Obstacle Avoidance for Mobile Robots

A Project Report

submitted by

MENTA SANDEEP

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **Obstacle Avoidance for Mobile Robots**, submitted by **Menta Sandeep**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology (Electrical Engineering)**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Arun D. Mahindrakar
Research Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th May 2016

ACKNOWLEDGEMENTS

I would like to thank my guide Dr. Arun D. Mahindrakar for giving me the opportunity to work under his guidance and supporting me throughout the project with a lot of patience and enthusiasm. I am indebted to him for having mentored me for over 2 years, helping me grow and learn the necessary skills required in executing a project successfully. I am grateful to him for understanding and supporting me through difficult times and failures, I cherish every interaction I have had with him during my stay here.

I express my gratitude to my friends and colleagues Akshit, Srinath and Aditya for all the constructive debates, help and insight into various problems related/unrelated to my project.

ABSTRACT

KEYWORDS: Mobile Robot; Obstacle avoidance.

The increasing applications of mobile robots in real world applications heralds the necessity for equipping the robots with capabilities to ensure safety while operating in environments with uncertainties. One of the major concern during their operation in any environment is avoiding obstacles while executing a task. The following report describes a simple algorithm which can be implemented on a mobile robot. The details about its implementation on a two wheeled non holonomic mobile robot are discussed and the experimental results are presented.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Mobile robots are robotic systems that can move and perform various tasks. Their uses and importance are steadily growing due to their adaptability to versatile applications. They are being used for various purposes in the real world, for example in factories to move resources around, autonomous exploration, monitoring and mapping of environments. The uncertainty in the workspace and environment of the robot makes its locomotion a challenging task. Hence the ability of the robot to detect and avoid obstacles gains a lot of importance as it is essential for the robot to complete its task. This problem has been known for quite long and many techniques and algorithms have been developed to enable the robot to safely manoeuvre through environments with convex obstacles which are static. Some of the earliest obstacle avoidance techniques are the Bug algorithms Sezer and Gokasan (2012) , M. Zohaib (2013), which use onboard sensors to avoid obstacles by tracking the obstacle walls, these algorithms need low processing power and can also avoid concave obstacles. The Artificial Potential method Sezer and Gokasan (2012), Oroko and Ikua (2012) is another easy technique for obstacle avoidance. This technique requires more information about the environment though and gets stuck in local minima. Orbital Obstacle avoidance techniques Adouane (2009) use switching control which makes the robot track a limit cycle about the obstacle and avoid it. Virtual Field Histogram(VFH) method Borenstein and Koren (1991) is a computationally demanding algorithm which allows the robot to successfully avoid even 'U' shaped objects. One of the main hurdles to implementing obstacle avoidance algorithms is the requirements on the sensing

capabilities of the robot, for example the VFH and Orbital Obstacle avoidance techniques.

This report describes the implementation of a simple algorithm along the lines of the bug algorithm to avoid obstacles. The robot used for implementation is a 2 wheeled robot, shown in Fig. 1.1 and Fig. 1.2. This robot is a non holonomic system which is equipped with two ultrasonic sensors. The limited sensing capabilities of the robot limit the possibility of implementing complicated algorithms or control laws. The algorithm described in this report has been designed to avoid rectangular obstacles with one of the faces perpendicular to the trajectory of the robot.

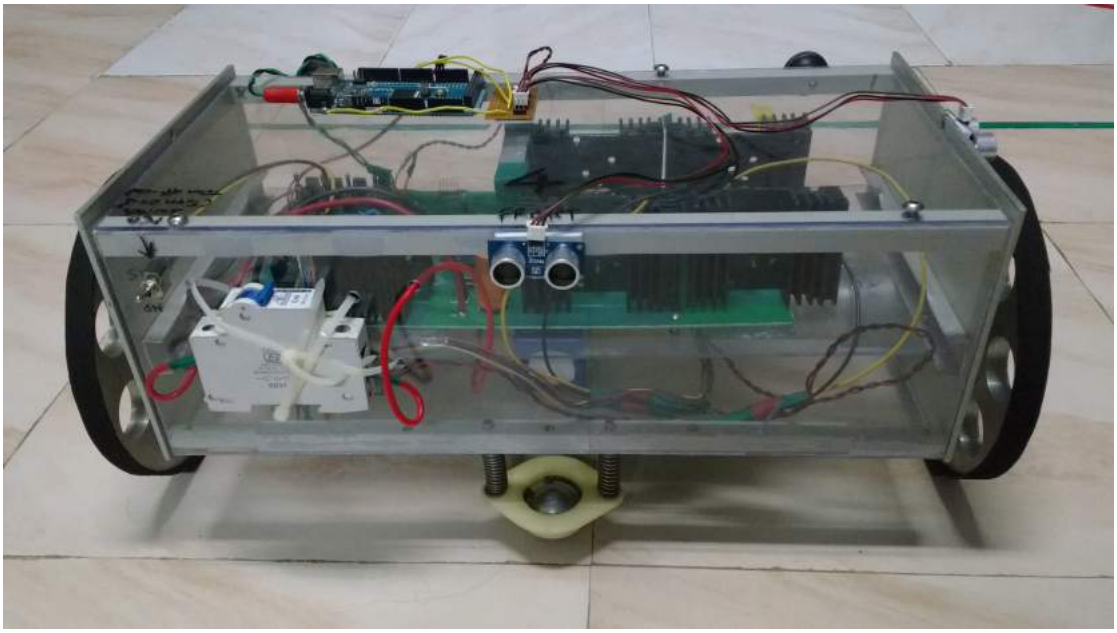

Figure 1.1: Front view of the robot

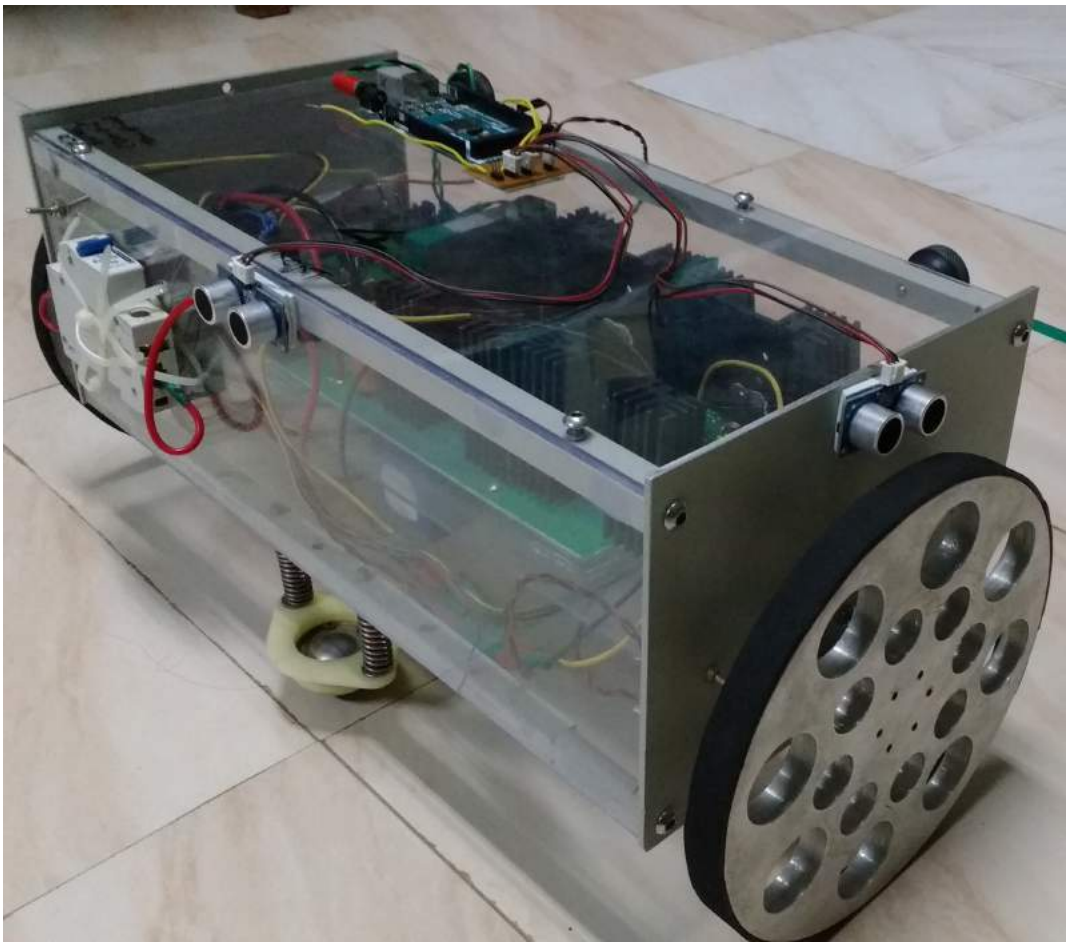

Figure 1.2: Side view of the robot

CHAPTER 2

HARDWARE DESCRIPTION

2.1 Measuring distance using Ultrasonic sensor

The sensor returns the distance to any object in the form of a pulse that is high for a duration proportional to the distance of the object in centimetres. The values are collected from the sensor using an Arduino Mega. The ultrasonic being used is PING))) 28015 sensor. It needs a power supply of 5V, which is drawn from the main power supply of the mobile robot. The signal pin is the one which takes the pulse input to trigger a measurement and gives an output pulse proportional to the distance. The signal pin is connected to one of the digital pins on the Arduino. This pin on the Arduino is continuously toggled between input and output to take readings. Sensor 1(front) is connected to pin 3 on Arduino and sensor 2(side) to pin 4. A narrow pulse of $5\mu s$ is passed to trigger the reading and then the value of the distance corresponding to the length of the pulse is calculated using the speed of sound which is roughly 1cm per $29\mu s$. An upper bound of 255cm is applied on the sensor readings as this would allow us to transmit the values to the PIC microcontroller in a single byte instead of 2 bytes which would be necessary in case we used the full range of the sensor, this does not cause any problems since the values which are of some consequence are much lower than 255. The code given in the following section is used to interface the Arduino with the sensors and transmit the values to the PIC.

Figure 2.1: PING))) 28015 Ultrasonic Sensor

2.2 Arduinio code

```
const int pingPin1 = 3;
const int pingPin2 = 4;
void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  Serial3.begin(19200,SERIAL_8E2);
}

void loop()
{
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  unsigned long duration, P1cm,P2cm,P3cm;
  byte temp1,temp2;
  //Sensor 1
```

```
pinMode(pingPin1, OUTPUT);
digitalWrite(pingPin1, LOW);
delayMicroseconds(2);
digitalWrite(pingPin1, HIGH);
delayMicroseconds(5);
digitalWrite(pingPin1, LOW);
pinMode(pingPin1, INPUT);
duration = pulseIn(pingPin1, HIGH);
P1cm = microsecondsToCentimeters(duration);
temp1=P1cm;
if(P1cm>=256) temp1=255;
//Sensor 2
pinMode(pingPin2, OUTPUT);
digitalWrite(pingPin2, LOW);
delayMicroseconds(2);
digitalWrite(pingPin2, HIGH);
delayMicroseconds(5);
digitalWrite(pingPin2, LOW);
pinMode(pingPin2, INPUT);
duration = pulseIn(pingPin2, HIGH);
P2cm = microsecondsToCentimeters(duration);
temp2=P2cm;
if(P2cm>=256) temp2=255;
while(!Serial3){}
Serial3.write(1);
Serial3.write(temp1);
Serial3.write(2);
Serial3.write(temp2);
delay(50);
```

```
}
long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per
     centimeter.
  // The ping travels out and back, so to find the
     distance of the
  // object we take half of the distance travelled.
   return microseconds / 29 / 2;
}

//This is the code used to receive values from the Arduino
   the computer
void serialEvent3() {
  while (Serial3.available()) {
     Serial.print("Sensor id = ");
     while(!Serial3.available()){}
     Serial.print(Serial3.read());
     Serial.print("t");
     Serial.print("Sensor val = ");
     while(!Serial3.available()){}
     Serial.print(Serial3.read());
       Serial.println();
  }
}
```

2.3 UART (Universal Asynchronous Receiver/Transmitter)

This is an asynchronous form of serial communication used to exchange data between two devices. An Arduino is used to interface the ultrasonic sensor with the PIC as there are not enough free pins on the PIC which can be accessed, which limits the number of sensors which can be added to the PIC directly. Using the Arduino instead allows the addition of more sensors without increasing the burden on the PIC microcontroller. For this purpose the UART2 port mapped to pins 21(Tx) and 22(Rx) of the PIC is connected to the third UART port of the Arduino, which are on pins 14(Tx) and 15(Rx). The pins 21 and 22 of the PIC can be accessed through the unused XBee slot on the board. The Tx of one device must be connected to the Rx of the other for successful communication. Hence, pin 14 of Arduino is connected to pin 22 of the PIC and pin 15 of Arduino is connected with pin 21 of PIC, as shown in Fig. 2.2. Another important requirement is that both the devices must have common ground for proper communication.

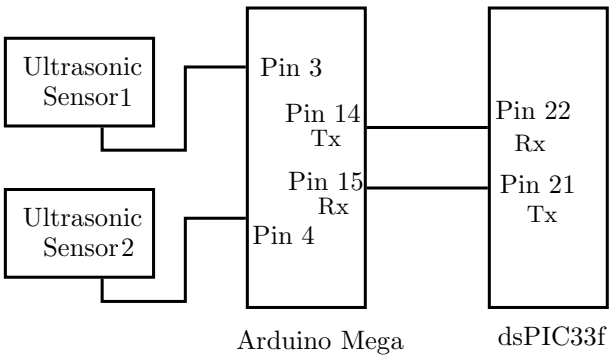

Figure 2.2: Block Diagram of connections between devices

The data is exchanged in packets with a start bit, two stop bits, even parity and at a baud rate of 19200. The register U2BRG controls the baud rate and a value of 128 supports a baud rate of 19200. The register U2MODE holds the values which decide the settings for the data packet and the U2STA register holds the settings for enabling and disabling the Rx or Tx and the corresponding interrupts and the conditions under which an interrupt is generated. Also the value in the U2STA register must be set to ensure that the idle state of the Tx pin is high. To be able to successfully read the values of sensors the Rx interrupt is generated whenever a byte is received by the Rx register U2RXREG and the values are read as described in later sections.

The appropriate initializations for Arduino are done in the `setup()` function given in Sec. 2.2. It can be seen in the code that UART3 of Arduino is initialized with a baud rate of 19200 and an 8 bit packet with even parity and 2 stop bits (SERIAL_8E2).

CHAPTER 3

SOFTWARE FRAMEWORK

In this section the framework of the program and the contents of the various parts of the code are explained. The code is spread across 6 files which are main.c, control.c, uart.c, timerint_enc.c, misc.c and variables_header.h. As mentioned earlier all the computations are performed at a frequency of 100 Hz using timer interrupts.

At start up the first file to be executed is main.c which has all the necessary initializations like setting all the pins to digital mode, declaring the pins to be input or output by setting the values of the bits corresponding to each pin in the TRISB register. Assigning various tasks to the pins is done using the functions RPINRx and RPORx corresponding to the appropriate function and pin. This file also has the settings for various functions, like settings for communication using the UART ports, for which the baud rate, parity, number of bits, enabling or disabling the Rx ad Tx and their interrupts and other properties are set. The digital filters for the encoders and the properties for the timer interrupts are also set. The code enters an infinite loop at the end of all initializations and waits for the interrupts to execute various parts of the code.

The timer interrupts occur once every 10ms and the timerint_enc.c file has the corresponding interrupt service routine. Every time an interrupt occurs the states x, y, orientation, linear and angular velocities are computed using various filters for removing noise in dead reckoning and by curve fitting linear and angular velocities. After all these computations the control inputs are calculated by executing the code in control.c file.

The rest of the interrupt service routines to handle the interrupts from both the UART ports are present in uart.c. UART1 is currently being used for transmitting states to the computer using a XBee which is done at the end of control.c and UART2 is used for obtaining the values of the sensor readings from Arduino. UART1 is primarily used for transmitting values and UART2 for receiving values. The protocol followed by Arduino to transmit the sensor readings is as follows. The Arduino first transfers a number as an identifier for the sensor before transmitting the corresponding reading, hence the first value received by the Rx buffer(U2RXREG) is the identifier which is followed by the sensor reading which is read into the appropriate variable for later use.

The control.c file has the main control law and the algorithm for obstacle avoidance. The robot initially waits for 5 seconds before it starts executing the task. Then it starts to move according to the control algorithm (Fig. 3.1). The control algorithm is as follows,

1. The robot computes the desired orientation to track a straight line towards the final point.
2. The robot turns by an angle equal to the difference of the desired orientation and initial orientation towards the final point.
3. The robot tracks a straight line using control inputs given by 3.1 till it encounters an obstacle, it continues to move towards the final point if it doesn't encounter an obstacle.
4. On encountering an obstacle the robot stops at a distance of 50cm from it and turns clockwise by 90 degrees.
5. It then moves in a straight line until it crosses the end of the obstacle by 50 cm and stops.
6. Then the robot turns counter-clockwise by 90 degrees.
7. The robot moves in a straight line till it passes the farther end of the obstacle.
8. The robot then computes the angle by which it would need to turn to reach the final point in a straight line.

9. Then it continues to move in a straight line till it reaches the final point or repeats the obstacle avoidance manoeuvre if it encounters another obstacle.

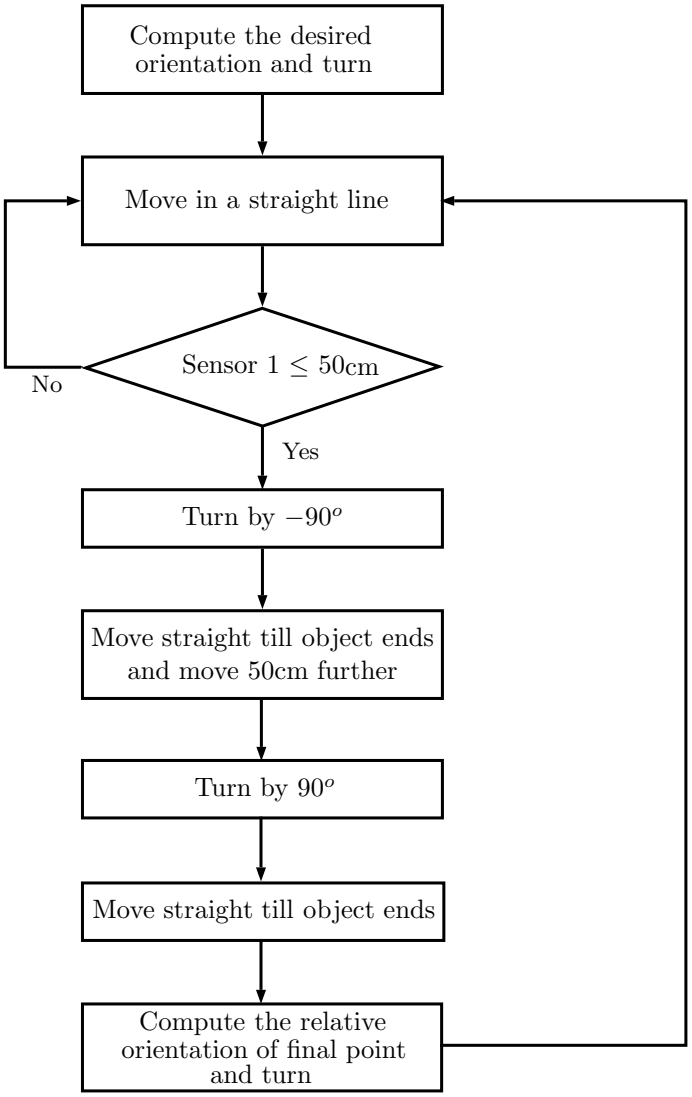

Figure 3.1: Flow diagram of obstacle avoidance algorithm

After each turn the obstacle is detected with the help of the second sensor, the readings from the second sensor are considered to check if the robot has crossed the obstacle, this is important to ensure that there is enough room for the robot to turn without hitting the obstacle. We need the robot to move past the edge of

obstacle by 50cm since the robot is approximately 50 cm wide and it turns about its centre which leaves a space of approximately 25 cm between the robot and the obstacle. Though in reality the inertia of the robot carries it further before stopping. Apart from the obstacle avoidance manoeuvre and turning, the robot is controlled using the control law 3.1 which is proven to be globally asymptotically stable which ensures that after the robot orients itself towards the target it traverses a straight line path. Since the robot is controlled by the control law there is no necessity for a stopping condition as the control inputs become insignificant when the robot gets sufficiently close to the target equilibrium manifold.

The control law used for moving towards the target in a straight line is,

$$\begin{aligned} v &= -k_v \tanh((x - x_d)\cos\theta + (y - y_d)\sin\theta) \\ \omega &= \pm k_\omega \tanh((x - x_d)\sin\theta - (y - y_d)\cos\theta) \end{aligned} \tag{3.1}$$

where x,y and θ are the states of the system and x_d and y_d are the position co-ordinates of the destination. This control law is globally asymptotically stable, hence asymptotically converges to the destination, it was presented in Muralidharan (2014).

This file has the code to convert the control inputs to voltage outputs based on the desired velocity(veld) and desired angular velocity (omegad). The voltages required are calculated using 2 empirical constants,k_v and k_ω, to factor in the effects of the difference between actual and desired linear velocity and actual and desired angular velocity along with the rate of rotation of the wheels. $k_v = 2$ and the values of k_ω are modified depending on the distance from the target point. The

following equations are used to compute the voltages,

$$\begin{aligned} force_1 &= -k_v(v - v_d) \\ \tau_1 &= -k_w(\omega - \omega_d) \\ control_1 &= (force_1 + \tau_1) \\ control_2 &= (force_1 - \tau_1) \\ voltage_1 &= 1203.0133 * psidot_1 + 3679.6488 * control_1 \\ voltage_2 &= 1203.0133 * psidot_2 + 3679.6488 * control_2 \end{aligned} \tag{3.2}$$

where v and ω are the states of the system, v_d and ω_d are the desired values which are obtained from 3.1, $psidot_1$ and $psidot_2$ are computed in timerint_enc.c and $voltage_1$ and $voltage_2$ are the voltages to be supplied to the right and left wheels respectively. These voltages are then converted to values required for PWM signals and written to the registers P1DC1 and P1DC2 for right and left wheels respectively. The required value of in the register for zero velocity PWM is 32000 and the calculated values are added and subtracted (since the wheels need to rotate in the opposite directions for the robot to move forward or backward in a direction) appropriately. Then the transmission of the states is initialized by writing the first state into the transmit buffer following which the ISR corresponding to the Tx complete of UART1 completes the transmission of the rest of the states.

The turning of the robot is executed using the function 'turn' which takes the angle through which the robot has to rotate and the initial orientation at which the turning starts. To make a turn and stop smoothly without overshooting due to inertia the following function is used to supply the desired angular velocity,

$$\omega_d = \omega_{max} \tanh(20((\theta_d - \theta)^2 + 0.01) \times \text{sign}(\theta_d - \theta)) \tag{3.3}$$

The function uses a $\tanh$ function as it is a smooth function which is also bounded.

The square of the error is used to ensure that the angular velocity drops significantly initially and approaches zeros slowly, using the error directly instead of squaring it causes overshooting. The signum function ensures the correct direction for correction and there is another drift term which is necessary as when the error approaches zero the desired angular velocity becomes very small and the robot doesn't reach the final orientation due to mechanical friction, the drift term ensures a minimum desired angular velocity which allows the robot to reach the desired final orientation within the allowed error tolerance. The function allows turning about the centre or right or left wheel by passing the value of the second parameter as 0 or 1 or 2 respectively.

An important observation is that during the operation of the robot if a delay function is introduced, the UART communication with the Arduino is disrupted entirely. Due to this reason no delays have been introduced at the transition between two motions, for example while changing from straight line motion to pure turning.

CHAPTER 4

EXPERIMENTAL RESULTS

The mobile robot is initially placed at (-3,0) with an orientation of 0^o and is required to go to the target point of (1,0) with any final orientation. A rectangular obstacle is placed in the path of the robot with the corners at points $(-1.28m, -0.3m)$, $(-0.81m, -0.3m)$, $(-0.81m, 0.23m)$ and $(-1.28m, 0.23m)$. The obstacle is shown in the plot as a black rectangle. Fig. 4.1 shows the path

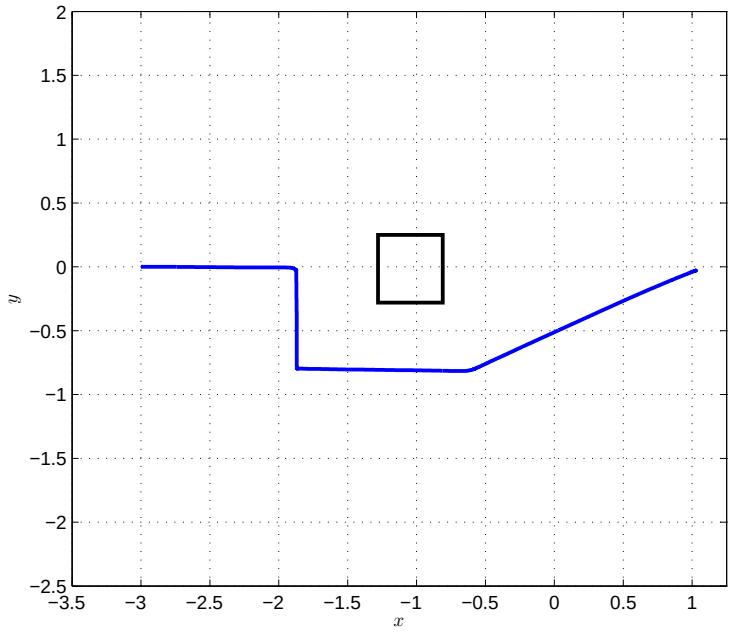

Figure 4.1: X-Y plot of the path traced by the mobile robot

traced by the mobile robot on encountering the obstacle. The robot stops at the the point $(1.024m, -0.003m)$ which is off the target point by $0.0242m$.

4.1 Discussion

From the plot we can clearly see that the mobile robot approaches the obstacle closer than $0.5m$ and overshoots the desired stopping distances from the edges due to its inertia. In order to reduce the effect of stopping suddenly or approaching the obstacle too closely the robot is slowed down when it gets within 80cm of an obstacle and also it is run at a low speed while performing obstacle avoidance. The overshoots do not impede the efficacy of the algorithm as the effect of overshoot in the position and orientation is captured due to the high frequency of dead reckoning and is corrected by the control law 3.1 which takes over again at the end of the obstacle avoidance manoeuvre.

The mobile robot is supplied fixed values of desired linear and angular velocities during the obstacle avoidance manoeuvre, which involves moving in a straight line and turning about its own centre. It is seen that robot moves in a curved path instead of a straight line when we set v_d to a non zero constant and ω_d to 0. Through experimentation it is seen that this behaviour can be corrected by supplying a bias to ω_d proportional to v_d. The proportionality constant in the linear relation between the bias and v_d varies slightly over time and might have to be recalibrated once in a while. This can be attributed to the various errors in modelling and to the difference in the behaviour of the motors.

The ability of the robot to reach the target point depends heavily on its initialization. The values of the initial position and orientation of the robot are passed as constants in the program. The sensitivity to the initial conditions arises due to the fact that the robot has no way of detecting the errors in initialization i.e. if the initial conditions passed to the robot through the program match with the position and orientation at which the robot is actually placed. Hence the robot

cannot make any corrections and proceeds to complete the task assuming zero initialization error. So care must be taken to ensure minimal error in initialization.

4.2 Future Directions

The algorithm is currently capable of avoiding rectangular obstacles with one of the faces perpendicular to the path of the robot, this was achieved using two sensors. One of the possible improvements to this algorithm is to add another sensor on the same face as sensor 2 or sensor 1 which can be used to orient the robot perpendicular to the face of the rectangular obstacle which would allow the current algorithm to manoeuvre around the obstacle.

The algorithm can be extended to avoid convex obstacles by placing two sensors on the side face which can then be used to track the wall of the obstacle. Adding more sensors to the robot would allow it to implement more control laws like VFH, artificial potential method and orbital obstacle avoidance algorithm which would need more data about the surroundings. That would allow the robot to avoid non convex obstacles too.

REFERENCES

1. **Adouane, L.** (2009). Orbital obstacle avoidance algorithm for reliable and online mobile robot navigation. 9th Conference on Autonomous Robot Systems and Competitions.

2. **Borenstein, J.** and **Y. Koren** (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, **7**(3), 278–288.

3. **M. Zohaib, R. R. N. J. M. R. K., M. Pasha** (2013). Control strategies for mobile robot with obstacle avoidance.

4. **Muralidharan, V.** (2014). *Smooth Asymptotic Stabilization of Position and Reduced Attitude for Nonholonomic Mobile Robots*. Ph.D. thesis, Indian Institute of Technology Madras.

5. **Oroko, J.** and **B. Ikua** (2012). Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: A review. *Sustainable Research and Innovation Proceedings*, **4**.

6. **Sezer, V.** and **M. Gokasan** (2012). A novel obstacle avoidance algorithm: Follow the gap method. *Robotics and Autonomous Systems*.