

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 4th year Mechanical student of IIITDM Kurnool am undertaking a final year project under the supervision 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 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 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

Certificate	i
Abstract	ii
Contents	iii
List of Figures	vi
List of Tables	vii
Abbreviations	viii
Symbols	ix
1 Introduction	1
1.1 Mobile robot	1
1.2 Robot Kinematics	1
1.3 Robot Dynamics	2
1.4 Types of Wheels	2
1.5 Sensing and Perception	4
1.5.1 Commonly used Sensors	6
1.5.2 Localization of Wheeled mobile robot	6
2 Kinematics Algorithms	7
2.1 Kinematics	7
2.1.1 Kinematic animation	9
2.1.2 Inverse kinematic animation	10
2.1.3 Differential wheel drive animation	10
2.1.4 Omini wheel animation	11
2.1.5 Mecanum wheel animation	12

3	Dynamics Algorithms	13
3.1	Forward Dynamics	13
3.1.1	Dynamic model animation	15
3.1.2	Differential wheel dynamic	15
3.1.3	Omini wheel dynamics	16
3.1.4	mecanum wheel dynamics	17
4	Differential Drive Mobile Robot	18
4.1	Forward Kinematics	18
4.2	Inverse Kinematics	19
5	Fixed path Differential Drive Motion	20
5.1	Key functions & words	20
5.2	Algorithm	20
6	Auto path Differential Drive Motion	23
6.1	Key functions & Definitions	23
6.2	Algorithm	23
7	Conclusion	27
7.1	Coverage	27
7.2	Summary of the Project	28
7.3	Contribution of the Project	29
7.4	Future Scope	30
A	kinematic animation	31
B	inverse kinematic animation	34
C	diff wheel animation	37
D	omini wheel animation	40
E	mecanum wheel animation	43
F	dynamic model animation	46
G	diff wheel dynamic	49
H	omini wheel dynamics	53
I	mecanum wheel dynamics	57

J	Fixed path Differential Drive Motion	61
K	2.Fixed path Differential Drive Motion	65
L	Map2	70
M	Auto path Differential Drive motion	72
N	2.Auto path Differential Drive motion	76
 References		 81
References		81

List of Figures

1.1	Types of wheels from left to right	3
1.2	Types of wheels drives	3
2.1	Kinematic resultants	7
2.2	Wheel configuration matrix	9
3.1	Dynamic resultants	14
4.1	Differential Drive Kinematics of a Wheeled Mobile Robot	18
5.1	Fixed path motion for map1	22
5.2	Fixed path motion for map2	22
6.1	PRM path planning for map1	25
6.2	Auto motion for map1	25
6.3	PRM path planning for map2	26
6.4	Auto motion for map2	26

List of Tables

1.1	Classifying Common Sensors	6
-----	--------------------------------------	---

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

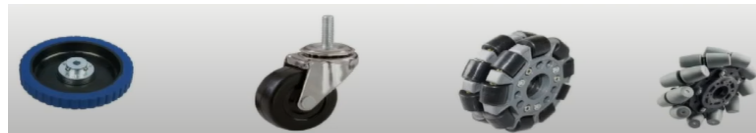
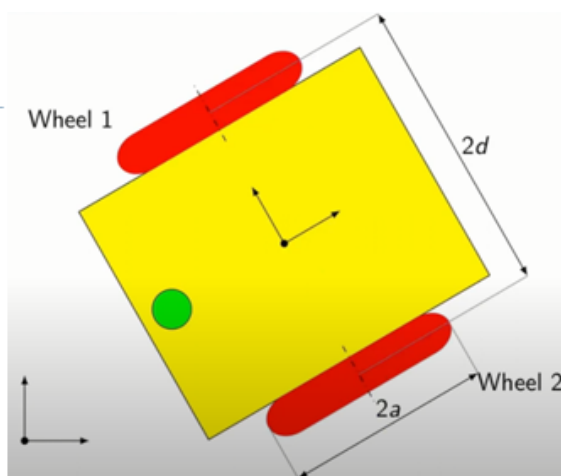
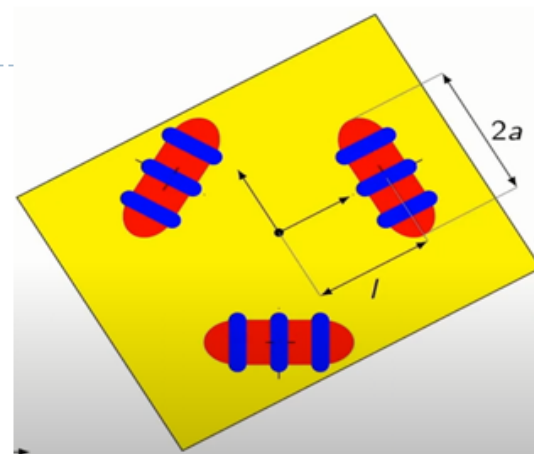
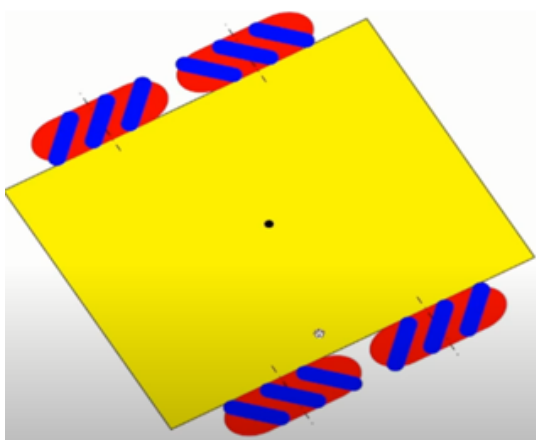


Figure 1.1: Types of wheels from left to right



Differential wheel drive

Omni-directional
Wheel drive

Mecanum wheel drive

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 processed 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 instructions and the feasibility will be checked
 - **Model**- The model for the robot will be made ready to test in practice and to achieve the required goal
 - **Prediction**- Using the model in practice or by using its simulation the performance will be predicted and the chances of its perfect achievement 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 built
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 required 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

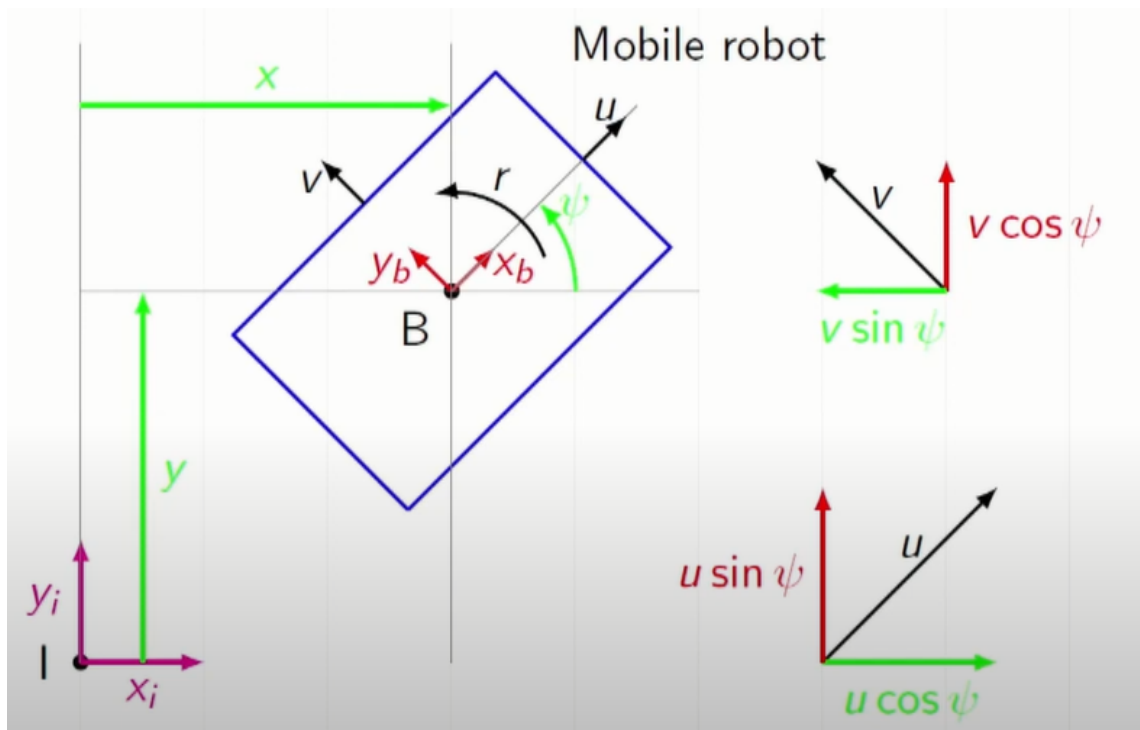


Figure 2.1: Kinematic resultants

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;

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

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

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

Where,

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

Again velocity can be written as

$$v = r\omega \Rightarrow \zeta = W\omega$$

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 $\zeta = W\omega$ based on the

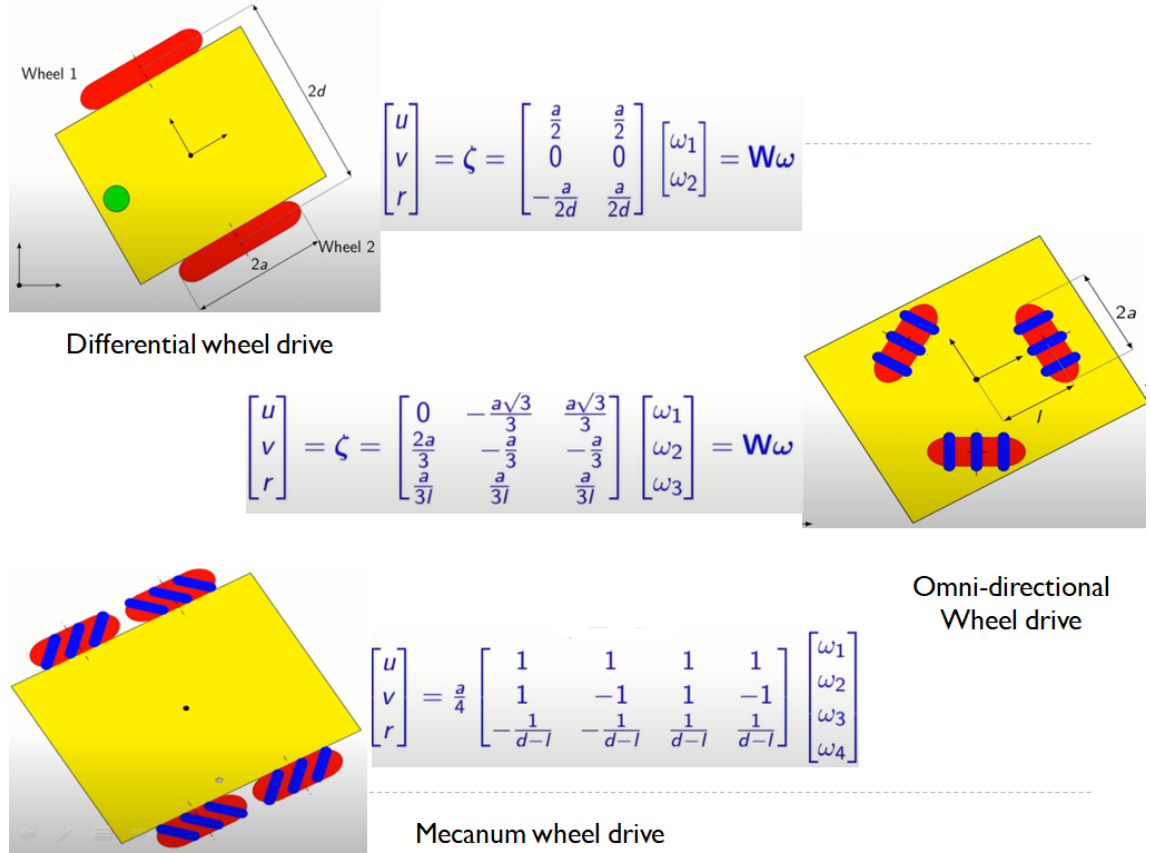


Figure 2.2: Wheel configuration matrix

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 $\zeta = (u,v,r)$

- 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
- Assigning 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-ordinates 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
- Assigning 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-ordinates 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](#)

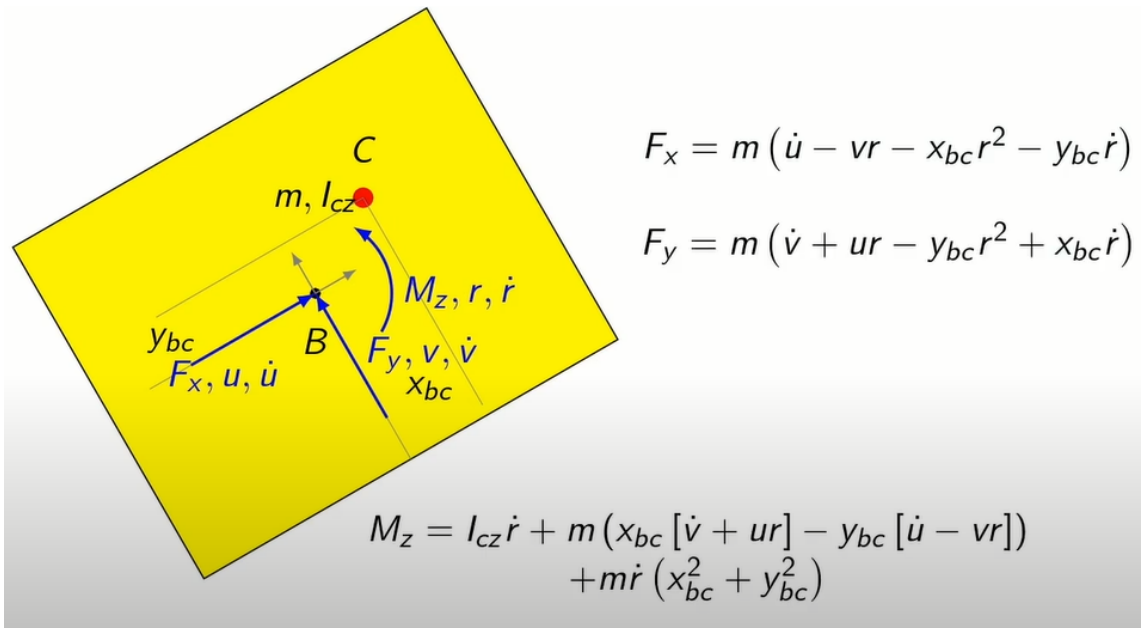
The algorithm goes as follows:

- 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
- Assigning 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-ordinates 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 $\eta, \dot{\eta}, \ddot{\eta}$, **problem of simulating or analyzing the robot**

Where,

- **D**-Inertia matrix,

$$\begin{aligned}
& \begin{bmatrix} m\dot{u} - mvr - mx_{bc}r^2 - my_{bc}\dot{r} \\ m\dot{v} + mur - my_{bc}r^2 + mx_{bc}\dot{r} \\ (I_{cz} + m(x_{bc}^2 + y_{bc}^2))\dot{r} + m(x_{bc}[\dot{v} + ur] - y_{bc}[\dot{u} - vr]) \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} \\
& \begin{bmatrix} m & 0 & -my_{bc} \\ 0 & m & mx_{bc} \\ -my_{bc} & mx_{bc} & I_{cz} + m(x_{bc}^2 + y_{bc}^2) \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} -mr(v + x_{bc}r) \\ mr(u - y_{bc}r) \\ mr(x_{bc}u + y_{bc}v) \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} \\
& \mathbf{D}\dot{\zeta} + \mathbf{n}(\zeta) = \boldsymbol{\tau}
\end{aligned}$$

Figure 3.1: Dynamic resultants

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

Again torque can be write as

$$\boldsymbol{\tau} = \mathbf{r}F \Rightarrow \boldsymbol{\tau} = \boldsymbol{\Gamma}\mathbf{F}$$

Where,

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

w.k.t, from kinematic relations $\dot{\eta} = J(\psi)\dot{\zeta}$ Applying Euler method to ζ i.e, $\zeta_{i+1} = \zeta_i + \dot{\zeta}dt$

From the dynamic relations we get $\dot{\zeta} = D^{-1}(\boldsymbol{\tau} - \mathbf{n}(\zeta))$ Again we can write $\boldsymbol{\tau}$ as $\boldsymbol{\tau} = \boldsymbol{\Gamma}k$

-mapping & k-wheel force $\boldsymbol{\Gamma}$ 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 & Inertia 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 $\dot{\zeta} = D^{-1} * (\tau - n_v)$ and $\zeta_{i+1} = \zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi) * (\zeta_i + dt * \dot{\zeta}_i))$
- 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 & Inertia 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 $\dot{\zeta} = D^{-1} * (\tau - n_v)$ and $\zeta_{i+1} = \zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi) * (\zeta_i + dt * \dot{\zeta}_i))$
- 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 & Inertia 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 $\dot{\zeta} = D^{-1} * (\tau - n_v)$ and $\zeta_{i+1} = \zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi) * (\zeta_i + dt * \dot{\zeta}_i))$
- 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 & Inertia 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 $\dot{\zeta} = D^{-1} * (\tau - n_v)$ and $\zeta_{i+1} = \zeta_i + dt * \dot{\zeta}_i$
- Status update $\eta_{i+1} = \eta_i + dt * (J(\psi) * (\zeta_i + dt * \dot{\zeta}_i))$
- 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

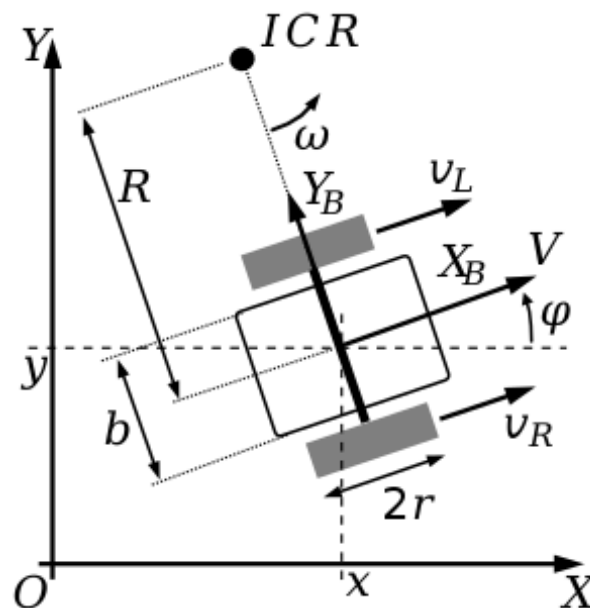


Figure 4.1: Differential Drive Kinematics of a Wheeled Mobile Robot

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

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

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

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

From V_R & V_L we get,

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

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

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

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,

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

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

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 is made so that obstacle's can't enter robot's cartesian space
5. Waypoints will be assigned
6. 2D visualization 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

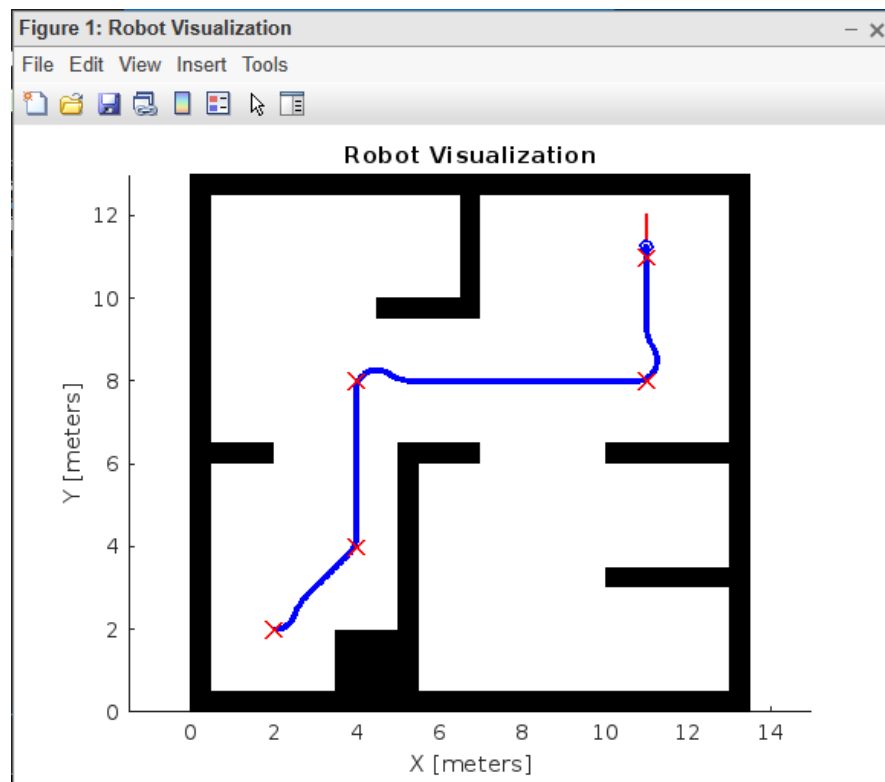


Figure 5.1: Fixed path motion for map1

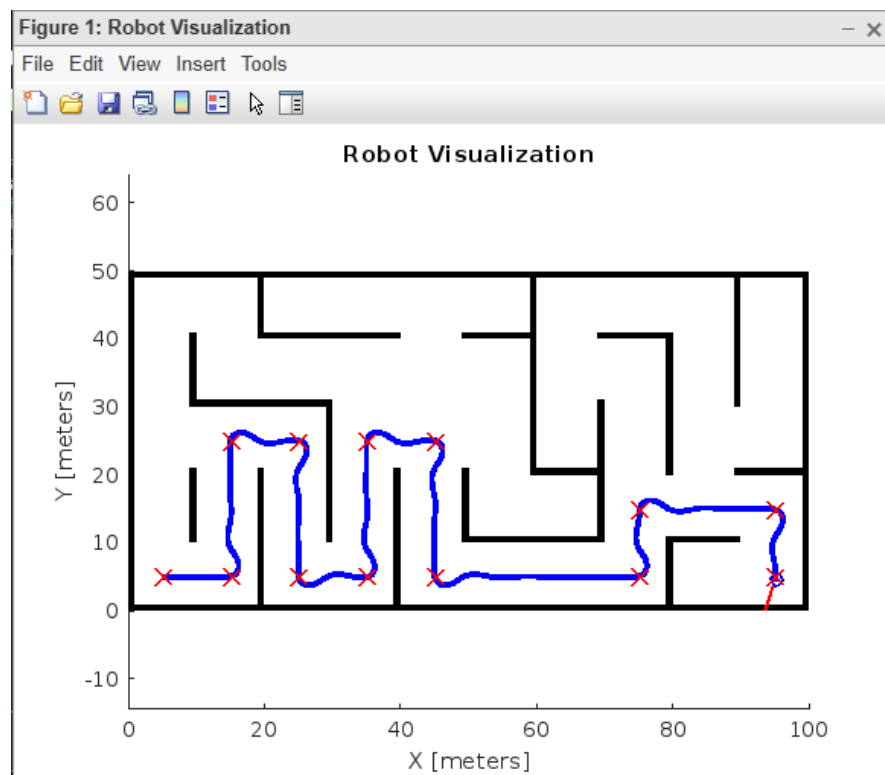


Figure 5.2: Fixed path motion for map2

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 is 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 visualization 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

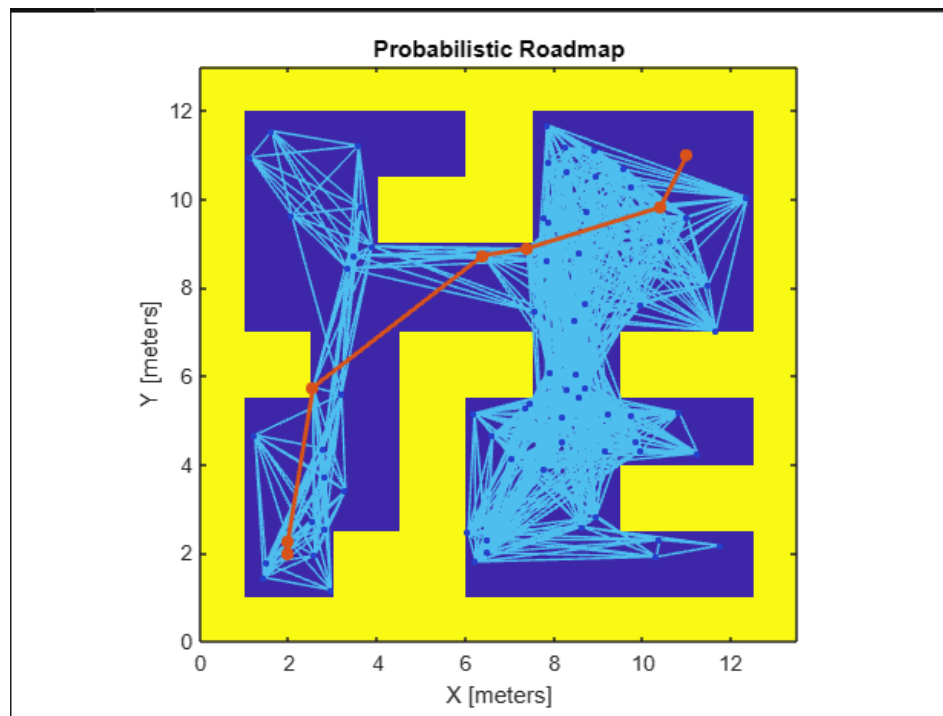


Figure 6.1: PRM path planning for map1

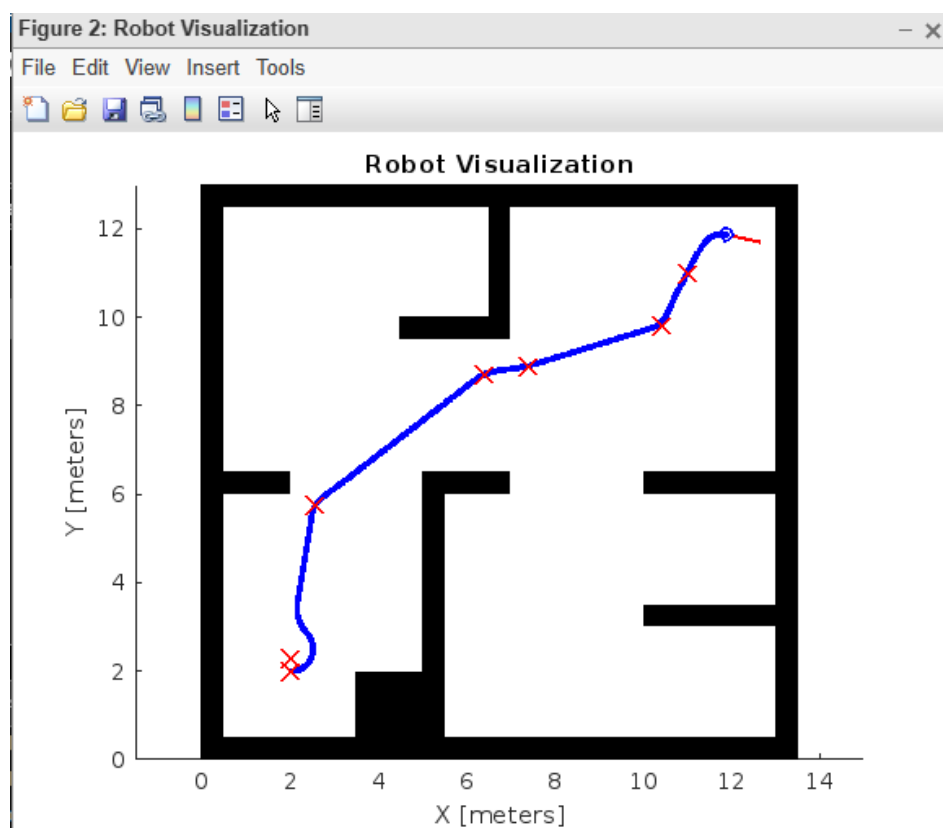


Figure 6.2: Auto motion for map1

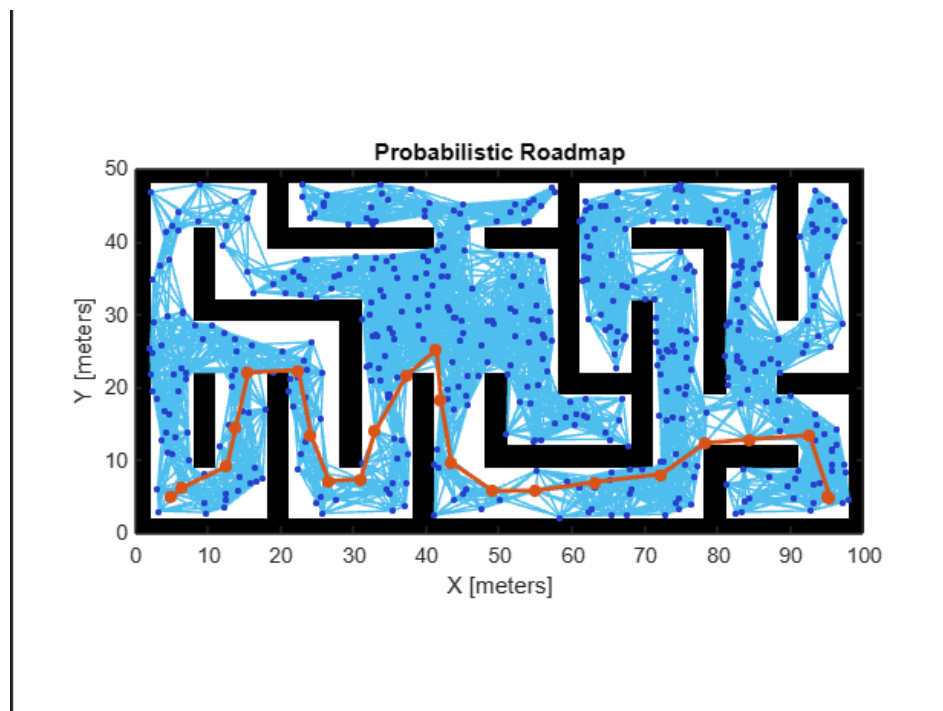


Figure 6.3: PRM path planning for map2

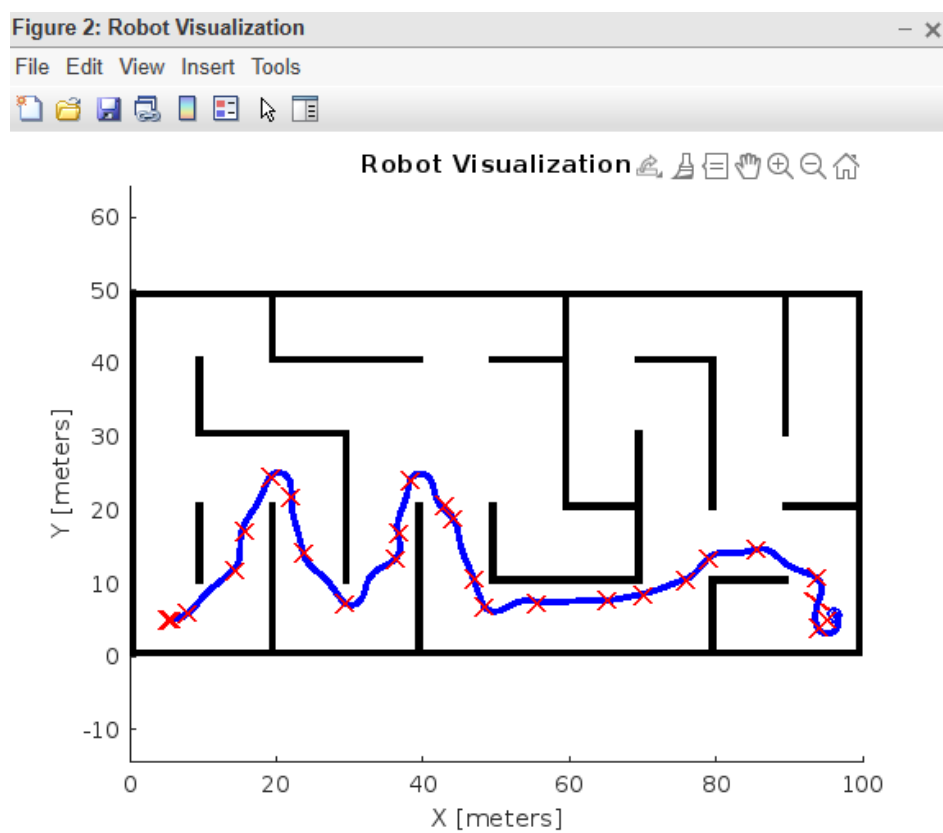


Figure 6.4: Auto motion for map2

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 familiarize 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 $x_{i+1} = x_i + \dot{y} * dt$ 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 fitness 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

```
1 clear;
2 clc;
3
4 %%Simulation parameters
5 dt = 0.1;    %step size
6 ts = 10;     %simulation time
7 t = 0:dt:ts;%time span
8
9 %initial conditions
10 x0 = 0;
11 y0 = 0;
12 psi0 = 0;
13
14 eta0 = [x0;y0;psi0];
15
16 eta(:,1) = eta0; %first element of the column matrix
17
18 %loop starts here
19 for i = 1:length(t)
20     psi = eta(3,i);%current orientation in rad
21     %jacobian matrix
22     J_psi = [cos(psi) -sin(psi) 0;
```

```

23         sin(psi) cos(psi) 0;
24         0 0 1;];
25     %inputs
26     u = 0.3; %x-axis velocity
27     v = 0; %y-axis velocity
28     r = 0.2; %angular velocity
29
30     zeta(:,i) = [u;v;r];
31
32     eta_dot(:,i)=J_psi * zeta(:,i);
33
34     eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler
35     method
36 end
37 %plotting
38 figure
39 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
40 hold on
41 plot(t, eta(2,1:i), 'b--'); %y in m
42 plot(t, eta(3,1:i), 'm-.'); %t in s
43 legend('x, [m]', 'y, [m]', '/psi, [rad]');
44 set(gca, 'fontsize', 24)
45 xlabel('t, [s]');
46 ylabel('/eta, [units]');
47
48 %animation (mobile robot motion animation)
49 l = 0.6; % length of mobile robot
50 w = 0.4; % width of the mobile robot
51 %Mobile robot coordinates
52 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
53         -w/2 -w/2 w/2 w/2 -w/2];
54 figure
55 for i = 1:length(t) %animation starts here

```

```
56     psi = eta(3,i);
57     R_psi=[cos(psi) -sin(psi);
58           sin(psi)  cos(psi)];;%rotation matrix
59     v_pos=R_psi*mr_co;
60     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
61     hold on, grid on
62     axis([-1 3 -1 3]), axis square
63     plot(eta(1,1:i),eta(2,1:i),'b-');
64     legend('MR','Path')
65     set(gca,'fontsize',24)
66     xlabel('x,[m]'); ylabel('y,[m]');
67     pause(0.1);
68     hold off
69 end %animation ends here
```

Appendix B

inverse kinematic animation

```
1 clear;
2 clc;
3
4 %%Simulation parameters
5 dt = 0.1;    %step size
6 ts = 10;     %simulation time
7 t = 0:dt:ts;%time span
8
9 %initial conditions
10 x0 = 0;
11 y0 = 0;
12 psi0 = 0;
13
14 eta0 = [x0;y0;psi0];
15
16 eta(:,1) = eta0; %first element of the column matrix
17
18 %loop starts here
19 for i = 1:length(t)
20     psi = eta(3,i);%current orientation in rad
21     %jacobian matrix
22     J_psi = [cos(psi) -sin(psi) 0;
```

```

23         sin(psi) cos(psi) 0;
24         0 0 1;];
25
26     %Desired states (generalized coordinates)
27     eta_d = [2-2*cos(0.1*t(i)); 2*sin(0.1*t(i)); 0.1*t(i)];
28     eta_d_dot = [2*0.1*sin(0.1*t(i)); 2*0.1*cos(0.1*t(i));
29     0.1;]; %derivative
30     %Vector of velocity input commands
31     zeta(:,i) = inv(J_psi) * eta_d_dot;
32     %time derivative of generalized coordinates
33     eta_dot(:,i)=J_psi * zeta(:,i);
34
35     %Position propagation using Euler method
36     eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %state
37     update
38 end
39
40 %plotting
41 figure
42 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
43 hold on
44 plot(t, eta(2,1:i), 'b--'); %y in m
45 plot(t, eta(3,1:i), 'm-.'); %t in s
46 legend('x, [m]', 'y, [m]', '/psi, [rad]');
47 set(gca, 'fontsize', 24)
48 xlabel('t, [s]');
49 ylabel('/eta, [units]');
50
51 %animation (mobile robot motion animation)
52 l = 0.6; % length of mobile robot
53 w = 0.4; % width of the mobile robot
54 %Mobile robot coordinates
55 mr_co=[-1/2 1/2 1/2 -1/2 -1/2;
56         -w/2 -w/2 w/2 w/2 -w/2;];

```



```
55 figure
56 for i = 1:length(t) %animation starts here
57     psi = eta(3,i);
58     R_psi=[cos(psi) -sin(psi);
59           sin(psi)  cos(psi)];%rotation matrix
60     v_pos=R_psi*mr_co;
61     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
62     hold on, grid on
63     axis([-1 3 -1 3]), axis square
64     plot(eta(1,1:i),eta(2,1:i),'b-');
65     legend('MR','Path')
66     set(gca,'fontsize',24)
67     xlabel('x,[m]'); ylabel('y,[m]');
68     pause(0.1);
69     hold off
70 end %animation ends here
```

Appendix C

diff wheel animation

```
1 clear;
2 clc;
3
4 %%Simulation parameters
5 dt = 0.1;    %step size
6 ts = 100;    %simulation time
7 t = 0:dt:ts;%time span
8
9 %vehicle(mobile robot) parameters (physical)
10 a = 0.2;    %radius of the wheel (fixed)
11 d = 0.5;    %distance b/t wheels
12
13 %initial conditions
14 x0 = 0.5;
15 y0 = 0.5;
16 psi0 = pi/4;
17
18 eta0 = [x0;y0;psi0];
19
20 eta(:,1) = eta0; %first element of the column matrix
21
22 %loop starts here
```

```

23 for i = 1:length(t)
24     psi = eta(3,i);%current orientation in rad
25     %jacobian matrix
26     J_psi = [cos(psi) -sin(psi) 0;
27              sin(psi) cos(psi) 0;
28              0 0 1];
29     %inputs
30     omega_1 = 0.5; %left wheel angular velocity
31     omega_2 = 0.2; %right wheel angular velocity
32
33     omega = [omega_1;omega_2];
34
35     %wheel configuration matrix
36     W = [a/2 a/2;
37          0 0;
38          -a/(2*d) a/(2*d)];
39
40     %velocity input commands
41     zeta(:,i) = W*omega;
42     eta_dot(:,i)=J_psi * zeta(:,i);
43
44     eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler
45     method
46 end
47 %plotting
48 figure
49 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
50 hold on
51 plot(t, eta(2,1:i), 'b--'); %y in m
52 plot(t, eta(3,1:i), 'm-.'); %t in s
53 legend('x, [m]', 'y, [m]', '/psi, [rad]');
54 set(gca, 'fontsize', 24)
55 xlabel('t, [s]');

```

```

56 ylabel('/eta, [units]');
57
58 %animation (mobile robot motion animation)
59 l = 0.6; % length of mobile robot
60 w = 2*d; % width of the mobile robot
61 %Mobile robot coordinates
62 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
63         -w/2 -w/2 w/2 w/2 -w/2;];
64 figure
65 for i = 1:5:length(t) %animation starts here
66     psi = eta(3,i);
67     R_psi=[cos(psi) -sin(psi);
68           sin(psi)  cos(psi)];%rotation matrix
69     v_pos=R_psi*mr_co;
70     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
71     hold on, grid on
72     axis([-1 3 -1 3]), axis square
73     plot(eta(1,1:i),eta(2,1:i),'b-');
74     legend('MR','Path')
75     set(gca,'fontsize',24)
76     xlabel('x,[m]'); ylabel('y,[m]');
77     pause(0.01);
78     hold off
79 end %animation ends here

```

Appendix D

omni wheel animation

```
1 clear;
2 clc;
3
4 %%Simulation parameters
5 dt = 0.1;    %step size
6 ts = 100;    %simulation time
7 t = 0:dt:ts;%time span
8
9 %vehicle(mobile robot) parameters (physical)
10 a = 0.2;    %radius of the wheel (fixed)
11 l_w = 0.5; %distance of wheel
12
13 %initial conditions
14 x0 = 0.5;
15 y0 = 0.5;
16 psi0 = pi/4;
17
18 eta0 = [x0;y0;psi0];
19
20 eta(:,1) = eta0; %first element of the column matrix
21
22 %loop starts here
```

```

23 for i = 1:length(t)
24     psi = eta(3,i);%current orientation in rad
25     %jacobian matrix
26     J_psi = [cos(psi) -sin(psi) 0;
27              sin(psi) cos(psi) 0;
28              0 0 1;];
29     %inputs
30     omega_1 = 0.5; %1st wheel angular velocity
31     omega_2 = 0.5; %2nd wheel angular velocity
32     omega_3 = 0.5; %3rd wheel angular velocity
33     %forward 0,-0.5,0.5 rotation clockwise 0.5,-0.5,-0.5
    anticlockwise 0.5,-0.5,0.5
34
35     omega = [omega_1;omega_2;omega_3];
36
37     %wheel configuration matrix
38     W = a/3*[ 0 -sqrt(3) sqrt(3);
39              2 -1 -1;
40              1/l_w 1/l_w 1/l_w];
41
42     %velocity input commands
43     zeta(:,i) = W*omega;
44     eta_dot(:,i)=J_psi * zeta(:,i);
45
46     eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler
    method
47 end
48
49 %plotting
50 figure
51 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
52 hold on
53 plot(t, eta(2,1:i), 'b--'); %y in m
54 plot(t, eta(3,1:i), 'm-.'); %t in s

```

```

55 legend('x, [m]', 'y, [m]', '/psi, [rad]');
56 set(gca, 'fontsize', 24)
57 xlabel('t, [s]');
58 ylabel('/eta, [units]');
59
60 %animation (mobile robot motion animation)
61 l = 2*l_w; % length of mobile robot
62 w = 0.4; % width of the mobile robot
63 %Mobile robot coordinates
64 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
65         -w/2 -w/2 w/2 w/2 -w/2;];
66 figure
67 for i = 1:5:length(t) %animation starts here
68     psi = eta(3,i);
69     R_psi=[cos(psi) -sin(psi);
70           sin(psi)  cos(psi)];%rotation matrix
71     v_pos=R_psi*mr_co;
72     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i), 'g')
73     hold on, grid on
74     axis([-1 3 -1 3]), axis square
75     plot(eta(1,1:i),eta(2,1:i), 'b-');
76     legend('MR', 'Path')
77     set(gca, 'fontsize', 24)
78     xlabel('x, [m]'); ylabel('y, [m]');
79     pause(0.01);
80     hold off
81 end %animation ends here

```

Appendix E

mecanum wheel animation

```
1 clear;
2 clc;
3
4 %%Simulation parameters
5 dt = 0.1;    %step size
6 ts = 10;     %simulation time
7 t = 0:dt:ts;%time span
8
9 %vehicle(mobile robot) parameters (physical)
10 a = 0.2;    %radius of the wheel (fixed)
11 l_w = 0.3;  %distance b/t wheels
12 d_w = 0.5;  %%distance b/t wheel and frame
13
14 %initial conditions
15 x0 = 0.5;
16 y0 = 0.5;
17 psi0 = 0;
18
19 eta0 = [x0;y0;psi0];
20
21 eta(:,1) = eta0; %first element of the column matrix
22
```



```

23 %loop starts here
24 for i = 1:length(t)
25     psi = eta(3,i);%current orientation in rad
26     %jacobian matrix
27     J_psi = [cos(psi) -sin(psi) 0;
28              sin(psi) cos(psi) 0;
29              0 0 1;];
30     %inputs
31     omega_1 = 0.5; %1st wheel angular velocity
32     omega_2 = 0.5; %2nd wheel angular velocity
33     omega_3 = 0.5; %3rd wheel angular velocity
34     omega_4 = 0.5; %4th wheel angular velocity
35     %forward w,w,w,w rotation clockwise w,w,-w,-w
anticlockwise -w,-w,w,w
36
37     omega = [omega_1;omega_2;omega_3;omega_4];
38
39     %wheel configuration matrix
40     W = a/4*[1 1 1 1;
41              1 -1 1 -1;
42              -1/(d_w-l_w) -1/(d_w-l_w) 1/(d_w-l_w) 1/(d_w-
43              l_w)];
44
45     %velocity input commands
46     zeta(:,i) = W*omega;
47     eta_dot(:,i)=J_psi * zeta(:,i);
48
49     eta(:,i+1) = eta(:,i) + dt * eta_dot(:,i); %Euler
method
50
51 end
52
53 %plotting
54 figure
55 plot(t, eta(1,1:i), 'r-'); %for plotting x in m

```

```

54 hold on
55 plot(t, eta(2,1:i), 'b--');      %y in m
56 plot(t, eta(3,1:i), 'm-.');      %t in s
57 legend('x, [m]', 'y, [m]', '/psi, [rad]');
58 set(gca, 'fontsize', 24)
59 xlabel('t, [s]');
60 ylabel('/eta, [units]');
61
62 %animation (mobile robot motion animation)
63 l = 2*l_w; % length of mobile robot
64 w = 2*d_w; % width of the mobile robot
65 %Mobile robot coordinates
66 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
67         -w/2 -w/2 w/2 w/2 -w/2];
68 figure
69 for i = 1:5:length(t) %animation starts here
70     psi = eta(3,i);
71     R_psi=[cos(psi) -sin(psi);
72            sin(psi)  cos(psi)]; %rotation matrix
73     v_pos=R_psi*mr_co;
74     fill((v_pos(1,:)+eta(1,i)), v_pos(2,:)+eta(2,i), 'g')
75     hold on, grid on
76     axis([-1 3 -1 3]), axis square
77     plot(eta(1,1:i), eta(2,1:i), 'b-');
78     legend('MR', 'Path')
79     set(gca, 'fontsize', 24)
80     xlabel('x, [m]'); ylabel('y, [m]');
81     pause(0.01);
82     hold off
83 end %animation ends here

```

Appendix F

dynamic model animation

```
1 clear;
2 clc;
3
4 %simulation parameters
5 dt=0.1; %step size
6 ts = 10; %simulation time
7 t = 0:dt:ts; %time span
8
9 %initial conditions
10 eta0 =[0;0;0]; %inital position and orientation of the
    vehicle
11 zeta0 = [0;0;0];%intial velocity of input commands
12
13 eta(:,1) = eta0;
14 zeta(:,1) = zeta0;
15
16 %robot parameters
17 m=10; %mass of the vehicle
18 Iz = 0.1; %inertia of the vehicle
19
20 xbc = 0; ybc = 0; %coordinates of mass centre
21
```

```

22 %state propagation
23
24 for i = 1:length(t)
25     u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);
26
27     %inertia matrix
28     D = [m 0 -ybc*m;
29          0 m xbc*m;
30          -ybc*m xbc*m Iz+m*(xbc^2+ybc^2)];
31     %other vector
32     n_v = [-m*r*(v+xbc*r);
33            m*r*(u-ybc*r);
34            m*r*(xbc*u+ybc*v)];
35     %input vector
36     tau(:,i) = [1;0.5;0];
37
38     %Jacobian matrix
39     psi = eta(3,i);
40     J_eta = [cos(psi) -sin(psi) 0;
41              sin(psi) cos(psi) 0;
42              0 0 1];
43
44     zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
45     zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);
46
47     eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
48     zeta_dot(:,i))); %state update
49
50 %plotting
51 figure
52 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
53 hold on
54 plot(t, eta(2,1:i), 'b--'); %y in m

```

```

55 plot(t, eta(3,1:i), 'm-.');      %t in s
56 legend('x, [m]', 'y, [m]', '/psi, [rad]');
57 set(gca, 'fontsize', 24)
58 xlabel('t, [s]');
59 ylabel('/eta, [units]');
60
61 %animation (mobile robot motion animation)
62 l = 0.6; % length of mobile robot
63 w = 0.4; % width of the mobile robot
64 %Mobile robot coordinates
65 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
66         -w/2 -w/2 w/2 w/2 -w/2];
67
68 figure
69 for i = 1:length(t) %animation starts here
70     psi = eta(3,i);
71     R_psi=[cos(psi) -sin(psi);
72           sin(psi)  cos(psi)];%rotation matrix
73     v_pos=R_psi*mr_co;
74     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i), 'g')
75     hold on, grid on
76     axis([-1 3 -1 3]), axis square
77     plot(eta(1,1:i),eta(2,1:i), 'b-');
78     legend('MR', 'Path')
79     set(gca, 'fontsize', 24)
80     xlabel('x, [m]'); ylabel('y, [m]');
81     pause(0.1);
82     hold off
83 end %animation ends here

```

Appendix G

diff wheel dynamic

```
1 clear;
2 clc;
3
4 %simulation parameters
5 dt=0.1; %step size
6 ts = 10; %simulation time
7 t = 0:dt:ts; %time span
8
9 %initial conditions
10 eta0 =[0;0;0]; %inital position and orientation of the
    vehicle
11 zeta0 = [0;0;0];%intial velocity of input commands
12
13 eta(:,1) = eta0;
14 zeta(:,1) = zeta0;
15
16 %vehicle(mobile robot) parameters (physical)
17 a = 0.2; %radius of the wheel (fixed)
18 d = 0.5; %distance b/t wheels
19
20 %robot parameters
21 m=10; %mass of the vehicle
```

```

22 Iz = 0.1; %inertia of the vehicle
23
24 xbc = 0; ybc = 0; %coordinates of mass centre
25
26 %state propagation
27
28 for i = 1:length(t)
29     u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);
30
31     %inertia matrix
32     D = [m 0 -ybc*m;
33          0 m xbc*m;
34          -ybc*m xbc*m Iz+m*(xbc^2+ybc^2)];
35
36     %other vector
37     n_v = [-m*r*(v+xbc*r);
38            m*r*(u-ybc*r);
39            m*r*(xbc*u+ybc*v)];
40
41     %Wheel input matrix (differential wheel drive)
42     Gamma = [1 1;
43              0 0;
44              -d d];
45
46     %Wheel inputs (traction forces)
47     F1 = 0.5; %left wheel force
48     F2 = 0.5; %right wheel force
49
50     kappa = [F1;F2];
51
52     %Input vector
53     tau(:,i) = Gamma*kappa;
54
55     %Jacobian matrix
56     psi = eta(3,i);
57     J_eta = [cos(psi) -sin(psi) 0;

```

```

56         sin(psi) cos(psi) 0;
57         0 0 1;];
58
59     zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
60     zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);
61
62     eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
        zeta_dot(:,i))); %state update
63 end
64
65 %plotting
66 figure
67 plot(t, eta(1,1:i), 'r-');      %for plotting x in m
68 hold on
69 plot(t, eta(2,1:i), 'b--');      %y in m
70 plot(t, eta(3,1:i), 'm-.');      %t in s
71 legend('x, [m]', 'y, [m]', '/psi, [rad]');
72 set(gca, 'fontsize', 24)
73 xlabel('t, [s]');
74 ylabel('/eta, [units]');
75
76 %animation (mobile robot motion animation)
77 l = 0.6; % length of mobile robot
78 w = 0.4; % width of the mobile robot
79 %Mobile robot coordinates
80 mr_co=[-l/2 l/2 l/2 -l/2 -l/2;
81         -w/2 -w/2 w/2 w/2 -w/2;];
82
83 figure
84 for i = 1:length(t) %animation starts here
85     psi = eta(3,i);
86     R_psi=[cos(psi) -sin(psi);
87            sin(psi) cos(psi)]; %rotation matrix
88     v_pos=R_psi*mr_co;

```



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

Appendix H

omni wheel dynamics

```
1 clear;
2 clc;
3
4 %simulation parameters
5 dt=0.1; %step size
6 ts = 10; %simulation time
7 t = 0:dt:ts; %time span
8
9 %initial conditions
10 eta0 =[0;0;0]; %inital position and orientation of the
    vehicle
11 zeta0 = [0;0;0];%intial velocity of input commands
12
13 eta(:,1) = eta0;
14 zeta(:,1) = zeta0;
15
16 %vehicle(mobile robot) parameters (physical)
17 a = 0.2; %radius of the wheel (fixed)
18 d = 0.5; %distance b/t wheels
19
20 %robot parameters
21 m=10; %mass of the vehicle
```

```

22 Iz = 0.1; %inertia of the vehicle
23
24 xbc = 0; ybc = 0; %coordinates of mass centre
25
26 %wheel configuartion parameters
27 l=0.3;
28 phi1 = 90*pi/180; phi2 = 210*pi/180; phi3 = 330*pi/180;
29
30 %state propagation
31 for i = 1:length(t)
32     u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);
33
34     %inertia matrix
35     D = [m 0 -ybc*m;
36          0 m xbc*m;
37          -ybc*m xbc*m Iz+m*(xbc^2+ybc^2)];
38     %other vector
39     n_v = [-m*r*(v+xbc*r);
40            m*r*(u-ybc*r);
41            m*r*(xbc*u+ybc*v)];
42
43     %Wheel input matrix (differential wheel drive)
44     Gamma = [cos(phi1) cos(phi2) cos(phi3);
45              sin(phi1) sin(phi2) sin(phi3);
46              1 1 1];
47
48     %Wheel inputs (traction forces)
49     F1 = 0;
50     F2 = -0.5;
51     F3 = 0.5; %lateral 1,-0.5,-0.5 rotate 1,0.5,0.5 forward
52     kappa = [F1;F2;F3];
53
54     %Input vector

```

```

55     tau(:,i) = Gamma*kappa;
56
57     %Jacobian matrix
58     psi = eta(3,i);
59     J_eta = [cos(psi) -sin(psi) 0;
60              sin(psi) cos(psi) 0;
61              0 0 1];
62
63     zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
64     zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);
65
66     eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
67     zeta_dot(:,i))); %state update
68
69 %plotting
70 figure
71 plot(t, eta(1,1:i), 'r-'); %for plotting x in m
72 hold on
73 plot(t, eta(2,1:i), 'b--'); %y in m
74 plot(t, eta(3,1:i), 'm-.'); %t in s
75 legend('x, [m]', 'y, [m]', '/psi, [rad]');
76 set(gca, 'fontsize', 24)
77 xlabel('t, [s]');
78 ylabel('/eta, [units]');
79
80 %animation (mobile robot motion animation)
81 l = 0.6; % length of mobile robot
82 w = 0.6; % width of the mobile robot
83 %Mobile robot coordinates
84 mr_co = [-1/2 1/2 1/2 -1/2 -1/2;
85          -w/2 -w/2 w/2 w/2 -w/2];
86
87 figure

```

```
88 for i = 1:length(t) %animation starts here
89     psi = eta(3,i);
90     R_psi=[cos(psi) -sin(psi);
91            sin(psi)  cos(psi)];%rotation matrix
92     v_pos=R_psi*mr_co;
93     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
94     hold on, grid on
95     axis([-1 3 -1 3]), axis square
96     plot(eta(1,1:i),eta(2,1:i),'b-');
97     legend('MR','Path')
98     set(gca,'fontsize',24)
99     xlabel('x,[m]'); ylabel('y,[m]');
100    pause(0.1);
101    hold off
102 end %animation ends here
```

Appendix I

mecanum wheel dynamics

```
1 clear;
2 clc;
3
4 %simulation parameters
5 dt=0.1; %step size
6 ts = 10; %simulation time
7 t = 0:dt:ts; %time span
8
9 %initial conditions
10 eta0 =[0;0;0]; %inital position and orientation of the
    vehicle
11 zeta0 = [0;0;0];%intial velocity of input commands
12
13 eta(:,1) = eta0;
14 zeta(:,1) = zeta0;
15
16 %vehicle(mobile robot) parameters (physical)
17 a = 0.2; %radius of the wheel (fixed)
18 d = 0.5; %distance b/t wheels
19
20 %robot parameters
21 m=10; %mass of the vehicle
```

```

22 Iz = 0.1; %inertia of the vehicle
23
24 xbc = 0; ybc = 0; %coordinates of mass centre
25
26 %wheel configuartion parameters
27 l=0.3;
28 phi1 = 90*pi/180; phi2 = 210*pi/180; phi3 = 330*pi/180;
29
30 %state propagation
31 for i = 1:length(t)
32     u = zeta(1,i); v = zeta(2,i); r = zeta(3,i);
33
34     %inertia matrix
35     D = [m 0 -ybc*m;
36          0 m xbc*m;
37          -ybc*m xbc*m Iz+m*(xbc^2+ybc^2)];
38     %other vector
39     n_v = [-m*r*(v+xbc*r);
40            m*r*(u-ybc*r);
41            m*r*(xbc*u+ybc*v)];
42
43     %Wheel input matrix (differential wheel drive)
44     Gamma = 1/sqrt(2)*[1 1 1 1;
45                        1 -1 1 -1;
46                        1-d 1-d -(1-d) -(1-d)];
47
48     %Wheel inputs (traction forces)
49     F1 = 0.2;
50     F2 = 0.2;
51     F3 = 0.2;
52     F4 = 0.2; %lateral f,-f,f,-f rotate f,f,-f,-f forward f
,f,f,f
53     kappa = [F1;F2;F3;F4];
54

```

```

55     %Input vector
56     tau(:,i) = Gamma*kappa;
57
58     %Jacobian matrix
59     psi = eta(3,i);
60     J_eta = [cos(psi) -sin(psi) 0;
61              sin(psi) cos(psi) 0;
62              0 0 1;];
63
64     zeta_dot(:,i) = inv(D)*(tau(:,i)-n_v);
65     zeta(:,i+1) = zeta(:,i) + dt*zeta_dot(:,i);
66
67     eta(:,i+1) = eta(:,i) + dt*(J_eta*(zeta(:,i)+dt*
68     zeta_dot(:,i))); %state update
69 end
70 %plotting
71 figure
72 plot(t, eta(1,1:i), 'r-');      %for plotting x in m
73 hold on
74 plot(t, eta(2,1:i), 'b--');    %y in m
75 plot(t, eta(3,1:i), 'm-.');    %t in s
76 legend('x, [m]', 'y, [m]', '/psi, [rad]');
77 set(gca, 'fontsize', 24)
78 xlabel('t, [s]');
79 ylabel('/eta, [units]');
80
81 %animation (mobile robot motion animation)
82 l = 0.6; % length of mobile robot
83 w = 0.6; % width of the mobile robot
84 %Mobile robot coordinates
85 mr_co = [-1/2 1/2 1/2 -1/2 -1/2;
86          -w/2 -w/2 w/2 w/2 -w/2;];
87

```



```
88 figure
89 for i = 1:length(t) %animation starts here
90     psi = eta(3,i);
91     R_psi=[cos(psi) -sin(psi);
92           sin(psi)  cos(psi)];%rotation matrix
93     v_pos=R_psi*mr_co;
94     fill((v_pos(1,:)+eta(1,i)),v_pos(2,:)+eta(2,i),'g')
95     hold on, grid on
96     axis([-1 3 -1 3]), axis square
97     plot(eta(1,1:i),eta(2,1:i),'b-');
98     legend('MR','Path')
99     set(gca,'fontsize',24)
100     xlabel('x,[m]'); ylabel('y,[m]');
101     pause(0.1);
102     hold off
103 end %animation ends here
```

Appendix J

Fixed path Differential Drive Motion

```
1 %% Simulation parameters
2 dt = 0.1;    %step size
3 ts = 25;     %simulation time
4 t = 0:dt:ts;%time span
5
6 %vehicle(mobile robot) parameters (physical)
7 r = 0.2;     %radius of the wheel (fixed)
8 b = 0.5;     %distance b/t wheels (base)
9
10 %initial conditions
11 x0 = 2;
12 y0 = 2;
13 psi0 = 0;
14
15 eta0 = [x0;y0;psi0];
16 eta = zeros(3,numel(t));    % Pose matrix
17 eta(:,1) = eta0; %first element of the column matrix
18 %% Path planning
19 % Load map and inflate it by a safety distance
20 close all
21 load exampleMap
22 inflate(map,r);
```

```
23 %inflate(map,radius) inflates each occupied position of the
24 % map by the radius given in meters.radius is rounded up to
25 % the nearest cell equivalent based on the resolution of
    the map.
26 % Every cell within the radius is set to true (1).
27
28 %%Desired route or waypoints
29 waypoints = [2 2;
30              4 4;
31              4 8;
32              11 8;
33              11 11]
34 %% Pure Pursuit Controller
35 controller = controllerPurePursuit;
36 controller.Waypoints = waypoints;
37 controller.LookaheadDistance = 0.35;
38 controller.DesiredLinearVelocity = 0.75;
39 controller.MaxAngularVelocity = 1.5;
40
41 %% Create visualizer
42 load exampleMap % Reload original (uninflated) map for
    visualization
43 viz = Visualizer2D;
44 viz.hasWaypoints = true;
45 viz.mapName = 'map';
46
47 %% Simulation loop starts here
48 rate = rateControl(1/dt);
49 for i = 1:numel(t)
50     % Run the Pure Pursuit controller and convert output
        to wheel speeds
```

```

51     [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.
52
53
54     % Inverse loop = inverseKinematics(dd,vRef,wRef)
55     % Calculates wheel speeds from linear and angular
velocity
56     wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)
/obj.wheelRadius;
57     wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)
/obj.wheelRadius;
58
59
60     % Compute the velocities, [v,w] = forwardKinematics(dd,
wL,wR);
61     % Calculates linear and angular velocity from wheel
speeds
62     v = 0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
63     w = (wR-wL)*r/b; %w = (wR-wL)*obj.wheelRadius/obj.
wheelBase;
64     zetaB = [v;0;w]; % Body velocities [vx;vy;w]
65
66     % Convert from body to world
67     psi = eta(3,i);%current orientation in rad
68     %jacobian matrix
69     J_psi = [cos(psi) -sin(psi) 0;
70             sin(psi) cos(psi) 0;
71             0 0 1;];
72     zeta = J_psi*zetaB;
73
74     % Perform forward discrete integration step
75     eta(:,i+1) = eta(:,i) + zeta*dt;

```

```
76  
77     % Update visualization  
78     viz(eta(:,i+1),waypoints)  
79     waitfor(rate);  
80 end
```

Appendix K

2.Fixed path Differential Drive Motion

```
1 %% Simulation parameters
2 dt = 0.1;    %step size
3 ts = 70;     %simulation time
4 t = 0:dt:ts;%time span
5
6 %vehicle(mobile robot) parameters (physical)
7 r = 0.2;     %radius of the wheel (fixed)
8 b = 0.5;     %distance b/t wheels (base)
9
10 %initial conditions
11 x0 = 5;
12 y0 = 5;
13 psi0 = 0;
14
15 eta0 = [x0;y0;psi0];
16 eta = zeros(3,numel(t));    % Pose matrix
17 eta(:,1) = eta0; %first element of the column matrix
18 %% Path planning
19 % Load map and inflate it by a safety distance
20 % close all
21 % load exampleMap
22 map = binaryOccupancyMap(100,50,1);
```

```
23 walls = zeros(100,50);
24 walls(1,1:100) = 1; % Top wall
25 walls(50,:) = 1; % Bottom wall
26 walls(:,1) = 1; % Left wall
27 walls(:,100) = 1; % Right wall
28 %vertical walls
29 walls(30:40,10) = 1;
30 walls(10:20,10) = 1;
31 walls(30:50,20) = 1;
32 walls(1:10,20) = 1;
33 walls(20:40,30) = 1;
34 walls(30:50,40) = 1;
35 walls(30:40,50) = 1;
36 walls(1:30,60) = 1;
37 walls(20:40,70) = 1;
38 walls(10:30,80) = 1;
39 walls(40:50,80) = 1;
40 walls(1:20,90) = 1;
41
42 %Horizontal walls
43 walls(20,10:30) = 1;
44 walls(10,20:40) = 1;
45 walls(10,50:60) = 1;
46 walls(40,50:70) = 1;
47 walls(30,60:70) = 1;
48 walls(10,70:80) = 1;
49 walls(40,80:90) = 1;
50 walls(30,90:100) = 1;
51
52 setOccupancy(map,[1 1],walls,"grid");
53 inflate(map,r);
54 %inflate(map,radius) inflates each occupied position of the
55 % map by the radius given in meters.radius is rounded up to
```

```
56 % the nearest cell equivalent based on the resolution of
    the map.
57 % Every cell within the radius is set to true (1).
58
59 %Desired route or waypoints
60 waypoints = [5 5;
61              15 5;
62              15 25;
63              25 25;
64              25 5;
65              35 5;
66              35 25;
67              45 25;
68              45 5;
69              75 5;
70              75 15;
71              95 15;
72              95 5];
73
74 %% Pure Pursuit Controller
75 controller = controllerPurePursuit;
76 controller.Waypoints = waypoints;
77 controller.LookaheadDistance = 1 ;
78 controller.DesiredLinearVelocity = 3;
79 controller.MaxAngularVelocity = 3;
80
81 %% Create visualizer
82 setOccupancy(map,[1 1],walls,"grid"); % Reload original (
    uninflated) map for visualization
83 viz = Visualizer2D;
84 viz.hasWaypoints = true;
85 viz.mapName = 'map';
86
87 %% Simulation loop starts here
```



```

88 rate = rateControl(1/dt);
89 for i = 1:numel(t)
90     % Run the Pure Pursuit controller and convert output
    to wheel speeds
91     [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.
92
93
94     % Inverse loop = inverseKinematics(dd,vRef,wRef)
95     % Calculates wheel speeds from linear and angular
    velocity
96     wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)
    /obj.wheelRadius;
97     wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)
    /obj.wheelRadius;
98
99
100    % Compute the velocities, [v,w] = forwardKinematics(dd,
    wL,wR);
101    % Calculates linear and angular velocity from wheel
    speeds
102    v =0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
103    w = (wR-wL)*r/b; %w = (wR-wL)*obj.wheelRadius/obj.
    wheelBase;
104    zetaB = [v;0;w]; % Body velocities [vx;vy;w]
105
106    % Convert from body to world
107    psi = eta(3,i);%current orientation in rad
108    %jacobian matrix
109    J_psi = [cos(psi) -sin(psi) 0;
110             sin(psi) cos(psi) 0;
111             0 0 1;];

```

```
112     zeta = J_psi*zetaB;
113
114     % Perform forward discrete integration step
115     eta(:,i+1) = eta(:,i) + zeta*dt;
116
117     % Update visualization
118     viz(eta(:,i+1),waypoints)
119     waitfor(rate);
120 end
```

Appendix L

Map2

```
1 myMap = binaryOccupancyMap(100,50,1)
2 walls = zeros(100,50);
3 walls(1,1:100) = 1; % Top wall
4 walls(50,:) = 1; % Bottom wall
5 walls(:,1) = 1; % Left wall
6 walls(:,100) = 1; % Right wall
7 %vertical walls
8 walls(30:40,10) = 1;
9 walls(10:20,10) = 1;
10 walls(30:50,20) = 1;
11 walls(1:10,20) = 1;
12 walls(20:40,30) = 1;
13 walls(30:50,40) = 1;
14 walls(30:40,50) = 1;
15 walls(1:30,60) = 1;
16 walls(20:40,70) = 1;
17 walls(10:30,80) = 1;
18 walls(40:50,80) = 1;
19 walls(1:20,90) = 1;
20
21 %Horizontal walls
22 walls(20,10:30) = 1;
```

```
23 walls(10,20:40) = 1;
24 walls(10,50:60) = 1;
25 walls(40,50:70) = 1;
26 walls(30,60:70) = 1;
27 walls(10,70:80) = 1;
28 walls(40,80:90) = 1;
29 walls(30,90:100) = 1;
30
31 setOccupancy(myMap,[1 1],walls,"grid")
32 show(myMap)
33 inflateRadius =0.2
34 inflate(myMap,inflateRadius)
35 show(myMap)
36 %%
37 map = binaryOccupancyMap(100,50,1)
38 setOccupancy(map,[1 1],walls,"grid");
39 setOccupancy(myMap,[1 1],walls,"grid")
40 show(myMap)
```

Appendix M

Auto path Differential Drive motion

```
1 %% Simulation parameters
2 dt = 0.1;    %step size
3 ts = 25;     %simulation time
4 t = 0:dt:ts;%time span
5
6 %vehicle(mobile robot) parameters (physical)
7 r = 0.2;     %radius of the wheel (fixed)
8 b = 0.5;     %distance b/t wheels (base)
9
10 %initial conditions
11 x0 = 2;
12 y0 = 2;
13 psi0 = 0;
14
15 eta0 = [x0;y0;psi0];
16 eta = zeros(3,numel(t));    % Pose matrix
17 eta(:,1) = eta0; %first element of the column matrix
18 %% Path planning
19 % Load map and inflate it by a safety distance
20 close all
21 load exampleMap
22 inflate(map,r);
```

```
23 %inflate(map,radius) inflates each occupied position of the
24 % map by the radius given in meters.radius is rounded up to
25 % the nearest cell equivalent based on the resolution of
    the map.
26 % Every cell within the radius is set to true (1).
27
28 % Create a Probabilistic Road Map (PRM) for mobileRobotPRM
    object and define associated attributes
29 planner = mobileRobotPRM(map);
30 planner.NumNodes = 75;
31 planner.ConnectionDistance = 5;
32
33 % Find a path in PRM using A* algorithm from the start
    point to a specified goal point
34 %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.
35 startPoint = eta0(1:2)';
36 goalPoint = [11, 11];
37 waypoints = findpath(planner,startPoint,goalPoint);
38 show(planner)
39
40 %% Pure Pursuit Controller
41 controller = controllerPurePursuit;
42 controller.Waypoints = waypoints;
43 controller.LookaheadDistance = 0.35;
44 controller.DesiredLinearVelocity = 0.75;
45 controller.MaxAngularVelocity = 1.5;
46
47 %% Create visualizer
48 load exampleMap % Reload original (uninflated) map for
    visualization
49 viz = Visualizer2D;
50 viz.hasWaypoints = true;
```

```

51 viz.mapName = 'map';
52
53 %% Simulation loop starts here
54 rate = rateControl(1/dt);
55 for i = 1:numel(t)
56     % Run the Pure Pursuit controller and convert output
    to wheel speeds
57     [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.
58
59
60     % Inverse loop = inverseKinematics(dd,vRef,wRef)
61     % Calculates wheel speeds from linear and angular
    velocity
62     wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)
    /obj.wheelRadius;
63     wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)
    /obj.wheelRadius;
64
65
66     % Compute the velocities, [v,w] = forwardKinematics(dd,
    wL,wR);
67     % Calculates linear and angular velocity from wheel
    speeds
68     v = 0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
69     w = (wR-wL)*r/b; %w = (wR-wL)*obj.wheelRadius/obj.
    wheelBase;
70     zeta = [v;0;w]; % Body velocities [vx;vy;w]
71
72     % Convert from body to world
73     psi = eta(3,i);%current orientation in rad
74     %jacobian matrix

```

```
75     J_psi = [cos(psi) -sin(psi) 0;
76              sin(psi) cos(psi) 0;
77              0 0 1;];
78     eta_dot = J_psi*zeta;
79
80     % Perform forward discrete integration step
81     eta(:,i+1) = eta(:,i) + eta_dot*dt;
82
83     % Update visualization
84     viz(eta(:,i+1),waypoints)
85     waitfor(rate);
86 end
```


Appendix N

2.Auto path Differential Drive motion

```
1 %% Simulation parameters
2 dt = 0.1;    %step size
3 ts = 60;     %simulation time
4 t = 0:dt:ts;%time span
5
6 %vehicle(mobile robot) parameters (physical)
7 r = 0.2;    %radius of the wheel (fixed)
8 b = 0.5;    %distance b/t wheels (base)
9
10 %initial conditions
11 x0 = 5;
12 y0 = 5;
13 psi0 = 0;
14
15 eta0 = [x0;y0;psi0];
16 eta = zeros(3,numel(t));    % Pose matrix
17 eta(:,1) = eta0; %first element of the column matrix
18 %% Path planning
19 % Load map and inflate it by a safety distance
20 % close all
21 % load exampleMap
22 map = binaryOccupancyMap(100,50,1);
```

```
23 walls = zeros(100,50);
24 walls(1,1:100) = 1; % Top wall
25 walls(50,:) = 1; % Bottom wall
26 walls(:,1) = 1; % Left wall
27 walls(:,100) = 1; % Right wall
28 %vertical walls
29 walls(30:40,10) = 1;
30 walls(10:20,10) = 1;
31 walls(30:50,20) = 1;
32 walls(1:10,20) = 1;
33 walls(20:40,30) = 1;
34 walls(30:50,40) = 1;
35 walls(30:40,50) = 1;
36 walls(1:30,60) = 1;
37 walls(20:40,70) = 1;
38 walls(10:30,80) = 1;
39 walls(40:50,80) = 1;
40 walls(1:20,90) = 1;
41
42 %Horizontal walls
43 walls(20,10:30) = 1;
44 walls(10,20:40) = 1;
45 walls(10,50:60) = 1;
46 walls(40,50:70) = 1;
47 walls(30,60:70) = 1;
48 walls(10,70:80) = 1;
49 walls(40,80:90) = 1;
50 walls(30,90:100) = 1;
51
52 setOccupancy(map,[1 1],walls,"grid");
53 inflate(map,r);
54 %inflate(map,radius) inflates each occupied position of the
55 % map by the radius given in meters.radius is rounded up to
```

```
56 % the nearest cell equivalent based on the resolution of
    the map.
57 % Every cell within the radius is set to true (1).
58
59 % Create a Probabilistic Road Map (PRM) for mobileRobotPRM
    object and define associated attributes
60 planner = mobileRobotPRM(map);
61 planner.NumNodes = 500;
62 planner.ConnectionDistance = 10;
63
64 % Find a path using PRM algorithm from the start point to a
    specified goal point
65 %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.
66 startPoint = eta0(1:2)';
67 goalPoint = [95,5];
68 waypoints = findpath(planner,startPoint,goalPoint);
69 show(planner)
70
71 %% Pure Pursuit Controller
72 controller = controllerPurePursuit;
73 controller.Waypoints = waypoints;
74 controller.LookaheadDistance = 1 ;
75 controller.DesiredLinearVelocity = 3;
76 controller.MaxAngularVelocity = 4;
77
78 %% Create visualizer
79 setOccupancy(map,[1 1],walls,"grid"); % Reload original (
    uninflated) map for visualization
80 viz = Visualizer2D;
81 viz.hasWaypoints = true;
82 viz.mapName = 'map';
83
```

```

84 %% Simulation loop starts here
85 rate = rateControl(1/dt);
86 for i = 1:numel(t)
87     % Run the Pure Pursuit controller and convert output
      to wheel speeds
88     [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.
89
90
91     % Inverse loop = inverseKinematics(dd,vRef,wRef)
92     % Calculates wheel speeds from linear and angular
      velocity
93     wL = (vRef - wRef*b/2)/r; %wL = (v - w*obj.wheelBase/2)
      /obj.wheelRadius;
94     wR = (vRef + wRef*b/2)/r; %wR = (v + w*obj.wheelBase/2)
      /obj.wheelRadius;
95
96
97     % Compute the velocities, [v,w] = forwardKinematics(dd,
      wL,wR);
98     % Calculates linear and angular velocity from wheel
      speeds
99     v = 0.5*r*(wL+wR); %v = 0.5*obj.wheelRadius*(wL+wR);
100    w = (wR-wL)*r/b; %w = (wR-wL)*obj.wheelRadius/obj.
      wheelBase;
101    zetaB = [v;0;w]; % Body velocities [vx;vy;w]
102
103    % Convert from body to world
104    psi = eta(3,i);%current orientation in rad
105    %jacobian matrix
106    J_psi = [cos(psi) -sin(psi) 0;
107             sin(psi) cos(psi) 0;

```

```
108         0 0 1;];
109     zeta = J_psi*zetaB;
110
111     % Perform forward discrete integration step
112     eta(:,i+1) = eta(:,i) + zeta*dt;
113
114     % Update visualization
115     viz(eta(:,i+1),waypoints)
116     waitfor(rate);
117 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.