Scientific Publisher, 1993

[6] Boyden, F.D.; Velinsky, S.A., "Dynamic modeling of wheeled mobile robots for high load applications," Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on , vol., no., pp.3071,3078 vol.4, 8-13 May 1994

[7] Vieira, F.C.; Medeiros, A.A.D.; Alsina P.J.; Araujo A.P., "Position and orientation control of a two-wheeled differentially driven nonholonomic mobile robot," ICINCO, ,2004. Proceedings.

[8] Egerstedt, M.; Johansson, K.; Lygeros, J.; Sastry, S., "Behavior based robotics using regularized hybrid automata," Decision and Control, 1999. Proceedings of the 38th IEEE Conference on , vol.4, no., pp.3400,3405 vol.4, 1999

[9] Borenstein, J.; Koren, Y., "Real-time obstacle avoidance for fast mobile robots," Systems, Man and Cybernetics, IEEE Transactions on, vol.19, no.5, pp.1179, 1187, Sept.-Oct. 1989

[10] Rodriguez, A.,A. ,"Linear System Analysis and Design", CONTROL3D LLC, 2004.

[11] Araujo, A.D.; Alsina, P.J.; Dias, S.M., "Nonholonomic wheeled mobile robot positioning controller using decoupling and variable structure model reference adaptive control," American Control Conference, 2006 , vol., no., pp.6 pp.,, 14-16 June 2006

[12] Luca, A.; Oriolo, G.; Samson, C.; Laumond, J. P., "Feedback Control of a Nonholomic Car-like Robot" in Robot Motion Planning and Control ,Lectures Notes in Control and Information Sciences, Springer.

[13] D'Andrea-Novel, B.; Bastin, G.; Campion, G., "Dynamic feedback linearization of nonholonomic wheeled mobile robots," Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on , vol., no., pp.2527,2532 vol.3, 12-14 May 1992

[14] Samson, C.; Ait-Abderrahim, K., "Feedback stabilization of a nonholonomic wheeled mobile robot," Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on , vol., no., pp.1242,1247 vol.3, 3-5 Nov 1991

[15] Samson, C; "Time-varying Feedback Stabilization of Car-like Wheeled Mobile Robots" The International Journal of Robotics Research, ISSN 0278-3649, 02/1993, Volume 12, Issue 1, pp. 55 - 64

[16] "Sharp GP2Y0A02YK0F Analog Distance Sensor", n.d. Web. 18 Nov. 2013. ¡http://www.po lolu.com/product/1137¿

[17] "Global Specialties - AAR", n.d. Web. 18 Nov. 2013. <http://www.globalsp ecialties.com/aar.html>

[18] "Mars Exploration Rover", n.d. Web. 18 Nov. 2013. <http://www.marsrover s.jpl.nasa.gov/science/>

[19] Egerstedt, M., "Control of Mobile Robot", Georgia Tech. University, Atlanta, 2013

[20] Siciliano, B.; Sciavicco, L.; Villani, L; Oriolo G., "Robotics: Modeling, Planning, and Control, Springer, 2009

APPENDIX A

CODES

```
const int Aphase =5 ;
const int Aenable =6;
const int Bphase = 9 ;
const int Benable =10 ;
const int EncoderPinA1 =2;
const int EncoderPinB1 =3;
const float positionloop_sampletime = 100 ;
const float speedloop_sampletime = 100 ;


unsigned long lastmilli_l = 0;
unsigned long lastmilli_r = 0;
unsigned long Lmilli_position = 0;
unsigned long Lmilli_speed = 0;
unsigned long pulsewidth_l = 10000 ;
unsigned long pulsewidth_r = 10000 ;

float d = 0 ;
float x = 0 ;
float y = 0 ;
float xt = 100 ;
float yt= 100 ;
float theta = 0 ;
float thetat = 0 ;
float tickr_r = 0;
float tickl_r = 0;


float vx = 0 ;
float vy = 0 ;
float velocity = 0 ;
float wrobot = 0 ;
float omega = 0 ;

float speed_reqr= 0 ;
float speed_reql= 0 ;
float speed_actr = 0;
float speed_actl= 0;

float PWM_valr = 0;
float PWM_vall = 0;

float xprevious = 0 ;
float yprevious = 0 ;
float thetaprevious = 0 ;
float errorx = 0 ;
float errory = 0 ;
```

```
float errortheta = 0 ;


volatile long countr = 0;
volatile long countl =0;


float Kpl = 50 ;
float Kil =  5  ;
float Kpr = 50 ;
float Kir =  5  ;
float Kpo = .4 ;
float Ktheta = 0.7 ;
float Kitheta = 0.001 ;
float error_r=0;
float last_error_r=0;
float pidTerm_r = 0;
float lastpidTerm_r = 0;
float error_l=0;
float last_error_l=0;
float pidTerm_l = 0;
float lasterrortheta = 0 ;
   float lastpidTerm_l = 0;
   float lastomega = 0 ;


float L = 9.9 ;
float R = 1.9 ;
float Dr = 0;
float Dl = 0;
float Dc = 0;

void setup() {
  // initialize the controller
  pinMode(Aenable, OUTPUT);
  pinMode(Aphase, OUTPUT);
  pinMode(Benable, OUTPUT);
  pinMode(Bphase, OUTPUT);
  pinMode(EncoderPinA1, INPUT);
  pinMode(EncoderPinB1, INPUT);
  digitalWrite(EncoderPinA1, HIGH);
  digitalWrite(EncoderPinB1, HIGH);
  attachInterrupt(0, rencoder, RISING);
  attachInterrupt(1, rencoder1, RISING);


  digitalWrite(Aphase, LOW);
  digitalWrite(Bphase, LOW);
  digitalWrite(Benable, LOW);
  digitalWrite(Benable, LOW);
```

```
}

void loop() {

    if(millis()-Lmilli_position>positionloop_sampletime)

    {
      Lmilli_position = millis() ;
      vx = updatevx( xt, x) ;
      vy = updatevy( yt,y) ;
    }

   if(millis()-Lmilli_speed > speedloop_sampletime)
   {
     Lmilli_speed = millis() ;

     if(atan2(yt,xt)>0)
     {
      thetat  = abs(atan2(vy,vx)) ;
     }
     else
     {  thetat  = -1*abs(atan2(vy,vx));
     }

      d = sqrt((xt-x)*(xt-x) + (yt-y)*(yt-y)) ;

       velocity = sqrt(vy*vy + vx*vx) ;
     wrobot  = update_w(thetat,theta) ;

      if(velocity>10)
      { velocity = 10 ;
      }
       if (d<4)
      {
     velocity = 0 ;
     wrobot  =  0 ;
     }
      speed_reqr = (2*velocity + wrobot*L)/(2*R*2*3.14) ;
      speed_reql = (2*velocity - wrobot*L)/(2*R*2*3.14) ;

      if(speed_reqr<0)
      {speed_reqr = 0 ;
      }

     if(speed_reql<0)

     {speed_reql = 0 ;
      }

     PWM_vall= updatePidl(PWM_vall, speed_reql, countl);
```

```
     PWM_valr= PWM_vall ;

       if(PWM_vall<0)
      {
        digitalWrite(Bphase,LOW);
        analogWrite(Benable,PWM_vall);
      }
      else
     {
        digitalWrite(Benable,LOW);
        analogWrite(Bphase,PWM_vall);
     }

    PWM_valr= updatePidr(PWM_valr, speed_reqr, countr);

       if(PWM_valr<0)
      {
        digitalWrite(Aphase,LOW);
        analogWrite(Aenable,PWM_valr);
      }
      else
     {
     digitalWrite(Aenable,LOW);
     analogWrite(Aphase,PWM_valr);
     }

     Dr = 2*3.14*R*countr/20 ;
     Dl = 2*3.14*R*countl/20 ;
     Dc = (Dr+Dl)/2 ;
     theta = thetaprevious + (Dr-Dl)/L ;
     x = xprevious + Dc*cos(theta) ;
     y = yprevious + Dc*sin(theta) ;
    xprevious = x ;
    yprevious = y ;
    thetaprevious = theta ;
    countr = 0 ;
    countl = 0;
   }

}


float updatevx(float targetvalue_x , float currentvalue_x)
{
  errorx = targetvalue_x - currentvalue_x ;
  vx = Kpo*errorx ;
  return(vx) ;
}

float updatevy(float targetvalue_y , float currentvalue_y)
```

```
{
  errory = targetvalue_y - currentvalue_y ;
  vy = Kpo*errory ;
  return(vy) ;
}

float update_w(float targetvalue_theta , float currentvalue_theta)
{
  errortheta = targetvalue_theta - currentvalue_theta ;
  omega = lastomega + Ktheta*(errortheta-lasterrortheta) +
      Kitheta*lasterrortheta ;
  lastomega = omega ;
  lasterrortheta = errortheta ;
  return(omega) ;
}


float updatePidr(int command1,float targetValue1, float currentValue1) {

  tickr_r = targetValue1*0.02*speedloop_sampletime ;
  error_r = tickr_r - currentValue1 ;
  pidTerm_r =lastpidTerm_r + (Kpr*(error_r - last_error_r)) +
      Kir*last_error_r ;
  if(pidTerm_r>255)
  pidTerm_r = 255 ;
  else if(pidTerm_r<0)
  pidTerm_r = 0 ;
  last_error_r = error_r;
  lastpidTerm_r = pidTerm_r ;
  return(pidTerm_r*1) ;
}

float updatePidl(float command2, float targetValue2, float currentValue2) {
  // compute PWM value
  tickl_r = targetValue2*.02*speedloop_sampletime ;
 error_l = tickl_r - currentValue2;
 pidTerm_l = lastpidTerm_l + Kpl*(error_l-last_error_l) +
     Kil*(last_error_l);
 if (pidTerm_l>255)
 pidTerm_l = 255 ;
 else if(pidTerm_l<0)
 pidTerm_l = 0 ;
  last_error_l = error_l ;
  lastpidTerm_l = pidTerm_l ;
  return(pidTerm_l*1);

}

void rencoder()
{
```

```
countr = countr + 1;
}

void rencoder1()
{
countl = countl + 1 ;

}
```

A.2 Posture Stabilization Arduino Code

```
const int Aphase =5 ;
const int Aenable =6;
const int Bphase = 9 ;
const int Benable =10 ;
const int EncoderPinA1 =2;
const int EncoderPinB1 =3;
const float positionloop_sampletime = 100 ;
const float speedloop_sampletime = 100 ;


unsigned long lastmilli_l = 0;
unsigned long lastmilli_r = 0;
unsigned long Lmilli_position = 0;
unsigned long Lmilli_speed = 0;
unsigned long pulsewidth_l = 10000 ;
unsigned long pulsewidth_r = 10000 ;

float x = 0 ;
float y = 0 ;
float xd = 100;
float yd= 100 ;
float xt = 0;
float yt= 0 ;
float d = 0 ;
float theta = 0 ;
float thetad = 0 ;
float thetat = 0 ;
float tickr_r = 0;
float tickl_r = 0;


float vx = 0 ;
float vy = 0 ;
float velocity = 0 ;
float wrobot = 0 ;
float omega1 = 0 ;
float omega2 = 0 ;
```

```
float beta = 0 ;
float alpha = 0 ;


float speed_reqr= 0 ;
float speed_reql= 0 ;
float speed_actr = 0;
float speed_actl= 0;

float PWM_valr = 0;
float PWM_vall = 0;

float xprevious = 0 ;
float yprevious = 0 ;
float thetaprevious = 0 ;
float errorx = 0 ;
float errory = 0 ;
float errortheta1 = 0 ;
float errortheta2 = 0 ;


volatile long countr = 0;
volatile long countl =0;


float Kpl = 50 ;
float Kil =  5  ;
float Kpr = 50 ;
float Kir =  5  ;
float Kpo = .4 ;
float Ktheta1 = 1;
float Kitheta1 = 0.001 ;
float Ktheta2 = 1;
float Kitheta2 = 0.001 ;
float error_r=0;
float last_error_r=0;
float pidTerm_r = 0;
float lastpidTerm_r = 0;
float error_l=0;
float last_error_l=0;
float pidTerm_l = 0;
float lasterrortheta1 = 0 ;
   float lastpidTerm_l = 0;
   float lastomega1 = 0 ;
float lasterrortheta2 = 0 ;
 float lastomega2 = 0 ;

float L = 10 ;
float R = 2 ;
float Dr = 0;
```

```
float Dl = 0;
float Dc = 0;

void setup() {
  // initialize the controller
  pinMode(Aenable, OUTPUT);
  pinMode(Aphase, OUTPUT);
  pinMode(Benable, OUTPUT);
  pinMode(Bphase, OUTPUT);
  pinMode(EncoderPinA1, INPUT);
  pinMode(EncoderPinB1, INPUT);
  digitalWrite(EncoderPinA1, HIGH);
  digitalWrite(EncoderPinB1, HIGH);
  attachInterrupt(0, rencoder, RISING);
  attachInterrupt(1, rencoder1, RISING);


  digitalWrite(Aphase, LOW);
  digitalWrite(Bphase, LOW);
  digitalWrite(Benable, LOW);
  digitalWrite(Benable, LOW);
}

void loop() {

    if(millis()-Lmilli_position>positionloop_sampletime)

    {
      Lmilli_position = millis() ;
      beta    =  atan2(yd-y,xd-x) ;
      alpha   =  beta - thetad ;
      d = sqrt((xd-x)*(xd-x) + (yd-y)*(yd-y)) ;
      xt = x + d*cos(alpha) ;
      yt = y + d*cos(alpha) ;
      vx = updatevx( xt, x) ;
      vy = updatevy( yt,y) ;

    }



   if(millis()-Lmilli_speed > speedloop_sampletime)
   {
     Lmilli_speed = millis() ;
      velocity = sqrt(vx*vx + vy*vy) ;
       beta    =  atan2(yd-y,xd-x) ;
      thetat   = atan2(vy,vx) ;
     wrobot   = update_w1(beta,theta) + update_w2(beta,thetad) ;

      if(velocity>10)
```

```
 { velocity = 10 ;
 }

 if(d<4)
 {   velocity = 0 ;
     wrobot  = 0 ;
 }

 speed_reqr = (2*velocity + wrobot*L)/(2*R*2*3.14) ;
 speed_reql = (2*velocity - wrobot*L)/(2*R*2*3.14) ;

PWM_vall= updatePidl(PWM_vall, speed_reql, countl);
PWM_valr= PWM_vall ;

  if(PWM_vall<0)
 {
   digitalWrite(Bphase,LOW);
   analogWrite(Benable,PWM_vall);
 }
 else
{
   digitalWrite(Benable,LOW);
   analogWrite(Bphase,PWM_vall);
}


PWM_valr= updatePidr(PWM_valr, speed_reqr, countr);


 if(PWM_valr<0)
 {
   digitalWrite(Aphase,LOW);
   analogWrite(Aenable,PWM_valr);
 }
 else
{
digitalWrite(Aenable,LOW);
analogWrite(Aphase,PWM_valr);
}

Dr = 2*3.14*R*countr/20 ;
Dl = 2*3.14*R*countl/20 ;
Dc = (Dr+Dl)/2 ;
theta = thetaprevious + (Dr-Dl)/L ;
x = xprevious + Dc*cos(theta) ;
y = yprevious + Dc*sin(theta) ;
xprevious = x ;
yprevious = y ;
thetaprevious = theta ;
countr = 0 ;
```

```
    countl = 0;
   }

}

float updatevx(float targetvalue_x , float currentvalue_x)
{
  errorx = targetvalue_x - currentvalue_x ;
  vx = Kpo*errorx ;
  return(vx) ;
}

float updatevy(float targetvalue_y , float currentvalue_y)
{
  errory = targetvalue_y - currentvalue_y ;
  vy = Kpo*errory ;
  return(vy) ;
}

float update_w1(float targetvalue_theta1 , float currentvalue_theta1)
{
  errortheta1 = targetvalue_theta1 - currentvalue_theta1 ;
  omega1 = lastomega1 + Ktheta1*(errortheta1-lasterrortheta1) +
      Kitheta1*lasterrortheta1 ;
  lastomega1 = omega1 ;
  lasterrortheta1 = errortheta1 ;
  return(omega1) ;
}

float update_w2(float targetvalue_theta2 , float currentvalue_theta2)
{
  errortheta2 = targetvalue_theta2 - currentvalue_theta2 ;
  omega2 = lastomega2 + Ktheta2*(errortheta2-lasterrortheta2) +
      Kitheta2*lasterrortheta2 ;
  lastomega2 = omega2 ;
  lasterrortheta2 = errortheta2 ;
  return(omega2) ;
}


float updatePidr(int command1,float targetValue1, float currentValue1) {

  tickr_r = targetValue1*0.02*speedloop_sampletime ;
  error_r = tickr_r - currentValue1 ;
  pidTerm_r =lastpidTerm_r + (Kpr*(error_r - last_error_r)) +
      Kir*last_error_r ;
  if(pidTerm_r>255)
  pidTerm_r = 255 ;
  else if(pidTerm_r<0)
  pidTerm_r = 0 ;
```

```
  last_error_r = error_r;
  lastpidTerm_r = pidTerm_r ;
  return(pidTerm_r*1) ;
}

float updatePidl(float command2, float targetValue2, float currentValue2) {
  // compute PWM value
  tickl_r = targetValue2*.02*speedloop_sampletime ;
 error_l = tickl_r - currentValue2;
 pidTerm_l = lastpidTerm_l + Kpl*(error_l-last_error_l) +
     Kil*(last_error_l);
 if (pidTerm_l>255)
 pidTerm_l = 255 ;
 else if(pidTerm_l<0)
 pidTerm_l = 0 ;
  last_error_l = error_l ;
  lastpidTerm_l = pidTerm_l ;
  return(pidTerm_l*1);

}

void rencoder()
{
countr = countr + 1;
}

void rencoder1()
{
countl = countl + 1 ;

}
```

A.3 Obstacle Avoidance Arduino Code

```
const int Aphase =5 ;
const int Aenable =6;
const int Bphase = 9 ;
const int Benable =10 ;
const int EncoderPinA1 =2;
const int EncoderPinB1 =3;
const float positionloop_sampletime = 100 ;
const float speedloop_sampletime = 100 ;
const int sensor = A3 ;


unsigned long lastmilli_l = 0;
unsigned long lastmilli_r = 0;
```

```
unsigned long Lmilli_position = 0;
unsigned long Lmilli_speed = 0;
unsigned long pulsewidth_l = 10000 ;
unsigned long pulsewidth_r = 10000 ;

float dobt = 0 ;
float d = 0 ;
float x = 0 ;
float y = 0 ;
float xt = 100 ;
float yt= 100 ;
float theta = 0 ;
float thetat = 0 ;
float tickr_r = 0;
float tickl_r = 0;


float vx = 0 ;
float vy = 0 ;
float velocity = 0 ;
float wrobot = 0 ;
float omega = 0 ;

float speed_reqr= 0 ;
float speed_reql= 0 ;
float speed_actr = 0;
float speed_actl= 0;

float PWM_valr = 0;
float PWM_vall = 0;

float xprevious = 0 ;
float yprevious = 0 ;
float thetaprevious = 0 ;
float errorx = 0 ;
float errory = 0 ;
float errortheta = 0 ;


volatile long countr = 0;
volatile long countl =0;


float Kpl = 50 ;
float Kil =  5  ;
float Kpr = 50 ;
float Kir =  5  ;
float Kpo = .3 ;
float Ktheta = 1 ;
float Kitheta = 0.001 ;
```

```
float error_r=0;
float last_error_r=0;
float pidTerm_r = 0;
float lastpidTerm_r = 0;
float error_l=0;
float last_error_l=0;
float pidTerm_l = 0;
float lasterrortheta = 0 ;
   float lastpidTerm_l = 0;
   float lastomega = 0 ;
float xo = 200 ;
float yo = 200 ;
float dobstacle = 0 ;
float dobstacle1 = 0 ;
float val = 0 ;

float L = 9.9 ;
float R = 1.9 ;
float Dr = 0;
float Dl = 0;
float Dc = 0;


void setup() {
  // initialize the controller
  pinMode(Aenable, OUTPUT);
  pinMode(Aphase, OUTPUT);
  pinMode(Benable, OUTPUT);
  pinMode(Bphase, OUTPUT);
  pinMode(EncoderPinA1, INPUT);
  pinMode(EncoderPinB1, INPUT);
  digitalWrite(EncoderPinA1, HIGH);
  digitalWrite(EncoderPinB1, HIGH);
  attachInterrupt(0, rencoder, RISING);
  attachInterrupt(1, rencoder1, RISING);


  digitalWrite(Aphase, LOW);
  digitalWrite(Bphase, LOW);
  digitalWrite(Benable, LOW);
  digitalWrite(Benable, LOW);
}

void loop() {

    if(millis()-Lmilli_position>positionloop_sampletime)

    {
      Lmilli_position = millis() ;
      vx = updatevx( xt, x) ;
```

```
    vy = updatevy( yt,y) ;

  }

 if(millis()-Lmilli_speed > speedloop_sampletime)
 {
   Lmilli_speed = millis() ;
   val = analogRead(sensor);


   if (val > 400)
  { dobstacle = 20 ;
    xo = x + dobstacle*cos(theta) ;
    yo = y + dobstacle*sin(theta) ;
    dobt = sqrt((xt-x)*(xt-x) + (yt-x)*(yt-y)) ;
  }

d = sqrt((xt-x)*(xt-x) + (yt-y)*(yt-y)) ;

 dobstacle1 = sqrt((xo-x)*(xo-x) + (yo-y)*(yo-y)) ;

  if (dobstacle1 > 20)
  {
   if(atan2(yt,xt)>0)
   {
    thetat  = abs(atan2(vy,vx)) ;
   }
   else
   {  thetat  = -1*abs(atan2(vy,vx));
   }
  }
  else
  {
 if (dobt > d +10)
   {
    thetat  = atan2(vy,vx) ;
   }
    else

   {
     thetat = -1*atan2(y-yo,x-xo) ;

   }
  }
    velocity = sqrt(vy*vy + vx*vx) ;
    wrobot  = update_w(thetat,theta) ;

    if(velocity>10)
    { velocity = 10 ;
    }
```

```
 if(d<4)
 {   velocity = 0 ;
     wrobot  = 0 ;
 }

 speed_reqr = (2*velocity + wrobot*L)/(2*R*2*3.14) ;
 speed_reql = (2*velocity - wrobot*L)/(2*R*2*3.14) ;

 if(speed_reqr<0)
 {speed_reqr = 0 ;
 }

if(speed_reql<0)

{speed_reql = 0 ;
 }

PWM_vall= updatePidl(PWM_vall, speed_reql, countl);
PWM_valr= PWM_vall ;

  if(PWM_vall<0)
 {
   digitalWrite(Bphase,LOW);
   analogWrite(Benable,PWM_vall);
 }
 else
{
   digitalWrite(Benable,LOW);
   analogWrite(Bphase,PWM_vall);
}


PWM_valr= updatePidr(PWM_valr, speed_reqr, countr);


 if(PWM_valr<0)
 {
   digitalWrite(Aphase,LOW);
   analogWrite(Aenable,PWM_valr);
 }
 else
{
digitalWrite(Aenable,LOW);
analogWrite(Aphase,PWM_valr);
}

Dr = 2*3.14*R*countr/20 ;
Dl = 2*3.14*R*countl/20 ;
Dc = (Dr+Dl)/2 ;
```

```
     theta = thetaprevious + (Dr-Dl)/L ;
     x = xprevious + Dc*cos(theta) ;
     y = yprevious + Dc*sin(theta) ;
    xprevious = x ;
    yprevious = y ;
    thetaprevious = theta ;
    countr = 0 ;
    countl = 0;
   }

}


float updatevx(float targetvalue_x , float currentvalue_x)
{
  errorx = targetvalue_x - currentvalue_x ;
  vx = Kpo*errorx ;
  return(vx) ;
}

float updatevy(float targetvalue_y , float currentvalue_y)
{
  errory = targetvalue_y - currentvalue_y ;
  vy = Kpo*errory ;
  return(vy) ;
}

float update_w(float targetvalue_theta , float currentvalue_theta)
{
  errortheta = targetvalue_theta - currentvalue_theta ;
  omega = lastomega + Ktheta*(errortheta-lasterrortheta) +
      Kitheta*lasterrortheta ;
  lastomega = omega ;
  lasterrortheta = errortheta ;
  return(omega) ;
}


float updatePidr(int command1,float targetValue1, float currentValue1) {

  tickr_r = targetValue1*0.02*speedloop_sampletime ;
  error_r = tickr_r - currentValue1 ;
  pidTerm_r =lastpidTerm_r + (Kpr*(error_r - last_error_r)) +
      Kir*last_error_r ;
  if(pidTerm_r>255)
  pidTerm_r = 255 ;
  else if(pidTerm_r<0)
  pidTerm_r = 0 ;
  last_error_r = error_r;
  lastpidTerm_r = pidTerm_r ;
```

```
  return(pidTerm_r*1) ;
}

float updatePidl(float command2, float targetValue2, float currentValue2) {
  // compute PWM value
  tickl_r = targetValue2*.02*speedloop_sampletime ;
 error_l = tickl_r - currentValue2;
 pidTerm_l = lastpidTerm_l + Kpl*(error_l-last_error_l) +
     Kil*(last_error_l);
 if (pidTerm_l>255)
 pidTerm_l = 255 ;
 else if(pidTerm_l<0)
 pidTerm_l = 0 ;
  last_error_l = error_l ;
  lastpidTerm_l = pidTerm_l ;
  return(pidTerm_l*1);

}

void rencoder()
{
countr = countr + 1;
}

void rencoder1()
{
countl = countl + 1 ;

}
```

A.4 Cartesian Stabilization Matlab Code

```
clc ;
clear all ;
close all ;
m=0;
k1 = 10;
n= 0 ;
xn = zeros(4,10000) ;
yn = zeros(4,10000);
Tnew = zeros(4,10000) ;
thetan = zeros(4,10000) ;
v =  zeros(4,10000) ;
w = zeros(4,10000) ;

for K=0.1:0.2:0.7
    m= m+1 ;
```

```
    e1 = 0;
    l = 0;
e2 = 0;
eo1 = 0;
eo2 = 0;
xf = 0;
yf = 0;
E = [e1 ; e2] ;
U = [0 ; 0] ;
 theta1 = 0;
 xp = 0;
 yp = 0 ;
 thetap = 0 ;
 setheta = 0;
 edisp = 0 ;

xt = 100 ;
yt = 100 ;
kp = .5 ;
ki = 0 ;
DelT = 0.001 ;
c = 5 ;
d = 5 ;
n = 0 ;
lastw = 0;


while(n<100000)
    n=n+1;
    Tnew(m,n+1) = DelT*n ;


    E(1)= xt - xn(m,n) ;
    E(2) = yt - yn(m,n) ;

    thetar = atan2(E(2),E(1)) ;
    d = sqrt(E(1)^2 + E(2)^2);
    etheta = thetar - thetan(m,n) ;
    edisp = sqrt(E(1)^2 + E(2)^2)*cos(etheta) ;

    v(m,n+1) = K*edisp ;
    w(m,n+1) = kp*etheta ;

   if (d==0)
     w(m,n+1)= 0 ;
     v(m,n+1) = 0 ;
   end

    Tnew(m,n+1) = DelT*n ;
```

```
    thetan(m,n+1) = w(m,n+1)*DelT + thetap ;
    xndot = (v(m,n+1)*cos(thetan(m,n)));
    yndot = (v(m,n+1)*sin(thetan(m,n)));

    xn(m,n+1) = xndot*DelT + xp ;
    yn(m,n+1) = yndot*DelT + yp ;
    xp = xn(m,n+1) ;
    yp = yn(m,n+1) ;
    thetap= thetan(m,n+1) ;

end
end
```

A.5 Posture Stabilization Matlab Code

```
clc ;
clear all ;
close all ;
m=0;
k1 = 10;
n=0;
xn = zeros(4,5000) ;
yn = zeros(4,5000);

Tnew = zeros(4,5000) ;
thetan = zeros(4,5000) ;
thetar = zeros(4,5000) ;
v = zeros(4,5000) ;
w = zeros(4,5000) ;

for K= 0.1:0.2:0.7
    m= m+1 ;
    e1 = 0;
    l = 0;
e2 = 0;
eo1 = 0;
eo2 = 0;
xf = 0;
yf = 0;
E = [e1 ; e2] ;
U = [0 ; 0] ;
 theta1 = 0;
 xp = 0;
 yp = 0 ;
 edisp = 0 ;
 thetap = 0 ;
 setheta = 0;
xd = 100 ;
```

```
yd = 100 ;
thetad = pi()/2 ;
kp = .4 ;
kp1 = 1 ;
ki = 0 ;
DelT = 0.001 ;
c = 5 ;
d = 5 ;
n = 0 ;
lastw = 0;
xt = 0 ;
yt = 0 ;

while(n<100000)
    n=n+1;

    beta = atan2(yd-yn(m,n),xd-xn(m,n));
    alpha = beta - thetad ;
    gamma = beta - thetan(m,n) ;
    d = sqrt((yd-yn(m,n))^2 + (xd-xn(m,n))^2) ;
    xt = xn(m,n) + d*cos(beta + alpha);
    yt = yn(m,n) + d*sin(beta + alpha) ;


    E(1)= xt - xn(m,n) ;
    E(2) = yt - yn(m,n) ;

    thetar(m,n) = atan2(E(2),E(1)) ;


    %thetar(m,n) = 2*beta - thetad ;

  etheta = thetar(m,n) - thetan(m,n) ;

    edisp = sqrt(E(1)^2 + E(2)^2)*cos(etheta) ;

   v(m,n+1) = K*edisp ;



 % if (v(m,n) < 0)

% v(m,n) = 0 ;
 % end

%  if (v(m,n) > 10)

% v(m,n) = 10 ;
%  end
```

```
   % w = lastw + kp*(etheta-setheta) + ki*setheta ;
   %  w = kp*gamma + kp1*alpha ;
   w(m,n+1) = kp*etheta ;

    setheta = etheta ;
    lastw = w(m,n+1) ;

   if (d==0)
     w(m,n+1)= kp*(thetad - thetan(m,n)) ;
     v(m,n+1) = 0 ;
   end

    Tnew(m,n+1) = DelT*n ;

    thetan(m,n+1) = w(m,n+1)*DelT + thetap ;

    xndot = (v(m,n+1)*cos(thetan(m,n)));
    yndot = (v(m,n+1)*sin(thetan(m,n)));


    xn(m,n+1) = xndot*DelT + xp ;
    yn(m,n+1) = yndot*DelT + yp ;

    xp = xn(m,n+1) ;
    yp = yn(m,n+1) ;
    thetap= thetan(m,n+1) ;

end
end
```

A.6 Switched Control Matlab Code

```
 clc ;
clear all ;
close all ;
m=0;
k1 = 10;
n=0;
xn = zeros(5000,1) ;
yn = zeros(5000,1);
Tnew = zeros(5000,1) ;
K = 0.1 ;
Ko = 0.1 ;
m= m+1 ;
e1 = 0;
l = 0;
e2 = 0;
```

```
eo1 = 0;
eo2 = 0;
xf = 0;
yf = 0;
E = [e1 ; e2] ;
U = [0 ; 0] ;
 thetan = 0 ;
 theta1 = 0;
 xp = 0;
 yp = 0;
 thetap = 0 ;
 setheta = 0;
xt = 100;
yt = 100 ;
kp = .7 ;
ki = 0 ;
DelT = 0.1 ;
c = 5 ;
d = 5 ;
d1 = 100 ;
n = 0 ;
lastw = 0;
while(n<40000)
    n=n+1;


     xo = 50 ;
     d1 = 100 ;
    for yo=30:0.01:70 ;
    d= sqrt((xo-xn(n,1))*(xo-xn(n,1)) + (yo-yn(n,1))*(yo-yn(n,1))) ;
  if d<d1
      d1=d ;
      xf = xo ;
      yf = yo ;
  end
    end

   E(1)= xt - xn(n,1) ;
    E(2) = yt - yn(n,1) ;
    Eo(1) = xf - xn(n,1) ;
    Eo(2) = yf -yn(n,1) ;
   % K(1) = (vo*(1-exp(abs(E(1))^2*-2)))/abs(E(1)) ;
   % K(2) = (vo*(1-exp(abs(E(2))^2*-2)))/abs(E(2)) ;

   if d1>10

   v = K*sqrt( E(1)^2 + E(2)^2) ;
   w = kp*(atan2(E(2),E(1))- thetan) ;

   elseif d1<=12
```

```
    v = Ko*sqrt( Eo(1)^2 + Eo(2)^2) ;
    w = -1*kp*(atan2(Eo(2),Eo(1))- thetan) ;

   end

     if ( v>0)
    v = min(10,v) ;
    else
    v = 0 ;
    end

    Tnew(n+1,1) = DelT*n ;

    thetan = w*DelT + thetap ;
    thetap= thetan ;
    xndot = (v*cos(thetan));
    yndot = (v*sin(thetan));

    xn(n+1,1) = xndot*DelT + xp ;
    yn(n+1,1) = yndot*DelT + yp ;
    xp = xn(n+1,1) ;
    yp = yn(n+1,1) ;

end
```

A.7 Blending Control Matlab Code

```
 clc ;
clear all ;
close all ;
m=0;
k1 = 10;
n=0;
xn = 0 ;
yn = 0 ;
Tnew = 0 ;
K = 0.1 ;
Ko = 0.5 ;

    e1 = 0;
    l = 0;
e2 = 0;
eo1 = 0;
eo2 = 0;
xf = 0;
yf = 0;
E = [e1 ; e2] ;
```

```
U = [0 ; 0] ;
 thetan = 0 ;
 theta1 = 0;
 xp = 0;
 yp = 0;
 thetap = 0 ;
 setheta = 0;
xt = 100;
yt = 100 ;
kp = 1 ;
ki = 0 ;
DelT = 0.1 ;
c = 5 ;
d = 5 ;
d1 = 100 ;
n = 0 ;
lastw = 0;
beta = .2 ;



while(n<600)
    n=n+1;


     xo = 50 ;
     d1 = 100 ;
    for yo=30:0.01:70 ;
    d= sqrt((xo-xn(n))*(xo-xn(n)) + (yo-yn(n))*(yo-yn(n))) ;
  if d<d1
      d1=d ;
      xf = xo ;
      yf = yo ;
  end
    end

    E(1) = xt - xn(n) ;
    E(2) = yt - yn(n) ;
    Eo(1) = xf - xn(n) ;
    Eo(2) = yf -yn(n) ;

   % K(1) = (vo*(1-exp(abs(E(1))^2*-2)))/abs(E(1)) ;
   % K(2) = (vo*(1-exp(abs(E(2))^2*-2)))/abs(E(2)) ;

    sigma = 1 - exp(-1*beta*d1) ;



    v = 0.1*sigma*sqrt(E(1)^2 + E(2)^2) + -1*0.1*(1-sigma)*(Eo(1)^2 +
       Eo(2)^2) ;
```

```
    v = min(20,v) ;

    w = sigma*1*(atan2(E(2),E(1)) - thetan) +
        -1*1*(1-sigma)*(atan2(Eo(2),Eo(1)) - thetan) ;


    Tnew(n+1) = DelT*n ;

    thetan = w*DelT + thetap ;
    thetap= thetan ;
    xndot = (v*cos(thetan));
    yndot = (v*sin(thetan));

    xn(n+1) = xndot*DelT + xp ;
    yn(n+1) = yndot*DelT + yp ;
    xp = xn(n+1) ;
    yp = yn(n+1) ;
```

A.8 Boundary Following Matlab Code

```
 clc ;
clear all ;
close all ;
m=0;
k1 = 1;
n=0;
xn = 0 ;
yn = 0;
Tnew = 0 ;
xf1 = 0 ;
xf2 = 0 ;
xf3 = 0 ;
yf1 = 0 ;
yf2 = 0 ;
yf3 = 0 ;
cdistance = 0 ;
Mccw = [0 -1 ; 1 0 ] ;
Mcw = [0 1 ; -1 0 ] ;
dto = 200 ;


 K= .1 ;
    m= m+1 ;
    e1 = 0;
    l = 0;
e2 = 0;
```

```
eo1 = 0;
eo2 = 0;
xf = 0;
yf = 0;
E = [e1 ; e2] ;
Eo =[eo1 ; eo2] ;
Ko = 0.1 ;

U = [0 ; 0] ;
 thetan = 0 ;
 theta1 = 0;
 xp = 0;
 yp = 0 ;
 thetap = 0;
 setheta = 0;
xt = 100 ;
yt = 100 ;
kp = 7 ;
ki = 0 ;
DelT = 0.1 ;
c = 5 ;
d1 = 0 ;
n = 0 ;
lastw = 0;
Ko = .5 ;

while(n<10000)
    n=n+1;

    d =  0 ;
    d1 = 100 ;
    xo = 50 ;
    for yo=30:0.01:70 ;
    d= sqrt((xo-xn(n))*(xo-xn(n)) + (yo-yn(n))*(yo-yn(n))) ;
  if d<d1
      d1=d ;
      xf1 = xo ;
      yf1 = yo ;
  end
    end


    d =  0 ;
    d2 = 100 ;
    yo = 30 ;
    for xo=30:0.01:50 ;
      d= sqrt((xo-xn(n))*(xo-xn(n)) + (yo-yn(n))*(yo-yn(n))) ;
      if (d<d2)
           d2=d ;
           xf2 = xo ;
```

```
          yf2 = yo ;
      end
   end

    d =  0 ;
   d3 = 100 ;
yo = 70 ;
   for xo=30:0.01:50 ;
     d= sqrt((xo-xn(n))*(xo-xn(n)) + (yo-yn(n))*(yo-yn(n))) ;
         if (d<d3)
            d3=d ;
            xf3 = xo ;
            yf3 = yo ;
         end
   end

    d =  0 ;
   d4 = 100 ;
xo = 30 ;
   for yo=60:0.01:70 ;
     d= sqrt((xo-xn(n))*(xo-xn(n)) + (yo-yn(n))*(yo-yn(n))) ;
         if (d<d4)
            d4=d ;
            xf4 = xo ;
            yf4 = yo ;
         end
   end

   if d1> d2
       D = d2 ;
       xf = xf2 ;
       yf = yf2 ;
       if d2 > d3
           D = d3 ;
           xf = xf3 ;
           yf = yf3 ;
       else
           D = d2 ;
            xf = xf2 ;
           yf = yf2 ;
       end


   else
       D = d1 ;
       xf = xf1 ;
       yf = yf1 ;
        if  d1 > d3
           D = d3 ;
            xf = xf3 ;
```

```
          yf = yf3 ;
      else
         D = d1 ;
         xf = xf1 ;
         yf = yf1 ;
     end
 end

 if d4 < D
     D = d4 ;
     xf = xf4 ;
     yf = yf4 ;
 end


 E(1)  = xt - xn(n) ;
 E(2)  = yt - yn(n) ;
 Eo(1) = xf - xn(n) ;
 Eo(2) = yf - yn(n) ;

 dt = sqrt(((xt-xn(n))*(xt-xn(n))) + ((yt-yn(n))*(yt-yn(n)))) ;


 if (dt < dto)

     if D>7

v = K*sqrt( E(1)^2 + E(2)^2) ;

w = kp*(atan2(E(2),E(1))- thetan) ;

   end
 end

   if (D<=7 && D>=4)

 v = Ko*sqrt( Eo(1)^2 + Eo(2)^2) ;
% v = .5 ;
  if atan2(E(2),E(1)) >= 0
   w =  kp*( atan2(Eo(2),Eo(1)) + pi()/2 - thetan) ;
  else
   w =  kp*( atan2(Eo(2),Eo(1)) - pi()/2 - thetan) ;
  end
  dto = sqrt( E(2)^2 + E(1)^2) ;
   end


   if D<4
 v = Ko*sqrt( Eo(1)^2 + Eo(2)^2) ;
 w = -1*kp*(atan2(Eo(2),Eo(1)) - thetan) ;
```

```
      end

   Tnew(n+1) = DelT*n ;
   thetap= thetan ;
   xp = xn(n) ;
   yp = yn(n) ;
   thetan = w*DelT + thetap ;
   xndot = (v*cos(thetan));
   yndot = (v*sin(thetan));

   xn(n+1) = xndot*DelT + xp ;
   yn(n+1) = yndot*DelT + yp ;

end
```

APPENDIX B

CIRCUIT DIAGRAMS

A.A.R Robot Schematic[17]

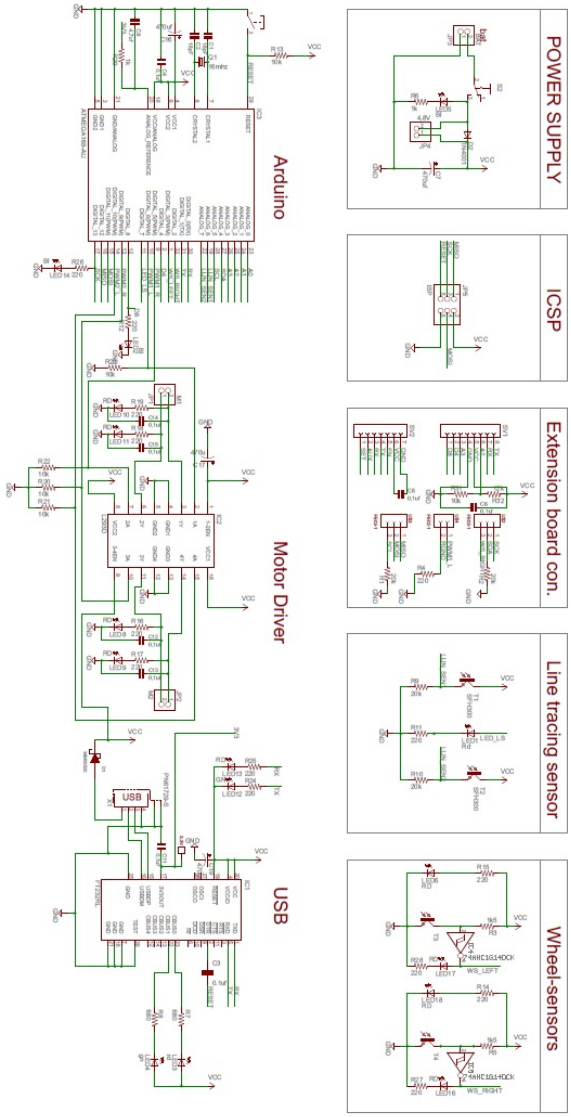

Figure B.1: AAR Robot Schematic

I.R Sensor Distance to Voltage Graph[16]