AUTONOMOUS ROBOT NAVIGATION BASED ON RECURRENT NEURAL NETWORKS

by

MARIAM QUSAI AL-SAGBAN

A Thesis Presented to the Faculty of the
American University of Sharjah
College of Engineering
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Mechatronics Engineering

Sharjah, United Arab Emirates
May 2012

© 2012 Mariam Qusai Al-Sagban. All rights reserved.

Acknowledgements

First and foremost, I thank Allah the Most Gracious, the Most Merciful, for giving me the will and power to complete my thesis research, and allowing me to pass through such a painful and enjoyable experience. I would like to express my deep and sincere gratitude to my supervisor, Dr. Rached Dhaouadi, for his detailed and constructive comments, and for his important support throughout this work. I wish to extend my warmest thanks to all those who have helped me with my work. Special Thanks goes to Mr. Narayanan for helping me building PCBs for my robot platform, Mrs. Lalitha for co-ordinating to share some of her lab resources, and to my brother Hamid who gave me a quick start in Linux. I am also particularly grateful to my family for their unconditional love, support, and understanding. Finally, it is with pleasure that I express my appreciation to all my friends and colleagues who supported me and aided me in both the good and hard times.

Abstract

The main objective of this research is to present a reactive navigation algorithm for wheeled mobile robots under non-holonomic constraints and in unknown environments. Two techniques are proposed: a geometrical based technique and a neural network based technique. The mobile robot travels to a pre-defined goal position safely and efficiently without any prior map of the environment by modulating its steering angle and turning radius. The dimensions and shape of the robot are incorporated to determine the set of all possible collision-free steering angles. The algorithm then selects the best steering angle candidate. In the geometrical navigation technique, a safe turning radius is computed based on an equation derived from the geometry of the problem. On the other hand, the neural-based technique aims to generate an optimized trajectory by using a user-defined objective function which minimizes the traveled distance to the goal position while avoiding obstacles. A mobile robot is developed to test the performances of the two algorithms. The results demonstrate that the algorithms are capable of driving the robot safely across a variety of indoor environments.

Search Terms: Robots, reactive navigation, obstacle avoidance, autonomous ground robots, recurrent neural networks, autonomous ground robots, navigation.

Table of Contents

List of Figures

1. Introduction

Mobile robots have rapidly evolved over the past years to encompass a wide spectrum of applications: Robots are assisting in driving vehicles, aiding in medical tasks, and taking charge in hazardous rescue missions. During the catastrophic oil spill in the Gulf of Mexico, for example, multiple underwater vehicles helped monitor the spread of oil in the ocean [4]. Another event that also highlighted the importance of robots is the Japanese Fukushima nuclear crisis [5]. Robots were used to monitor the radiation and assist in clean up operations.

Autonomous navigation is a key feature in all of these applications. It deals with the problem of navigating to a target location while avoiding collision with obstacles that may be present in the environment. The building functionc which compose an autonomous navigation system are: perception, localization, path planning, and motion control. Perception is the robot ability to perceive and extract information about its surrounding environment through sensors. Localization is the robot ability to locate its position in the global frame. Path planning is the robot ability to decide the required actions inorder to achieve its goal. Finally, motion control is the robot ability to execute a desired trajectory through its actuators [6]. Each of these functions is a research area of its own. The Robot control scheme is depicted in Figure 1.1

The focus of this thesis is path planning in unknown environment. The problem comes in several variants that assume different givens and constraints. It is classified according to the supplied information as:

- Global Path Planning: It requires full knowledge of the workspace; a global map is supplied as an input.

- Local Path Planning (Sensor-Based Path Planning): It requires partial knowledge of

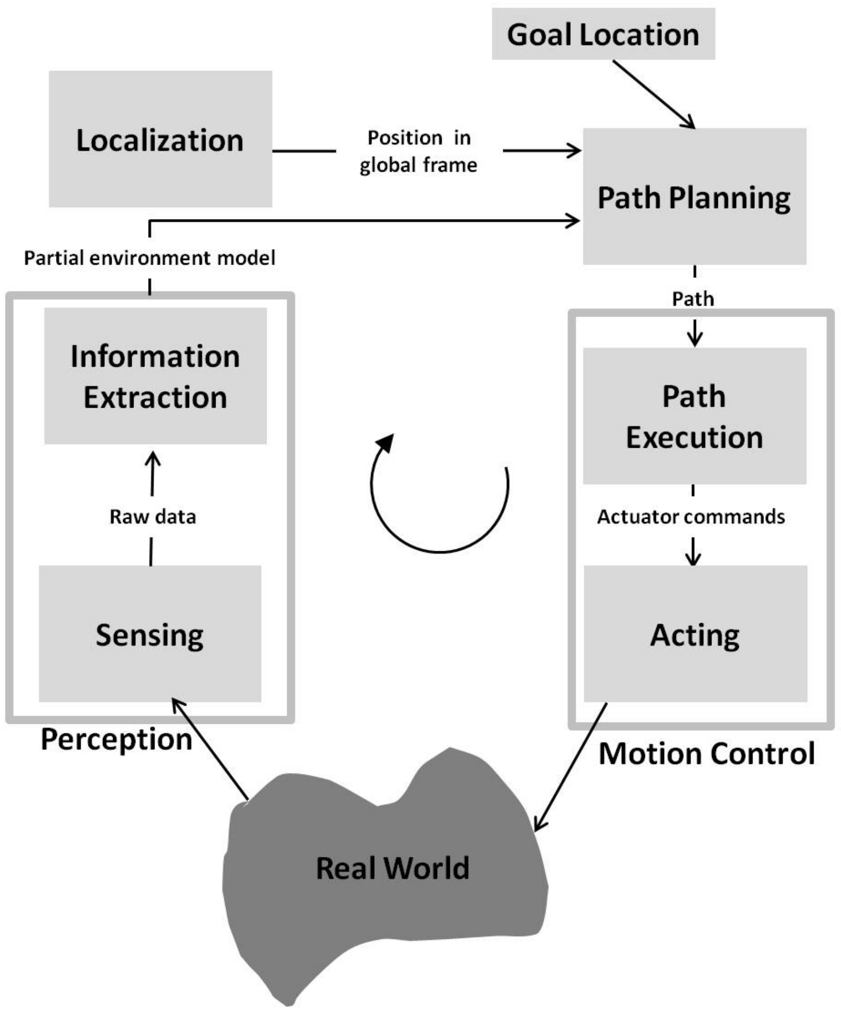


Figure 1.1: Robot scheme

the workspace; an incomplete map is supplied.

- Reactive Navigation (Obstacle Avoidance): No a priori information is required about the workspace. Instead, obstacles are discovered in real time while the robot is executing its motion.

While the algorithms developed for global path planning may produce efficient paths and guarantee global convergence, reactive navigation algorithms do not have this properties. This is due to the inherited local nature of the obstacle avoidance problem. Because a global map of the environment is not required, the robot may produce inefficient paths or converge to a local minimum (trap situation). The combination of global path planning and reactive navigation produces a sensor-based path planning strategy which has softer requirements and may be able to guarantee global convergence.

Implementing path planning and obstacle avoidance techniques on real mobile robots impose different types of constraints such as kinematic and dynamic constraints, as well as time constraints. Due to the kinematic constraint, a robot can not reach a desired point in space instantly; instead, it must follow a curve. Path planning and obstacle avoidance techniques that disregard this constraint are not safe because the transitional curve may intersect with obstacles and hence a collision may occur.

The aim of this work is to produce a novel obstacle avoidance technique for wheeled mobile robots under non-holonomic constraints and unknown environments. Experimental validation is carried out on a mobile robot built at the mechatronics center to verify the operation of the algorithm.

This thesis is organized as follows: Essential autonomous navigation concepts is presented in chapter 2. Related work on reactive navigation algorithms is discussed in chapter 3. Chapter 4 presents the autonomous navigation algorithm and chapter 5 presents the experimental results. Finally, the conclusions are drawn in chapter 6.

2. Autonomous Navigation Concepts

The Configuration Space

A complete specification of the location of every point on the robot is called a configuration, q. The set of all possible configurations is called a configuration space or C-space ($q \in \mathcal{C}$). For a two-dimensional robot, the robot configuration can be fully described by rigidly attaching a frame to the robot and then specifying the position and orientation of this frame. Thus, the configuration of a rigid object moving in the plane is specified by the triple $q = (x, y, \theta)$, and the configuration space can be represented by $\mathcal{C} = \mathbb{R}^2 \times SO(2)$, where $SO(2)$ is the special orthogonal group of 2-D rotations [7].

To describe collisions, some additional notations need to be introduced. Let $\mathcal{W}$ denote the workspace in which the robot moves. For a robot moving in a plane, the workspace can be represented by a Cartesian space $W = \mathbb{R}^2$. Let the subset of the workspace occupied by the obstacles be $O \subset \mathcal{W}$ and the subset of the workspace occupied by the robot at configuration q be $\mathcal{A}(q) \subset \mathcal{W}$. For a robot to avoid collision it should not arrive at a configuration that will bring it to physical contact with any obstacle. The set of configurations for which the robot collides with an obstacle is known as the obstacle configuration space and is defined by

$$\mathcal{C}_{obst} = \{q \in \mathcal{C} | \mathcal{A}(q) \bigcap O = \emptyset\}. \tag{2.1}$$

On the other hand, the set of collision-free configurations is known as the free configuration space. It is defined as the set difference

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst} \tag{2.2}$$

Consider a rigid robot that translates in the plane $\mathcal{W} = \mathbb{R}^2$. For this case, the robot's configuration space is two-dimensional and hence easy to visualize $\mathcal{C} = \mathbb{R}^2$. As depicted in Figure 2.1, the robot has a circular shape and there is a single obstacle in the workspace. The boundary of the obstacle configuration space can be found by sliding the robot around the workspace obstacle while tracing the configurations it went through. Motion planning for the robot in the workspace has been converted to motion planning for a point robot in the configuration space [8].

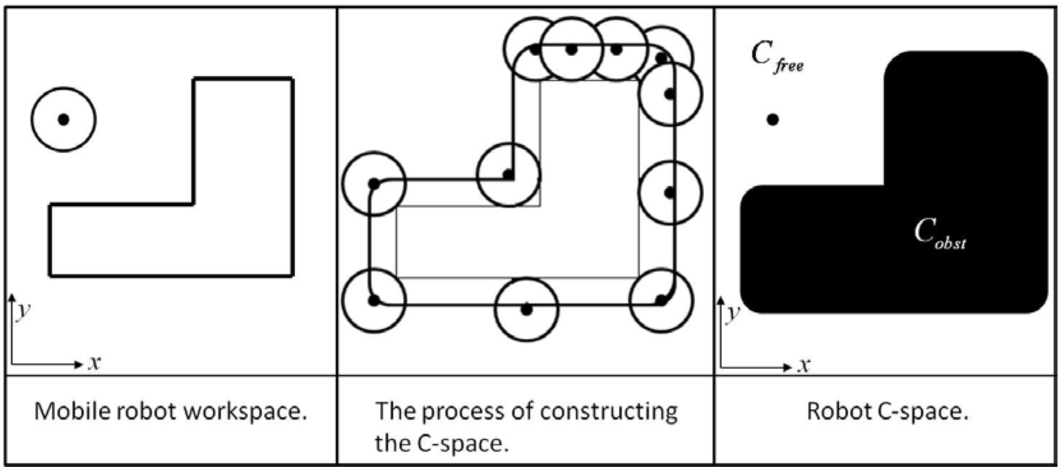


Figure 2.1: Construction of the configuration space.

Obstacle Avoidance Definition

Let q_{target} be a target configuration. In time t_i the robot is at configuration $q(t_i)$. With the aid of onboard sensors, the robot senses a portion of the environment. Let the set of workspace obstacles seen at configuration $q(t_i)$ be $O(q(t_i)) \subset \mathcal{W}$. The objective is to compute a motion control vector u_i such that:

- The trajectory does not collide with the obstacles $\mathcal{A}(Q_{t_i,T}) \bigcap O(q(t_i)) = \emptyset$,where $Q_{t_i,T}$ is the set of configurations of the trajectory followed from $q(t_i)$ to $q(t_i + T)$. $T > 0$ is the sampling period.

- It makes the vehicle progress to the target location $F(q(t_i), q_{target}) < F(q(t_i + T), q_{target})$, where $F : \mathcal{C} \times \mathcal{C} \to \mathbb{R}^+$ be a function that evaluates the progress of one configuration to another [1].

The solution of the problem is a sequence of control vectors $\{u_1, \ldots, u_n\}$ computed in real-time that guide the robot eventually to the target configuration while avoiding the sensed obstacles in the environment as shown in Figure 2.2.

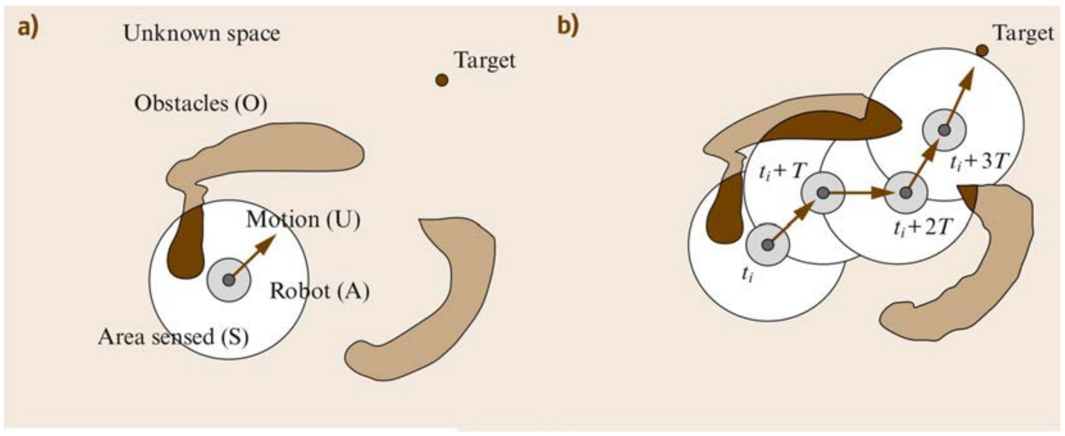


Figure 2.2: Obstacle Avoidance problem[1].

Kinematics of a Two-Wheel Differential Drive Robot

A two wheel differential drive robot is composed of two coaxial, fixed, and active wheels and one passive wheel to guarantee stability. The robot is steered by modulating the velocities of the active wheels. If the wheels have equal velocities the robot moves in a straight line; If one wheel is faster than the other the robot turns; and if both wheels turn at equal speeds but in opposite directions the robot pivots. One major advantage of this robot is the availability of a zero turning radius. Motion in any direction can be achieved by an initial rotation. In additional, this robot has a simple mechanical structure, a simple kinematic model, and low fabrication cost. On the other hand, this robot has few drawbacks: the wheels must be driven along exactly the same velocity profile, which can be challenging considering variations between wheels, motors, and environmental differences. Also, it is difficult for the robot to move on irregular surfaces. Moreover, if one active wheel loses contact with the ground, then the orientation of the robot may change abruptly [9].

There are two types of non-holonomic constraints governing the motion of the robot platform: no lateral slip constraint and pure rolling constraint [10]. The no lateral slip constraint implies that the robot's center point velocity is only in the direction of the axis of symmetry and its lateral component is zero. It is given by

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0. \tag{2.3}$$

The pure rolling constraint implies that the robot wheels have a pure rolling motion without any slipping and is given by

$$\dot{x}\cos\theta + \dot{y}sin\theta + L\dot{\theta} = \omega_r R_w, \tag{2.4}$$

$$\dot{x}\cos\theta + \dot{y}sin\theta - L\dot{\theta} = \omega_l R_w. \tag{2.5}$$

Robot kinematics refers to the equations of motion or the mathematical relations between the robot position and its right and left wheel velocities without reference to force and mass. In this section, we analyze the kinematics of a two-wheeled ground vehicle that moves on a differential drive mechanism as shown in Figure 2.3.

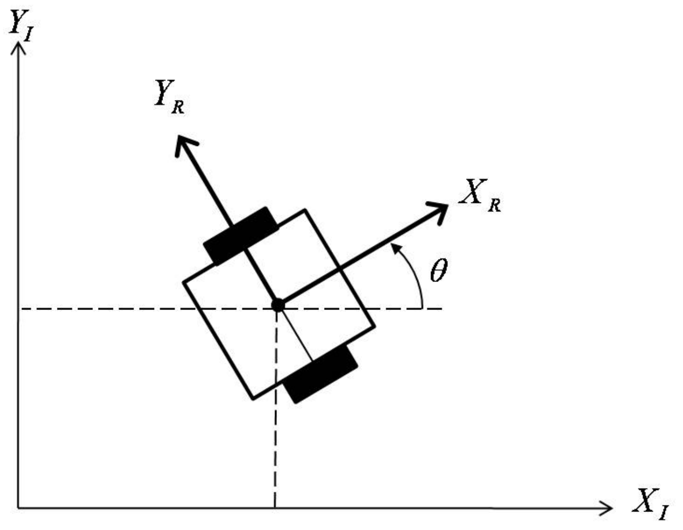


Figure 2.3: Kinematics of a two-wheel robot.

Let the rotational velocities of the left and right wheel be ω_L and ω_R respectively and R_w be the wheel radius, then assuming no slipping of the wheels, the wheels translational velocities are:

$$v_l = \omega_l R_w, \tag{2.6}$$

$$v_r = \omega_r R_w. \tag{2.7}$$

Let the robot velocity in the local frame be v, the angular velocity about its Instantaneous Center of Rotation (ICR) axis be ω, and let L be half the distance between the wheels, then:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2L} & \frac{1}{2L} \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix}. \tag{2.8}$$

Let θ be the robot orientation with respect to the global x-axis, then the robot velocity vector in the global frame is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{2.9}$$

The instantaneous turning radius r_c is shown in Figure 2.4 and can be evaluated by:

$$r_c = L\left(\frac{v_r + v_l}{v_r - v_l}\right). \tag{2.10}$$

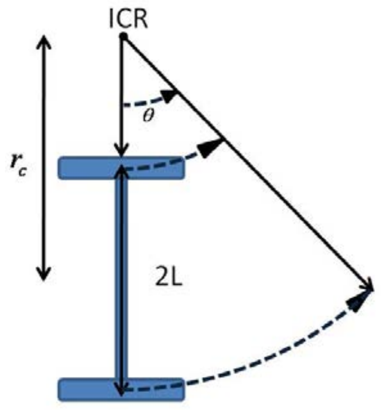


Figure 2.4: Instantaneous turning radius.

Dead Reckoning

Dead reckoning is a way to measure the robot's position in the global frame. It estimates the position by starting from a known location and integrating the incremental movements. The incremental movements are measured through wheel encoders and a compass. The robot orientation θ is right away available from the compass sensor while the robot (x, y) position is estimated by first measuring the angular velocities of the left and right wheels ω_l, and ω_r using the encoders. The encoders measure the angular velocities by recording the arrival time of the encoder pulses. Let p be the encoder resolution and Δt be the time elapsed between two consecutive rising edges of the encoder, then the wheel angular velocities can be defines as

$$\omega_{r,l} = \frac{2\pi}{p\Delta t}. \tag{2.11}$$

Next, we use the robot kinematics equations (2.6-2.9) to find the velocities components $\dot{x}$ and $\dot{y}$. Let T denote a fixed sampling time. In order to find x and y, we need to perform trapezoidal integration

$$x = x_{old} + \frac{T}{2}(\dot{x} + \dot{x}_{old}), \tag{2.12}$$

$$y = y_{old} + \frac{T}{2}(\dot{y} + \dot{y}_{old}). \tag{2.13}$$

Note that the measurements are integrated to compute the position. Therefore, the error in position accumulates over time and hence the dead reckoning cannot provide a meaningful position estimate in the long run.

Dead reckoning is subjected to two types of errors: systematic errors and non-systematic errors. Unequal diameters and misalignment of wheels contribute to systematic errors. On the other hand, possible sources of non-systematic errors are traveling over uneven floors and wheel slippage.

3. Literature Review

We describe here a number of representative techniques to solve the obstacle avoidance problem as well as a number of neural-based algorithms. Obstacle avoidance techniques can be divided into two groups: Methods that compute motion in one step, and methods that do that in more than one step. The first group directly maps the sensor information into a motion control command. An example of one step methods is the potential field method [11]. The second type computes motion in more than one step. It is further divided into two categories. The first category includes methods that compute an intermediate subset of motion controls, and then choose among them, such as the vector field histogram [12]. The second category includes methods that compute intermediate high level information and translate it to an appropriate motion control such as nearness diagram [13]. On the other hand, neural-based techniques can be classified into two categories: techniques that use multiple neural networks and methods that use a single neural network. In this section, we describe the algorithms outlined and state their benefits and drawbacks. Finally, the contribution of our proposed method is brought forward.

An elegant solution to the motion planning problem is the Potential Field Method (PFM) first proposed by Khatib in 1986 [11]. The method computes the motion command in one step by mapping the sensor information directly into a vehicle steering direction. PFM uses a physical analogy that the robot is a particle moving in the configuration space under the influence of an artificial field (see figure 3.1). The target location produces an attractive force, F_{att}, while obstacles produce repulsive forces F_{rep}. The total force, F_{tot}, acting on the robot is a weighted sum of the two forces. The total force points to the most promising direction to avoid obstacles while driving to the target configuration. $F_{att}(q_{t_i}), F_{rep}(q_{t_i}), \text{and} F_{tot}(q_{t_i})$

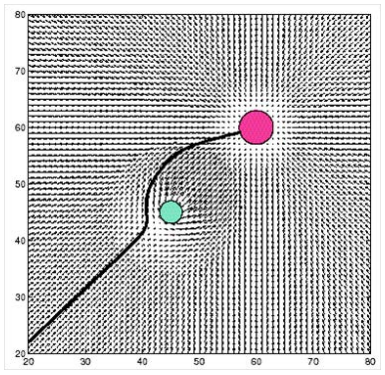


Figure 3.1: Motion Direction with PFM [2].

are given by equations 3.1, 3.2, 3.3.

$$F_{att}(q_{t_i}) = K_{att} n_{q_{target}} \tag{3.1}$$

$$R_{safe} = \begin{cases} F_{rep}(q_{t_i}) = K_{rep} \sum_j \frac{1}{d(q_{t_i}, p_j)} - \frac{1}{d_0} n_{p_j} & d(q_{t_i}, p_j) < d_0; \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

$$F_tot(q_{t_i}) = F_{att}(q_{t_i}) + F_{rep}(q_{t_i}) \tag{3.3}$$

where K_{att} and K_{rep} are the weights of the forces, d_0 is the influence distance of the obstacle p_j, q_{t_i} is the current vehicle configuration, and $n_{q_{target}}$ and n_{p_j} are the unit vectors that point from q_{t_i} to the target and each obstacle point p_j, respectively.

From equation 3.1 we can notice that the target has a global effect on the robot steering. No matter where the robot is located, there is an attractive force that is pulling it towards the target location. On the other hand, equation 3.2 shows that obstacles have a local effect on F_{tot}. Only obstacles that are within the proximity of the robot influence the steering direction.

PFM is one of the widely used techniques in motion planning. This is due to its clear mathematical formulation. However, PFM has several drawbacks. PFM does not allow the robot to temporarily move away from the target configuration. There can be some scenarios that initially requires movements away from the target. Another drawback is that PFM can get caught in local minima when $F_{att} = F_{rep}$ and hence $F_{tot} = 0$. Figure 3.2 shows an example of a local minimum. Also, PFM can produce oscillatory motion when passing through a door or a narrow corridor. The side walls produce forces that are equal in magnitude but opposite in direction if the robot is exactly at the middle. However, if the robot for example slightly deviates to the left direction, then the left wall will produce a larger force and therefore, the robot will steer to the right. After steering to the right, the right wall will produce a higher force and the robot steers left. The steering commands will keep on fluctuating between left and right resulting in oscillatory motion.

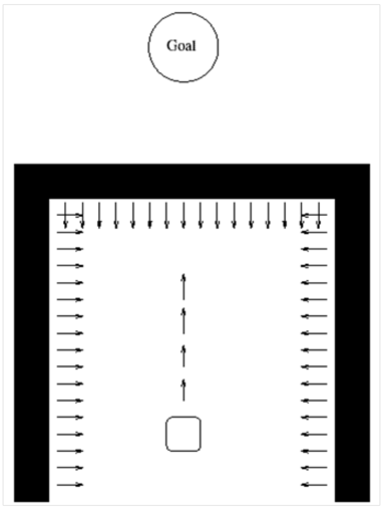


Figure 3.2: Local Minimum Problem [2].

Another popular approach to motion planning is the vector field histogram (VFH) developed by Borenstein and Koren in 1991 [12]. The environment is modeled as a two dimensional Cartesian histogram grid as shown in Figure 3.3. The value of each cell in the grid represents the certainty that an obstacle lies within the cell. A portion of this histogram is transformed to a one dimensional polar histogram. From the polar histogram the gaps that are large enough for the robot to pass through are identified. The gap closest to the goal location is used to generate the motion direction. The algorithm is further developed to take into account the robot kinematic constraints [14], and to consider the consequences of choosing a particular direction before it chooses its new direction [15]. The algorithm has the advantage of dealing with sensor uncertainties. However, the algorithm gets trapped in local minima and it is difficult to tune the empirical threshold associated with it.

- Low Safety 1 (LS1): This action moves the robot away from the nearest obstacle, and directs it towards the gap that is closest to the goal of the free walking area. The free walking area is the closest navigable region to the goal location.

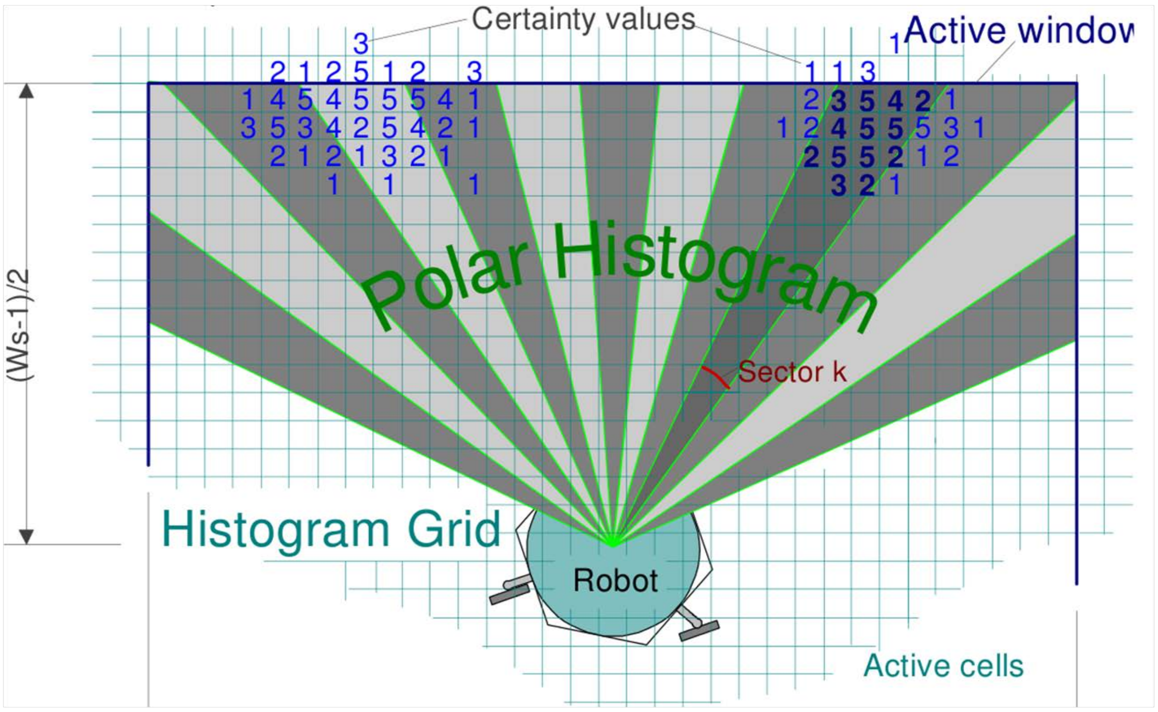


Figure 3.3: Construction of polar histogram.

- Low Safety 2 (LS2): Centers the robot between the two closest obstacles at both sides of the gap (closest to the goal) of the free walking area, while moving the robot toward this gap.
- High Safety Goal in Region (HSGR): Moves the robot towards the goal.
- High Safety Wide Region (HSWR): Moves the robot alongside the obstacle.
- High Safety Narrow Region (HSNR): Move the robot through the center of the free walking area.

The use of neural networks for robot navigation is introduced in [16]. The authors describe modularity as a manifestation of the principle of divide and conquer, which enables us to solve complex tasks by dividing them to smaller sub-tasks. The individual solutions are then combined to acquire the final solution. When applying the concept of modularity to neural networks, a better generalization is achieved because the network can be better trained. they solve the obstacle avoidance problem by using four neural networks. All of these networks cooperate to make a decision on the direction the robot should take. The space around the robot is divided into quadrants named: north, east, south, and west. The

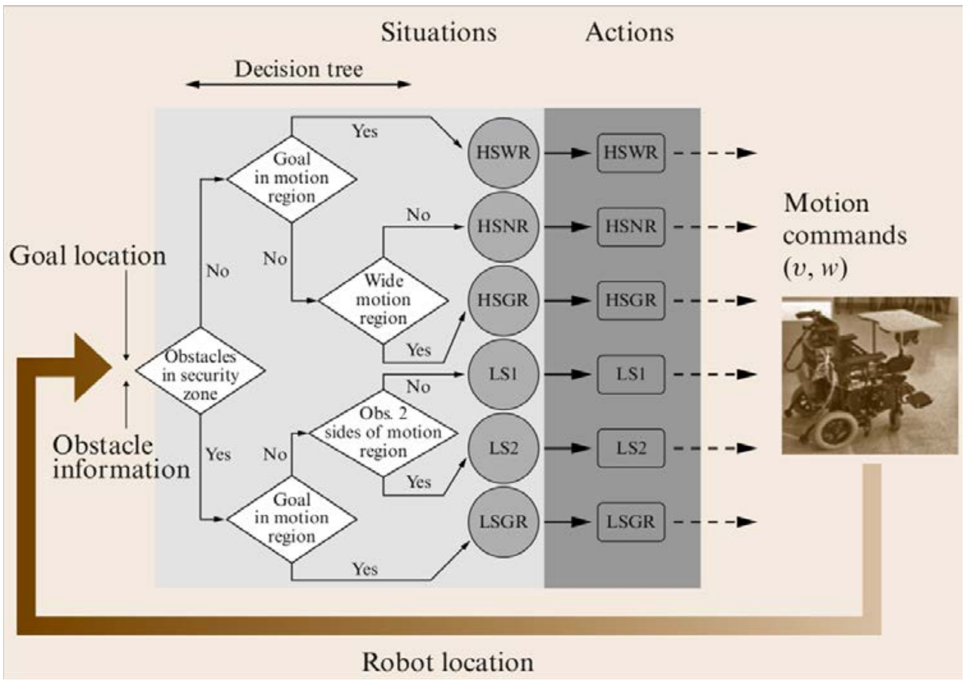


Figure 3.4: Criteria to define situations.

north is always along the front direction of the robot and is different from the earth magnetic north. The first neural network is called the north module. The output can take four possibilities: front, front right, front left, and not front. The inputs to this network are the range measurements residing in the north quadrant. The other three networks are identical and their outputs indicate the confidence that an obstacle is present in their respective quadrant. The inputs to each of these networks are the range measurements in their respective quadrant. When the north network labels the front direction as occupied (not front), the other modules' outputs are considered to determine the motion direction that the robot should follow. All networks were trained using the back-propagation algorithm and verified through simulation.

The algorithm presented in [17] uses two neural networks to solve the path planning problem. The first neural network solves the 'find space' problem while the second solves the 'find path' problem. The two neural networks are connected in cascade i.e. the output of the first network is connected to the input of the second. The role of the 'find space' network, as the name suggests, is to find the free configuration space. The area in the front side of the robot is divided into 9 space segments. If the motion across a space segment is possible then it takes a value of '1', otherwise it takes '0'. The 'find space' network is composed of four layers: input layer, Principle Component Analysis (PCA) layer, hidden layer, and output layer. The range measurements which is composed of 29 elements are fed to the input layer. Then, the PCA layer condenses the input by finding the set of uncorrelated data from the range measurements. Next, nonlinear classification of the condensed input is performed through the hidden and output layers. The output layer consists of 9 neurons where each neuron is associated with a space segment. The learning of this network is achieved through unsupervised and supervised methods. PCA condenses the input data to the principle components in an unsupervised manner while the feed-forward network requires supervised training. The second 'find path' neural network is responsible of finding the robot motion direction. The network is composed of three layers: input, hidden, and output. The output vector of the 'find space' neural network is fed as an input to the 'find path' network along with 9 additional inputs called goal segments. The goal segments are segments in the front side of the robot that contain information about the desired goal location: If a segment contains the target coordinates then it takes the value of '1', otherwise it is set to '0'. This network is trained using a decision table that contains all potential combinations of the space segments and the goal segments. The algorithm is further enhanced by adding two neural networks (H and D) to give the robot the capabilities of passing through narrow openings. The network 'H' recognizes when the robot is located in hazardous narrow locations. It acts like a switch which decides whether the output of the 'D' network or the output of the 'find path' network should be used. The 'D' network executes safe motion in narrow locations.

The autonomous navigation problem in [18] is viewed as a static pattern classification problem. The input vector consists of obstacle information and relative target orientation. The output can take one of three possible classes: right turn, forward, and left turn. The relationship between the input and output pairs of the training dataset is constructed by a decision table. A Probabilistic Neural Network (PNN) is used as a classifier. When a (testing) input vector is presented, the PNN calculates an Euclidean distance vector from the input vector to the training input vectors. The elements of the vector indicate how close the input is to a training input. Then, the elements that belong to a single class are summed. The class that produced the highest value is selected. The advantage of using PNN is that it needs no analytical model of the mapping function between the sensors data and the control action. However, a huge amount of data is required to be generated in order to train the network.

The proposed algorithm in [19] uses several neural networks to solve the obstacle avoidance problem. It divides the problem into: avoiding obstacles, following a wall, and passing through a door. A neural network is dedicated for each sub-problem. Nevertheless, all these neural networks have the same architecture and training procedure. The neural network is a feed-forward network consisting of three layers: input, output, and hidden. The input to the neural network is the robot accessible space. The accessible space is found by combining information from the laser range finder, the wheelchair dimensions, and the encoder sensors. The output of the network is the steering and velocities. The number of neurons in the hidden layer is determined by using a Bayesian framework which is an automated way to determine the optimal number of neurons in the hidden layer based on a value known as the evidence. Supervised learning is used to train the neural network. A dataset is generated for each sub-problem. The dataset for 'passing through a door' is generated by having the vehicle follow a number of predefined paths in environments that resemble the event of 'passing through a door'. The datasets for 'following a wall', and 'avoiding obstacles' are generated similarly. Based on their experimental results, the authors claimed that their algorithm produces smoother trajectories than the widely used VFH algorithm. However, it is not an equal comparison: the trajectory produced by the VFH algorithm is compared with the trajectory produced by a specific neural network. The former solves the problem of avoiding obstacles while navigating to a goal location for any obstacle arrangement, while the latter solves only the problem of avoiding obstacles for a specific obstacle arrangement. By combining the neural networks in a single algorithm via a framework that recognizes the current event and selects the appropriate network outputs not only a better comparison can be made, but also the technique will be autonomous. The authors divide the obstacle avoidance problem into sub-problems because they claimed that a single neural network could not provide the desired performance. However, It is not clear whether it is necessary to separate the 'avoiding obstacle' and 'passing through a door' tasks since they do not produce a conflict in training the network. A training conflict is created when the same input pattern is mapped to two different output values.

In this thesis, we present an obstacle avoidance technique based on recurrent neural networks that take into consideration the kinematic constraints of differential drive robots. While the common trend is to use more than one neural network, we use a single *dynamic* neural network. As discussed earlier, multiple neural networks were useful because they avoid conflict in training when a single pattern takes totally different values at different instances. In fact, this conflict in training suggests that the obstacle avoidance problem is a dynamic problem that should be solved using dynamic methods. A recurrent neural network is a dynamic network that has a context layer. The context layer provides us with a tool to look at the current input 'in context'. With such a tool, we will have a further dimension in which to classify patterns and hence conflict in training that is due to similar obstacle scenarios will no longer be encountered. Nevertheless, recurrent neural networks come at a cost. They are more difficult to train than static networks. In order to guarantee optimal convergence, we require the neural network learning environment to satisfy certain conditions that are derived using Lyapunov stability method. Also, the earlier presented neural networks techniques generated a dataset by manually driving the robot across different scenarios, as opposed to our methodology which automates the process of generating a dataset by using a computer algorithm. However, the dataset is sub-optimal because it was not generated by an expert human driver. Therefore, we train our neural network using the real-time recurrent learning algorithm along with a customized objective function to equip the robot with the capability of improving its learning while in motion.

4. Obstacle Avoidance Algorithm

The proposed obstacle avoidance algorithm can be outlined as follows: At instant t_i, the robot is at configuration $q(t_i)$ and senses a portion of the environment $\mathcal{P}(q(t_i))$. The robot uses $\mathcal{P}(q(t_i))$ to build a partial polar map of the workspace. From the polar map, the workspace obstacles set $O(q(t_i))$ is identified and the configuration space $\mathcal{C}$ is computed. Next, we operate in $\mathcal{C}$ to simplify motion planning of a robot to motion planning of a point. The desired steering angle $\gamma_{desired}$ is found by identifying all the gaps in the environment and selecting the best candidate. To take into account the non-holonomic constraints, the radius of curvature r_c is computed such that $Q_{t_i,T}$ which is the set of configurations of the trajectory followed from $q(t_i)$ to $q(t_i+T)$ does not intersect with any obstacle, $\mathcal{A}(Q_{t_i,T}) \bigcap O(q(t_i)) = \emptyset$. This is achieved by restricting the radius of curvature to an adaptive upper bound. Finally, the robot executes the control action $u_i = (\gamma_{desired}, r_c)$. The process is repeated until the robot converges to q_{target}. The algorithm is pictorially illustrated in Figure 4.1.

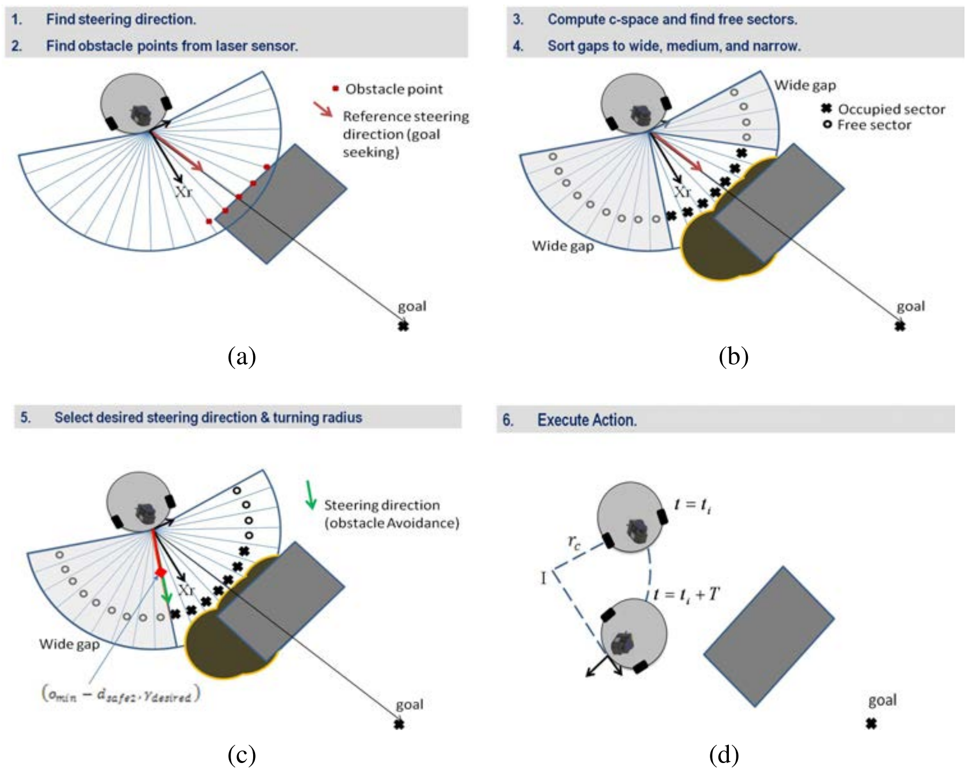

1. Find steering direction.
2. Find obstacle points from laser sensor.
Obstacle point
Reference steering direction (goal seeking)
Xr
goal
(a)
3. Compute c-space and find free sectors.
4. Sort gaps to wide, medium, and narrow.
Wide gap
Occupied sector
Free sector
Xr
Wide gap
goal
(b)
5. Select desired steering direction & turning radius
Steering direction (obstacle Avoidance)
Xr
Wide gap
$(o_{min} - d_{safe2} \cdot \gamma_{desired})$
goal
(c)
6. Execute Action.
$t = t_i$
r_c
I
$t = t_i + T$
goal
(d)

Figure 4.1: Reactive Navigation Algorithm in Action.

Identify the Reference Steering Angle

The reference steering angle, γ_{ref} represents the steering angle the robot takes in the absence of obstacles. It is an intermediate tool that will later help us find $\gamma_{desired}$. $\gamma_{desired}$ is derived as follows. Let the robot configuration shown in Figure 4.2 be:

$$q_r = (x_r, y_r, \theta_r), \tag{4.1}$$

where (x_r, y_r) is the position of the robot in the $x-y$ plane, and $\theta_r \in [0, 2\pi)$ is the robot's orientation with respect to the x axis.
Let the target configuration be:

$$q_{target} = (x_{target}, y_{target}, \theta_{target}). \tag{4.2}$$

Let $\vec{u}_e$ be the vector connecting the robot reference point to the target location. The phase angle of $\vec{u}_e$ is given by:

$$\alpha = \arctan\frac{y_{target} - y_r}{x_{target} - x_r}. \tag{4.3}$$

To correct the error in orientation, the robot should turn by a reference steering angle γ_{ref} defined as:

$$\gamma_{ref} = \alpha - \theta_r, \; -\pi \leq \gamma_{ref} \leq \pi. \tag{4.4}$$

The range of γ_{ref} is chosen such that the smaller turning angle is selected. The robot can turn clockwise (right) or counter clockwise (left). Figure 4.2 illustrates the steps taken by the robot to move to the target configuration. As time passes, the robot's x axis gets aligned with the error vector. When this is achieved, the robot proceeds in a straight line to the target.

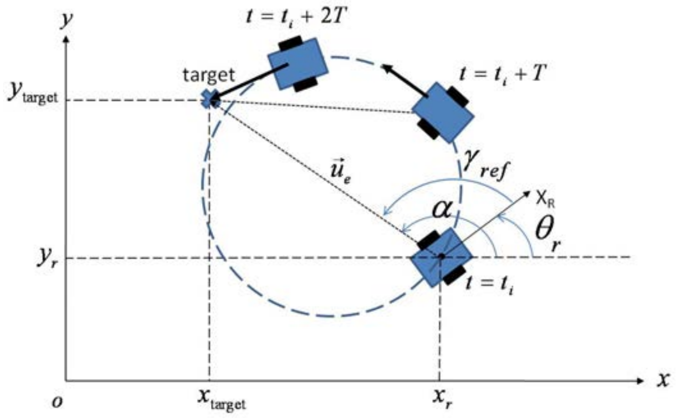


Figure 4.2: The robot's trajectory to q_{target} in the absence of obstacles.

Model the Environment

A partial polar map of the workspace is constructed in the local frame. The laser range finder is programmed to scan the 200° front view of the robot in 20 sectors, with 10° angular resolution, as shown in Figure 4.3. The sensor returns a set of points:

$$\mathcal{P}(q(t_i)) = \{p_1, p_2, ..., p_j, ..., p_{20}\}. \tag{4.5}$$

A point p_j is expressed by a pair (d_j, β_j) where d_j is the distance between the robot and

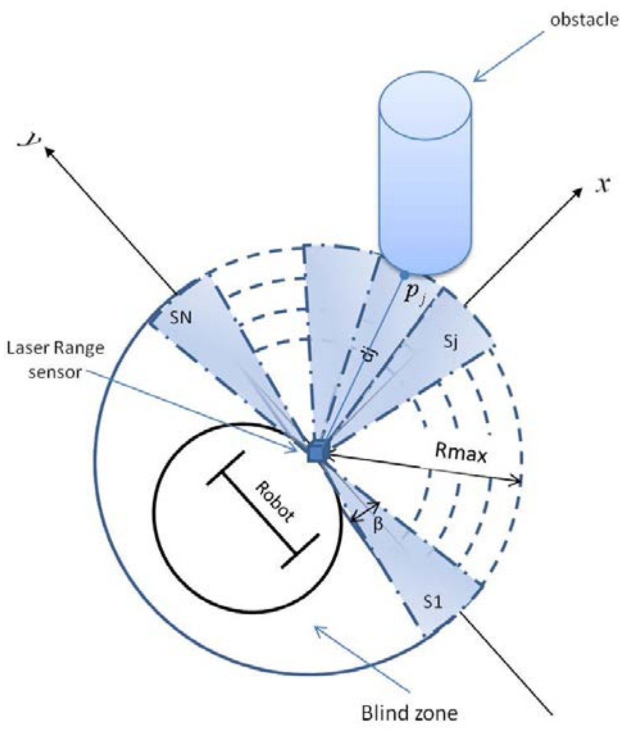


Figure 4.3: Polar map of the workspace.

the obstacle at sector j. β_j is the orientation of the j^{th} sector, S_j, with respect to the local x axis. The subset of workspace obstacles seen at configuration $q(t_i)$ is identified by applying a threshold on d_j,

$$O(q(t_i)) = \{p_j \in \mathcal{P}(q(t_i)) | d_j \leq R_{safe}\}. \tag{4.6}$$

The choice of R_{safe} plays an important role in the obstacle avoidance algorithm. If R_{safe} is large, then the obstacle avoidance will start too soon which results in a sub-optimal path. Also, by selecting a large R_{safe}, the algorithm may fail to detect any gaps in the environment and therefore incorrectly reports a trap situation. For example, the robot in Figure 4.4 successfully detects a gap in the environment with R_{safe1}, but fails to do so when using R_{safe2}. The detection range R_{safe} is allowed to take different values depending on the situation encountered:

$$R_{safe} = \begin{cases} 0.1\ m & \text{if robot is close to target configuration;} \\ 0.5\ m & \text{otherwise.} \end{cases} \tag{4.7}$$

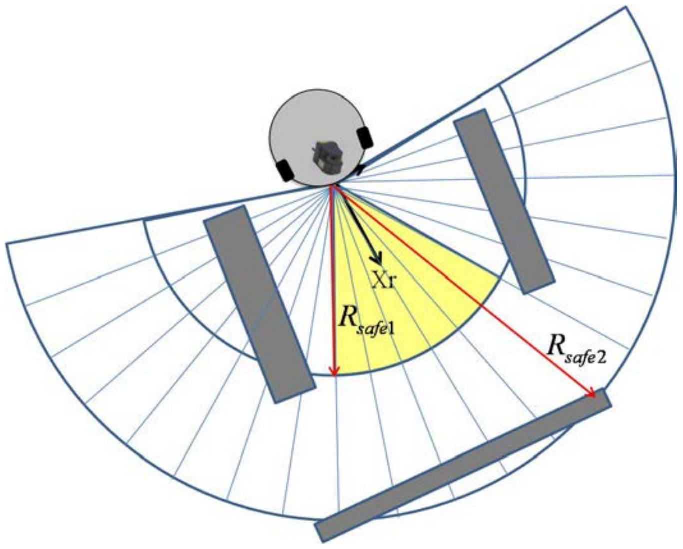


Figure 4.4: The effect of using a large value for R_{safe}.

The robot is considered close to the target configuration if:

$$(x_{target} - x_r)^2 + (y_{target} - y_r)^2 \leq \varepsilon \tag{4.8}$$

Compute the Configuration Space

Given the workspace obstacles set O, the objective is to compute C_{obst} for a 2-D robot disk of radius R. First, consider the case where only a point obstacle exists in the workspace, $O = \{p_j\}$. To find $\mathcal{C}_{obst}$, we slide the robot around p_j and trace the configurations it went through, as illustrated in Figure 4.5. Hence, $\mathcal{C}_{obst}$ is enclosed by a circle C_j of radius R and center $I_j = (I_{j,x}, I_{j,y})$:

$$\mathcal{C}_{obst} = \{q \in \mathcal{C} | (x - I_{j,x})^2 + (y - I_{j,y})^2 \leq R^2\}, \tag{4.9}$$

$$I_{j,x} = R + d_j \cos\beta_j, \tag{4.10}$$

$$I_{j,y} = d_j \sin\beta_j, -100^\circ \leq \beta_j \leq 100^\circ. \tag{4.11}$$

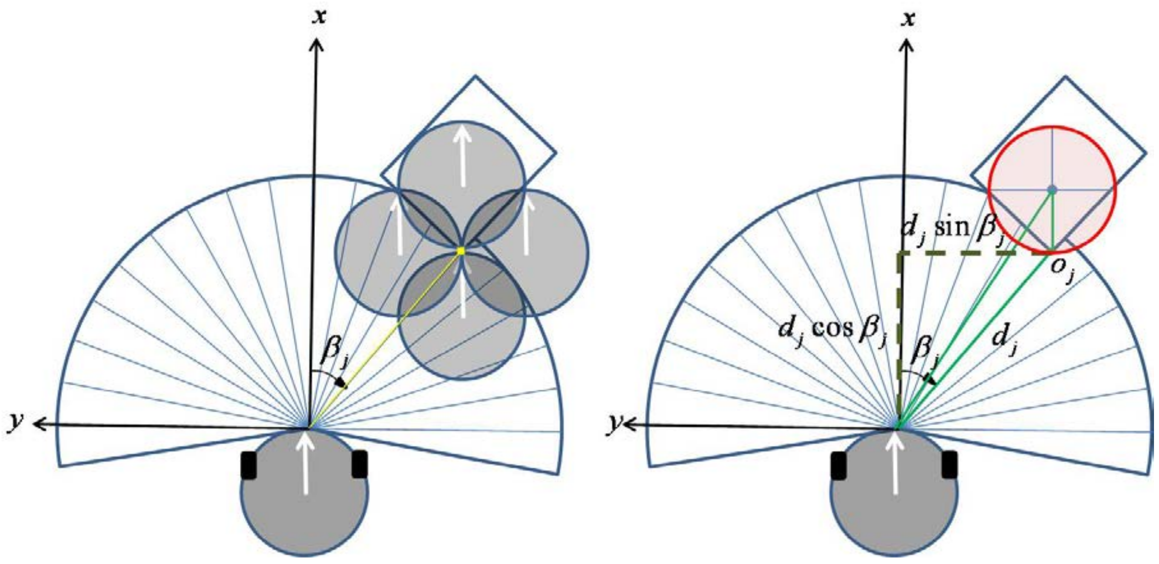


Figure 4.5: C-space Algorithm.

Next, we find L_i which is the radial distance between the robot and the boundary of $\mathcal{C}_j$ at angle β_i. The equation of C_j in polar coordinates is:

$$\rho_j = \sqrt{I_{j,x}^2 + I_{j,y}^2}, \qquad \phi_j = atan2(\frac{I_{j,y}}{I_{j,x}}), \tag{4.12}$$

$$L_i^2 + \rho_j^2 - 2\rho_j L_i \cos(\beta_i - \phi_j) = R^2. \tag{4.13}$$

Equation 4.12 can be solved for L_i, giving:

$$L_i = \min\{\rho_j \cos(\beta_i - \phi_j) \pm \sqrt{R^2 - \rho_j^2 \sin^2(\beta_i - \phi_j)}\}, \qquad \alpha_{min} \le \beta_i \le \alpha_{max}. \tag{4.14}$$

Equation 4.14 has a real value if α_{min} and α_{max} are selected as:

$$\alpha_{min} = \min\{\phi_j \pm \sin\frac{R}{\rho_j}\}, \qquad \alpha_{max} = \max\{\phi_j \pm \sin\frac{R}{\rho_j}\} \tag{4.15}$$

The above analysis is for the case when O contains a single obstacle point. In the common case where O consists of m obstacle points, C_{obst} is found by:

$$\mathcal{C}_{obst} = \bigcup_{1 \le j \le m} \mathcal{C}_j. \tag{4.16}$$

The obstacle points have been enlarged by exactly the robot radius R. Although using the exact robot dimension allows the detection of small spaces, the algorithm will be sensitive to control errors. To avoid these errors, the radius of C_j is modified to $a = R + d_{safe}$. In our implementation d_{safe} is chosen to be 20% of the robot radius.

Select Desired Steering Angle

The sectors in C are classified as free or occupied. The $j^t hsector S_j$ is occupied if $L_j \leq R_{safe}$; otherwise it is free. Adjacent free sectors are grouped together to form gaps. Let N_{free} denote the number of sectors forming a gap. The gaps are classified as:

$$gap = \begin{cases} wide & \text{if } N_{free} > 3, \\ medium & \text{if } N_{free} = 3, \\ narrow & \text{if } N_{free} < 3. \end{cases} \tag{4.17}$$

Every gap edge is a candidate for the desired steering angle. The angle of the gap edge that has the minimum cost is selected to be the desired steering angle $\gamma_{desired}$. The selection of the desired steering angle is first done within the wide gaps. If none is available the search is performed within the medium gaps. The final choice is the narrow gaps. The cost function is

$$Cost = c_1(\gamma_{ref} - \beta_j) + c_2\beta_j. \tag{4.18}$$

The first term in the cost function represents how close the desired steering direction is to the goal location and the second term represents how close the current steering direction is to the current robot heading. The coefficients c_1 and c_2 are chosen to be 0.7 and 0.3 respectively. However, if the robot detects oscillation in its motion then c_1 and c_2 are chosen to be 0.3 and 0.7 respectively. This choice give more weight for steering angles that produces smother trajectory. The robot is considered to be in oscillatory motion if one of the the following conditions is true:

$$A(t) = R \,\&\, A(t-T) = L \,\&\, A(t-2T) = R, \tag{4.19}$$

$$A(t) = L \,\&\, A(t-T) = R \,\&\, A(t-2T) = L, \tag{4.20}$$

where $A(t)$ is the action taken by the robot, R and L refer to turn right and turn left motions respectively, and T is the sampling time. R_{safe} associated with an oscillation motion is used for the next 5 samples after an oscillation is detected. Figure 4.6 illustrates a case where the robot trajectory oscillates. At time t_0, the polar map has two gaps: G1 and G2. The robot steers towards G2 because it is closer to q_{target}. However, at $t = t_0 + T$, the robot achieves a better view and therefore G2 no longer exists. Hence, the robot steers towards G1. This action will bring the robot back to its initial state and G1 and G2 will appear in the polar map. The robot keeps performing the same action again and again and as a result it produces an oscillatory trajectory and gets trapped in this loop.

a cost function is introduced to select a steering angle that will maintain a smooth trajectory as will be described later in this chapter.

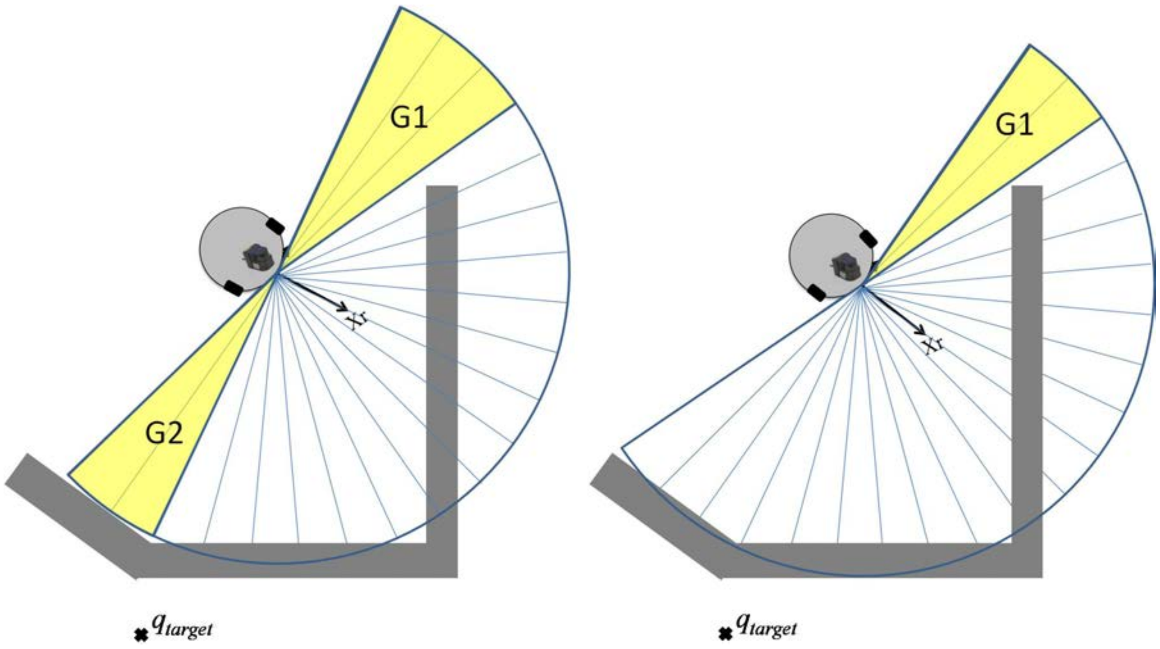


Figure 4.6: A problematic situation where the robot trajectory oscillates.

The pseudo code for selecting a desired steering angle is given as:

```
Let $\mathcal{G}$ be the set of angles that falls in a gap.
Let $\mathcal{G}_{wide}$ be the set of angles that falls in a wide gap.
Let $\mathcal{G}_{medium}$ be the set of angles that falls in a medium gap.
Let $\mathcal{G}_{narrow}$ be the set of angles that falls in a narrow gap.
if $\gamma_{ref} \in \mathcal{G}$ then
    $\gamma_{desired} = \gamma_{ref}$
else
    if $\mathcal{G}_{wide} \neq \phi$ then
```

```
        $$\gamma_{desired} = \arg \min_{\beta_j \in \mathcal{G}_{wide}} Cost(\beta_j).$$
    else if $\mathcal{G}_{medium} \neq \phi$ then
        $$\gamma_{desired} = \arg \min_{\beta_j \in \mathcal{G}_{medium}} Cost(\beta_j).$$
    else if $\mathcal{G}_{narrow} \neq \phi$ then
        $$\gamma_{desired} = \arg \min_{\beta_j \in \mathcal{G}_{narrow}} Cost(\beta_j).$$
    else
        Turn 180° around.
    end if
end if
```

Select Desired Radius of Curvature

Due to the kinematic constraints, the robot can not achieve the desired steering angle instantly. Instead, the robot follows a circular arc if the wheels' velocities are constant. The path from the initial configuration to the final configuration may intersect with $\mathcal{C}_{obst}(q(t_i))$ causing a collision. Figure 4.7 shows a case where a collision occurs because the robot steered left with a relatively large radius. Therefore, using a radius of curvature that is a function of the surrounding obstacles is safer than using a fixed radius for all obstacle scenarios. Let S_0 be the sector that contains the local x axis and $S_{desired}$ the sector that contains the desired steering angle $S_{desired}$. Let L_j^m be the distance between the obstacle point o_j and the reference point m shown in Figure 4.8. The relationship between L_j and L_j^m is given by:

$$L_j^m = \sqrt{(L_j cos\beta_j + a)^2 + (L_j sin\beta_j)^2}, \tag{4.21}$$

where a is the distance between the robot reference point and the reference point m. Define L_{min} as the distance of the nearest obstacle point that exists anywhere between S_0 and $S_{desired}$. The turning radius r_c is chosen such that the trajectory passes through the point $(L_{min}, \gamma_{desired})$ as shown in Figure 4.8. The turning radius is derived as follows. Consider the isosceles triangle where the two equal sides have length r_c and the remaining side has

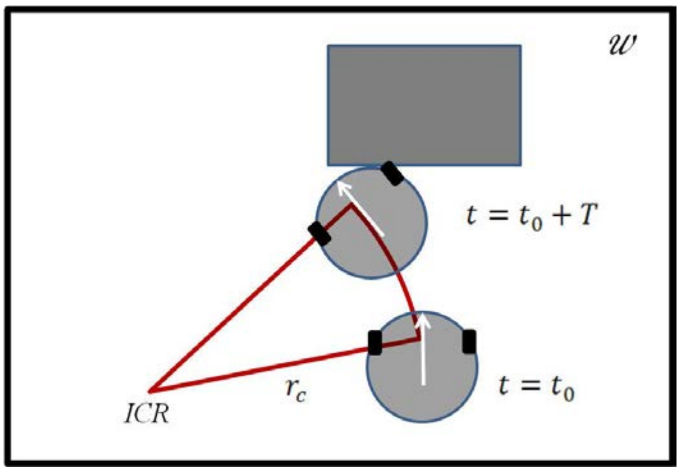


Figure 4.7: The robot collides with an obstacle because it uses a large turning radius

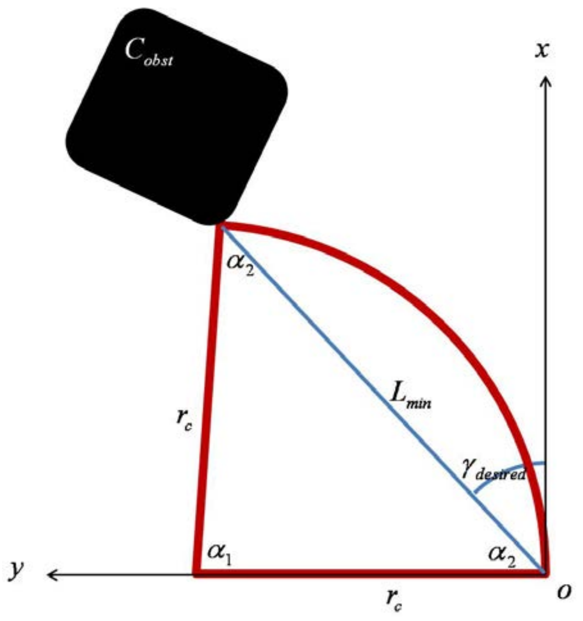


Figure 4.8: Turning Radius Selection.

length L_{min}. From the law of cosines,

$$L_{min}^2 = 2r_c^2 - 2r_c^2 \cos\alpha_1. \tag{4.22}$$

α_1 can be found as

$$\alpha_1 + 2\alpha_2 = 180, \tag{4.23}$$

$$\alpha_2 = 90 - \gamma_{desired}, \tag{4.24}$$

$$\Rightarrow \alpha_1 = 2\gamma_{desired}. \tag{4.25}$$

Using the double angle formula and equation 4.22, we can find r_c as

$$r_c = \frac{L_{min}}{2sin(\gamma_{desired})}. \tag{4.26}$$

To include a safety buffer, the turning radius is designed to pass through the point $(L_{min} - d_{safe2}, \gamma_{desired})$ instead. Also, the turning radius r_c saturates if it is greater than a threshold value r_{large}. In our implementation, d_{safe2} is selected to be $1.2R$ and $r_{large} = 0.5m$.

5. Dynamic Neural Networks

Artificial Neural Networks (ANNs) are classified into Static Neural Networks (SNNs) and Dynamic Neural Networks (DNNs). Neurons in SNN are soley connected via feedforward connections. Such networks are memmory-less, therefore, it statically maps the input space to the output space. Applications suitable for this type of networks include static pattern recognition. On the other hand, neurons in DNN are allowed to have feedback loops. Hence, DNN is capable of capuring temporal dependencies. Appliations for this type of networks include sequential pattern recognition (such as speech recognition), and time series prediction that are beyond the power of SNN. One type of DNNs is Diagonal Recurrent Networks (DRNs). Figure 5.1 depicts the network architecture and the notation used. The network weights are divided into 3 groups: Weights between input and hid-

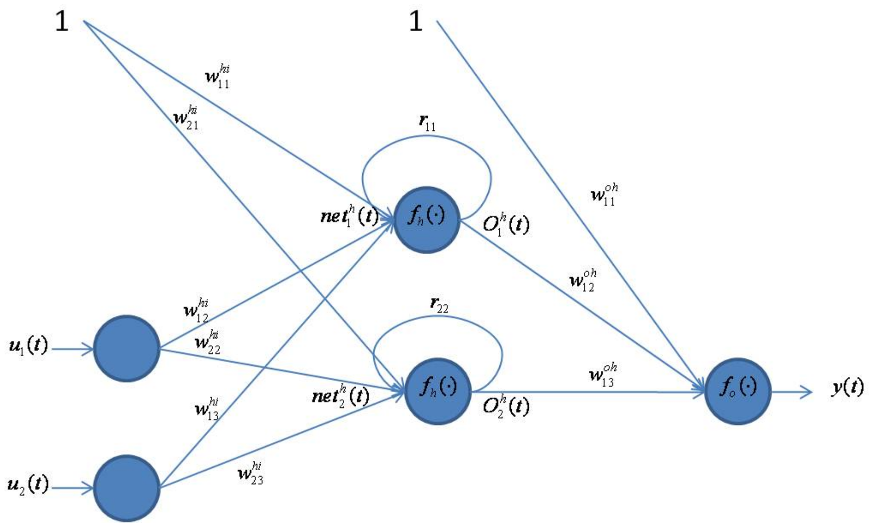


Figure 5.1: Neural Network Architecture.

den layers W^{hi}, recurrent (self-feedback) weights in hidden layer R, and weights between hidden and output layers W^{oh}.

$$W^{hi} = \begin{pmatrix} w^{hi}_{11} & w^{hi}_{12} & w^{hi}_{13} \\ w^{hi}_{21} & w^{hi}_{22} & w^{hi}_{23} \end{pmatrix}, \qquad R = \begin{pmatrix} r^{h}_{11} & 0 \\ 0 & r^{h}_{22} \end{pmatrix},$$
$$W^{oh} = \begin{pmatrix} w^{oh}_{11} & w^{oh}_{12} & w^{oh}_{13} \end{pmatrix}.$$

For all groups, the index is composed of two terms. The first term referes to the destinatin neuron and the second referes to the source neuron. For example, w^{hi}_{ij} is the weight of the connection from the jth input unit to the ith hidden unit. To model the DRN, we introduce the following notations:
$u_i(t)$: The ith external input at time t.
$net^h_i(t)$: Net input to the ith hidden unit at time t.
$net^o(t)$: Net input to the output neuron at time t.
$O^h_i(t)$: Output of the ith hidden neuron at time t.
$y(t)$: Output of the output neuron at time t.
f_h: Hidden layer activation function.
f_o: Output layer activation function.
η: is the network learning rate.
Now, we can write the output of the network as

$$y(t) = f_o(net^o(t)). \tag{5.1}$$

The net input at the output neuron is given by

$$net^o(t) = w^{oh}_{11} + w^{oh}_{12} O^h_1(t) + w^{oh}_{13} O^h_2(t). \tag{5.2}$$

In matrix format, equation 5.2 can be written as

$$net^o(t) = W^{oh} O^h_b(t), \tag{5.3}$$
$$O^h_b(t) = \begin{pmatrix} 1 \\ O^h_1(t) \\ O^h_2(t) \end{pmatrix}. \tag{5.4}$$

The output of the i^{th} hidden unit is given by

$$O_i^h(t) = f_h(net_i^h(t)), i \in \{1,2\}. \tag{5.5}$$

The net input at the i^{th} hidden neuron is given by

$$net_i^h(t) = w_{i1}^{hi} + w_{i2}^{hi}u_1(t) + w_{i3}^{hi}u_2(t) + r_{ii}^h O_i^h(t-1), i \in \{1,2\}. \tag{5.6}$$

In matrix format, equation 5.6 can be written as

$$net^h(t) = W^{hi}X(t) + RO^h(t-1), \tag{5.7}$$

$$X(t) = \begin{pmatrix} 1 \\ u_1(t) \\ u_2(t) \end{pmatrix}, \qquad O^h(t-1) = \begin{pmatrix} O_1^h(t-1) \\ O_2^h(t-1) \end{pmatrix}. \tag{5.8}$$

Let D(t) denote the desired output value. Then, define the error vector $e(t)$ as

$$e(t) = D(t) - y(t). \tag{5.9}$$

Let the overall network error at time t be

$$J(t) = \frac{1}{2}e^T(t)e(t). \tag{5.10}$$

Assume that the network run starts at time t_o up to some final time t_f, then the total error is given by

$$J_{total} = \sum_{t=t_o}^{t_f} J(t). \tag{5.11}$$

The objective is to adjust the network weights such that the total error J_{total} is minimized. The network weights are adjusted according to the gradient descent method as

$$\Delta w_{ij} = -\eta \frac{\partial J_{total}}{\partial w_{ij}}. \tag{5.12}$$

Since the total error is just the sum of the instantaneous errors, then

$$\Delta w_{ij} = \sum_{t_o}^{t_f} \Delta w_{ij}(t). \tag{5.13}$$

and $\Delta w_{ij}(t)$ is given by

$$\Delta w_{ij}(t) = -\eta \frac{\partial J(t)}{\partial w_{ij}}, \tag{5.14}$$

$$\Delta w_{ij}(t) = \eta(D(t) - y(t)) \frac{\partial y(t)}{\partial w_{ij}}. \tag{5.15}$$

Now, we differentiate $y(t)$ with respect to w_{ij}^{hi}, r_{ii}, and w_{ij}^{oh} to get

$$\frac{\partial y}{\partial w_{11}^{hi}} = f_o'(net^o(t)) w_{12}^{oh} \frac{\partial O_1^h(t)}{\partial w_{11}^{hi}}, \tag{5.16}$$

$$\frac{\partial y}{\partial w_{12}^{hi}} = f_o'(net^o(t)) w_{12}^{oh} \frac{\partial O_1^h(t)}{\partial w_{12}^{hi}}, \tag{5.17}$$

$$\frac{\partial y}{\partial w_{13}^{hi}} = f_o'(net^o(t)) w_{12}^{oh} \frac{\partial O_1^h(t)}{\partial w_{13}^{hi}}, \tag{5.18}$$

$$\frac{\partial y}{\partial w_{21}^{hi}} = f_o'(net^o(t)) w_{13}^{oh} \frac{\partial O_2^h(t)}{\partial w_{21}^{hi}}, \tag{5.19}$$

$$\frac{\partial y}{\partial w_{22}^{hi}} = f_o'(net^o(t)) w_{13}^{oh} \frac{\partial O_2^h(t)}{\partial w_{22}^{hi}}, \tag{5.20}$$

$$\frac{\partial y}{\partial w_{23}^{hi}} = f_o'(net^o(t)) w_{13}^{oh} \frac{\partial O_2^h(t)}{\partial w_{23}^{hi}}, \tag{5.21}$$

$$\frac{\partial y}{\partial r_{11}} = f_o'(net^o(t)) w_{12}^{oh} \frac{\partial O_1^h(t)}{\partial r_{11}}, \tag{5.22}$$

$$\frac{\partial y}{\partial r_{22}} = f_o'(net^o(t)) w_{13}^{oh} \frac{\partial O_2^h(t)}{\partial r_{22}}. \tag{5.23}$$

This can be written in a compact way as

$$\frac{\partial y}{\partial w_{ij}^{hi}} = f_o'(net^o(t)) w_{1,i+1}^{oh} \frac{\partial O_i}{\partial w_{ij}^{hi}}, \tag{5.24}$$

$$\frac{\partial y}{\partial r_{ii}} = f_o'(net^o(t)) w_{1,i+1}^{oh} \frac{\partial O_i}{\partial r_{ii}}. \tag{5.25}$$

To completely define the gradient of the output with respect to the weights, the derivatives of the output of the hidden layer with respect to the weights are found as

$$\frac{\partial O_1^h(t)}{\partial w_{11}^h} = f_h'(net_1^h(t))(1 + r_{11}\frac{\partial O_1^h(t-1)}{\partial w_{11}^h}), \tag{5.26}$$

$$\frac{\partial O_1^h(t)}{\partial w_{12}^h} = f_h'(net_1^h(t))(u_1(t) + r_{11}\frac{\partial O_1^h(t-1)}{\partial w_{12}^h}), \tag{5.27}$$

$$\frac{\partial O_1^h(t)}{\partial w_{13}^h} = f_h'(net_1^h(t))(u_2(t) + r_{11}\frac{\partial O_1^h(t-1)}{\partial w_{13}^h}), \tag{5.28}$$

$$\frac{\partial O_2^h(t)}{\partial w_{21}^h} = f_h'(net_2^h(t))(1 + r_{22}\frac{\partial O_2^h(t-1)}{\partial w_{21}^h}), \tag{5.29}$$

$$\frac{\partial O_2^h(t)}{\partial w_{22}^h} = f_h'(net_2^h(t))(u_1(t) + r_{22}\frac{\partial O_2^h(t-1)}{\partial w_{22}^h}), \tag{5.30}$$

$$\frac{\partial O_2^h(t)}{\partial w_{23}^h} = f_h'(net_2^h(t))(u_2(t) + r_{22}\frac{\partial O_2^h(t-1)}{\partial w_{23}^h}), \tag{5.31}$$

$$\frac{\partial O_1^h(t)}{\partial r_{11}} = f_h'(net_1^h(t))(O_1^h(t-1) + r_{11}\frac{\partial O_1^h(t-1)}{\partial r_{11}}), \tag{5.32}$$

$$\frac{\partial O_2^h(t)}{\partial r_{22}} = f_h'(net_2^h(t))(O_2^h(t-1) + r_{22}\frac{O_2^h(t-1)}{r_{22}}). \tag{5.33}$$

This can be written in a more compact way as

$$\frac{\partial O_i(t)}{\partial w_{ij}^{hi}} = f_h'(net_i^h(t))((X_j(t))_j + r_{ii}\frac{\partial O_i^h(t-1)}{\partial w_{ij}^{hi}}), \tag{5.34}$$

and

$$\frac{\partial O_i(t)}{\partial r_{ii}} = f_h'(net_i^h(t))(O_j^h(t-1) + r_{ii}\frac{\partial O_i^h(t-1)}{\partial r_{ii}}). \tag{5.35}$$

The gradient of the output with respect to the output layer weights is

$$\frac{\partial y(t)}{\partial w_{11}^{oh}} = 1, \tag{5.36}$$

$$\frac{\partial y(t)}{\partial w_{12}^{oh}} = O_1^h(t), \tag{5.37}$$

$$\frac{\partial y(t)}{\partial w_{13}^{oh}} = O_2^h(t). \tag{5.38}$$

This can be written in a more compact form as

$$\nabla_{W_{oh}} y(t) = O_b^h(t). \tag{5.39}$$

After deriving all required equations, the update rules for the network weights are

$$\Delta w_{ij}^{hi}(t) = \eta e(t)\frac{\partial y(t)}{\partial w_{ij}^{hi}}, \tag{5.40}$$

$$\Delta r_{ij}(t) = \eta e(t)\frac{\partial y(t)}{\partial r_{ij}}, \tag{5.41}$$

$$\Delta w_{ij}^{oh}(t) = \eta e(t)\frac{\partial y(t)}{\partial w_{ij}^{oh}}. \tag{5.42}$$

To gurantee convergence, and for faster learning adaptive learning rate η is used. To prove stability, we need to find a lyapunov function $V(t)$ where $V(t) > 0$and$\Delta V(t) < 0$ Let $V(t)$ be given by

$$V(t) = \frac{1}{2}e^2(t). \tag{5.43}$$

The changes of 5.43 due to training is given by

$$\begin{aligned}\Delta V(t) &= V(t+1) - V(t) && (5.44)\\ &= \frac{1}{2}\big(e^2(t+1) - e^2(t)\big) && (5.45)\\ &= \frac{1}{2}\Big(\big(e(t)+\Delta e(t)\big)^2 - e^2(t)\Big) && (5.46)\\ &= \Delta e(t)\big(e(t) + \frac{1}{2}\Delta e(t)\big). && (5.47)\end{aligned}$$

The changes of the error due to training is given by

$$\Delta e(t) = \Big(\frac{\partial e(t)}{\partial W}\Big)^T \Delta W, \tag{5.48}$$

where W is an arbitrary weight vector in R^n. From the update equations 5.40, 5.41, 5.42, ΔW is given by

$$\Delta W = \eta e(t)\frac{\partial y(t)}{\partial W}. \tag{5.49}$$

Subsituting 5.48, 5.49 and $\frac{\partial e(t)}{\partial W} = -\frac{\partial y(t)}{\partial W}$ in equation 6.15:

$$\begin{aligned}\Delta V &= \Big(\frac{\partial e(t)}{\partial W}\Big)^T \eta e(t)\frac{\partial y(t)}{\partial W}\big(e(t) + \frac{1}{2}\Big(\frac{\partial e(t)}{\partial W}\Big)^T \eta e(t)\frac{\partial y(t)}{\partial W}\big) && (5.50)\\ &= -\eta e^2(t)\frac{\partial y(t)}{\partial W}^2 + \frac{1}{2}\eta^2 e(t)^2\frac{\partial y(t)}{\partial W}^4. && (5.51)\end{aligned}$$

Let $g(t) = \frac{\partial y(t)}{\partial W}$, then

$$\Delta V = -e^2(\eta g(t)^2 - \frac{1}{2}\eta^2 g(t)^4) \tag{5.52}$$

Hence, the convergence is guranteed for

$$0 < \eta < \frac{2}{g^2(t)}. \tag{5.53}$$

6. Neural Network based Mobile Robot Navigation Algorithm

Controller Architecture

Training phase 1: Coarse tuning. A neural network is proposed to work in collaboration with the navigational algorithm developed earlier to optimize the navigational capabilities of the overall system. While the navigational algorithm avoids obstacles, it does not contain any constraints on the length of the path. The neural network is incorporated into the system to minimize the length of the path taken. The neural network selects the optimum trajectory based on obstacle information, target configuration, and the radius of turn proposed by the navigational algorithm. The neural network outputs the most promising radius of turn of the robot trajectory. For the neural network to achieve this objective, it needs to overcome few challenges. One of those challenges is that the 'correct' radius of turn is not available. Hence, the training cannot be conducted in a supervised manner where a dataset is available that helps the network form a map between the input data and the desired value. To overcome this obstacle, a hybird training scheme is proposed. First, the neural network will receive supervised training based on a sub-optimal dataset. Second, the neural network will be trained to adjust its weights to produce an optimum value based on an evaluation function. The training phases are further explained in the following paragraphs.

Training phase 1: Coarse tuning. In this phase the network is trained to map the input values to a desired value. The data set is generated using the navigational algorithm. The training is based on backpropogation and is done offline. The purpose of this training phase is to provide an adequate initial set of weights for the next phase as opposed of starting phase 2 from random variables. The data set is generated by having the robot maneuver a number of obstacle scenarios as shown in Figure 6.1. The different scenarios are as follows:

- Scenario 1: Moving in a gap between two obstacles. The gap width is slightly wider than the robot.
- Scenario 2: Avoiding a narrow gap and contouring an obstacle on the right.
- Scenario 3: Contouring an obstacle on the left.
- Scenario 4: Starting in a dead end situation

The testing data set consists of 1 scenario where the robot contours an obstacle on the right as depicted in Figure 6.1e. The robot trajectories are produced by the navigational algorithm described in [20]. The training data set consists of 295 different instances while the testing data set consists of 58 instances. The off-line training scheme is shown in Figure 6.3. The sum squared error (SSE) of the trained network is shown in Figure 6.2a. The network output versus the desired output for the training and testing data sets are shown in Figure 6.2b and Figure 6.2c.

Training phase 2: Fine tuning. The optimum turning radius is unknown. However, the radius taken by a system can be evaluated. In this phase of training, the network recieves feedback about its performance based on an evaluation function. The evaluation function is described as:

$$J = \frac{1}{2}e^2 = \frac{1}{2}(e_x^2 + e_y^2) \tag{6.1}$$

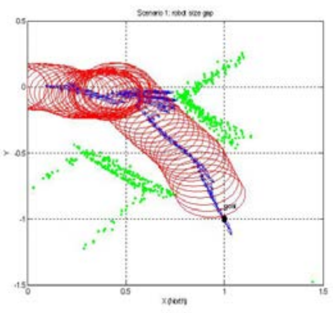
where:

$$e_x = x_t - x \tag{6.2}$$

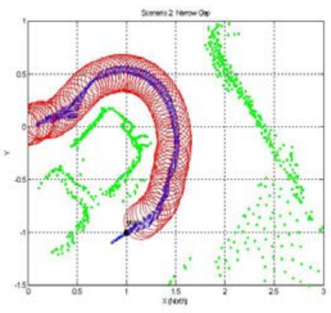
$$e_y = y_t - y \tag{6.3}$$

The derivative of the evaluation function with respect to the weights is given by:

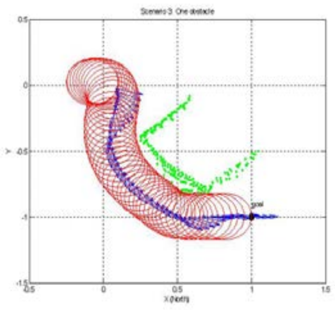
$$\frac{\partial J}{\partial W} = -(e_x J_x + e_y J_y)\frac{\partial r}{\partial W} \tag{6.4}$$



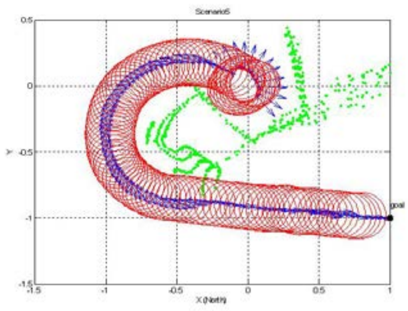
(a)



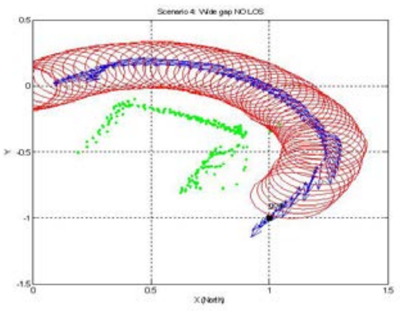
(b)



(c)



(d)



(e)

Figure 6.1: Training and testing scenarios. (a) Training scenario 1. (b) Training scenario 2. (c) Training scenario 3. (d) Training scenario 4. (e) Testing scenario.

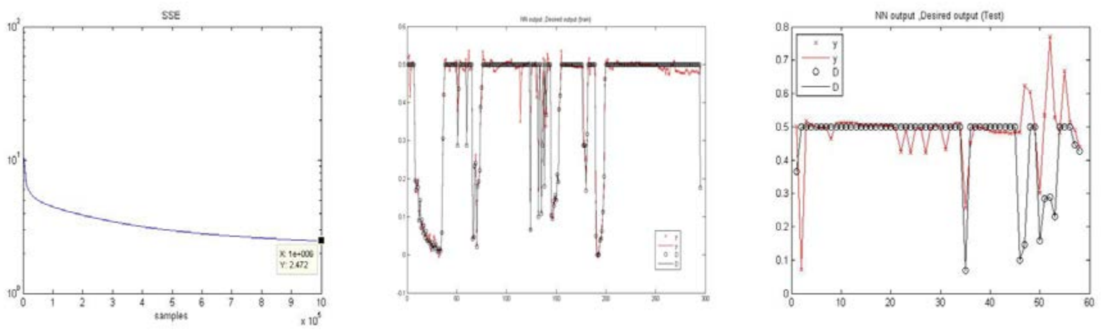
where J_x and J_y are the sensitivities of the system and are given by:

$$J_x = \frac{\partial x}{\partial r} = sign\left(\frac{x(t) - x(t-1)}{r(t) - r(t-1)}\right) \tag{6.5}$$

$$J_y = \frac{\partial y}{\partial r} = sign\left(\frac{y(t) - y(t-1)}{r(t) - r(t-1)}\right) \tag{6.6}$$

$\frac{\partial r}{\partial W}$ is estimated online using RTRL:

$$\frac{\partial r}{\partial W} = \frac{\partial O}{\partial W} \tag{6.7}$$



(a) (b) (c)

Figure 6.2: Training and testing scenarios. (a) The sum squared error versus the number of training epochs. (b) Network output versus the desired network output using the training data set. (c) Network output versus the desired network output using the testing data set.

The weight is updated according to:

$$\Delta W = -\eta \frac{\partial J}{\partial W} \tag{6.8}$$

Subsituting eq.6.1 into 6.8 gives:

$$\Delta W = -\eta e \frac{\partial e}{\partial W} \tag{6.9}$$

The training phases are shown in Figure 6.3 and 6.4.

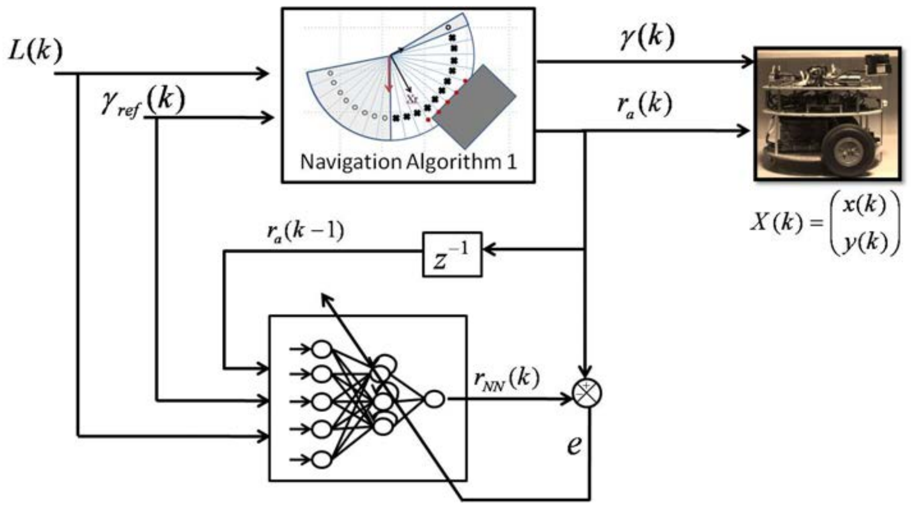


Figure 6.3: Phase 1 Training.

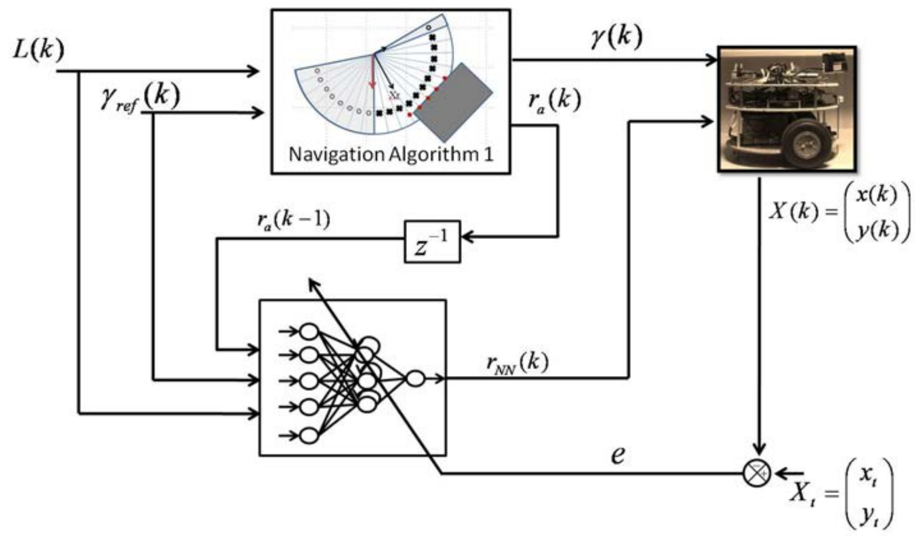


Figure 6.4: Phase 2 Training

System Stability Proof

We use lyapunov theorem to prove system stability: Define the lyapunov function as:

$$V(k) = \frac{1}{2}e^2(k) \geq 0 \tag{6.10}$$

To prove system stability, we need to show that $\Delta V \leq 0$.Define ΔV as:

$$\Delta V = V(k+1) - V(k) \tag{6.11}$$

Subsitute eq.6.10 in eq.6.11

$$\Delta V = \frac{1}{2}\Big(e^2(k+1) - e^2(k)\Big) \tag{6.12}$$

$e(k+1)$ can be written as:

$$e(k+1) = e(k) + \Delta e(k) \tag{6.13}$$

Subsitute eq.6.13 in eq.6.12:

$$\Delta V = e(k)\Delta e(k) + \frac{1}{2}\Delta^2 e(k) \tag{6.14}$$

$$= \Delta e(k)\Big(e(k) + \frac{1}{2}\Delta e(k)\Big) \tag{6.15}$$

$\Delta e(k)$ is given by:

$$\Delta e(k) = \left[\frac{\partial e(k)}{\partial W}\right]^T \Delta W \tag{6.16}$$

Subsituting eq.6.9 in 6.16 gives:

$$\Delta e(k) = -\eta e \| \frac{\partial e}{\partial W} \|^2 \tag{6.17}$$

In order to find $\frac{\partial e(k)}{\partial W}$,the chain rule is used:

$$\frac{\partial e}{\partial W} = \frac{\partial r}{\partial W}\left[\frac{\partial e}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial e}{\partial y}\frac{\partial y}{\partial r}\right] \tag{6.18}$$

where:

$$\frac{\partial e}{\partial x} = -\frac{e_1}{e} \tag{6.19}$$

$$\frac{\partial e}{\partial y} = -\frac{e_2}{e} \tag{6.20}$$

Subsituting equations (6.5, 6.6,6.19,6.20) in eq.6.18 gives:

$$\frac{\partial e}{\partial W} = -\frac{\partial r}{\partial W}\left(\frac{e_1 J_x + e_2 J_y}{e}\right) \tag{6.21}$$

Subsituting 6.21 in 6.16, gives:

$$\Delta e(k) = -\frac{\eta}{e} \| \frac{\partial r}{\partial W} \|^2 (e_x J_x + e_y J_y)^2 \tag{6.22}$$

Subsituting 6.22 in 6.15:

$$\Delta V = \eta \| \frac{\partial r}{\partial W} \|^2 (e_x J_x + e_y J_y)^2 \Big(-1 + \frac{\eta}{2e^2} \| \frac{\partial r}{\partial W} \|^2 (e_x J_x + e_y J_y)^2\Big) \tag{6.23}$$

the term $\eta \| \frac{\partial r}{\partial W} \|^2 (e_x J_x + e_y J_y)^2 \geq 0$, hence inorder to have $\Delta V \leq$0, η should be chosen as:

$$\eta \leq \frac{2e^2}{\| \frac{\partial r}{\partial W} \|^2 (e_x J_x + e_y J_y)^2} \tag{6.24}$$

Design DRN

Step 1 and 2. The dataset is divided into training, and testing: The training dataset are used to adjust the network parameters to match the desired output in the training set. Once the network is trained, the testing dataset is used to validate the trained network. It is important for the training dataset to be representative because this will effect the accuracy and generalization capabilities of the network: The network cannot be more accurate than the data used to train it and multilayer networks can generalize only within the training input range. They do not have the ability to accurately extrapolate beyond the training input range. Before training the inputs are scaled into the range $[-1,1]$ using:

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}(y_{max} - y_{min}) + y_{min}; \tag{6.25}$$

The maximum and minimum values are estimated from the trainig dataset, hence, testing data presented to the network are scaled using the estimated min/max values of the trainig dataset.

Step 3 and 4: Create and configure the network. The structure of the network is illustrated in. The implemented network has 22 input units, 5 hidden units, and 1 output unit. The activation functions for the hidden and output layers are tanh, and linear respectively. The network is trained using RTRL. For the network to be able to generalize and in other words not suffer from overfitting, regulaization has to be performed. Normally, the number of hidden units are varied manually for the best performance in training and validation datasets.

Step 5 and 6: Initialize the weights and biases and train the network.

Step 7: Validate the network. The DRN function is described below:

```
1: Inputs: $W^{hi}$, $R$, $W^{oh}$, $\eta$, $x_{old}$, $y_{old}$, $r_{old}$ $O^h_{old}$, $\frac{\partial O_{old}}{\partial W^{hi}}$, $\frac{\partial O_{old}}{\partial R}$, $x_{target}$, and $y_{target}$.
2: $in \leftarrow 22$
3: $hi \leftarrow 5$
4: $ou \leftarrow 1$
5: $f_h \leftarrow$ ‘tanh’
```

```
 6: $f_o \leftarrow$ ‘linear’
 7: $u_{min} \leftarrow [.01 * ones(1,20), -180, 0]$
 8: $u_{max} \leftarrow [3 * ones(1,20), 180, .05]$
 9: $D_{min} \leftarrow 0$
10: $D_{max} \leftarrow 0.5$
11: $u_n = 2(x - u_{min})./(u_{max} - u_{min}) - 1$ { Normalize inputs to $[1, -1]$}
12: Compute neural networks forward dynamics:
        $$O^h = f_h\Big(W^{hi}\big(1\ u_n\big)^T + RO^h_{old}\Big), \qquad r_n = W^{oh}\big(1\ O^h\big)^T.$$
13: Compute error $J$ and system sensitivities $J_x$, $J_y$:
        $$J = \frac{1}{2}(e_x^2 + e_y^2)$$
        $$J_x = sign(\frac{x - x_{old}}{r_n - r_{old}}) \qquad J_y = sign(\frac{y - y_{old}}{r_n - r_{old}}).$$
14: for $i = 1$ to $hi$ do
15:     for $j = 1$ to $in + 1$ do
16:         $\frac{\partial O}{\partial w^{hi}_{ij}} = f'_h(net^h_i)(X_j + r_{ii}\frac{\partial O_{old}}{\partial w^{hi}_{ij}})$
17:     end for
18: end for
19: for $i = 1$ to $hi$ do
        $\frac{\partial O}{\partial r_{ii}} = f'_h(net^h_i)(O^h_{old}(i) + r_{ii}\frac{\partial O_{old}}{\partial r_{ii}})$
20: end for
21: $\frac{\partial r}{\partial W_{oh}} = (O^h_b)^T$;
22: for $i = 1$ to $hi$ do
23:     for $j = 1$ to $in + 1$ do
            $\frac{\partial r}{\partial w^{hi}_{ij}} = f'_o(net^o)w^{oh}_{1,i+1}\frac{\partial O_i}{\partial w^{hi}_{ij}}$
24:     end for
25: end for
26: for $i = 1$ to $hi$ do
```

```
        $\frac{\partial r}{\partial r_{ii}} = f_o'(net^o) w^{oh}_{1,i+1} \frac{\partial O_i}{\partial r_{ii}}$
27: end for
28: Update Weights:
        $\Delta W^{hi} = \eta(e_x J_x + e_y J_y)\frac{\partial r}{\partial W^{hi}}$        $\Delta R = \eta(e_x J_x + e_y J_y)\frac{\partial r}{\partial R}$
        $\Delta W^{oh} = \eta(e_x J_x + e_y J_y)\frac{\partial r}{\partial W^{oh}}$        $W^{hi} = W^{hi} + \Delta W^{hi}$
        $R = R + \Delta R$        $W^{oh} = W^{oh} + \Delta W^{oh}$
29: $g = \left[\frac{\partial r}{\partial W^{hi}}, \frac{\partial r}{\partial R}, \frac{\partial r}{\partial W^{oh}}\right]$
30: $g_{norm} = g^T g$
31: if $g_{norm} > g_{max}$ then
32:         $g_{max} = g_{norm}$
33: end if
34: $\eta = \frac{1}{g_{max}}$
35: Update old values:
        $x_{old} = x$        $y_{old} = y$
        $r_{old} = r_n$        $O^h_{old} = O^h$
        $\frac{\partial O_{old}}{\partial W^{hi}} = \frac{\partial O}{\partial W^{hi}}$        $\frac{\partial O_{old}}{\partial R} = \frac{\partial O}{\partial R}$
36: $r = \frac{D_{max}-D_{min}}{2}(r_n + 1) + D_{min}$ {Scale output to $[D_{min}, D_{max}]$}
37: return r, $W^{hi}$, $R$, $W^{oh}$, $\eta$, $x_{old}$, $y_{old}$, $r_{old}$ $O^h_{old}$, $\frac{\partial O_{old}}{\partial W^{hi}}$, $\frac{\partial O_{old}}{\partial R}$.
```

DRN Obstacle Avoidance Pseudocode

```
1: Set initial conditions
    $\eta = 0.001$
    $c_1 = 0.7$ , $c_2 = 0.3$
    $R_{safe} = 0.5$
    $x_{goal} = 1, y_{goal} = -1$
```

```
2:  load initial weights for $W_{hi}$, $W_{oh}$, $R_h$
3:  Set serial communication parameters, e.g. baudrate
4:  Set direction and initial value of DIO pins
5:  open text file to record robot status {At this point the supervisor should connect the
    laser cable to COM2}
6:  while start button not pressed do
              Do nothing
7:  end while
8:  Read robot heading from compass sensor (theta)
9:  Request measurments collected by microcontroller (NR , NL , omegaR , OmegaL ,
    dis).
10: Read obstacle distribution from laser sensor (Obst)
11: Compute robot linear and angular velocities according to:
    v = 0.5*r*(omegaR+omegaL)
    xdot = v*cos(theta)
    ydot = v*sin(theta)
12: Find x, y position of the midpoint of wheel through trapezoidal integration
    x = x_old +Ts/2*(xdot+xdot_old)
    x_old = x
    xdot_old = xdot
    y = y_old + Ts/2*(ydot+ydot_old)
    y_old =y
    ydot_old = ydot
13: Find location of reference point at the laser location
    x_laser = x + 0.1*cos(theta)
    y_laser = y + 0.1*sin(theta)
14: Compute error between refrence point and the goal configuration
    dy = ygoal-y_laser
    dx = xgoal-x_laser
    alpha = atan2(dy,dx)
    angle = diffangles(theta, alpha). Note: $-180^\circ \leq angle \leq 180^\circ$
    angle_no_obst = angle
```

```
15: Limit the obstacle scan angle to $[-100, 100[$ and save in (laser_pt)
16: Enlarge obstacle in propotion to the robot radius R and the safety buffer Rsafe.
17: Find the distance between laser reference point and the enlarged obstacle (dlarge)
18: Classify cspace sectors to free or occupied
19: **if** angle_no_obst is occupied **then**
20:         Form gaps by grouping neighbouring sectors
21:         Classify gaps to wide, medium , and narrow
22:         Select best gap from wide angle, medium, narrow, turn $180^\circ$ on spot
23:         Select turning radius: Finf min obdt point that robot encountert in path
            from initial pose to final pose
24:         Compute dmin with respect to reference point at mid of wheel axes
25:         Include safety distance: d_min_m = d_min_m -1.2R
26:         Saturate block for dmin
27:         **if** $error \leq 0.3$ **then**
28:                 **if** $dmin \geq 0.1$ **then**
                        dmin_m = 0.1
29:                 **end if**
30:         **end if**
31: **end if**
32: Calculate rc
    rc = fabs(dmin_m/2/sin(angle))
33: **if** $error \leq 0.3$ **then**
34:         rc=.3
35: **end if**
36: **if** $rc \geq 0.5$ **then**
37:         rc = 0.5
38: **end if**
39: Calculate radius using DRN (rn)
40: **if** $rn \leq 0$ **then**
41:         rn=0
42: **end if**
43: **if** $error \leq 0.3$ **then**
```

```
44:         if $rn \geq 0.3$ then
45:                 rn = 0.3
46:         end if
47: end if
48: if $rn \geq 0.5$ then
49:         rn =0.5
50: end if
51: Convert rc to duty cycle
    duty_cycle = floor(15*(rnL-1)/(rnL+1))
52: if angle=500 or reverse=1 then
53:         duty_cycle = 0x1F
54:         move(rotate_left, duty_cycle)
55: else if $-2 \leq angle$ and $angle \leq 2$ then
56:         duty_cycle = 0x0F
57:         move(forward, duty_cycle);
58: else if angle≥ angle_thresh then
59:         move(left, duty_cycle);
60: else if angle≤ -angle_thresh then
61:         move(right, duty_cycle);
62: end if
63: Compute the error between current and goal location
    error = (xgoal-x_laser)*(xgoal-x_laser)+(ygoal-y_laser)*(ygoal-y_laser);
64: if $error \leq .3$ then
65:         R_safe = .2
66: else
67:         R_safe = .5;
68: end if
69: if oscillatory motion detected then
70:         c1=0, c2 = .7
71: else
72:         c1=.7, c2 = .3
73: end if
```

```
74: **if** error$\leq$ dis_thresh **then**
75:                  Terminate motion
76: **end if**
```

7. Robot Platform and Experimental Results

Mobile Robot Architecture

The mobile robot platform is designed to operate in an indoor environment with a solid flat surface. A differential steering system is employed to generate forward and steered motion. The platform provides a rich computing environment consisting of a single board computer and a microcontroller. It is also equipped with obstacle detection sensors such as laser range sensor and ultrasonic as well as localization sensors such as encoders and a compass. The platform has the flexibility of the addition of new sensors. The mobile robot platform is shown in Figure 7.1.

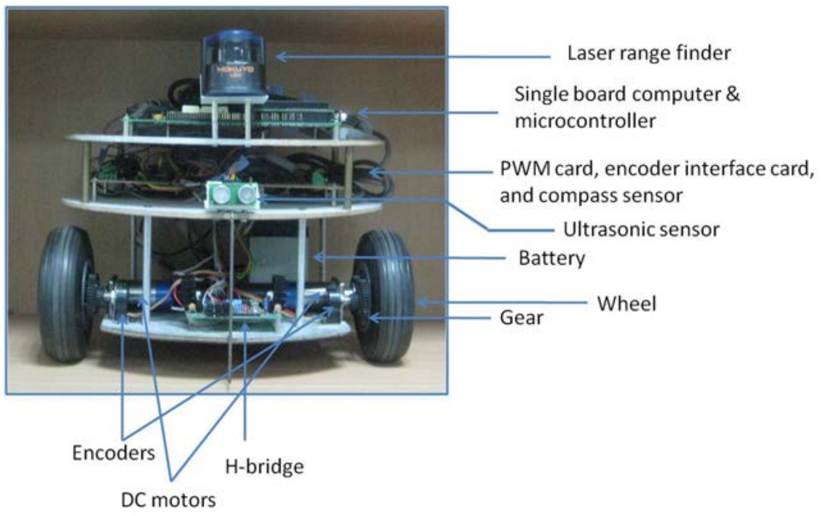

Laser range finder
Single board computer & microcontroller
PWM card, encoder interface card, and compass sensor
Ultrasonic sensor
Battery
Wheel
Gear
Encoders
H-bridge
DC motors

Figure 7.1: Mobile robot platform.

The general functions the robot perform to avoid obstacles are depicted in 7.2. The target location is given by an external source such as a human operator. The robot estimates its current location in the localization stage by processing the odometry measurements from the microcontroller. It also estimates the environment at the current robot configuration from the range measurements provided by the laser. The environment estimate together with the difference between the current and target locations are supplied to the obstacle avoidance algorithm which determines the desired wheel velocities to avoid the obstacles and reach the target configuration.

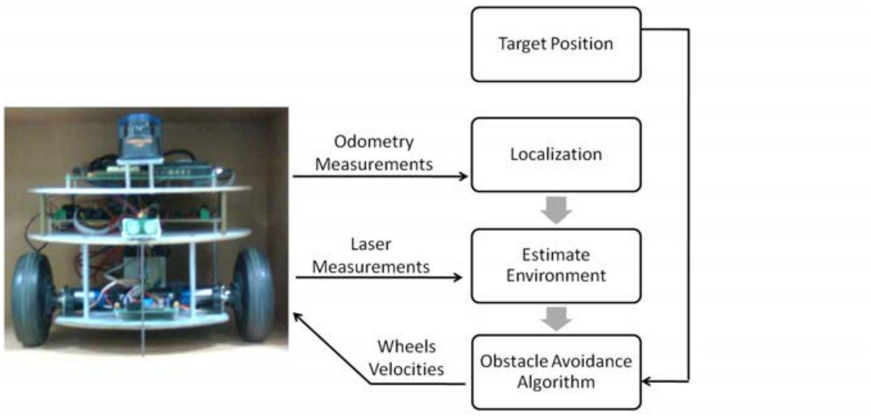


Figure 7.2: Mobile robot general functions.

Figure 7.3 illustrates how the different components communicate together. The work flow starts from the SBC where the desired robot velocity (duty cycle) is determined and transmitted through digital input/output (DIO) lines to the PWM card. In the PWM card, the actual PWM signal is generated and supplied to the motors driving system (H-bridges) to deliver the corresponding voltages to the dc motors. The motion of the robot is generated by the motors driving the rubber wheels. Encoders count the number of revolutions the wheels take in the form of 128 pulses for every revolution. These pulses are fed to the encoders interface circuit which performs quadrature counting to enhance the encoders resolution by a factor of 4. The encoders interface also measure the direction of motion (forward or backward). The microcontroller captures the clock of the encoders signals to determine the wheels position and velocities.Also, it triggers the sonar sensor to take new range measurement. The sonar sensor measures the relative distance between the robot and the nearest obstacle. Next, the microcontroller arranges the measurements (position,velocity, and range distance) into a packet and waits for a request from the SBC. Once the SBC requests data by setting the corresponding DIO high, the packet is transmitted through the serial port. The SBC asks the laser sensor to send new range measurements and the compass sensor to send new robot orientation. Serial communication is used to interface the SBC with the sensors. The SBC should process all data and update the motor velocity accordingly. The cycle continues till a desired robot configuration is reached. A delay is added in the main loop in order to have a 1s sampling time. The details of the mechanical and electrical designs are described in the following sections.

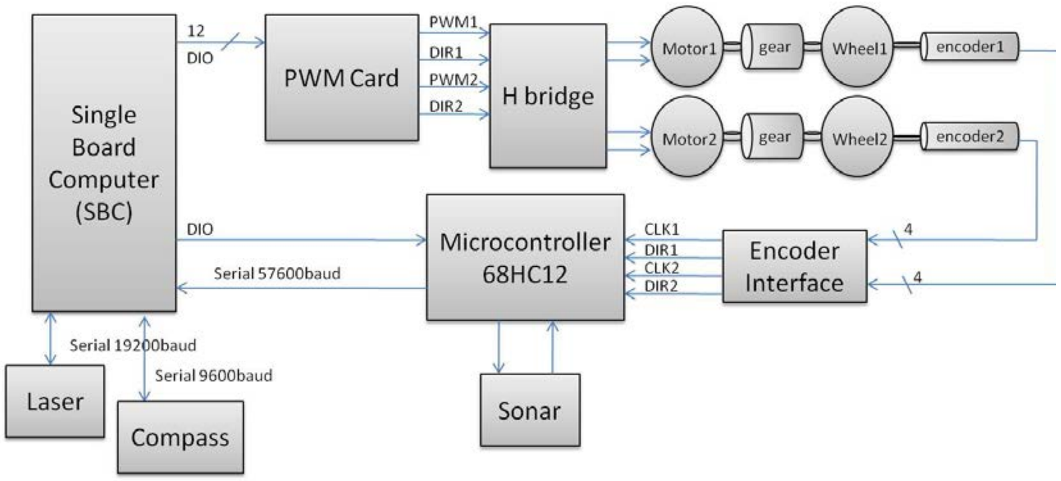


Figure 7.3: Hardware Block Diagram.

Mechanical design.

Robot chassis. The platform is circular in shape and is made up of three wooden layers. The bottom layer houses the wheels, motors, encoders, batteries, and electric drive system. The middle layer houses electronic cards and ultrasonic sensors while the upper layer houses the embedded computing boards and the laser range sensor.

Locomotion. The locomotion of the platform was chosen to maximize the maneuverability, and stability of the robot given the desired work space environment. The environment the platform is intended to work in is a laboratory environment which has a solid flat surface. An optimal and widely used choice for such requirements is differentially steering system. Three wheels are used: two motorized wheels and one caster wheel to guarantee stable balance. On the circular chassis of the robot, the two motorized wheels are placed at the front and the unpowered caster wheel is placed at the rear. The front wheels have a radius of $7cm$ and are made of rubber. The caster wheel is made of frictionless plastic material.

The maneuverability of this locomotion features a zero turn radius: motion in any direction may be achieved by an initial rotational motion without changing its ground footprint. The controllability aspect of this locomotion has the drawback of that the two motors attached to the two wheels must be driven along exactly the same velocity profile, which can be challenging considering variations between wheels, motors, and environmental differences.

Electrical design.

Power system. The robot is powered by one lead acid 12 V, 7 Ah battery and two Ni-Mh 7.2 V, 4200mAh batteries connected in series to provide 14.4 V. Figure 7.4 illustrates the power distribution of the system. The 14.4 V source provides power to the microcontroller and the laser sensor while the 12 V battery provides power to the motors, single board computer, onboard electronics and the remaining sensors.

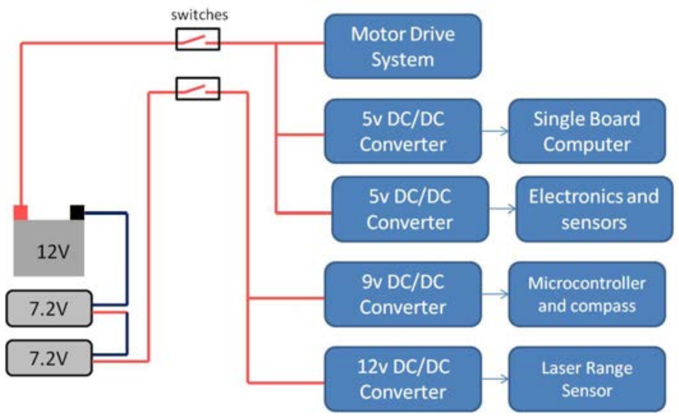


Figure 7.4: Power Supply System.

Drive system. The motors attached to the wheels are harmonic drive micro dc motors. Each motor is controlled by an H-bridge through a pulse width modulated (PWM) signal which sets the motor velocity and direction. The PWM signal has a resolution of four bits, resulting in sixteen different levels of speed. The PWM signal is generated using a PWM card that has 8 input channels to specify the PWM duty cycle for both motors. The duty cycle is specified using the digital I/O lines of the SBC.

Embedded computing design. As stated in the design consideration, the robot is designed to have a rich computing environment. This will allow the robot to house computationally expensive sensors as well as execute a complex path planning algorithm. The robot has two processors consisting of a single board computer and a microcontroller.

Single Board Computer. The brain of the robot is a PC compatible single board computer (SBC). The main advantage of SBC is that it has the computation and memory capability of a moderate PC running at a 5 Vdc. The SBC is equipped with an AMD Elan520 processor that is clocked at 133MHz. It is running linux as its operating system. The responsibilities assigned to the SBC are as follows:

- Set the velocity of the robot by specifying the duty cycle of the PWM signal that will be generated by an external PWM card.
- Receive data packets from the microcontroller that contains information about the wheels positions and velocities and the sonar measurement. The SBC will use these data to make intelligent decision about the robot navigation and obstacle avoidance problem.

The current functionality of the SBC is depicted in the Figure 7.5.

Microcontroller. The main task for the microcontroller is to interface with the various sensors that have precise timing requirements such as encoders and ultrasonic sensors. The microcontroller measures the position and speed of the left and right wheels. It also generates a 10us signal that pings (triggers) the ultrasonic sensor and measures its pong (echo) which is a pulse whose width varies in proportion to the measured distance. Figure7.6 shows the the connections between the sensors and the microcontroller. The microcontroller performs position measurements. The clock of the encoder is connected to an input capture pin. This pin generates an interrupt when a rising edge has occurred. The number of times the rising edge had occurred is saved in a variable memory location. These variables are incremented or decremented depending on the DIR pin of 7084.

Also, the microcontroller performs speed measurements. The angular speed of each wheel is given by $\omega = \frac{\Delta\varphi}{\Delta t}$. For an encoder with a resolution of p = 128 pulse/rev, the angle swept between 2 rising edges of the encoder clock is $\Delta\varphi = \frac{2\pi}{p} = 0.0213 rad$. Thus, ω can be computed if the time period between the two edges is measured. This is done using the timer module of the microcontroller. It should be noted that the number of times the

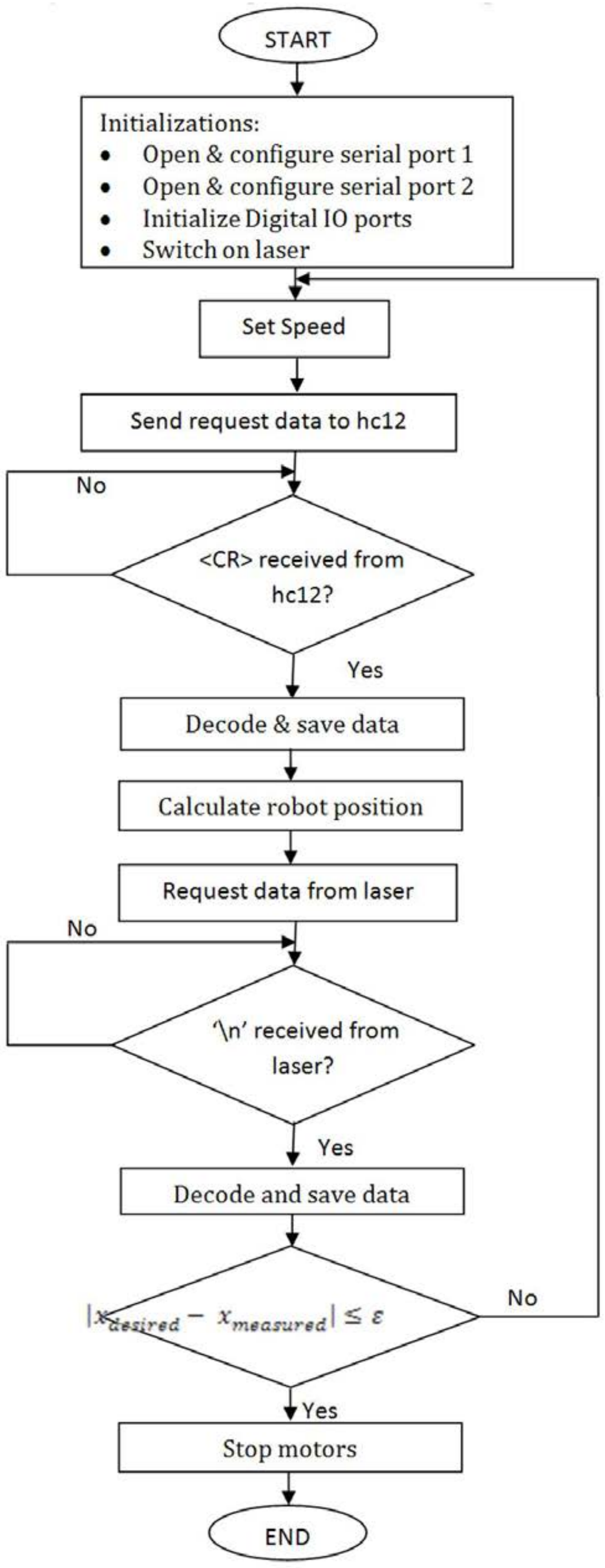

START
Initializations:
• Open & configure serial port 1
• Open & configure serial port 2
• Initialize Digital IO ports
• Switch on laser
Set Speed
Send request data to hc12
No
<CR> received from hc12?
Yes
Decode & save data
Calculate robot position
Request data from laser
No
'\n' received from laser?
Yes
Decode and save data
$|x_{desired} - x_{measured}| \leq \varepsilon$
No
Yes
Stop motors
END

Figure 7.5: The SBC flowchart.

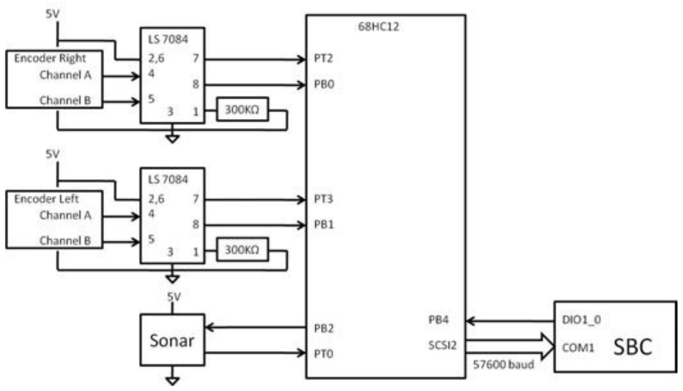
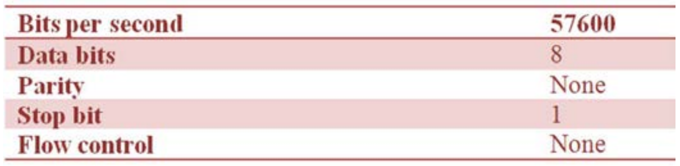


Figure 7.6: The connections between the microcontroller and the sensors.

microcontroller timer resets because of overflow should be tracked. This is essential for objects moving at slow velocities. The 68HC12 microcontroller bus clock is $24MHz$. The prescaler is set to 16 for reasons that will become apparent when discussing the sonar operation. Thus, the maximum period the timer exhibits before overflowing is $T_{max} = 16(\frac{2^{16}}{f}) = 43.7ms$.

The microcontroller obtain measurements from the sonar sensor. The 68HC12 send a trigger pulse of 10us width at the beginning of the main code. The time just before the sonar is triggered is recorded. The sonar produces an echo which is a pulse whose width is proportional to the measured distance from the obstacle. The 68hc12 is programmed to generate an interrupt when the falling edge of the echo signal is detected. To make sure that the microcontroller timer does not overflow before the falling edge of the echo event, a prescaler of 16 is chosen. With this prescaler, the maximum pulse width that can be measured is 43.7, which is larger that the maximum width that can be generated by the sonar (40ms).

The microcontroller uses a serial communication with the SBC. The serial communications settings are describe in Figure 7.7. The order of the data sent by 68HC12 is depicted



Bits per second	**57600**
Data bits	8
Parity	None
Stop bit	1
Flow control	None

Figure 7.7: Communication settings.

in Figure 7.8. All the data are decimal values, therefore, characters are used to segment between the different sensors values. The sensors readings are defined as float. The size of

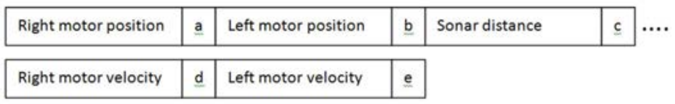


Figure 7.8: Data packet.

a float is 4 bytes. The size of a character is 1 byte. Thus, the size of the whole data packet is 5*(4bytes+1byte) = 25bytes. The time needed to transmit this packed is: $t = \frac{25}{115200} = 0.217ms$. The SBC requests the microcontroller to send data through serial by using digital output pins. When I/O pin, PB4, of microcontroller is set high, microcontroller transmits data. The functionality of the microcontroller is depicted in Figure 7.9.

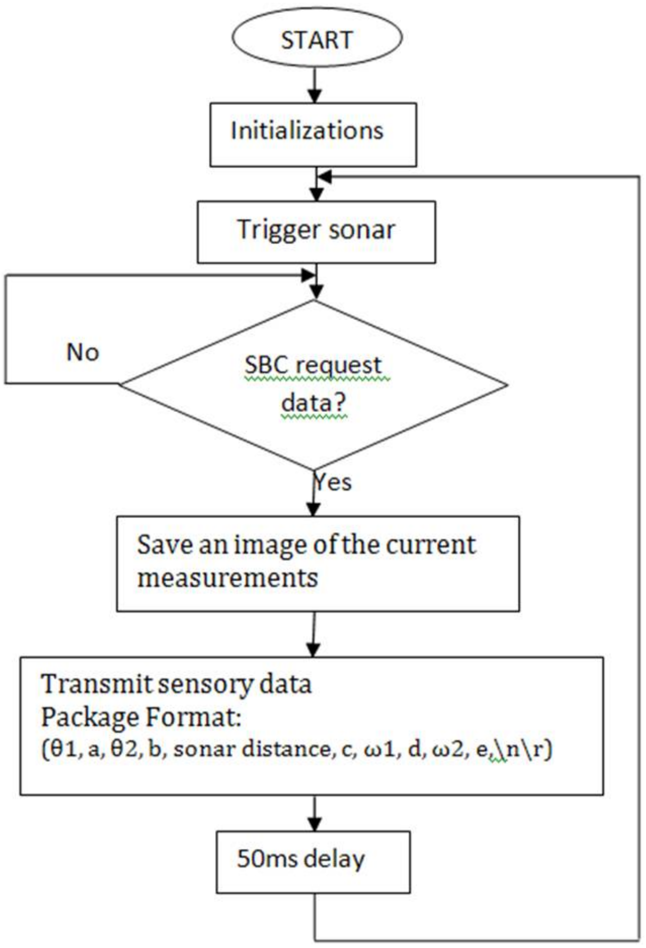


Figure 7.9: Microcontroller flow chart of the main program.

Laser range sensor. Acquiring information about the presence of an object and its location remotely without physical contact is a key measurement in a wide range of ap-

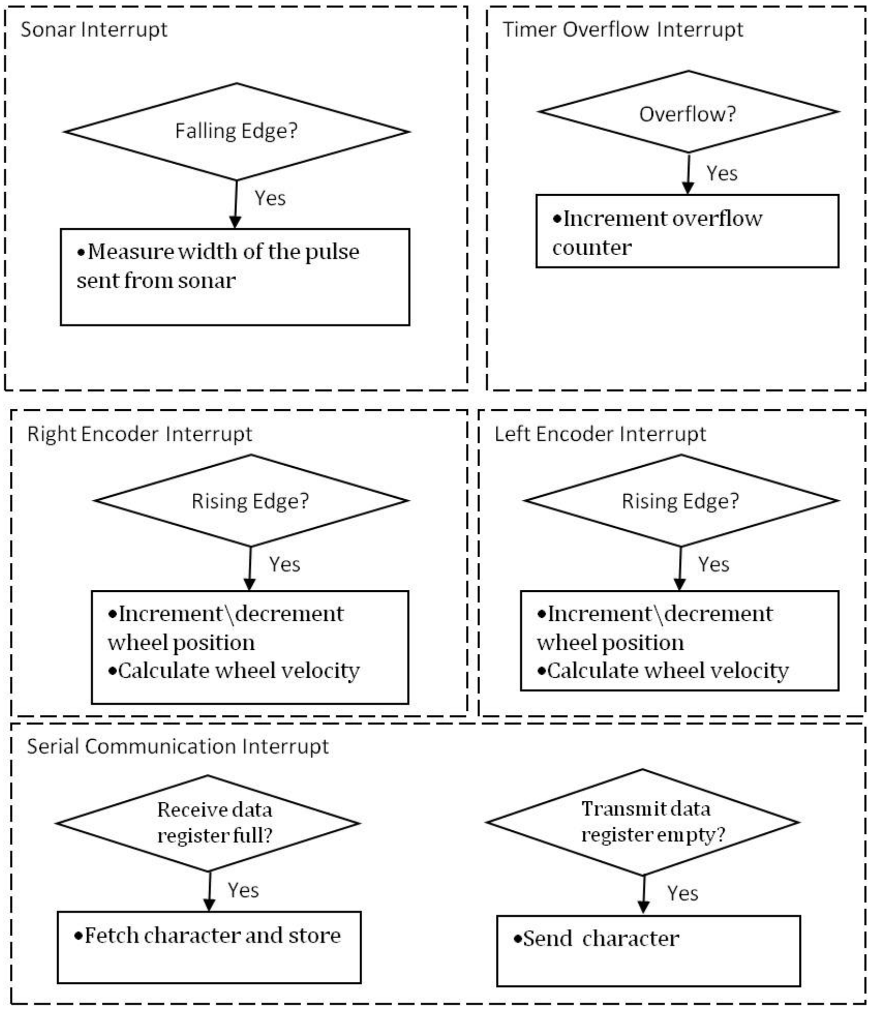

Sonar Interrupt
Falling Edge?
Yes
•Measure width of the pulse sent from sonar
Timer Overflow Interrupt
Overflow?
Yes
•Increment overflow counter
Right Encoder Interrupt
Rising Edge?
Yes
•Increment\decrement wheel position
•Calculate wheel velocity
Left Encoder Interrupt
Rising Edge?
Yes
•Increment\decrement wheel position
•Calculate wheel velocity
Serial Communication Interrupt
Receive data register full?
Yes
•Fetch character and store
Transmit data register empty?
Yes
•Send character

Figure 7.10: Microcontroller interrupts.

plications. Optical (laser) sensors, ultrasonic sensors, computer vision sensors are different sensors that are capable of acquiring such measurements. Optical is one of the most accurate and fast ranging sensors. This is owed to its optical nature. Light propagates extremely fast with a speed of 3×10^8 m/s allowing the sensor to generate new measurements at a high rate. Also the shortwave allows the detection of extremely small objects such as cloud particles. To conclude, optical range finders generate high resolution and high frequency data. Thus, its computation requirement falls between the minimal requirement of ultrasonic and the expensive requirement of computer vision.

The principle of operation of the optical range sensor is based on the phase shift measurement technique. It measures the distance by first emitting a continuous light beam. Once the beam hits an object, it gets reflected coaxially (isotropically) back to the sensor. The sensor measures the phase difference between the transmitted and received wave. The distance can be computed as:

$$d = \frac{\phi\lambda}{4\pi} = \frac{\phi c}{4\pi f} \tag{7.1}$$

Where:

d is the distance to a target,

ϕ is the measured phase shift,

c is the speed of light,

f is the modulation frequency,

and λ is the modulation wavelength.

The phase shift is measured by multiplying the transmitted signal with the received signal,

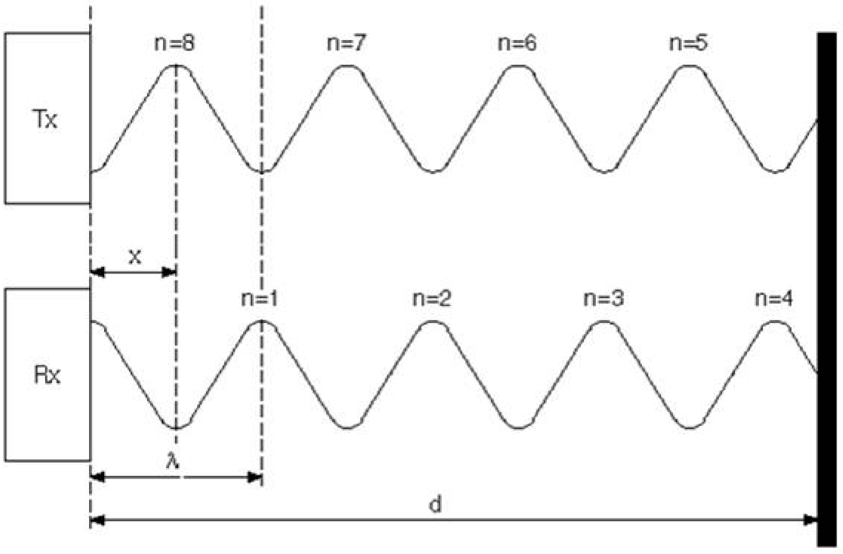


Figure 7.11: Relationship between outgoing and reflected waveforms, where x is the distance corresponding to the differential phase.[3]

then averaging their product over many cycles. This can be expressed mathematically as:

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \sin(\frac{2\pi c}{\lambda}t) + \frac{4\pi d}{\lambda} sin(\frac{2\pi c}{\lambda})dt \tag{7.2}$$

which reduces to:

$$Acos(\frac{4\pi d}{\lambda}) \tag{7.3}$$

Where:

t is time,

T is the averaging interval,

and A is a gain.
As it can be seen, the actual phase shift measured is the cosine of the phase shift, not the phase shift itself. As a result, there will be more than one distance that map to the same phase shift ϕ as expressed in the following equation:

$$\cos(\phi) = \cos(\frac{4\pi d}{\lambda}) = \cos(\frac{2\pi(x+n\lambda)}{\lambda}) \tag{7.4}$$

where:

$$d = \frac{x+n\lambda}{2} = \text{true distance to target}, \tag{7.5}$$

x is the distance corresponding to differential phase ϕ, n is the number of complete modulation cycles.
This introduces a so called ambiguity interval R_a for scenarios where the round trip distance exceeds λ.Thus, sensors transmitting only a single wave have to limit their maximum measurable distance to less than the ambiguity interval R_a. To solve the ambiguity, successive measurements of the same object using two different modulation frequencies can be made. Thus, two equations with x and n as unknowns can be generated allowing to uniquely solve the ambiguity and determine the true distance d.
The main components composing the hardware of the sensor are: light source, light detector, rotating mirror (mirror + motor). The mirror serves in converting the vertical transmitted beam into a horizontal beam and the horizontal received beam into a vertical beam. The hardware of the sensor is depicted in Figure 7.12.

The specifications that can be used to describe and evaluate an optical range sensor are described below. For example, a certain application may desire a high detection range while another real time application, such as fast moving robots, may require fast response time.
Laser Specifications:

- Light source
- Modulated frequency: this affects the accuracy of the measurements and the minimum object size that can be detected.
- Detection range
- Accuracy

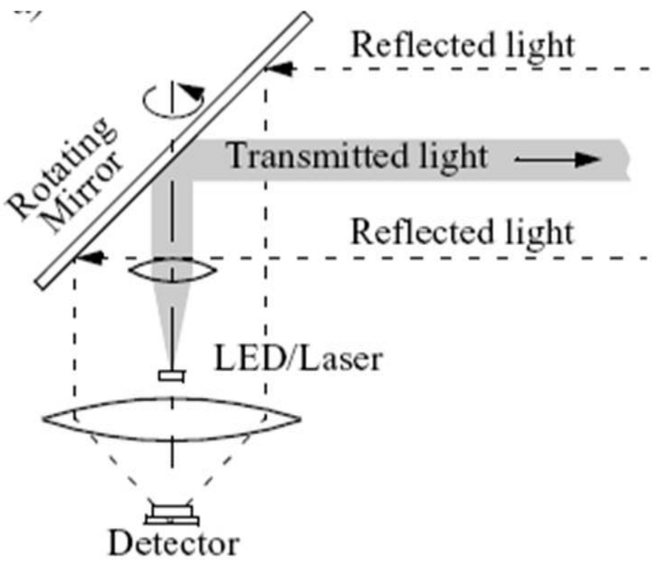


Figure 7.12: Laser Sensor Hardware.

- Angular resolution
- Response time
- Dimensions
- Weight
- Environment: Indoor, outdoor
- Power consumption

Hokuyo sensor

The communication between the sensor and the host can be established using serial RS232C or USB cable. Sensor's data are encoded to reduce the transmission time between the host and sensor. These data should be decoded at the host side before processing them. The Three-Character encoding technique is applied. This encoding technique is used to express data having maximum length of 18 bits. Encoding is done by separating data into upper, middle and lower 6 bits and then 30H is added to convert them into ASCII characters. Figure 7.13 and Figure 7.14 show examples of character encoding and decoding examples.

Measurement connection and data points

This section gives some basic information on sensor's measurement parameters. These parameters are important when reading the measurement data from the sensor. Figure 7.15

5,432 mm = 1010100111000_2

$\downarrow$ Separation

000001_2 010100_2 111000_2

$\downarrow$ Hexadecimal Equivalent

1H 14H 38H

$\downarrow$ Add 30H

31H 44H 68H

$\downarrow$ ASCII Equivalent

1 D h

Figure 7.13: Character encoding example

1 D h = 1 D h

$\downarrow$ Hexadecimal Equivalent

31H 44H 68H

$\downarrow$ Subtract 30H

1H 14H 38H

$\downarrow$ Binary Equivalent

000001_2 010100_2 111000_2

$\downarrow$ Merge

000001010100111000_2

$\downarrow$ Decimal Equivalent

5,432

Figure 7.14: Character decoding example

shows the sensor's measurement details.

The scanner rotates in an anti-clockwise direction when viewed from top. Detection Range (E) is the maximum angle used by the sensor scans for measurement. Angular Resolution is defined as the 360 degrees divided by the Slit Division (F). Measurement points are called Steps. Step 0 is the first measurement point. Step A is the initial measurement point in the detection range. Step B is the sensor front step. Step C is the end point of the detection range. Step D is the last measurement point. Figure7.16 shows the measurement parameters of some sensor models.

Essential Sensor Commands. BM command- Measurement enable command Initially the laser is switched off and the sensor's measurement state is disabled by default. The operator may notice that the sensor LED is blinking on start up indicating that the measurement state is disabled. To enable the measurement state, a serial command has to be sent of the following format:

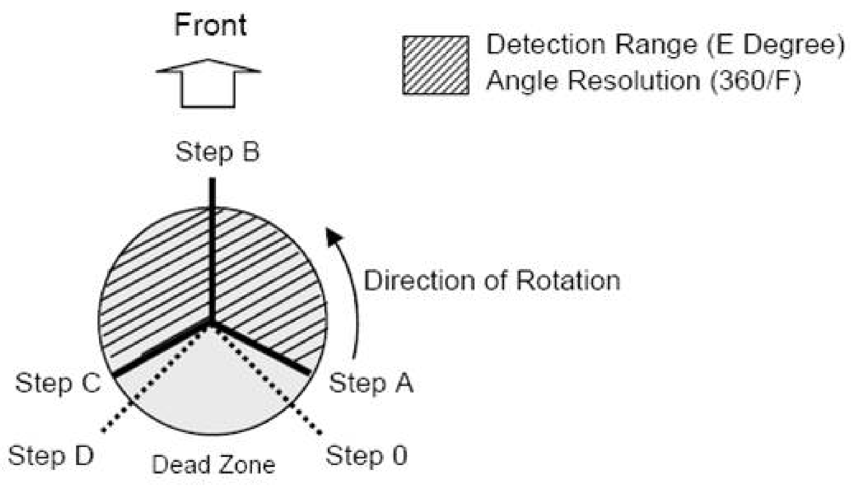


Figure 7.15: Sensor Measurement parameters

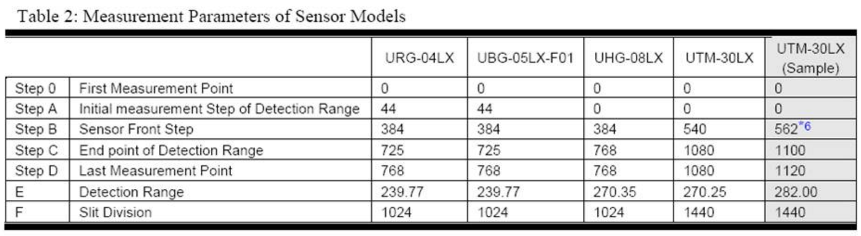


Table 2: Measurement Parameters of Sensor Models

		URG-04LX	UBG-05LX-F01	UHG-08LX	UTM-30LX	UTM-30LX (Sample)
Step 0	First Measurement Point	0	0	0	0	0
Step A	Initial measurement Step of Detection Range	44	44	0	0	0
Step B	Sensor Front Step	384	384	384	540	562[*6]
Step C	End point of Detection Range	725	725	768	1080	1100
Step D	Last Measurement Point	768	768	768	1080	1120
E	Detection Range	239.77	239.77	270.35	270.25	282.00
F	Slit Division	1024	1024	1024	1440	1440

Figure 7.16: The measurement parameters of some sensor models

BM\r

Notice that the commands are case sensitive.

The sensor is expected to reply with the command echo followed with the sensor status which can take one of the following:

0 command received with no errors

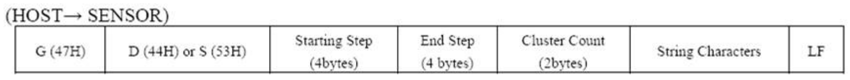
1 unable to control due to laser malfunction

2 laser is already on

GD/GS command send sensors latest measurement upon the reception of the command

Whenever the sensor receives this command it supplies the latest measurement data to the host. If the laser is switched off, it should be switched on by sending BM-Command before the measurement. The laser should be switched off if necessary by sending QT-Command after a measurement is complete.

- Starting Step and End Step: Starting step and End Step can be any points between 0 and maximum step. End Step should be always greater than Starting step. Example:



(HOST→ SENSOR)

G (47H)	D (44H) or S (53H)	Starting Step (4bytes)	End Step (4 bytes)	Cluster Count (2bytes)	String Characters	LF

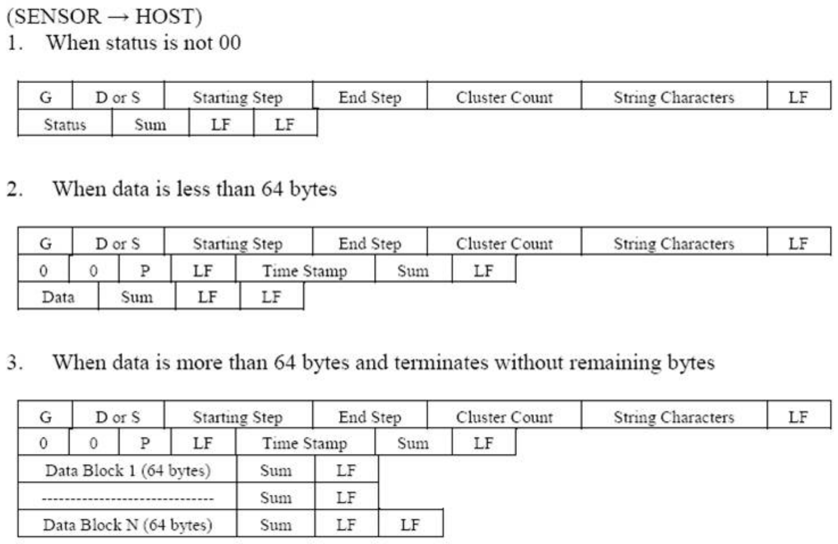
Figure 7.17: GD command

To obtain data from step 10 to 750: Starting point : 0010 (30H,30H, 31H, 30H) End Point : 0750 (30H, 37H, 35H, 30H) Note that the starting/End step has to be specified using four digits where each digit is a byte.

- Cluster Count: Cluster Count is the number of adjacent steps that can be merged into single data. It has a range 0 to 99. When cluster count is more than 1, step having minimum measurement value (excluding error) in the cluster will be the output data. Example: If Cluster Count is 3 and measurement values of 3 adjacent steps in this cluster are 3059, 3055 and 3062, the received data from the sensor will be 3055.

- Scan Interval: Skipping the number of scans when obtaining multiple scan data can be set in Scan Interval. The value should be in decimal. Number of Scans: User can request number of scan data by supplying the count in Number of Scan. If Number of Scan is set to 00 the data is supplied indefinitely unless canceled using [QT-Command] or [RS-Command]. The value should be in decimal.

The sensor reply will vary with the size of the data:



(SENSOR → HOST)

1. When status is not 00

G	D or S	Starting Step	End Step	Cluster Count	String Characters	LF
Status	Sum	LF	LF			

2. When data is less than 64 bytes

G	D or S	Starting Step	End Step	Cluster Count	String Characters	LF
0	0	P	LF	Time Stamp	Sum	LF
Data	Sum	LF	LF			

3. When data is more than 64 bytes and terminates without remaining bytes

G	D or S	Starting Step	End Step	Cluster Count	String Characters	LF
0	0	P	LF	Time Stamp	Sum	LF
Data Block 1 (64 bytes)	Sum	LF				
-----------------------------	Sum	LF				
Data Block N (64 bytes)	Sum	LF	LF			

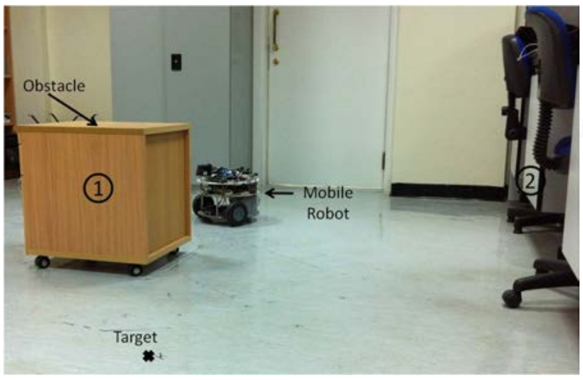
Figure 7.18: Sensor to host

Experimental Results for Geometrical Obstacle Avoidance Algorithm

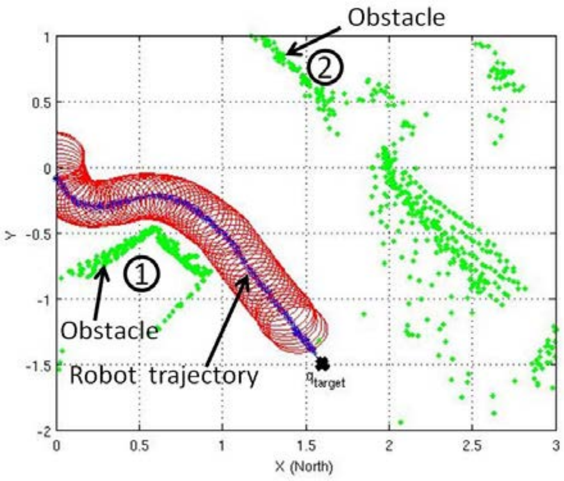
The obstacle avoidance algorithms are tested on the mobile robot platform described earlier. The testing is conducted indoors in a lab environment; the lab furniture are to be avoided. The obstacles are arranged in five different scenarios that vary in difficulty. For all scenarios, the robot initial configuration is $(0,0,-90°)$ while the target x-y location is $(1.6m,-1.5m)$. Hence, the initial error in position is $2.1932m$

Environment scenario 1. In scenario 1, shown in Figure 7.19a, an obstacle is placed along the robot direct path, which is the straight line that connects the robot's initial configuration to the target configuration. There are also other obstacles surrounding the robot. The robot's trajectory from its initial configuration to its target configuration is depicted in Figure 7.19b. The length of the trajectory is $2.2711m$. The robot velocities in the global frame are smooth as shown in Figure 7.19c. The control action is shown in Figure 7.19d. It should be noted that when the robot moves in a straight line (-2° $\leq \gamma_{desired} \leq 2°$), the turning radius is infinite. However, to keep the plot in range the infinite radius are given as 0.5.

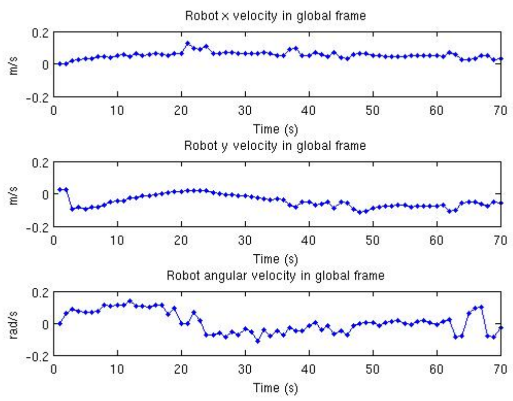
For a detailed analysis of the algorithm, we look at the intermediate values of some of the critical time samples as shown in Figure 7.20. At sample = $18s$, the robot can only see the front side of the square obstacle. Therefore, most of the front sectors of the robot in the polar histogram are classified as occupied. The reference steering angle is around $-20°$, hence, it belongs to an occupied sector. The best alternative candidate is specified by $\gamma_{desired}$ which is equal to 30°. The turning radius is around $0.205m$ as shown in Figure 7.19d. Figure 7.20b shows the robot at sample $26s$. The robot can only view the right side of the obstacle. γ_{ref} is around $-54°$ and $\gamma_{desired}$ is chosen to be $-10°$. The robot turns right with a radius of $0.5m$. Finally, Figure7.20c shows the robot at sample $49s$. γ_{ref} resides in a free sector. Hence, $\gamma_{desired}$ is simply equated to γ_{ref} and the robot moves straight to q_{target}. The robot completed the course in $70s$.



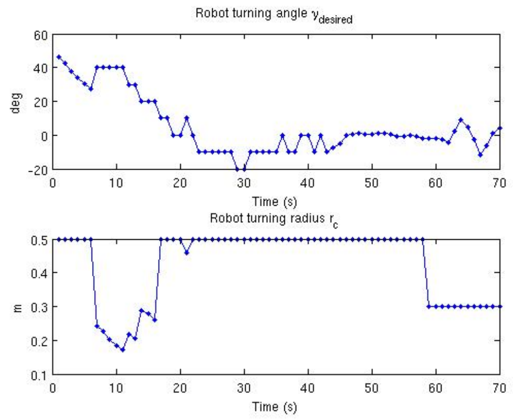
(a)



(b)

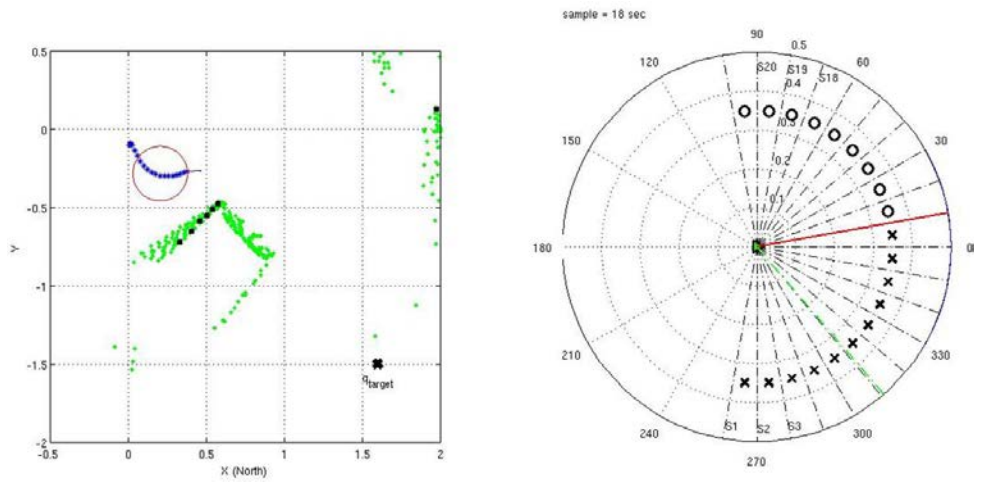


(c)

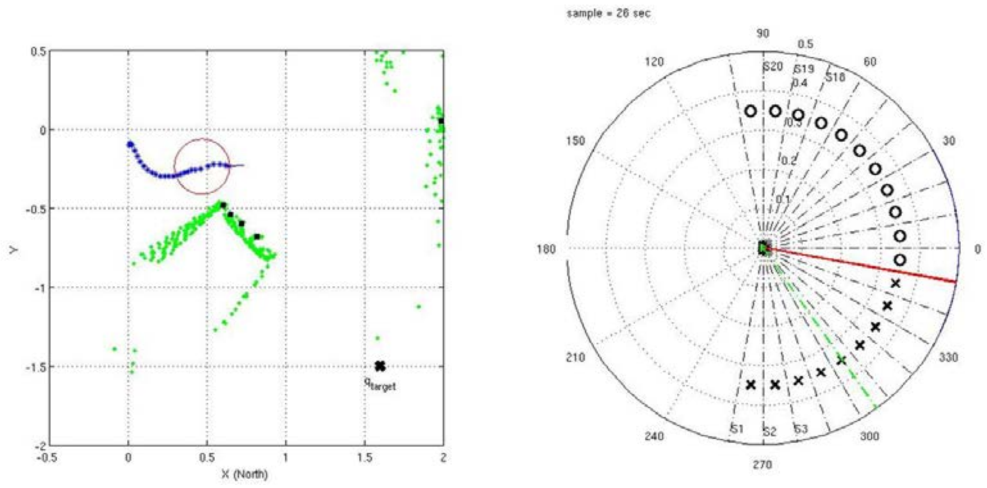


(d)

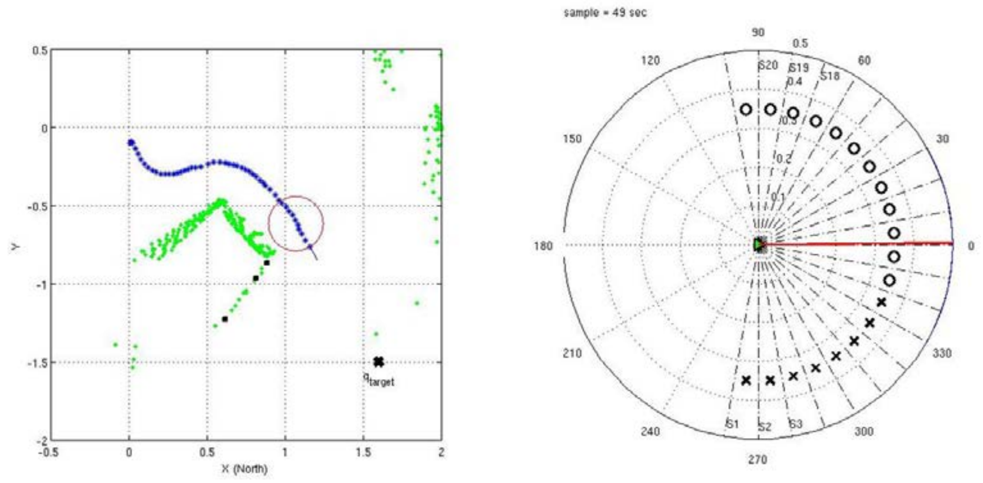
Figure 7.19: Experiment 1 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue. (c) describes the robot velocities; (d) describes the robot control vector.



(a)



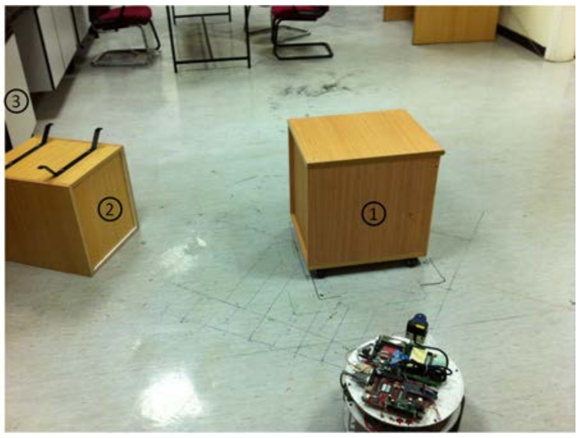
(b)



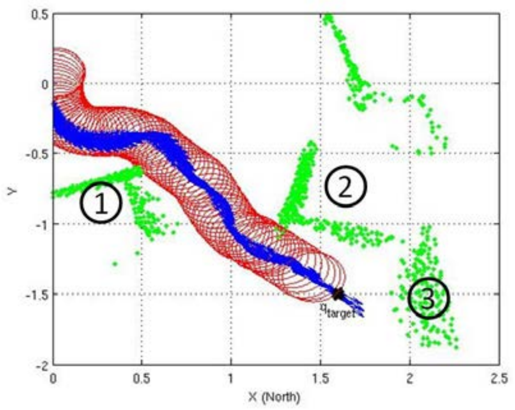
(c)

Figure 7.20: Snapshot of the algorithm parameters at different instances. The obstacles seen by the robot at each instant of time are shown in the Cartesian coordinates as black dots. The solid yellow line is an approximation to the obstacle contour. The polar histogram shows the classified sectors and the reference steering angle is shown as a solid red line while the desired steering angle is shown as a dashed green line.

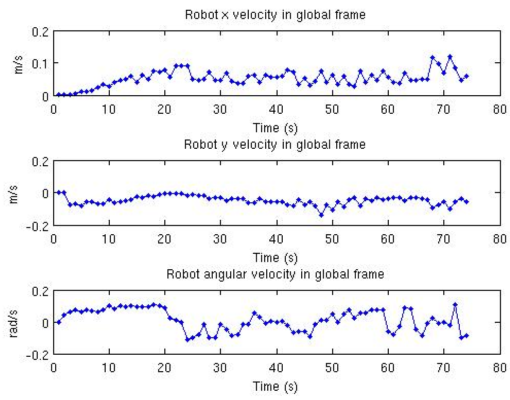
Environment scenario 2. A more complex environment is tested in scenario 2, shown in Figure 7.21a. Obstacles 1 and 2 are oriented to form a passage that has a wide entrance and narrow exit. Nevertheless, the passage is wide enough for the robot to pass through. The robot successfully completes the mission in $74s$ with a trajectory of length $2.2539m$, shown in Figure 7.21b. The robot velocities and control action are depicted in Figure 7.21c, and Figure 7.21d. The polar histograms when the robot enters, moves inside, and exits the passage are shown in Figure 7.22. Towards the end of the trajectory, specifically at sample $59s$, the detection range R_{safe} is reduced from $0.5m$ to $0.2m$. This is especially useful because the target configuration is positioned in the proximity of obstacles 2 and 3. If R_{safe} retained its original large value, the robot will take unnecessary maneuvers to approach q_{target}.



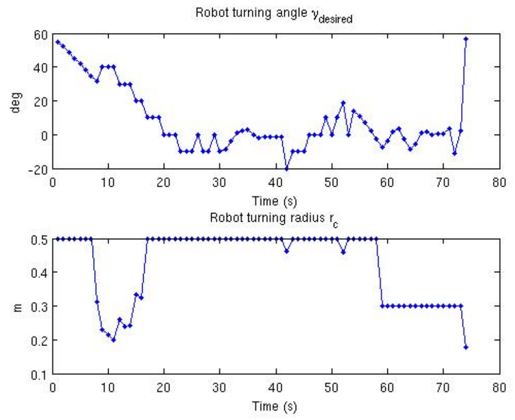
(a)



(b)

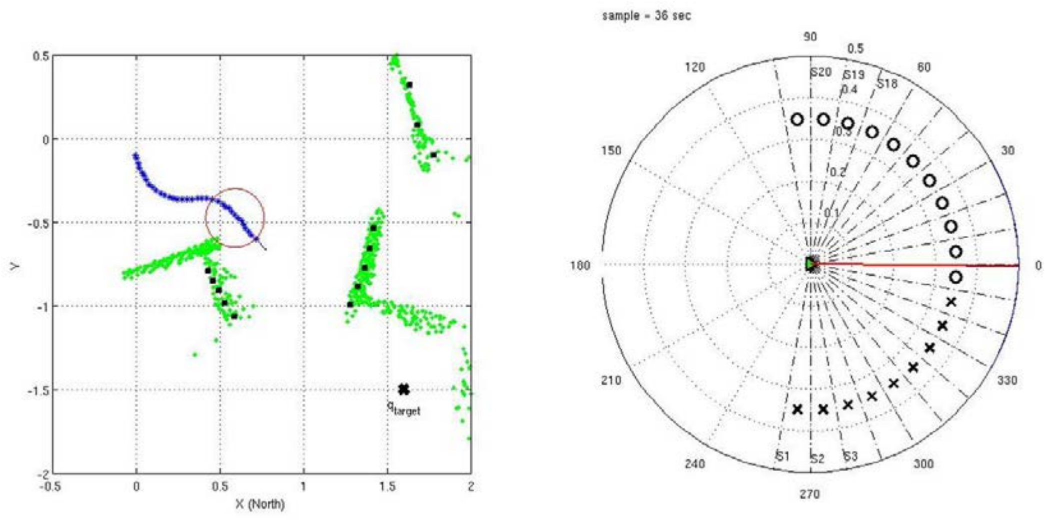


(c)

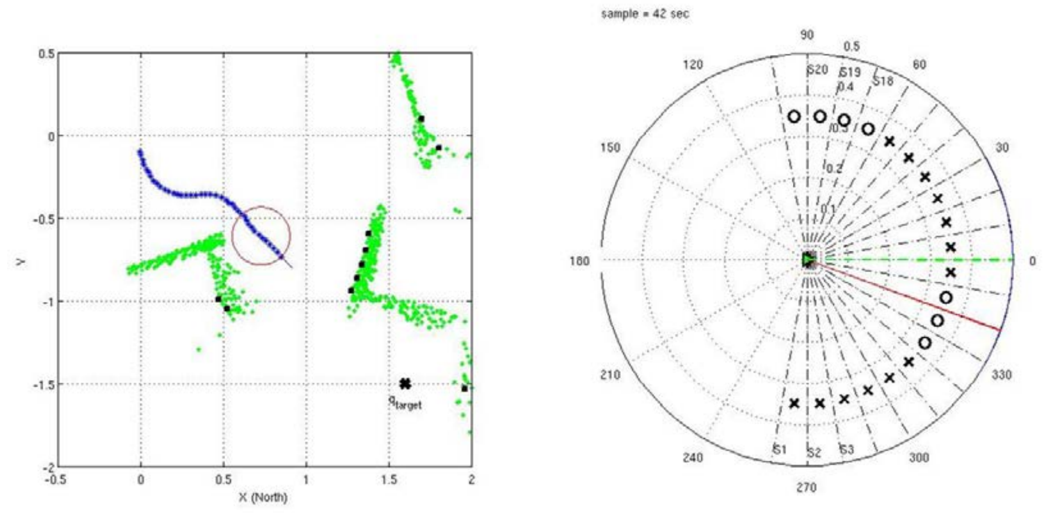


(d)

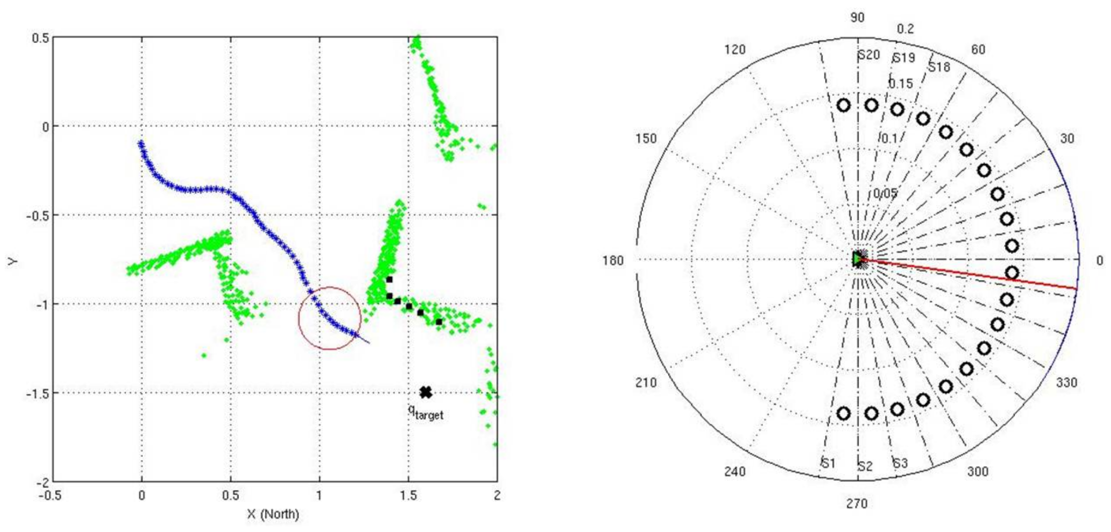
Figure 7.21: Experiment 2 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue. (c) describes the robot velocities; (d) describes the robot control vector.



(a)



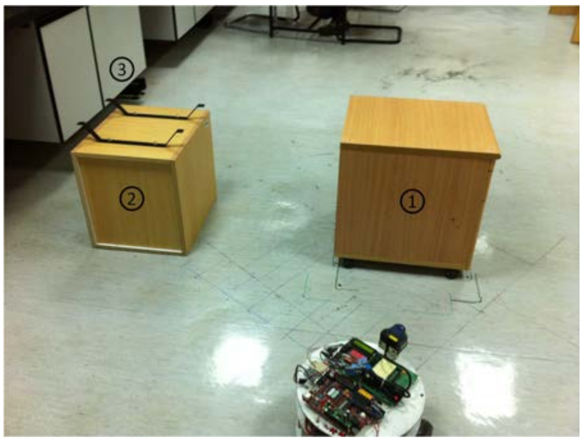
(b)



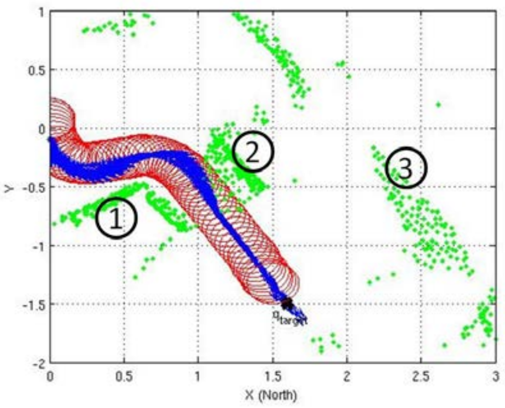
(c)

Figure 7.22: (a) shows the robot entering the passage; (b) shows the robot inside the passage. (c) shows the robot exiting the passage;

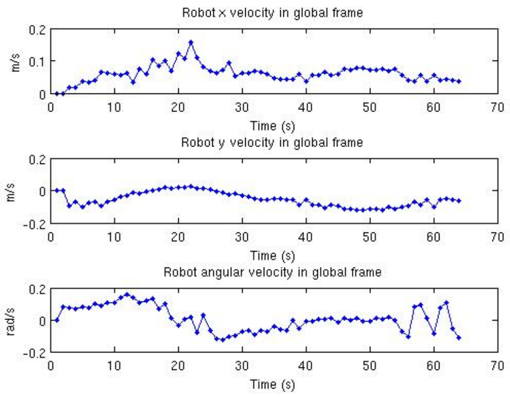
Environment scenario 3. In scenario 3, the obstacles are arranged such that the passage between them are narrower than the one in scenario 2 as shown in Figure 7.30a. Here, the robot is tested on its ability to detect navigable gaps that are slightly larger than the robot size. The robot successfully passes the narrow passage and settled at the target configuration as depicted in Figure 7.23b. The robot velocities and control action are depicted in Figure 7.23c and Figure 7.23d. The length of the trajectory is 2.3792*m*. The instances where the robot detects a passage, moves through it, and exits it are shown in Figure 7.24.



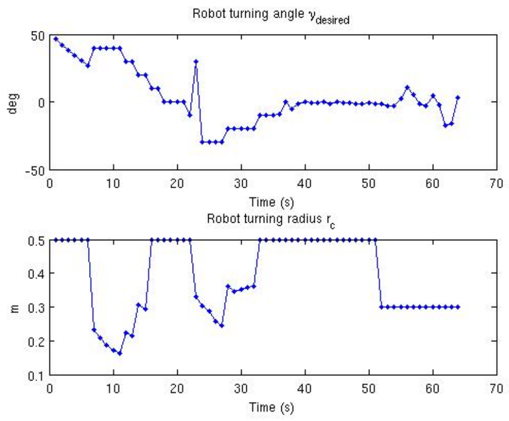
(a)



(b)

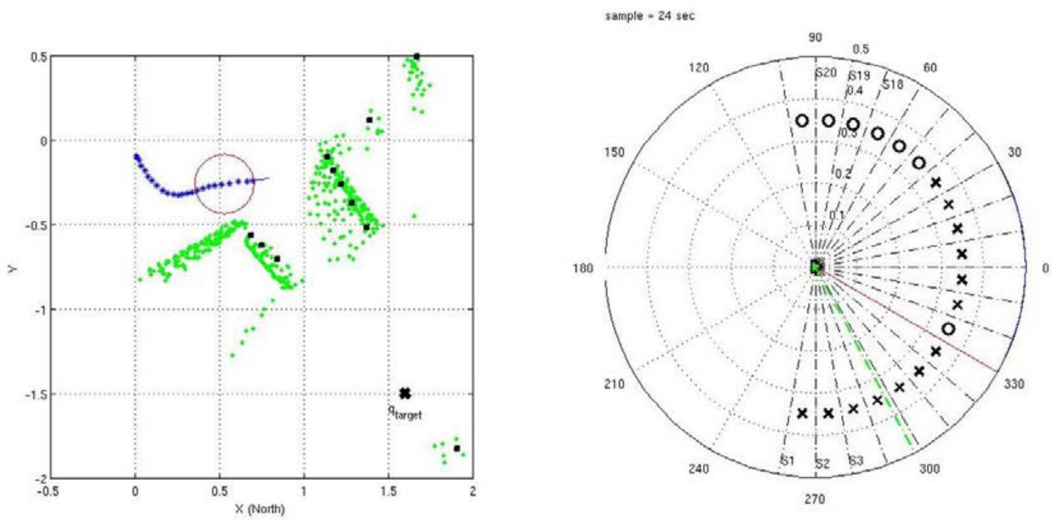


(c)

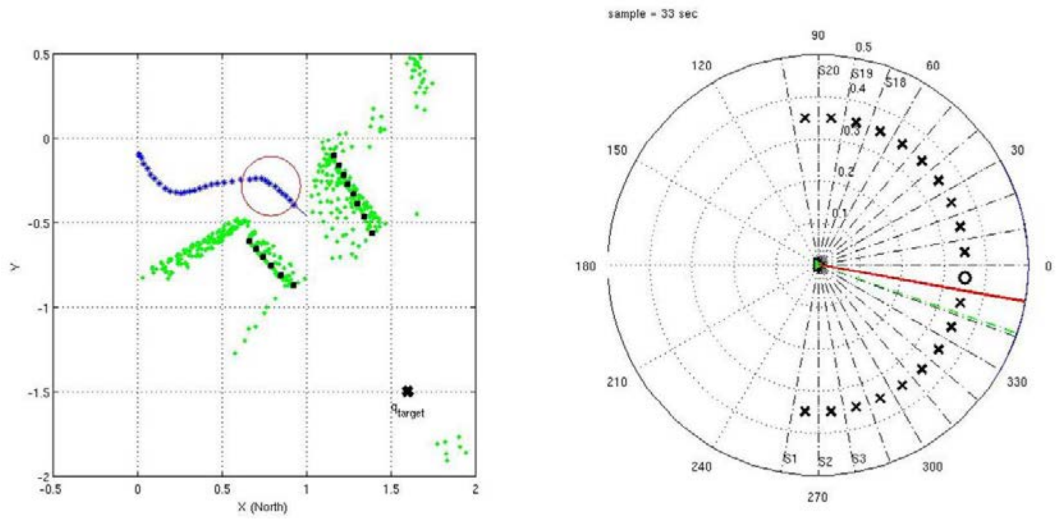


(d)

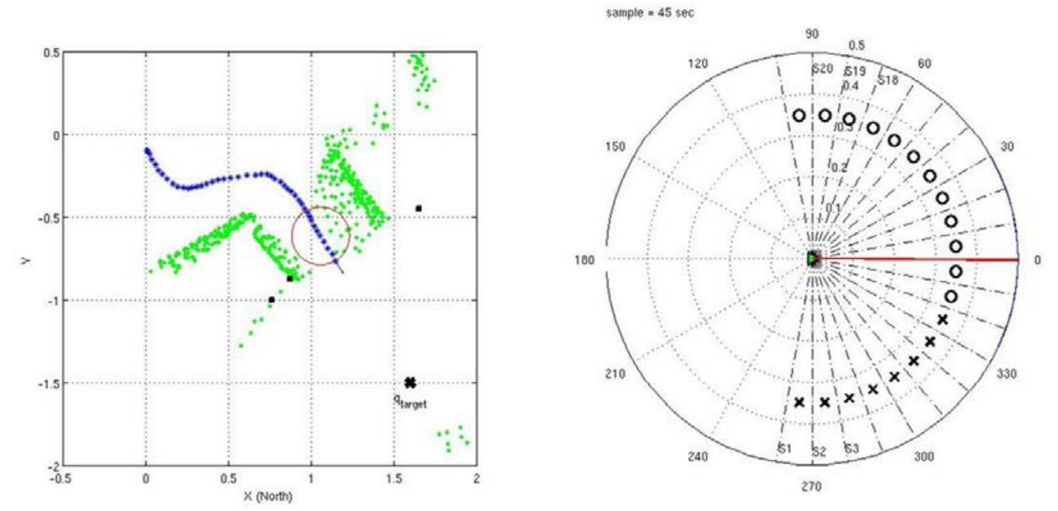
Figure 7.23: Experiment 3 using obstacle avoidance algorithm. (a) shows the robot testing environment; (b) shows the robot trajectory. (c) describes the robot velocities; (d) describes the robot control vector.



(a)



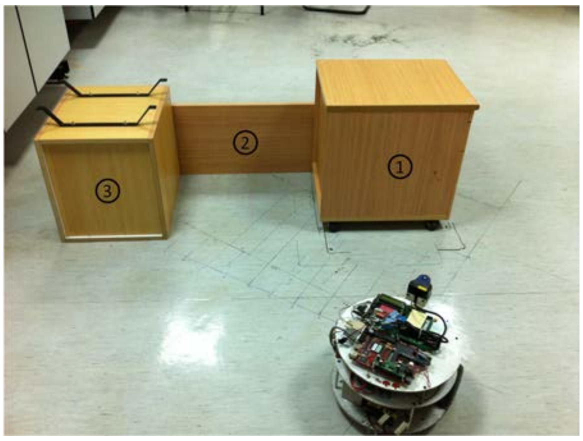
(b)



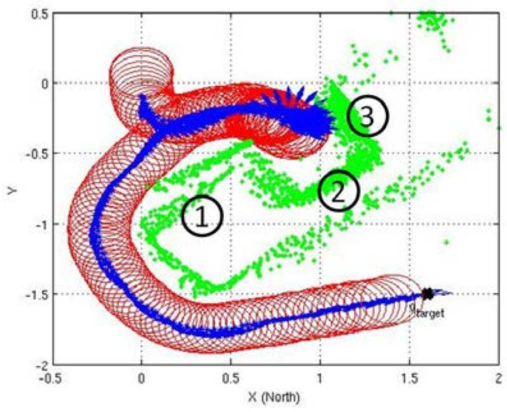
(c)

Figure 7.24: (a) shows the robot entering the passage; (b) shows the robot inside the passage. (c) shows the robot exiting the passage;

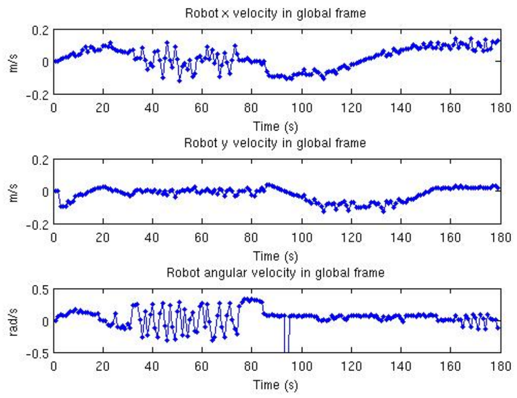
Environment scenario 4. Scenario 4 is similar to scenario 3, however, an obstacle is added to block the passage exit in order to form a dead end. The scenario is depicted in Figure 7.25a while the trajectory is shown in Figure 7.25b. The robot velocities and control action are depicted in Figure 7.25c and Figure 7.25d respectively. During samples $30s$ to $84s$ the robot exhibits some fluctuations. The turning angle $\gamma_{desired}$ direction fluctuates between left and right as shown in Figure 7.25c. At time $25s$ the robot detects two gaps: a narrow gap and a large gap, as shown in Figure 7.26a. The robot turns towards the narrow one because it is closer to q_{target}. However, at time sample $31s$, the robot gets closer and finds out that the narrow gap is in fact blocked as shown in Figure 7.26b. Hence, the robot steers left. After turning left, the dead end will appear as a gap and the robot attempts to turns towards it as shown in Figure 7.26c. Despite the oscillations, the robot completes the mission at instant $179s$ after traveling $6.0243m$.



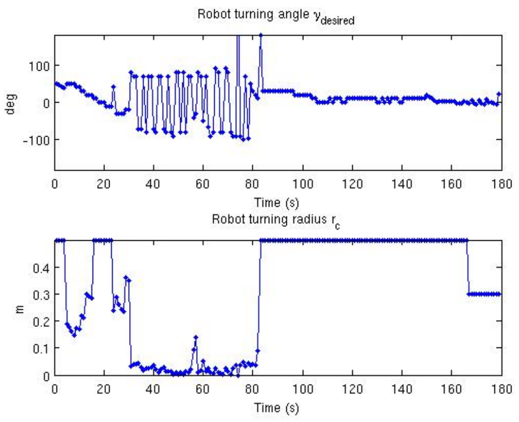
(a)



(b)

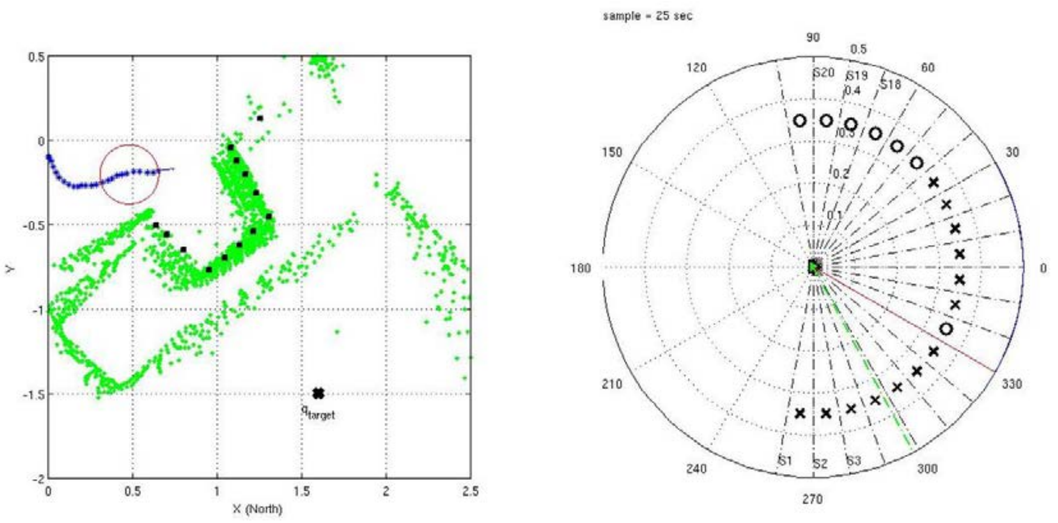


(c)

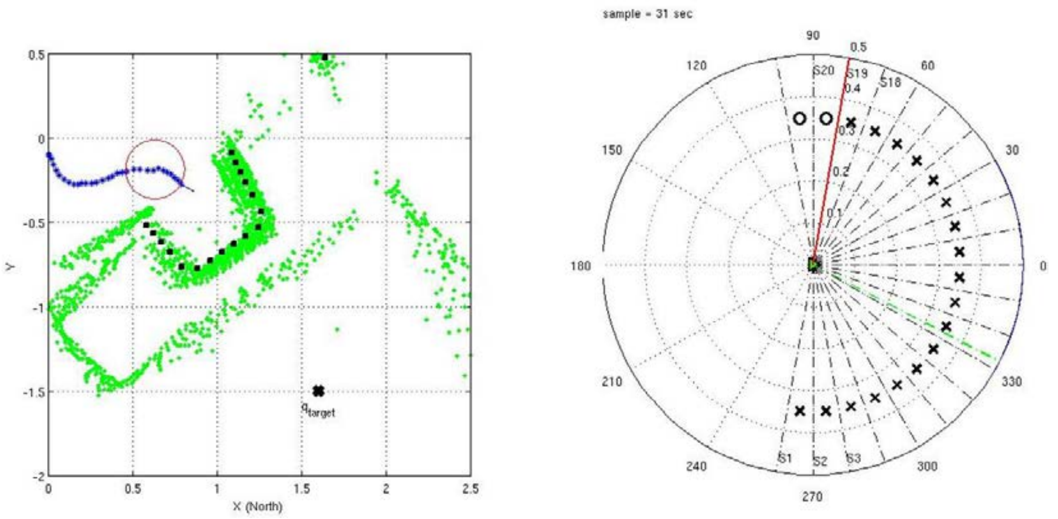


(d)

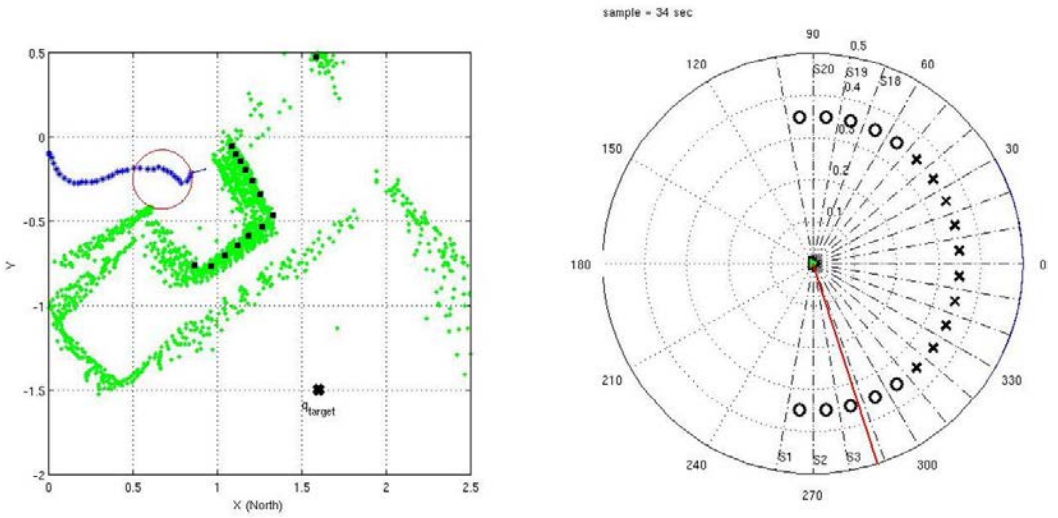
Figure 7.25: Experiment 4 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.



(a)



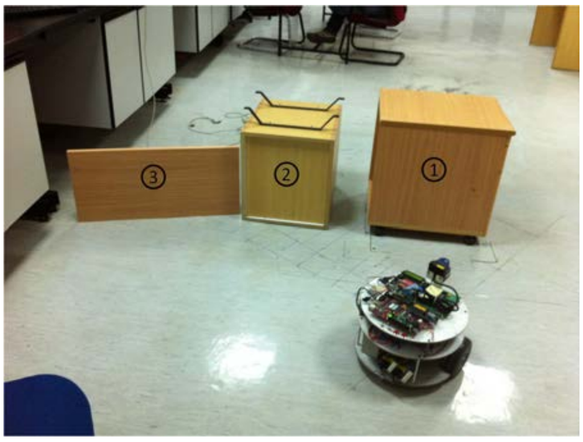
(b)



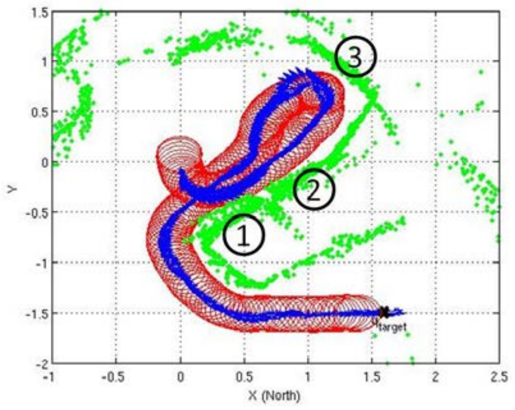
(c)

Figure 7.26: (a) robot detects a passage; (b) robot discovers a dead end and attempt to turn away (c) after the robot moves away, the dead end appears as a gap.

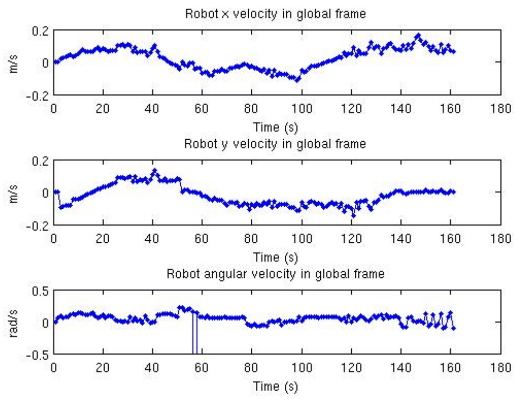
Environment scenario 5. The three obstacles in scenario 5 are placed such that the robot initial steering takes it to a blocked path as depicted in Figure 7.32a. However, the robot constant search for a gap eventually brings it to the target configuration as shown in Figure 7.27b. The robot velocities and control action are depicted in Figure 7.27c and Figure 7.27d respectively.



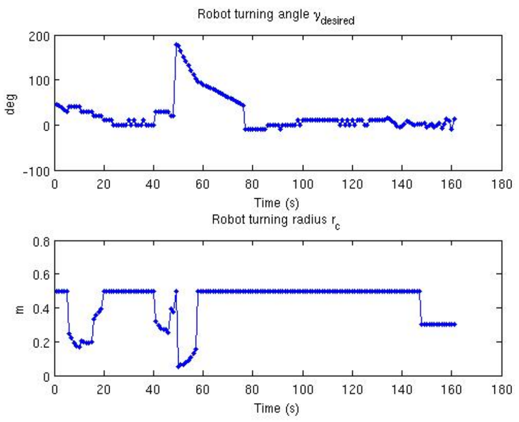
(a)



(b)



(c)



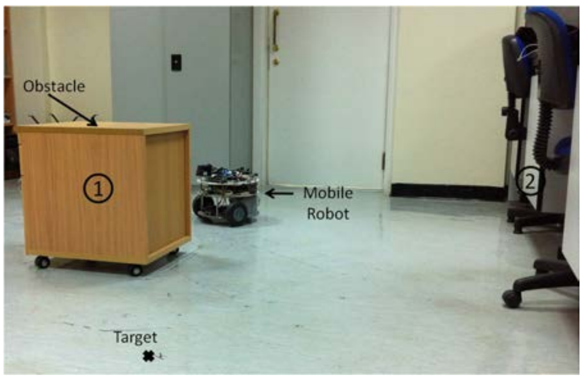
(d)

Figure 7.27: Experiment 5 using obstacle avoidance algorithm. (a) shows the robot testing environment; (b) shows the robot trajectory. (c) describes the robot velocities; (d) describes the robot control vector.

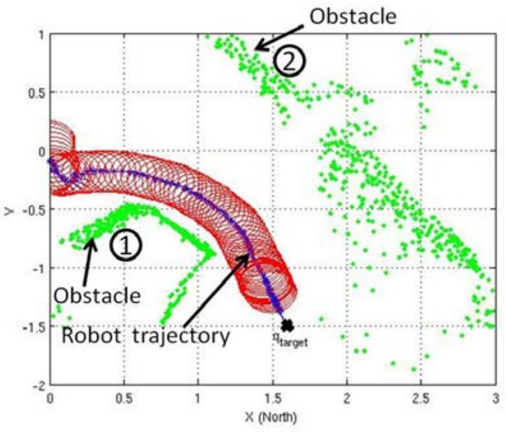
Experimental Results for Neural Network-based Algorithm

The neural network algorithm is tested on the robot described earlier. The scenario used to test the network are the same ones in section 7.

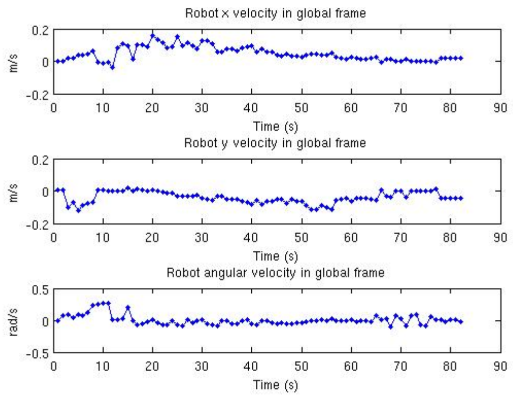
Environment scenario 1. In scenario 1, shown in Figure 7.28a, the robot successfully avoids the obstacle. The trajectory is shown in Figure 7.28b. The robot velocities and control action are shown in Figure 7.28c and Figure 7.28d respectively. The length of the trajectory is $2.4157m$ and it is completed in $82s$.



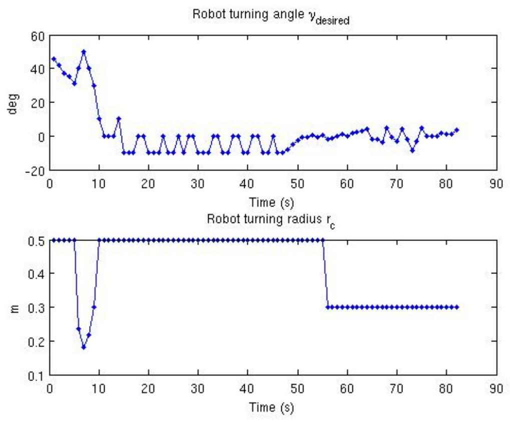
(a)



(b)



(c)



(d)

Figure 7.28: Experiment 1 using neural network-based algorithm. (b) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

Environment scenario 2. In scenario 2 depicted in Figure 7.29a, the robot successfully avoids the 2 obstacles, and drives its way to q_{target} as shown Figure 7.29b. The robot velocities and control action are shown in Figure 7.29c and Figure 7.29d respectively. The length of the trajectory is 2.3016 and it takes $106s$ to execute.

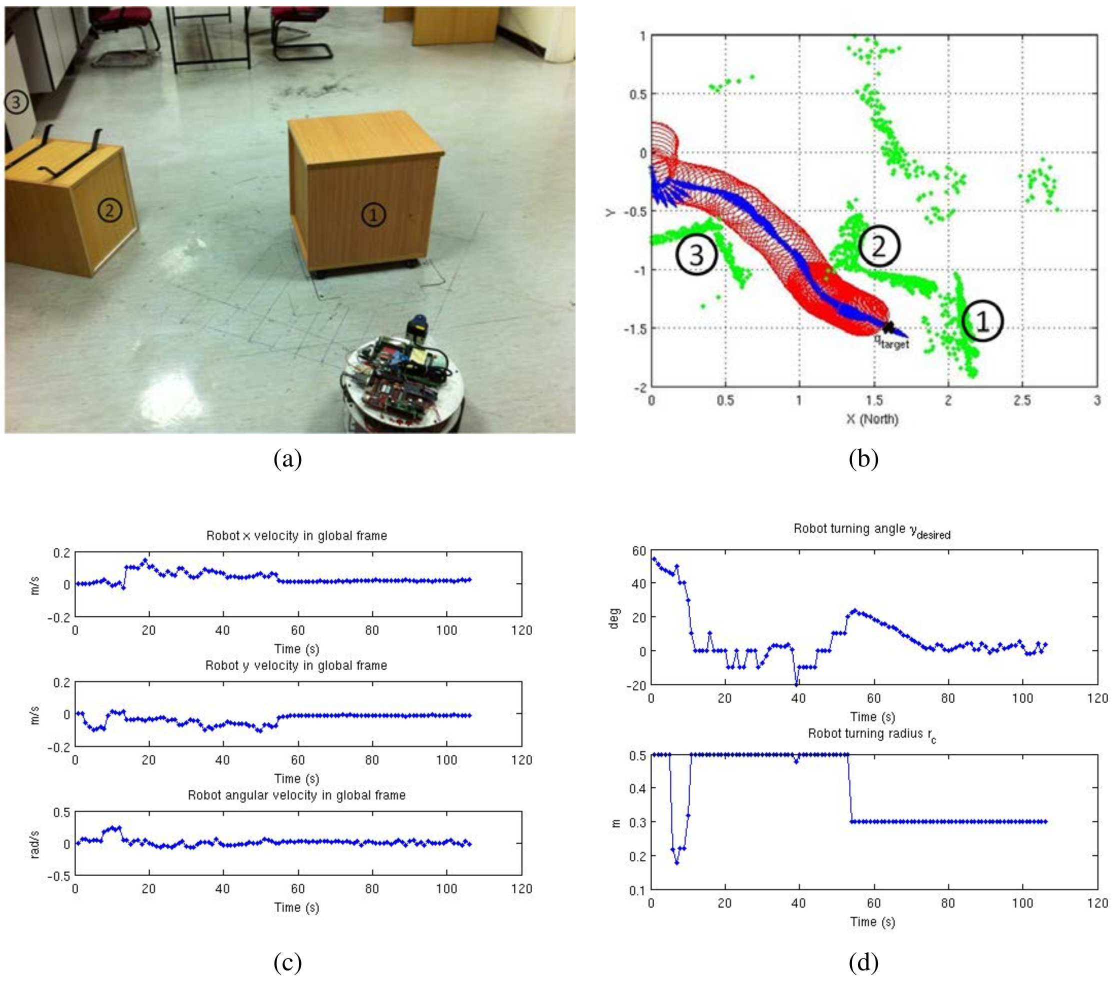
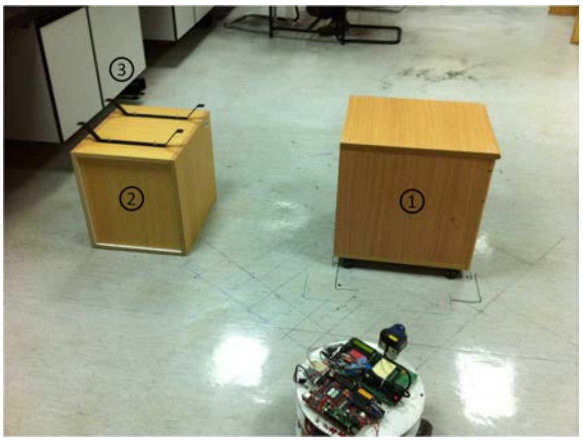
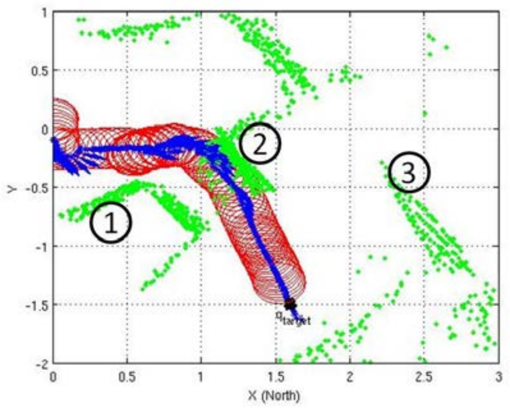


Figure 7.29: Experiment 2 using neural network-based algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

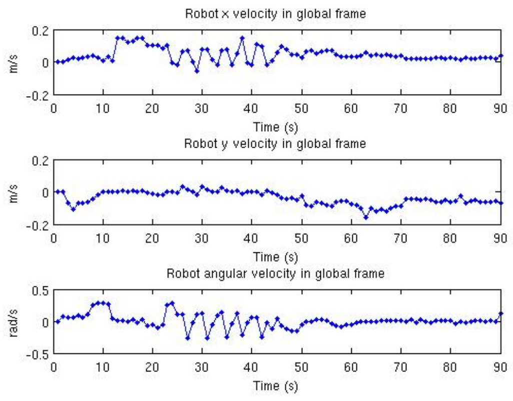
Environment scenario 3. In scenario 3 depicted in Figure 7.30a, the robot correctly identified the gap as navigable and went in between. However, the left side of the robot touched the obstacle. The trajectory is shown in Figure 7.30b and the robot motion and control action are shown in Figure 7.30c and Figure 7.30d. The length of the trajectory is 2.1349*m* and is completed in 90*s*.



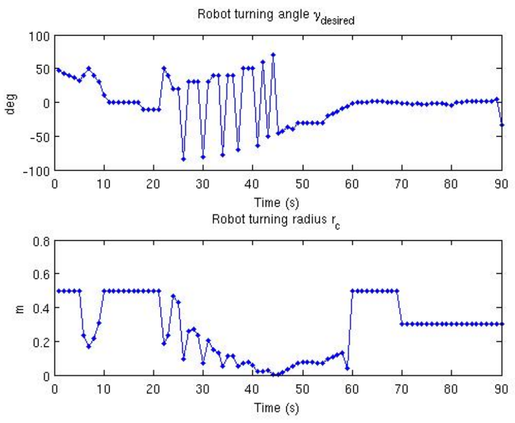
(a)



(b)



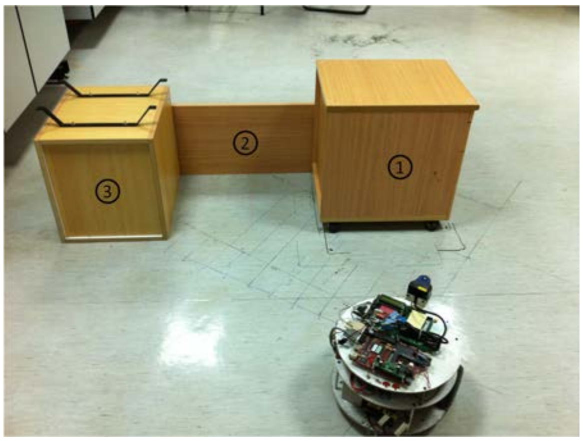
(c)

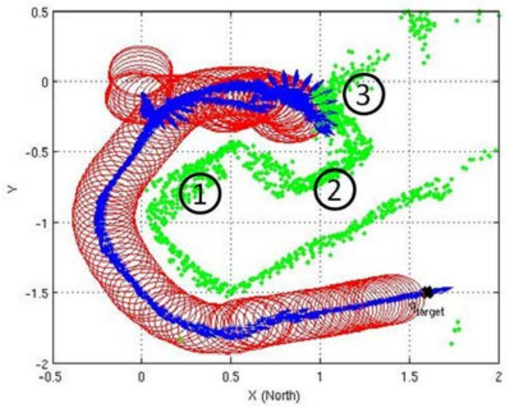


(d)

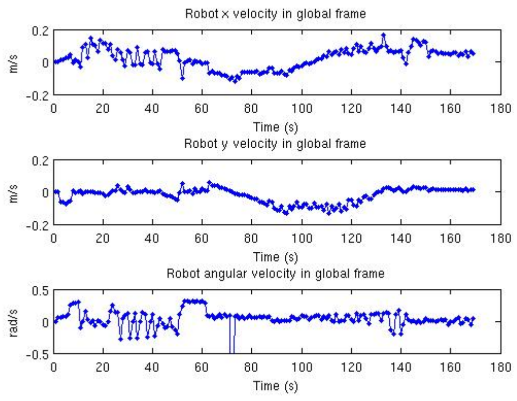
Figure 7.30: Experiment 3 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

Environment scenario 4. In scenario 4 depicted in Figure 7.31a, the robot discovers the dead end and turns around the obstacles to reach q_{target}. The trajectory is depicted in Figure 7.31b. The robot velocities and control action are depicted in Figure 7.31c and Figure 7.31d. The trajectory exhibited some fluctuations. The length of the trajectory is $5.4778m$ and is completed in $169s$.

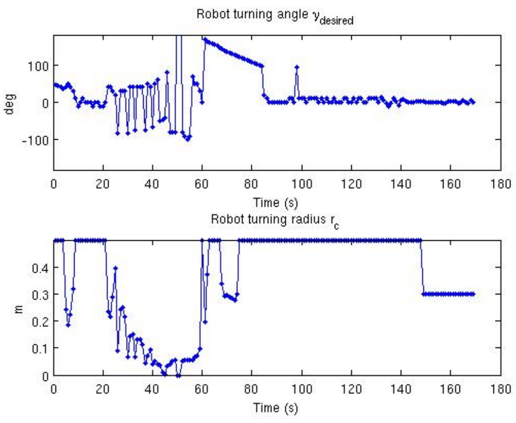

(a)



(b)



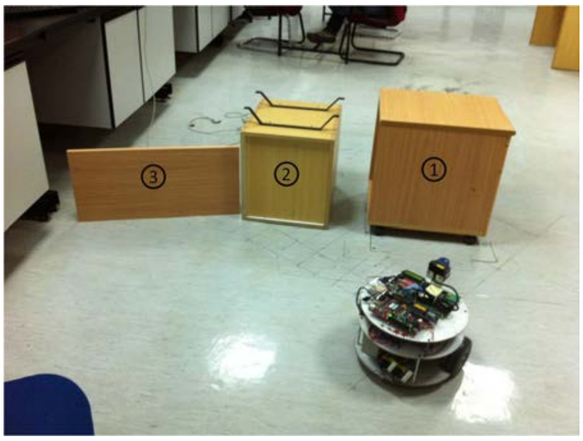
(c)



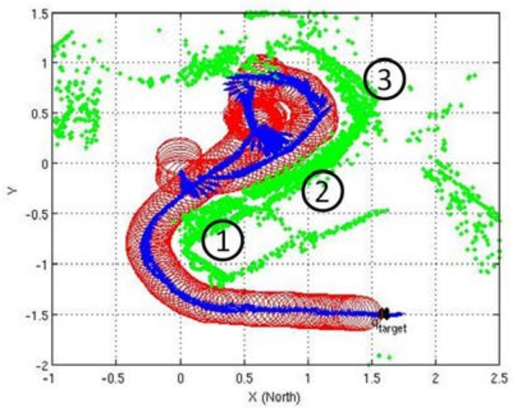
(d)

Figure 7.31: Experiment 4 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

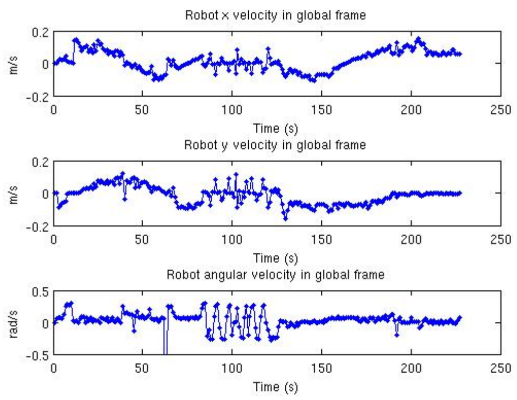
Environment scenario 5. In scenario 5 depicted in Figure 7.32a, the robot avoids the narrow gap, discovers a blocked path, and eventually progresses to the target configuration. The trajectory is depicted in Figure 7.32b and the robot velocities and control action are depicted in Figure 7.32c and 7.32d. The length of the trajectory is 7.4457m and it takes 227s.



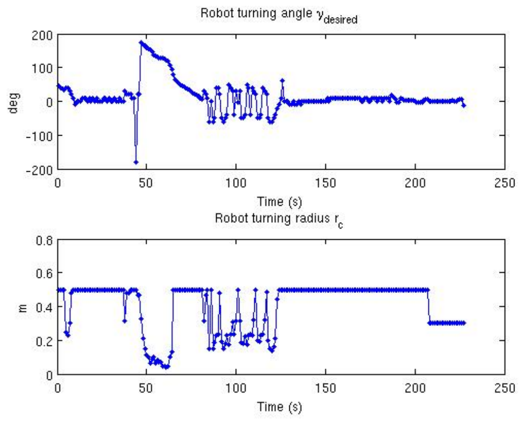
(a)



(b)



(c)



(d)

Figure 7.32: Experiment 5 using obstacle avoidance algorithm. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

8. Conclusions

The thesis presents two reactive navigation algorithms for wheeled mobile robot under non-holonomic constraints and in unknown environments. The mobile robot travels to a pre-defined goal position safely and efficiently without any prior map of the environment. The first method incorporates the dimensions and shape of the robot to determine the set of all possible collision-free steering angles. The steering angle that falls in the widest gap and is closest to the target is selected. The next stage in the algorithm takes into account the non-holonomic constraints of differentially steered robots by computing circular trajectories with adaptive radius of curvature. The second reactive navigation algorithm introduces a neural network based reactive navigation algorithm. The algorithm aims to generate an optimized path by using a user-defined objective function which minimizes the traveled distance to the goal position while avoiding obstacles. To this end, a diagonal recurrent neural network (DRNN) has been employed to achieve the necessary generalization capability across a variety of indoor environments. The network is trained through off-line learning followed by an on-line learning algorithm with guaranteed convergence. A mobile robot was built and used to assess the performances of the algorithms. The performances of the algorithms are verified over a variety of real unstructured indoor environments using an autonomous mobile robot platform. The results demonstrated that the algorithm is capable of driving the robot safely through different obstacle arrangements.

Appenndix A: Convert Matlab Code to C Code Guide

Matlab Embedded coder can be used to automatically convert Matlab code into C/C++ code. The generated C/C++ code can be pakaged as an executable or library. The steps needed are explained below and are illustrated by an example:

- Install prerequiste products
- Set up C/C++ compiler
- Make Matlab code compliant with the embedded matlab subset
- Choose your desired package format
- Write your main function
- Compile the generated C/C++ code

Install prerequiste products

The software packages needed to be installed on your PC are: Matlab, Simulink, C/C++ compiler, and the Real-Time Workshop.

Set up C/C++ Compiler

Run ‘mex -setup’ in matlab command window. Choose your desired compiler. Make sure you address any repoted warnings saying that matlab does not support the current compiler version.

Make Matlab Code Compliant with Embedded Matlab Subset

Not all Matlab functions can be converted. A subset of Matlab functions called Embedded Matlab subset is supported. An example of unsupported function is 'plot.' You can check at design time for violations by adding '%#eml' to your Matlab file. In this case, The code analyzer will report the detected violations. To further make sure that all your code is compliant with the embedded matlab subset, by generating a mex function and checking for violation at runtime. For a simple example of a compliant matlab code see absval.m.

```
absval.m
%#eml
funtion y = absval(u)
y = abs(u);
```

Choose your desired Package Format

The Embedded matlab coder can convert the Matlab file into three packages: Mex, Embeddable C/C++ code and complie it into an executable, Embedded C/C++ code and compile it into a library.
To generate a Mex use: emlc MyFcn
To generate embeddable matlab code and compile it into a library: emlc -T rtw:lib MyFcn
To generate a library for our previous code example run:
emlc -T rtw:lib absval -eg0.0

After envoking this command, several files will be generated. Two important files we will use further in our code are: absval.h and absval.a.

Write your main Function

Make sure your matlab function include the header file of the generated function. In our example its absval.h. Also Make sure that you use the initialize and terminate functions before and after you call the matlab function.

```
/*
```

```
** main.c
*/
#include <stdio.h>
#include <stdlib.h>
#include "absval.h"

int main(int argc, char *argv[])
{
    absval_initialize();

    printf("absval(-2.75)=%g\n", absval(-2.75));

    absval_terminate();
    return 0;
    }
```

Compile the generated C/C++ code

If you have chosen to convert the Matlab code into a C/C++ library, then you can complile your main code in linux command prompt as follows:

gcc -o main main.c absval.a

To run the executable:

./main

Vita

Mariam Qusai Al-Sagban was born on December 9, 1983, in Abu Dhabi, United Arab Emirates. She graduated from Rosary School, Abu Dhabi in 2001. She attended the American University of Sharjah in Sharjah, UAE from which she graduated with honor (cum laude), in 2006. Her degree was a Bachelor of Science in Electrical Engineering and a minor in Applied and Computational Mathematics.

In 2007, Ms. Al-Sagban began a Masters program in Mechatronics Engineering at the American University of Sharjah. She was awarded the Master of Science degree in Mechatronics Engineering in 2012.

Publications

- Al-Sagban, M.; El-Halawani, O.; Lulu, T.; Al-Nashash, H.; Al-Assaf, Y.; , "Brain computer interface as a forensic tool," Mechatronics and Its Applications, 2008. ISMA 2008. 5th International Symposium on , vol., no., pp.1-5, 27-29 May 2008
- M. Al-Sagban and R. Dhaouadi, Reactive Navigation Algorithm for Wheeled Mobile Robots under Non-Holonomic Constraints, IEEE International Conference on Mechatronics (ICM 2011), Istanbul, Turkey, 13-15 April 2011.

Bibliography

[1] J. Minguez, F. Lamiraux, and J.-P. Laumond, “Motion planning and obstacle avoidance,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 827–852, 10.1007/978-3-540-30301-5_36. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30301-5_36

[2] B. Kuipers, “Potential fields and model predictive control,” www.cs.utexas.edu/~pstone/.../week9b-ben-potential-fields.ppt.

[3] H. R. Everett, *Sensors for mobile robots: theory and application*. Natick, MA, USA: A. K. Peters, Ltd., 1995.

[4] “Robots attempt record breaking pacific ocean voyage,” http://www.bbc.co.uk/news/technology-15790088.

[5] “Honda shows smarter robot, helps in nuclear crisis,” http://www.taiwannews.com.tw/etn/news_content.php?id=1753308.

[6] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Scituate, MA, USA: Bradford Company, 2004.

[7] W. H. Mark Spong and M. Vidyasagar, *Robot Modeling and Control*. New York, USA: Wiley, 2006.

[8] M. V. Mark W. Spong, Seth Hutchinson, *Principles of robot motion: Theory, algorithms, and implementation*. Cambridge, Mass. [u.a.]: MIT Press, 2005.

[9] G. Campion and W. Chung, “Wheeled robots,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 391–410, 10.1007/978-3-540-30301-5_18. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30301-5_18

[10] O. Mohareri and R. Dhaouadi, “A neural network based adaptive tracking controller for nonholonomic wheeled mobile robots with unknown dynamics,” in *Proc. of the ASME 2010 International Mechanical Engineering Congress Exposition (IMECE2010)*, vol. 6, November 12-18 2010.

[11] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[12] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278 –288, jun. 1991.

[13] J. Minguez and L. Montano, "Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios," *IEEE Transactions on Robotics and Automation*, no. 1, pp. 45–59, Feb.

[14] I. Ulrich and J. Borenstein, "VFH+: reliable obstacle avoidance for fast mobile robots," vol. 2, may. 1998, pp. 1572 –1577 vol.2.

[15] ——, "VFH*: local obstacle avoidance with look-ahead verification," vol. 3, 2000, pp. 2505 –2511 vol.3.

[16] C. Silva, M. Crisostomo, and B. Ribeiro, "Monoda: a neural modular architecture for obstacle avoidance without knowledge of the environment," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 6, 2000, pp. 334 –339 vol.6.

[17] D. Janglová, "Neural Networks in Mobile Robot Motion," *Inernational Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 15–22, 2004.

[18] V. Castro, J. Neira, C. Rueda, J. Villamizar, and L. Angel, "Autonomous Navigation Strategies for Mobile Robots using a Probabilistic Neural Network (PNN)," *33rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 2795–2800, Nov.

[19] H. T. Trieu, H. T. Nguyen, and K. Willey, "Advanced obstacle avoidance for a laser based wheelchair using optimised bayesian neural networks," in *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, aug. 2008, pp. 3463 –3466.

[20] M. Al-Sagban and R. Dhaouadi, "Reactive navigation algorithm for wheeled mobile robots under non-holonomic constraints," in *Proceedings of the IEEE International Conference on Mechatronics (ICM11)*, April 2011, pp. 504–509.