Centre for Product Design and Manufacturing

IISc Bangalore

Project Title: Control and Navigation of Differential Drive Mobile Robot

Video Link: [My_Work]

Funding Agency: 1. Department of Heavy Industries, Government of India

2. CEFC Smart Factory, IISC Bangalore

3. Robotics Innovations Lab, CPDM, IISC Bangalore

Project Overview

- Project consists of a programmable mobile robot(built using solidworks, 3-D printed and Arduino)
- Mobile robot communicates on MATLAB over Wi-Fi
- Install color-coded sticker on robot top to serve as a marker to assist the image processing algorithm which uses a webcam to detect location and orientation of the robot.
- Program mobile robot to perform tasks
  - *Path Following*
  - *Moving objects with a forklift*
  - *Obstacle avoidance*

Learning Objectives

- Assemble and configure different robotic components with the Arduino boards
- Capabilities about Arduino MKR1000 and Arduino MKR Motor Carrier.
- Understanding of robotics fundamentals
  - *Working of differential drive robots and simulating their behavior*
  - *Control position or speed of differential drive robots*
  - *Perform localization and navigation*
- Use MATLAB for programming the mobile robot.
- Vision-based image processing algorithm using a webcam for localization and navigation tasks.

Components Required

- Arduino MKR1000 1 nos.
- Arduino MKR Motor Carrier 1 nos.
- DC Motors with Encoders 2 nos.
- Standard Micro Servo 1 nos.
- Webcam 1 nos.
- Micro USB Cable 1 nos.
- Lipo battery 1 nos.

- Ultrasonic Sensor Module 1 nos.
- IR sensor Module 5 nos.
- Caster wheel 1 nos.
- DC motor mounting brackets 2 nos.
- Distance spacers 4 nos.
- Wheels 2 nos.
- Some additional components like screwdriver, nuts, bolts,color coded stick etc.

Exercises

Following exercises to be performed

- Exercise 1: Characterize a DC Gear Motor using MATLAB
- Exercise 2: Control the Mobile Rover and Forklift using MATLAB.
- Exercise 3: Kinematic and Dynamic Modelling of the Mobile Robot and Drive the Robot using Open Loop Control
- Exercise 4: Closed Loop Control of Rover Position and Orientation to Follow Specific Instructions
- Exercise 5: Vision based Localization using Webcam to Calculate Robot and Object Position in the Arena
- Exercise 6: Navigate the Arena and Move the Object
- Exercise 7: Line Following and Obstacle Avoidance

EXERCISE 1: CHARACTERISE A DC GEAR MOTOR USING MATLAB

In this section, we will write a MATLAB program that automatically measures the motor speed while issuing many different PWM commands to characterize the response of the 300:1 DC gearhead motor. Later, we will use these observations to determine the required PWM command to rotate the motor at a desired rotational speed.

Characterizing using PWM based Speed Control of DC Motor

Principle:

- Supplied a fixed voltage (Here, fixed voltage = battery voltage (11.1V)).
- Motor starts rotating immediately as voltage is applied.
- Voltage is then removed and the motor 'coasts.
- By continuing this voltage on/off cycle with a varying duty cycle, the motor speed can be controlled.

Equations Guiding Operation of DC Gear Motor

Voltage Equation

$$V = E_b + I_a * R_a$$

Induced Back Emf

$$E_b = k_f*\Phi*\omega = K_m*(N*300)$$

Torque Equation

$$T_a = k_t*\Phi*I_a = K_T*I_a$$

Where

V = Terminal Voltage

E_b = Induced back emf

I_a = Armature current

R_a = Armature Resistance

k_f = Constant based on machine construction

k_t = Constant based on machine construction

Φ = magnetic flux

ω = angular speed of the motor

K_m = voltage constant

N = shaft speed(in RPM)

K_T = Torque constant

Results

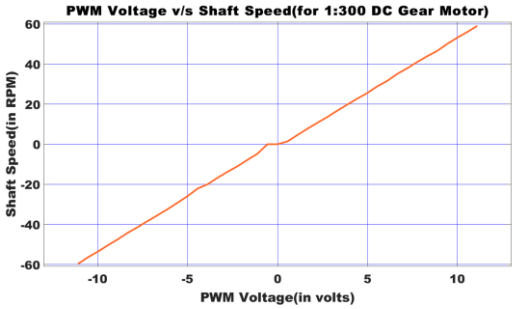


Fig.1: Motor Steady State Response

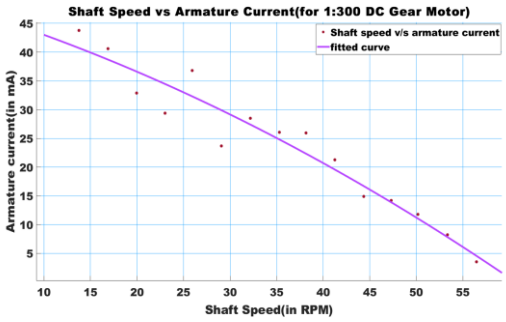


Fig.2: Shaft Speed v/s Armature Current

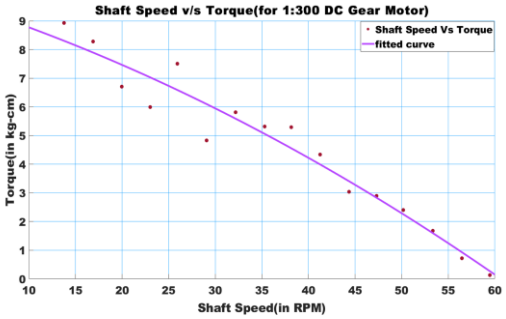


Fig.3: Shaft Speed v/s Torque

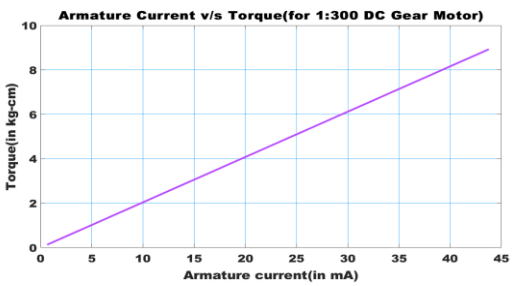


Fig.4: Armature Current v/s Torque

In Fig. 1, we can notice some interesting features of the speed-PWM relationship that you can deduct from studying the graph. There is a "dead zone" around PWM=0, where there is zero rotational speed for non-zero PWM commands. Furthermore, there may be some non-monotonic portions of the curve, where the measured speed does not increase with increasing PWM command. This typically happens around PWM = +/- 1, but could also result from experimental error.

EXERCISE 2: CONTROL THE ROVER AND FORKLIFT FROM MATLAB

The mini mobile rover is a differential drive robot that has separate DC motors controlling each of its wheels. We can control the direction of travel by varying the speed of each wheel without the need for any separate steering mechanism. In this exercise, you will learn how to drive the rover and operate the forklift using MATLAB.

In this exercise, we will learn to:

- Connect MATLAB to the rover over Wi-Fi
- Control basic movement of a differential drive robot.
- Drive the rover in a straight line.
- Drive the rover in circles.
- Control the forklift of the rover.

2.1 Connect to the Hardware via Wifi

1. Connect the Arduino MKR1000 to the system via a micro USB cable.
2. Enter the following command in the control window of MATLAB

   >> arduinosetup % Set up our Arduino board with the MATLAB environment.

Follow the steps as shown:

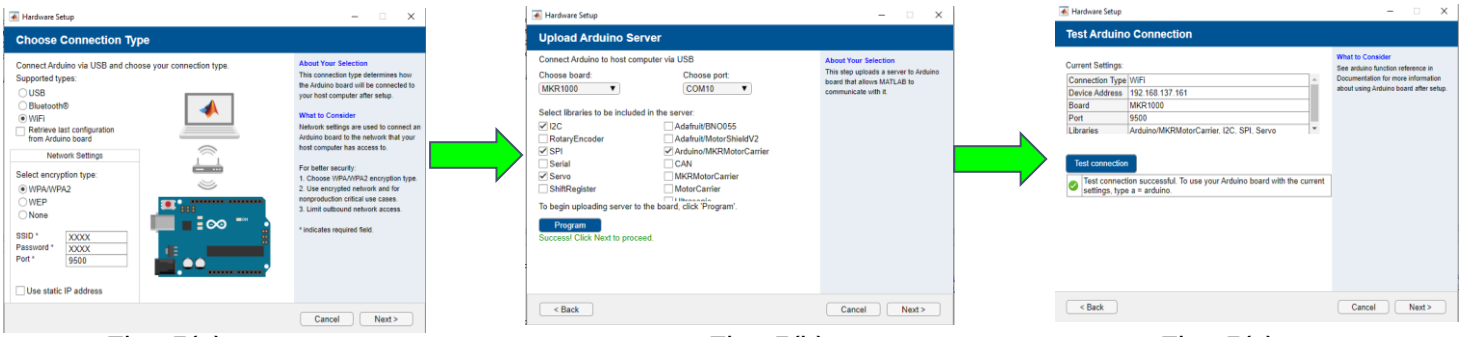


Fig.: 5(a) Fig.: 5(b) Fig.: 5(c)

Fig. 5: Arduino setup

Note: Do not switch off power supply to Arduino board after setup

In Fig. 5(a), we will be selecting the Wi-Fi option. Select the encryption type of the Wi-Fi and enter the SSID and the password corresponding to it. Make sure that the system in use is also connected to the same Wi-Fi.
Select the libraries required. The ones shown in Figure 5(b) are the basic libraries which needed to be uploaded for the project. Later on, in obstacle avoidance problem, we shall select the Ultrasonic library as well. After selection, press "Program" to upload to the Arduino server. After few minutes, it should show Success as shown in Figure 5(b).
In Fig. 5(c), we clear the workspace of MATLAB before running this setup. If everything is good, then the Test connection will show successful.

1.2 CONTROL THE ROVER FROM MATL AB

I. Drive in a straight line

We probably know intuitively that the rover will travel in a straight line if both wheels spin at the same speed and in the same direction. Let's spin the rover's DC motors at the same speed and observe whether it goes in straight line as expected.
MATLAB CODE for driving the rover in a straight line

```
clear;
clc;
a = arduino(); % Initialise the arduino object
carrier = addon(a,'Arduino/MKRMotorCarrier'); % Intermediary between
% Arduino object and DC and servo motors
dcmleft = dcmotor(carrier,4); % Object giving control of the motor connected
to M4
dcmright = dcmotor(carrier,3); % Object giving control of the motor connected
to M3
dcmleft.DutyCycle = 0.25;
dcmright.DutyCycle = 0.25;
start(dcmleft); % start left motor
start(dcmright); % start right motor
pause(3); % Ensure that the rover move for 3 seconds
stop(dcmleft); % stop left motor
stop(dcmright); % stop the right motor
```

Note: Do not switch off power supply to Arduino board after setup. Otherwise have to do the entire setup again.

Measure how far the rover moved when using these commands. Since the information coming from the encoders attached to each one of the DC motors is not being read yet, there is no way to ensure that the rover will either move completely straight (there might be slight differences in behaviour between the motors), or that it will move the same distance each time the program is executed (as time goes by, the batteries will have less and less charge and will behave differently).

REVIEW/SUMMARY

- We have seen how to drive the rover in a straight line and in circles.
- We have seen how to control the forklift, which will be used in later exercises to lift the target.

Exercise 3: Kinematic and Dynamic Modelling of the Mobile Robot and Drive the Robot using Open Loop Control

In Exercise 1, we learned how to make the rover drive in a straight line and in circles. However, to move the rover along more complex paths we need to understand the underlying kinematic equations that relate the speed of each wheel to the rover velocity and direction of travel

In this exercise, we will learn to:

- Understand kinematic equations of the rover
- Simulate the rover using kinematic equations.
- Model and simulate rover movement using MATLAB.
- Perform open-loop control of the rover.
- Use MATLAB models to control the actual rover.

3.1 KINEMATICS OF THE ROVER

The following diagram shows the basic kinematics of the rover (or any differential drive robot):

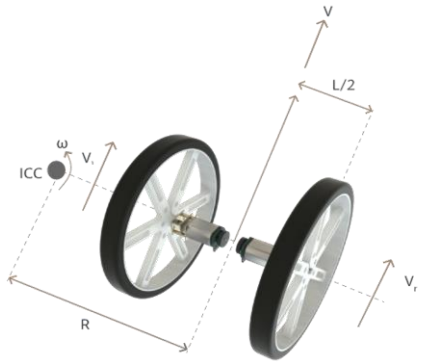


Fig. 6: Basic kinematics of the differential drive wheeled mobile robot

Parameters of differential drive robots as in Fig. 6
Radius of rotation (**R**),
Rate of rotation (**ω**),
Instantaneous center of curvature (**ICC**),
Forward velocity (**v**),
Wheel velocities (**v_l, v_r**)(**v_l** = forward velocity for the left wheel , **v_r** - forward velocity for the right wheel),
and **Rover length** (**L**)

Kinematic Modelling of the Differential Drive Robot

Rate of rotation: $\omega = (v_r - v_l)/L$

Forward velocity: $v = (v_r + v_l)/2$

Wheel velocities (vl, vr) based on **wheel radius** (r) and **rotational speeds** (ω_l, ω_r):

$$v_l = \omega_l.r$$

$$v_r = \omega_r.r$$

Plugging these values into the previous equations yield:

$$\omega = r.(\omega_r - \omega_l)\ /L$$

$$v = r.(\omega_r + \omega_l)\ /2$$

Rearranging these equations:

$$\omega_r + \omega_l = 2.v/r$$

$$\omega_r - \omega_l = L\omega\ /r$$

Solving for ωr and ωl:

$$\omega_r = (v + (L/2)*\omega)\ /r$$

$$\omega_l = (v - (L/2)*\omega)\ /r$$

In matrix form:

$$\begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} = (1/r)* \begin{pmatrix} 1 & -L/2 \\ 1 & L/2 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

For robot used, L = 12 cm and r = 4.5 cm.

Dynamic Modelling of the Differential Drive Robot[1]

Lagrange dynamic approach very powerful method for formulating the equations of motion of mechanical systems.

Differential Drive Mobile Robot dynamics can be expressed as a function **driving motor torques**(τ_R, τ_L) and linear and angular velocities derived from the kinematic modelling as:

$$\left(m + \frac{2I_w}{R^2}\right)\dot{v} - m_c\, d\omega^2 = \frac{1}{R}(\tau_R + \tau_L)$$

$$\left(I + \frac{2L^2}{R^2} I_w\right)\dot{\omega} + m_C d\omega v = \frac{L}{R}(\tau_R - \tau_L)$$

Here $\mathbf{m_c}$ = Mass of the robot without driving wheels
$\mathbf{I_w}$ = moment of inertia of wheel
$\mathbf{I}$ = moment of inertia of the robot

Simulating the Robot's Motion[2]

The ***location (x, y)*** and ***heading*** (θ) of the rover in an ***X-Y coordinate system*** can be described using the following equations:

$$\theta(t) = \int_0^t \omega(t)\, dt \qquad x(t) = \int_0^t v(t) \cos\theta\, dt \qquad y(t) = \int_0^t v(t) \sin\theta\, dt$$

Drive the Mobile Robot using open Loop Control

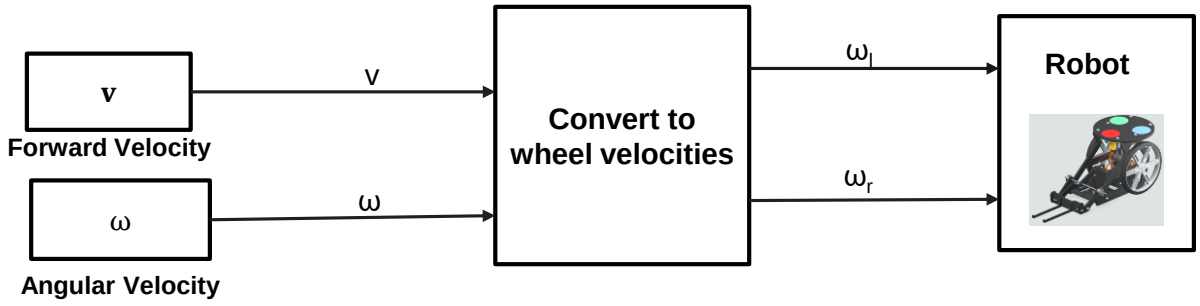


Fig.7: Open Loop Control of the Mobile Robot

Input Commands: Forward Velocity, Angular Velocity

.

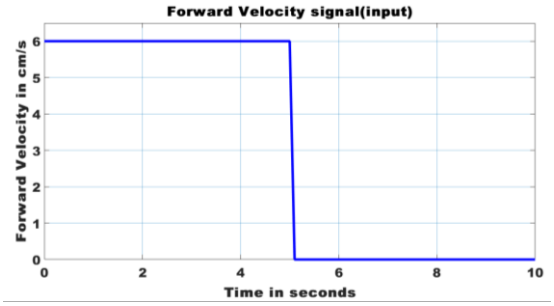


Fig. 8(a)

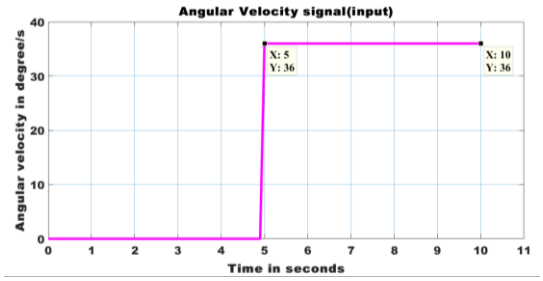


Fig. 8(b)

Fig. 8: Input Signal Commands

Results

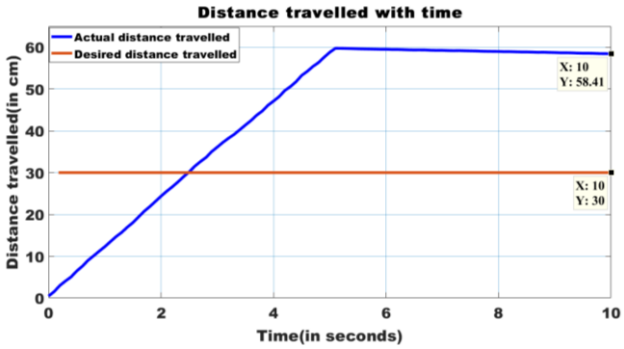


Fig. 9: Distance travelled by the robot

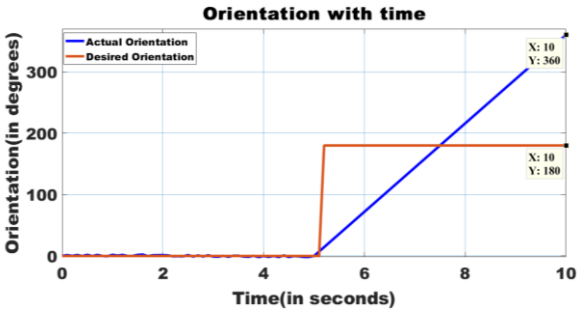


Fig. 10: Orientation of the robot

Final Percentage Error = (58.41-30)/30 X 100% = 95.03%

(For distance travelled)

Final Percentage Error = (360-180)/180 * 100 = 100%

(For Orientation)

As we can see that the rover does not travel the desired distance. In general, it's very difficult to accurately control your system using open-loop control unless we have a very accurate model of your motor behaviour and operating environment. In our case, the motor characterization was not rigorous and a simple relationship for PWM versus motor speed was used. That said, even if the characterization were rigorous there would be issues if you want to operate the rover on a surface different than the surface used for motor characterization, or if the battery is not operating at the same voltage.

RE VIEW/SUMMARY

- We saw how to use kinematic equations and inverse kinematics to
- simulate the rover movement.
- We saw how to translate that understanding to control the rover in the real world.
- We saw that the mathematical modelling needs to be tweaked to better accommodate real-life conditions.

Exercise 4: Closed-loop Control of Robot Position and Orientation to Follow Specific Instructions

In this exercise, we will use to implement a PID controller that controls the distance moved by the rover. We will learn how to read the encoder data from the robot's wheels and use this data as feedback to your controller. We will also learn how to tune PID values of the controller to optimize performance.
In this exercise, we will learn to:

- Understand basics of a PID controller design controller.
- Tune parameters of a controller.
- Implement a closed loop controller on the rover.
- Program the rover to follow specific tasks.

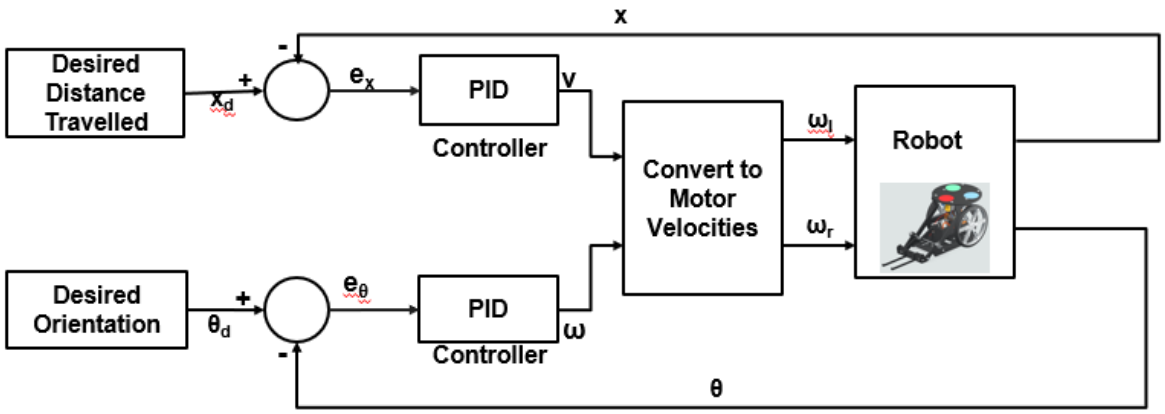


Fig.11: Closed Loop Control of the Robot

Here

x_d = Desired distance travelled
θ_d = Desired Orientation
x = current distance travelled
θ = current orientation

v = Forward velocity
ω = rate of rotation
ω_l = left wheel angular velocity
ω_r = left wheel angular velocity

e_x= error signal for controlling distance
e_θ= error signal for orientation

Experimental Results

Desired distance to be covered = 30 cm
Desired orientation = 180°

Results

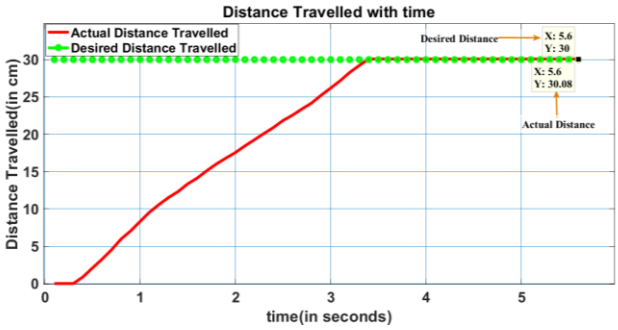


Fig.12: Control Distance Travelled

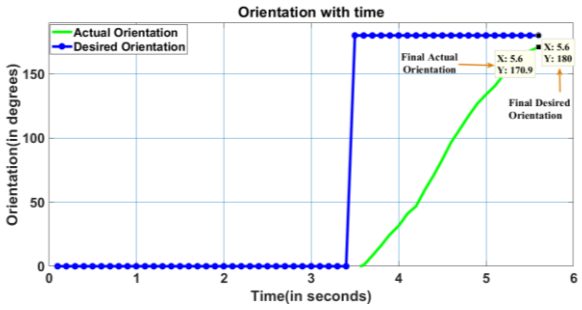


Fig.13: Control Orientation

After **trial-and-error method**,
Kp = 0.25, Ki = 0.05 and Kd = 0.005
Rise time = 2.5 seconds;
Percentage error = 0.29%

After trial-and-error method,
Kp = 3, Ki = 0.75 and Kd = 0.001
Rise time = 1.5 seconds
Percentage error = 5.55%

Performance: Open Loop Control v/s Close Loop Control

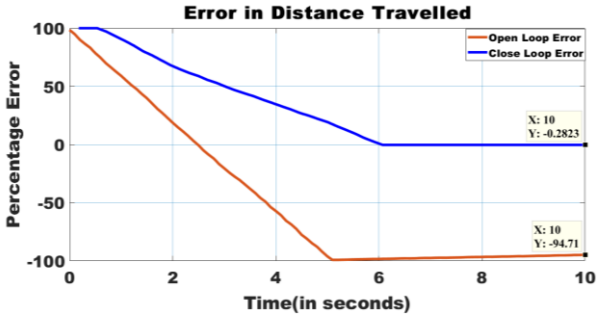


Fig.14: Performance Comparison: Error in Distance Travelled

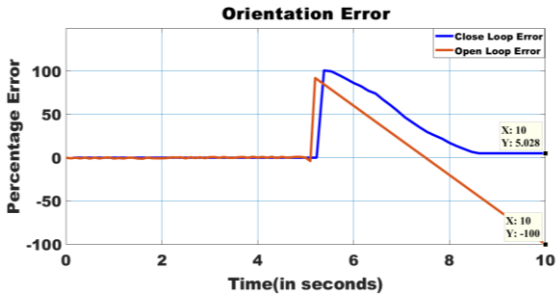


Fig.15: Performance Comparison: Orientation Error

Exercise 5: Vision based Localization using Webcam to Calculate Robot and Object Position in the Arena

When working with mobile robots, it's often important to know their location as they move around. This exercise shows how to use a webcam with a localization algorithm to calculate the robot's location and heading. To ensure the robot remains in the webcam's field of view, the robot's movement will be constrained within a designated "arena".

In this exercise, we will learn to:

- Set up and calibrate the rover's operating environment.
- Use localization to calculate the rover's position and orientation.

❖ Building an Arena for the Rover

The main requirement for the arena is that it must have a white background. The size of the arena is flexible, but it must fit within the field of view of your webcam. The localization algorithm requires that the webcam be placed at least 100 cm (Approximately 3.5 feet) above the arena.

Adjust the location of the webcam and the size and placement of the arena until it fits within the field of view as shown in the image below.

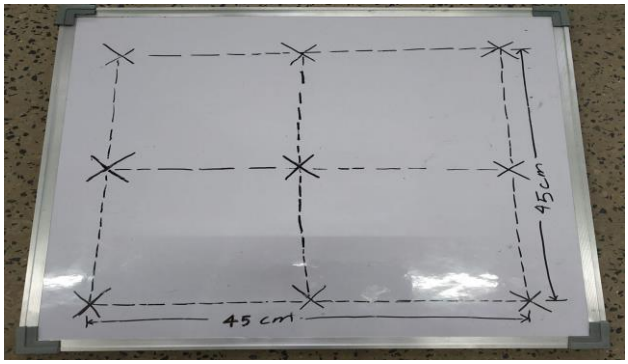


Fig.16: Arena for the robot to navigate

While build the arena, ensure that the white background extends beyond the actual size of the arena. Based on our experience, roughly a 5 cm padding on all sides is needed for the image processing algorithm. This will ensure that there is some buffer space between the floor and the arena. Mark the corners of the arena, centres of all edges and the overall centre point with an X that is visible from at least 100 cm away, as we saw in the picture. A black marker was used for this operation.

❖ Calibrate The Arena And The Environment

For the calibration to work, the algorithm needs to know the arena dimensions. Enter your arena's height and width

1. Enter Arena's Dimension

```
>> arenaHeight= 45;
>> arenaWidth = 45;
```

Note: Execute calibration process every time there are changes to the environment setup.

2. Initialize webcam and compute transformation that gives orthogonal view

Initialize the webcam and compute the transformation that gives the orthogonal view," the webcam will be initialized. Later, the transformation that gives the orthogonal view of the rover will be computed.
Now let's look at the transformation portion of this step. Typically, it is hard to place the webcam directly over the arena, and instead the camera is placed where it takes images from some downward angle. Image processing can be used to translate the image to a corrected version of itself, which will make it easier for our localization algorithm to calculate the position.

To compute the transformation of the image, click the four corners of the arena in the image.

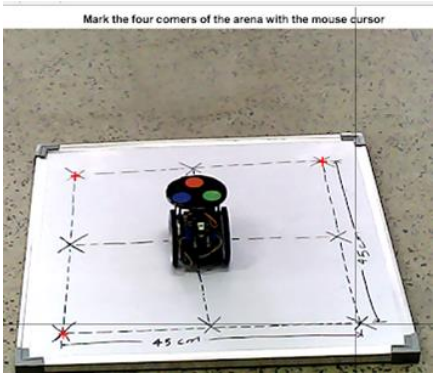


Fig. 17: Webcam Image

Important: Ensure that the rover is in the arena. If it is not, we need to execute this section of the code once more with the rover placed in the arena.

Because the webcam takes pictures from an angle, the rover's height is altered differently for various distances.

3. Compute the location offset for the robot

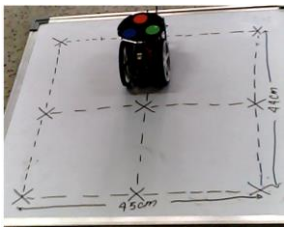


Fig. 18(a): Up

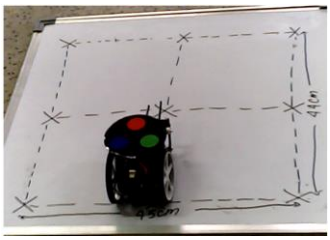


Fig.18(b):Down

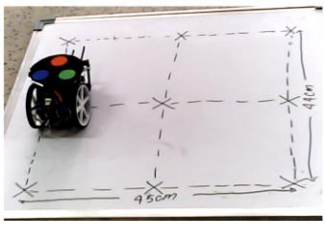


Fig.18(c):Left

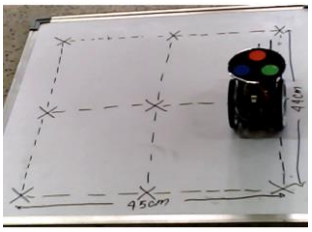
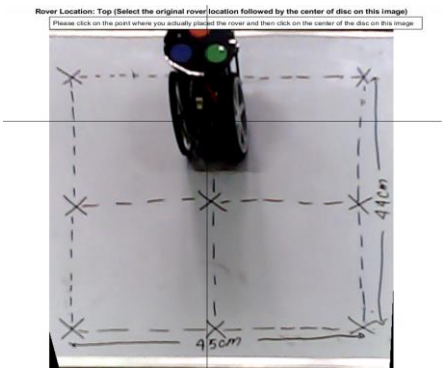


Fig.18(d): Right

Fig. 18: Capturing images of the robot on the four edges of the arena (Top, Bottom, Left and Right)



Click the points specified in the pop-up figures (center of robot and the center of disc) to compute offsets (**Example for top position shown**).

Fig. 19: Calibrating Location offset for top position

NOTE: The localization results depend heavily on these steps and can cause issues if the calibration is not done accurately.

Now we shall calibrate the colour threshold of the disc, to account for the lighting. On the image that appears, ensure that you click the circles in this order: Red first, Green next, and then Blue.

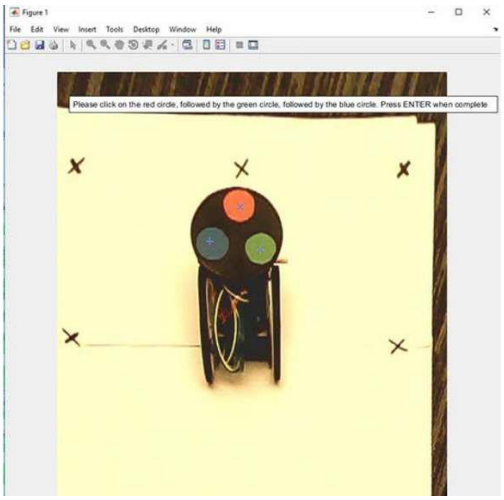


Fig. 20: Color Threshold for the Disc

With that, we have now calibrated the parameters to get an orthogonal view of the rover, corrected for any location offsets within that view and accounted for how lighting can affect the RGB values for colours. All the calibration data is saved and will be reused during the execution of the localization steps.

Similar steps would be performed for calibrating the object.

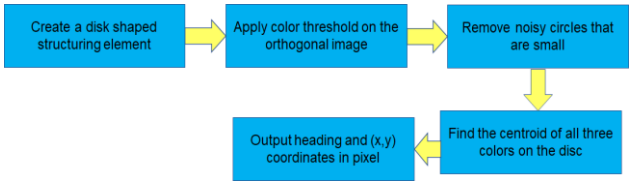
LOCALIZATION OF THE ROBOT AND THE OBJECT

Steps to perform Localisation using Webcam

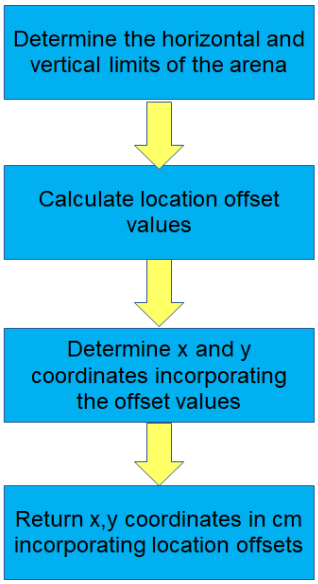
1. Prepare the MATLAB Workspace
2. Enter the dimensions of the arena and calculate conversion factor.
3. Initialise the webcam
4. (a) Get the location and heading of the robot
    - Calculate disc centre's (x,y) coordinates using **robotPosAng** function in pixels
    - Convert (x,y) coordinates to centimeters incorporating offsets using **getLocation** function

   (b) Get the location of the object
    - Calculate object's (x,y) coordinates using **objectPos** function
    - Convert (x,y) coordinates to centimeters incorporating offsets using **getLocation** function

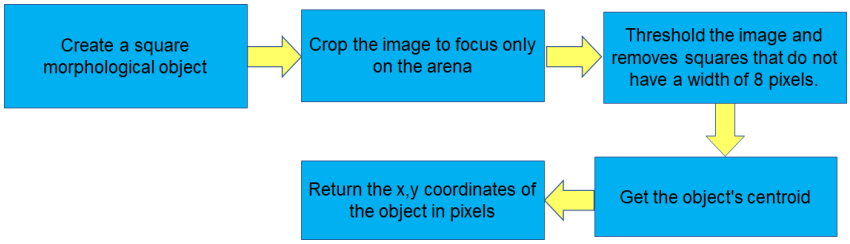
Overview of robotPosAng function



Overview of getLocation function



Overview of objectPos function



Result

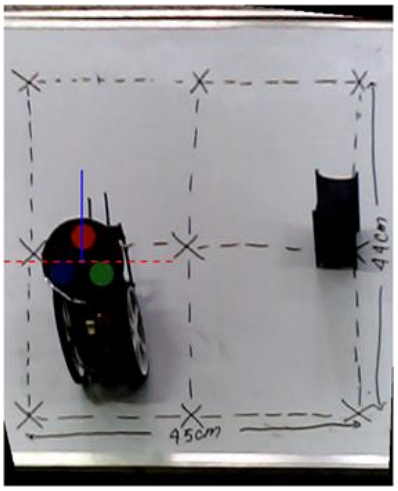
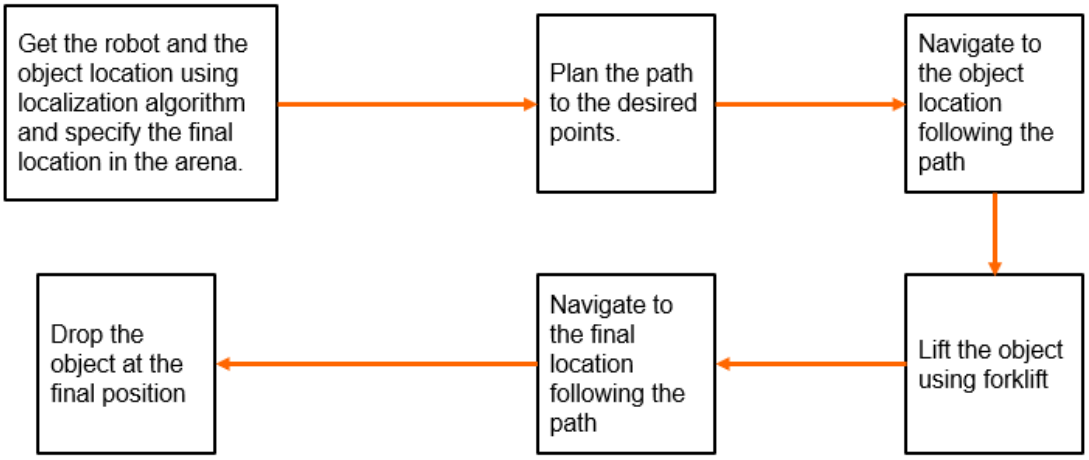


Fig. 21: Heading of the robot

Exercise 6: Navigate the Arena and Move the Object

Overview of the mobile robot navigation operation



Starting position of the robot and the object is obtained using vision-based localisation using webcam

Final position is assigned to in the arena

The distance to be travelled between two points are calculated using Euclidean Distance

The **heading of the robot towards a desired location** is calculated as follows:

- Determine which quadrant (Fig. 22) the object/location is in
- Determine the desired heading for the robot using the arctan function

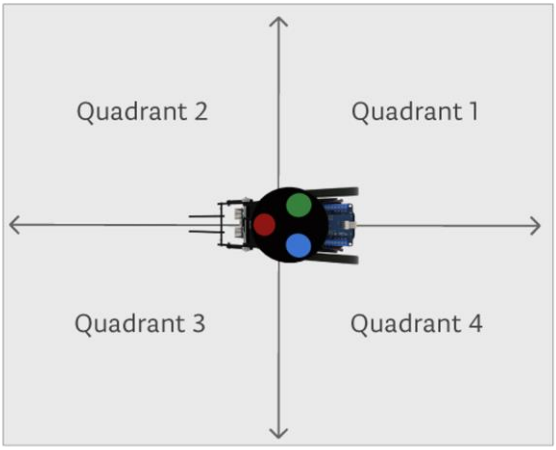


Fig.22: Quadrants of the Robot

Results

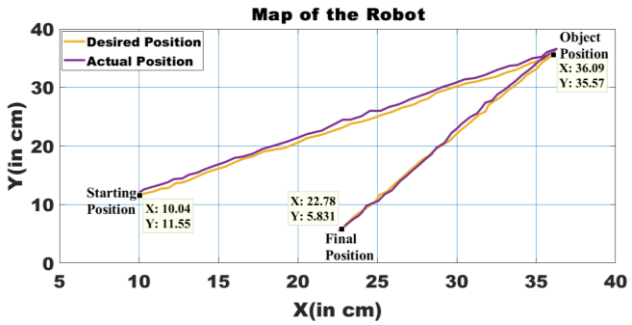


Fig. 23: Map of the mobile robot

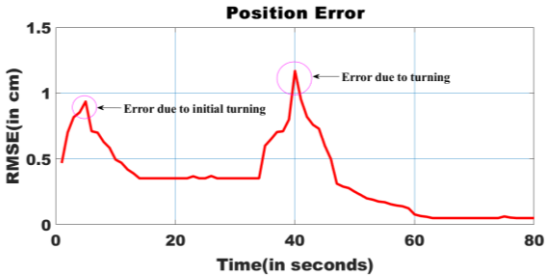


Fig.24: Position Error

Exercise 7: Line Following and Obstacle Avoidance

Line Following using PID Control

- Use ***5 IR sensors*** to get information about the position of the robot w.r.t. the track.
- Using the feedback from the IR sensor to control the velocity of the motor.
- To control the motor speeds, **PID controller** shall be used.

Sensor Reading	Bot Position
1 1 0 1 1	Centre Position
1 1 0 0 0	Right Position
0 0 0 1 1	Left Position

At max, only 2 sensors will be on the line.

For IR Sensor
Digital output **0** represents IR sensor is on **white line**
Digital output **1** represents IR sensor is on **black background**

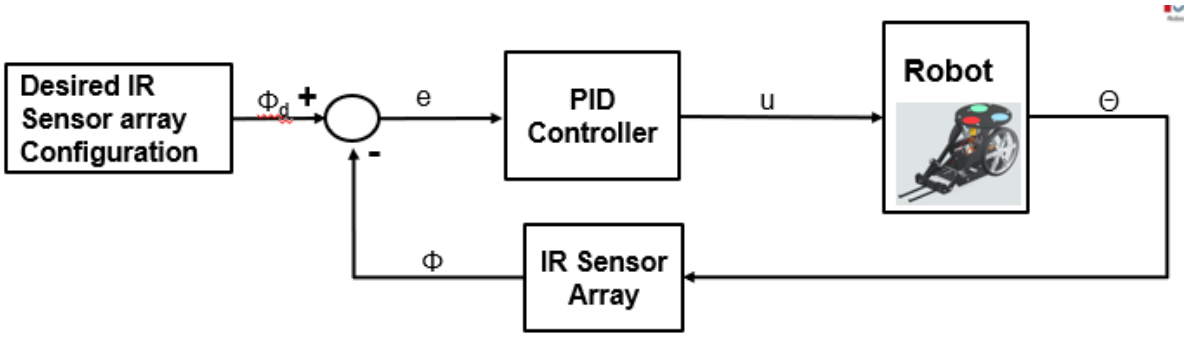
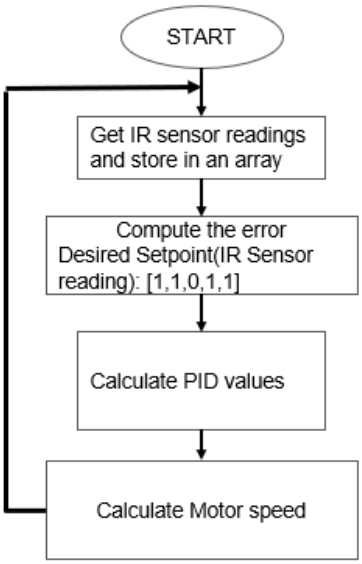


Fig.25: Block Diagram for Closed Loop Control

Calculation of positional error from sensor data[3]

$$Error = Reference\ input - \sum_{0}^{4} \frac{\text{sensor reading}(^{0}/_{1}) * \text{corresponding weight}}{number\ of\ sensors\ reading\ 0}$$

From Table 1, the **centre position** is taken as the **reference input**.
The sensors are numbered from the left to the right. **An error value of zero** refers to the robot being exactly on the center of the line. A **positive error** means that the robot has deviated to the right and needs to go turn left and a **negative error** value means that the robot has deviated to the left and needs to turn right.



Desired position of the IR sensor array:
Middle IR sensor must be at the white line

PID constants used to control the motor speed:
Kp = 25
Ki = 0.01
Kd = 0.0005

Condition:
Both Motor speed = 0, when
IR sensor array values = [1,1,1,1,1] or [0,0,0,0,0]

Results

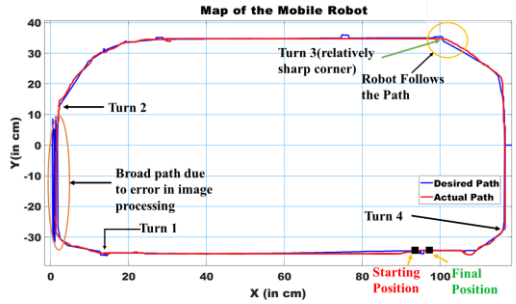


Fig. 26: Map of the Mobile Robot

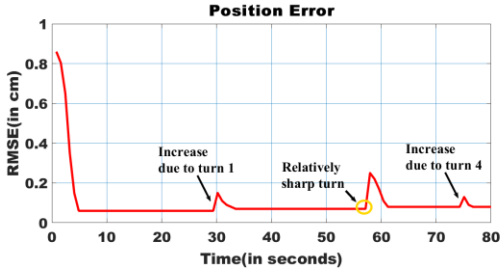
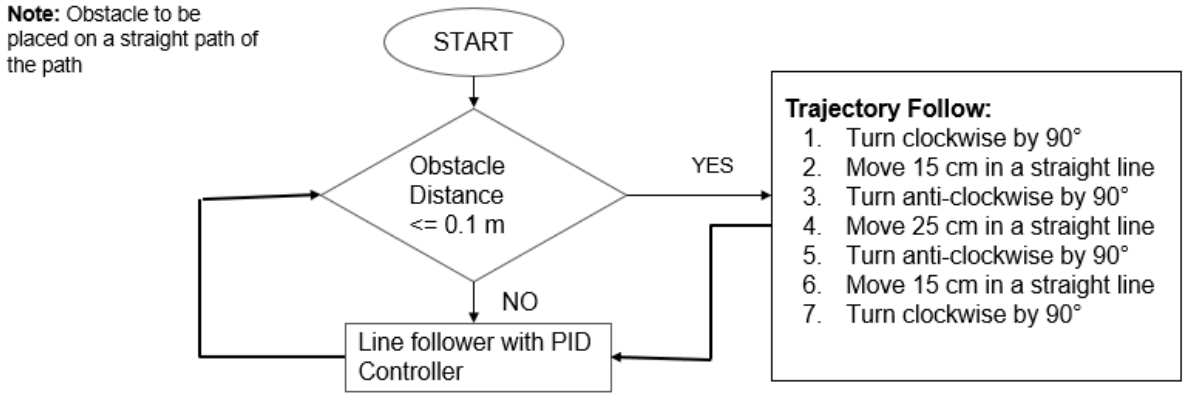


Fig. 27: Position Error

Line Following with Obstacle Avoidance Algorithm Flowchart



Results

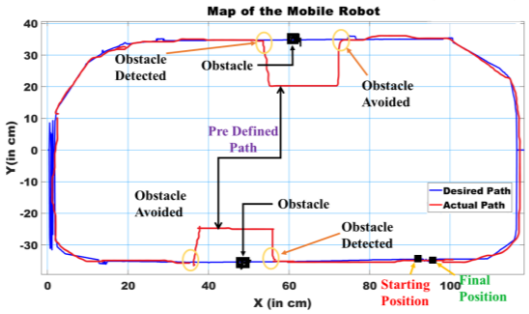


Fig.28: Map of the Mobile Robot

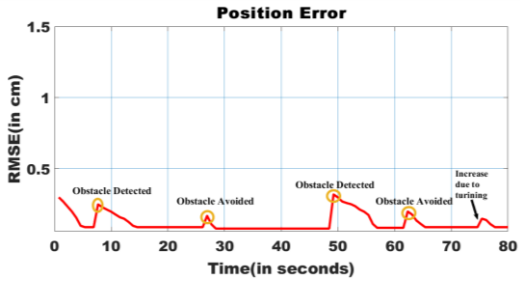


Fig. 29: Position Error

References

[1] (2013). Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework. Advances in Robotics & Automation. 02. 10.4172/2168-9695.1000107.

[2] Z. U. Abideen, M. B. Anwar and H. Tariq, "Dual Purpose Cartesian Infrared Sensor Array Based PID Controlled Line Follower Robot for Medical Applications," *2018 International Conference on Electrical Engineering (ICEE)*, 2018, pp. 1-7, doi: 10.1109/ICEE.2018.8566871.