2018

Real-time Path Planning and Obstacle Avoidance for Mobile Robots with Actuator Faults

Ravindra Vibha Bellur
Wright State University

Real-time Path Planning and Obstacle Avoidance for Mobile Robots with Actuator Faults

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

by

Vibha Bellur Ravindra
B.E., Visvesvaraya Technological University, 2015

2018
Wright State University

Wright State University
COLLEGE OF GRADUATE SCHOOL

July 25, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Vibha Bellur Ravindra ENTITLED Real-time Path Planning and Obstacle Avoidance for Mobile Robots with Actuator Faults BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Electrical Engineering.

Xiaodong Zhang, Ph.D.
Thesis Director

Brian Rigling, Ph.D.
Chair, Department of Electrical Engineering

Committee on
Final Examination

Dr. Xiaodong Zhang

Dr. Luther Palmer

Dr. Saiyu Ren

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Bellur Ravindra, Vibha. M.S.E.E, Department of Electrical Engineering, Wright State University, 2018. *Real-time Path Planning and Obstacle Avoidance for Mobile Robots with Actuator Faults.*

In this research, a fault-tolerant path planning and obstacle avoidance algorithm is implemented for mobile ground robots. The fault-tolerant system architecture consists of the following three components: (1) an online fault diagnosis module for detecting and estimating actuator faults in the mobile robot; (2) a localization and mapping module for dynamic terrain mapping and obstacle detection; (3) a path planning and obstacle avoidance module for generating way points from a given point to the desired destination, avoiding the detected obstacles and moving along these way points. The online fault diagnostic information is used to compensate for the effect of faults, enabling the robot to automatically move to the destination even in the presence of actuator faults and obstacles. A Qbot2 robot with Kinect vision sensor from Quanser is used for real-time demonstration of the effectiveness of the fault-tolerant system.

Contents

List of Figures

Acknowledgment

I would like to take this opportunity to thank my advisor, Dr. Frank who has been so supportive and helpful through out my entire Master's degree. I can't even begin to say how grateful I am to Dr. Frank. Thank you.

Secondly, I would like to thank my family, my wonderful parents for their love and encouragement they have been giving me at all times. I am blessed to have a supportive and understanding brother who has been with me through my entire journey.

A big thank you to all my friends who have supported me during my thesis. Will always be grateful. Special thanks to my roommates!

Finally, help at all times was extended by the Almighty. All my work is dedicated to Him. Om Sai Ram.

Dedicated to

The Almighty, Dr. Frank and my parents

Introduction

1.1 Literature Survey

Unmanned Ground Vehicles (UGV) are mobile robots whose movements are restricted to the ground and as the name suggests, without any human on board. The applications of UGVs are numerous when combined with algorithms for path planning, obstacle avoidance etc in various fields like military [1], civil [2], agriculture [3], film industry to name a few. These applications require a robust model of the system which can effectively overcome any undesirable effects and achieve the assigned task. Many research groups have developed various algorithms for path planning and obstacle avoidance [4]. Further research on these applications and algorithms has increasingly attracted attention [5].

Mapping of the environment is the primary step towards achieving a fully autonomous system. Along with mapping, the location of the robot in the map should be known. Simultaneous Localization and Mapping algorithms (SLAM) are developed very commonly to generate the map of the environment online, updating the map and making use of this map to locate the position of the vehicle. Different approaches for localization are explored like odometric localization, particle filtering [6], monte carlo localization [7]. For known environments, GPS is used to localize the position of the robot in the map [8]. In GPS denied environments, it is necessary to have onboard sensors like, the laser range finders [9], kinect depth sensor [10], or monocular camera sensors [11] for mapping. Good odometric data is necessary to know how well the robot can estimate its position in the map [12].

Different approaches are mentioned in the literature for path planning [13], including the local planning method which is a dynamic planning method and the global planning method which is a static planning approach [14]. These methods may not be available in events where sudden changes happen in the trajectory of the mobile robot. To overcome some of these limitations, [15] has defined a new approach called the Near shortest path for Mobile Robot which has minimum consumption of energy and is faster. Various algorithms are used in literature for obstacle avoidance as in [16] [17].

However, all the above mentioned algorithms in literature assume the absence of faults in the system components. Faults in the systems are unpredictable discrepancies that can have negative effects on the operations of the system. The self- driving ability of the robot reduces human effort in making decisions and helps in performing tasks autonomously, thus increasing the efficiency of developing more applications. With increased autonomy comes increased responsibility in designing a reliable system. Lack of human intervention makes it a challenging task to accomplish. Because of this, UGVs are more prone to accidents and faults which could be due to various factors. Specifically, in literature, the faults that are considered are the sensor, actuator, and the process faults. These autonomous vehicles need to operate reliably at all times even during the presence of faults. Thus, implementation of a reliable fault diagnosis and accommodation algorithm marks a crucial step in the development of unmanned vehicles. Since it is unmanned, it is hard to manage the faults at all times by human intervention. In order to tolerate faults, a robust fault-tolerant system needs to be designed [18].

Three steps are involved in the fault diagnostics task: Fault detection to detect the presence of faults in the system, fault estimation to estimate the magnitude of fault, and fault accommodation to accommodate to enable successful navigation of the robot to the target in the presence of the faults [19].

1.2 Research Motivation

This thesis is motivated to implement a real-time mapping, path planning, and obstacle avoidance algorithm and test the working of these algorithms on an unmanned autonomous ground vehicle. A robot called QBot2 developed by Quanser is used for the experiments in this lab. In the previously developed design model by Quanser, the algorithms were validated only for static obstacles. The robot was manually made to move around the room and all the obstacles to map the environment. By doing so, a 2D map was generated. This generated 2D map was stored as an image file and was later used to implement other algorithms. Hence, mapping was done only once and the processing was done offline. This saved image was fed as the input to an algorithm which helped in pre-processing the image file and converting it into a binary image. A different program containing the path planning algorithm took this binary image as the input and produced the trajectory from the defined start point to the target. The generated path was fed as an input to a motion control model which moved the robot accordingly. This comprised of running three separate programs which was a tedious process and a lot of steps to follow before we could actually get the robot to move, which was time consuming. Also, the effectiveness of the vehicle could not be validated in real-time. Since, it involves storing the map as an image and then processing it, the path planning is therefore, done offline. In such a case, the map is a static one and only those obstacles which are present during the generation of the map can be avoided, and any new obstacle that could arise at a later stage is not detected. This contradicts the term autonomous. The robot is dependent on the map which is created beforehand and thus, is restricted to movement only in a known environment.

Based on these limitations, this dissertation was motivated to extend the design to an extent where the robot could independently map the environment, generate the path, and move along the way points with minimal interaction by humans, thereby, justifying the term 'autonomous'. Specifically, this research is aimed at implementing all the algorithms such that the mapping, path planning, and motion control algorithms are done online and simultaneously with each other until the robot reaches the target. This way, improving from the previous design, the robot can also be made to avoid dynamic obstacles. The algorithm is no longer constrained to static obstacles.

In order to increase the reliability of systems in unknown environments, development of a reliable system which can operate at all times, even in the presence of faults and uncertainties is necessary. This motivated in designing and implementing a fault diagnosis algorithm which detects the presence of fault, estimates the unknown fault, and accommodates the fault. An actuator fault which is caused due to the difference between the input wheel command and the nominal wheel command is considered. The fault considered is a constant bias actuator fault, which could occur due to slippage of the wheel.

Finally, all these components are integrated using MATLAB/Simulink and are implemented on the experimental setup to demonstrate the effectiveness of the fault-tolerant system. The algorithms are validated on a real time test environment to show the effectiveness of the fault-tolerant system.

1.3 Thesis Organization

This research is documented in the following order:

- **Chapter 2:** This chapter explains the modeling of the ground robot, its forward and inverse kinematics, and also describes the experimental setup used in this research.

- **Chapter 3:** This chapter describes the implementation of a robust fault diagnosis algorithm. The fault detection and fault estimation schemes are also described in this chapter.

- **Chapter 4:** This chapter provides an insight about mapping the environment, the A star algorithm for planning a path around the obstacle, and the motion of the robot along the trajectory. Integration of these components with the fault diagnosis module to achieve a fault-tolerant system is also explained in this chapter.

- **Chapter 5:** This chapter shows the real-time experimental results validating the effectiveness of the fault-tolerant system.
- **Chapter 6:** Some concluding remarks along with some future research directions are included in this chapter.

Mobile Robot Model and Experimental Setup

2.1 Mobile Robot Dynamic Model

The ground robot runs on a differential drive mechanism [20], i.e., it has two wheels which can be controlled and are placed on a common axis. The wheels receive control signals through the DC motors. A relation is developed between the motion of the robot and the motion of the individual wheels of the robot. To obtain such a relation, the motion of the robot about a point is considered. Let r be the radius of each wheel and the rotational velocity for the left and the right wheels be ω_l and ω_r, respectively. The left and the right wheel velocities can thus be written respectively as:

$$v_L = \omega_L r \tag{2.1}$$

$$v_R = \omega_R r \tag{2.2}$$

By varying the relative speeds between the wheels, the trajectory of the robot can be decided. Based on this, the robot can be made to move in a straight line, in a curved path, or spin about a point. Rotation of both the wheels at the same velocity will cause the robot to rotate about a point at a certain distance. This point around which the robot rotates is called the Instantaneous Center of Curvature. Let the distance from the center of the robot which is the mid-point between the axis connecting both the wheels to the Instantaneous Center of Curvature be r_C. Let the distance between the two wheels be l.

Let the robot's heading direction be ψ and the velocity of the robot chassis be V_c. The motion of the robot can thus be written as:

$$v_C = \dot{\psi} r_C. \tag{2.3}$$

The left and the right wheel velocities can also be defined in terms of the Instantaneous Center of Curvature as follows:

$$v_L = \dot{\psi}(r_C - \frac{l}{2}) \tag{2.4}$$

$$v_R = \dot{\psi}(r_C + \frac{l}{2}). \tag{2.5}$$

The motion of the left and right wheels can be related to the motion of the robot chassis Given the distance from the robot to the Instantaneous Center of Curvature, r_C, the angular velocity(i.e., $\omega_C = \dot{\psi}$), the motion of the left and right wheels can be related to the motion of the robot chassis as follows:

$$v_C = \frac{v_R + v_L}{2} \tag{2.6}$$

$$\omega_C = \dot{\psi} = \frac{v_R - v_L}{l} \tag{2.7}$$

$$r_C = \frac{l}{2} \frac{v_R + v_L}{v_R - v_L}. \tag{2.8}$$

2.2 Forward and Inverse Kinematics

Kinematics is a branch of mechanics that deals with the motion of objects. The forward and inverse kinematics are used to determine the linear and rotational velocities of the robot or to determine the commands sent to the left and right wheels such that the robot follows a given trajectory.

2.2.1 Forward Kinematics

By making use of the Forward kinematics, the speed of the robot and the angular velocity can be found based on the left and right wheel speeds. It is governed by the following equations:

$$v_C = \frac{1}{2}(v_R + v_L) \tag{2.9}$$

$$\omega_C = \dot{\psi} = \frac{1}{l}(v_R - v_L), \tag{2.10}$$

where, l is the distance between the two wheels, v_C is the velocity of the robot chassis, ω_C is the angular velocity, v_R and v_L are the right and left wheel velocities, respectively. The velocity of the robot is expressed in the body frame and not in the global frame which is used for the map of the environment. For the coordinate frame transformation, a rotation matrix as shown below is used:

$$\text{R} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix transformation is used to represent the motion of the robot which is in the local frame to the motion in the global frame. The inverse of this rotation matrix exists and is given as follows:

$$R^{-1} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The state vector, X is defined in terms of the states, x and y(positions) and the heading angle, ψ as follows.

$$\text{X}=\begin{bmatrix} x \\ y \\ \psi \end{bmatrix}.$$

$$\dot{X}=\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} V_c \cos\psi \\ V_c \sin\psi \\ \omega_c \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(v_R + v_L)\cos\psi \\ \frac{1}{2}(v_R + v_L)\sin\psi \\ \omega_c \end{bmatrix}.$$

The heading angle becomes zero when the robot's forward direction aligns with the x-axis in the global frame. From the above equation we can thus compute the linear velocity of the robot given the heading angle and the speed of the wheels. This describes the forward kinematics of the robot.

2.2.2 Inverse Kinematics

For the robot to follow a specified trajectory or to move at a desired speed, appropriate commands to the wheels should be given. The inverse kinematics helps in manipulating the appropriate wheel velocities, given the robot's speed and the angular velocity, ω_c. The equations given below are used to calculate the right and left wheel velocities, respectively:

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} v_C + \frac{1}{2} l \omega_C \\ v_C - \frac{1}{2} l \omega_C \end{bmatrix}.$$

These equations describe the inverse kinematics of the robot.

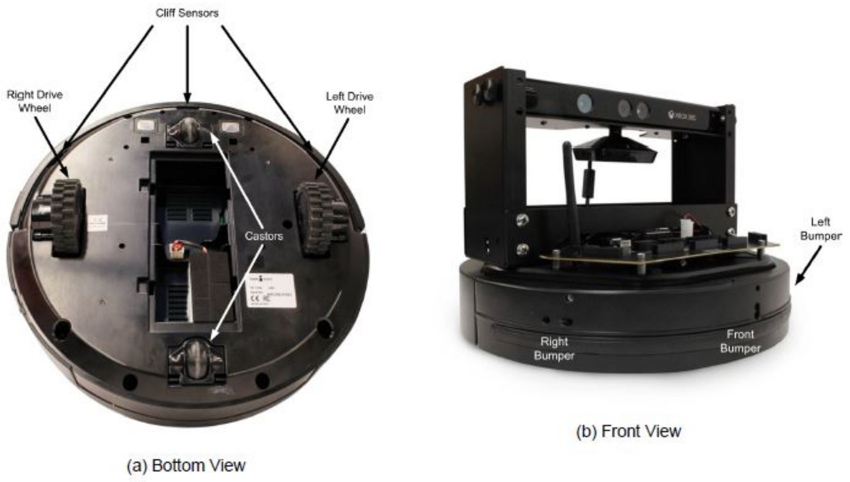

Figure 2.1: QBot2

2.3 Experimental Setup

A general setup of the robot which was used during the research is as described below. The ground robot used for this research is called the QBot2 provided by Quanser. The Qbot has two wheels mounted on the same axis. This is based on the differential drive configuration. Two other wheels, called the castor wheels are used in the front and back of the robot to stabilize the platform. This does not affect the movement of the robot. This differential drive configuration of the robot is used quite often as it is simple to maneuver. The QBot is integrated with an on-board computer called Gumstix DuoVero Zephyr controller. WiFi is setup to communicate with the controller. The QBot has the Data Acquisition Card (DAQ) to measure the sensors on board. The Qbot also has two high resolution encoders on the wheels to track their rotation. Along with the encoders, it is embedded with other sensors like the IMU which has the three axis gyroscope to track the rotation angle and the angular rates. It also is equipped with cliff sensors, bumpers and wheel drop sensors. The robot's front and bottom view are shown in figures 2.1a 2.1b, respectively.

The main type of sensor that is used for the research is the Kinect Computer Vision

Sensor. The Qbot is equipped with a Microsoft Kinect sensor (XBOX 360) which is an RGB depth sensor. It uses an infrared laser and a monochrome sensor for measuring the depth, and it has an RGB camera to process the image. The image and the depth capture occurs at a rate of 30 frames per second and has a resolution of 640x480. The range of the depth sensor varies between 0.5 m to 6 m. It has a 57 degree Horizontal Field of View (FOV) and 43 degrees vertical FOV. The kinect sensor can be tilted vertically up to 21.5 degrees to measure objects outside the horizontal FOV. It is as shown in Figure 2.2.

The control algorithm is implemented using MATLAB/Simulink. A software called QUARC developed by Quanser is used. This software acts as an interface between the software and the hardware. The code written in MATLAB/Simulink is converted into C++ and helps the Simulink block diagram to be run in real time. The entire experimental setup of the system is as shown in Figure 2.3. The controller design is implemented on the host computer in Matlab/Simulink. The code is sent to the target which is the robot with the micro controller on it. The communication between the host and the target is done via WiFi.

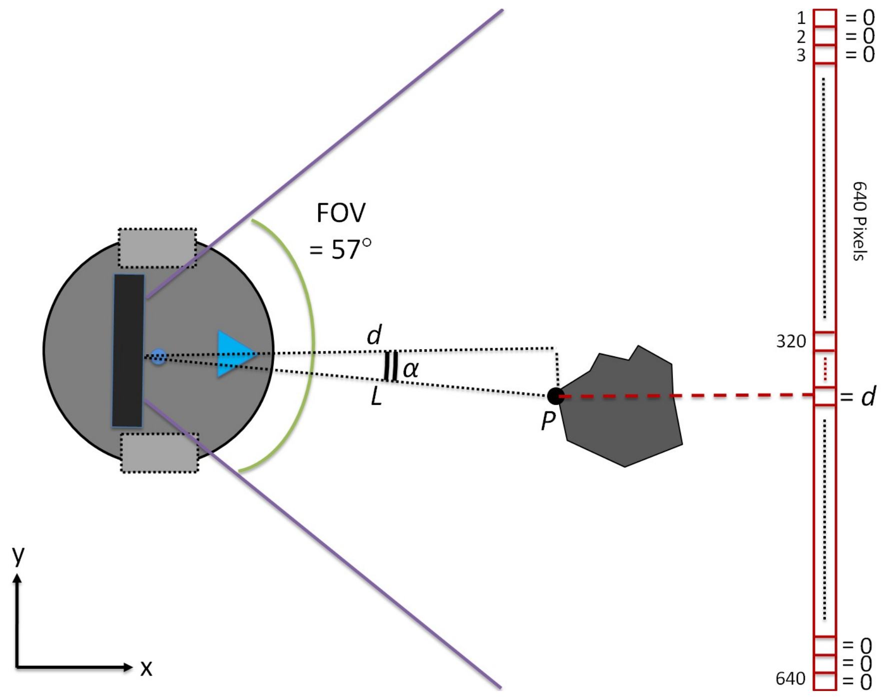

Figure 2.2: Depth Data from the Kinect sensor [21]

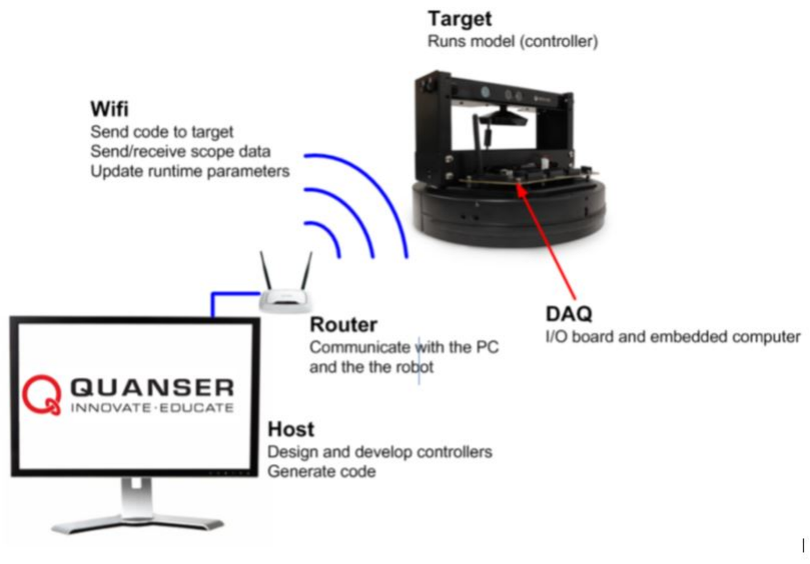

Figure 2.3: Experimental Setup [21]

Fault Diagnosis and Estimation

Faults in systems are unpredictable changes which causes discrepancies in the normal behavior of the system. These effects occurring in the mobile robots are undesirable, yet it is something that is bound to happen with complicated systems. The faults can be either due to external factors or due to internal factors. These faults can occur due to various reasons, for example, loss of effectiveness due to a wheel breakage, or there could be faults in measurements from the sensors. Such faults could have negative effects on the system performance and lead to catastrophic situations. Thus, before leading to such situations, the effect of the fault on the system must be accommodated. Robust and reliable fault-tolerant systems must be developed in order to eliminate the undesirable effects. Fault-tolerant system developed in this thesis comprises of three steps:

- Fault Detection
- Fault Estimation
- Fault Accommodation

3.1 Fault Detection

In the fault detection step, which is the first step towards fault-tolerant control, the performance of the robot is monitored to detect the occurrence of faults in the system. Below, we describe the fault detection algorithm [19].

The system dynamics are given by:

$$\dot{X} = \phi(X,u) + \beta(t - T_0)f(X,u) + \eta(X,u,t) \tag{3.1}$$

where, X= $\begin{bmatrix} x \\ y \end{bmatrix}$ is the measurable state vector, u is the control input, ϕ is the nominal system dynamics represented as ϕ= $\begin{bmatrix} V_c \cos\psi \\ V_c \sin\psi \end{bmatrix}$, β $(t - T_0)$ is the time profile of the fault occurring at some unknown time T_0 , which is assumed to be a step function in this research, $\eta(X,u,t)$ is the unknown modeling uncertainty, and $f(X,u)$ is the unknown fault function.

3.1.1 Fault Detection Algorithm

Based on the series-parallel adaptive estimation algorithm [22], the State estimator is designed as:

$$\dot{\hat{X}} = -a_m(\hat{X} - X) + \phi(X,u) \tag{3.2}$$

where $a_m > 0$ is a design constant, $\hat{X}$ is the estimate for the state vector X and the state estimation error, $\tilde{X}$ is given by

$$\tilde{X} = X - \hat{X}. \tag{3.3}$$

Before the occurrence of fault (i.e., for t $< T_0$), $\beta(t - T_0)f(X,u)$ = 0,

the state estimation error dynamics are as follows:

$$\dot{\tilde{X}} = -a_m(X - \hat{X}) + \eta(X,u,t) \tag{3.4}$$

$$\dot{\tilde{X}} = -a_m\tilde{X} + \eta(X,u,t). \tag{3.5}$$

3.1.2 Derivation of Threshold

Solving the above error dynamics, we obtain:

$$\tilde{X}(t) = e^{-a_m(t-t_0)}\tilde{X}(0) + \int e^{-a_m(t-\tau)}\eta(X(\tau),(\tau),\tau)d\tau \tag{3.6}$$

From the triangular law of inequality, the above equation can be written as:

$$|\tilde{X}(t)| \leq e^{-a_m(t-t_0)}\tilde{X}(0) + |\int_0^t e^{-a_m(t-\tau)}\eta(X,u,\tau)d\tau|. \tag{3.7}$$

By choosing the initial condition of the state estimator as $\hat{X}(0) = X(0)(i.e., \tilde{X}(0) = 0)$, the equation can be reduced as follows:

$$|\tilde{X}| \leq \int e^{-a_m(t-\tau)}|\eta(X(\tau),u(\tau),\tau)|d\tau$$

The fault detection threshold can be thus chosen as follows:

$$V(t) = \int e^{-a_m(t-\tau)}|\bar{\eta}(X(\tau),u(\tau),\tau)|d\tau \tag{3.8}$$

where, V(t) is the threshold for fault detection,. $\bar{\eta}$ is a known bounding function on the modeling uncertainty (i.e., $|\eta| \leq \bar{\eta}$). By the above design, it is guaranteed that the residual $\tilde{X}$ remains below the threshold before fault occurrence (i.e.,$|\tilde{X}(t)| \leq V(t)$ for $t < T_0$). On the other hand, if the residue, $\tilde{X}$ exceeds the above described threshold, it indicates the presence of a fault.

3.2 Fault Estimation

At this point, the robot loses track of the trajectory it is supposed to follow and starts moving randomly colliding with the obstacles. As soon as the fault occurs, the fault detection algorithm shoots the residue above the threshold indicating the presence of the fault. Once this behavior is noticed, the fault estimation algorithm is used to get an optimal estimate of the fault.

3.2.1 Fault Estimation Algorithm

After the occurrence of fault (i.e., for $t > T_0$), the system dynamics is given as follows:

$$\dot{X} = \phi(X) + \eta(X, u, t) + f(X, u). \tag{3.9}$$

The presence of faults in the system can be represented by the fault function $f(x)$, which represents the change in the system dynamics due to the presence of fault. The fault function considered in this thesis is assumed to be an abrupt fault with an unknown constant bias $\theta \in \Re$ which occurs to the left and right driving motors/wheels. Note that if θ is 0, then the system is a healthy system. Whereas, a nonzero value of θ represents a faulty system.

The nominal system dynamics in the absence of faults are represented as:

$$\dot{X} = \begin{bmatrix} V_c^0 \cos\psi \\ V_c^0 \sin\psi \end{bmatrix},$$

where V_c^0 is the nominal robot chassis velocity. In the presence of a bias fault in the left and right actuators, we have

$$V_c = (v_R + \theta + v_L + \theta)/2 = V_c^0 + \theta \tag{3.10}$$

Thus, the system dynamics can be represented as follows:

$$\dot{X} = \begin{bmatrix} (V_c^0 + \theta)\cos\psi \\ (V_c^0 + \theta)\sin\psi \end{bmatrix} + \eta(X, u, t) = \begin{bmatrix} V_c^0 \cos\psi + \theta\cos\psi \\ V_c^0 \sin\psi + \theta\sin\psi \end{bmatrix} + \eta(X, u, t)$$

Thus, the fault function $f(x, u)$ in (3.9) is $\begin{bmatrix} \theta \cos\psi \\ \theta \sin\psi \end{bmatrix}$

Based on the adaptive parameter estimation algorithm,

$$\dot{\hat{X}} = -\lambda(\hat{X} - X) + \phi(X) + \hat{\theta}b \tag{3.11}$$

Where $\hat{X}$ represents the state estimate, $\lambda > 0$ is the learning rate, $\hat{\theta}$ is the estimate of the unknown fault magnitude, and

$$b = \begin{bmatrix} \cos\psi \\ \sin\psi \end{bmatrix}$$

The adaptive law for parameter estimation can be obtained as follows [22]:

$$\dot{\hat{\theta}} = \gamma \epsilon b \tag{3.12}$$

where, $\epsilon = x - \hat{x}$ is the state estimation error, and γ is the learning rate.

Thus, the estimated fault parameter $\hat{\theta}$ can be used for fault accommodation.

Mapping, localization, Path Planning and Motion planning

The overall architecture of the system is as shown in figure 4.1

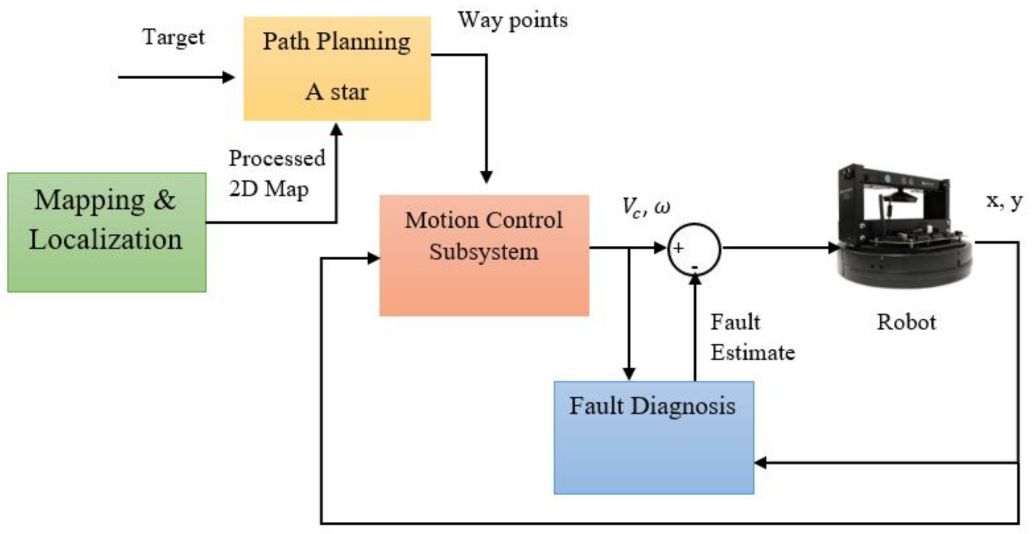

Figure 4.1: System Architecture

4.1 Mapping

It is necessary for the robot to know its environment before making a decision about its movement towards the goal. Thus, mapping is an essential step for autonomous vehicles. In order to map the environment, various sensors can be made use of. In this research, for mapping the environment, the depth information from the Microsoft Kinect sensor is used, and to locate the position of the robot in the environment, the robot's position and orientation information is used [23]. When these two are combined together, mapping of the environment can be done autonomously. The resolution of the Kinect sensor is 480x640 i.e., the depth data is represented as a 480x640 image. Based on the dimension of mapping i.e., either 2D or 3D, the number of rows of the depth data are chosen accordingly. For a 2D mapping, only one row is used. In order to do the 3D mapping, the same algorithm for the 2D mapping can be used, but for all the rows of the depth data [24]. Utilizing the other rows of the depth data will also help in locating the obstacles which are higher than the robot's horizon. In order to map a 360 degree environment around the robot, the robot should be moved accordingly.

Considering one row of the depth data for 2D mapping, each row has 640 pixels. The distance from the kinect sensor to the obstacle is stored in each pixel which is in millimeters. Any object which is at a distance less than 0.5m from the robot gives an invalid data, and such values are made zero in the software. Thus, in order to receive legit values, the robot should be moved accordingly. Also, implementing other types of sensors like laser sensors could resolve this issue.

By looking at Figure 2.2, it is considered that the point that needs to be mapped is at the 400th pixel. The distance measure obtained from the kinect sensor of this pixel is represented as 'd', where 'd' is the distance from the camera to the obstacle. To get the y coordinate, the angle, α is needed, and it is represented as P_y = $L\sin(\alpha) = d\tan\alpha$ and $P_x = d$ mm. This angle α is calculated based on the distance between the desired point and the center of the chosen row (320th pixel), and the horizontal Field of View (57^0) of the sensor as given below:

$$\alpha = (400 - 320)\frac{57}{640} \tag{4.1}$$

Thus, the position P can be written as, P=(P_x, P_y)= $(d, d\tan\alpha)$. P can be mapped in the global frame if the pose (x, y, θ) of the robot is known. The pose of the robot initially is set to zero. Thus, the point where the robot is placed initially, becomes the origin of the map. The location of the robot, its pose, and orientation are tracked using the particle filtering algorithm given the sensor information from the kinect. As the robot moves around, each point is mapped into the global frame. The mapped points are used to create an occupancy grid of the map. The map gets updated in real time. Thus, mapping is done dynamically. The areas which are in black represents the obstacles. Areas of the map in gray are those parts of the environment which are not explored. These are the unknown areas. The parts of the map which are in white are the explored and obstacle free areas where the robot is free to move about.

4.2 Localization

The location of the robot, its pose and orientation in the map are determined using the particle filter algorithm [25]. In this algorithm, several random variables/particles are initialized. Each variable or particle represents a copy of the robot. And each variable is associated with a weight which represents the precision of the particle's location. Using the average weighted sum of all the variables, an estimate of the particle of interest can be generated. This algorithm involves two steps:

- Prediction: In this step, according to the existing data, the location of each variable is predicted.
- Update: Based on the latest information from the sensor, the weights of the particles

are updated. The particles with the least weights are removed and the one with the highest weight is used to give the current location of the robot. If the particle's information is closer to the sensor information, then more weight is given to that particle.

As the robot keeps moving, these particles keep initializing to new locations and the prediction and updating process keeps continuing. The locations converge as the weights keep adjusting. An example of the mapped environment with the location of the robot in the map is shown in Figure 4.2. As seen from the figure, the gray areas are unexplored. The black fields are the obstacles and the white areas are free spaces. The gray circle in the map represents the location of the robot.

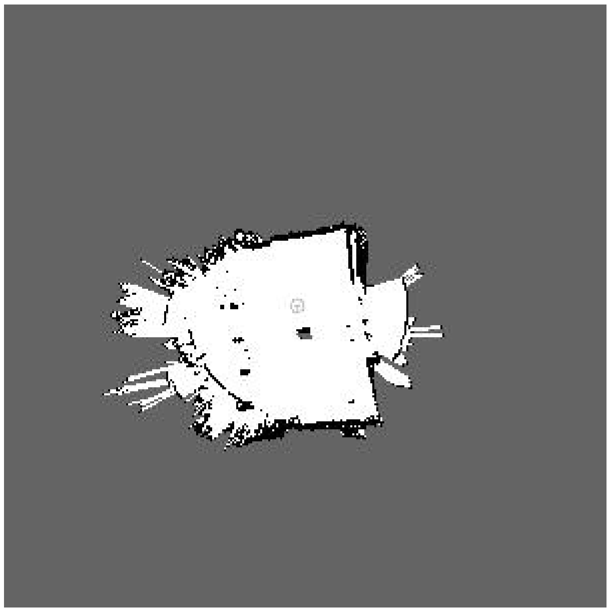

Figure 4.2: 2D Map

4.3 Path Planning

Path planning is a primitive function of any unmanned vehicle. In various applications like cleaning robots, gaming, path planning is used to get the optimal path between two points. Several path planning algorithms exist such as VGraph, Djikstra's algorithm, Potential Field, A Star, Coverage path planning, to name a few. The algorithm that has been used in this research is the A star algorithm which is explained next.

4.3.1 A Star

The A^* algorithm is one of the most commonly used path planning algorithms because of its simplicity and speed [26]. In A^*, the algorithm sees the map as a 2D occupancy grid. Each grid is assumed to be a node. From the robot's current position, all the cells surrounding it, i.e., the neighbouring cells are checked for the presence of an obstacle and for the cost. The optimal node will be chosen for the next step of the robot. In this way, the algorithm is run until the robot reaches the last node or the target [27].

While creating the occupancy grid, the information regarding the cells' occupancy is provided. The knowledge of the start and the end point is needed which is provided by the user. Based on this data, the cells are checked to find the optimal next cell and is added to the list of free and optimal cells. A heuristic approach is used for the A^* algorithm. Initially, the distance (cost) between the current node and the goal node is calculated. Then, it calculates all the neighboring nodes which are free (White spaces) and which are not visited beforehand for the heuristic cost i.e, the expected cost from all the neighboring cells to the target. This heuristic cost is represented as $h(n)$. The cost it needs to travel from the current node to all the adjacent cells is also calculated. This is called the path cost and is represented as $g(n)$. Once, the heuristic and the path cost are found, it calculates the entire cost from the current cell to the goal cell. This overall cost is represented by $f(n)$ and is given as

$$f(n) = g(n) + h(n). \tag{4.2}$$

For an optimized solution, the total cost, $f(n)$ should be used. Using just the heuristic or just the path cost will not give an optimal path. Thus, the A^* algorithm produces a set of way points from the robot's current position to the target, which is then fed to the motion control model.

4.4 Motion Planning and control

After mapping and path planning, the way points which are produced from the A^* algorithm are fed to the motion control model, which converts the points into a form suitable to command the robot's wheels. The way points that are generated from the A star algorithm are fed one after the other to the motion control model. The motion control model has a proportional controller. The error between the robot's current position and the way point is calculated and fed to the proportional controller as a feedback. This is done to control the position of the robot. The motion planning model outputs the appropriate robot velocity and the angular rate. This is fed to the Inverse kinematic block which converts it to the right and left wheel commands. The motor commands are written using the QUARC HIL Write block. These wheel commands move the robot accordingly and the encoder values of the wheels are read and are then used to get the current position of the robot. Thus, the robot follows the way points as they are produced.

4.5 Obstacle Avoidance

For any application of unmanned systems, the main requirement for it is to follow a specific trajectory to reach the goal destination. As much as the path planning algorithm plays an important role in this, so does the obstacle avoidance. There are two types of obstacles that needs to be avoided: the static obstacle and the dynamic obstacle. The A star algorithm as discussed above generates the shortest path from one point to another point. Along with producing the optimal path, it should also generate a path which is free of obstacles. The robot must become aware of the surroundings to see where the obstacle

lies. For such an application, different sensors can be used. For the purposes of this research, the Microsoft Kinect Sensor is used to get the information about the room around the robot. The Kinect sensor provides the depth data which is converted into a 2D map showing areas in black, gray and white, representing obstacles, unexplored area, and area that is clear of any obstacles, respectively. The white areas are represented as 255 on the gray scale, the black areas as 0, and the unexplored gray areas as 100. The unexplored gray areas could either have obstacles or can be free. Without getting either a 0 or 255 from those areas, the robot considers the gray areas as obstacles. Thus, the 100's are converted into 0's. Now the map is in terms of just two values, either 0 or 255. The data is normalized as 0 being 0 and 255 to be 1. Based on these, before feeding the map to the A star algorithm, the above mentioned conversion of gray areas into 0's is done. The pre-processed map is shown in Figure 4.3. After feeding it to the A star algorithm, each cell around where the robot currently is, will be checked to see if it is a 1 or 0. If the cell is a 0, it is put to a list called CLOSED where lies all the cells which are occupied. Next, the distance from the current position to all the free cells will be calculated, and the optimal path will be developed. This avoids the movement of the robot into occupied areas.
Initially when the robot is turned on, it is made to rotate around its origin. The robot sees a 360 degree environment of its surroundings knowing the static obstacles. It then plans a path from its initial position to the target. The planning is done initially for static obstacles. For static obstacles, no obstacle arises in between the robot and the trajectory once the mapping is done initially. Static obstacles, therefore, can be used only if the environment is known before hand and if it is guaranteed that no obstacle arises in between the robot and the target.

As the robot follows the way points and approaches the target, the depth data of the environment is updated continuously. Thus, path planning is done online the entire time until the robot reaches the goal. If an obstacle arises in between the target and the robot, new path is generated from its current position. The path where the robot has moved is considered to be obstacle free. Such an obstacle avoidance algorithm is called dynamic because the robot sees a dynamic map where an unknown dynamic obstacle arises abruptly. Initially, the mapping is done for static obstacles when the robot has no idea about any obstacle that arises in the future. As the robot approaches the target, since, the mapping is done dynamically, any obstacle that arises in between is mapped and the A star produces suitable way points and these obstacles are avoided. This online process of planning a new path around the obstacle whenever it arises on its way towards the target is called dynamic obstacle avoidance.

4.6 Integration

One of the objectives of this thesis is to integrate the above algorithms for mapping, path planning, obstacle avoidance, and motion control of the robot and to implement the system in real time. At first, mapping was done entirely by moving the robot manually using the keyboard commands. The obtained map was saved as an image and the image was processed using blob analysis to detect the presence of obstacles. This way of mapping gave us information only about the static obstacles. The obtained map was scaled to a smaller dimension. This processed map was fed to the A star algorithm which generated the way points.
The mapping model that was developed for the purposes of this research involves very minimal interaction with the keyboard. The robot is rotated around its center only once, manually. This model uses the depth data obtained from the Kinect sensor, i.e., a depth map of the surrounding is obtained. The mapping is done autonomously. With the help of keyboard interface, the robot is rotated at its origin to initiate the path planning process. This step is required so that the robot knows what lies around it and can make a decision about its next move. Once the robot knows its surroundings, the gray scale map is generated. The generated map was then processed. The map is converted to a binary map

representing 0’s and 1’s i.e, the black and white areas. In order to maintain sufficient distance from the obstacle while traversing, an additional layer (buffer space) is added around the obstacle as shown in Figure 4.3. From the figure, the addition of a layer of buffer around the obstacles can be observed. This ensures that even if the robot gets too close to the obstacle, it won’t collide with it. The location of the robot in the map is also obtained through localization. Thus, the first step was to get the right map which clearly differentiates the obstacles and the free spaces in the map.

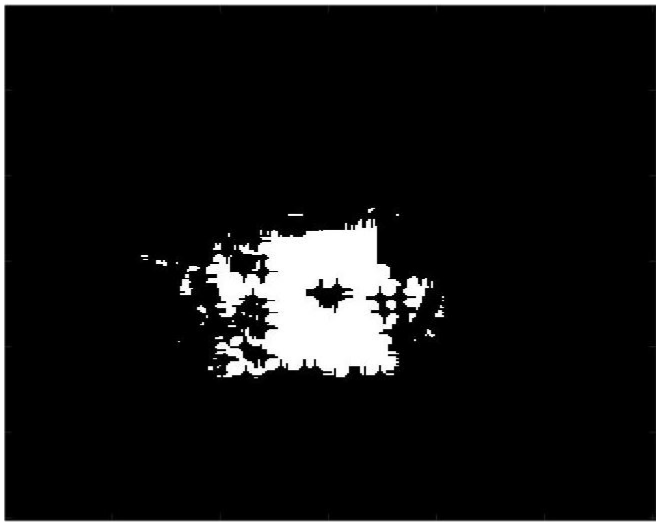

Figure 4.3: Preprocessed Map

In the next step, the processed binary map with buffer around the obstacles is fed to the path planning system block. The path planning block is embedded with the A Star algorithm which makes use of the binary map to produce the optimal path to get to the target. In this step, the origin of the robot and the target is defined. The point where the robot is placed initially is the origin. The A star algorithm produces the way points from the start to the goal. This algorithm is triggered in the beginning after the map is generated. This algorithm is kept running throughout the process. As the robot keeps moving, new way points are generated depending on the presence of obstacles until the target is reached. The way points generated are sent to the next block which is the motion control for the robot.

The motion control subsystem takes the way points as inputs. The motion control subsystem and the path planning subsystem run at the same time. The way points are continuously fed to the motion control model. The way points that are generated are used to control the movement of the robot. Along with the next way point for the robot to move to, the current location of the robot is also given as the inputs. The way points are then compared with the present location of the robot, and the pose of the robot is controlled by using a proportional controller in the feedback loop. The movement of the robot is brought to a halt as soon as the final target is reached. Once the robot stops, it is ensured that the target has reached.

4.7 Fault Tolerant control

Along with planning a path and controlling the movement of the robot, a fault in the form of a constant bias was introduced to the robot. This fault makes the robot gain extra velocity and it affects the trajectory and the robot starts moving in circles colliding with obstacles. Fault detection and estimation algorithms are implemented and the magnitude of fault is estimated. This estimated fault is fed back in a loop and is eliminated. The magnitude of the fault is estimated using the above mentioned fault estimation technique given in equations (3.11) and (3.12) and is represented as $\hat{\theta}$. This estimated fault should be eliminated for the robot to reach the target without losing its control. After fault occurrence, the fault detection algorithm detects the presence of the fault. The fault estimation algorithm gives an estimate of the magnitude of the fault, and, the fault diagnosis information is used to compensate for the effect of the fault. Specifically, the estimated fault magnitude is subtracted from the wheel command. This accommodates the fault in the wheels. Thus, after the fault is accommodated, the robot reaches the target as desired.

Experimental Results

Experimental results validating the algorithms implemented and described in the previous chapters are included in this chapter. The robot's origin is defined as (0,0) and the target chosen to test is straight ahead at a distance of 2.6m from the robot (2.6,0). The performance of the robot is tested for two types of obstacles: Static obstacles which are present during the generation of the map and dynamic obstacles, which indicates the obstacles that arises during the movement of the robot towards the target. These dynamic obstacles are not present when the environment is mapped initially.

5.1 Path planning for Static Obstacles

5.1.1 Absence of Faults

The robot is placed at the origin and is made to spin about its origin. This will produce a map of the environment. While testing for static obstacles, no obstacles are introduced after the generation of the map. This means, once the map and the way points are generated, there are no new obstacles introduced in the way of the robot. The A star algorithm produces the way points i.e., a path to the target, and the robot follows these way points. At this stage, the robot does not encounter any obstacle along its way, and no fault is introduced. Without the presence of either the obstacle or fault, the robot is expected to follow a straight line and reach the target without any deviations. This is represented in Figure 5.1. As seen from the figure, the robot takes a straight path. There is a slight deviation from the straight path. This error is due to odometric localization and closed-loop control error. Additionally, the residue doesn't exceed the fault detection threshold. The presence of residue below the threshold indicates the the absence of fault. This is shown clearly in Figure 5.2

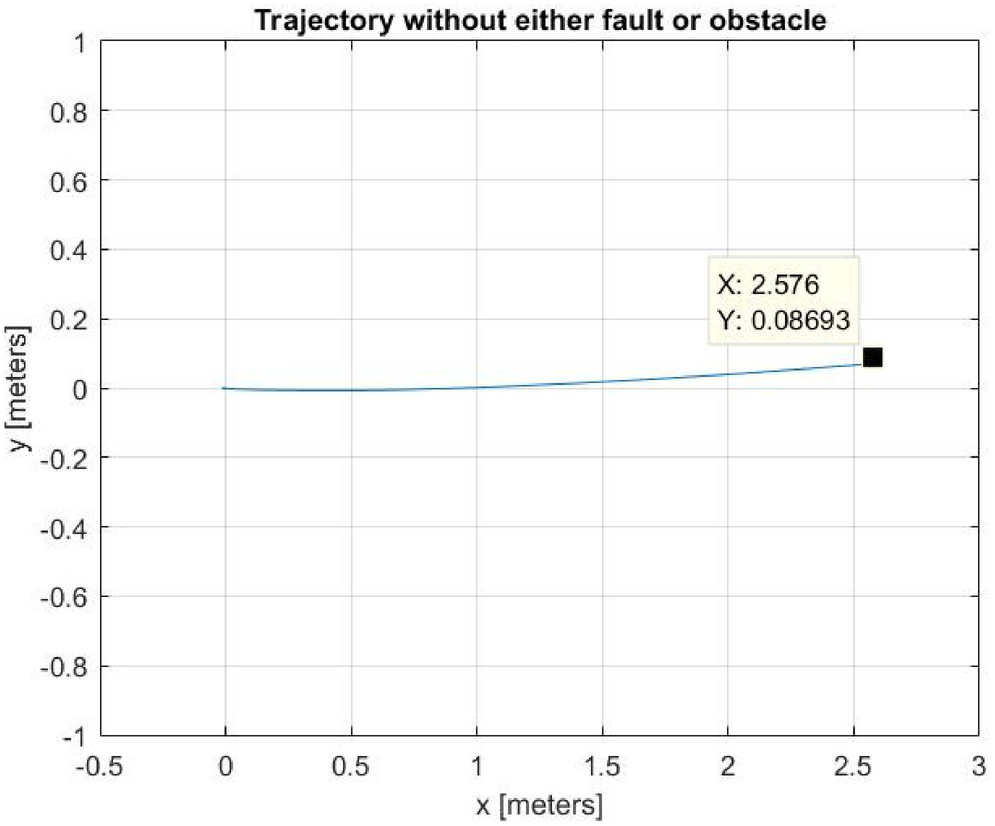

Figure 5.1: Trajectory for static obstacles

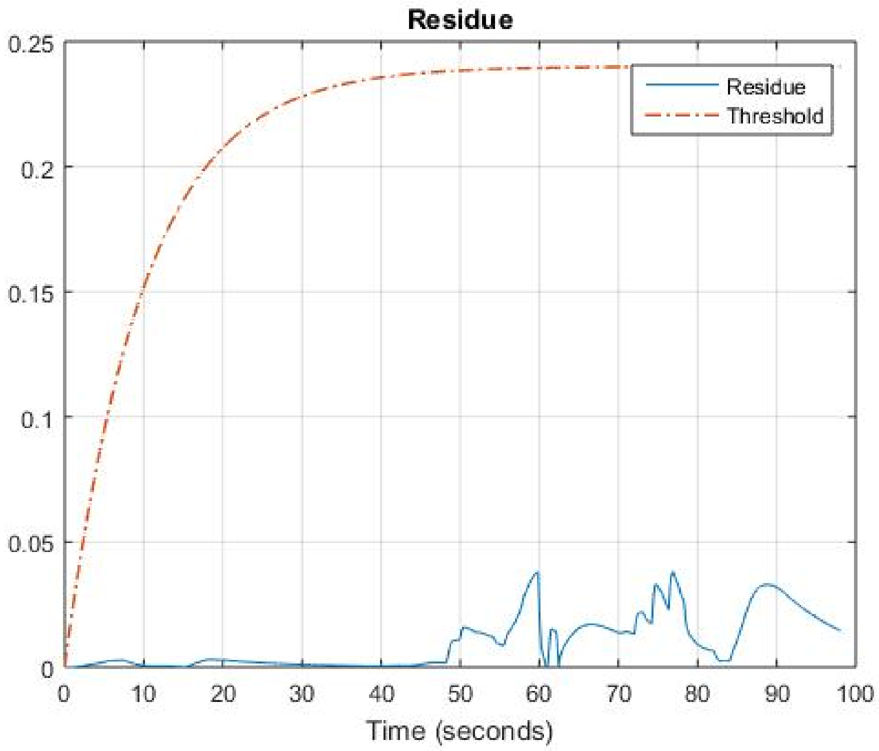

Figure 5.2: Residue before the occurrence of fault

5.1.2 Presence of Faults

No Fault Accommodation

Next, a faulty case is considered. In this, a fault of a constant bias with magnitude 0.2 was introduced to both the wheels at 60 seconds. The presence of fault causes the robot to move in circles, thus not following the desired trajectory. The robot travels along the right trajectory until the fault was introduced. The introduction of fault at 60s causes the robot to mess up the trajectory. Such faults could lead in the robot colliding with the obstacles. This undesirable behavior is shown in Figure 5.3. The robot loses track of the trajectory after the introduction of fault as seen in this figure.

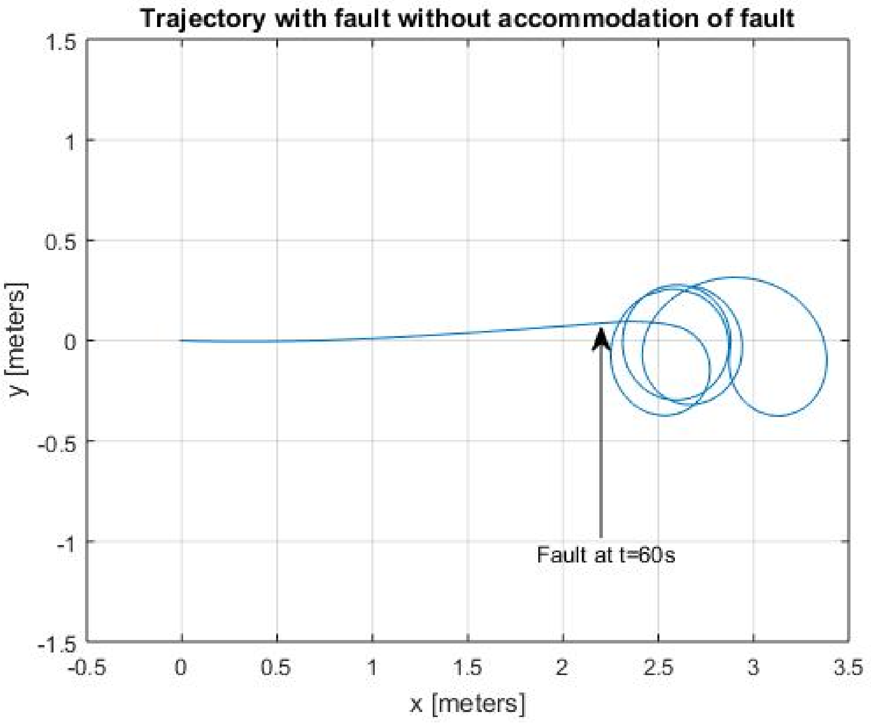

Figure 5.3: Trajectory in the presence of fault without Fault Accommodation

As seen from the fault detection algorithm, the residue should exceed the threshold indicating the presence of fault. From Figure 5.4, this behavior is clearly seen. A fault was introduced at 60s and the residue immediately shot up above the threshold at this instant. This indicates the presence of fault in the system.

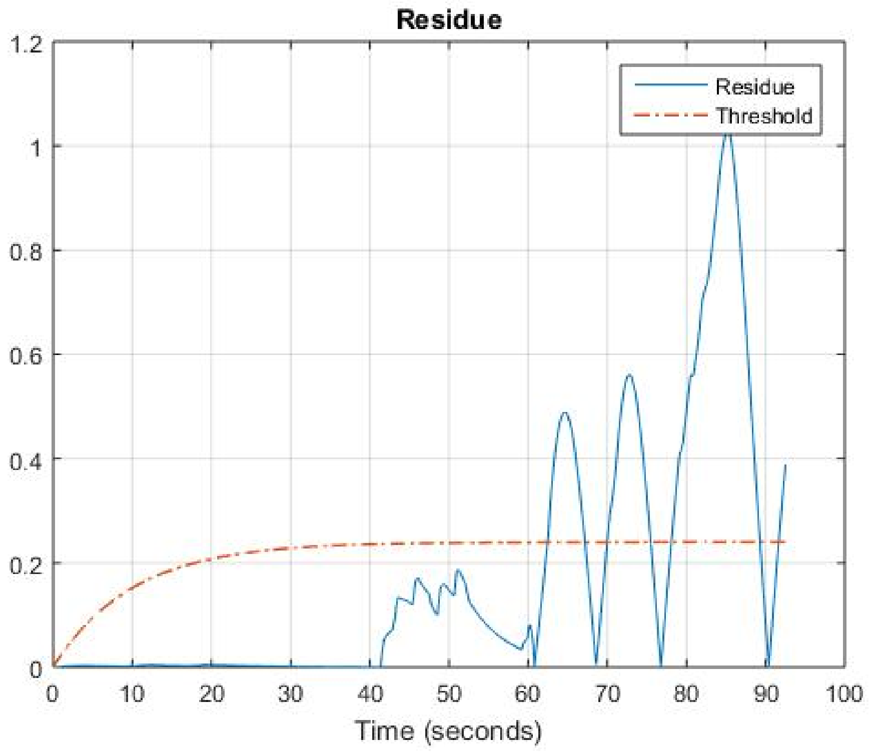

Figure 5.4: Residue after fault occurrence

Fault Estimation and Accommodation

Once the fault is detected, the magnitude of fault is estimated using the algorithm described in Chapter 3. The estimated fault parameter was used for the fault accommodation. This helps the robot to continue in the generated trajectory and reach the target. This response of the robot with fault accommodation is shown in Figure 5.5. The introduction of fault caused the robot to deviate from its trajectory around 1.5m from its origin indicating the presence of fault. This fault was detected and the estimation algorithm was used to estimate the magnitude of the fault parameter. Once it was accommodated, the robot reached the destination as seen from this figure. Since the fault introduced was a constant bias fault of magnitude of 0.2 to both the wheels, the estimated fault magnitude is at 0.2 as seen from Figure 5.6. The fault estimate reaches 0.2 in about 10 seconds.

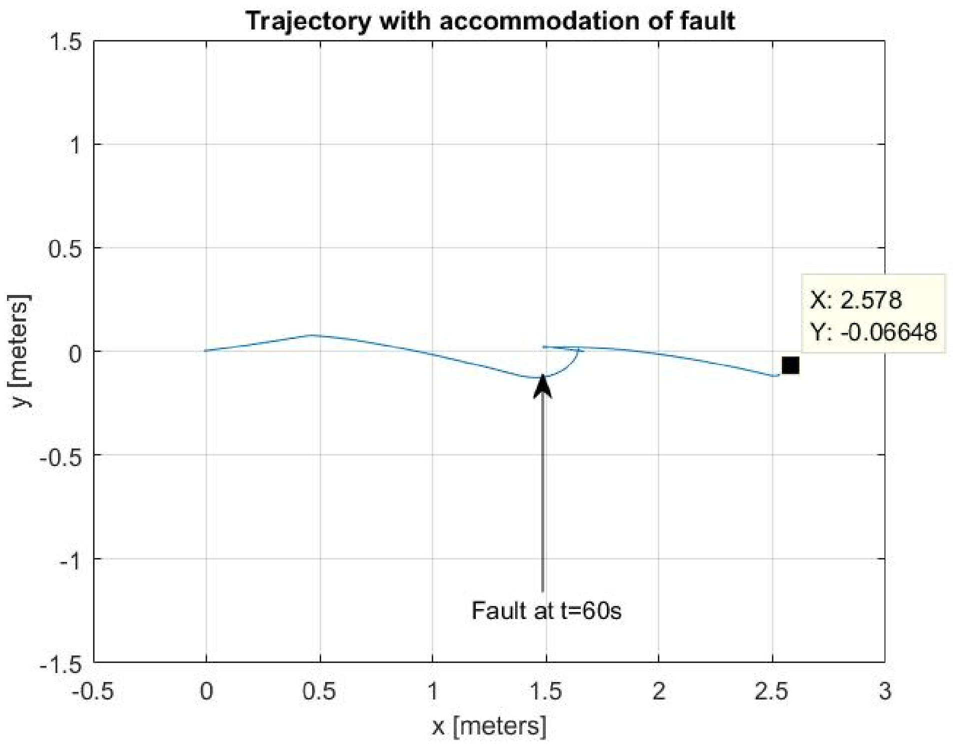

Figure 5.5: Trajectory after fault accommodation

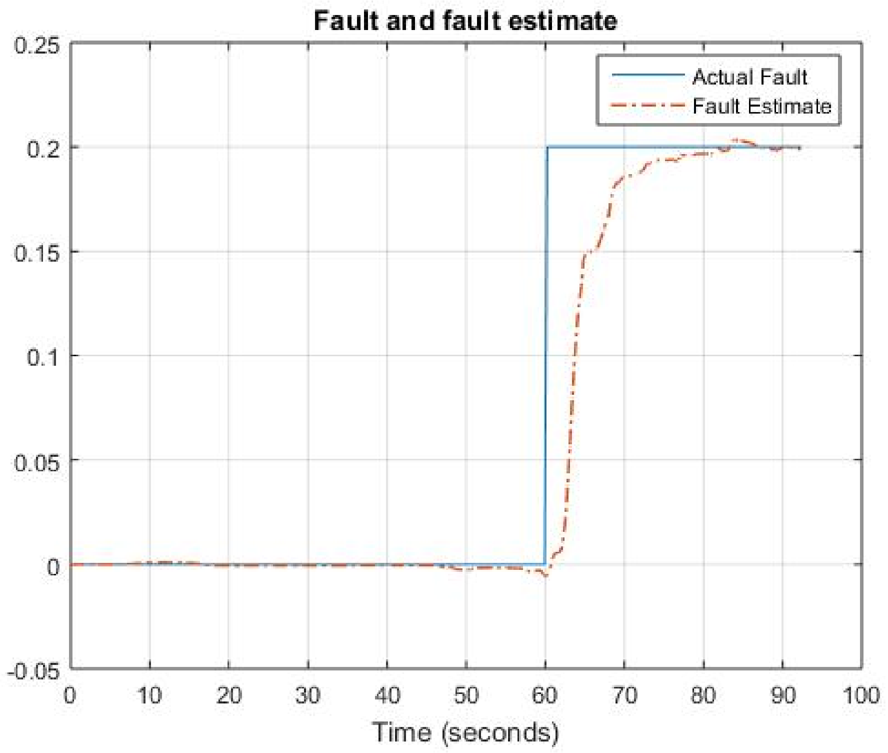

Figure 5.6: Fault Magnitude Estimate

5.2 Path planning for Dynamic Obstacles

5.2.1 Absence of Faults

Results for path planning in the presence of dynamic obstacles are shown in this section. Dynamic obstacle here means, an obstacle appears in the way of the robot while the robot is in motion. These obstacles are not present at the time of generation of the map which is done before enabling the movement of the robot. These obstacles occur at a later instant of time while the robot is in motion. The planning of a path around these obstacles which occur as soon as an obstacle is introduced are shown in the results below.
At first, a case where no fault is present and an obstacle is introduced is considered. Figure 5.7 shows the behavior of the robot in the presence of a dynamic obstacle. The obstacle is represented as a black square in the figures which appears while the robot is approaching the target. As the robot sees a new obstacle in its path while traversing through the generated trajectory, a new path is generated by the A star algorithm. The robot, hence, avoids the obstacle and reaches the desired target. Since no fault is considered for this case, the robot is expected to generate a new path around the obstacle and reach the target as seen in the Figure 5.7. The robot reaches the target which is at a distance of 2.5m from the origin. This deviation of 0.1m from the final target can be due to odometric localization errors or due to closed loop control errors. The absence of fault can be verified from Figure 5.8, where the residue lies below the threshold which indicates the absence of fault.

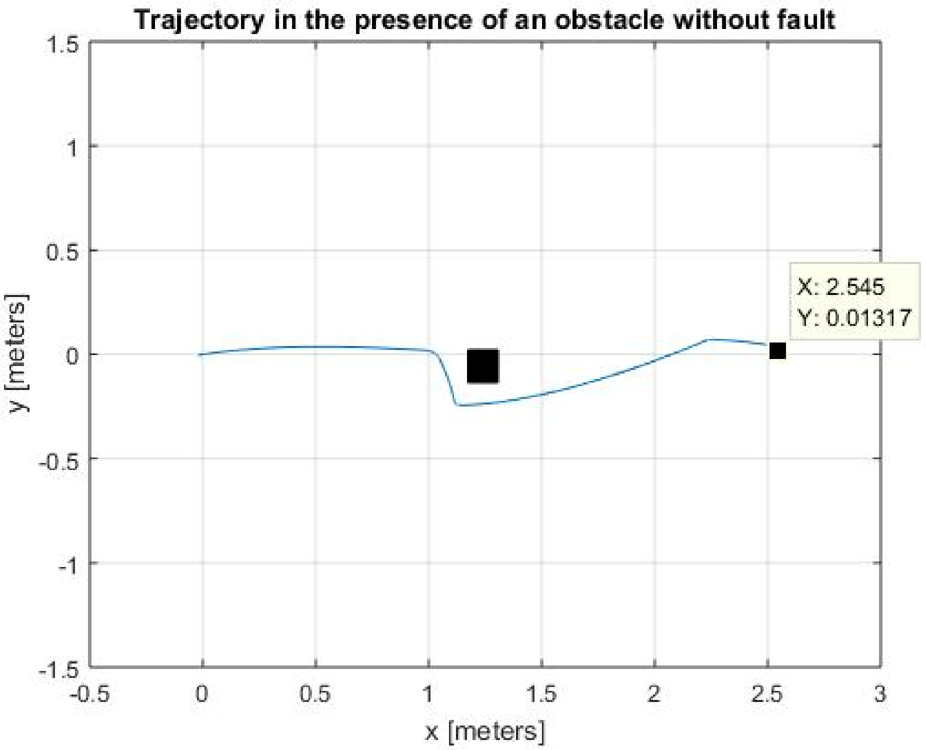

Figure 5.7: Trajectory in the presence of a dynamic obstacle

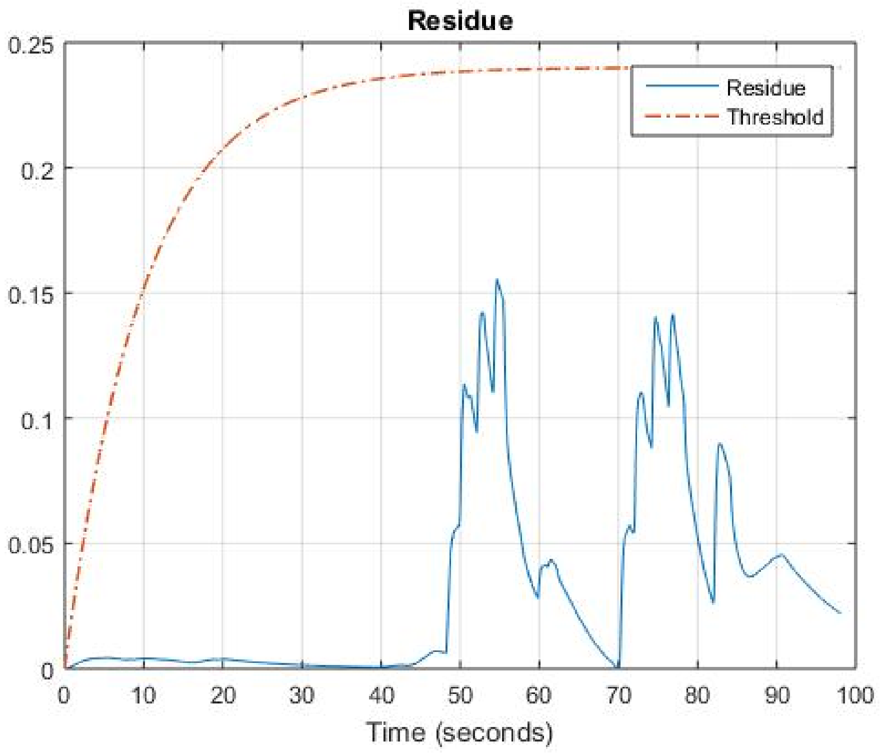

Figure 5.8: Residue before the occurrence of fault

5.2.2 Presence of Faults

No Fault Accommodation

A faulty case is considered next. A fault of a constant bias with magnitude 0.2 was introduced to both the wheels at 60 seconds. The dynamic obstacle was also introduced in the robot's way. The introduction of fault without accommodation, caused the robot to move in circles colliding with the obstacle. Such a case is undesirable. From Figure 5.9, it can be seen that the fault is introduced at 60s at which time the robot has travelled for around 1.3m from the origin. This fault causes the robot to lose track of its trajectory and collides with the obstacle, thus, not reaching the target. The presence of fault can be detected using the fault detection algorithm as shown in Figure 5.10. The fault was added at the 60th second at which point the residue exceeds the threshold immediately.

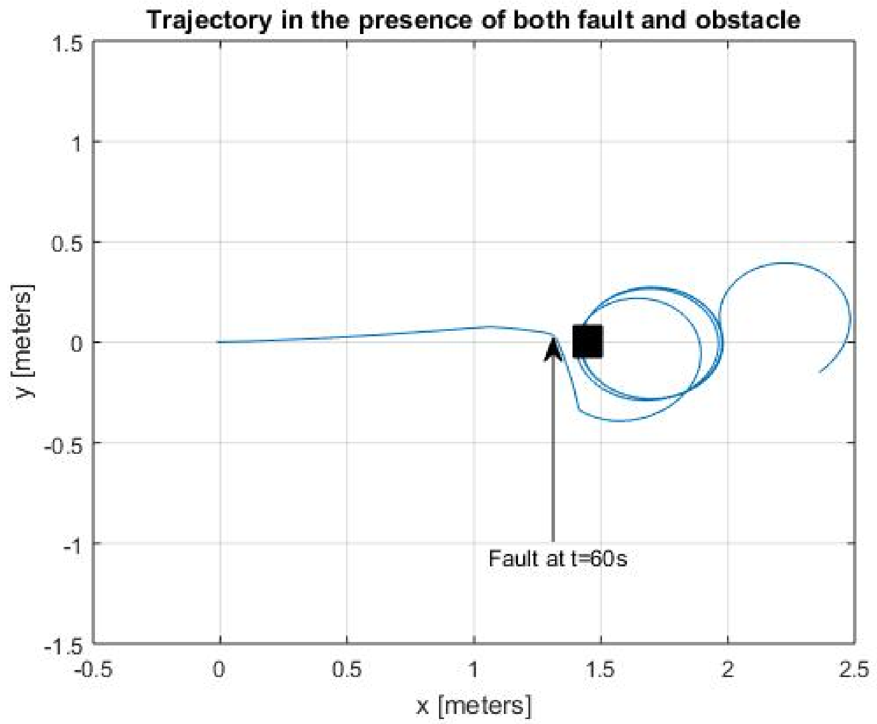

Figure 5.9: Trajectory in the presence of fault and a dynamic obstacle without accommodation of fault

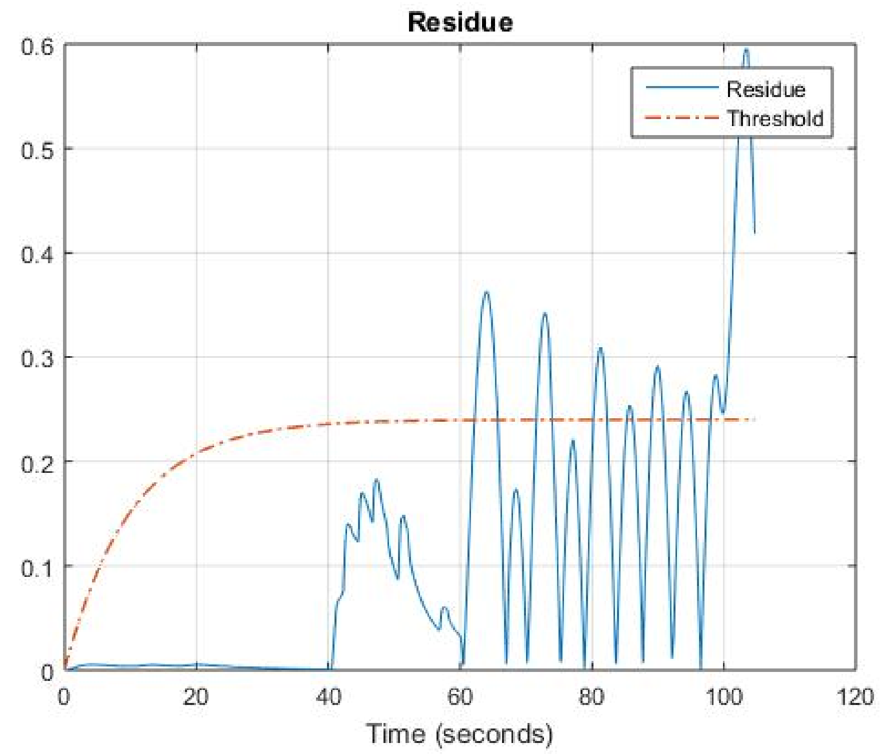

Figure 5.10: Residue after the occurrence of fault

Fault Estimation and Accommodation

After detecting the fault, the fault magnitude is estimated to accommodate the faults in the wheels. As soon as the fault was added at 60 seconds, the robot deviates from its actual trajectory. Estimating the fault magnitude and accommodating it, will mitigate this effect and causes the robot to reach the target successfully even in the presence of both the fault and the obstacle. This is shown in Figure 5.11. Also, the estimation of fault can be seen from Figure 5.12. The fault magnitude is estimated to be about 0.2. The estimate reaches the final value after about 10 seconds. This estimated fault magnitude was used for fault accommodation.

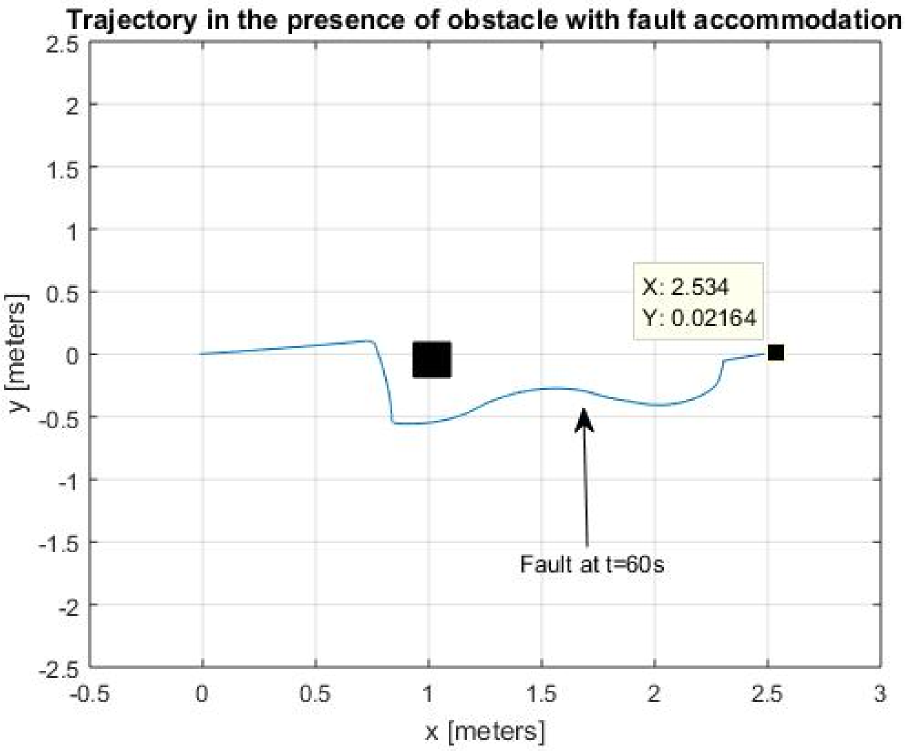

Figure 5.11: Trajectory after fault accommodation in the presence of a dynamic obstacle

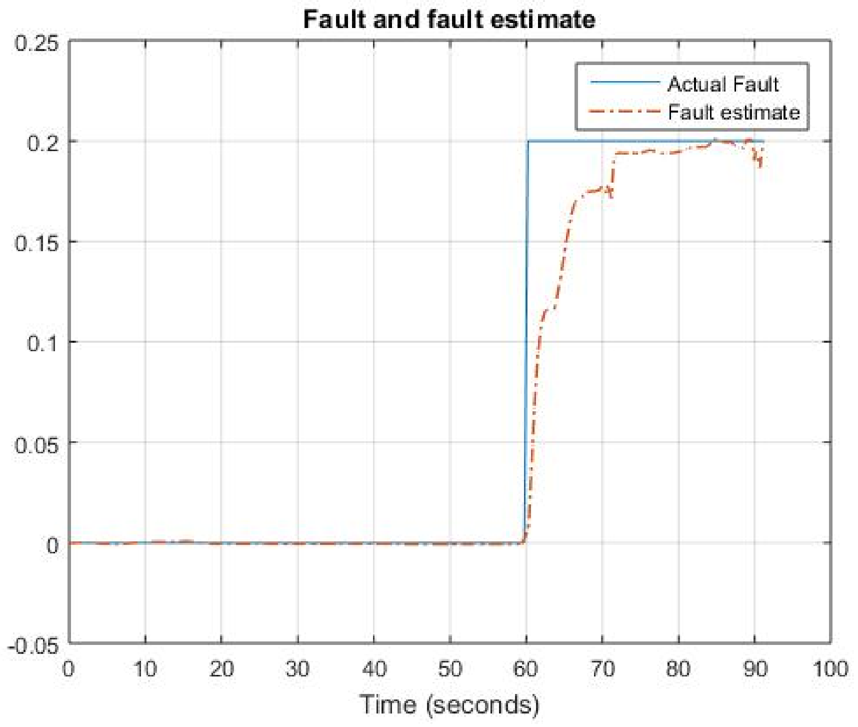

Figure 5.12: Fault Magnitude Estimate

Motor Commands

The motor command given to the robot is shown in Figure 5.13. From the figure, the introduction of fault to the motor command at t=60s is clearly seen by the dotted line, named Vf. This represents the velocity command given to the robot after the fault is introduced. The solid line in Figure 5.13 given by V0, represents the nominal velocity, i.e., the wheel command before the introduction of fault in the wheels. Vf is the same as the nominal velocity, V0, until the introduction of fault. Once the fault is introduced, the input command shoots up causing undesirable changes in the behavior of the robot. Figure 5.14 shows the motor commands after the fault is accommodated. Before 60 seconds, it is clear from the figure that there is no fault and the motor command, Vf, follows the nominal input command, V0. The fault was introduced at t=60s which is evident from the figure. After the fault was estimated and accommodated, the motor command follows the nominal input command, V0.

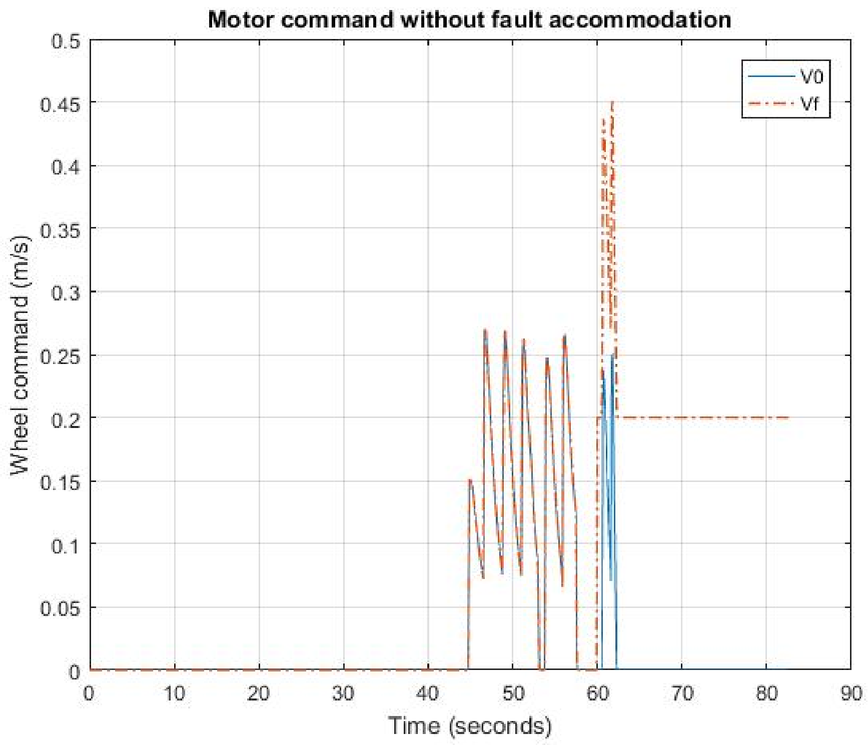

Figure 5.13: Motor command when there is no fault accommodation

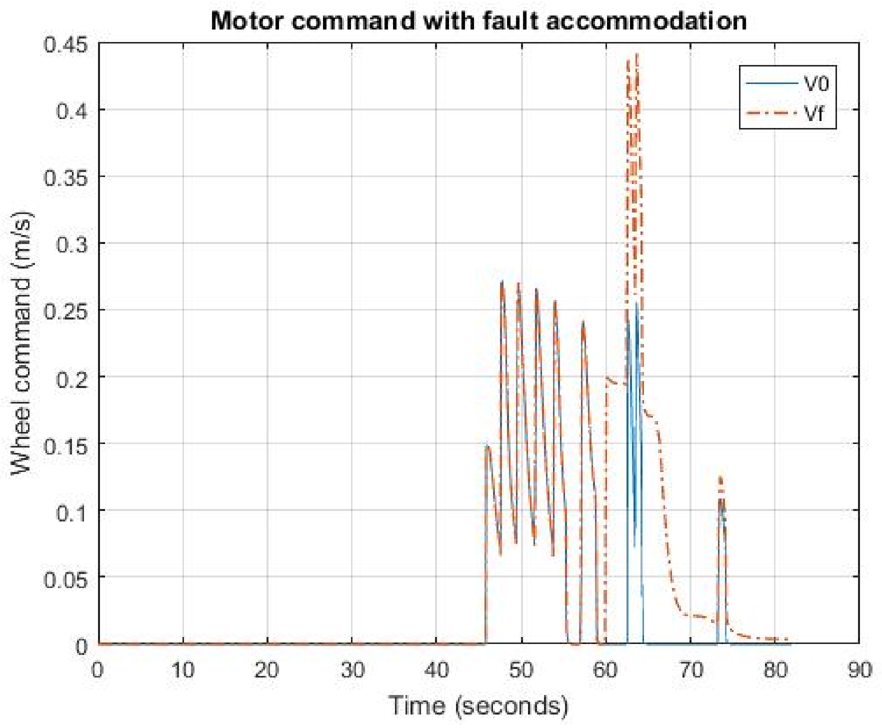

Figure 5.14: Motor command after fault accommodation

Conclusion

6.1 Summary

The growth of applications on unmanned vehicles has been tremendous in the past decade. The first step towards making a vehicle unmanned needs a robust path planning and obstacle avoidance algorithm. A sturdy, reliable, and robust system is necessary in order for it to be autonomous. The robot should have the ability to rectify any undesirable effects like faults in the sensors or actuators that could occur as the robot approaches the target. In order to develop such an effective system and to rectify any abrupt faults occurring in the system, a robust fault-tolerant algorithm needs to be developed.

One of the primary contributions of this research was integrating the mapping, path planning, and motion control model altogether in order to make the entire system autonomous and to work in real time. The Kinect Sensor data was utilized to get the map of the environment. This map was processed in real time to convert to a form usable for the path planning algorithm. The A star algorithm for path planning was implemented in real time using MATLAB/Simulink. The pre-processed map and the target positions were fed as the inputs to the A star algorithm and the robot was initialized. The output of the A star algorithm are the way points which was fed to the motion control system. Once, all these systems were triggered, the robot automatically mapped the environment, planned a path and moved along the way points to reach the destination.

The next contribution to this research was developing the model for dynamic obstacle avoidance. Dynamic obstacles considered here means any obstacle that was not known while generating the map. The obstacles appeared at a later time when the robot was in motion towards its target. The A star algorithm was enabled at all times until the robot reached the target successfully, thereby, avoiding any obstacle that came in the way. A new path was developed around the obstacle.

Another important contribution to this research is the development of a fault diagnosis and fault-tolerant control algorithm to achieve successful navigation and control of the robot even in the presence of faults. A bias actuator fault was introduced to both the wheels of the robot. The fault detection algorithm was able to quickly determine the presence of fault, and the adaptive parameter estimation algorithm was implemented to estimate the magnitude of fault. Based on the estimated fault, the effect of the fault was successfully accommodated.

6.2 Future Work and developments

From the above shown results, it is clear that there are some odometric localization errors which can be avoided in the future for better results. Different types of controllers apart from a proportional controller can be implemented for optimization.
There are some limitations with using the Microsoft Kinect Sensor. The distance data to any object closer than 0.5m or farther than 5m to the robot is considered invalid. The software zeros down this value. The robot considers there is no obstacle up to 0.5m from it. Thus, any obstacle within 0.5m from the robot will not be detected and the robot can collide with it. The FOV of the Kinect sensor is 57°. To get the entire view of the environment, the robot needs to be rotated 360°. To overcome all these limitations,

different sensor like a Laser sensor can be used.

A bias actuator fault is considered in this thesis. The investigation of sensor faults and other types of actuator faults is another interesting direction for future research.

The Kinect sensor updates the map at a rate of 10-20 seconds. The robot can successfully avoid any obstacle that arrives well within the time the robot arrives at that point. Any obstacle that pops up suddenly right in front of the robot may not be detected, and the robot can collide with it. These issues can be overcome by using Laser sensors instead of the Kinect Sensor. At this point, the velocity of the dynamic obstacle is also not considered. The research can be further enhanced by addressing these limitations.

Bibliography

[1] Joanne Pransky. “Mobile robots: big benefits for US military”. In: *Industrial Robot: An International Journal* 24.2 (1997), pp. 126–130.

[2] Ravindra S Bisht and Soju J Alexander. “Mobile Robots for Periodic Maintenance and Inspection of Civil Infrastructure: A Review”. In: *International Conference on Machines and Mechanisms*. 2013.

[3] Kazunobu Ish, Hideo Terao, and Noboru Noguchi. “Navigation of an agricultural autonomous mobile robot”. In: *Advanced Robotics* 13.3 (1998).

[4] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986.

[5] Meng Guanglei and Pan Haibing. “The application of ultrasonic sensor in the obstacle avoidance of quad-rotor UAV”. In: *Guidance, Navigation and Control Conference (CGNCC), 2016 IEEE Chinese*. IEEE. 2016.

[6] Petar M Djuric et al. “Particle filtering”. In: *IEEE signal processing magazine* 20.5 (2003), pp. 19–38.

[7] Frank Dellaert et al. “Monte carlo localization for mobile robots”. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[8] Zui Tao et al. “Mapping and localization using GPS, lane markings and proprioceptive sensors”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*. 2013.

[9] Abraham Bachrach et al. “RANGE–Robust autonomous navigation in GPS-denied environments”. In: *Journal of Field Robotics* 28.5 (2011).

[10] Muhammad Fahmi Abdul Ghani, Khairul Salleh Mohamed Sahari, and Loo Chu Kiong. “Improvement of the 2D SLAM system using Kinect sensor for indoor mapping”. In: *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*. IEEE. 2014.

[11] Xinzheng Zhang, Ahmad B Rad, and Yiu-Kwong Wong. “Sensor fusion of monocular cameras and laser rangefinders for line-based simultaneous localization and mapping (SLAM) tasks in autonomous mobile robots”. In: *Sensors* 12.1 (2012).

[12] Daniel Pizarro et al. “Localization of mobile robots using odometry and an external vision sensor”. In: *Sensors* 10.4 (2010).

[13] Antonio Sgorbissa and Renato Zaccaria. “Planning and obstacle avoidance in mobile robotics”. In: *Robotics and Autonomous Systems* 60.4 (2012).

[14] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[15] Hoc Thai Nguyen and Hai Xuan Le. “Path planning and Obstacle avoidance approaches for Mobile robot”. In: *arXiv preprint arXiv:1609.01935* (2016).

[16] Kai-Hui Chi and Min-Fan Ricky Lee. “Obstacle avoidance in mobile robot using neural network”. In: *Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on*. IEEE. 2011, pp. 5082–5085.

[17] Phuong DH Nguyen, Carmine T Recchiuto, and Antonio Sgorbissa. “Real-time path generation and obstacle avoidance for multirotors: a novel approach”. In: *Journal of Intelligent & Robotic Systems* 89.1-2 (2018).

[18] G Heredia et al. “Sensor and actuator fault detection in small autonomous helicopters”. In: *Mechatronics* 18.2 (2008).

[19] Mohsen Khalili. “Distributed adaptive fault-tolerant control of nonlinear uncertain multi-agent systems”. PhD thesis. Wright State University, 2017.

[20] Thomas Hellström. *Kinematics equations for differential drive and articulated steering*. Department of Computing Science, Umeå University, 2011.

[21] *Quanser user manual*.

[22] Petros A Ioannou and Jing Sun. *Robust adaptive control*. Vol. 1. PTR Prentice-Hall Upper Saddle River, NJ, 1996.

[23] N Namitha et al. “Point Cloud Mapping Measurements Using Kinect RGB-D Sensor and Kinect Fusion for Visual Odometry”. In: *Procedia Computer Science* 89 (2016), pp. 209–212.

[24] Michael Riis Andersen et al. “Kinect depth sensor evaluation for computer vision applications”. In: *Technical Report Electronics and Computer Engineering* 1.6 (2012).

[25] Lisa Turner and Christopher Sherlock. “An introduction to particle filtering”. In: *Lancaster University, Lancaster* (2013).

[26] ECJ Hall. “Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications”. In: (1993).

[27] Patrick Lester. “A* pathfinding for beginners”. In: *online]. GameDev WebSite. http://www. gamedev. net/reference/articles/article2003. asp (Acesso em 08/02/2009)* (2005).

[28] Denis F Wolf et al. “Autonomous terrain mapping and classification using hidden markov models”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005.

[29] Mogens Blanke et al. *Diagnosis and fault-tolerant control*. Vol. 2. Springer, 2006.

[30] Akshay Kumar Guruji, Himansh Agarwal, and DK Parsediya. “Time-efficient A* algorithm for robot path planning”. In: *Procedia Technology* 23 (2016).