Automatic Control of Wheeled Mobile Robots

Sachin Daluja & Subhasis Behera

May 2004

Contents

List of Figures

Abstract

The project aims to design and develop a universal Wheeled Mobile Robot, to implement different kinematic models and test various control strategies. A detailed study of kinematics of Wheeled Mobile Robots is done. The objective is to develop an appropriate kinematic model of a mobile robot and test various time varying feedback control algorithms on this model to control its motion from a given starting position to desired goal position. Various control strategies are reviewed and compared for trajectory tracking and posture stabilization problems in an environment free of obstacles. Other problems such as parking problem is also reviewed. The strategies are simulated in MATLAB using ode23 solver. An appropriate mathematical model for a car-type mobile robot is derived to determine the relationship between the actual pattern of motion followed by the wheels of the vehicle. A control software is developed for interfacing with the driving hardware. The control software includes the implementation of the various algorithms pertaining to the kinematic model and strategies, software for stepper motor control and certain plotting functions for displaying the status of the mobile robot. A control circuit is made using *ULN2003* ICs, which convert the output signals generated by the control software to the required form so as to be used by the stepper motors. From the comparison of the obtained results, a guideline is provided for Wheeled Mobile Robot end-users.

Chapter 1

Introduction

Over the past twenty years the design and control of wheeled mobile robots (WMRs) has been heavily studied due to the challenging theoretical nature of the problem (i.e., WMRs are nonlinear, under-actuated systems subject to non-holonomic constraints imposed by a pure rolling and non-slipping assumption) and the wide range of applications that are well suited to their use (e.g.,munitions handling, exploration, security and monitoring, etc.). One of the major areas of application of WMRs is in control of vehicle motions for Intelligent Vehicle Highway Systems (IVHS). The California PATH (Partners for Advanced Transit and Highways) is among the leading platforms for the research in this area. Most of these works have been concentrated on tracking and posture stabilization problems. The tracking problem is to design a control law which makes a WMR follow a given trajectory. Stationary state feedback technique was used for this and stable controllers were proposed by many authors, which were implemented and tested in various WMRs successfully. The posture stabilization problem is to stabilize the vehicle to a desired final posture starting from any initial posture (posture means both position and orientation of the mobile robot from base). Such posture stabilization problem of mobile robot is more difficult than the tracking problem in the sense that a nonholonomic system with more degrees of freedom than control inputs cannot be stabilized by any static state feedback control law.

Apart from the difficulty associated with the control of multi-degree WMRs, difficulty is associated with control of these systems from a computer and the dynamics of the actuation systems. Recently, there are also many researches on the control of a car-like WMR for real application. Unlike the conventional differential type two wheeled WMRs, the car-like mobile robot has a low limit on turning radius and this constraint makes the problem more difficult. Samson [2] proposed a time-varying feedback control, which was the first theoretical approach. A few years ago, chained form systems have been introduced to model the kinematics of nonholonomic mechanical system.

The approaches for this problem can be largely divided into two categories. The first one is open loop control, i.e. the nonholonomic path planning and the second one is the closed-loop control, i.e. state feedback control. In the nonholonomic path planning, authors assume inputs (v, delta) in the kinematic model as a function of time, then assume and modify function to fit their purpose. but it is generally difficult to find or modify inputs v, delta which transfer a car-like mobile robot to a desired posture and a tracking control should be designed because it is only path-planning. In contrast to this, the most important merit of state feedback control in posture stabilization is that it can be directly used as a controller without any path-planning. In this work a motion control of a car-type mobile robot is discussed.

1.1 The Objective

The primary objective of the following work is to design a physical setup to test various feedback control strategies for the control of a WMR. Both the car type and differential type WMRs are tested in the system. The physical vehicle is a three-wheeled WMR, having two wheels for driving and one for steering. However the same vehicle can serve as a differentially driven system when a castor wheel replaces the front steering wheel. The objectives include studying and developing feedback control strategies for solving some of the typical control problems in IVHS, Kinematic modeling of WMRs according to the constraints of geometry and slip free motion, developing a physical system to implement the kinematic model. The objective also includes, interfacing the physical system with the computer, designing a control circuit for controlling the stepper motors which drive the vehicle, from signals generated by the computer.

1.2 Basic concepts

The main system can be considered to constitute of the following 3 subsystems:

1. Mechanical

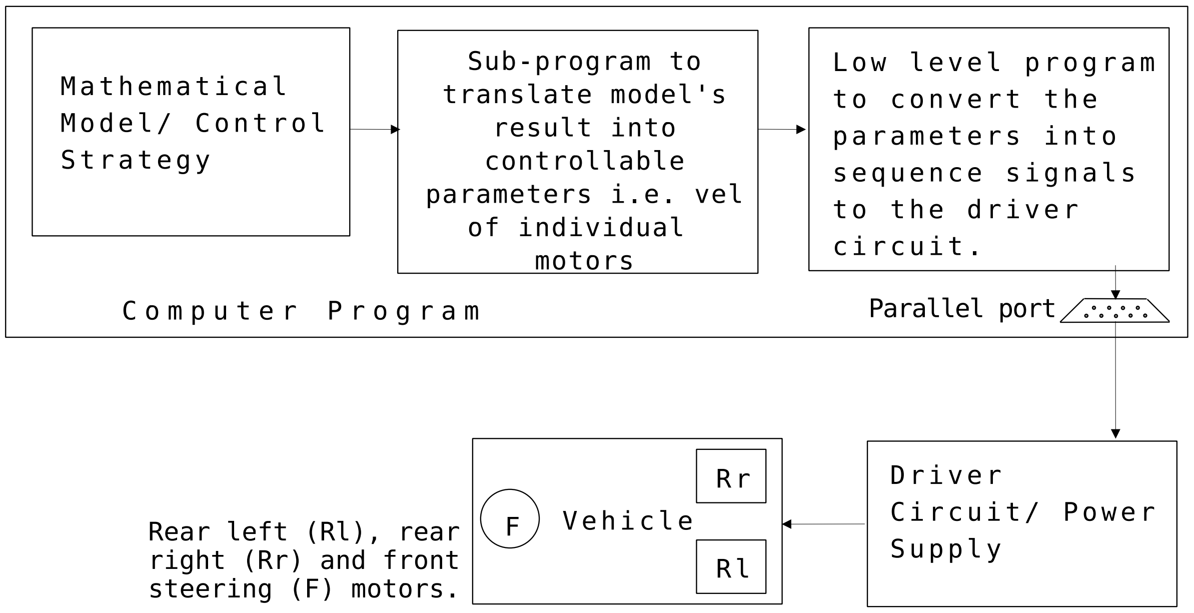

Mathematical Model/ Control Strategy
Sub-program to translate model's result into controllable parameters i.e. vel of individual motors
Low level program to convert the parameters into sequence signals to the driver circuit.
Computer Program
Parallel port
Driver Circuit/ Power Supply
Vehicle
F
Rr
Rl
Rear left (Rl), rear right (Rr) and front steering (F) motors.

Figure 1.1: Basic Control System.

2. Electronic
3. Software

The **mechanical subsystem** comprises of a 3-wheeled vehicle consisting of 2 rear wheels powered by individual stepper motors and a front steering wheel powered by another stepper motor. It also includes the reduction gearing mechanism incorporated to step up the torque generated by the steppers and to increase the positioning accuracy.

The mechanical system derives its input from an **electronic subsystem** which is basically an electronic stepper motor driving circuit. The logic for the circuitry is generated from the computer's parallel port, by the software subsystem.

The **software subsystem** comprises of the mathematical model of the WMR, the various control strategies, the stepper motor driving algorithms, and various plotting functions used for representing the vehicle motion on the computer display.

The following schematic shows the interrelationship between these subsystems.

1.3 Overview of the report

A brief idea of each chapter is presented here.

In chapter 2, a brief introduction to feedback control is given with a discussion about Open and Closed loop control systems. The different types of vehicle models (WMRs) are also discussed. The kinematics of the models are studied and the state variable models developed. Different control problems (parking problem, tracking problem, obstacle avoidance, etc.) associated with Intelligent Vehicle Highway Systems (IVHS) are studied and Control strategies are developed to solve them. The above control laws are simulated in MATLAB and the results examined.

In chapter 3, the kinematic model of the vehicle is discussed which gives the values of velocities for the individual wheels of the WMR. The algorithm for achieving this objective in the vehicle is touched upon. The various unavoidable errors are also modeled into the algorithm. This includes the kinematic constraints imposed due to the geometry and the discrete nature of the actuation device (stepper motors), as well as the nonholonomic slips occurring at various wheels.

Chapter 4 discusses the physical structure of the WMR. Chapter 5 deliberates upon the driving hardware of the system. A brief classification of stepper motors is done. The stepper motor architecture is discussed briefly. The electronic hardware to control the stepper motors are deliberated upon.

Chapter 6 describes the software subsystem which is the core of the control system. Software includes algorithms for translating kinematic variables into the system control variables, stepper motor control, and graph plotting algorithms. It also includes the computer programming implementation of the kinematic model of the WMR.

In chapter 7, the test results of the implementation of the strategies will be given. The results are compared with the ideal characteristics. The limitations of the system are described and the scope of the future work is presented.

Chapter 2

Control Strategies

Closed loop system is defined as one in which certain part of the system forcing signals (called inputs) are determined, at least in part, by certain of the responses of the system (called outputs). Hence the system inputs are a function of the system outputs, and the system outputs are a function of system inputs. The relationship is clear from Fig.2.1.

At a kinematic level, the posture (posture means both the position and orientation of the mobile robot from the base) of WMR are naturally and conveniently described (1) by the Cartesian coordinates x, y and the angle θ. In fact, control inputs are $\{\omega_i$: i=[1,n]; θ_j: j=[1,m]$\}$, the rates of rotations of the driving wheels and the steering angles of the steering wheels, but they are not suitable for developing a universal model of WMR kinematics because of at least two reasons: (i) the model will be dependent upon particular mechanism design; (ii) they are not independent themselves, and the substantial kinematic redundancy, that exists in each case of n+m>3, is completely eliminated through the rolling compatibility conditions[1]. As a consequence, for convenience v and ω, the linear and angular velocities of WMR, are commonly considered as control inputs.

$x = [\begin{matrix} x & y & \theta \end{matrix}]^T$ and $u = [\begin{matrix} v & \omega \end{matrix}]^T$ can be denoted as the state and control vectors respectively. The kinematics model of WMR can be presented as:

$$\dot{x} = B(x) \cdot u \tag{2.1}$$

where,

$$B(x) = \begin{bmatrix} \sin\theta & 0 \\ \cos\theta & 0 \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

The above model is general and applies to any WMR, providing a unified means of motion planning and control. For a particular vehicle, the necessary equations, describing its kinematics and dynamics, should be added to relate v and ω to {ω_i: i=[1,n]; θ_j: j=[1,m]}.

It is often convenient, for the purpose of analysis, synthesis and design, to consider (at least in the beginning) a WMR with a simple kinematic structure and low number of driven/steered wheels. This calls for the concept of kinematically equivalent WMR (EWMR), that is based on the functional capabilities rather than on the mechanical design of the vehicle. Two WMRs are kinematically equivalent, if they ave the same linear and angular velocities for every time instant when performing the same desired motion. The respective relations can be easily derived from geometric reasoning, because both WMR and EWMR should have the same angular velocities, the axles of their wheels should intersect in one and the same point, the instantaneous center of rotation (ICR); adding the conditions of pure rotation without slipping[1] uniquely determines the rates of rotation of the driving wheels and the steering angles of the steering wheels[1].

Some basic structures of EWMRs possessing the minimum number of wheels: an EWMR is said to be in minimal form, if it comprises the minimum necessary and sufficient (for a prescribed task) number of joints. Three such minimal form EWMRs, all having three wheels configured are described here as follows:

1. one, both driving and steering , front wheel and two passive rear wheels[3], [4];
2. one (possibly two - "car-like WMR") steering front wheel and two driving rear wheels;
3. two driving front and one (possibly two) passive rear wheels (castor).

Comparing the minimal form EWMR listed, the following should be pointed out:

- The first two structures have a common feature- steering and driving are separated, while in the third case both are performed through the driving wheels;

- It is difficult to give preference to one of them, because all the three are simple enough and mechanically sound, the kinematic relations are similar.

In the following work the second and third kinematic models will be discussed and control strategies will be developed to control them. These control strategies will then be implemented in physical systems. The second model is called car-type WMR and the third model is called differentially driven WMR. These two types of models are discussed in the sections that follow.

2.1 Differentially driven mobile robot

In this section, a detailed study of automatic control problem of a differentially driven mobile robot is done and state feedback controllers are developed to achieve the desired objective. The differentially driven mobile robot is the conventional WMR used in most of the applications. It has no limit on turning radius and this makes the WMR very versatile. This WMR can turn through any given angle and even can turn through 360 degrees without needing to change its position. The control parameters are also lesser. The control problem can be defined as follows.

The kinematic model for differentially driven WMR is described here for the nonholonomic constraint of pure rolling and non-slipping. Based on the kinematic model, the differentiable, time-varying kinematic controllers for the regulation control problem will be analyzed here. Given an initial posture assumed by the vehicle and a final desired posture, a state feedback controller has to be designed, so as to stabilize the vehicle to the desired posture starting from the initial posture in an environment free from obstacles. Later various standard problems in IVHS such as parking problem, obstacle avoidance will be studied and controllers will be developed to solve the problems from posture stabilization approach.

The model have two rear wheels driven independently, which drive as well as steer the vehicle and a front wheel on a castor. The origin of the local coordinate system attached to the vehicle frame of reference is located at the rear of the vehicle midway between the two rear wheels. The longitudinal axis of the reference frame attached to the vehicle is the line passing through the center-line of the vehicle and the lateral axis perpendicular to the longitudinal axis, passing through the two rear wheels. the longitudinal axis is oriented from the rear wheels towards the front wheel and the lateral axis is oriented from the rear right wheel towards the rear left wheel.

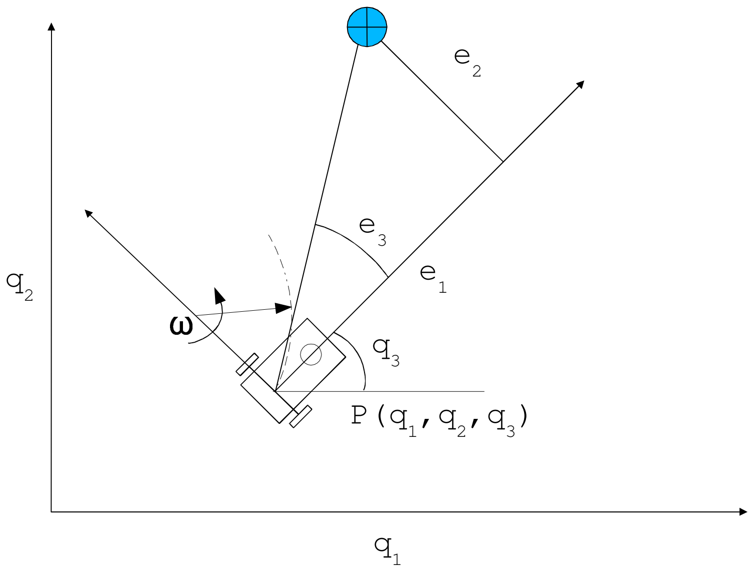


Figure 2.1: Differentially driven WMR.

The instantaneous position of the origin of the reference frame attached to the vehicle is given by (q_1, q_2, q_3). The position of the destination point (x_d, y_d, θ_d) to be traced in the reference frame attached to the vehicle is given by (e_1, e_2, e_3). Where,

e_1	= The instantaneous longitudinal coordinate of the desired point to be traced with respect to the reference system of the vehicle.
e_2	= The instantaneous lateral coordinate of the desired point to be traced with respect to the reference system attached to the vehicle.
e_3	= The instantaneous angular coordinate of the desired point to be traced with respect to the reference system of the vehicle.

The conversion of the global co-ordinates to the local co-ordinates yields the following:

$$e_1 = (x_d - q_1) \times \cos q_3 + (y_d - q_2) \times \sin q_3 \tag{2.3}$$

$$e_2 = -(x_d - q_1) \times \sin q_3 + (y_d - q_2) \times \cos q_3 \tag{2.4}$$

$$e_3 = \tan^{-1}\left(\frac{y_d - q_2}{x_d - q_1}\right) - q_3 \tag{2.5}$$

The kinematic model for the nonholonomic constraint of pure rolling and non-slipping is given as follows.

$$\begin{aligned} \dot{q}_1 &= v\cos\theta \\ \dot{q}_2 &= v\sin\theta \\ \dot{q}_3 &= \omega \end{aligned} \tag{2.6}$$

The above equation satisfies nonholonomic constraints of pure rolling and non-slipping for a differentially driven WMR. q_1 and q_2 denote the location of the center of axle between two rear wheels, i.e., the origin of the reference frame attached to the vehicle. e_1 and e_2 are the longitudinal and lateral axis respectively in the reference frame attached to the vehicle. q_3 is the orientation of the mobile robot with respect to the longitudinal axis (q_1) in the global frame of reference. v denotes the forward longitudinal velocity and ω denotes the angular velocity of the origin of the vehicle reference frame. The differentially driven mobile robot is controlled by the relative differential velocity of the rear wheels, i.e., there are two control inputs (v,ω), but it has three degrees of freedom (q_1, q_2, q_3) in the plane.

Thus, the problem is to control the differentially driven mobile robot with two inputs, steering angle and forward velocity of the rear wheels, to move to a desired posture in an obstacle-free environment.

2.1.1 Control in an obstacle free environment

The control objective of this model is to force the actual Cartesian position and orientation to a constant reference position and Orientation. The model is modeled with respect to the global reference frame. Given a global reference plane in which the instantaneous position and orientation of the model is given by (q_1, q_2, q_3). The vehicle has to start at a position (x, y, θ) and has to reach a given point (x_d, y_d, θ_d) with respect to the global reference plane.

The Control objective is to design a controller for the kinematic model given by equation (2.6) that forces the actual Cartesian position and orientation to a constant reference position and orientation. Based on this control objective a simple time varying controller was proposed as follows.

$$\begin{aligned} v &= k_1 \times e_1 && (2.7) \\ \omega &= k_2 \times e_3 + e_2^2 \times \sin t && (2.8) \end{aligned}$$

This is a general controller which satisfies Lyapunov's criterion for stability. Here, the longitudinal velocity is directly proportional to the longitudinal error in the reference system attached to the vehicle. The rate at which the vehicle should be turned is proportional to the angular orientation of the desired position with respect to the reference frame attached to the vehicle, that is e_3. The term $e_2^2 \times \sin t$ is added to the angular speed term due to the following reasons.

1. To ensure that the vehicle does not gets locked in a position when the longitudinal error e_1 is zero but the lateral error e_2 is still not zero. This kind of a situation can arise when the destination point lies in the lateral axis of the vehicle co-ordinate system. In such a case the vehicle would have stopped had the proportional controller been applied. Even though the destination point was not reached.

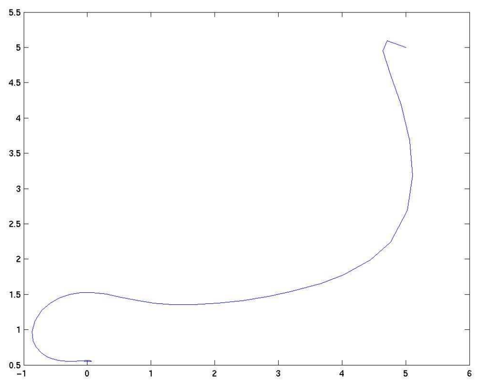
2. To stabilize the vehicle to a desired final orientation. Which is necessary in case the vehicle has to be parked in a desired orientation.

After substituting equation(2.7) and equation(2.8) into equation(2.6), the following closed loop error system was developed:

$$\begin{aligned} \dot{q_1} &= k_1 \times [(x_d - q_1) \times \cos q_3 + (y_d - q_2) \times \sin q_3] \times \cos q_3 && (2.9) \\ \dot{q_2} &= k_1 \times [(x_d - q_1) \times \cos q_3 + (y_d - q_2) \times \sin q_3] \times \sin q_3 && (2.10) \\ \dot{q_3} &= -k_2 \times \left[\tan^{-1}\left(\frac{y_d - q_2}{x_d - q_1}\right) - \theta\right] + k_3 \times [-(x_d - q_1) \times \sin q_3 + (y_d - q_2) \times \cos q_3]^2 \times \sin t && (2.11) \end{aligned}$$

The above model is modeled in MATLAB using state feedback approach and a strategy is tested for the vehicle to reach different destination points. The MATLAB model is in Appendix B1. Some plots are shown below in which, the vehicle starts from (0, 0, 0) and reaches a point in various quadrants. The parameters: k1 = 1 and k2 = 1. The model is simulated for 10 seconds. The plots show the trajectory of the vehicle in a plane.

The above plots show pretty smooth trajectories and seems to be flawless for all the positions. However it has certain disadvantages associated with it


(a) The trajectory of the vehicle when, k1 = 1 and k2 =1 .

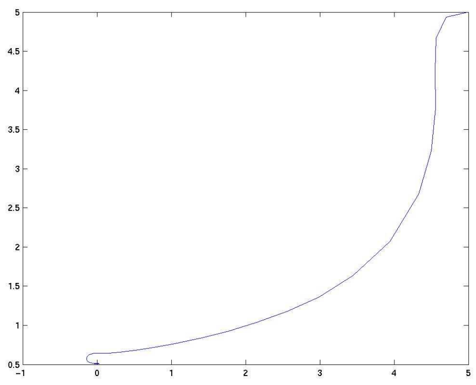

(b) The trajectory of the vehicle with the optimal values of the parameters, k1 = 2 and k2 =0.1 .

Figure 2.2: Simulation results for differentially driven WMR in MATLAB.

when the vehicle approaches the destination point. The speed of the vehicle drops reasonably and the vehicle takes a lot of time to stabilize to the desired orientation. This can be solved by adopting a higher speed when the vehicle has reached reasonably close to the destination point. But that results in a fast zigging motion towards the end of travel, which is also undesirable. A reasonable compromise between the time and smoothness of motion has to be done in this case depending on the application.

2.2 Car-like mobile robot

In this section, a detailed study of automatic control problem of a car-like mobile robot is done and state feedback controllers are developed to achieve the desired objective. Unlike the conventional differentially driven mobile robot, the car-like mobile robot has a low limit on turning radius and this constraint makes the problem more difficult. The control problem can be defined as follows:

Given an initial posture assumed by the vehicle and a final desired posture, a state feedback controller has to be designed, so as to stabilize the vehicle to the desired posture starting from the initial posture in an environment free from obstacles. Later various standard problems in IVHS such as parking problem,

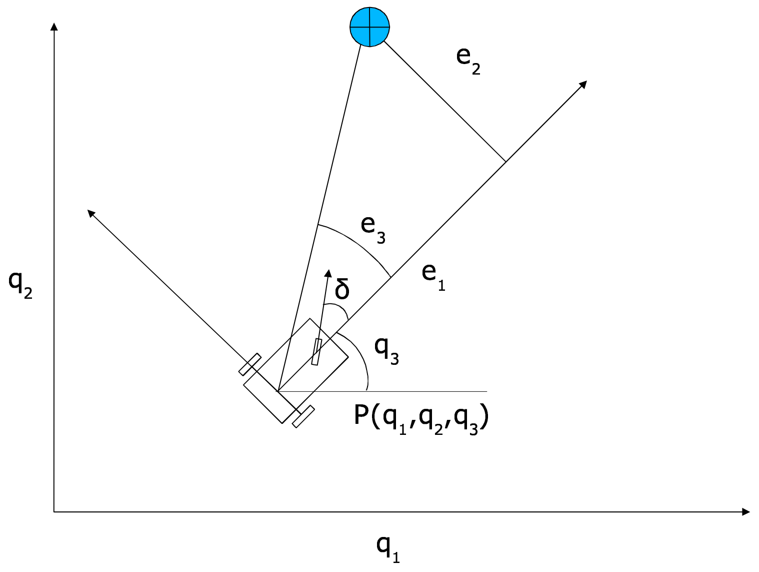


Figure 2.3: Global and local co-ordinate systems of the WMR.

obstacle avoidance will be studied and controllers will be developed to solve the problems from posture stabilization approach.

The vehicle can be described as a three wheeled system having two rear and one front wheel. The front wheel does the steering and the two rear wheels drive the vehicle following the non slip condition. The origin of the local coordinate system attached to the vehicle frame of reference is considered to be located midway between the two rear wheels. The longitudinal axis of the reference frame attached to the vehicle is the line passing through the center-line of the vehicle and the lateral axis perpendicular to the longitudinal axis, passing through the two rear wheels. The longitudinal axis is oriented from the rear wheels towards the front wheel and the lateral axis is oriented from the rear right wheel towards the rear left wheel.

The instantaneous position of the origin of the reference frame attached to the vehicle is given by (q_1, q_2, q_3). The position of the point to be traced (x_d, y_d, θ) in the reference frame attached to the vehicle is given by (e_1, e_2, e_3). Where,

e_1	= The instantaneous longitudinal coordinate of the desired point to be traced with respect to the reference system of the vehicle.
e_2	= The instantaneous lateral coordinate of the desired point to be traced with respect to the reference system attached to the vehicle.

e_3 = The instantaneous angular coordinate of the desired point to be traced with respect to the reference system of the vehicle.

The conversion of the global co-ordinates to the local co-ordinates yields the following:

$$e_1 = (x_d - q_1) \times \cos q_3 + (y_d - q_2) \times \sin q_3 \tag{2.12}$$
$$e_2 = -(x_d - q_1) \times \sin q_3 + (y_d - q_2) \times \cos q_3 \tag{2.13}$$
$$e_3 = \tan^{-1}\left(\frac{y_d - q_2}{x_d - q_1}\right) - q_3 \tag{2.14}$$

The kinematic model of a car-like mobile robot in fig2.2 is given by,

$$\begin{aligned} \dot{q}_1 &= v\cos\theta \\ \dot{q}_2 &= v\sin\theta \\ \dot{q}_3 &= \frac{v}{l}\tan\delta \end{aligned} \tag{2.15}$$

The above equation satisfies nonholonomic constraints of pure rolling and non-slipping. q_1 and q_2 denote the location of the center of axle between two rear wheels. q_3 is the orientation of the car-like mobile robot with respect to the longitudinal axis (q_1), δ is the steering angle with respect to the body and v denotes the forward velocity of the rear wheels and l is the wheel base. The car-like mobile robot is usually controlled by the steering angle and the velocity of the rear wheels, i.e., there are two control inputs (v, δ), but it has three degrees of freedom (q_1, q_2, q_3) in the plane.

Thus, the problem is to control the car-like mobile robot with two inputs, steering angle and forward velocity of the rear wheels, to move to a desired posture with and without the consideration of obstacles. First the problem is considered in an obstacle-free environment.

2.2.1 Control in an obstacle-free environment

The control objective of this model is to force the actual Cartesian position and orientation to a constant reference position and Orientation. The model is modeled with respect to the global reference frame. Given a global reference plane in which the instantaneous position and orientation of the model is given by (q_1, q_2, q_3). The vehicle has to start at a position (x, y, θ) and has to reach a given point (x_d, y_d, θ) with respect to the global reference plane.

The Control objective is to design a controller for the kinematic model given by equation (2.1) that forces the actual Cartesian position and orientation to a constant reference position and orientation. Based on this control objective a simple time varying controller was proposed as follows.

$$v = v_0 + v_{par} \times e_1 \tag{2.16}$$

$$\delta = c_{par} \times e_3 \tag{2.17}$$

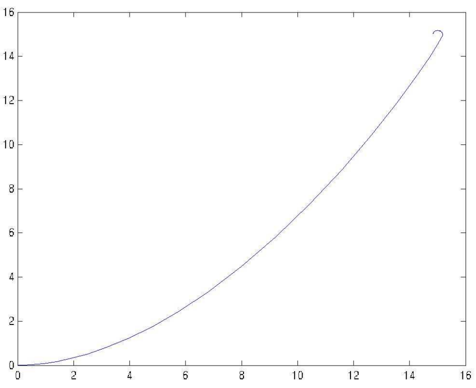
This is a simple proportional controller with a small modification in the velocity. Here, the longitudinal velocity is directly proportional to the longitudinal error in the reference system attached to the vehicle. The constant value, v_0 is added to the velocity term to ensure that the vehicle does not gets locked in a position when the longitudinal error e_1 is zero but the lateral error e_2 is still not zero. This kind of a situation can arise when the destination point lies in the lateral axis of the vehicle co-ordinate system. The vehicle would have stopped in such a case had the proportional controller been applied, even though the destination point was not reached. The rate at which the front steering wheels should be turned is proportional to the angular orientation of the desired position with respect to the reference frame attached to the vehicle, that is e_3. Here v_{par} and c_{par} are positive constant control gains. After substituting equation(2.16) and equation(2.17) in equation(2.15), the following closed loop error system was developed:

$$\dot{q}_1 = v_{par} \times e_1 \times \cos q_3 \tag{2.18}$$

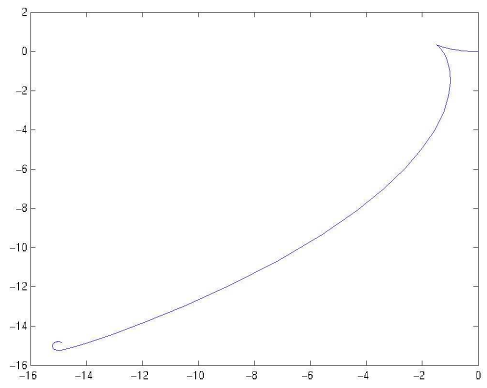
$$\dot{q}_2 = v_{par} \times e_1 \times \sin q_3 \tag{2.19}$$

$$\dot{q}_3 = \left(\frac{v_{par} \times e_1}{l}\right) \times \tan(c_{par} \times e_3) \tag{2.20}$$

The above model is modeled in MATLAB using state feedback approach and a strategy is tested for the vehicle to reach different destination points. The objective of testing in MATLAB is to check the suitability of the strategy. The MATLAB model is in Appendix B2. Some plots are shown below in which, the vehicle starts from $(0, 0, 0,)$ and reaches a point in various quadrants. The parameters: c_{par}



(a) Trajectory while tracing (5, 5) .



(b) Trajectory while tracing (-5, -5) .

Figure 2.4: Simulation results for car-like WMR in MATLAB.

= 1 and v_{par} = 1. The model is simulated for 10 seconds. The plots show the trajectory of the vehicle in a plane.

The above plots show pretty smooth trajectories and seems to be flawless for all the positions. However it has certain disadvantages associated with it when the destination point distance is comparable with the vehicle dimensions. Practically in such a situation, it is even difficult for the humans to drive properly and the driving strategy is different from the approach when the destination point is located at a distance. In such a situation, a different control strategy should be employed to tackle the problem and it constitutes a different control problem. However the vehicle can be driven to a distance reasonably close to the destination point by the above strategy and can be chosen to stop at a certain distance, when it starts to behave improperly.

The strategy exhibited a very interesting property. In case the destination point lied behind the starting point, instead of turning all the way round, it traveled backwards till it faced the goal position and then moved straight to that. This is quiet similar to the way humans drive the vehicles. However the strategy was unable to orient the vehicle in a final desired orientation. So it is not possible to park this vehicle at a chosen orientation. This constitutes another problem and is the subject of discussion in the following section.

2.2.2 Control strategy with consideration of obstacles and Parking problem

The control objective here is to control the car-like mobile robot with two inputs, steering angle and forward velocity of the rear wheels, to move to a desired posture with the consideration of obstacles.

It is well known from the experience of driving that the parking maneuver consists of two steps, which are stabilization of a car-like mobile robot to a desired line ($y = 0, \theta = 0$) and a desired point by go-forwards or go-backward ($x = 0$), that is;

- First step: $y, \theta \to 0$
- second step: $x \to 0$

2.2.2.1 Control algorithm for first step

An algorithm is proposed which stabilizes y andθ for the first step. In this algorithm, nonlinear state feedback law is used and v is assumed to be a given non-zero constant. The target posture in this case can be assumed to be the origin in the global frame ($q_1 = 0, q_2 = 0, q_3 = 0$). The following is the proposed control law.

$$v = v_0, -v_0 \tag{2.21}$$

$$\delta = \tan^{-1}\left\{-\frac{l}{v}\left(k_2 q_3 + k_1 v \frac{\sin q_3}{q_3} q_2\right)\right\} \tag{2.22}$$

Where, $v_0$is constant and k_1 ,k_2 are positive scalar values. Then q_2 and q_3 converges asymptotically to zero.

v_0 is a pre-defined constant. However, the sign of v can be freely varied. The situation where the mobile robot should change the sign of v occurs in the following cases:

1. When the mobile robot can not maintain the velocity due to obstacles.
2. When some measured states exceed limit values.

This freedom of changing v could be used in avoiding obstacles or restricting some measures such as the distance from the current position to the origin. Moreover, a proper choice of v at the start position can make a more efficient path.

At the start the sign of v is selected by the following rule.

If $\cos(\alpha - q_3) > 0$, then $v < 0$ else $v > 0$.
where, $\alpha = \tan^{-1}\left\{\frac{q_2}{q_1}\right\}$

The physical meaning of this rule is that if the front part of the WMR is closer to the origin than the rear part, the WMR will go forwards, or it will go backwards.

2.2.2.2 Control algorithm for second step

If both q_2 and q_3 becomes small enough as a result of controller in equation(2), we use the following algorithm;

$$v = -k_3 \times q_1 \tag{2.23}$$

$$\delta = 0 \tag{2.24}$$

where k_3 is positive and real. Since we assumed q_2 and q_3 are arbitrarily small through the first algorithm, q_1 exponentially converges to zero.

2.2.3 Singular case

This algorithm is designed for short distance such as parking. If the algorithm is used for long distances where obstacle exists, singular case can occur where the mobile robot repeats going forward and backward for obstacles. But, this case can be overcome by using an intermediate posture. The above algorithm can be applied to a general parking situation where there are obstacles around the WMR.

2.3 Control strategy based on strategic positions

This is a car-type WMR, where the problem of reaching a given point can be tackled from another approach. That can be summarized as follows. The space in the vehicle frame of reference is divided into a number of different geometric regions. The behavior of the vehicle can be modeled in a particular way according to the presence of the point in a particular region of space till it reaches in a very close vicinity of the desired point when it finally stops. Thus the behavior of the vehicle in this model is pretty predictable and hence various control strategies can be applied to the model easily.

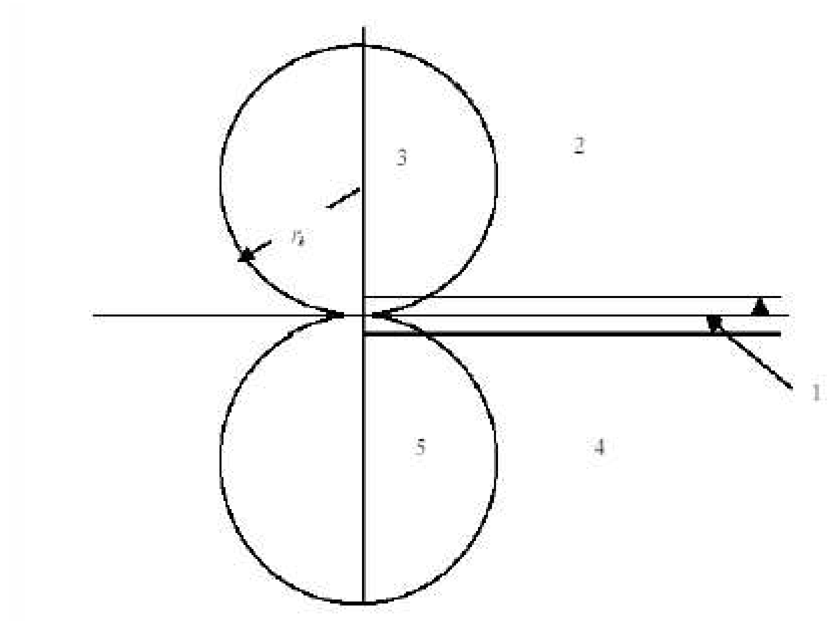


Figure 2.5: The kinematic representation of three-position steering model.

The strategy can be described as follows.

The space is divided into five discrete regions with respect to the vehicle frame of reference. These regions can be defined as follows.

1. This portion can be defined in the vehicle Cartesian coordinate reference plane as,
   $x_l \geq 0$, and $-\epsilon \leq y_l \leq \epsilon$
   This is the thin rectangular strip of width 2ϵ along the longitudinal axis in the vehicle frame of reference.

2. This portion can be defined in the vehicle Cartesian coordinate reference plane as,
   $y_1 > \epsilon$, for $x_l \geq 0$
   $y_1 \geq 0$, for $x_l \leq 0$
   and, $x_l^2 + (y_l - r_b^2) > r_b^2$
   This is the entire positive y plane, except for portion 1 and 3.

3. This portion can be defined in the vehicle Cartesian coordinate reference plane as,
   $x_l^2 + (y_l - r_b^2) \leq r_b^2$
   This is the portion inside a circle of radius $r_b$in the positive y plane, as shown in the figure.

4. This portion can be defined in the vehicle Cartesian coordinate reference plane as,
   $y_1 < \epsilon$, for $x_l \geq 0$
   $y_1 \leq 0$, for $x_l \leq 0$
   and, $x_l^2 + (y_l + r_b^2) > r_b^2$
   This is the entire negative y plane, except for portion 1 and 3.

5. This portion can be defined in the vehicle Cartesian coordinate reference plane as,
   $x_l^2 + (y_l + r_b^2) \leq r_b^2$
   This is the portion inside a circle of radius in the positive y plane, as shown in the figure.

The control objective is to design a controller for the kinematic model given by equation (1) that forces the actual Cartesian position and orientation to a constant reference position and orientation. Based on this control objective a simple time varying controller was proposed as follows. For controlling the velocity the following strategy is adopted.

```
if($e_1^2 + e_2^2 \geq c^2$)
  $v = V_0$
else
  $v = 0$
```

This means that the velocity is a constant and has a value $v = V_0$, for all the points in the space except for the points inside a circle of radius c. this circle is the region in which we can choose the vehicle to finally stop. This can be chosen as small as required for the vehicle to stop at a very close vicinity of the desired point. For the angular velocity control, the following conditional control strategy is adopted. The angular velocity fed to the system is region specific. A step angular displacement value is fed to the system as follows.

1. For region 1 the angular deviation is,
2. For regions 2 and 5 the value of the angular deviation is,
3. For regions 3 and 4 the value of the angular deviation is,

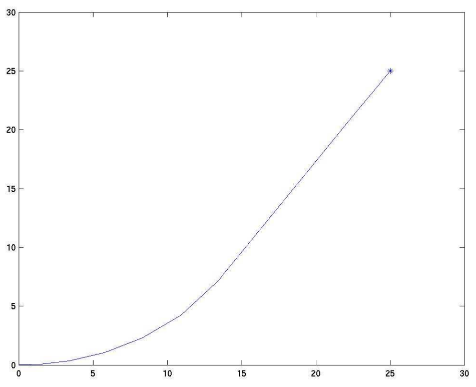
This simply means that, the step angular displacement is constant and its sign +ve, -ve or 0 is chosen according to the instantaneous location of the desired point.

In the previous discussion of car-type WMR, the variation in angular change δ was discrete. Small change in the value of δ is acceptable, but when the angular change is large, delta cannot be varied through discrete values. Here the strategy requires δ to vary through large discrete values. So the state variable model needed a modification so as to make the variation in δ continuous, when a discrete value is chosen by the strategy. A fourth state variable is included in the model. This variable is the actual angular change that takes place when a step change in the desired angle change takes place. That means there is a time lag between the required value of angular change and the actual change. This time lag factor is taken care by a parameter k. The modified model is given below.

$$\dot{q}_1 = v \cos q_3, \tag{2.25}$$

$$\dot{q}_2 = v \sin q_3, \tag{2.26}$$

$$\dot{q}_3 = \frac{v}{l} \tan q_4, \tag{2.27}$$


(a) Trajectory while tracing (25, 25) .

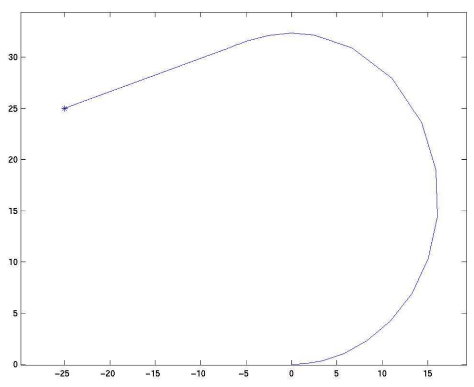

(b) Trajectory while tracing (-25, 25) .

Figure 2.6: Simulation results for 3-position steering strategy in MATLAB.

$$\dot{q}_4 = \frac{(\delta_d - q_4)}{k} \tag{2.28}$$

Here,

v = the longitudinal velocity applied to the vehicle.

δ_d= the desired value of instantaneous angular deflection provided to the wheels of the vehicle.

δ_d is a function of the region in which it lies. It is defined in equation 2.

The above model and strategy was tested for various values of state variables and parameters. In the plots are shown below in which, the vehicle starts from $(0, 0, 0)$ and reaches various destination points. The parameters are: $v_0 = 10$, err $= 0.01$ and, $c = 0.1$.

The plots show that the vehicle reaches the destination points quiet smoothly. No problem was identified in tracking the destination point with this strategy. However the strategy was not very efficient in the sense that the vehicle had to go all the way round to face a desired point before it was able to reach there.

Chapter 3

Kinematic Model

The purpose of mathematical model for WMRs is to determine the relationship between the pattern of motion followed by the different parts of the vehicle. This motion is determined by the kinematic relationship between the degrees of freedom of he motion of the wheels. For a WMR, when the wheels don't skid, the motion is determined by the constraints of geometry of the system. This kind of dynamic system is called a nonholonomic system. The mathematical model of a WMR gives the values of the actual vehicle speeds at various wheels (the two rear wheels and the front steering wheel), when the vehicle is following a certain pattern of motion. these values can then be implemented in the WMR to control its motion.

3.1 Geometric model

Fig. 3.1 is the geometric representation of the car-type WMR with minimum number of degrees of freedom.
Here,

ρ	= radius of curvature of the path of the center of gravity of the vehicle.
l	= length of the vehicle.
2b	= width of the vehicle.
δ	= the orientation of the front wheel (steering wheel) of the vehicle w.r.t. the longitudinal direction of the vehicle.
v	= the longitudinal speed of the vehicle

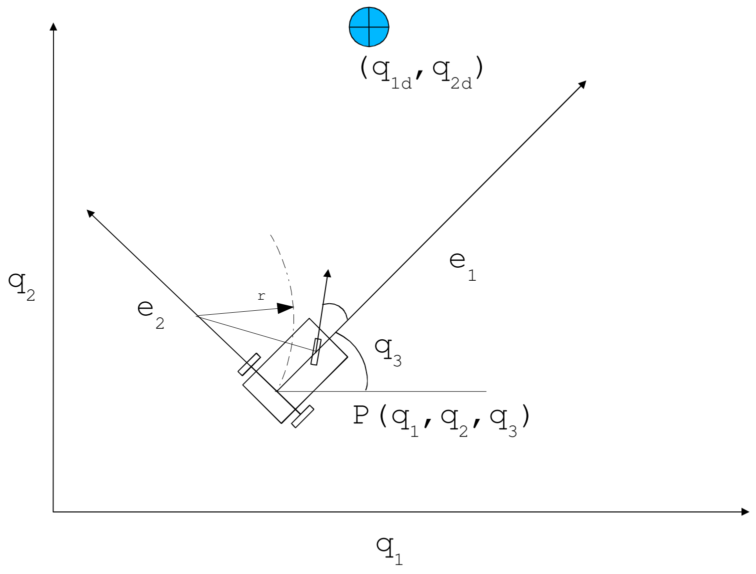


Figure 3.1: Co-ordinate system of the WMR.

ω	= angular velocity of the vehicle center, w.r.t. the instantaneous center of rotation.

The center of the vehicle is assumed to be lying in the mid point of the two rear wheels and it is the origin of the coordinate system attached to the vehicle. So from the geometry;

$$tan\delta = \frac{l}{\rho} \tag{3.1}$$

$$\omega = \frac{v}{\rho} = \frac{v}{l}\tan\delta \tag{3.2}$$

The above equation represents the constraint imposed on the motion due to the geometry of the car-type vehicle. the vehicle has to follow this constraint of motion always to undergo slip free motion.

3.2 The kinematic controllers:

The above equations represent the kinematic constraints of the vehicle. The vehicle has to obey the above condition always for slip-free motion. From the above equations it is clear that the control parameters of the vehicle motion are the longitudinal speed, v and the angular orientation of the front wheel, δ. For automatic control of the vehicle, v and δ need to be generated by the algorithm automatically in certain intervals of time. The efficiency of the WMR depends upon the wise selection of the control law to generate the values for the control parameters. There are various standard and specific controllers for all types of control problems which includes the problems of WMRs also as discussed in the previous chapter. Here we will only discuss the controllers simulated in the previous chapter.

3.2.1 Proportional controller for car-type WMR:

The control law for the proportional controller discussed in 2.3 gives the following control law.

$$\begin{aligned} v &= v_{par} \times e_1 + v_0 \qquad (3.3) \\ \delta &= c_{par} \times e_3 \qquad (3.4) \end{aligned}$$

This control law is simple and almost similar to a proportional controller. The simulation of this controller is done in the previous section and the results are quite interesting and desirable.

3.3 Calculating actual values of vehicle parameters

The equations in the previous section give the velocity at the center of axle joining the rear wheels and the angle by which the steering angle of the front wheel is to be turned. These values are however required to be converted into the actual values of velocities and angle for the two rear wheels of the physical system. This is discussed in the following:

The vehicle motion can be divided into two different modes; viz, straight mode and steering mode. Any journey of the vehicle is actually composed of a number of straight and steering modes of travel. Considering a vehicle of length ‘l’ and width ‘2b’, the following relations can be established among the control parameters.

1. In the straight mode the vehicles travels in a straight line without any steering. In this case the steering wheel is held straight and the two rear wheels

(v_{Rl} and v_{Rr}) rotate with the same speed, i.e., the speed given by control algorithm, v.

$$\delta = 0 \tag{3.5}$$

$$v_{Rl} = v_{Rr} = v \tag{3.6}$$

2. In steering mode the vehicle steers either toward left or right direction. The steering wheel is oriented in the required direction and the two rear wheels have to be provided motion in the desired manner following the constraints of motion. The control algorithm gives the value of v, i.e., the speed of center of the axis joining the rear wheels. from this v, the velocities of the rear left and the rear right wheels (v_{Rl} and v_{Rr}), can be found out through the following relationship. These relationships are based on the geometry of the vehicle and the constraints of non slipping as shown in Fig. 3.1.

$$\delta = \delta \tag{3.7}$$

$$v_{Rl} = v\left(1 - \frac{b}{l} \times \tan\delta\right) \tag{3.8}$$

$$v_{Rr} = v\left(1 + \frac{b}{l} \times \tan\delta\right) \tag{3.9}$$

The values generated by the above equations are the exact decimal values and usually fractional numbers, and many times generate recurring values. however these values may not be achieved always, due to the following limitation of the hardware. The actuation device used in the WMR i.e., stepper motors can take discrete steps only. Hence it cannot attain all the discrete values of angles generated by the above equations. In such a case the required value would lie between two discrete values achievable by the motor, separated by the step angle of the motor. So one of the ways to solve this problem, could be to choose one of the nearest values (usually the nearer) and use it in the vehicle. This can be done by rounding off the actual value to the nearest achievable value with respect to the step size. The rounding off operation can be done by the rounding off algorithm, so that the number of steps to be turned by the motors becomes whole numbers. The rounding off operation can be done in the following steps as follows:

δ can be changed to δ_{a} (achievable value of δ), using the following relation.

$$\delta_a = \left[round \left(\frac{\delta}{step - angle} \right) \times step - angle \right] \div t \tag{3.10}$$

Here, step-angle = the step size of the stepper motor in terms of angle.

The rear wheel velocities can be converted into the achievable values by the following relationships.

$$v_{Rl_a} = \left[round \left(\frac{v_{Rl} \times t}{step - distance} \right) \times step - distance \right] \div t \tag{3.11}$$

Here, step-distance is the step size in terms of the liner distance traveled by the wheels of the WMR.

Since the speed of rear left wheel has changed, the speed of the rear right wheel will also change by a corresponding amount. The modified speed of the rear right wheel is given by the following expression:

$$v_{Rr} = v_{Rr} \left(\frac{1 + \frac{b}{l} \times tan\delta_a}{1 - \frac{b}{l} \times tan\delta_a} \right) \tag{3.12}$$

The above expression for speed of the rear right wheel has to be rounded off to obtain the achievable speed v_{Rr_a} as follows:

$$v_{Rr_a} = \left[round \left(\frac{v_{Rr} \times t}{step - distance} \right) \times step - distance \right] \div t \tag{3.13}$$

Thus from the above equations, the values of the control parameters $(\delta_a, v_{Rl_a}, v_{Rr_a})$ for the three wheels of the vehicle are obtained. These values are used in the stepper algorithm to generate the signals to control the motors in the desired fashion.

One interesting thing to note here is that the motion is not 100% slip-free. When the vehicle is taking a turn, the v_{Rl_a} value can be arbitrarily chosen. But the velocity of the rear right wheel v_{Rr_a}, should correspond to equation 3., which is the modified value of velocity corresponding to the value of v_{Rl_a}. But this value may not be achievable by the wheel. Thus the velocity assumed by the right wheel is $v_{Rr_a}$and there is a small amount of slip in the motion. This slip can be reduced if the step-distance size of the wheels is very small. Since the slip occurring here is a geometric error, it can be controlled by modeling it into the kinematics. However any method of calculating the slip (to any amount of accuracy), can only help to calculate the slip and accurately keep an account of it. It does not mean that the slip has been removed.

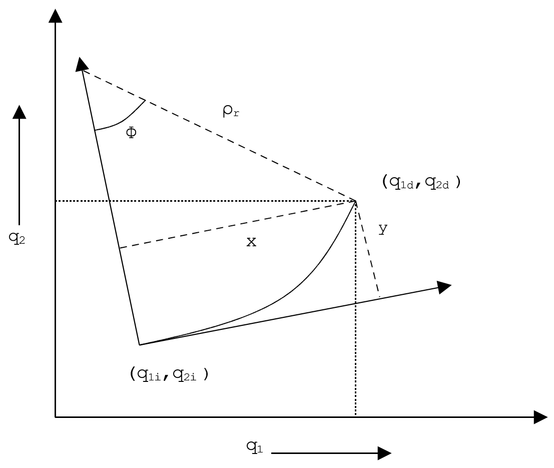


Figure 3.2: Calculating the next position.

3.4 Determining the next position

The system needs a state feedback response to implement closed loop control strategies. But, the physical system does not have any state feedback response to determine its current global or local position. So these values have to be determined from the geometry of motion. Since stepper motors are exact actuation devices, the relative displacements of the wheels can be calculated quite accurately from the kinematic relationships of the motion. This is of course under the assumption that there is no slipping in the wheels and the stepper motor is strong enough not to miss any step due to insufficient torque. Sufficiently strong stepper motors can be used to ensure that the motor provides enough torque to overcome missing of steps. Thus the stepper motor can also generate very accurate feedback, if modeled correctly. The above WMR is modeled in the following manner to give feedback.

The velocity of the center of vehicle or the origin of the local co-ordinate system attached to the vehicle is computed from the corrected velocity (v_{Rl_a}, v_{Rr_a}) assumed by the wheels.

$$v_a = \frac{v_{Rl_a}}{\left(1 - \frac{b}{l} tan\delta\right)} \tag{3.14}$$

$$\delta_a = \delta_a \tag{3.15}$$

Thus the values of actual longitudinal velocity v_a, and actual steering angle $\delta_a$assumed by the vehicle in the previous interval can be computed from the above relations. The angular velocity of the local co-ordinate system of the vehicle ω, can be found for the actual motion of the vehicle as:

$$\omega = \frac{v_a}{\rho} = \frac{v_a}{l} \tan \delta_a \tag{3.16}$$

Now the longitudinal increment (x), and the lateral increment (y) in a time interval, t , in the vehicle reference frame can be computed as:

$$x = \frac{l}{tan\delta} \times sin\left(\frac{v_a}{l} \times \tan \delta_a \times t\right) \tag{3.17}$$

$$y = \frac{l}{tan\delta} \times \left(1 - cos\left(\frac{v_a}{l} \times \tan \delta_a \times t\right)\right) \tag{3.18}$$

Hence the next position of the vehicle, in the global reference frame can be found out as,

$$q_1 = q_1 + (x \times \cos q_3 - y \times \sin q_3) \tag{3.19}$$

$$q_2 = q_2 + (x \times \sin q_3 + y \times \cos q_3) \tag{3.20}$$

$$q_3 = q_3 + \frac{v_a}{l} \times \tan \delta \times t \tag{3.21}$$

These values of the position co-ordinates, give the new position and orientation of the vehicle in the global reference frame. Then this point is treated as the instantaneous position of the vehicle, and the entire procedure is repeated until the vehicle reaches the destination point. The vehicle is assumed to follow the trajectory calculated by the geometry of motion flawlessly. The small error occurring due to slip at the wheels can however be neglected.

Chapter 4

WMR Structure

The physical structure of the car-type WMR that has been constructed to realize the motion as generated by the software is discussed in this chapter. The software generates the required signals to control the motion of the stepper motors in the desired fashion. These stepper motors are mounted on the structure of the WMR, which rotate the wheels and cause the motion of the WMR. So for the motion to be smooth and error-free, the vehicle structure needs to be perfect and the structural symmetry should be retained.

4.1 The description of the vehicle:

The vehicle structure consists of the following major parts.

1. Main body
2. Stepper motor mounting arrangement
3. Speed reduction gear box in the rear wheels
4. Rear wheel assembly
5. Speed reduction gear box in the front wheel
6. Front steering wheel

The main body of the vehicle, which supports all the components is a rectangular acrylic sheet.

The stepper motors for the rear wheels are mounted on the structure with the help of an aluminium angle as shown in the Fig. 4.1.

A single speed reduction gear box is mounted on the structure as shown in Fig. 4.1 between he motor and the rear wheels. This gear box reduces the speed by 60 times. The objective of using this, is to increase the resolution of the wheels so as to increase the accuracy of the system.

The rear wheels are attached to the gear box assembly directly. These are made of wood.

The gear box for the front wheel is attached parallel to the main structure as shown in Fig. 4.1. It is coupled with the motor which is mounted on the main structure.

The steering wheel is mounted directly on the structure, It takes motion from the gearbox, with the help of a gear assembly.

4.2 Geometrical parameters of the WMR

The relevant geometrical parameters of the WMR are enumerated below:

1. Length of the vehicle (distance between the axes of the rear and the front wheel), $l = 165mm$,
2. Width of the vehicle (spacing between the rear wheels), $2b = 205mm$,
3. Diameter of rear wheels $= 140mm$,
4. Diameter of front steering wheel $= 39mm$,
5. Wheel base (height of the axis of the rear wheels above the ground) $= 85mm$,
6. Step Distance (distance covered by either of the rear wheels in one step), $step\_distance = 0.128mm$,
7. Step Angle (angle turned by the steering wheel in one step), $step\_angle = 0.001656$ *radians*.

It is to be noted that of the above geometrical parameters, the ones which are required for the mathematical model of the WMR are l, b, $step\_distance$, and $step\_angle$ only.

4.3 Components of the WMR

The main components of the WMR structure are listed in the following table:

S.No.	Part Name	Description	Quantity
1.	Stepper Motors	Unipolar, 42mm dia, 7.5 degree/step, 85 ohm/winding	3
2.	Gearbox	Reduction ratio - 1:16	3
3.	Bevel gears	66 teeth, 23mm dia	3
4.	Bevel gears	60 teeth, 30mm dia	1
5.	Bevel Gears	77 teeth, 37mm dia	1
6.	Base	Acrylic sheet, 175x250 cm	1
7.	Rear Wheels	Wooden, 140mm dia	2
8.	Steering wheel	Plastic, 39mm dia	1

Fig. 4.1 shows illustrates the main components of the WMR.

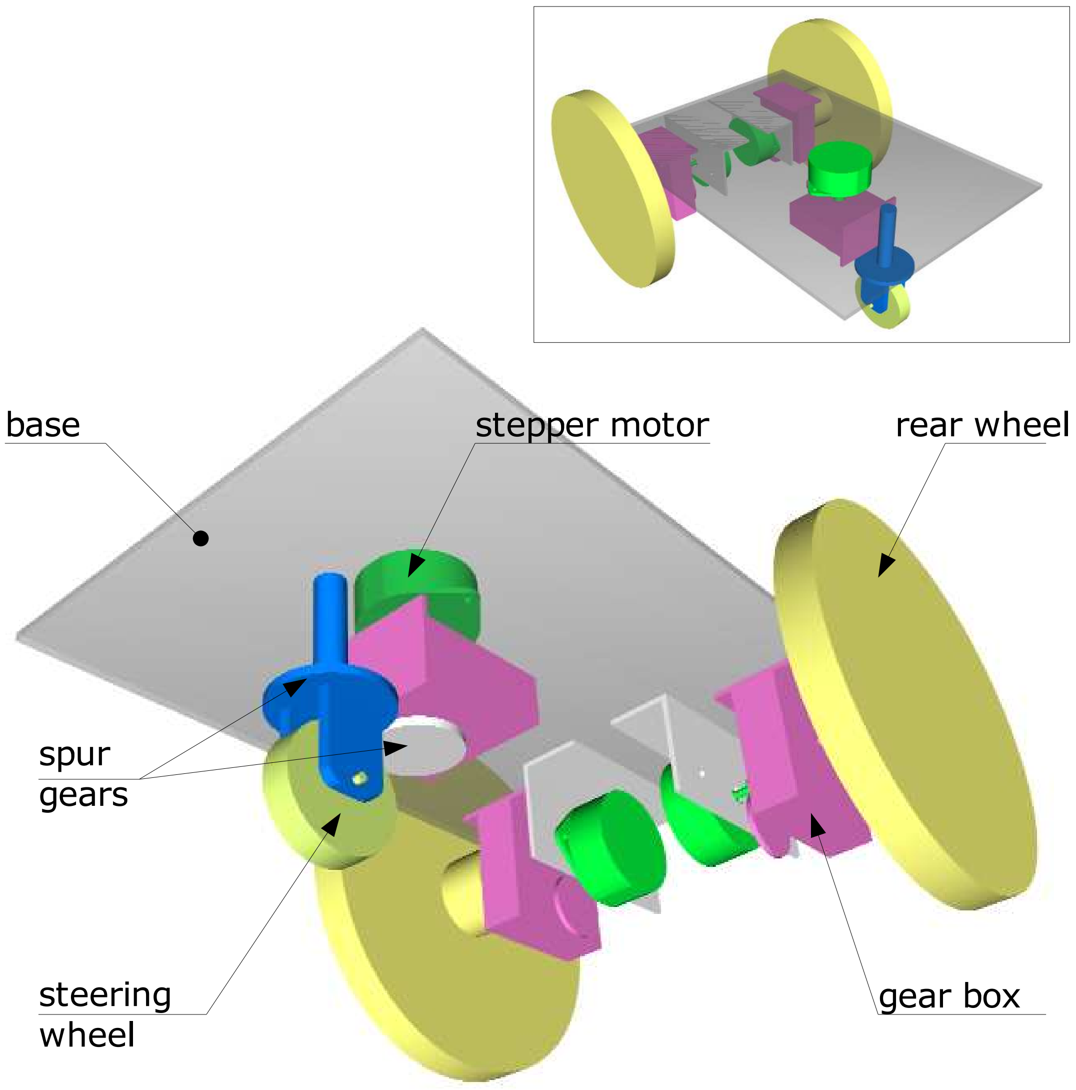


Figure 4.1: CAD model of the WMR.

Chapter 5

Driving Hardware

This chapter touches upon the driving hardware of the WMR, i.e. stepper motors and their control circuitry. It covers the basic principles of stepping motors including their classification and electronics of the basic control system. Stepper motor control software is, however discussed in chapter 6.

5.1 Stepper Motors

Stepping motors can be viewed as electric motors without commutators. Typically, all windings in the motor are part of the stator, and the rotor is either a permanent magnet or, in the case of variable reluctance motors, a toothed block of some magnetically soft material. All of the commutation must be handled externally by the motor controller, and typically, the motors and controllers are designed so that the motor may be held in any fixed position as well as being rotated one way or the other. Most steppers, can be stepped at audio frequencies, allowing them to spin quite quickly, and with an appropriate controller, they may be started and stopped instantly at controlled orientations.

For some application, there is a choice between using servomotors and stepping motors. Both types of motors offer similar opportunities for precise positioning, but they differ in a number of ways. Servomotors require analog feedback control systems of some type. Typically, this involves a potentiometer to provide feedback about the rotor position, and some mix of circuitry to drive a current through the motor inversely proportional to the difference between the desired position and the current position. In making a choice between steppers and servos, a number of issues must be considered; which of these will matter depends on the application. For example, the repeatability of positioning done with a servomotor generally depends on the geometry of the the motor rotor, while the repeatability of positioning done with a servomotor generally depends on the stability of the potentiometer and other analog components in the feedback circuit.

Stepper motors can be used in simple open-loop control systems; these are generally adequate for systems that operate at low accelerations with static loads, but closed loop control may be essential for for high accelerations, particularly if they involve variable loads. If a stepper in an open-loop control system is overtorqued, all knowledge of rotor position is lost and the system must be reinitialized. Servomotors are bot subject to this problem.

5.2 Types of stepper motors

Stepping motors come in two varieties, permanent magnet and variable reluctance (there are also hybrid motors, which are indistinguishable from permanent magnet motors from the controller's point of view). Lacking a label on the motor, you can generally tell the two apart by feel when no power is applied. Permanent magnet motors tend to "cog" as you twist the rotor with your fingers, while variable reluctance motors almost spin freely (although they may cog slightly because of residual magnetization in the rotor). You can also distinguish between the two varieties with an ohmmeter. Variable reluctance motors usually have three (sometimes four) windings, with a common return, while permanent magnet motors usually have two independent windings, with or without center taps. Center-tapped windings are used in unipolar permanent magnet motors.

Stepping motors come in a wide range of angular resolution. The coarsest motors typically turn 90 degrees per step, while high resolution permanent magnet motors are commonly able to handle 1.8 or even 0.72 degrees per step. With an appropriate controller, most permanent magnet and hybrid motors can be run in half-steps, and some controllers can handle smaller fractional steps or micro-steps.

For both permanent magnet and variable reluctance stepping motors, if just one winding of the motor is energized, the rotor (under no load) will snap to a fixed angle and then hold that angle until the torque exceeds the holding torque of the motor, at which point, the rotor will turn, trying to hold at each successive equilibrium point.

5.2.1 Variable Reluctance Motors

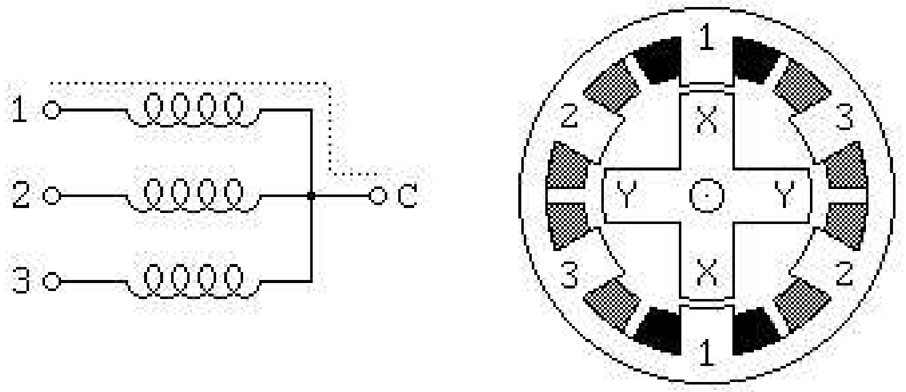


Figure 5.1: Variable Reluctance motor.

These motors have three windings, typically connected as shown in the schematic diagram in Fig. 5.1, with one terminal common to all windings. In use, the common wire typically goes to the positive supply and the windings are energized in sequence. The cross section shown in Fig. 5.1 is of 30 degree per step variable reluctance motor. The rotor in this motor has 4 teeth and the stator has 6 poles, with each winding wrapped around two opposite poles. With winding number 1 energized, the rotor teeth marked X are attracted to this winding's poles. If the current through winding 1 is turned off and winding 2 is turned on, the rotor will rotate 30 degrees clockwise so that the poles marked Y line up with the poles marked 2.

To rotate this motor continuously, we just apply power to the 3 windings in sequence. Assuming positive logic, where a 1 means turning on the current through a motor winding, the following control sequence will spin the motor illustrated in Fig. 5.1 clockwise 24 steps or 2 revolutions:

```
Winding 1 100100100100100100100100
Winding 2 010010010010010010010010
Winding 3 001001001001001001001001
          time --->
```

The motor geometry illustrated in Fig. 5.1, giving 30 degrees per step, uses the fewest number of rotor teeth and stator poles that performs satisfactorily. Using more motor poles and more rotor teeth allows construction of motors with smaller step angle. Toothed faces on each pole and a correspondingly finely toothed rotor allows for step angles as small as a few degrees.

5.2.2 Unipolar Motors

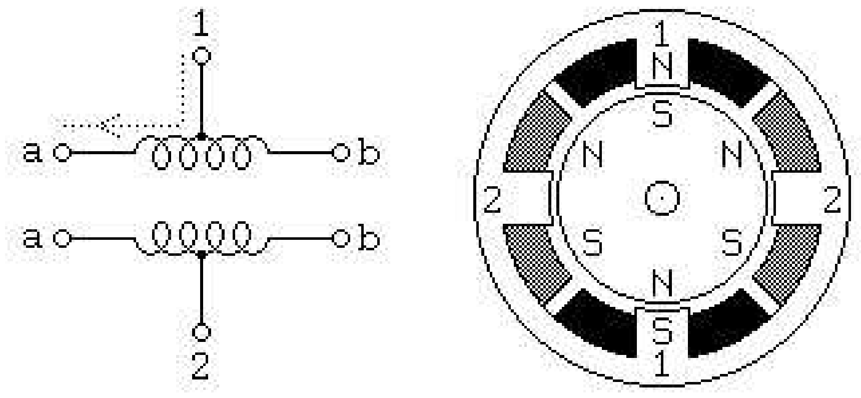


Figure 5.2: Unipolar motor.

Unipolar stepping motors, both Permanent magnet and hybrid stepping motors with 5 or 6 wires are usually wired as shown in the schematic in Fig. 5.2, with a center tap on each of two windings. In use, the center taps of the windings are typically wired to the positive supply, and the two ends of each winding are alternately grounded to reverse the direction of the field provided by that winding.

The motor cross section shown in Fig. 5.2 is of a 30 degree per step permanent magnet or hybrid motor – the difference between these two motor types is not relevant at this level of abstraction. Motor winding number 1 is distributed between the top and bottom stator pole, while motor winding number 2 is distributed between the left and right motor poles. The rotor is a permanent magnet with 6 poles, 3 south and 3 north, arranged around its circumference.

For higher angular resolutions, the rotor must have proportionally more poles. The 30 degree per step motor in the figure is one of the most common permanent magnet motor designs, although 15 and 7.5 degree per step motors are widely available. Permanent magnet motors with resolutions as good as 1.8 degrees per step are made, and hybrid motors are routinely built with 3.6 and 1.8 degrees per step, with resolutions as fine as 0.72 degrees per step available.

As shown in the figure, the current flowing from the center tap of winding 1 to terminal a causes the top stator pole to be a north pole while the bottom stator pole is a south pole. This attracts the rotor into the position shown. If the power to winding 1 is removed and winding 2 is energized, the rotor will turn 30 degrees, or one step.

To rotate the motor continuously, we just apply power to the two windings in sequence. Assuming positive logic, where a 1 means turning on the current through a motor winding, the following two control sequences will spin the motor illustrated in Fig. 5.2 clockwise 24 steps or 4 revolutions:

```
Winding 1a 1000100010001000100010001
Winding 1b 0010001000100010001000100
Winding 2a 0100010001000100010001000
Winding 2b 0001000100010001000100010
           time --->
Winding 1a 1100110011001100110011001
Winding 1b 0011001100110011001100110
Winding 2a 0110011001100110011001100
Winding 2b 1001100110011001100110011
           time --->
```

Note that the two halves of each winding are never energized at the same time. Both sequences shown above will rotate a permanent magnet one step at a time. The top sequence only powers one winding at a time, as illustrated in the figure above; thus, it uses less power. The bottom sequence involves powering two windings at a time and generally produces a torque about 1.4 times greater than the top sequence while using twice as much power. This type of energizing sequence is also known as a *two-phase on* sequence.

5.2.3 Bipolar Motors

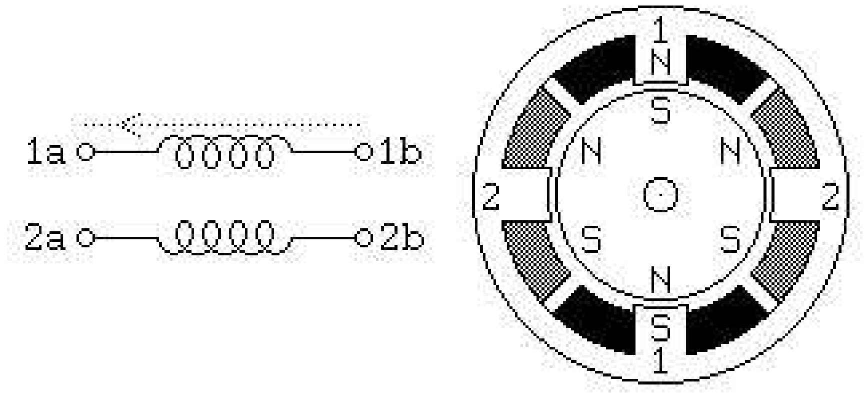


Figure 5.3: Bipolar motor.

Bipolar permanent magnet and hybrid motors are constructed with exactly the same mechanism as is used on unipolar motors, but the two windings are wired more simply, with no center taps. Thus, the motor itself is simpler but the drive circuitry needed to reverse the polarity of each pair of motor poles is more complex. The schematic in Fig. 5.3 shows how such a motor is wired, while the motor cross section shown here is exactly the same as the cross section shown in Fig. 5.2.

The control sequences for single stepping such a motor are shown below, using + and - symbols to indicate the polarity of the power applied to each motor terminal:

```
Terminal 1a +---+---+---+--- ++--++--++--++--
Terminal 1b --+---+---+---+- --++--++--++--++
Terminal 2a -+---+---+---+-- -++--++--++--++-
Terminal 2b ---+---+---+---+ +--++--++--++--+
            time --->
```

5.2.4 Bifilar Motors

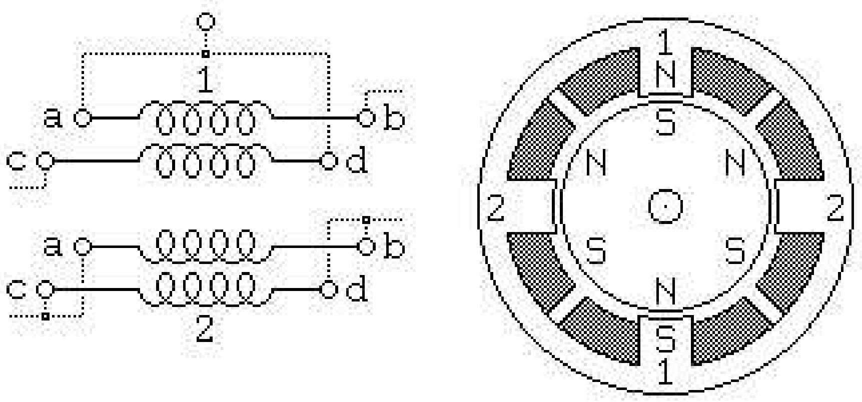


Figure 5.4: Bifilar motors.

Bifilar windings on a stepping motor are applied to the same rotor and stator geometry as a bipolar motor, but instead of winding each coil in the stator with a single wire, two wires are wound in parallel with each other. As a result, the motor has 8 wires, not four.

In practice, motors with bifilar windings are always powered as either unipolar or bipolar motors. Fig. 5.4 shows the alternative connections to the windings of such a motor.

To use a bifilar motor as a unipolar motor, the two wires of each winding are connected in series and the point of connection is used as a center-tap. Winding 1 in Fig. 5.4 is shown connected this way. To use a bifilar motor as a bipolar motor, the two wires of each winding are connected either in parallel or in series. Winding 2 in Fig. 5.4 is shown with a parallel connection.

5.2.5 Multiphase Motors

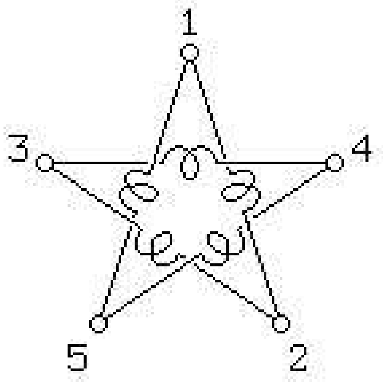


Figure 5.5: Multiphase motors.

A less common class of permanent magnet stepping motor is wired with all windings of the motor in a cyclic series, with one tap between each pair of windings in the cycle. The most common designs in this category use 3-phase and 5-phase wiring. These motors can provide more torque from a given package size because all or all but one of the motor windings are energized at every point in the drive cycle. Some 5-phase motors have high resolutions on the order of 0.72 degrees per step (500 steps per revolution).

With a 5-phase motor, there are 10 steps per repeat in the stepping cycle, as shown below:

```
Terminal 1 +++-----+++++-----++
Terminal 2 --+++++-----+++++---
Terminal 3 +-----+++++-----++++
Terminal 4 +++++-----+++++-----
Terminal 5 ----+++++-----+++++-
           time --->
```

5.3 Stepper motor control circuits

This section of the chapter deals with the basic final stage drive circuitry for unipolar stepper motors, the ones which are used in the WMR. This circuitry is centered on a single issue, switching the current in each motor winding on and off, and controlling its direction. The circuitry discussed in this section is connected directly to the motor windings and the motor power supply, and this circuitry is controlled by a digital system, which is the computer, that determines when the switches are turned on or off.

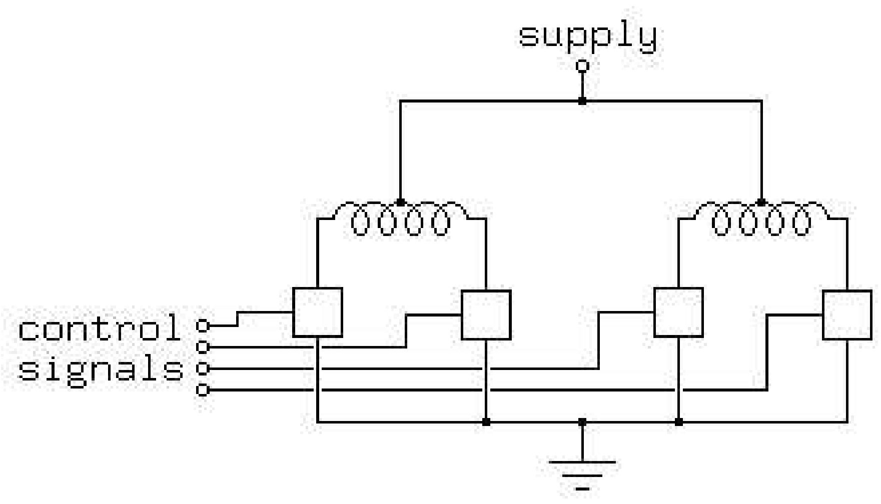


Figure 5.6: General control circuit of a unipolar stepper motor.

Typical controllers for unipolar stepping motors are variations on the outline shown in Fig. 5.6. In Fig. 5.6, boxes are used to represent switches; a control unit, not shown, is responsible for providing the control signals to open and close the switches at the appropriate times in order to spin the motors. The control unit is commonly a computer or programmable interface controller, with software directly generating the outputs needed to control the switches.

5.3.1 Practical control circuit for the WMR

The circuit shown in Fig. 5.6. is implemented by means of a *ULN2003* chip which handles the switching of the current through the four windings, according to the control signals generated by the computer, in association with the control software. The winding, which corresponds to the pin of the parallel port having a high(on) state, is energized.

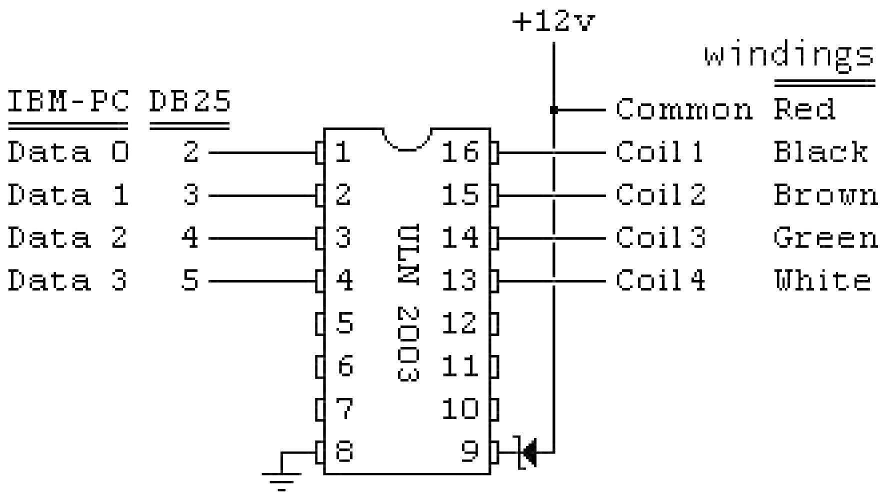


Figure 5.7: ULN2003 based unipolar motor driving circuit.

The circuit shown in the figure handles one unipolar stepper motor. However, the WMR is powered by three stepper motors. Thus each motor each stepper motor is controlled by its own *ULN2003* IC. The logic signals required for two stepper motors (4 each) are provided by data pins 2-9, which belong to parallel port address `0x378`, while the logic signals for the third stepper motor is obtained by pins numbered 1,14,16 and 17 which belong to parallel port address `0x37A` on a standard IBM compatible PC.

Chapter 6

Control Software

Software forms the core of the control system. It comprises the set of algorithms for performing the different functions involved as well as the implementation of these algorithms in the form of a computer program. The entire software system can be considered to consist of three components - stepper motor control software, WMR-specific functions which include the kinematic model, and graph plotting functions to display the status of motion in real time. The stepper motor control software contains the actual hardware level functions which generate the appropriate parallel port signals that interact with the electronic hardware. The language used for programming is C++ compiled under *TurboC++* compiler. In the sections that follow, the three software components enumerated above are discussed in detail.

The source code for the control software is listed in Appendix A.

6.1 Stepper motor control software

As discussed in the previous chapter, driving the stepper motors consist of switching the windings on and off in a particular sequence. The stepper motor control software thus centers on the generation of this sequence of signals. The sequence required at each state of the motor shaft depends on the previous state. The control software thus has a track of the current state of the motor, and it determines the next signal, which is a four bit sequence, to be generated at the parallel port according to this state by a suitable algorithm. Since the system consists of three stepper motors, the algorithm also includes the selection of the motor to be stepped and the direction in which the step is to be taken. The stepper motor control software consists essentially of a step function which takes the motor ID and direction as arguments:

```
step(motor ID, dir);
```

This step function is appropriately called by the WMR-specific functions. The basic stepping algorithm is described by Fig. 6.1. With four bits for one motor, the control of three motors require twelve bits. The parallel port organizes data pins into two sets of 8 and 4 bits each, with port addresses `0x378` and `0x37A` respectively. Thus port `0x378` handles two motors(for the rear driving wheels) and port `0x37A` handles the steering motor. The step function itself doesn't address the parallel port. After determining the next sequence set for the motor to be stepped, it calls a hardware level function which maps the three sequence sets into the bit values of the two ports.

```
outSignal();
```

The stepper motor control software contains certain additional functions for initializing the motors, for displaying the current position of the motors, and a logging function for logging all the steps and their directions taken by each motor. For initializing the motors, the motors are given the control signals corresponding to the last position in the stepping sequence. This make sure that stepping takes place, as we proceed with the first position onwards.

6.2 WMR specific functions

The kinematic model and the actual functions for controlling the motion of the WMR form the second aspect of the control software. This includes the incorporation of the WMR specific data such as the geometry, the values of the angle and the distance traveled in one step of the stepper motor in the form of program variables (`l, w, step_distance, step_angle`). The kinematic variables - velocity of the center, velocity of the left and the right rear wheels, steering angle and the co-ordinates are also specified here (`v, v_Rl, v_Rr, delta, q1, q2, q3`). The main functions defined in this part of the control software are the `setSteering()` and the `moveVehicle()` functions. The `moveVehicle()`function is the function which is actually called by the main program after it calculates the values of `v` and `delta` which are passed as arguments to this function:

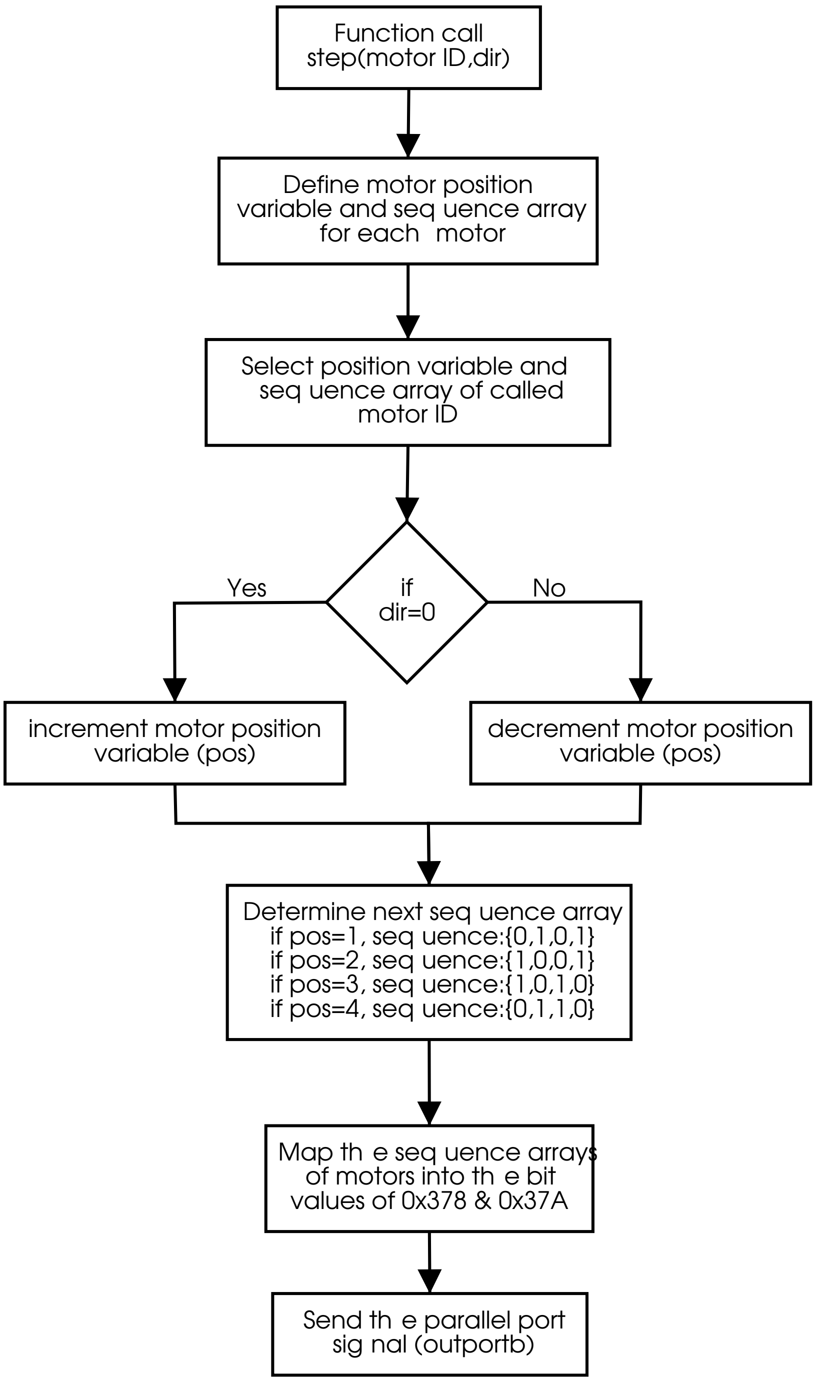

Function call
step(motor ID,dir)
Define motor position
variable and seq uence array
for each motor
Select position variable and
seq uence array of called
motor ID
if
dir=0
Yes
No
increment motor position
variable (pos)
decrement motor position
variable (pos)
Determine next seq uence array
if pos=1, seq uence:{0,1,0,1}
if pos=2, seq uence:{1,0,0,1}
if pos=3, seq uence:{1,0,1,0}
if pos=4, seq uence:{0,1,1,0}
Map th e seq uence arrays
of motors into th e bit
values of 0x378 & 0x37A
Send th e parallel port
sig nal (outportb)

Figure 6.1: Flowchart describing the stepping algorithm.

```
moveVehicle(delta, v, t, distance);
```

The parameter `t` specifies the time, in seconds, for which the WMR has to travel with the particular values of `v` and `delta`. the last argument `distance`, is optional and can be used to specify the distance for which the WMR has to travel instead of specifying the time `t`. The choice between `distance` and `t` is based on the strategy used for the WMR. The algorithm for this function is described by Fig. 6.2.

A peculiar problem encountered in the WMR motion function is that the only control over time is by means of the C++ `delay()` function. The only thing this function is capable of is to suspend the execution of the program for the specified duration. In order to step the rear motors independently, we need to step the motors at appropriate timings in the *step timing array* for the individual motors. Since there is no way to execute the stepping sequence simultaneously for the two motors, the problem is overcome by combining the step timing arrays for the individual motors into a single step timing array. The stepping sequence is finally executed by calling the `step()` function for the particular motor at each instant defined in the combined step timing array.

The second function, `setSteering()` is called by the `moveVehicle()` function itself. The `moveVehicle()` function passes the value of `delta` as argument to this function:

```
setSteering(delta);
```

This function first determines the increment in the value of the steering angle `delta` with respect to the current value. It then makes the steering motor to take the desired number of steps to reach the new value of `delta`. Fig. 6.3 shows the algorithm for the `setSteering()` function.

6.3 Plotting functions

The plotting portion of the software produces a graphical display of the instantaneous positions of the WMR in a two-dimensional co-ordinate system. It contains relevant functions for drawing the co-ordinate system, displaying the position of the WMR at any instant, displaying auxillary information on the screen such as the instantaneous co-ordinates (q1, q2, q3), and status of the WMR motion (i.e.

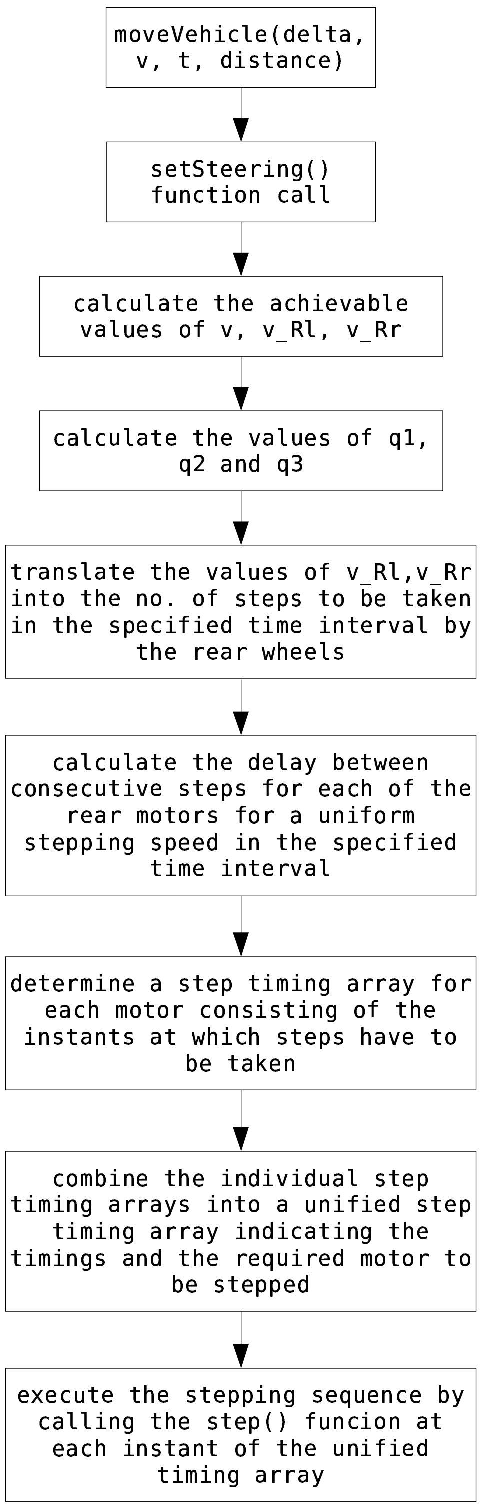

moveVehicle(delta, v, t, distance)
setSteering() function call
calculate the achievable values of v, v_Rl, v_Rr
calculate the values of q1, q2 and q3
translate the values of v_Rl,v_Rr into the no. of steps to be taken in the specified time interval by the rear wheels
calculate the delay between consecutive steps for each of the rear motors for a uniform stepping speed in the specified time interval
determine a step timing array for each motor consisting of the instants at which steps have to be taken
combine the individual step timing arrays into a unified step timing array indicating the timings and the required motor to be stepped
execute the stepping sequence by calling the step() funcion at each instant of the unified timing array

Figure 6.2: Flowchart describing the `moveVehicle()` function.

steering or moving etc.). The most important part of the plotting functions is the algorithm for mapping the WMR co-ordinate system into the screen co-ordinate system. As opposed to the WMR co-ordinate system, the screen co-ordinate system, i.e. the pixel positions start at the top-left corner of the screen and increase from left to right along the width and from top to bottom along the height. The basic transformations for mapping the WMR co-ordinate system into the screen co-ordinate system are enumerated below.

```
q1_p=q1*q1_SF+q1_offset;
q2_p=screen_h-(q2*q2_SF+q2_offset);
```

`q1_p` is the horizontal pixel co-ordinate corresponding to the `q1` axis and `q2_p` is the vertical pixel co-ordinate corresponding to the `q2` axis. Thus the WMR co-ordinates `(q1,q2)` are mapped into the screen co-ordinates `(q1_p,q2_p)`. `q1_SF` & `q2_SF` are the scaling factors along the `q1` and `q2` axis respectively. `q1_offset` and `q2_offset` are the horizontal and vertical offsets of the origin of the WMR co-ordinates from the top-left corner of the screen.

The instantaneous position of the the WMR is displayed by drawing a point at the appropriate screen co-ordinates. This is done by means of a `plot()` function which carries the transformation and draws a pixel on the screen by using low-level *TurboC++* graphics functions:

```
plot(q1,q2,special_flag)
```

The `special_flag` argument can be used to specify the color or style of the plotted point. The `plot()` function is called at appropriate situations by the WMR-specific functions when the stepping sequence of the driving rear motors is executed.

6.4 Program organization

The control software is organized into a set of header files which correspond to the three components as discussed in the beginning of this chapter. In addition to these three header files, we have the main program which includes these header files and which contains the strategy for the determination of the values of `v` and `delta`. For a simple **predetermined path** following program, the main program serves to simply input the values of `v`, `delta` and `t`. The control is then transferred to the header files which handle the relevant tasks. For **automatic tracking**, the main program contains the algorithm for calculating the values of `v`, `delta` and `t` in a loop which terminates when the desired position has been reached. The general structure of the main program is thus as follows:

```
include<stepper.h>
include<wmr.h>
include<plot.h>
main()
{
initializeMotors();
initializeGFX();
---main motion loop---
calculating v,delta;
moveVehivle(delta,v,t);
----------------------
closeGFX();
}
```

The header files are included in the beginning of the program so that the main program can access the functions defined in these header files. The functions for initializing the motors and the graphics display are called before the other routines. The we have the main motion loop of the WMR in which the `v` and `delta` are calculated according to the mode of motion i.e., trajectory tracking or automatic control. Finally, after the motion loop is executed, the graphics are closed by calling the `closeGFX()` function.

Chapter 7

Testing and Results

The following are the objectives achieved by the project.

1. The kinematics of different types of WMRs (Differentially driven and Car-type WMR) are studied. Some control strategies are discussed to control various problems of vehicle automation system. The models and strategies are simulated in MATLAB using state variable approach.

2. The control parameters for the control of the vehicle are developed from the conditions of nonslip motion and the geometrical constraints of the system.

3. Various stepper motors were tested and unipolar stepper motor was selected to use in the driving system.

4. A control circuit was made by using *ULN2003* ICs, which converted the output signals generated by the control software to the required form so as to be used by the stepper motors.

5. A control software is developed for interfacing with the driving hardware. It includes the implementation of the various algorithms pertaining to the kinematic model and strategies, software for stepper motor control and certain plotting functions for displaying the status of the WMR.

6. A visual display of the vehicle motion and a log file is generated by the control software during the actual run of the WMR for testing and analyzing the motion.

7.1 Test results

The following results were obtained during the actual run of the WMR :

- It was observed that the hardware could not admit any arbitrary value of velocity as generated by the control law or selected by the user in predefined trajectory tracking, due to the limitations put by the stepping speed of the motors.
- The vehicle speed was very small; which was due to the high gear reduction (60 times) in the wheels of the vehicle.
- Fig. 7.1 a,b and 7.2 show some of the typical simulation results as generated by the graphics display.
- A typical log file generated during execution is presented here.

A typical log file generated during execution:

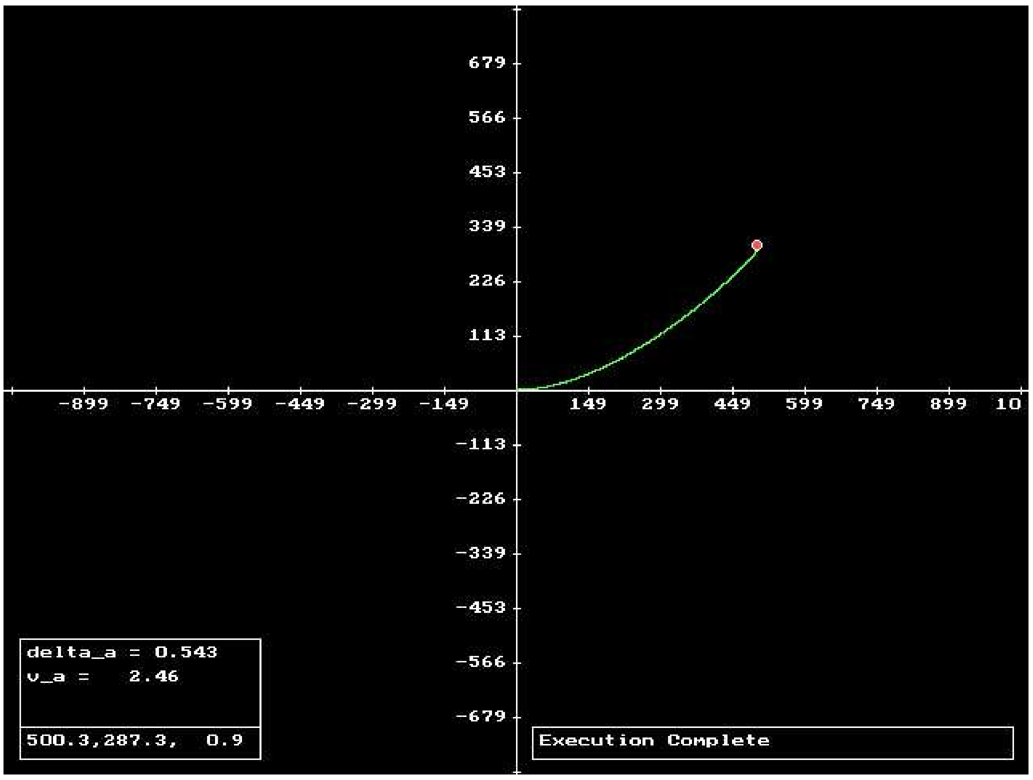
```
---Interval 1 Step Log---
delta_a=0.539883, delta_a_increment=0.539883, motor F-steps=326,
with delay=10ms, in dir=0
v_a=4.486036, v_Rl_a=2.816, v_Rr_a=6.156072
Global co-ordinates of vehicle CoG: [8.9704840.146169,0.032586]
motor Rl - 44 steps
motor Rr - 96 steps
-----------------------
---Interval 2 Step Log---
delta_a=0.515042, delta_a_increment=-0.024841, motor F-steps=15,
with delay=10ms, in dir=1
v_a=4.540436, v_Rl_a=2.944, v_Rr_a=6.136873
Global co-ordinates of vehicle CoG: [18.0404610.583327,0.063736]
motor Rl - 46 steps
motor Rr - 96 steps
-----------------------
---Interval 3 Step Log---
delta_a=0.491857, delta_a_increment=-0.023185, motor F-steps=14,
with delay=10ms, in dir=1
v_a=4.508598, v_Rl_a=3.008, v_Rr_a=6.009195
Global co-ordinates of vehicle CoG: [27.0296541.28931,0.093016]
motor Rl - 47 steps
motor Rr - 96 steps
-----------------------
```

7.2 Scope of the future work

The following are the various improvements that can be done over the existing software and hardware system in order to achieve better performance:

1. Various sophisticated control laws can be implemented to solve different problems, like parking problem, obstacle avoidance, sloving a maze etc.

2. Position determining sensors can be used in the WMR to obtain state feed-back response. These sensing information can be directly used as feedback

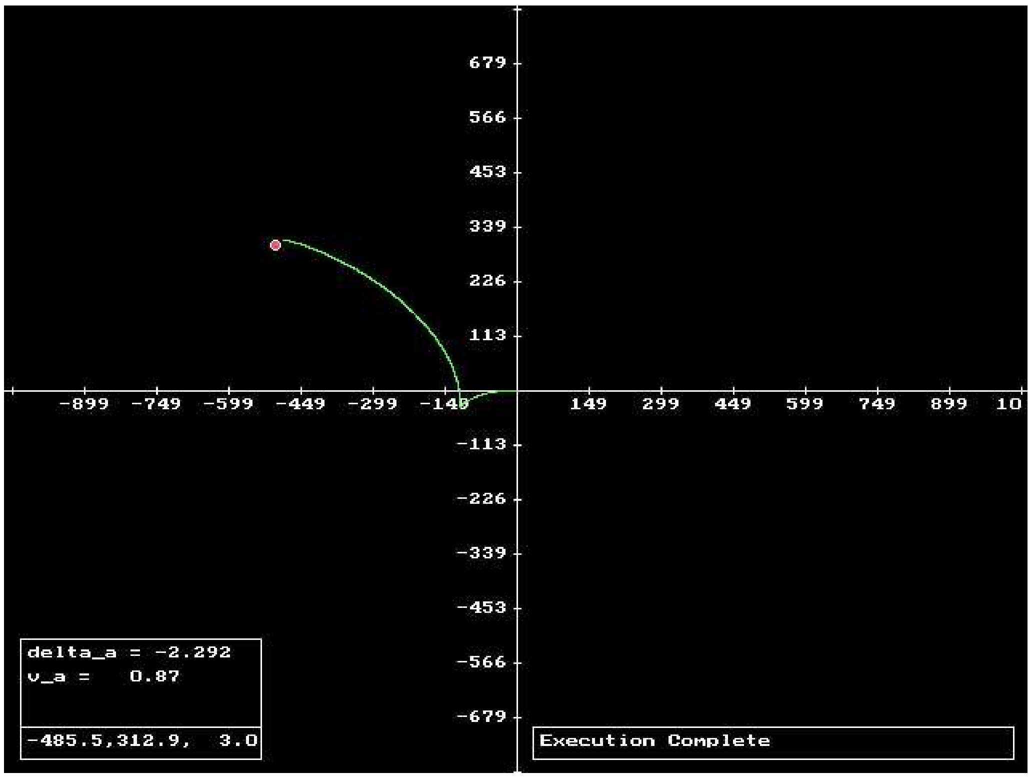


```
delta_a = 0.543
v_a =    2.46

500.3,287.3,   0.9
```

```
Execution Complete
```

(a) Trajectory while tracking a point (500,300) .



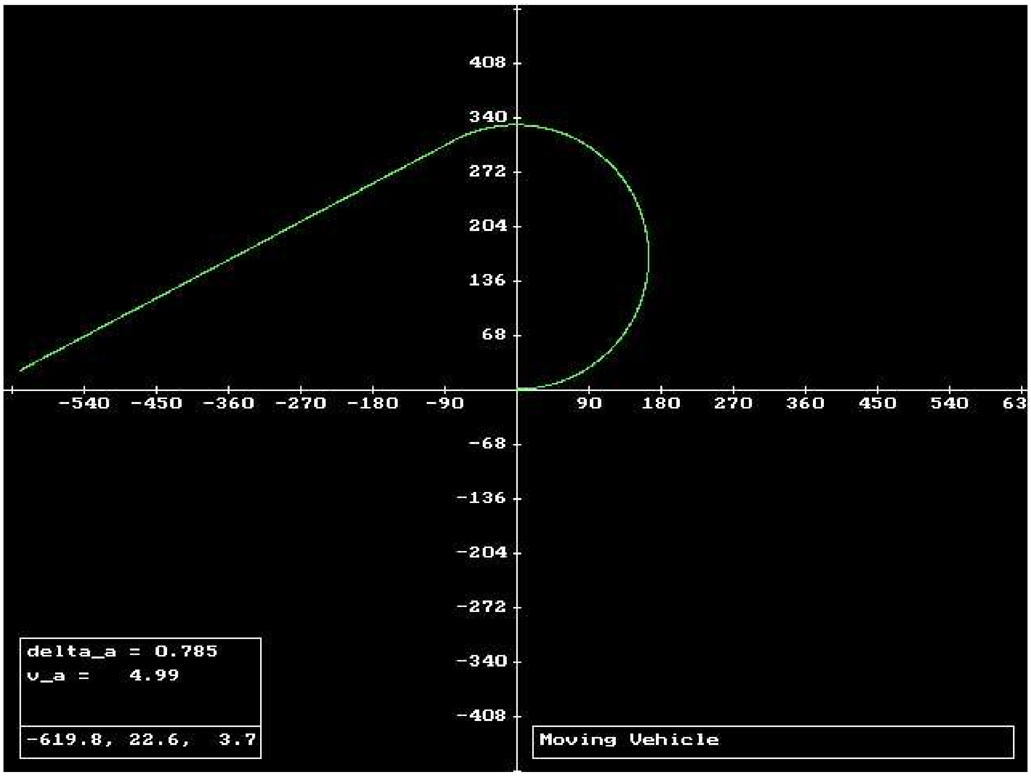
```
delta_a = -2.292
v_a =    0.87

-485.5,312.9,   3.0
```

```
Execution Complete
```

(b) Trajectory while tracing a point (-500,300) .

Figure 7.1: Simulation Results for automatic control.



```
delta_a = 0.785
v_a =    4.99

-619.8, 22.6,   3.7
```

```
Moving Vehicle
```

Figure 7.2: The WMR following an arbitary predefined path.

signal in the control strategy, instead of using kinematically calculated position of the vehicle which are theoretically calculated and can be different from the actual values.

3. The vehicle structure can be modified to increase the speed without compromising much with the acurracy.

Bibliography

[1] Alexander, J., and J. Maddocks. On the kinematics of wheeled mobile robots. In *Autonomous Robot Vehicles* (Alexander, J.,and J. Maddocks, Eds.). Springer-Verlag, New York, 1990, pp.5-24.

[2] Borenstein, J., and Y. Koren. Real time obstacle avoidance for fast mobile robots. *IEEE Trans. Systems, Man, and Cybernetics*, Vol.SMC-19, No.5, 1989, pp.1179-1187.

[3] Cox, I. Blanche: Position estimation for an autonomous robot vehicle. In *Autonomous Robot Vehicles* (Alexander, J., and J. Maddocks, Eds.). Springer-Verlag, New York, 1990, pp.1127-1132.

[4] Cox, I. Blanche - An experiment in guidance and navigation of an autonomous robot vehicle. IEEE Trans. Robotics and Autom., Vol.7, No.2, 1991, pp 193-204.

[5] Patarinski S., H. Van Brussel, and H. Thielmans. Kinematics and control of wheeled mobile robots.

[6] Sungon, L., M. Kim, Y. Youm, and W. Chung. Control of a car like mobile robot for parking problem. *Proc. of IEEE Int. Conf. on Robotics and Automation,* 1999.

[7] Jones, Douglas W. Control of Stepping Motors. Department of Computer Science, The University of IOWA. *http://cs.uiowa.edu/~jones/step/* .

Appendix A

Software Code

A.1 Header Files

A.1.1 stepper.h

```
/*
Header file containing basic 3 axis stepper motor
driving functions.
2 motors driven by 8 bits of 0x378,
one driven by 4 bits of 0x37A.
Pin-up: (0x378)-|2|3|4|5|  |6|7|8|9| (0x37A)-|1|14|16|17|
                B Br O Y   B Br O Y - Wires - B Br  O  Y
*/
#include<iostream.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include<fstream.h>
#include<iomanip.h>
// ---Global Variables---
int F[4],Rl[4],Rr[4];
int motor=1;     // F=1, Rl=2, Rr=3
int dir=0;       // 0=clockwise, 1=anticlockwise
int pos_F,pos_Rl,pos_Rr;      // step position flags
```

```
ofstream outfile("Log.txt");  // Logfile
char stat_flag='L';  // L/l=logging, D/d=display,
                     //B/b=both, N/n=none
unsigned int F_count=0,Rl_count=0,Rr_count=0;
// step counts initialized to 0
int F_pos[5000],Rl_pos[5000],Rr_pos[5000];// step log array
// ------
// ---Function Declarations---
void outSignal();
void initializeMotors();
void step(int motor, int dir);
void showMotorStatus();
void promptForStatus();
void showStepLog();
void write(int var);
void write(unsigned int var);
void write(float var);
void write(char* string);
void write(int var, int width);
void write(unsigned int var, int width);
void write(float var, int width);
// ------
void outSignal()
{
  int dec_378=0;
  for(int i=0;i<4;i++)
  {
        dec_378 += Rl[i]*pow(2,i);
  }
  for(i=4;i<8;i++)
  {
        dec_378 += Rr[i-4]*pow(2,i);
  }
  int dec_37A=0;
  for(i=0;i<4;i++)
  {
        dec_37A += F[i]*pow(2,i);
  }
  outportb(0x378,dec_378);
  outportb(0x37A,dec_37A);
}
void initializeMotors()
{
  F[0]=1;F[1]=0;F[2]=1;F[3]=1; pos_F=4;         // Front
  Rl[0]=0;Rl[1]=1;Rl[2]=1;Rl[3]=0; pos_Rl=4;    // Rear Left
  Rr[0]=0;Rr[1]=1;Rr[2]=1;Rr[3]=0; pos_Rr=4;    // Rear Right
  outSignal();
}
void step(int motor, int dir)
{
  switch(motor)
  {
        case 1:
        {
          if(dir==0)
          {
                pos_F++;
                if(pos_F>4)
                  pos_F=1;
          }
          else
          {
                pos_F--;
                if(pos_F<1)
                  pos_F=4;
          }
          switch(pos_F)
          {
                case 1:
                {
```

```
              F[0]=1;F[1]=0;F[2]=0;F[3]=0;break;
            }
            case 2:
            {
              F[0]=0;F[1]=1;F[2]=0;F[3]=0;break;
            }
            case 3:
            {
              F[0]=0;F[1]=1;F[2]=1;F[3]=1;break;
            }
            case 4:
            {
              F[0]=1;F[1]=0;F[2]=1;F[3]=1;break;
            }
      }
      //F_pos[F_count++]=dir;
      break;
    }
    case 2:
    {
      if(dir==0)
      {
            pos_Rl++;
            if(pos_Rl>4)
              pos_Rl=1;
      }
      else
      {
            pos_Rl--;
            if(pos_Rl<1)
              pos_Rl=4;
      }
      switch(pos_Rl)
      {
            case 1:
```

```
            {
              Rl[0]=0;Rl[1]=1;Rl[2]=0;Rl[3]=1;break;
            }
            case 2:
            {
              Rl[0]=1;Rl[1]=0;Rl[2]=0;Rl[3]=1;break;
            }
            case 3:
            {
              Rl[0]=1;Rl[1]=0;Rl[2]=1;Rl[3]=0;break;
            }
            case 4:
            {
              Rl[0]=0;Rl[1]=1;Rl[2]=1;Rl[3]=0;break;
            }
      }
      //Rl_pos[Rl_count++]=dir;
      break;
    }
    case 3:
    {
      if(dir==1)
      {
            pos_Rr++;
            if(pos_Rr>4)
              pos_Rr=1;
      }
      else
      {
            pos_Rr--;
            if(pos_Rr<1)
              pos_Rr=4;
      }
      switch(pos_Rr)
      {
```

```
                case 1:
                {
                  Rr[0]=0;Rr[1]=1;Rr[2]=0;Rr[3]=1;break;
                }
                case 2:
                {
                  Rr[0]=1;Rr[1]=0;Rr[2]=0;Rr[3]=1;break;
                }
                case 3:
                {
                  Rr[0]=1;Rr[1]=0;Rr[2]=1;Rr[3]=0;break;
                }
                case 4:
                {
                  Rr[0]=0;Rr[1]=1;Rr[2]=1;Rr[3]=0;break;
                }
          }
          //Rr_pos[Rr_count++]=dir;
          break;
        }
  }
  outSignal();
}
void showMotorStatus()
{
  write("\n-------------");
  write("\nF  - Pos:");write(pos_F);
  write(", Current seq:");
  for(int i=0;i<4;i++)
    write(F[i]);
  write("\nRl - Pos:");write(pos_Rl);
  write(", Current seq:");
  for(i=0;i<4;i++)
    write(Rl[i]);
  write("\nRr - Pos:");write(pos_Rr);
  write(", Current seq:");
  for(i=0;i<4;i++)
    write(Rr[i]);
  write("\n-------------");
}
void promptForStatus()
{
  do
  {
    cout<<"\nLog staus (L) / Display onscreen (D)
    / Both (B) / None (N):";
    stat_flag=getche();
  }
 while(stat_flag!='L'&&stat_flag!='l'&&
stat_flag!='D'&&stat_flag!='d'
&&stat_flag!='B'&&stat_flag!='b'&&
stat_flag!='N'&&stat_flag!='n');
}
void showStepLog()
{
  int i,j,k;
  write("\n---Step Log---");

  write("\nMotor F, steps=");write(F_count);
  write(":-\n");
  for(i=0;i<F_count;i++)
  {
        write(F_pos[i]);write(" ");
        if((i+1)%25==0)
          write("\n");
  }

  write("\nMotor Rl, steps=");write(Rl_count);
  write(":-\n");
  for(j=0;j<Rl_count;j++)
  {
        write(Rl_pos[j]);write(" ");
        if((j+1)%25==0)
          write("\n");
  }

  write("\nMotor Rr, steps=");write(Rr_count);
  write(":-\n");
  for(k=0;k<Rr_count;k++)
  {
        write(Rr_pos[k]);write(" ");
        if((k+1)%25==0)
          write("\n");
  }

  write("\n------");
}
void write(int var)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<var;
}
void write(unsigned int var)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<var;
}
```

```
void write(float var)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<var;
}
void write(char* string)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
    cout<<string;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<string;
}
void write(int var, int width)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<setw(width)<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<setw(width)<<var;
}
void write(unsigned int var, int width)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<setw(width)<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<setw(width)<<var;
}
void write(float var, int width)
{
  if(stat_flag=='D'||stat_flag=='d'||
  stat_flag=='B'||stat_flag=='b')
        cout<<setw(width)<<var;
  if(stat_flag=='L'||stat_flag=='l'||
  stat_flag=='B'||stat_flag=='b')
        outfile<<setw(width)<<var;
}
```

A.1.2 wmr.h

```
/*
Functions to move the WMR by a pre-specified path,
which may be a straight line or an arc of
some radius.
*/
#include<process.h>
// ---Global Variables---
const float step_angle=(3.14159/1897),step_distance=0.128;
                           // units in radians and mm
const float l=165,b=102.5; // l=length, 2b=width, in mm
float v=0,delta=0;         // current centroidal speed(mm/s)
                           // and steering angle(radians)
float q1=0,q2=0,q3=0;      // instantaneous global co-ordinates
                           // of vehicle CoG q1,q2 in mm and
                           // q3 in radians
float v_Rl,v_Rr;           // theoretical velovity of rear left
                           // and right wheel
float v_Rl_a=0,v_Rr_a=0;   // actual achievable velocities
float v_a=0,delta_a=0;     // actual achievable centroidal
                           // velocity and steering angle
```

```
const int steering_delay=10;     // delay for steering steps,
                                 // front motor, in ms
int interval_count=0;            // interval counter
const float delta_a_min=0.001;   // min value of delta_a for
                                // making a finite radius
float q1_int[800],q2_int[800],q3_int[800];
// instantaneous global co-ordinates in the time
interval - for plotting purposes only
// ------
// ---Function Declarations---
int round(float num);
void moveDist(float dist, float speed, int motor, int dir);
void moveAngle(float angle, float omega, int motor, int dir);
void setSteering(float _delta);
void moveVehicle(float _delta, float _v,
     float _t, float _distance);
// ------
int round(float num)
{
  if((num-floor(num))<0.5)
        return floor(num);
  else
        return ceil(num);
}
void moveDist(float dist, float speed, int motor, int dir)
{
  int steps=int(dist/step_distance);
  // dist is integral multiple of step_distance
  int time_delay=int(1000/(speed/step_distance));// time delay is in ms
  write("\nMotor ID=");write(motor);
  write(", steps=");write(steps);write(",    time_delay=");
  write(time_delay);write(", dir=");write(dir);
  for(int i=0;i<steps;i++)
  {
        step(motor,dir);
        delay(time_delay);
  }
}
void moveAngle(float angle, float omega, int motor, int dir)
{
  int steps=int(angle/step_angle);
  int time_delay=int(1000/(omega/step_angle));
  write("\nMotor ID=");write(motor);write(", steps=");
  write(steps);write(", time_delay=");write(time_delay);
  write(", dir=");write(dir);
  for(int i=0;i<steps;i++)
  {
        step(motor,dir);
        delay(time_delay);
  }
}
void setSteering(float _delta)
{
  float delta_a_inc, delta_a_inc_cur=0;
  delta_a_inc=(round(_delta/step_angle))*step_angle-delta_a;
  delta_a=(round(_delta/step_angle))*step_angle;
  write("\ndelta_a=");write(delta_a);
  write(", delta_a_increment=");write(delta_a_inc);
  write(", motor F-steps=");write(abs(round(delta_a_inc/step_angle)));
write(", with delay=");write(steering_delay);write("ms");
  if(delta_a_inc>0)
        write(", in dir=0");
  if(delta_a_inc<0)
        write(", in dir=1");
  displayStatus("Steering");

  if(delta_a_inc>0)
  {
        while(delta_a_inc_cur<delta_a_inc)
        {
          step(1,0);
      delay(steering_delay);
          delta_a_inc_cur+=step_angle;
        }
  }
  if(delta_a_inc<0)
  {
        while(delta_a_inc_cur>delta_a_inc)
        {
          step(1,1);
      delay(steering_delay);
          delta_a_inc_cur-=step_angle;
        }
  }
}
void moveVehicle(float _delta, float _v, float _t, float _distance)
{
  int dir;
  float x,y;  // x & y increments in LCS per interval
  interval_count++;
  int p=0;
  write("\n---Interval ");write(interval_count);
  write(" Step Log---");

  if(_v>=0)
        dir=0;
  else
        dir=1;

  if(_t==-1)
        _t=_distance/_v;
  setSteering(_delta);

  //Calculating v_a, v_Rl_a & v_Rr_a
  v_Rl=_v*(1-(b/l)*tan(delta_a));
```

```
v_Rl_a=(round((v_Rl*_t)/step_distance)*step_distance)/_t;
v_Rr_a=v_Rl_a*(1+(b/l)*tan(delta_a))/(1-(b/l)*tan(delta_a));
v_a=v_Rl_a/(1-(b/l)*tan(delta_a));
write("\nv_a=");write(v_a);write(",v_Rl_a=");
write(v_Rl_a);write(", v_Rr_a=");write(v_Rr_a);
//------

displayPar(delta_a,v_a);
displayPosition(q1,q2,q3);

//Calculating intermediate positions
float t_int=_t/100;
float x_int,y_int;
q1_int[0]=q1;q2_int[0]=q2;q3_int[0]=q3;
for(p=0;p<100;p++)
{
      if(fabs(delta_a)>delta_a_min)
      {
        x_int=(l/tan(delta_a))*sin((v_a/l)*tan(delta_a)*t_int);
        y_int=(l/tan(delta_a))*(1-cos((v_a/l)*tan(delta_a)*t_int));
        q1_int[p+1]=q1_int[p]+(x_int*cos(q3_int[p])
        -y_int*sin(q3_int[p]));
        q2_int[p+1]=q2_int[p]+(x_int*sin(q3_int[p])
        +y_int*cos(q3_int[p]));
        q3_int[p+1]=q3_int[p]+((v_a/l)*tan(delta_a)*t_int);
      }
      else
      {
        x_int=v_a*t_int;
        y_int=0;
        q1_int[p+1]=q1_int[p]+(x_int*cos(q3_int[p])
        -y_int*sin(q3_int[p]));
        q2_int[p+1]=q2_int[p]+(x_int*sin(q3_int[p])
        +y_int*cos(q3_int[p]));
        q3_int[p+1]=q3_int[p]+0;
```

```
  }
}
for(int r=0;r<p;r++)
{
      write("\nq1_int[");write(r);write("]=");
      write(q1_int[r]);write(" ,q2_int[");write(r);
      write("]=");write(q2_int[r]);
}
//------

//Calculating final values of q1,q2 & q3
if(fabs(delta_a)>delta_a_min)
      {
        x=(l/tan(delta_a))*sin((v_a/l)*tan(delta_a)*_t);
        y=(l/tan(delta_a))*(1-cos((v_a/l)*tan(delta_a)*_t));
        q1+=x*cos(q3)-y*sin(q3);
        q2+=x*sin(q3)+y*cos(q3);
        q3+=((v_a/l)*tan(delta_a)*_t);
      }
      else
      {
        x=v_a*_t;
        y=0;
        q1+=x*cos(q3)-y*sin(q3);
        q2+=x*sin(q3)+y*cos(q3);
        q3+=0;
  }
write("\nGlobal co-ordinates of vehicle CoG:
[");write(q1,4);write(q2,4);write(",");write(q3,4);write("]");
//------

//Calculating step timing array
if((((fabs(v_Rl_a)*_t)/step_distance)<1.002)||
(((fabs(v_Rr_a)*_t)/step_distance)<1.002))
{
      displayStatus("Parameters out of range. Exiting...");
      write("\nParameters out of range. Exiting...");
      closeGFX();
      exit(1);
      /*if(v_Rl_a<=v_Rr_a)
        v_Rl_a=(round(1.02)*step_distance)/_t;*/
}

unsigned int delay_Rl=(_t/(((fabs(v_Rl_a)*_t)/
step_distance)-1))*1000;
unsigned int delay_Rr=(_t/(((fabs(v_Rr_a)*_t)/s
tep_distance)-1))*1000;

unsigned int timer_Rl[1000],timer_Rr[1000],timer_main[2000][2];
timer_Rl[0]=0;
motor id to be stepped.
timer_Rr[0]=0;
int i=0,j=0;
while((delay_Rl*(i+1))<=(_t*1000))
{
      i++;
      timer_Rl[i]=delay_Rl*i;
}
while((delay_Rr*(j+1))<=(_t*1000))
{
      j++;
      timer_Rr[j]=delay_Rr*j;
}
int k=1,l=1,m=0,flag;
timer_main[m][0]=0; timer_main[m][1]=5;
for(k=1;k<=i;k++)
{
      flag=1;
      while((l<=j)&&(flag==1))
      {
```

```
    if(timer_Rl[k]<timer_Rr[l])
    {
          timer_main[++m][0]=timer_Rl[k];
          timer_main[m][1]=2;
          flag=0;
    }
    else
    {
          if(timer_Rl[k]==timer_Rr[l])
          {
            timer_main[++m][0]=timer_Rl[k];
            timer_main[m][1]=5;
            l++;
            flag=0;
          }
          else
          {
            timer_main[++m][0]=timer_Rr[l];
            timer_main[m][1]=3;
            l++;
          }
    }
  }
}
for(l;l<=j;l++)
{
      if(timer_Rl[i]!=timer_Rr[l])
      {
        timer_main[++m][0]=timer_Rr[l];
        timer_main[m][1]=3;
      }
}

if(stat_flag=='D'||stat_flag=='d'||
stat_flag=='B'||stat_flag=='b')
```

```
{
      cout<<"\nmotor Rl - "<<(i+1)<<" steps";
      cout<<"\nmotor Rr - "<<(j+1)<<" steps";
}
if(stat_flag=='L'||stat_flag=='l'||
stat_flag=='B'||stat_flag=='b')
{
      outfile<<"\nmotor Rl - "<<(i+1)<<
      " steps with step times:-\n";
      for(p=0;p<=i;p++)
      {
        outfile<<setw(4)<<timer_Rl[p]<<"|";
        if((p+1)%25==0)
              outfile<<"\n";
      }
      outfile<<"\nmotor Rr - "<<(j+1)<<
      " steps with step times:-\n";
      for(p=0;p<=j;p++)
      {
        outfile<<setw(4)<<timer_Rr[p]<<"|";
        if((p+1)%25==0)
              outfile<<"\n";
      }
      outfile<<"\ncombined step timing array:-\n";
      for(p=0;p<=m;p++)
      {
        outfile<<setw(4)<<timer_main[p][0]<<
        "("<<timer_main[p][1]<<")|";
        if((p+1)%25==0)
              outfile<<"\n";
      }
}
write("\n----------------------");
//------
```

```
  //Executing step sequence
  displayStatus("Moving Vehicle");
  step(2,dir);
  step(3,dir);
  p=0;
  for(int n=1;n<=m;n++)
  {
        delay(timer_main[n][0]-timer_main[n-1][0]);
        if(timer_main[n][1]==5)
        {
          step(2,dir);
          step(3,dir);
        }
        else
        {
          step(timer_main[n][1],dir);
        }

        while(((t_int*1000*p)>=timer_main[n-1][0])&&
((t_int*1000*p)<=timer_main[n][0]))
        {
          plot(q1_int[p],q2_int[p],0);
          displayPosition(q1_int[p],q2_int[p],q3_int[p]);
          p++;
        }

  }
  //------
  plot(q1,q2,0);
}
```

A.1.3 plot.h

```
/*
Plotting functions using turboc++ graphics.
Screen pixels start from top-left corner with
1st pixel as (0,0).
*/
#include<graphics.h>
#include<stdio.h>
// ---Global Variables---
const int screen_w=640,screen_h=480;
for VGAHI driver
float q1_SF=4,q2_SF=4;
co-ordinate axis respectively. can change in the main prog
int q1_offset=320,q2_offset=240; //should be a portin of
                                 //screen_w & screen_h
int box1_x=10,box1_y=395,box1_w=150,box1_h=75;
//defines the position of the display box 1
int box2_x=330,box2_y=450,box2_w=300,box2_h=20;
//defines the position of the display box 2
int position_update_int=4//specifies intervals after which
                         //intermediate co-ords are displayed
                         //on screen
// ------
// ---Function Declarations---
void initializeGFX();
void initializeGFX(float _q1_SF, float _q2_SF);
//initialize the plot
void closeGFX();
void plot(float _q1, float _q2, int flag);
//flag specifies the type of entity to be plotted.
//0=point, 1=small filled circle.
void displayPar(float _delta, float _v_a);
void displayPosition(float _q1, float _q2, float _q3);
void displayStatus(char* string);
```

```
int q1ToScreenx(float _q1);
int q2ToScreeny(float _q2);
float screenxToq1(int x);
float screenyToq2(int y);
// ------
void initializeGFX()
{
  //registerbgidriver(EGAVGA_driver);
  int driver=DETECT,mode=VGAHI;
  //q1_SF=_q1_SF;
  //q2_SF=_q2_SF;
  initgraph(&driver, &mode, "");
  setcolor(EGA_WHITE);
  setlinestyle(SOLID_LINE,0,NORM_WIDTH);
  line(screen_w/2,0,screen_w/2,screen_h-1);
  line(0,screen_h/2,screen_w-1,screen_h/2);
  setfillstyle(SOLID_FILL,EGA_BLACK);
  bar3d(box1_x,box1_y,box1_x+box1_w,box1_y+box1_h,0,0);
  line(box1_x,box1_y+55,box1_x+box1_w,box1_y+55);
  bar3d(box2_x,box2_y,box2_x+box2_w,box2_y+box2_h,0,0);
  const int marker_w=screen_w/14;
  const int marker_h=screen_h/14;
  int x,y;
  char ch[10];
  settextjustify(CENTER_TEXT,TOP_TEXT);
  x=screen_w/2;
  y=screen_h/2;
  while(x<screen_w)
  {
        x+=marker_w;
        line(x,y-2,x,y+2);
        sprintf(ch,"%i",int(screenxToq1(x)));
        outtextxy(x,y+5,ch);
  }
  x=screen_w/2;
  while(x>=0)
  {
        x-=marker_w;
        line(x,y-2,x,y+2);
        sprintf(ch,"%i",int(screenxToq1(x)));
        outtextxy(x,y+5,ch);
  }
  settextjustify(RIGHT_TEXT,CENTER_TEXT);
  x=screen_w/2;
  while(y<screen_h)
  {
        y+=marker_h;
        line(x-2,y,x+2,y);
        sprintf(ch,"%i",int(screenyToq2(y)));
        outtextxy(x-5,y,ch);
  }
  y=screen_h/2;
  while(y>=0)
  {
        y-=marker_h;
        line(x-2,y,x+2,y);
        sprintf(ch,"%i",int(screenyToq2(y)));
        outtextxy(x-5,y,ch);
  }
}
void initializeGFX(float _q1_SF, float _q2_SF)
{
  q1_SF=_q1_SF;
  q2_SF=_q2_SF;
  initializeGFX();
}
void closeGFX()
{
  getch();
  closegraph();
```

```
}
void plot(float _q1, float _q2, int flag)
//copies _q1 & _q2 of q1 & q2 used.
{
 int q1_p=_q1*q1_SF+q1_offset;
 //valid values of (q1_p,q2_p) range
 from 0,0 to 639,479 in VGAHI.
 int q2_p=screen_h-(_q2*q2_SF+q2_offset);
 switch(flag)
 {
   case 0:
   {
         putpixel(q1_p,q2_p,EGA_LIGHTGREEN);
         break;
   }

   case 1:
   {
         setfillstyle(SOLID_FILL,EGA_LIGHTRED);
         fillellipse(q1_p,q2_p,3,3);
         break;
   }
 }
}
void displayPar(float _delta_a, float _v_a)
{
  char ch[50];
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setfillstyle(SOLID_FILL,EGA_BLACK);
  bar(box1_x+1,box1_y+1,box1_x+box1_w-1,box1_y+54);
  sprintf(ch,"delta_a = %5.3f",_delta_a);
  outtextxy(box1_x+5,box1_y+5,ch);
  sprintf(ch,"v_a = %6.2f",_v_a);
  outtextxy(box1_x+5,box1_y+20,ch);
}
```

```
void displayPosition(float _q1, float _q2, float _q3)
{
  char ch[50];
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setfillstyle(SOLID_FILL,EGA_BLACK);
  bar(box1_x+1,box1_y+56,box1_x+box1_w-1,box1_y+box1_h-1);
  sprintf(ch,"%5.1f,%5.1f,%5.1f",_q1,_q2,_q3);
  outtextxy(box1_x+5,box1_y+60,ch);
}

void displayStatus(char* string)
{
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setfillstyle(SOLID_FILL,EGA_BLACK);
  bar(box2_x+1,box2_y+1,box2_x+box2_w-1,box2_y+box2_h-1);
  outtextxy(box2_x+5,box2_y+5,string);
}
int q1ToScreenx(float _q1)
{
  return int(_q1*q1_SF+q1_offset);
}
int q2ToScreeny(float _q2)
{
  return int(screen_h-(_q2*q2_SF+q2_offset));
}
float screenxToq1(int x)
{
  return (x-q1_offset)/q1_SF;
}
float screenyToq2(int y)
{
  return (screen_h-q2_offset-y)/q2_SF;
}
```

A.2 Main Programs

A.2.1 Main program for automatic control

```
#include "..\stepper.h"
#include "..\plot.h"
#include "..\wmr.h"
void main()
{
  float xd=-100, yd=100;  // co-ordinates of destination point,
                          // values in mm
  float v0=4.5,vpar=0.08,cpar=1;// parameters
  float t=2;                    // time interval in s
  float e1,e2,e3;               // local co-ordinates of destination
                                // point (mm,mm,radians)
  float x_err=20,y_err=20; // final closeness to destination point
  clrscr();
  promptForStatus();
  cout<<"\nEnter destination co-ordinates:-";
  cout<<"\nx=";
  cin>>xd;
  cout<<"y=";
  cin>>yd;
  initializeGFX(.3,.3);
  plot(xd,yd,1);
  plot(q1,q2,0);
  displayStatus("Execution Begins");
  while((fabs(xd-q1)>x_err)||(fabs(yd-q2)>y_err))
  {
        e1= (xd-q1)*cos(q3)+(yd-q2)*sin(q3);
        e2= -(xd-q1)*sin(q3)+(yd-q2)*cos(q3);
        e3= atan((yd-q2)/(xd-q1))-q3;
        if((abs(e1)>50)&&(e1!=0))
        {
        v=v0*((e1)/abs(e1));
        delta=cpar*e3;
        }
        else
        if(e1==0)
        {
        v=v0;
        delta=cpar*e3;
        }
        else
        {
        v=vpar*e1;
        delta=cpar*e3;
        }
        moveVehicle(delta, v, t, -1);
  }
  displayStatus("Execution Complete");
  closeGFX();
}
```

A.2.2 Main program for pre-defined path tracking (Trajectory tracking)

```
#include "..\stepper.h"
#include "..\plot.h"
#include "..\wmr.h"
void main()
{
clrscr();
initializeMotors();
promptForStatus();
initializeGFX(.5,.5);
plot(q1,q2,0);
displayStatus("Execution Begins");
moveVehicle(0.7853, 5, 15, -1);
moveVehicle(0, 5, 15, -1);
moveVehicle(0.7853, 5, 15, -1);
moveVehicle(0.35, 5, 15, -1);
displayStatus("Execution Complete");
showStepLog();
closeGFX();
}
```

Appendix B

MATLAB code

B.1 The MATLAB program for the differentially driven vehicle:

```
% usage: qd = regulation1( t, q, flg, k1, k2 );
% inputs:
% k1 - forward velocity constant;
% k2 - angular constant;
% p - the coefficient of angle;
% q - state variables[x; y; theta];
% t - time;
% outputs:
% qd - derivatives of states[xd; yd; thetad];
function qd = regulation1(t, q, flg, k1, k2);
function qd = wmr(t,q,flg,x,y,a,k1,k2,k3)
qd(1, 1) = -k1*((q(1) - x)*cos(q(3)) +
           (q(2) - y)*sin(q(3)))*cos(q(3));
qd(2, 1) = -k1*((q(1) - x)*cos(q(3)) +
           (q(2) - y)*sin(q(3)))*sin(q(3));
qd(3, 1) = -k2*(q(3) - a) +
           k3*(((x - q(1))*sin(q(3)) +
          (q(2) -  y)*cos(q(3)))^2)*(sin(t));
```

B.2 The MATLAB program for the car-type vehicle

```
% To model single track kinematics - provides derivatives.
% Only steer control;
% Usage: qd = stm1der(t, q, flg, xdes,
% ydes, thetades, l, vpar, cpar)
% Inputs:
% t - time
% q - sate variables [x; y; theta];
% flg - not used;
% l - length of car;
% vpar - vehicle parameters kv - length and velocity
% cpar - control parameters
% Outputs:
% qd - derivatives of state [xd; yd; thetad];
function qd = stm1der(t, q, flg, xdes, ydes,
thetades, l, cpar, vpar)
v=velocitycontrol1(t, q, xdes, ydes, vpar);
delta=steercontrol1(t, q, xdes, ydes, cpar);
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
```

B.2.1 Velocity control

```
% To provide velocity command for controlling the vehicle,
allows reversing the car;
% Usage: delta = velocitycontrol1(t, q, vpar)
% Inputs:
% t - time
% q - state [x; y; theta]
% vpar - gain
% Outputs:
% v - velocity;
function v = velocitycontrol1(t, q, xdes, ydes, vpar, dpar) ddist=(xdes-q(1
v=vpar*ddist;
```

B.2.2 Steer control

```
% To provide steer angle for controlling the vehicle;
% Usage: delta = steercontrol1(t, q, cpar)
% Inputs:
% t - time
% q - state [x; y; theta]
% cpar - k gain
% Outputs:
% delta - steer angle
function delta = steercontrol1(t, q, xdes, ydes, cpar)
alpha=atan2((ydes-q(2)),(xdes-q(1)))-q(3);
delta=cpar*alpha;
```

B.3 MATLAB program for parking problem

```
function qd = parking1(t,q,flg,l,v0,k1,k2,k3,
xd,yd,thetad,x_err,y_err,theta_err); alpha=atan(q(2)/q(1));
if((abs(q(2))<y_err)&(abs(q(3))<theta_err))
v=-k3*q(1);
delta=0;
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
else if(q(1)<0)
if(cos(alpha-q(3))>0)
```

```
v=v0;
delta=atan((-l/v)*(k2*q(3)+k1*v*(sin(q(3))/q(3))*q(2)));
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
else v=-v0;
delta=atan((-l/v)*(k2*q(3)+k1*v*(sin(q(3))/q(3))*q(2)));
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
end
else
if(cos(alpha-q(3))>0)
v=-v0;
delta=atan((-l/v)*(k2*q(3)+k1*v*(sin(q(3))/q(3))*q(2)));
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
else
v=v0;
delta=atan((-l/v)*(k2*q(3)+k1*v*(sin(q(3))/q(3))*q(2)));
qd(1,1)=v*cos(q(3));
qd(2,1)=v*sin(q(3));
qd(3,1)=v/l*tan(delta);
end
end
end
```

B.4 MATLAB program for the model having three strategic positions

```
function qd = perfect2( t, q, flg, l, delta, v0, x, y, err );
```

```
r = l /( tan(delta) );
e1 = -(q(1) - x)*cos(q(3)) - (q(2) - y)*sin(q(3));
e2 = -(x - q(1))*sin(q(3)) - (q(2) - y)*cos(q(3));
v = velcontrol2( t, q, x, y, v0 ,delta );
if ( (e1 >= 0) & (abs(e2) <= err) )
qd(1, 1) = v*cos(q(3));
qd(2, 1) = v*sin(q(3));
qd(3, 1) = 0;
elseif((((e1>=0)&(e2>err))|((e1<0)&(e2>=0)))
&(e1^2+(e2-r)^2>r^2))|((((e1>=0)&(e2<-err))|
((e1<0)&(e2<0)))&(e1^2+(e2+r)^2<r^2)))
qd(1, 1) = v*cos(q(3));
qd(2, 1) = v*sin(q(3));
qd(3, 1) = (v / l)*tan(delta);
elseif((((e1>=0)&(e2<-err))|((e1<0)&(e2<0)))
&(e1^2+(e2+r)^2>r^2))|((((e1>=0)&(e2>err))|
((e1<0)&(e2>0)))&(e1^2+(e2-r)^2<r^2)))
qd(1, 1) = v*cos(q(3));
qd(2, 1) = v*sin(q(3));
qd(3, 1) = (v/l)*tan(delta);
end
```

B.4.1 The velocity control

```
function v = velcontrol2( t, q, x, y, l, v0, err, delta)
r = l /( tan (delta) );
e1 = - (q(1) - x)*cos(q(3)) - (q(2) - y)*sin(q(3));
e2 = - (x âq (1))*sin(q(3)) - (q(2) - y)*cos(q(3));
c = sqrt(2*r*err);
if( e1^2 + e2^2 >= c^2)
v = v0;
else
v = 0;
```

```
end
```

B.5 MATLAB program for the modified model of 3 strategic position

```
function qd = perfect5( t, q, flg, l, delta,
v0, x, y, err, c, k );
r = l /(tan(delta));
deltad = deltad( t, q, flg, l, delta,
v0, x, y, err, c, k );
v = velcontrol( t, q, x, y, v0, c );
qd(1, 1) = v*cos(q(3));
qd(2, 1) = v*sin(q(3));
qd(3, 1) = (v / l)*tan(q(4));
qd(4, 1) = ( deltad - q(4) ) / (k);
```

B.5.1 The function for deltad

```
function deltad = deltad(t,q,flg,l,delta,v0,x,y,err,c,k);
r=l/(tan(delta));
e1 = - (q(1) - x)*cos(q(3)) - (q(2) - y)*sin(q(3));
e2 = - (x - q(1))*sin(q(3)) - (q(2) - y)*cos(q(3));
v = velcontrol( t, q, x, y, v0, c );
if( (e1 >= 0) & ( abs(e2) <= err ) )
deltad = 0;
elseif( ( ( (e1 >= 0) & ( abs(e2) <= err) ) &
( e1^2 + (e2 + r)^2 < r^2) ) | ( ( ( e1 >= 0 ) &
(abs(e2) <= err ) ) & ( e1^2 + ( e2 + r )^2 < r^2 ) ) )
( ( ( e1 >= 0 ) & ( abs(e2) <=err) ) &
(e1^2 + (e2 + r)^2 < r^2 ) )
deltad=0;
```

```
elseif ( ( ( ( ( e1 >= 0) & ( e2 > err ) ) |
( ( e1 < 0 ) &
( e2 >= 0 ) ) ) & ( e1^2 + (e2 - r)^2 >= r^2) ) |
( ( ( ( e1 >= 0) & ( e2 < -err ) ) |
( ( e1 < 0 ) & ( e2 < 0 ) ) ) & (e1^2 + (e2 + r)^2 < r^2)))
deltad=delta;
elseif ( ( ( ( ( e1 >= 0 ) & ( e2 < - err ) ) |
( ( e1 < 0 ) & ( e2 < 0 ) ) ) & ( e1^2 +
( e2 + r)^2 >= r^2 ) ) | ( ( ( ( e1 >= 0 ) &
( e2 > err ) ) | ( ( e1 < 0 ) & ( e2 > 0 ) ) ) &
( e1^2 + (e2 âr )^2 < r^2 ) ) )
deltad=-delta;
end
```