Motion Planning for Mobile Robot

A report submitted in partial fulfilment of the requirements

for the degree of B.Tech. Mechanical Engineering

with specialization in Design and Manufacturing

by

J.Ravi Kiran

(Roll No: 119ME0033)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KURNOOL

April 2023

Indian Institute of Information Technology Design and Manufacturing Kurnool
Kurnool-518008, Andhra Pradesh, India

CERTIFICATE

This is to certify that the Project titled **"Motion Planning for Wheeled Mobile Robot"** submitted by **J.Ravi Kiran** has/have been to the **Indian Institute of Information Technology Design and Manufacturing Kurnool** in partial fulfillment of requirements for the degree of **Bachelor of Technology** is a record of bonafied project work carried out by him/her under my/our supervision. The results embodied in the report have not been submitted to any other university or institute for the award of any degree or diploma.

Supervisor
Dr. Mani Prakash, PhD
Assistant
Professor
Department of Mechanical Engineering

Head of Department
Dr. M. Pullarao
Assistant Professor
Department of Mechanical Engineering

Date:

Place: Indian Institute of Information Technology Design and Manufacturing Kurnool

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited or from whom proper permission has not been taken when needed. The work presented in this report would have not been possible without my close association with many people. I take this opportunity to extend my sincere gratitude and appreciation to all those who made this project work possible.

Date:

Place: Department of Mechanical Engineering, IIITDM Kurnool

________________________________ (Signature)

________________________________ (Name of the Student)

________________________________ (Enrollment Number)

Certificate

I, **J.Ravi Kiran**, with Roll No: **119ME0033** hereby declare that the material presented in the Project Report titled **Motion Planning for Mobile Robot** represents original work carried out by me in the **Department of Mechanical Engineering** at the **Indian Institute of Information Technology, Design and Manufacturing, Kurnool** during the years **2022–2023**. With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student’s Signature

In my capacity as supervisor of the above-mentioned work, I certify that the work presented in this Report is carried out under my supervision, and is worthy of consideration for the requirements of B.Tech. Project work.

Advisor’s Name: **Dr. Mani Prakash, Ph.D** Advisor’s Signature

Abstract

I **J.Ravi Kiran 119ME0033** as a 4^{th} year Mechanical student of IIITDM Kurnool am undertaking a final year project under the super vision of **Dr. Mani Prakash, Ph.D**, in the field of mobile robotics on the wheeled mobile topic.The project objective is to make a full fledged motion planning of the wheeled mobile robot, I have studied the theory and concepts of accompanied with their application like how to simulate in MATLAB with desired angular velocities or required force of the wheels, and finally I made model for Differential Drive wheeled mobile robot that can accomplish the task of starting from a point and reaching the desired destination by reading the map with a genetic algorithm that includes Probabilistic Road Map followed by motion control with the the help of MATLAB and by making use of inverse and forward kinematic equations of differential drive all along with avoiding static obstacle's and another that will be performing the same but by following the given route.Along with the usage of sensors and and how the raw data which was sensed can be put to use like prediction and perception that can be made good use to the mobile robot .

Contents

References **81**

List of Figures

List of Tables

Abbreviations

FEA	**F**inite **E**lement **A**nalysis
FEM	**F**inite **E**lement **M**ethod
LVDT	**L**inear **V**ariable **D**ifferential **T**ransformer
RC	**R**einforced **C**oncrete
PRM	**P**robabilistic **R**oad **M**ap

Symbols

η	generalized coordinates
$\dot{\eta}$	derivative of generalized coordinates
ζ	velocity input commands
τ	torque
ω	angular velocity inputs matrix
κ	force inputs matrix

For/Dedicated to/To my…

Chapter 1

Introduction

1.1 Mobile robot

A mobile robot is an automatic machine that is capable of locomotion(effector its move due to its action on environment) Mobile robot covers that roll, walk, fly or swim i.e,land-based, air-based, water-based,..etc Types of land-based mobile robots wheeled, legged, tracked, hybrid Wheels are more suitable and effective on flat surface than legs along with more stability Legged mobile robot requires higher degree of freedom which make its mechanically complex

1.2 Robot Kinematics

Kinematics is the study of motion that concerned with the motion of objects without reference to the forces which cause the motion

Forward differential kinematics:-For given velocity input commands, finding the derivatives of generalized coordinates (finding the system's motion).**Simulating or analyzing the system in velocity level.**

$$\dot{\eta} = J(\psi)\zeta$$

Inverse differential kinematics:-For the desired (given) derivatives of generalized coordinates (or given position trajectory), finding the corresponding velocity input commands.**Controlling the system in velocity level.**

$$\zeta = J^{-1}(\psi)\dot{\eta}$$

1.3 Robot Dynamics

Dynamics is the study of system that undergo change of state as time evolves. In case of mechanical system such as robots, the change of states involves motion
Forward dynamics:- For a given input vector τ, calculating the resulting motion of the robot, that is $\eta, \dot{\eta}, \ddot{\eta}$,**problem of simulating or analyzing the robot**
Inverse dynamics:- For a given desired trajectory $\eta, \dot{\eta}, \ddot{\eta}$,find the required input vector,τ . **problem of controlling the robot**

1.4 Types of Wheels

- Conventional Wheels
  - **Fixed Wheels** - Offers infinite resistance to lateral force
  - **Steered or Rotatable wheels**
- Non-conventional Wheels
  - **Omni-directional wheels** - Offers almost zero friction to lateral force
  - **Mecanum wheels** - Offers no friction to lateral force and also uses that force with the help of passive wheels

Figure 1.1: Types of wheels from left to right

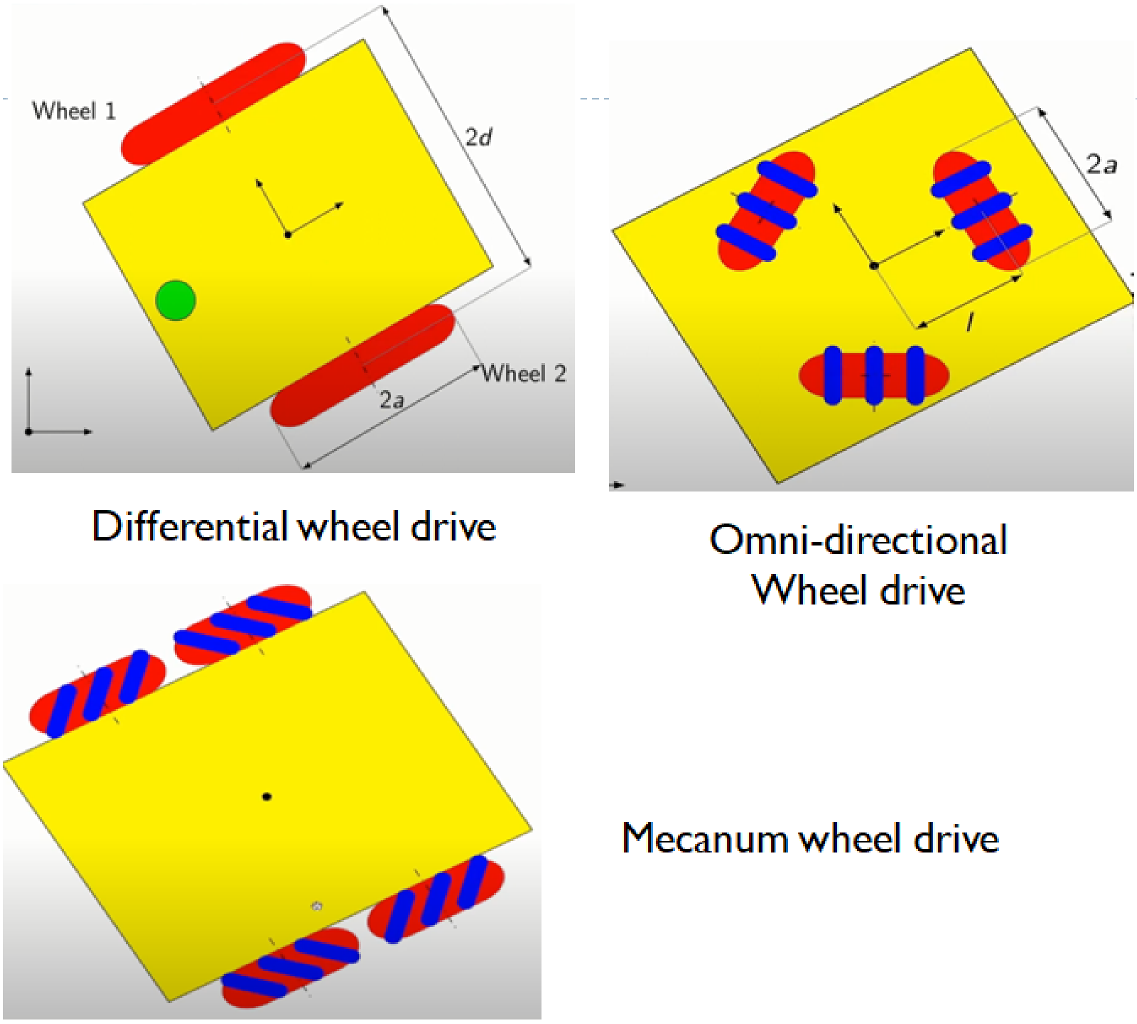


Figure 1.2: Types of wheels drives

1.5 Sensing and Perception

The data or information that is existing in the Real World Environment around the mobile robot is processes as below

1. **Perception** is of translating sensory impressions into a coherent and unified view of the world around. This is achieved by following the below steps
    - **Feature extraction**- With the use of sensors the required data will be extracted. The sensors are classified as below
        - **Exteroceptive sensors**- These acquire information from the robot s environment
        - **Proprioceptive sensors**- They are used to measure the internal state of the system
        - **Active sensors**- Energy is emitted to the environment
        - **Passive sensors**- Energy is absorbed from the environment
    - **Matching**-The data that has been acquired will be associated with the existing data.
    - **Updating**- If there are any kind of changes in the data between the existing one and the newly obtained it will be updated
    - **Model Integration**- Now with the set of updated data the further steps of mobile robot activities will be planned along with a list of instruction and the fesibility will be checked
    - **Model**- The model for the robot will be made ready to test in practical and to achieve the required goal
    - **Prediction**- Using the model in practice or by using it's simulation the performance will be predicted and the chances of it's perfect achivement will be updated to the "Matching" stage and the process will be repeated
2. **Localization Map Building**- Along with the help of knowledge data base and the Environmental Model Local Map a Localization Map will be builded
3. **Cognition Path Planning**- On using the map the position of the robot will be recognized in the global map with the guidance of the mission commands the requiered path will be established

4. **Motion Control-** By giving the actuator commands the path execution will be accomplished and the results will be seen in the real world completing the loop back to real world

1.5.1 Commonly used Sensors

A:active, P:passive, PC:proprioceptive, EC:extroceptive

Common Sensors			
General classification(typical use)	Sensor Systems	PC or EC	A or P
Tactile sensors	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Non-contact proximity sensors	EC	A
Wheel motor sensors(wheel motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoder	PC	A
	Magnetic encoder	PC	A
	Inductive encoder	PC	A
	Capacitive encoder	PC	A
Heading sensors	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P

Table 1.1: Classifying Common Sensors

1.5.2 Localization of Wheeled mobile robot

In this phase the Encoders will be giving the data about the status of the interior of the robot combining it with the map data base the position of the robot will be predicted which can be call as odometry. Now on comparison of the predicted position with present position obtained by observation through the sensors if there is any difference it will be updated to the predicted position and process repeats for the next position until it reached to the desired goal.

Chapter 2

Kinematics Algorithms

2.1 Kinematics

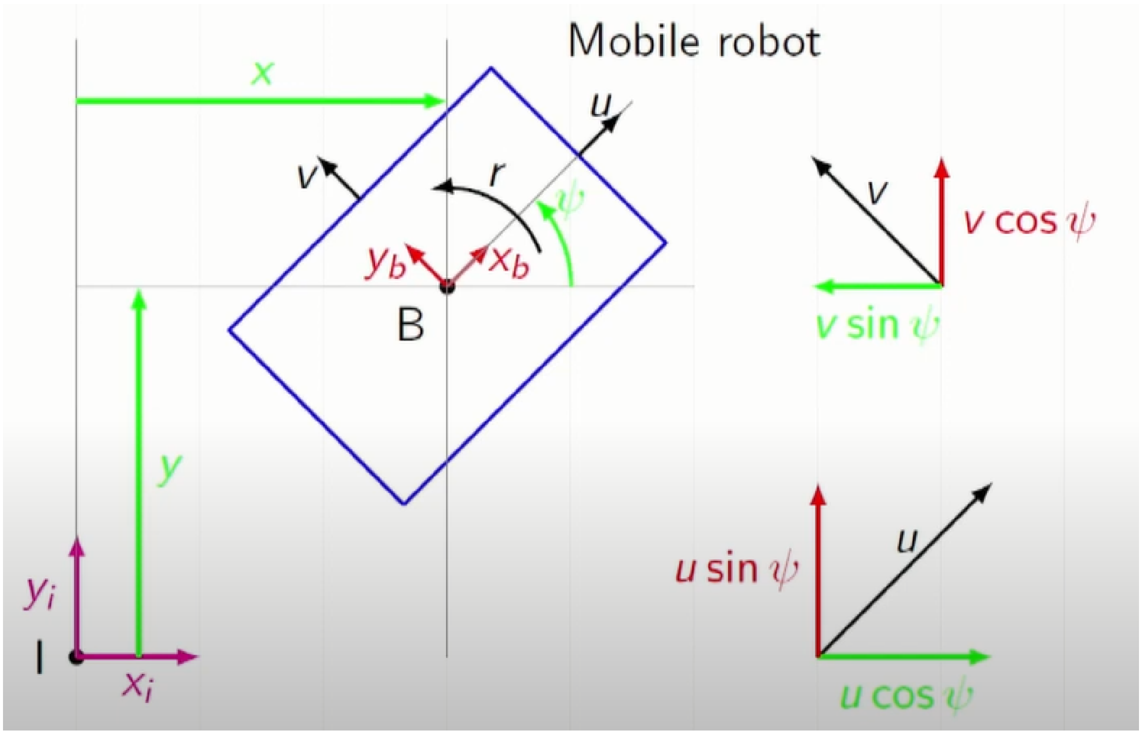


Figure 2.1: Kinematic resultants

Where,x: Forward displacement of the mobile robot w.r.t. I y: Lateral displacement of the mobile robot w.r.t. I ψ: Angular displacement of the mobile robot w.r.t. I u: Forward velocity of the mobile robot w.r.t. B v: Lateral velocity of the mobile robot w.r.t. B r: Angular velocity of the mobile robot w.r.t. B.

Writing the resultant velocities in the form of matrix we get;

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} u\cos\psi - v\sin\psi \\ u\sin\psi + v\cos\psi \\ r \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

$$\Rightarrow \dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\psi})\boldsymbol{\zeta}$$

Where,

- $\dot{\eta}$ -derivatives of generalized coordinates,
- $J(\psi)$ -Jacobian matix which maps input of coordinates,
- ζ -velocity input commands

Again velocity can be written as

$$v = r\omega \Rightarrow \boldsymbol{\zeta} = \boldsymbol{W}\boldsymbol{\omega}$$

Where

- W-wheel input or configuration matrix,
- ω-wheel angular velocity

Wheel configuration matrix is different for different wheel drives as shown in Fig 2.2

The angular velocity of the wheels are given and simulations are made.Using Euler method $x_{i+1} = x_i + \dot{x}_i\delta t$ and replacing x and $\dot{x}$ with η and $\dot{\eta}$. Giving the $\boldsymbol{\zeta} = \boldsymbol{W}\boldsymbol{\omega}$ based on the

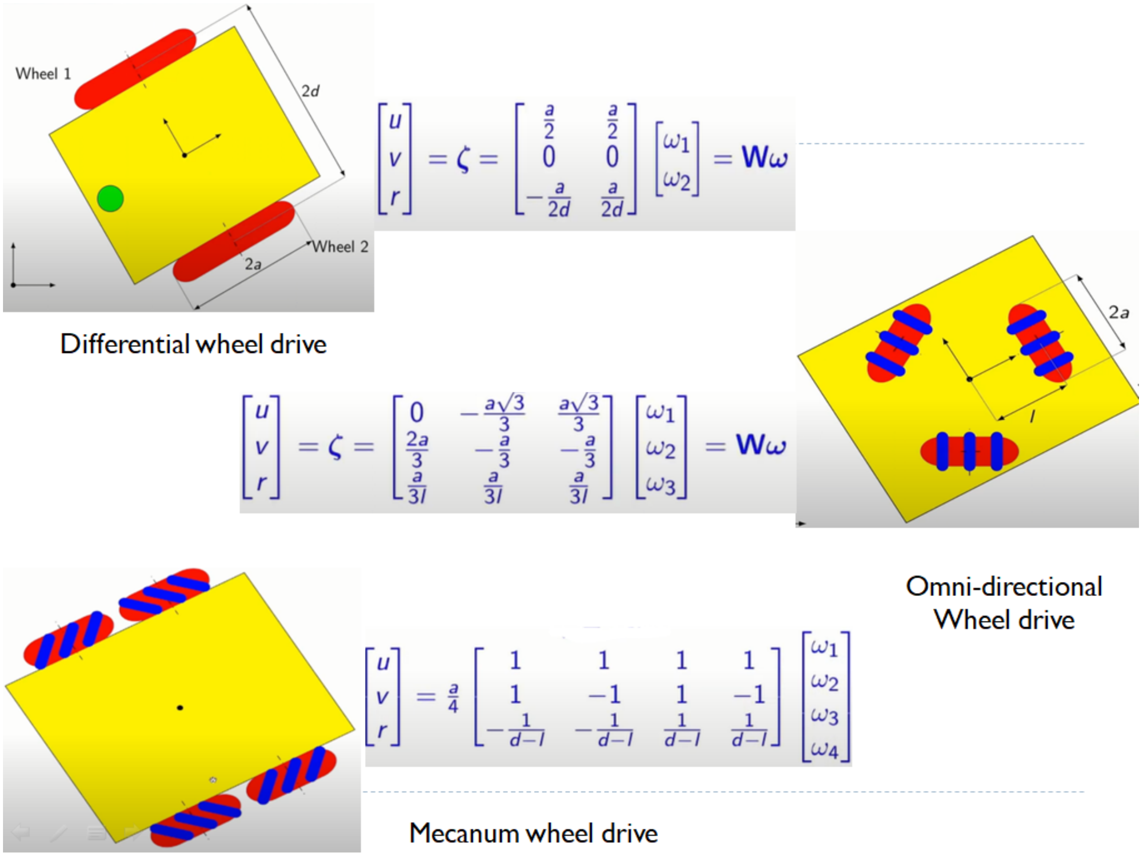


Figure 2.2: Wheel configuration matrix

wheel configuration And assigning the angular velocity of each wheel forward kinematics can be achieved

2.1.1 Kinematic animation

The matlab code to this process is available in Appendix:A
The algorithm goes as follows:

- Setting the simulation parameters of time
- Giving the initial conditions here position (x,y,psi)
- Making Jacobian matrix $J(\psi)$ using psi
- Inputs given ζ = (u,v,r)

- Applying $\dot{\eta_i} = J(\psi) * \zeta_i$
- Euler's method $\eta_(i+1) = \eta_i + dt * \dot{\eta_i}$
- plotting animation using robot co-ordinated and fill to make robot object. Position is gained by multiplying rotation matrix with robot co-ordinates

2.1.2 Inverse kinematic animation

The matlab code to this process is available in Appendix:B
The algorithm goes as follows:

- Setting the simulation parameters of time
- Giving the initial conditions here position (x,y,psi)
- Making Jacobian matrix $J(\psi)$ using psi
- Given the desire generalized coordinates η_d
- Finding input velocities (u,v,r) using $\zeta = inv(J(\psi) * \eta_d$
- Time derivative of generalized coordinates $\dot{\eta_i} = J_psi * \zeta_i$
- State update using Euler's method $\eta_(i+1) = \eta_i + dt * \dot{\eta_i}$
- plotting animation using robot co-ordinated and fill to make robot object. Position is gained by multiplying rotation matrix with robot co-ordinates

2.1.3 Differential wheel drive animation

Given the angular velocities of each wheel doing forward kinematics i.e, deriving the input velocities from angular velocities and making the motion.The matlab code to this process is available in Appendix:C
The algorithm goes as follows:

- Setting the simulation parameters of time
- Giving the initial conditions here position (x,y,psi)
- Making Jacobian matrix $J(\psi)$ using psi
- Assingning the left & right wheel angular velocities $\omega = \omega_1, \omega_2$ respectively
- Finding input velocities using $\zeta = W * omega$, where W is the wheel configuration matrix
- Finally applying Euler's method $\eta_(i + 1) = \eta_i + dt * \dot{\eta_i}$
- plotting animation using robot co-ordinated and fill to make robot object. Position is gained by multiplying rotation matrix with robot co-ordinates

2.1.4 Omini wheel animation

Given the angular velocities of each wheel doing forward kinematics i.e, deriving the input velocities from angular velocities and making the motion.The matlab code to this process is available in Appendix:D
The algorithm goes as follows:

- Setting the simulation parameters of time and vehicle i.e,radius of wheel and base distance
- Giving the initial conditions here position (x,y,psi)
- Making Jacobian matrix $J(\psi)$ using psi
- Assingning the three wheel angular velocities $\omega = \omega_1, \omega_2, \omega_3$ respectively
- Finding input velocities using $\zeta = W * omega$, where W is the wheel configuration matrix
- Finally applying Euler's method $\eta_(i + 1) = \eta_i + dt * \dot{\eta_i}$
- plotting animation using robot co-ordinated and fill to make robot object. Position is gained by multiplying rotation matrix with robot co-ordinates

2.1.5 Mecanum wheel animation

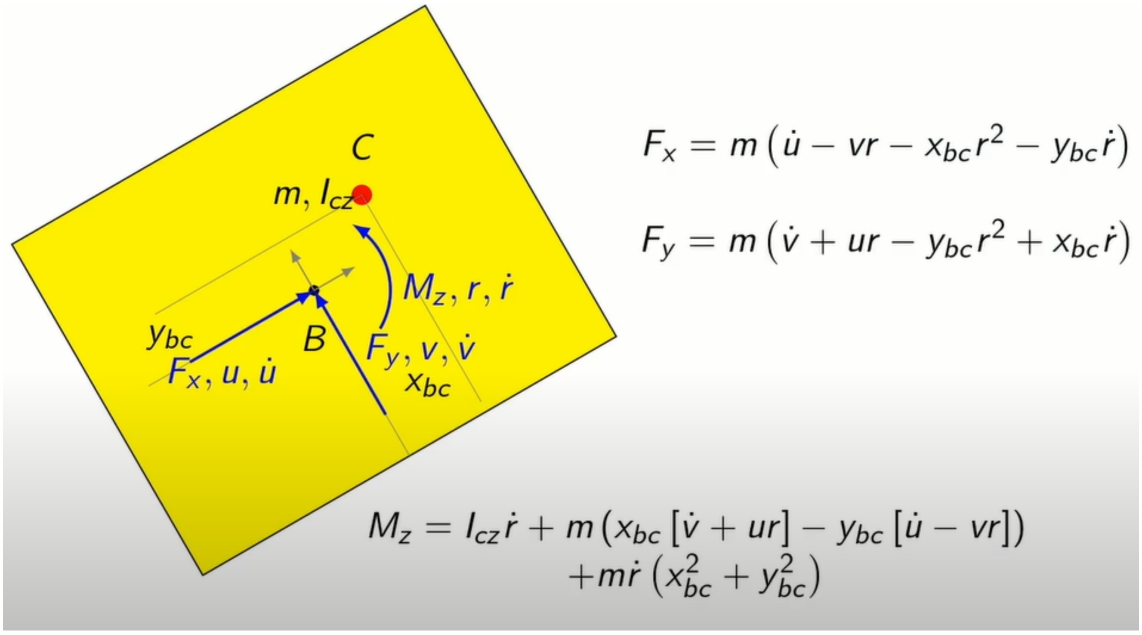
Given the angular velocities of each wheel doing forward kinematics i.e, deriving the input velocities from angular velocities and making the motion.The matlab code to this process is available in Appendix:E
The algorithm goes as follows:

- Setting the simulation parameters of time and radius of wheel and distance wheels, wheel & frame
- Giving the initial conditions here position (x,y,psi)
- Making Jacobian matrix $J(\psi)$ using psi
- Assingning the three wheel angular velocities $\omega = \omega_1, \omega_2, \omega_3, \omega_4$ respectively
- Finding input velocities using $\zeta = W * omega$, where W is the wheel configuration matrix
- Finally applying Euler's method $\eta_(i + 1) = \eta_i + dt * \dot{\eta}_i$
- plotting animation using robot co-ordinated and fill to make robot object. Position is gained by multiplying rotation matrix with robot co-ordinates

Chapter 3

Dynamics Algorithms

3.1 Forward Dynamics



Forward dynamics:- For a given input vector τ, calculating the resulting motion of the robot, that is η, $\dot{\eta}$, $\ddot{\eta}$,**problem of simulating or analyzing the robot**

Where,

- **D**-Inertia matrix,

$$\begin{bmatrix} m\dot{u} - mvr - mx_{bc}r^2 - my_{bc}\dot{r} \\ m\dot{v} + mur - my_{bc}r^2 + mx_{bc}\dot{r} \\ \left(I_{cz} + m\left(x_{bc}^2 + y_{bc}^2\right)\right)\dot{r} + m\left(x_{bc}\left[\dot{v} + ur\right] - y_{bc}\left[\dot{u} - vr\right]\right) \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix}$$

$$\begin{bmatrix} m & 0 & -my_{bc} \\ 0 & m & mx_{bc} \\ -my_{bc} & mx_{bc} & I_{cz} + m\left(x_{bc}^2 + y_{bc}^2\right) \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} -mr\left(v + x_{bc}r\right) \\ mr\left(u - y_{bc}r\right) \\ mr\left(x_{bc}u + y_{bc}v\right) \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix}$$

$$\mathbf{D}\dot{\boldsymbol{\zeta}} + \mathbf{n}\left(\boldsymbol{\zeta}\right) = \boldsymbol{\tau}$$

Figure 3.1: Dynamic resultants

- $\dot{\zeta}$ -derivatives of generalized coordinates,
- $n(\zeta)$ -other matirx
- $\boldsymbol{\tau}$ -Torque

Again torque can be write as

$$\tau = rF \Rightarrow \boldsymbol{\tau} = \boldsymbol{\Gamma F}$$

Where,

- Γ -mapping,
- k -wheel force,
- $\boldsymbol{\tau}$ -Torque

w.k.t, from kinematic relations $\dot{\eta} = J(\psi)\zeta$ Applying Euler method to ζ i.e, $\zeta_{i+1} = \zeta_i + \dot{\zeta}dt$ From the dynamic relations we get $\dot{\zeta} = D^{-1}(\tau - n(\zeta))$ Again we can write τ as $\tau = \Gamma k$ -mapping & k-wheel force Γ is different for different wheel drives For ex: differential drive $\begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & d \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$, d-distance b/t wheels, $F_1 \& F_2$-force of wheel 1 & 2

3.1.1 Dynamic model animation

The determination of motion from the given force of each wheel.The matlab code for achieving it is available in Appendix:F
The algorithm goes as follows:

- Setting the parameters of time
- Giving the initial conditions of position η
- Assigning the parameters of robot i.e,mass & Ineria along with co-ordinates of centre of mass
- Making Inertia matrix(D),other matrix (n_v) and Jacobian matrix$J(\psi)$
- Giving the input force vector τ
- Finding velocity ζ by applying $\dot{\zeta} = D^{-1} * (tau_i - n_v)$ and $\zeta_{i+1} = zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi * (\zeta_i + dt * \dot{\zeta})$
- Animation is made by fill and finding position of robot which is obtained via multiplying robot co-ordinates with rotation matrix $R(\psi)$

3.1.2 Differential wheel dynamic

The determination of motion from the given force of each wheel.The matlab code for achieving it is available in Appendix:G
The algorithm goes as follows:

- Setting the parameters of time
- Giving the initial conditions of position η
- Assigning the parameters of robot i.e,mass & Ineria along with co-ordinates of centre of mass

- Making Inertia matrix(D),other matrix (n_v) and Jacobian matrix$J(\psi)$
- Giving the input force vector $\kappa = F_1, F_2$
- Determining $\tau = \Gamma * \kappa$, where Γ is the wheel input matrix
- Finding velocity ζ by applying $\dot{\zeta} = D^{-1} * (tau_i - n_v)$ and $\zeta_{i+1} = zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi * (\zeta_i + dt * \dot{\zeta})$
- Animation is made by fill and finding position of robot which is obtained via multiplying robot co-ordinates with rotation matrix $R(\psi)$

3.1.3 Omini wheel dynamics

The determination of motion from the given force of each wheel.The matlab code for achieving it is available in Appendix:H
The algorithm goes as follows:

- Setting the parameters of time
- Giving the initial conditions of position η
- Assigning the parameters of robot i.e,mass & Ineria along with co-ordinates of centre of mass along with wheel configuration parameters
- Making Inertia matrix(D),other matrix (n_v) and Jacobian matrix$J(\psi)$
- Giving the input force vector $\kappa = F_1, F_2, F_3$
- Determining $\tau = \Gamma * \kappa$, where Γ is the wheel input matrix
- Finding velocity ζ by applying $\dot{\zeta} = D^{-1} * (tau_i - n_v)$ and $\zeta_{i+1} = zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi * (\zeta_i + dt * \dot{\zeta})$
- Animation is made by fill and finding position of robot which is obtained via multiplying robot co-ordinates with rotation matrix $R(\psi)$

3.1.4 mecanum wheel dynamics

The determination of motion from the given force of each wheel.The matlab code for achieving it is available in Appendix:H
The algorithm goes as follows:

- Setting the parameters of time
- Giving the initial conditions of position η
- Assigning the parameters of robot i.e,mass & Ineria along with co-ordinates of centre of mass along with wheel configuration parameters
- Making Inertia matrix(D),other matrix (n_v) and Jacobian matrix$J(\psi)$
- Giving the input force vector $\kappa = F_1, F_2, F_3, F_4$
- Determining $\tau = \Gamma * \kappa$, where Γ is the wheel input matrix
- Finding velocity ζ by applying $\dot{\zeta} = D^{-1} * (tau_i - n_v)$ and $\zeta_{i+1} = zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi * (\zeta_i + dt * \dot{\zeta})$
- Animation is made by fill and finding position of robot which is obtained via multiplying robot co-ordinates with rotation matrix $R(\psi)$

Chapter 4

Differential Drive Mobile Robot

4.1 Forward Kinematics

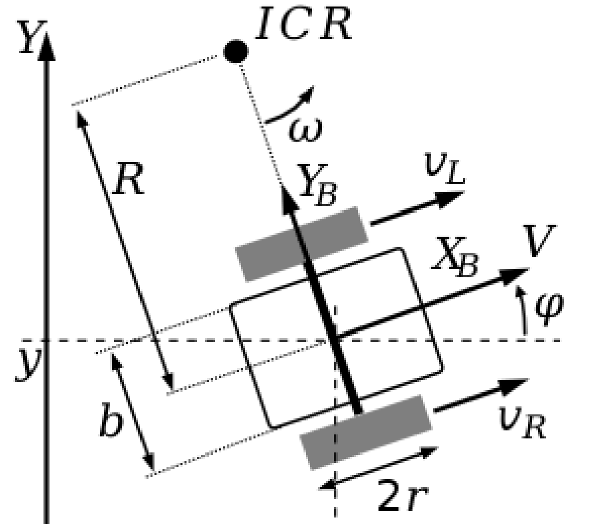


Figure 4.1: Differential Drive Kinematics of a Wheeled Mobile Robot

For given velocities of right and left wheels finding the robots velocity

We know that in general $v = r\omega$ from the figure r for right and left wheels are $R + (b/2)$ and $R - (b/2)$ respectively where b is the base of the robot.

Hence we can write

$$V_R = (R + (b/2))\omega_{rob}$$

$$V_L = (R - (b/2))\omega_{rob}$$

$$\implies V_{rob} = (V_L + V_R)/2 = (\omega_L + \omega_R)/(2 * r)$$

From V_R & V_L we get,

$$\omega_{rob} = (V_R - V_L)/b$$

$$\&R = V_{rob}/\omega_{rob}$$

$$\implies R = b(V_L + V_R)/2(V_R - V_L)$$

4.2 Inverse Kinematics

For desired ω_{rob} & V_{rob} i.e, $V_{rob} = (V_L + V_R)/2$ & $\omega_{rob} = (V_R - V_L)/b$ we need to find individual wheel velocities. On solving these both we get,

$$V_R = V_{rob} + \omega_{rob}(b/2) \implies \omega_R = (V_{rob} + \omega_{rob}(b/2))/r$$

$$V_L = V_{rob} - \omega_{rob}(b/2) \implies \omega_L = (V_{rob} - \omega_{rob}(b/2))r$$

Chapter 5

Fixed path Differential Drive Motion

5.1 Key functions & words

- **inflate -** inflate (map , radius) inflates each occupied position of the map by the radius given in meters . radius is rounded up to the nearest cell equivalent based on the resolution of the map. Every cell within the radius is set to true (1)
- **waypoints -** the path to follow
- **controllerPurePursuit -** takes desired linear and angular velocities along with look ahead distance of pursuiter
- **controller -** [vel , angvel] = controller (pose) processes the vehicle ’ s position and orientation , pose , and outputs the linear velocity , vel , and angular velocity , angvel .
- **visualization -** used to visualize the map

5.2 Algorithm

The simulation of diff drive wheeled mobile robot for an example map from matlab and self-created second map through a given fixed path. The matlab codes for these are available in Appendix J and Appendix K. And the matlab code for the self-made map can be viewed in Appendix L

1. The parameter for the simulation are assigned
2. Wheeled robot's parameters are given
3. Initial pose/eta is allotted
4. Map is loaded and inflation in made so that obstacle's can't enter robot's cartesian space
5. Waypoints will be assigned
6. 2D visuailization of the map is made with the help of visualizer
7. controllerPurePursuit takes desired linear and angular velocities along with look ahead distance of pursuiter
8. controller function is used to find the reference linear and angular velocities of the pursuiter with respect to the pose
9. Inverse kinematics is used to find the individual wheel's velocities with respect to reference velocity found before as desired robots velocity
10. Forward kinematics is made to make the robot move using the individual wheel velocities found in the previous step
11. Visualization of motion with respect to the eta/pose through waypoints with are given rate

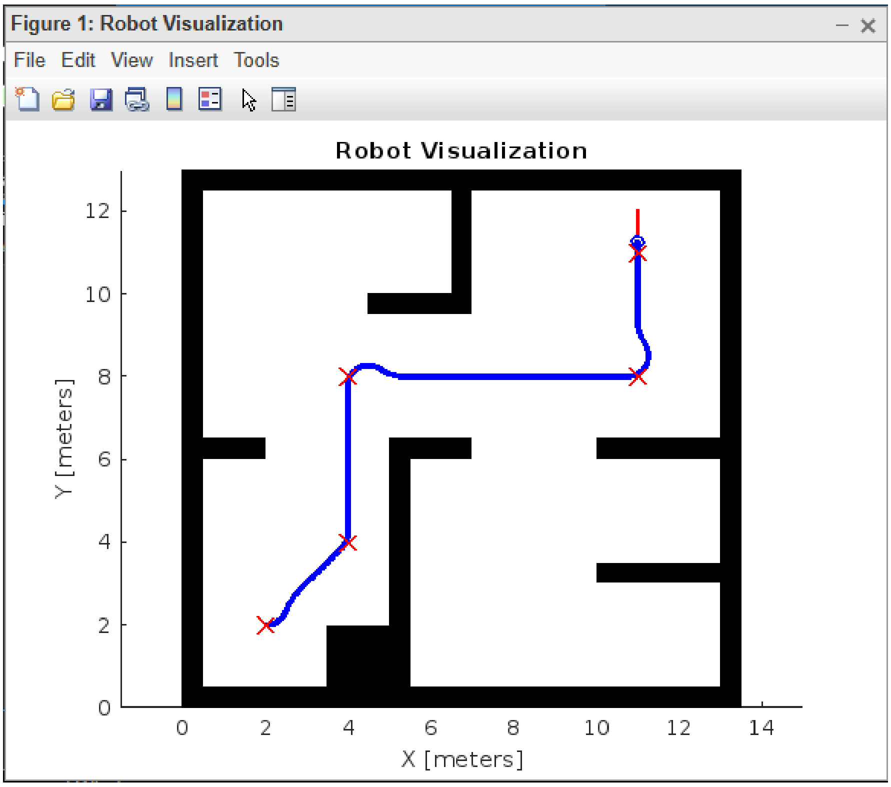


Figure 5.1: Fixed path motion for map1

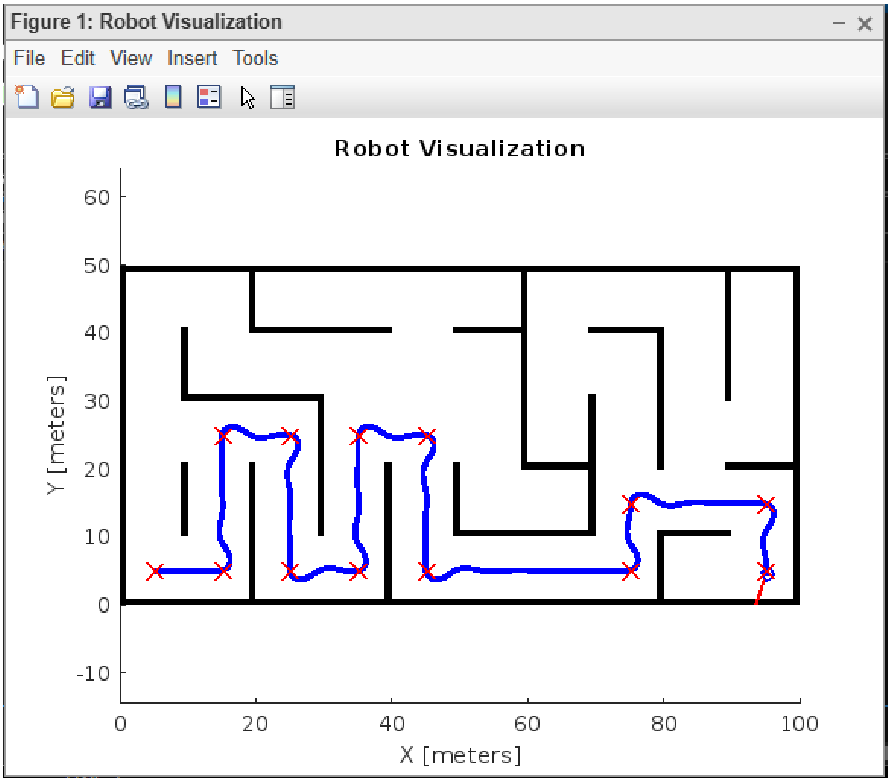


Figure 5.2: Fixed path motion for map2

Chapter 6

Auto path Differential Drive Motion

6.1 Key functions & Defintions

- **Probabilistic Road Map(PRM) -** Random node are made in the space and then they are connected making a network of paths
- **A* algorithm -** uses a $function f(n) = g(n) + h(n)$ to find the optimal from start and goal points, where $g(n)$ & $h(n)$ are the function to find the distance to reach from start to n^{th} node and n^{th} node to goal
- **planner -** used to access the PRM function
- **findpath -** uses A* algorithm to optimize path

6.2 Algorithm

The simulation of auto drive wheeled mobile robot for an example map from matlab and self-created second map through a optimized path generated. The matlab codes for these are available in Appendix M and Appendix N. And the matlab code for the self-made map can be viewed in Appendix L

1. The parameter for the simulation are assigned

2. Wheeled robot's parameters are given
3. Initial pose/eta is allotted
4. Map is loaded and inflation in made so that obstacle's can't enter robot's cartesian space
5. planner is used to initialize PRM to the map with given number of node and the distance between the nodes
6. Start and goal point are allotted
7. A* algorithm is used to find the optimal waypoints to reach the goal
8. 2D visuailization of the map is made with the help of visualizer
9. controllerPurePursuit takes desired linear and angular velocities along with look ahead distance of pursuiter
10. controller function is used to find the reference linear and angular velocities of the pursuiter with respect to the pose
11. Inverse kinematics is used to find the individual wheel's velocities with respect to reference velocity found before as desired robots velocity
12. Forward kinematics is made to make the robot move using the individual wheel velocities found in the previous step
13. Visualization of motion with respect to the eta/pose through waypoints with are given rate

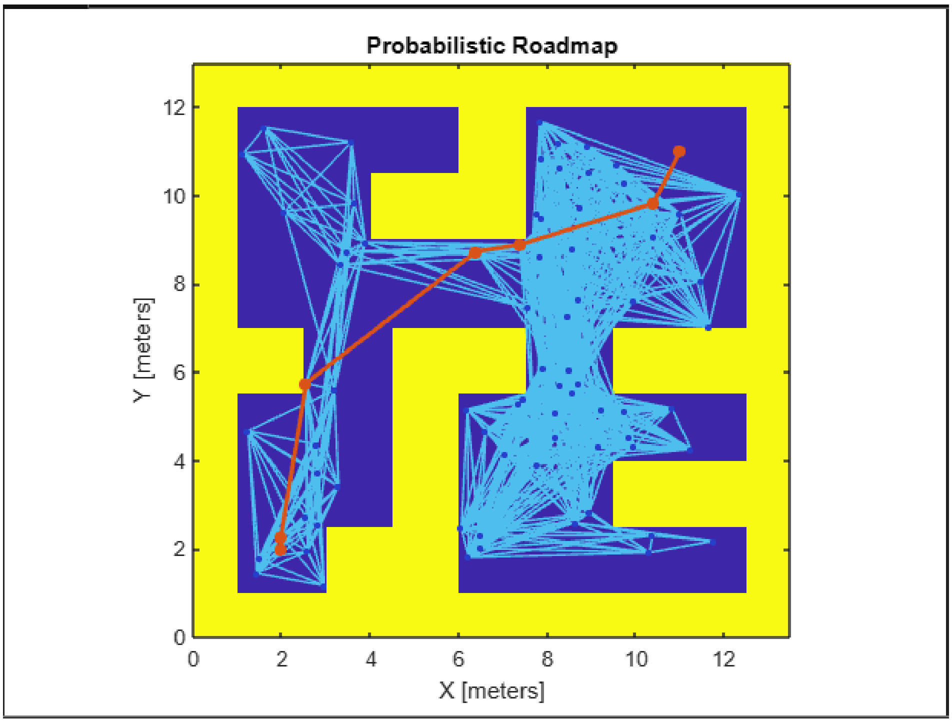


Figure 6.1: PRM path planning for map1

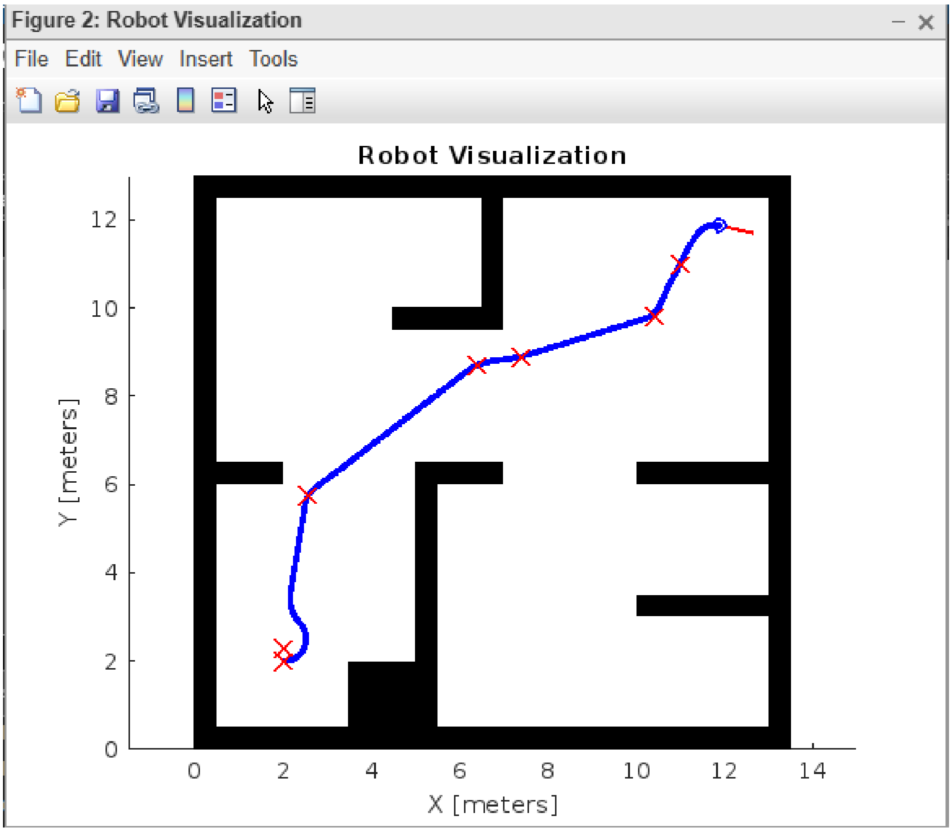


Figure 6.2: Auto motion for map1

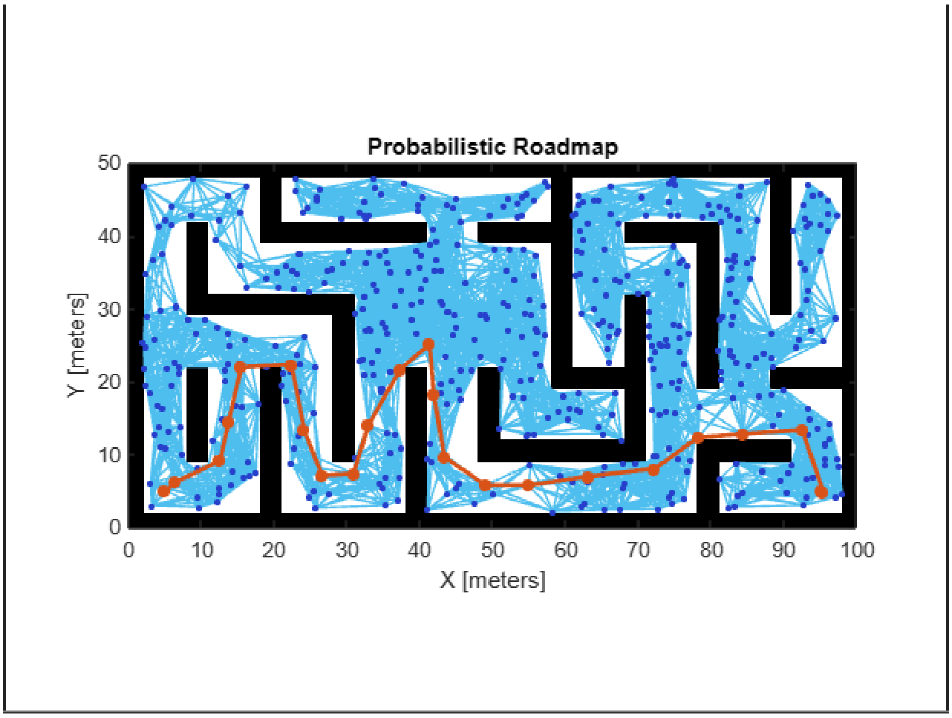


Figure 6.3: PRM path planning for map2

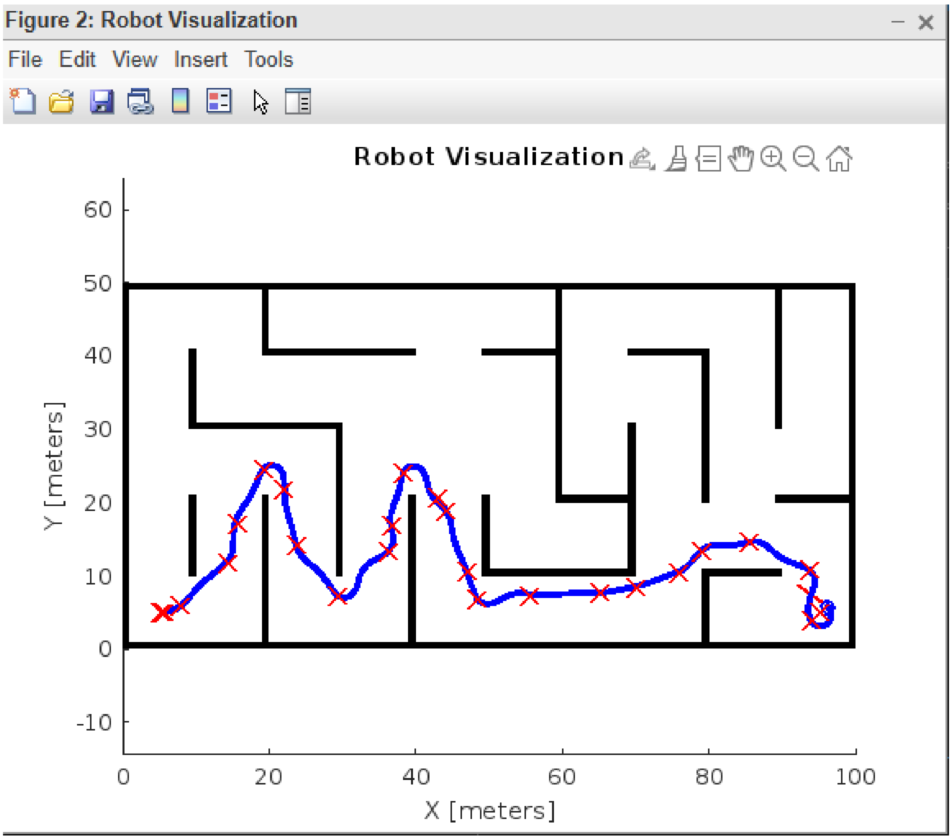


Figure 6.4: Auto motion for map2

Chapter 7

Conclusion

7.1 Coverage

The importance of the Wheeled Mobile Robot that is the wheels are less complex to configure when compared to a free legged robot considering its center of mass with be hard to maintain even though their capability to achieved the task in any uneven terrain is commendable the complexity is far hard to match and reach the fruits of achievements. And the use of mobile robot is as it says it has a mobility which is required to almost all of the practical uses.

Starting with the types of wheel's the robots are classified that is the wheels that offer only lateral motion are not better use in order to make the robot agile. Then comes the steerable and rotatable wheels concluding the conventional type if wheels.Then the types of wheel drives Differential drive that uses fixed wheels as active one and a passive steering wheel, Omni-directional wheel drive that makes use of lateral force and offers almost nil resistance to other forced as a benifit of having passive wheels embedded, Mecanum wheel drive makes of both lateral and longitudinal forces

Followed by the how the mobile robot makes use of the information from the real world which taken by sensors the generally used sensors and their classification is tabulated along with and with view of what they are used for and how they are categorized. Then the process of perception is discussed in detail then the localization of mobile robot in a given robot which is a critical point in answers the three main questions which are robot must answer they are Where it is? Where it should go? How it reaches there? The cognition path planning is useful to familarilze the robot with a global map with the guidance of the mission commands

Finally the Kinematics of different wheel drives is discussed along with a matlab code to make a simulation in forward kinematics that is given the velocities of each wheel the robot will be moving by the use of some matrix functions and Euler's method which is $x_{i+1} = x_i + \ddot{y} * dt$ along with some animation in matrix laboratory that is MATLAB

7.2 Summary of the Project

Motion planning for wheeled mobile robot in a given map provided there are no dynamic obstacles this is done for one of the wheel drives discussed above namely Differential Drive. Mentioning the kinematics both forward kinematics and inverse kinematics equations to accomplish the task the use of Probabilistic Road Map(PRM) followed by A*algorithm completes the path planning the rest is motion controlling that is made possible with the help of one of the matlab function controlPurePursuit which is then combined its results with inverse and forward kinematics of the differential drive. And the vizualization is along made to make the things more like simulation

The Probabilistic Road Map(PRM) is one of the genetic alogorithm which in general makes population of solution and then it uses nature selection that is the survival of the fittest method to conclude a single solution and this fittness funtion here is done in the form of A*algorithm which uses and fuction $f(n) = g(n) + h(n)$, where $f(n)$ determines the optimal path for the robot and $g(n)$ is the distance of min from the start point to one of the point in the path and then from that point to the goal distance is given by $h(n)$ here the goal may not be the final one but the intermediate one the is in the process of reaching the goal. While A*alogrithm does this all of it is done after a network of paths is formed by probabilistic road map which accomplishes this by make randoem nodes all over the map and the connects them one by one to be precise one after other in sequence until there will be no node to reach then it start's the same process from another node that has no yet in the network or not at all connected and repeated the process from it.

Like discussed in Chapter 2 the localization of the robot is done here in the given map and then the motion of the robot is also done as mentioned above that gives a view for the robot how to navigate in the given map avoiding the stationary obstacles in this case it is the walls that the wheeled mobile robot is avoiding and accomplishing the task of reaching the desired goal

7.3 Contribution of the Project

The motion planning of the wheeled mobile robot can be put to better use when it is implemented in a working space like manufacturing area or packing region and transporting the packed goods the code can be done multiple robot given that all robot are in sync of one another movements the process can be done at a steady pace resulting in completing the task.

Here the differential drive is make of use similarly other drives can also be used to their motion planning with their kinematic equations. The use of this motion and path planning is already in use inside the packing and transporting the goods in facilities like amazon and flipkart where the main task is to deliver the orders to the customers and the wheeled mobile robots can pick and place them inorder to delivery the.

To achieve those task according to their facilities the layout will be changed and the number of robots will be increased but the code is same for all the robots and when they all are acting in the same time the process is done in mind that the avoiding of collision not only with the obstacle's but also with fellow robot also hence the system can be sometimes a centralized in order to make are synchronized action at the same time even in the given map I self-created or the example map of matlab when the start and goal points are changed the wheeled mobile robot can accomplish the task with no difficulty given that there is a possibility to complete the task. Sometimes there will be some robots are needed to move in a fixed path like they are meant to be checking and dialgonizing the situation or analysing the process done smoothly.

7.4 Future Scope

The use of sensors is not done in the project even though the use of sensors and types of them are discussed. With the use of sensors wheeled mobile robot can even avoid the dynamic obstacles which will be a major advantage. As in real life the dynamic obstacle are a common scenario

The path planning and motion planning can be advanced in separately then they can be used not only for wheeled mobile robot but also for other field robots providing better results

Making of sensors to be able to navigate in an unknown territory that are in term called as SLAM robot that is robots that can Simultaneous Localization and Mapping for better performance

Appendix A

kinematic animation

```
clear;
clc;

%%Simulation parameters
dt = 0.1;    %step size
ts = 10;     %simulatioon time
t = 0:dt:ts;%time span

%initial conditions
x0 = 0;
y0 = 0;
psi0 = 0;

eta0 = [x0;y0;psi0];

eta(:,1) = eta0; %first element of the column matrix

%loop starts here
for i = 1:length(t)
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
```

```
                sin(psi) cos(psi) 0;
                0 0 1;];
    %inputs
    u = 0.3; %x-axis velocity
    v = 0; %y-axis velocity
    r = 0.2; %angular velocity

    zeta(:,i) = [u;v;r];

    eta_dot(:,i)=J_psi * zeta(:,i);

    eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler method
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.4; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];
figure
for i = 1:length(t) %animation starts here
```

```
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix B

inverse kinematic animation

```
clear;
clc;

%%Simulation parameters
dt = 0.1;    %step size
ts = 10;     %simulatioon time
t = 0:dt:ts;%time span

%initial conditions
x0 = 0;
y0 = 0;
psi0 = 0;

eta0 = [x0;y0;psi0];

eta(:,1) = eta0; %first element of the column matrix

%loop starts here
for i = 1:length(t)
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
```

```
             sin(psi) cos(psi) 0;
              0 0 1;];

    %Desired states (generalized coordinates)
    eta_d = [2-2*cos(0.1*t(i)); 2*sin(0.1*t(i)); 0.1*t(i)];
    eta_d_dot = [2*0.1*sin(0.1*t(i)); 2*0.1*cos(0.1*t(i));
   0.1;]; %derivative
    %Vector of velcity input commands
    zeta(:,i) = inv(J_psi) * eta_d_dot;
    %time derivative of generalized coordinates
    eta_dot(:,i)=J_psi * zeta(:,i);

    %Position propagation using Euler method
    eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %state
   update
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.4; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];
```

```
figure
for i = 1:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix C

diff wheel animation

```
clear;
clc;

%%Simulation parameters
dt = 0.1;   %step size
ts = 100;    %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
a = 0.2;   %radius of the wheel (fixed)
d = 0.5;    %distance b/t wheels

%initial conditions
x0 = 0.5;
y0 = 0.5;
psi0 = pi/4;

eta0 = [x0;y0;psi0];

eta(:,1) = eta0; %first element of the column matrix

%loop starts here
```

```
for i = 1:length(t)
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
    %inputs
    omega_1 = 0.5; %left wheel angular velocity
    omega_2 = 0.2; %right wheel angular velocity

    omega = [omega_1;omega_2];

    %wheel configuration matrix
    W = [a/2 a/2;
         0 0;
         -a/(2*d) a/(2*d)];

    %velocity input commands
    zeta(:,i) = W*omega;
    eta_dot(:,i)=J_psi * zeta(:,i);

    eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler method
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
```

```
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 2*d; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];
figure
for i = 1:5:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.01);
    hold off
end %animation ends here
```

Appendix D

omini wheel animation

```
clear;
clc;

%%Simulation parameters
dt = 0.1;    %step size
ts = 100;     %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
a = 0.2;    %radius of the wheel (fixed)
l_w = 0.5; %distance of wheel

%initial conditions
x0 = 0.5;
y0 = 0.5;
psi0 = pi/4;

eta0 = [x0;y0;psi0];

eta(:,1) = eta0; %first element of the column matrix

%loop starts here
```

```
for i = 1:length(t)
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
    %inputs
    omega_1 = 0.5; %1st wheel angular velocity
    omega_2 = 0.5;  %2nd wheel angular velocity
    omega_3 = 0.5;  %3rd wheel angular velocity
    %forward 0,-0.5,0.5 rotation clockwise 0.5,-0.5,-0.5
   anticlockwise 0.5,-0.5,0.5

    omega = [omega_1;omega_2;omega_3];

    %wheel configuration matrix
    W = a/3*[ 0 -sqrt(3) sqrt(3);
              2 -1 -1;
              1/l_w 1/l_w 1/l_w];

    %velocity input commands
    zeta(:,i) = W*omega;
    eta_dot(:,i)=J_psi * zeta(:,i);

    eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler
   method
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
```

```
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 2*l_w; % length of mobile robot
w = 0.4; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];
figure
for i = 1:5:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.01);
    hold off
end %animation ends here
```

Appendix E

mecanum wheel animation

```
clear;
clc;

%%Simulation parameters
dt = 0.1;    %step size
ts = 10;     %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
a = 0.2;    %radius of the wheel (fixed)
l_w = 0.3; %distance b/t wheels
d_w = 0.5; %%distance b/t wheel and frame

%initial conditions
x0 = 0.5;
y0 = 0.5;
psi0 = 0;

eta0 = [x0;y0;psi0];

eta(:,1) = eta0; %first element of the column matrix

```

```
%loop starts here
for i = 1:length(t)
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
    %inputs
    omega_1 = 0.5; %1st wheel angular velocity
    omega_2 = 0.5;  %2nd wheel angular velocity
    omega_3 = 0.5;  %3rd wheel angular velocity
    omega_4 = 0.5;  %4th wheel angular velocity
    %forward w,w,w,w rotation clockwise w,w,-w,-w anticlockwise -w,-w,w,w

    omega = [omega_1;omega_2;omega_3;omega_4];

    %wheel configuration matrix
    W = a/4*[1 1 1 1;
             1 -1 1 -1;
             -1/(d_w-l_w) -1/(d_w-l_w) 1/(d_w-l_w) 1/(d_w-l_w)];

    %velocity input commands
    zeta(:,i) = W*omega;
    eta_dot(:,i)=J_psi * zeta(:,i);

    eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler method
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
```

```
hold on
plot(t, eta(2,1:i), 'b--');      %y in m
plot(t, eta(3,1:i), 'm-.');      %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 2*l_w; % length of mobile robot
w = 2*d_w; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];
figure
for i = 1:5:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.01);
    hold off
end %animation ends here
```

Appendix F

dynamic model animation

```
clear;
clc;

%simulation parameters
dt=0.1; %step size
ts = 10; %simulation time
t = 0:dt:ts; %time span

%initial conditions
eta0 =[0;0;0];  %inital position and orientation of the
   vehicle
zeta0 = [0;0;0];%intial velocity of input commands

eta(:,1) = eta0;
zeta(:,1) = zeta0;

%robot parameters
m=10; %mass of the vehicle
Iz = 0.1; %inertia of the vehicle

xbc = 0; ybc = 0; %coordinates of mass centre

```

```
%state propagation

for i = 1:length(t)
    u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);

    %inertia matrix
    D = [m 0 -ybc*m;
         0 m xbc*m;
         -ybc*m xbc*m Iz+m*(xbc^2+ybc^2);];
    %other vector
    n_v = [-m*r*(v+xbc*r);
            m*r*(u-ybc*r);
            m*r*(xbc*u+ybc*v);];
    %input vector
    tau(:,i) = [1;0.5;0];

    %Jacobian matrix
    psi = eta(3,i);
    J_eta = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];

    zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
    zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);

    eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
   zeta_dot(:,i))); %state update
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
```

```
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.4; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];

figure
for i = 1:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix G

diff wheel dynamic

```
clear;
clc;

%simulation parameters
dt=0.1; %step size
ts = 10; %simulation time
t = 0:dt:ts; %time span

%initial conditions
eta0 =[0;0;0];  %inital position and orientation of the
   vehicle
zeta0 = [0;0;0];%intial velocity of input commands

eta(:,1) = eta0;
zeta(:,1) = zeta0;

%vehicle(mobile robot) parameters (physical)
a = 0.2;   %radius of the wheel (fixed)
d = 0.5;    %distance b/t wheels

%robot parameters
m=10; %mass of the vehicle
```

```
Iz = 0.1; %inertia of the vehicle

xbc = 0; ybc = 0; %coordinates of mass centre

%state propagation

for i = 1:length(t)
    u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);

    %inertia matrix
    D = [m 0 -ybc*m;
         0 m xbc*m;
         -ybc*m xbc*m Iz+m*(xbc^2+ybc^2);];
    %other vector
    n_v = [-m*r*(v+xbc*r);
            m*r*(u-ybc*r);
            m*r*(xbc*u+ybc*v);];

    %Wheel input matrix (differential wheel drive)
    Gamma = [1 1;
             0 0;
             -d d;];

    %Wheel inputs (traction forces)
    F1 = 0.5;      %left wheel force
    F2 = 0.5;    %right wheel force

    kappa = [F1;F2];
    %Input vector
    tau(:,i) = Gamma*kappa;

    %Jacobian matrix
    psi = eta(3,i);
    J_eta = [cos(psi) -sin(psi) 0;
```

```
             sin(psi) cos(psi) 0;
             0 0 1;];

    zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
    zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);

    eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*zeta_dot(:,i))); %state update
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.4; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];

figure
for i = 1:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
```

```
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix H

omini wheel dynamics

```
clear;
clc;

%simulation parameters
dt=0.1; %step size
ts = 10; %simulation time
t = 0:dt:ts; %time span

%initial conditions
eta0 =[0;0;0];  %inital position and orientation of the vehicle
zeta0 = [0;0;0];%intial velocity of input commands

eta(:,1) = eta0;
zeta(:,1) = zeta0;

%vehicle(mobile robot) parameters (physical)
a = 0.2;   %radius of the wheel (fixed)
d = 0.5;    %distance b/t wheels

%robot parameters
m=10; %mass of the vehicle
```

```
Iz = 0.1; %inertia of the vehicle

xbc = 0; ybc = 0; %coordinates of mass centre

%wheel configuartion parameters
l=0.3;
phi1 = 90*pi/180; phi2 = 210*pi/180; phi3 = 330*pi/180;

%state propagation
for i = 1:length(t)
    u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);

    %inertia matrix
    D = [m 0 -ybc*m;
         0 m xbc*m;
         -ybc*m xbc*m Iz+m*(xbc^2+ybc^2);];
    %other vector
    n_v = [-m*r*(v+xbc*r);
            m*r*(u-ybc*r);
            m*r*(xbc*u+ybc*v);];

    %Wheel input matrix (differential wheel drive)
    Gamma = [cos(phi1) cos(phi2) cos(phi3);
             sin(phi1) sin(phi2) sin(phi3);
             1 1 1;];

    %Wheel inputs (traction forces)
    F1 = 0;
    F2 = -0.5;
    F3 = 0.5; %lateral 1,-0.5,-0.5 rotate 1,0.5,0.5 forward
        0,-0.5,0.5
    kappa = [F1;F2;F3];

    %Input vector
```

```
    tau(:,i) = Gamma*kappa;

    %Jacobian matrix
    psi = eta(3,i);
    J_eta = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];

    zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
    zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);

    eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
   zeta_dot(:,i))); %state update
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.6; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];

figure
```

```
for i = 1:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix I

mecanum wheel dynamics

```
clear;
clc;

%simulation parameters
dt=0.1; %step size
ts = 10; %simulation time
t = 0:dt:ts; %time span

%initial conditions
eta0 =[0;0;0];  %inital position and orientation of the
    vehicle
zeta0 = [0;0;0];%intial velocity of input commands

eta(:,1) = eta0;
zeta(:,1) = zeta0;

%vehicle(mobile robot) parameters (physical)
a = 0.2;   %radius of the wheel (fixed)
d = 0.5;    %distance b/t wheels

%robot parameters
m=10; %mass of the vehicle
```

```
Iz = 0.1; %inertia of the vehicle

xbc = 0; ybc = 0; %coordinates of mass centre

%wheel configuartion parameters
l=0.3;
phi1 = 90*pi/180; phi2 = 210*pi/180; phi3 = 330*pi/180;

%state propagation
for i = 1:length(t)
    u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);

    %inertia matrix
    D = [m 0 -ybc*m;
         0 m xbc*m;
         -ybc*m xbc*m Iz+m*(xbc^2+ybc^2);];
    %other vector
    n_v = [-m*r*(v+xbc*r);
            m*r*(u-ybc*r);
            m*r*(xbc*u+ybc*v);];

    %Wheel input matrix (differential wheel drive)
    Gamma = 1/sqrt(2)*[1 1 1 1;
                       1 -1 1 -1;
                       l-d l-d -(l-d) -(l-d);];

    %Wheel inputs (traction forces)
    F1 = 0.2;
    F2 = 0.2;
    F3 = 0.2;
    F4 = 0.2; %lateral f,-f,f,-f rotate f,f,-f,-f forward f
   ,f,f,f
    kappa = [F1;F2;F3;F4];

```

```
    %Input vector
    tau(:,i) = Gamma*kappa;

    %Jacobian matrix
    psi = eta(3,i);
    J_eta = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];

    zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
    zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);

    eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*zeta_dot(:,i))); %state update
end

%plotting
figure
plot(t, eta(1,1:i), 'r-');      %for plotting x in m
hold on
plot(t, eta(2,1:i), 'b--');     %y in m
plot(t, eta(3,1:i), 'm-.');     %t in s
legend('x, [m]','y, [m]','/psi, [rad]');
set(gca,'fontsize',24)
xlabel('t, [s]');
ylabel('/eta, [units]');

%animation (mobile robot motion animation)
l = 0.6; % length of mobile robot
w = 0.6; % width of the mobile robot
%Mobile robot coordinates
mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
       -w/2 -w/2 w/2 w/2 -w/2;];

```

```
figure
for i = 1:length(t) %animation starts here
    psi = eta(3,i);
    R_psi=[cos(psi) -sin(psi);
           sin(psi)  cos(psi);];%rotation matrix
    v_pos=R_psi*mr_co;
    fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
    hold on, grid on
    axis([-1 3 -1 3]), axis square
    plot(eta(1,1:i),eta(2,1:i),'b-');
    legend('MR','Path')
    set(gca,'fontsize',24)
    xlabel('x,[m]'); ylabel('y,[m]');
    pause(0.1);
    hold off
end %animation ends here
```

Appendix J

Fixed path Differential Drive Motion

```
%% Simulation parameters
dt = 0.1;   %step size
ts = 25;    %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
r = 0.2;   %radius of the wheel (fixed)
b = 0.5;    %distance b/t wheels (base)

%initial conditions
x0 = 2;
y0 = 2;
psi0 = 0;

eta0 = [x0;y0;psi0];
eta = zeros(3,numel(t));    % Pose matrix
eta(:,1) = eta0; %first element of the column matrix
%% Path planning
% Load map and inflate it by a safety distance
close all
load exampleMap
inflate(map,r);
```

```
%inflate(map,radius) inflates each occupied position of the
% map by the radius given in meters.radius is rounded up to
% the nearest cell equivalent based on the resolution of
    the map.
% Every cell within the radius is set to true (1).

%Desired route or waypoints
waypoints = [2 2;
             4 4;
             4 8;
             11 8;
             11 11]
%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.35;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 1.5;

%% Create visualizer
load exampleMap % Reload original (uninflated) map for
    visualization
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';

%% Simulation loop starts here
rate = rateControl(1/dt);
for i = 1:numel(t)
     % Run the Pure Pursuit controller and convert output
    to wheel speeds
```

```
    [vRef,wRef] = controller(eta(:,i)); %[vel,angvel] = controller(pose) processes the vehicle's position and orientation, pose, and outputs the linear velocity, vel, and angular velocity, angvel.


    % Inverse loop = inverseKinematics(dd,vRef,wRef)
    % Calculates wheel speeds from linear and angular velocity
    wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)/obj.wheelRadius;
    wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)/obj.wheelRadius;


    % Compute the velocities, [v,w] = forwardKinematics(dd,wL,wR);
    % Calculates linear and angular velocity from wheel speeds
    v =0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
    w = (wR-wL)*r/b;  %w = (wR-wL)*obj.wheelRadius/obj.wheelBase;
    zetaB = [v;0;w]; % Body velocities [vx;vy;w]

    % Convert from body to world
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
    zeta = J_psi*zetaB;

     % Perform forward discrete integration step
    eta(:,i+1) = eta(:,i) + zeta*dt;
```

```

    % Update visualization
    viz(eta(:,i+1),waypoints)
    waitfor(rate);
end
```

Appendix K

2.Fixed path Differential Drive Motion

```
%% Simulation parameters
dt = 0.1;    %step size
ts = 70;     %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
r = 0.2;    %radius of the wheel (fixed)
b = 0.5;     %distance b/t wheels (base)

%initial conditions
x0 = 5;
y0 = 5;
psi0 = 0;

eta0 = [x0;y0;psi0];
eta = zeros(3,numel(t));     % Pose matrix
eta(:,1) = eta0; %first element of the column matrix
%% Path planning
% Load map and inflate it by a safety distance
% close all
% load exampleMap
map = binaryOccupancyMap(100,50,1);
```

```
walls = zeros(100,50);
walls(1,1:100) = 1; % Top wall
walls(50,:) = 1; % Bottom wall
walls(:,1) = 1; % Left wall
walls(:,100) = 1; % Right wall
%verical walls
walls(30:40,10) = 1;
walls(10:20,10) = 1;
walls(30:50,20) = 1;
walls(1:10,20) = 1;
walls(20:40,30) = 1;
walls(30:50,40) = 1;
walls(30:40,50) = 1;
walls(1:30,60) = 1;
walls(20:40,70) = 1;
walls(10:30,80) = 1;
walls(40:50,80) = 1;
walls(1:20,90) = 1;

%Horizontal walls
walls(20,10:30) = 1;
walls(10,20:40) = 1;
walls(10,50:60) = 1;
walls(40,50:70) = 1;
walls(30,60:70) = 1;
walls(10,70:80) = 1;
walls(40,80:90) = 1;
walls(30,90:100) = 1;

setOccupancy(map,[1 1],walls,"grid");
inflate(map,r);
%inflate(map,radius) inflates each occupied position of the
% map by the radius given in meters.radius is rounded up to
```

```
% the nearest cell equivalent based on the resolution of
    the map.
% Every cell within the radius is set to true (1).

%Desired route or waypoints
waypoints = [5 5;
             15 5;
             15 25;
             25 25;
             25 5;
             35 5;
             35 25;
             45 25;
             45 5;
             75 5;
             75 15;
             95 15;
             95 5];

%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 1 ;
controller.DesiredLinearVelocity = 3;
controller.MaxAngularVelocity = 3;

%% Create visualizer
setOccupancy(map,[1 1],walls,"grid"); % Reload original (
    uninflated) map for visualization
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';

%% Simulation loop starts here
```

```
rate = rateControl(1/dt);
for i = 1:numel(t)
     % Run the Pure Pursuit controller and convert output to wheel speeds
    [vRef,wRef] = controller(eta(:,i)); %[vel,angvel] = controller(pose) processes the vehicle's position and orientation, pose, and outputs the linear velocity, vel, and angular velocity, angvel.


    % Inverse loop = inverseKinematics(dd,vRef,wRef)
    % Calculates wheel speeds from linear and angular velocity
    wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)/obj.wheelRadius;
    wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)/obj.wheelRadius;


    % Compute the velocities, [v,w] = forwardKinematics(dd,wL,wR);
    % Calculates linear and angular velocity from wheel speeds
    v =0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
    w = (wR-wL)*r/b;  %w = (wR-wL)*obj.wheelRadius/obj.wheelBase;
    zetaB = [v;0;w]; % Body velocities [vx;vy;w]

    % Convert from body to world
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
```

```
    zeta = J_psi*zetaB;

     % Perform forward discrete integration step
    eta(:,i+1) = eta(:,i) + zeta*dt;

    % Update visualization
    viz(eta(:,i+1),waypoints)
    waitfor(rate);
end
```

Appendix L

Map2

```
myMap = binaryOccupancyMap(100,50,1)
walls = zeros(100,50);
walls(1,1:100) = 1; % Top wall
walls(50,:) = 1; % Bottom wall
walls(:,1) = 1; % Left wall
walls(:,100) = 1; % Right wall
%verical walls
walls(30:40,10) = 1;
walls(10:20,10) = 1;
walls(30:50,20) = 1;
walls(1:10,20) = 1;
walls(20:40,30) = 1;
walls(30:50,40) = 1;
walls(30:40,50) = 1;
walls(1:30,60) = 1;
walls(20:40,70) = 1;
walls(10:30,80) = 1;
walls(40:50,80) = 1;
walls(1:20,90) = 1;

%Horizontal walls
walls(20,10:30) = 1;
```

```
walls(10,20:40) = 1;
walls(10,50:60) = 1;
walls(40,50:70) = 1;
walls(30,60:70) = 1;
walls(10,70:80) = 1;
walls(40,80:90) = 1;
walls(30,90:100) = 1;

setOccupancy(myMap,[1 1],walls,"grid")
show(myMap)
inflateRadius =0.2
inflate(myMap,inflateRadius)
show(myMap)
%%
map = binaryOccupancyMap(100,50,1)
setOccupancy(map,[1 1],walls,"grid");
setOccupancy(myMap,[1 1],walls,"grid")
show(myMap)
```

Appendix M

Auto path Differential Drive motion

```
%% Simulation parameters
dt = 0.1;    %step size
ts = 25;     %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
r = 0.2;   %radius of the wheel (fixed)
b = 0.5;    %distance b/t wheels (base)

%initial conditions
x0 = 2;
y0 = 2;
psi0 = 0;

eta0 = [x0;y0;psi0];
eta = zeros(3,numel(t));    % Pose matrix
eta(:,1) = eta0; %first element of the column matrix
%% Path planning
% Load map and inflate it by a safety distance
close all
load exampleMap
inflate(map,r);
```

```
%inflate(map,radius) inflates each occupied position of the
% map by the radius given in meters.radius is rounded up to
% the nearest cell equivalent based on the resolution of
    the map.
% Every cell within the radius is set to true (1).

% Create a Probabilistic Road Map (PRM) for mobileRobotPRM
    object and define associated attributes
planner = mobileRobotPRM(map);
planner.NumNodes = 75;
planner.ConnectionDistance = 5;

% Find a path in PRM using A* algorithm from the start
    point to a specified goal point
%A* needs to determine the cost(least distance travelled,
    shortest time) of the path from the start node to n and
    the cost of the cheapest path from n to the goal.
startPoint = eta0(1:2)';
goalPoint  = [11, 11];
waypoints = findpath(planner,startPoint,goalPoint);
show(planner)

%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.35;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 1.5;

%% Create visualizer
load exampleMap % Reload original (uninflated) map for
    visualization
viz = Visualizer2D;
viz.hasWaypoints = true;
```

```
viz.mapName = 'map';

%% Simulation loop starts here
rate = rateControl(1/dt);
for i = 1:numel(t)
     % Run the Pure Pursuit controller and convert output to wheel speeds
    [vRef,wRef] = controller(eta(:,i)); %[vel,angvel] = controller(pose) processes the vehicle's position and orientation, pose, and outputs the linear velocity, vel, and angular velocity, angvel.


    % Inverse loop = inverseKinematics(dd,vRef,wRef)
    % Calculates wheel speeds from linear and angular velocity
    wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)/obj.wheelRadius;
    wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)/obj.wheelRadius;


    % Compute the velocities, [v,w] = forwardKinematics(dd,wL,wR);
    % Calculates linear and angular velocity from wheel speeds
    v =0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
    w = (wR-wL)*r/b;  %w = (wR-wL)*obj.wheelRadius/obj.wheelBase;
    zeta = [v;0;w]; % Body velocities [vx;vy;w]

    % Convert from body to world
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
```

```
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
             0 0 1;];
    eta_dot = J_psi*zeta;

     % Perform forward discrete integration step
    eta(:,i+1) = eta(:,i) + eta_dot*dt;

    % Update visualization
    viz(eta(:,i+1),waypoints)
    waitfor(rate);
end
```

Appendix N

2.Auto path Differential Drive motion

```
%% Simulation parameters
dt = 0.1;    %step size
ts = 60;     %simulatioon time
t = 0:dt:ts;%time span

%vehicle(mobile robot) parameters (physical)
r = 0.2;    %radius of the wheel (fixed)
b = 0.5;     %distance b/t wheels (base)

%initial conditions
x0 = 5;
y0 = 5;
psi0 = 0;

eta0 = [x0;y0;psi0];
eta = zeros(3,numel(t));     % Pose matrix
eta(:,1) = eta0; %first element of the column matrix
%% Path planning
% Load map and inflate it by a safety distance
% close all
% load exampleMap
map = binaryOccupancyMap(100,50,1);
```

```
walls = zeros(100,50);
walls(1,1:100) = 1; % Top wall
walls(50,:) = 1; % Bottom wall
walls(:,1) = 1; % Left wall
walls(:,100) = 1; % Right wall
%verical walls
walls(30:40,10) = 1;
walls(10:20,10) = 1;
walls(30:50,20) = 1;
walls(1:10,20) = 1;
walls(20:40,30) = 1;
walls(30:50,40) = 1;
walls(30:40,50) = 1;
walls(1:30,60) = 1;
walls(20:40,70) = 1;
walls(10:30,80) = 1;
walls(40:50,80) = 1;
walls(1:20,90) = 1;

%Horizontal walls
walls(20,10:30) = 1;
walls(10,20:40) = 1;
walls(10,50:60) = 1;
walls(40,50:70) = 1;
walls(30,60:70) = 1;
walls(10,70:80) = 1;
walls(40,80:90) = 1;
walls(30,90:100) = 1;

setOccupancy(map,[1 1],walls,"grid");
inflate(map,r);
%inflate(map,radius) inflates each occupied position of the
% map by the radius given in meters.radius is rounded up to
```

```
% the nearest cell equivalent based on the resolution of
    the map.
% Every cell within the radius is set to true (1).

% Create a Probabilistic Road Map (PRM) for mobileRobotPRM
    object and define associated attributes
planner = mobileRobotPRM(map);
planner.NumNodes = 500;
planner.ConnectionDistance = 10;

% Find a path using PRM algorithm from the start point to a
     specified goal point
%A* needs to determine the cost(least distance travelled,
    shortest time) of the path from the start node to n and
    the cost of the cheapest path from n to the goal.
startPoint = eta0(1:2)';
goalPoint  = [95,5];
waypoints = findpath(planner,startPoint,goalPoint);
show(planner)

%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 1 ;
controller.DesiredLinearVelocity = 3;
controller.MaxAngularVelocity = 4;

%% Create visualizer
setOccupancy(map,[1 1],walls,"grid"); % Reload original (
    uninflated) map for visualization
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';

```

```
%% Simulation loop starts here
rate = rateControl(1/dt);
for i = 1:numel(t)
     % Run the Pure Pursuit controller and convert output
   to wheel speeds
    [vRef,wRef] = controller(eta(:,i)); %[vel,angvel] =
   controller(pose) processes the vehicle's position and
   orientation, pose, and outputs the linear velocity, vel,
    and angular velocity, angvel.


    % Inverse loop = inverseKinematics(dd,vRef,wRef)
    % Calculates wheel speeds from linear and angular
   velocity
    wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)
   /obj.wheelRadius;
    wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)
   /obj.wheelRadius;


    % Compute the velocities, [v,w] = forwardKinematics(dd,
   wL,wR);
    % Calculates linear and angular velocity from wheel
   speeds
    v =0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
    w = (wR-wL)*r/b;  %w = (wR-wL)*obj.wheelRadius/obj.
   wheelBase;
    zetaB = [v;0;w]; % Body velocities [vx;vy;w]

    % Convert from body to world
    psi = eta(3,i);%current orientation in rad
    %jacobian matrix
    J_psi = [cos(psi) -sin(psi) 0;
             sin(psi) cos(psi) 0;
```

```
              0  0  1;];
    zeta = J_psi*zetaB;

     % Perform forward discrete integration step
    eta(:,i+1) = eta(:,i) + zeta*dt;

    % Update visualization
    viz(eta(:,i+1),waypoints)
    waitfor(rate);
end
```

References

[1] S. G. Tzafestas, *Introduction to mobile robot control*. Elsevier, 2013.

[2] R. Fierro and F. L. Lewis, "Control of a nonholomic mobile robot: Backstepping kinematics into dynamics," *Journal of robotic systems*, vol. 14, no. 3, pp. 149–163, 1997.

[3] K. D. Iagnemma, A. Rzepniewski, S. Dubowsky, P. Pirjanian, T. L. Huntsberger, and P. S. Schenker, "Mobile robot kinematic reconfigurability for rough terrain," in *Sensor Fusion and Decentralized Control in Robotic Systems III*, vol. 4196. SPIE, 2000, pp. 413–420.

[4] E. Ivanjko, T. Petrinic, and I. Petrovic, "Modelling of mobile robot dynamics," in *7th EUROSIM Congress on Modelling and Simulation*, vol. 2, 2010.

[5] K. Shabalina, A. Sagitov, and E. Magid, "Comparative analysis of mobile robot wheels design," in *2018 11th International Conference on Developments in esystems Engineering (dese)*. IEEE, 2018, pp. 175–179.

[6] A. Hoover and B. D. Olsen, "Sensor network perception for mobile robotics," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 342–347.

[7] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[8] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning: Sensors and techniques," *Journal of robotic systems*, vol. 14, no. 4, pp. 231–249, 1997.

[9] M. Betke and L. Gurvits, “Mobile robot localization using landmarks,” *IEEE transactions on robotics and automation*, vol. 13, no. 2, pp. 251–263, 1997.

[10] Y. Chung, C. Park, and F. Harashima, “A position control differential drive wheeled mobile robot,” *IEEE Transactions on Industrial Electronics*, vol. 48, no. 4, pp. 853–863, 2001.

[11] G. Klancar, D. Matko, and S. Blazic, “Mobile robot control on a reference path,” in *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.* IEEE, 2005, pp. 1343–1348.

[12] Y. Zhang, N. Fattahi, and W. Li, “Probabilistic roadmap with self-learning for path planning of a mobile robot in a dynamic and unstructured environment,” in *2013 IEEE International Conference on Mechatronics and Automation*. IEEE, 2013, pp. 1074–1079.

[13] H. Krawinkler, M. Zohrei, B. Lashkari-Irvani, N. Cofie, and H. Hadidi-Tamjed, “Recommendations for experimental studies on the seismic behavior of steel components and materials,” Department of Civil and Environmental Engineering, Stanford Unniversity, Report, September 1983.

[14] J. Lemaitre and J. L. Chaboche, *Mechanics of Solid Materials*. Cambridge University Press, 1994.

[15] A. Bower, *Applied Mechanics of Solids*. Taylor & Francis, 2011.

[16] ABAQUS, *Abaqus 6.11 Online Documentation*. Dassault Systemes, 2011.