REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

CONTROL OF AUTONOMOUS MOBILE ROBOTS USING REINFORCEMENT LEARNING

Celalettin Doğukan ENGİN

MASTER OF SCIENCE THESIS

Department of Mechatronics Engineering

Mechatronics Engineering Program

Advisor

Assoc. Prof. Dr. Aydın YEŞİLDİREK

May, 2019

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

LOCATION ANALYSIS OF THE EMERGENCY SERVICE CENTERS OF A CASE COMPANY

A thesis submitted by Celalettin Doğukan ENGİN in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE is approved by the committee on 06.07.2019 in Department of Mechatronics Engineering, Mechatronics Engineering Program.

Assoc. Prof. Dr. Aydın YEŞİLDİREK

Yıldız Technical University

Advisor

Approved By the Examining Committee

Assoc. Prof. Dr. Aydın YEŞİLDİREK, Advisor
Yıldız Technical University

Assoc. Prof. Dr. Cüneyt YILMAZ, Member
Yıldız Technical University

Prof. Dr. Fikret ÇALIŞKAN, Member
İstanbul Technical University

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of Control of Autonomous Mobile Robots Using Reinforcement Learning supervised by my supervisor, Assoc. Prof. Dr. Aydın YEŞİLDİREK. In the case of a discovery of false statement, I am to acknowledge any legal consequence.

Celalettin Doğukan ENGİN

Signature

Dedicated to my family

and my love Şirin.

ACKNOWLEDGEMENTS

First thing first, I would like to express my gratefulness to my instructive advisor Assoc. Prof. Dr. Aydın Yeşidirek for the continuous support of my master's study and research. His guidance helped me through the research and writing of this thesis.

Also, I appreciate the guidance of the defense committee for perfecting this study with their recommendations and advices.

Last but not the least, I would like to thank the love of my life Şirincan for bearing the tough days of preparing this work. Also, I thank my family: my father İlyas for being my forever supporter and the light in my way, my mother Aydan for giving me life and taking care of me, and as a best friend rather than just a brother Baturalp for being a great example for hardworking.

Celalettin Doğukan Engin

TABLE OF CONTENTS

LIST OF SYMBOLS

a	Current Action
A	Action Vector
b	Belief State
b'	Next Belief State
$B(q)$	Input Matrix
b_T	Total damping
C	Center of mass of the Robot
d	Distance of the Robot from Origin Point
E_π	Expected value for policy and discount factor
F	Force
$F(\dot{q})$	surface friction matrix
$G(q)$	Gravitational vector
h	Robot height
I	Inertial Moment
J_{load}	Inertia of the Load(Robot itself)
J_m	Motor Inertia
J_T	Total Inertia
K_b	Motor back EMF constant
K_D	Derivative Gain
K_I	Integral Gain
K_p	Proportional Gain
K_t	Motor Torque Constant
K_{tac}	Tachometer Constant
L	Half Distance between Wheels
La	Armature Inductance
M	Markov Decision Process tuple Matrix
m	Mass
$M(q)$	Symmetric positive definite inertial matrix
n	Gear Ratio
N	Filter Coefficient
O	Observation Vector
o	Observation
Q*	Optimal Action-Value Function
r	Reward
R	Radius of Wheel
$R(\theta)$	Orthogonal Rotation Matrix
R_a	Armature Resistance
s	Current State
S	State Vector
s'	Next State

t	Current Time
T	Transition Matrix
V	Value
$V(q, \dot{q})$	centripetal matrix
V^*	Optimal Value Function
V_L	Left Wheel Translational Velocity
V_R	Right Wheel Translational Velocity
v_{robot}	Translational Velocity of Robot
V_{in}	Input Voltage
X_I	X Position of Robot on Inertial Frame
X_R	X Position of Robot on Robot Frame
Y_I	Y Position of Robot on inertial Frame
Y_R	Y Position of Robot on Robot Frame
Z	Observation Function
γ	Discount Factor
θ	Orientation of the Robot on Inertial Frame
λ	Lagrange multiplier vector
$\Lambda^T(q)$	kinematic constraints matrix
π	Policy
π^*	Optimal Policy
τ	Torque
φ_L	Left Wheel Angular Displacement
φ_R	Right Wheel Angular Displacement
ω	Angular Velocity of Wheel
ω_{robot}	Angular Velocity of Robot
θ_i	Q-network parameter(weight)

LIST OF ABBREVIATIONS

AHRS	Attitude and Heading Reference System
AI	Artificial Intelligence
DC	Direct Current
DP	Dynamic Programming
DQN	Deep Q-Network
EMF	Electro Magnetic Field
FIVE	Fuzzy Rule Interpolation in Vague Environment
FQ	Fuzzy Q-learning
FRIQ	Fuzzy Rule Interpolation based Q-learning
GPI	Generalized Policy Iteration
IMU	Inertial Measurement Unit
IR	Infrared Reflective
MCM	Monte Carlo Methods
MDP	Markov Decision Process
PEAS	Performance Environment Actuators Sensors
PI	Policy Iteration
PID	Proportional Interagral Derivative
PMDC	Permanent Magnet Direct Current
POMDP	Partially Observable Markov Decision Processes
RL	Reinforcement Learning
SARSA	State Action next State next Action
TDM	Temporal difference Methods
VE	Vague Environment

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Control of Autonomous Mobile Robots Using Reinforcement Learning

Celalettin Doğukan ENGİN

Department of Mechatronics Engineering

Master of Science Thesis

Advisor: Assoc. Prof. Dr. Aydın YEŞİLDİREK

Modern robots are designed for complicated tasks and control operations, such as manipulating objects, assisting experts, navigating and exploring unknown territories. Designing a classical control algorithm for robots to execute these tasks is a complicated process. The designer has to consider all the possibilities and situations that robot may face. This is an intense time-consuming process. So, a better solution according to pre-programming a robot for all the tasks, is that the robot to learn such tasks by their selves. One of the most useful method for self-training of modern robots, is reinforcement learning.

This thesis presents a comparative model based study of different reinforcement learning algorithms such as Q-Learning, State Action next State next Action(SARSA), Deep Q-Network (DQN) and Fuzzy Rule Interpolation based Q-learning(FRIQ) in terms of the total time and number of the steps to reach the goal. For accomplishing this aim, a Simulink model of a two-wheel differential drive mobile robot, which consists of a chassis with two fixed and in line with each other electric motors, has been developed and placed in a two-dimensional maze. Model is based not only on dynamic and kinematic analysis of robot motion but also its tachometer and proximity sensors. Velocity control of each electric motor has been done by individual controllers by the help of tachometer feedback and orientation control of the robot is done thanks to the yaw angle sensor feedback. The robot is equipped with proximity sensors to determine the distances of objects within its surroundings. By the help of the feedback of each proximity sensor, reinforcement learning algorithms determines a motion strategy to self-train itself and explore the unknown environment.

Key-words: artificial intelligence, reinforcement learning, autonomous robot, control.

ÖZET

Otonom Mobil Robotların Pekiştirmeli Öğrenme ile Kontrolü

Celalettin Doğukan ENGİN

Mekatronik Mühendisliği Bölümü

Yüksek Lisans Tezi

Danışman: Doç. Dr. Aydın YEŞİLDİREK

Modern robotlar; nesneleri yönetmek, uzmanlara yardım etmek, bilinmeyen bölgelerde gezinmek ve buraları keşfetmek gibi zor görevler için tasarlanmışlardır. Robotların bu görevleri yerine getirmesi için klasik bir kontrol algoritması tasarlamak ise oldukça karmaşık bir işlemdir. Tasarımcı, robotun karşılaşabileceği tüm olasılıkları ve durumları göz önünde bulundurmalıdır. Bu zaman alıcı bir süreç olup, algoritma tasarımcısının bilinmeyen bir çevre için tüm olasılıkları göz önünde bulundurabilmesi açısından çok da makul ve gerçekleştirilebilir olmayan bir iştir. Bu nedenle, bir robotun akla gelebilecek tüm olasılıklara göre yapması gereken görevler için önceden programlanması işine kıyasla, robotların bu tür görevleri ve olasılıkları kendi başına öğrenmesi daha iyi bir çözüm olarak karşımıza çıkmaktadır. Modern robotların kendi kendilerini eğitmesi için en kullanışlı yöntemlerden biri pekiştirmeli öğrenmedir.

Yapılan çalışma; Q tipi öğrenme, Durum Aksiyon sonraki Durum sonraki Aksiyon(SARSA), Derin Q Ağı(DQN) ve Bulanık Kural Interpolasyonu Temelli Q tipi öğrenme(FRIQ) gibi farklı pekiştirmeli öğrenme algoritmalarının, bir robot modelinin iki boyutlu bir labirent ortamında başlangıç noktasından hedefe ulaşana kadar geçen toplam zaman ve adım sayısına bağlı olarak performans kıyasını içermektedir. Bu amaçla iki adet elektrik motorunun direk olarak iki arka tekerleği tahrik ettiği ve bu tekerleklerin tek bir şaside birleştiği diferansiyel sürüşlü bir mobil robotun Simulink ortamında modeli geliştirildi. Daha sonra bu robot modeli iki boyutlu bir labirentin içine yerleştirildi. Bu çalışma, robotun dinamik ve kinematik analizlerini, takometre, yönelim açısı ve yakınlık sensörlerinin modellenmesini içermektedir. İki farklı elektrik motorunun hız kontrolü yine farklı kontrolcüler tarafından kendi üzerlerine takılı olan takometrelerin geri beslemeleri ile yapılmıştır. Yönelim açısı sensörünün geri beslemesi sayesinde robot oryantasyonunun kontrolü gerçekleştirilmiştir. Robot, etrafındaki nesnelere olan mesafesini belirlemek için yakınlık sensörleriyle 45 derecelik açıyla her yöne olacak şekilde donatılmıştır. Her bir yakınlık sensörünün geri beslemesi sayesinde, robot bulunduğu durumu tespit eder ve pekiştirmeli öğrenme algoritmaları vasıtasıyla kendini her yeni adımda eğiterek içerisinde bulunduğu çevreyi keşfeder.

Anahtar kelimeler: yapay zeka, pekiştirmeli öğrenme, otonom robot, kontrol.

1
Introduction

Robots are widely used in industry and in our daily environments. Traditional robots that are being used as physically fixed in their industrial environment where is mostly controlled. With the invention of mobile robots that are able to move autonomously in an unpredictable or partially unknown environment, it is possible to use them in various applications like medical service, space exploration, underwater survey tasks etc.

Xia briefly summarizes autonomous navigation with below sub-features [1]:

1. Perception: collecting environmental information through sensors.
2. Exploration: strategy for direction selection.
3. Mapping: using sensor information to build the representation or the model of the environment.
4. Localization: estimating the robot position on spatial map.
5. Path planning: finding a path towards to goal.
6. Path execution: executing the set of actions with the optimal route for reaching the goal including adapting environmental changes and obstacle avoidance.

It is a complicated process for mobile robots to execute complex tasks with a classical control algorithm. The designer has to consider all the possibilities and situations that robot may face. This is an intense time-consuming process. So, a better solution according to pre-programming a robot for various missions, is that the robot to learn such tasks by their selves. One of the most useful method for self-training of modern robots, is reinforcement learning (RL).

Figure 1. 1 Various AI Applications

The powerful and legit applications of mobile robots such as Curiosity which is the NASA's Mars exploration robot at upper left-hand corner, Google's self-driving car at upper right-hand corner, Blackghost which is an autonomous underwater assault robot at lower left-hand corner, Kiva which is the warehouse organizer robot of Amazon at lower right-hand corner of Figure 1.1 motivated this study from the beginning. This dissertation focuses on the robots that can learn, explore and improve their capabilities autonomously, make rational decisions.

1.1 Literature Review

Machine learning simply enables an agent to get experienced by an example data or experiences. The main concepts of machine learning can be identified as:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning
4. Hybrid Learning Approaches

Table 1. 1 Comparison of Main Machine Learning Concepts

Supervised Learning	**Unsupervised Learning**	**Reinforcement Learning**
Learning to approximate reference answers	Learning underlying data structure	Learning optimal strategy by trial and error
Needs correct answers	No feedback required	Needs feedback on agent's own actions
Model does not affect the input data	Model does not affect the input data	Agent can affect it's own observations

Table 1.1 shows the advantages and the disadvantages between main learning concepts. Hybrid learning approaches such as supervised/unsupervised or supervised/reinforcement learning combinations often lead to good results. For example, supervised/unsupervised hybrid learning approach is widely being used in banking sector to detect anomalies on accounts transactions.

In supervised learning, there should be an access to the examples of explicit target for each single input as pairs for training purposes. Supervised learning which finds patterns in the dataset of right answers to learn from, can be categorized under two groups: regression and classification. Regression is valid for the cases that involves estimating or predicting a continuous quantity. This method is widely used in statistics for prediction and forecasting. Classification is used when there is a need of assigning multiple observations into different categories rather than estimating continues quantities.

Unsupervised learning tasks are the case when there is unobservable or infeasible targets, outputs, or answers from dataset. These tasks are the problem of clustering which is simply consisted of categorizing the observations which are relatively similar to or different from each other. This method widely used for marketing problems such as identifying buying habits or segments of customer.

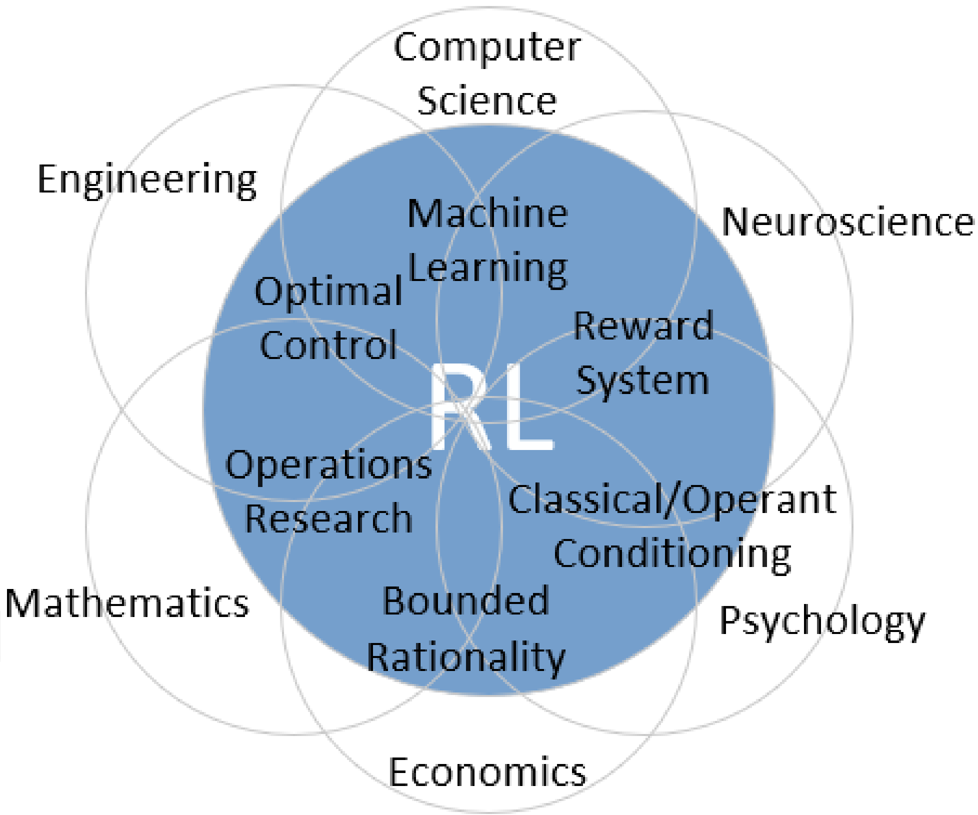


Figure 1. 2 Application Fields of RL

A relatively newer learning type is called RL which is used when there is not a dataset of input-output pair for training purposes. When there is an infinite or enormous solution space, RL dominates an agent to interact with its environment for various tasks. RL find itself a wide range of application area which can be described by Figure 1.2.

When the previous works has been checked, the followings has been observed. Altuntas is focused on mobile robot navigation by developing a comparative study between the most popular RL algorithms: Q-learning and SARSA. She worked on a system that avoids the obstacles and reaches to the goal. As a result she found out that SARSA converges to optimal values faster than Q-learning with a learning rate of 90% to 70% [2]. Zhang investigated the use of deep RL for robot navigation in unknown territories by developing a network based on A3C architecture which uses depth images, elevation maps and three dimensional orientation of the robot as inputs [3]. Stone demonstrate the performance comparison of different epsilon valued Q-Learning methods on RoboCup-Soccer platform and declared that the smaller the epsilon value shorter the learning duration [4]. Kopsa implemented RL on the autorotation of a simple point mass helicopter model for achieving a reduction in Height-Velocity restriction curve using MCM [5]. Tutsoy implemented a central pattern based RL for the control of humanoid robot leg's each joint and found out that TDM is not fast enough for this task [6]. Coors presented the deep convolutional neural networks with Q-learning algorithm on a mobile robot which is in human environment and declared that continuous agent is able to learn a good navigation strategy faster however the quality of the best policy learnt is higher for discrete control agent [7]. Yao introduced a new model-based RL algorithm called LAM-API which learns linear action modes and then use them for off-policy evaluation and control [8]. Kamalapurkar developed a new model based RL algorithm and rigorous Lyanupov-based methods and implemented this algorithm on a trajectory tracking problem for performance evaluation [9]. Liang is also presented a model-based RL which is named IRL/PILCO algorithm but this time he applied it onto a high-dimensional space [10]. Farahmand worked for a comparative study for visual servoing which includes model-based and model-free RL algorithms and find out that both methods are performing relatively same [11]. Bruin introduced a method that used two experience replay databases with different overwrite policies and for sampling experiences from both databases deep RL is being used [12]. Mnih and Lillicrap represented a comparative study between DQN and Q-learning and declared that the performance of DQN is better relatively [13], [14]. Ren implemented deep RL for a soft robotic application and presented the implementation challenges [15]. Gu presented an asynchronous deep off-policy RL approach that can be used for complex robotic manipulation skills [16]. Zhang experimented deep RL on a real robot in a maze-like environment, and observed that deep RL speeds up the performance [17]. Sheoran made a comparative analysis of FRIQ-learning and Q-learning methods for maze problems and declared that FRIQ-learning is converging and finding the goal faster [18]. Vincze made a comparative analysis study of FIVE FRIQ-learning method and standard FRI method and declared that FIVE FRIQ-learning method is advantageous according to other one [19].

1.2 Objectives of the Study

This paper consists of a model-based study of the comparison of most popular RL algorithms such as Q-Learning, SARSA, DQN and FRIQ. For the aim of accomplishing this comparative study, a realistic and fully defined Simulink model of a differential drive mobile robot, which consists of a chassis with two fixed and in line with each other electric motors, has been developed and placed on a two-dimensional environment which can simply be named as a maze. Model is not only based on dynamic and kinematic analysis of robot motion but also its tachometer and proximity sensors. Velocity control of each electric motor has been done by individual controllers by the help of tachometer feedback. The robot is equipped with proximity sensors to determine the obstacles within its surroundings. By the help of the feedback of each proximity sensor, RL algorithms determines a motion strategy to self-train itself and explore the unknown environment. Thus, under same conditions the performance of different algorithms are compared to each other with respect to the convergence to goal state in terms of steps and timing.

1.3 Hypothesis

This study focuses on the smart control in autonomous navigation by the sensory outputs of robot learning. Even for the users without programming ability, the aim of this study to introduce the robot capabilities to be more adaptable and extendable for the unknown situations. For this purpose, the different RL algorithms are compared by controlling a realistic differential drive robot model with those decision-making algorithms on a well-defined two-dimensional environment to enable it for reaching the goal position.

For a healthy construction of the simulation environment, and mobile robot model that is intended to be enabled for navigating autonomously by RL based control algorithms, it is a must consider all the details of the system requirements. Initially, it is necessary to specify the performance measure embodying criterion for success. This is also called by Russel as PEAS (Performance, Environment, Actuators, Sensors) description [20].

At this point, performance can be specified by below items:

1. Reaching the destination
2. Minimizing trip duration
3. Maximizing safety by avoiding obstacles.

Partial observability which is caused by the noisy or inaccurate sensor information or the missing parts of the environment because of the sensor capability, makes the environment stochastic. So that the next state of the robot is defined by the current state or the unclear results of the latest actions of the robot.

Third criteria of PEAS is the actuator which is in this case are the two DC electrical motors that are fixed to rear wheels. These DC motors are controlled by separated PID circuits as a low-level controller. Setpoints to those PID controllers comes from RL based high-level controllers. The detailed explanation of this level by level control system is presented in Chapter 3.

Last criteria of PEAS is sensors which are the interaction and interface of the robot with the environment. Knudson categorized the sensors as internal and external state sensors [21]. Internal state sensors ensure the internal data of the motion of the robot such as acceleration, velocity, position. Gyroscope and tachometer which are also being used during the mobile robot model development of this dissertation, is also two of the good examples for this sensor type. External state sensors provide the external information about the environment such as visual, infrared, laser sensors. For the robot model to interact with the environment eight proximity sensors are placed on the robot to detect the obstacles within its surrounding.

Since the system has a complex structure, a machine learning algorithm is necessary for the robot to learn how to navigate autonomously. Since it is inconvenient to design all possible situations to teach the agent, classical coding method is not feasible or since there isn't any example data, a supervised learning algorithm cannot be used.

1.4 Organization of the Dissertation

This study consists of five chapters and organization details are given by followings briefly.

- In Chapter 1, the outline and the focus of this thesis is defined with respect to the results of literature survey. The significance of this study is introduced with the simulation environment, and the general perspective of the mobile robot model that has been developed for the benchmark purpose of the decision-making algorithms of RL.
- Chapter 2 begins mobile robot concepts in the literature and continues with the mathematical model of the differential drive mobile robot including: the electrical actuators, proximity sensors, orientation sensor, tachometer, robot kinematics and dynamics of mobile robot. For the aim of speeding up the simulations to have a benchmark data between different RL algorithms, the Simulink model has been linearized. Finally, by the test drive robot's motion, sensor feedbacks and the interaction with its environment are validated.
- Chapter 3 represents the different reinforcement learning methods including the similarities and the differences between them. The robot control proposition is also given by this chapter.
- Chapter 4 introduces the comparable and measurable performance results of various RL algorithm which are SARSA, Q-learning, DQN and FRIQ-learning. The comparison criteria include: time to reach goal, and the count of motion steps.
- In Chapter 5, the summary of the key points of this study and the discussion for further researches are given.

2
Mobile Robot Model

2.1 Mobile Robots

A wheeled mobile robot with a smart control system is capable of an autonomous motion. Mobile robots can be categorized according to their locomotion type. Mohareri identifies locomotion systems for mobile robots as [22]:

1. Differential Drive
2. Steered Wheels: Tricycles, Bicycles and Wagon
3. Synchronous Drive
4. Omni-Directional Drive
5. Car drive: Two-steer (Ackerman steering)

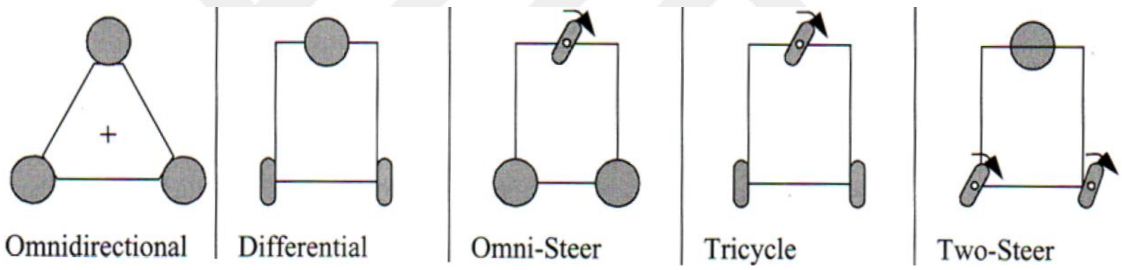


Figure 2. 1 Mobile Robot Locomotion Types [22]

The different types of locomotion systems for mobile robots are shown in Figure 2.1. Champion identifies these five type as according to the pair of indices as (W,H), where the W corresponds to mobility degree and H stands for the steer ability [23].

Omnidirectional robots do not have steering wheels and equipped by Castor or Swedish wheels which are able to move either way on a plane without changing their orientation. Thus, having no steering wheels equals steer ability to 0 and moving to any direction without changing its orientation equals mobility degree to 3. As a result, it can be declared that omnidirectional robots can be categorized under Type (3,0) robots. In contrast, other four classes of robots have restricted mobility which means their degree of mobility is less than 3.

Type (2,0) robots do not have conventional centered orientable wheels so steer ability goes to 0. These robots consist of single common axle with several conventional fixed wheels which allows only two-dimensional motion in the plane so mobility equals to 2. Differential drive robots are under this class.

Type (2,1) robots do not have conventional fixed wheel and at least one conventional centered oriental wheel which means no steering wheel with at least one fixed wheel with a common axle. No steering makes steering ability to 0 and two-dimensional motion in the plane equals mobility to 2. Tricycle robots are under this class.

Type (1,1) robots have one or more conventional fixed wheel on a single common axle with one or more conventional centered orientable wheels. Centers of steering wheels should not coincide with the fixed wheels on common axle. Orientability brings steering ability to 1. The mobility ability is restricted to the orientation of the steering wheel so it is only movable on one-dimensional plane which makes mobility to 1. Omni-steer robots can be an example to this type.

Type (1,2) do not have any conventional fixed wheels and involve at least two conventional centered orientable wheels. This makes steering ability to 2. The orientation of the two steering wheels restrict one-dimensional motion on the plane so steering ability goes to 1. Two-steer robots can be identified under this category.

Two-wheel differential drive robots which are classified under Type (2,0), are one of the most popular and used structures in mobile robot applications because of its simplicity. It consists of a chassis with two fixed in-line with each other electric actuators. This kind of robots do not have steering wheel and the mobility is restricted to two-dimensional plane. Electric motors are driven by the driver which gets commands from a controller that collects the required feedbacks through the sensors of robot. The third wheel which is also the front one as the fulcrum, is able to rotate freely in all directions and has negligible influence over the kinematics of the robot since it is non-actuated and dummy.

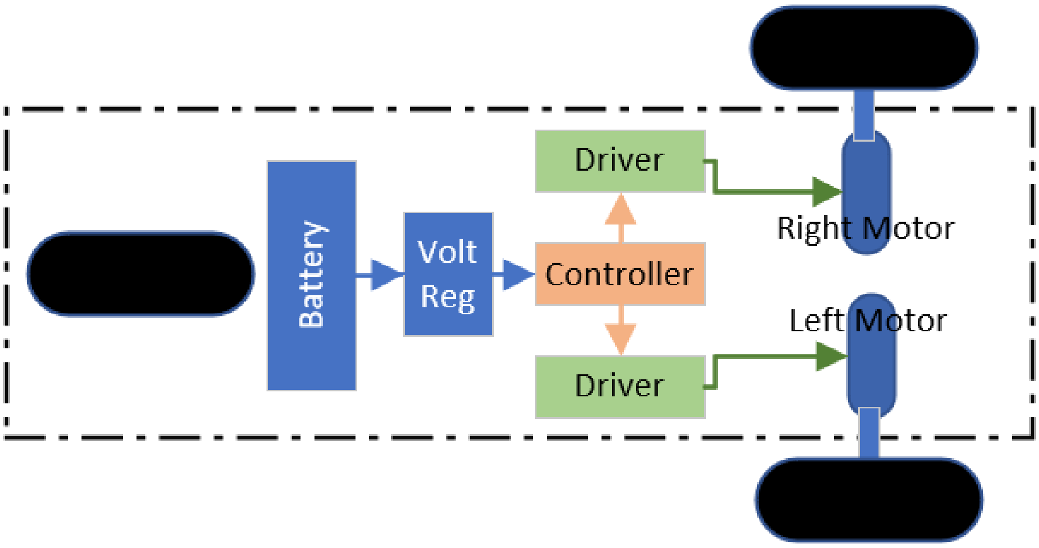


Figure 2. 2 Concept of Differential Drive Mobile Robot Model

Because of the modeling and controlling simplicity two-wheel differential drive robot has been chosen to study on. This robot model is developed by MATLAB. The concept of the robot model that has been developed for this study can be seen in Figure 2.2.

2.2 Mathematical Modeling

There are different ways of modeling a mobile robot, however the modeling in each way consists of kinematic and dynamic modeling. Kinematic modeling is focused on the pure motion with respect to the geometry and mathematics excluding the applied forces. Dynamic modeling is also the study motion with respect to the energies and applied forces. Actuator and sensor modeling are also necessary for building the control system.

In this dissertation, the developed robot model consists of two DC motors that are driven by two different PID controllers, a tachometer, eight proximity sensors for controller feedbacks. After building the accurate model it is necessary to linearize and simplify it for speeding up the simulation of different RL methods.

Two different coordinate systems have been defined for the purpose of determining and controlling the position of the mobile robot in its own plane.

2.2.1 Coordinate System

As mentioned above there is a need of defining two different coordinate systems.

1. Inertial Frame: This can be defined as the fixed reference coordinate system or the global frame which represents the environment that mobile robot moves in and denoted as $\{X_I, Y_I\}$.
2. Robot Frame: This is the local frame attached to mobile robot which is denominated as $\{X_R, Y_R\}$.

Dhaoudi showed the coordinate systems as in Figure 2.3 [24]. It is assumed that the robot is symmetrical according to the vertical axis of wheel axis. The origin of robot is the point A where is also in the middle between wheels. Center of mass of the robot is defined by point C which is at the distance *d* from the origin point A.

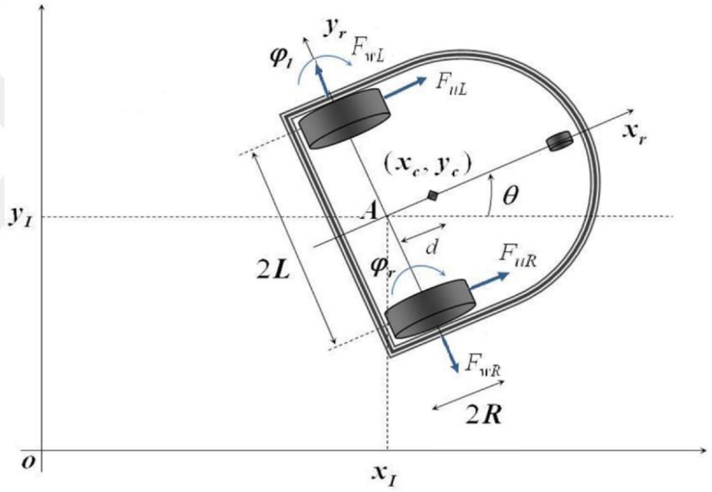


Figure 2. 3 Robot Position and Orientation in Inertial Frame [24]

The robot position in the inertial and robot frames are as below:

$$X^r = [x^r \quad y^r \quad \theta]^T \tag{2.1}$$

$$X^I = [x^I \quad y^I \quad \theta]^T \tag{2.2}$$

The translation between two coordinates according to a random given point is through orthogonal rotation matrix which is given in following equations.

$$X^I = R(\theta)X^r \tag{2.3}$$

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

2.2.2 Kinematic Model

Kinematic modeling of the robot is a prerequisite for the purpose of declaring the velocity of the robot as a function of the geometric parameters of the robot rather than applied forces. There are two types of kinematics: forward and inverse kinematics. These terms can be understood as a function and its inverse. Joint angles are the input and the coordinates of the end-effectors are the output for the forward kinematic models. For a redundant system it is necessary to have a forward kinematics since for the same end-effector coordinates it would not be possible to have unique configuration. So, in this model forward kinematics is the case.

Neimark and Fufaev represented the main assumptions during the development of the kinematic model as: no lateral slip motion and pure rolling [25]. To be more specific with the no lateral slip motion definition, it can be said that the robot is capable only for curvilinear motion no sideward.

$$\dot{y}^I = 0 \tag{2.6}$$

By this assumption the orthogonal matrix gives the velocity in inertial frame as following equation.

$$\dot{x}^I sin\theta + \dot{y}^I cos\theta = 0 \tag{2.7}$$

The pure rolling term refers to the fact each wheel is interfaced with the ground with the one single point which is observable in Figure 2.4 where is robot's wheel focused right view.

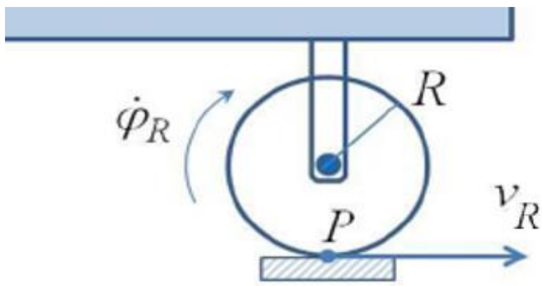


Figure 2. 4 Right Side View of the Robot Focusing Wheel Area

By avoiding the slipping and involving pure rolling constraints, the relationship of wheel's angular velocity and linear speed on the robot frame as following equations for right and left wheels in a row.

$$\begin{cases} V_R = R\dot{\varphi}_R \\ V_L = R\dot{\varphi}_L \end{cases} \tag{2.8}$$

Herewith, on the robot frame, the average linear speed of robot can be expressed as:

$$v_{robot} = \frac{V_R + V_L}{2} = R\,\frac{\dot{\varphi}_R + \dot{\varphi}_L}{2} \tag{2.9}$$

On the robot frame, the angular velocity of robot is predicated by the below equation where the radius of the wheel is R and the half distance between wheels is L.

$$\omega_{robot} = \frac{V_R - V_L}{2L} = R\,\frac{\dot{\varphi}_R - \dot{\varphi}_L}{2} \tag{2.10}$$

Thus, the velocities on the inertial frame can be expressed by the orthogonal matrix that has been presented by equation (2.4) and can be showed as the following equation which also stands for the forward kinematic model in this study.

$$\begin{bmatrix} \dot{x}^I \\ \dot{y}^I \\ \dot{\theta}^I \end{bmatrix} = \begin{bmatrix} \frac{Rcos\theta}{2} & \frac{Rcos\theta}{2} \\ \frac{Rcos\theta}{2} & \frac{Rcos\theta}{2} \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} \tag{2.11}$$

A further way to represent inertial frame velocities can be done by the robot frame velocities.

$$\begin{bmatrix} \dot{x}^I \\ \dot{y}^I \\ \dot{\theta}^I \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{robot} \\ \omega_{robot} \end{bmatrix} \tag{2.12}$$

This formation brings us the following equations which will be used through the model development for the relationship of the linear speeds of inertial and robot coordinate systems.

$$\begin{cases} V_X = \frac{V_R + V_L}{2} sin\theta \\ V_Y = -\frac{V_R + V_L}{2} cos\theta \end{cases} \tag{2.13}$$

2.2.3 Odometry of Differential Drive

It is supposed that the robot is rotating around the instantaneous center of curvature (ICC) with avoiding slippage and having a pure rolling like Figure 2.5 for the purpose of expressing the actual position of the robot on the inertial frame.

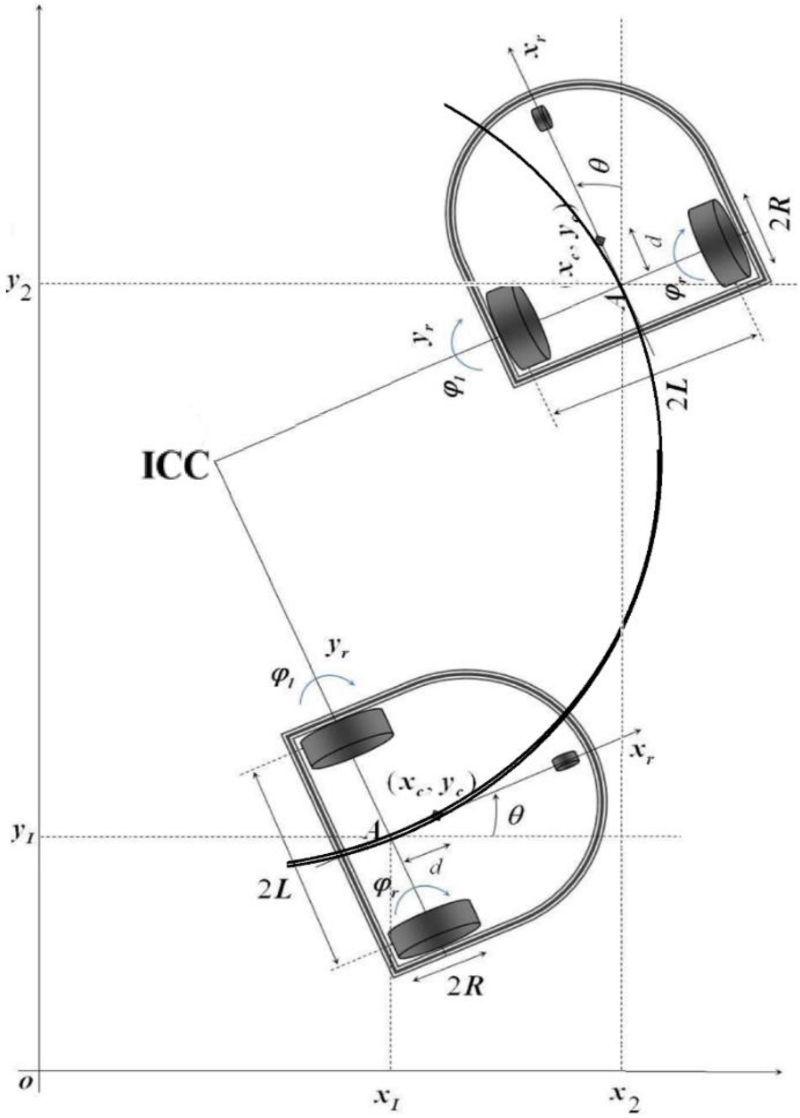


Figure 2. 5 Motion of Differential Drive Robot

For the purpose of defining the distance from ICC point to the mid-point A, lets express the ICC as point C. Thus, the line segment |CA| is found by using following equation.

$$|\mathrm{CA}| = L\frac{V_R + V_L}{V_R - V_L} \tag{2.14}$$

Maximilian expresses the trajectory of the robot by assuming the robot is moving on a straight-line tangent for the infinite short time [26]. During the infinite short time *dt*, robot is hovering around ICC with the angular velocity of $\omega_{robot}(t)$ and traveling along the curvilineal trajectory with a linear velocity of $v_{robot}(t)$. Based on these two velocity components the robot position on inertial frame and and the traveled distance in each direction can be expressed as following equations.

$$\begin{cases} x(t) = dx = X_2 - X_1 = V_X(t)dt \\ y(t) = dy = Y_2 - Y_1 \ = V_Y(t)dt \\ \theta(t) = d\theta = \theta_2 - \theta_1 \ = \omega(t)dt \end{cases} \tag{2.15}$$

Terms of X_1, X_2, Y_1 and Y_2 are the start and end points on inertial frame, these can also be observed on Figure 2.5. In addition, θ_1 and θ_2 are the initial and ending orientation of the robot and what can be observed by Figure 2.5 is that the difference between θ_1 and θ_2 is 90 degrees. By substituting the equation (2.9) with (2.10) and manipulating, we have:

$$\begin{cases} x(t) = \int v_{robot} \sin\big(\theta(t)\big)\, dt + X_0 \\ y(t) = \int v_{robot} \cos\big(\theta(t)\big)\, dt + Y_0 \\ \qquad \theta(t) = \int \omega(t) dt + \theta_0 \end{cases} \tag{2.16}$$

In order to improve the equation (2.16) for using it with model development on MATLAB, combined the equations (2.16) and (2.13), it can be reached to following equations.

$$\begin{cases} x(t) = \int \frac{V_R + V_L}{2} \sin\big(\theta(t)\big)\, dt + X_0 \\ y(t) = \int \frac{V_R + V_L}{2} \cos\big(\theta(t)\big)\, dt + Y_0 \\ \qquad \theta(t) = \int \frac{V_R - V_L}{2L} dt + \theta_0 \end{cases} \tag{2.17}$$

2.2.4 Electric Motor Model

Mostly used actuating systems in robotic systems are direct current motors. In this study a permanent magnet direct current (PMDC) motor has been used as electric actuator. Salem introduced a mobile robot model including the electric motor modeling considering the system dynamics and disturbances that are dependent to platform shape and dimension [27].

Unloaded open loop transfer function that inputs the input voltage *$V_{in}(s)$* and outputs the angular velocity *$\omega(s)$*, is given by below equation where K_t is the motor torque constant, R_a is the armature resistance, L_a is the armature inductance, J_m is the motor inertia, K_b is the motor back EMF constant, L is the mobile robot width, h is the mobile robot height and *n* is the gear ratio.

$$\frac{\omega(s)}{V_{in}(s)} = \frac{K_t}{(L_a s + R_a)(J_m s + b_m) + K_t K_b} \tag{2.18}$$

With the load input, total inertia J_T and the total damping b_T becomes as the following equations.

$$\begin{cases} b_T = b_m + b_{load} n^2 \\ J_T = J_m + J_{load} n^2 \end{cases} \tag{2.19}$$

J_{load} can be expressed as the inertia of the load which is the mobile robot itself in this study.

$$J_{load} = \frac{Lh^3}{12} \tag{2.20}$$

Thus, the loaded and geared open loop transfer function of the mobile robot becomes:

$$\frac{\omega_{robot}(s)}{V_{in}(s)} = \frac{K_t/n}{(L_a J_T)s^2 + (J_T R_a + b_T L_a)s + R_a b_{T+} K_t K_b} \tag{2.21}$$

The design parameters of the mobile robot is given in Appendix A.

2.2.5 Sensor Selection and Modeling

There is three kind of sensors used in this study: two tachometers to get the speed feedback of the mobile robot's each wheel, one orientation sensor, and eight proximity sensors to get the environmental feedback of the robot. An Infrared Reflective (IR) sensor is selected for this study.

2.2.5.1 Proximity Sensor Selection

There are many types of proximity sensors however IR sensors are used in this work. There is no direct equation to express the IR sensor transfer function so in this study Sharp GP2Y0A21YK™ IR distance proximity sensor is selected. Figure 2.6 involves the graph which gives the transfer function of the sensor. Datasheet of this sensor is given in Appendix B.

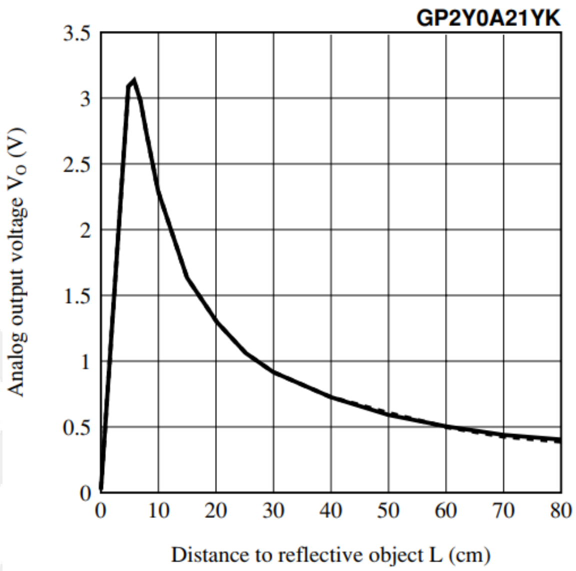


Figure 2. 6 Transfer Function of IR Sensor

It can be observed from the above graph that if there is any object with 3 cm or 12 cm away from the sensor, a 2V transducer voltage is read. This confusion can be vanished by assuming the distance of the object will be higher than 10 cm all the time. Thus, the right measurement is 12 cm so the robot will be all the time on the safe side.

For the interaction of the mobile robot with the environment eight IR sensors are placed on the robot platform with equal angle to each other which is 45 degrees. This is also observable by Figure 2.7.

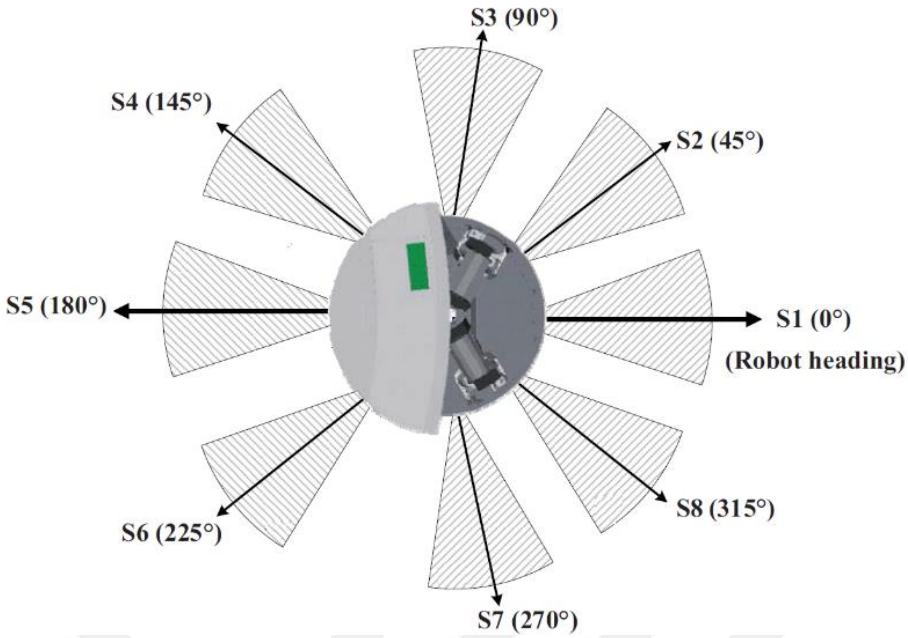


Figure 2. 7 Placement of the IR Sensors on the Robot

2.2.5.2 Tachometer Model

For the purpose of measuring the output angular speed of the wheel it is necessary to use tachometer. This measurement is used as a feedback for the low-level controller. For the dynamics of tachometer where K_{tac} stands for the tachometer constant, the following transfer function can be expressed. The tachometer constant that is used in this dissertation is given in Appendix A.

$$v_{robot}(t) = K_{tac}\frac{d\theta(t)}{dt} = K_{tac}\omega(t) \tag{2.22}$$

2.2.5.3 Orientation Sensor Model

Orientation feedback for heading(yaw) angle is crucially important in this two-dimensional system, since the high-level controller determines the heading angle for the next motion of the robot and forwards this request to medium-level controller that uses orientation feedback to create the linear speed input for the DC motor drivers.

There are many ways to determine the heading angle of the system. Gade, generalized heading estimation and determination methods under seven categories and represented a comparison between each method which are also observed by Figure 2.8, in terms of accuracy, prerequisites, and cost [28]. However, as he also told in his dissertation, these seven methods are not able to cover all the methods, but it gives a good overview. For example, Attitude and Heading Reference System (AHRS) is not included in his study. AHRS is a complex system that being used in aviation sector for determining the roll, pitch, and yaw angles of an aircraft. Main difference between an Inertial Measurement Unit(IMU) and AHRS is the on- board processing unit of AHRS. Since this study do not require this kind of an expensive and complex system for operation, it is unnecessary to deep dive on this system.

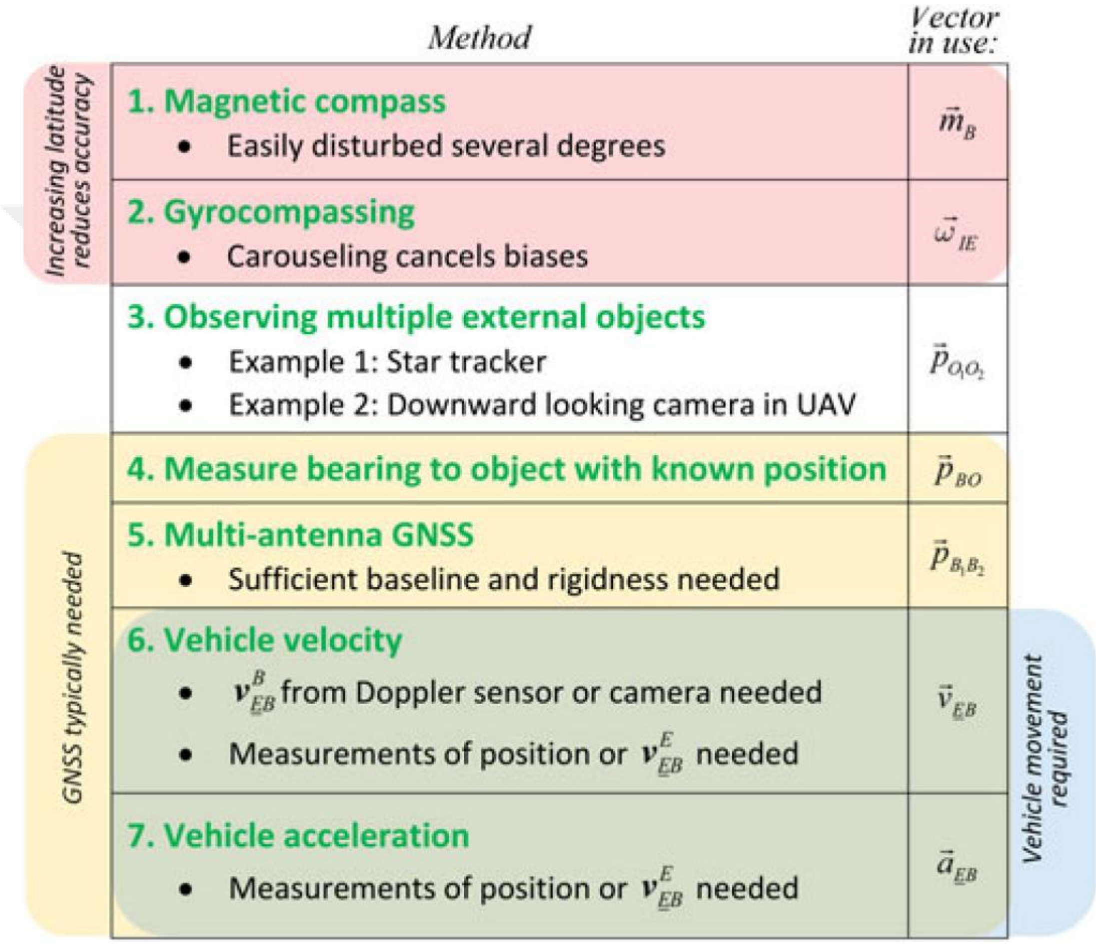


Figure 2. 8 Simplified Summary of Heading Estimation Methods [28]

Method 1 and Method 2 are dependent to Earth's movements. First one which is named as magnetic compass, is based on the relative magnetic difference of the heading according to the magnetic vector field of the earth. Second one which is named as gyro-compassing, is based on the angular velocity of the Earth as a vector relative to inertial space. Since the direction of this vector is known, it is possible to define the geographic north and south pole by this method. These methods are easily disturbed by other magnetic sources around, or the altitude where the system located since when the altitude increases magnetic power of the earth decreases. Since the heading accuracy for our system is significant, the highly distortable side of these methods, make them infeasible for our study.

Method 3 requires and imaging sensor such as camera or sonar to observe at least two objects which should have already been stored under a database for recognition purposes. For our case, this method is not feasible due to the need of an imaging sensor such as camera, and a database for object recognition.

For the rest of the methods it is necessary to know the current or initial position of the of the robot. This is accomplished by GNSS feedback. Method 4 combines Method 3 with GNSS feedback. For using Method 5 it is necessary to have multi-antenna GNSS which acts as body-fixed vector. Method 6 is mostly used for underwater applications which is also necessary to have a Doppler radar or underwater acoustic Doppler sensor. Thus, for these methods it is necessary to have a good GNSS coverage, sufficient space on robot, and robot platform rigidness. Eventually these three methods are not the case for our study.

Method 7 is based on the system's acceleration vector. This method is often available since it uses accelerometers. It is thus an important method for many MEMS-based navigation systems. For this method a horizontal acceleration feedback is required. The increasement in the availability of MEMS lead to a significant growth in IMUs. Nguyen has also represented a yaw angle-based control system for an autonomous vehicle using Kalman filter based disturbance observer [29]. The control logic that he has proposed which is also shown by Figure 2.9, involves a gyroscope, and a GNSS system as we also see with Method 7.

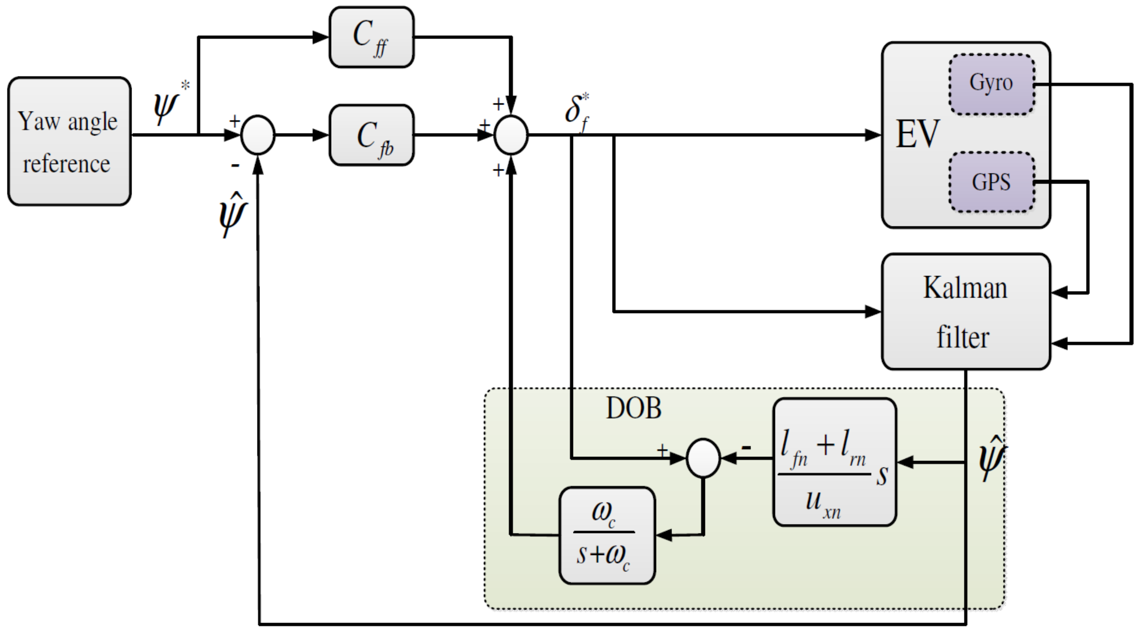


Figure 2. 9 An Application Example for Method Seven [28]

The last method and the application example of Nguyen is also the case for this dissertation if the study has been implemented in real life when concerning the other disturbances such as slipping. However, for the simulation case, it is necessary to represent two simplification for this methodology in order this method to be used in our system. First simplification is removing GPS out, since the initial position and the yaw angle of the mobile robot is known for our simulation case. Hall represented an orientation sensor calibration device which moves the orientation sensor to different angles in roll, pitch, and yaw planes in order to constitute a relationship between the output voltage, and the angle moved from one specific angle to another one [30]. The calibration application of the developed mechanism was on Microstrain 3DM-GX1™ gyro enhanced orientation sensor. The datasheet of this sensor is given in Appendix C. The trial consists of moving the sensor from one specific angle to another one as eventually this is also the case for our system. While there was only motion on yaw frame and no movement on the pitch and roll frames, the following relationship between the moving angle and sensor voltage output is developed. During the test, sensor was moved in increments of 30 degrees, the results showed a linear variation of voltage for moved angle and stable voltage for stationary angles.

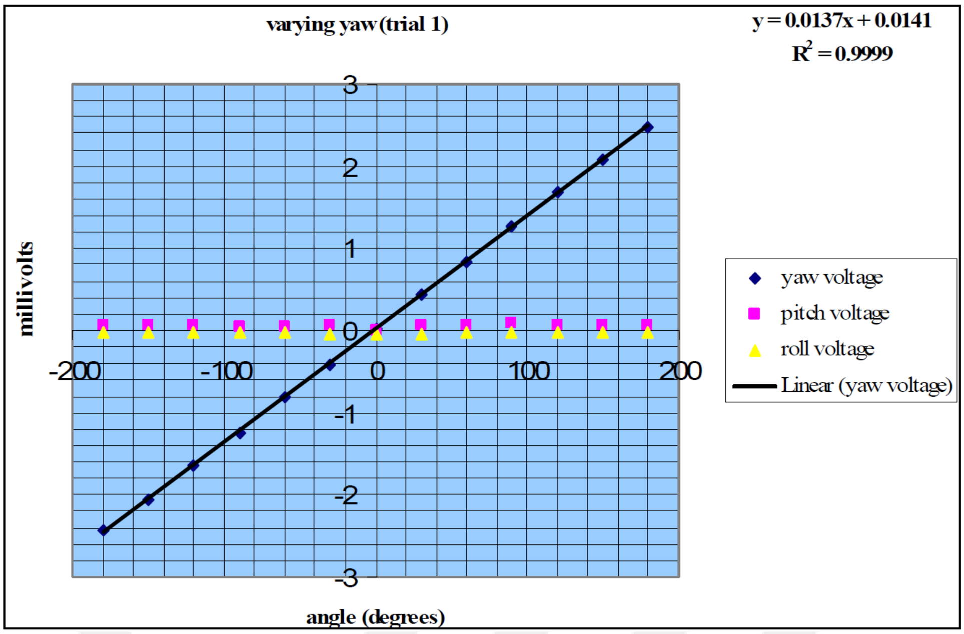


Figure 2. 10 Yaw angle versus Voltage Output Graph [30]

According to Hall's study, the lines of regression for these graphs showed nearly same and identical slopes with each other test. Since moving the heading angle by the minimum of 45 degrees for each orientation motion, is the case for our study, the second simplification for Method 7, is being used in this dissertation. In this study, the following transfer function that is generated by the measurements shown in Figure 2.10, is being used.

$$y = 0.0137x + 0.0141 \tag{2.23}$$

2.2.6 Dynamic Model

Neimark introduced the following equation of motion a non-holonomic differential drive mobile robot with generalized coordinates ($q_1, q_2, q_3, \ldots, q_m$) [25].

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)\tau - \lambda^T(q)^\lambda \tag{2.24}$$

Subject to the m constraints, $M(q)$ is an mxm symmetric positive definite inertia matrix, $V(q, \dot{q})$ is the centripetal matrix, $F(\dot{q})$ is the surface friction matrix, $G(q)$ is the gravitational vector, τ_d is the vector of unknown disturbances unmodeled dynamics, $B(q)$ is the input matrix, τ is the input vector, $\Lambda^T(q)$ is the matrix associated with the kinematic constraints, and λ is the Lagrange multipliers vector[36]. Dhaouadi simplified this general form into the following equation for the differential drive robots while the potential energy of the system is zero [24].

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} = B(q)\tau - \lambda^T(q)^\lambda \tag{2.25}$$

In the following two sections, an energy-based Lagrange approach, direct Newton-Euler approach for deriving dynamic equations of mobile robot system, has been explained.

2.2.6.1 Lagrange Approach

Lagrangian dynamics approach treats the system in terms of kinetic and potential energies by expressing them through systematically derived differential equations. Lagrange equation is generalized by the following form.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) + \frac{\partial L}{\partial q_i} = F - \lambda^T(q)^\lambda \tag{2.26}$$

L is the subtraction of P from T where the T refers to total kinetic energy, P refers to total potential energy and F is the generalized force vector of the system. When the generalized coordinates are defined by:

$$q = [\, X^I \; Y^I \; \theta \; \varphi_R \; \varphi_L]^T \tag{2.27}$$

The total kinetic energy is the summation of the kinetic energy of robot itself, left wheel and right wheel and in the following equations they are given.

$$\begin{cases} T_{Robot} = \frac{1}{2} m_R {v_{Robot}}^2 + \frac{1}{2} I_R \dot{\theta}^2 \\ T_{wL} = \frac{1}{2} m_w {V_L}^2 + \frac{1}{2} I_M \dot{\theta}^2 + \frac{1}{2} I_W {\dot{\varphi}_L}^2 \\ T_{wR} = \frac{1}{2} m_w {V_R}^2 + \frac{1}{2} I_M \dot{\theta}^2 + \frac{1}{2} I_W {\dot{\varphi}_R}^2 \end{cases} \tag{2.28}$$

m_R is the mass of the mobile robot platform without electrical actuators and driving wheels, m_W is the mass of each driving wheel with the electrical actuators, I_R is the inertial moment of the mobile robot according to center of mass, I_W is the inertial moment of the wheels with connected the DC motor along the wheel axis, I_M stands for the inertial moment of the driving wheel along the wheel diameter.

Using the kinematic relations that has expressed by equations (2.9), (2.13), and (2.17) and manipulating them, it is possible to reach below equation which returns the total kinetic energy.

$$T = \frac{1}{2}(m_R + m_w)(\dot{X^I}^2 + \dot{Y^I}^2) - m_R d\dot{\theta}\left(\dot{Y}^I cos\theta - \dot{X}^I sin\theta\right) + \frac{1}{2}I_W\left(\dot{\varphi_R}^2 + \dot{\varphi_L}^2\right) + \frac{1}{2}(I_R + m_R d^2 + 2m_w L^2 + 2I_M)\dot{\theta}^2 \tag{2.29}$$

Whereas the potential energy of the system is zero, the coefficients of the related kinematic constraints (C_1, C_2, C_3, C_4, C_5) becomes:

$$\begin{cases} C_1 = (m_R + m_w)\ddot{X}^I - (m_R + m_w)d\dot{\theta}^2 sin\theta - (m_R + m_w)d\dot{\theta}^2 cos\theta \\ C_2 = (m_R + m_w)\ddot{Y}^I - (m_R + m_w)d\dot{\theta}^2 cos\theta - (m_R + m_w)d\dot{\theta}^2 sin\theta \\ C_3 = (I_R + m_R d^2 + 2m_w L^2 + 2I_M)\ddot{\theta} - (m_R + m_w)\ddot{X}^I sin\theta + (m_R + m_w)\ddot{Y}^I cos\theta \\ C_4 = I_W \ddot{\varphi}_R - \tau_R \\ C_5 = I_W \ddot{\varphi}_L - \tau_L \end{cases} \tag{2.30}$$

τ_R refers to the torque input value of the right wheel and τ_L is the same for the left one. Thus, the kinematics constraints matrix becomes:

$$\Lambda^T(q) = [\, C_1\ C_2\ C_3\ C_4\ C_5]^T \tag{2.31}$$

As a result, it is possible to build the sub-elements of equation (2.21) by the followings by accepting m_T the total mass of the robot which is also equals to the sum of m_R and m_W:

$$\begin{cases} M(q) = \begin{bmatrix} m_T & 0 & -m_T d sin\theta & 0 & 0 \\ 0 & m_T & m_T d cos\theta & 0 & 0 \\ -m_T d sin\theta & m_T d cos\theta & (I_R + m_R d^2 + 2m_w L^2 + 2I_M) & 0 & 0 \\ 0 & 0 & 0 & 2I_M & 0 \\ 0 & 0 & 0 & 0 & I_W \end{bmatrix} \\ V(q,\dot{q}) = \begin{bmatrix} 0 & -m_T d\dot{\theta} cos\theta & 0 & 0 & 0 \\ 0 & -m_T d\dot{\theta} sin\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ B(q) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \Lambda^T(q)\lambda = \begin{bmatrix} -sin\theta & cos\theta & cos\theta \\ cos\theta & sin\theta & cos\theta \\ 0 & L & -L \\ 0 & -R & 0 \\ 0 & 0 & -R \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{bmatrix} \end{cases} \tag{2.32}$$

By the help of forward kinematic model which is given by equation (2.11), it is possible to express the generalized coordinate velocities by following equation.

$$\begin{bmatrix} \dot{x}^I \\ \dot{y}^I \\ \dot{\theta}^I \\ \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} = \begin{bmatrix} \frac{Rcos\theta}{2} & \frac{Rcos\theta}{2} \\ \frac{Rcos\theta}{2} & \frac{Rcos\theta}{2} \\ \frac{R}{2L} & -\frac{R}{2L} \\ 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} \tag{2.33}$$

Equation (2.33) is also the written form of following equation where η is the reduced vector:

$$\dot{q} = S(q)\eta \tag{2.34}$$

The derivative of equation (2.34) with respect to the time provides:

$$\ddot{q} = \dot{S}(q)\eta + S(q)\dot{\eta} \tag{2.35}$$

Substituting equations (2.34) and (2.35) under equation (2.36) gives:

$$M(q)[\dot{S}(q)\eta + S(q)\dot{\eta}] + V(q,\dot{q})S(q)\eta = B(q)\tau - \lambda^T(q)^{\lambda} \tag{2.36}$$

It can be proved that the $S(q)$ matrix has a relation with $\lambda^T(q)$ matrix:

$$S^T(q)\lambda^T(q) = 0 \tag{2.37}$$

Therefore, multiplying each side on equation (2.36) by S^T matrix finally brings:

$$\begin{aligned} S^T(q)M(q)S(q)\dot{\eta} + S^T(q)[M(q)\dot{S}(q) + V(q,\dot{q})S(q)]\eta \\ = S^T(q)B(q)\tau - S^T(q)\lambda^T(q)^{\lambda} \end{aligned} \tag{2.38}$$

Because of the equation (2.37), the last term of equation (2.38) is identically zero. Thus, the new main matrices become:

$$\begin{cases} \overline{M}(q) = S^T(q)M(q)S(q) \\ \overline{V}(q,\dot{q}) = S^T(q)[M(q)\dot{S}(q) + V(q,\dot{q})S(q)] \\ \overline{B}(q) = S^T(q)B(q) \end{cases} \tag{2.39}$$

As a result, the reduced form of the dynamic equations finally represented by:

$$\overline{M}(q)\dot{\eta} + \overline{V}(q,\dot{q})\eta = \overline{B}(q)\tau \tag{2.40}$$

Where;

$$I_T = \left(I_R + m_R d^2 + 2m_w L^2 + 2I_M\right) \tag{2.41}$$

The new main matrices finally become:

$$\begin{cases} \overline{M}(q) = \begin{bmatrix} I_W + \dfrac{R^2}{4L^2}(m_T L^2 + I_T) & \dfrac{R^2}{4L^2}(m_T L^2 + I_T) \\ \dfrac{R^2}{4L^2}(m_T L^2 - I_T) & I_W + \dfrac{R^2}{4L^2}(m_T L^2 + I_T) \end{bmatrix} \\ \overline{V}(q,\dot{q}) = \begin{bmatrix} 0 & \dfrac{R^2}{2L} m_R d\dot{\theta} \\ -\dfrac{R^2}{2L} m_R d\dot{\theta} & 0 \end{bmatrix} \\ \overline{B}(q) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{cases} \tag{2.42}$$

By using the kinematic model equations (2.9) and (2.10) it is possible the express the torque values in terms of angular and linear velocities of the robot by the following equations.

$$\begin{cases} (m_T + \dfrac{2I_W}{R^2})\dot{v}_{robot} - m_R d\omega_{robot}{}^2 = \dfrac{1}{R}(\tau_R + \tau_L) \\ \left(I_T + \dfrac{2L^2 I_W}{R^2}\right)\dot{\omega}_{robot} + m_R d\omega_{robot} v_{robot} = \dfrac{1}{R}(\tau_R - \tau_L) \end{cases} \tag{2.43}$$

2.2.6.2 Newton-Euler Approach

The free body diagram of the system is given by Figure 2.2 where F_{uR} and F_{uL} are the longitudinal and F_{wR} and F_{wL} are the lateral forces that are affecting mobile robot from right and left wheels. v_u is the longitudinal, v_w is the lateral velocity of the robot on robot frame and expressed by below equations:

$$\begin{cases} v_u = \dot{r} \\ v_w = r\dot{\theta} \end{cases} \tag{2.44}$$

a_u is the longitudinal acceleration and a_w is the lateral acceleration of the robot on robot frame and expressed by below equations:

$$\begin{cases} a_u = \dot{v}_u - v_w\dot{\theta} \\ a_w = \dot{v}_w - v_u\dot{\theta} \end{cases} \tag{2.45}$$

The only forces that are affecting the robot is its own actuator action on the wheels. Since the robot is accepted as rigid body, using Newton's second law of motion it is earthly to introduce the relationship between forces, torques, and accelerations.

There are two main motions occurs on the system: translation and rotation. Translation is separated into two as lateral and longitudinal. The dynamic equations for translational motion are given by followings.

$$\begin{cases} m_T a_u = F_{uL} + F_{uR} \\ m_T a_w = F_{wL} - F_{wR} \end{cases} \tag{2.46}$$

At the center of mass, the rotation of robot which is around the vertical axis, is given by following equation.

$$J_T \ddot{\theta} = (F_{uR} - F_{uL})L + (F_{wR} - F_{wL})d \tag{2.47}$$

Substituting equation (2.44) and (2.45) under (2.46) and (2.47), it is possible to get following equations:

$$\begin{cases} \dot{v}_u = v_w \dot{\theta} + \dfrac{F_{uL} + F_{uR}}{m_T} \\ \dot{v}_w = -v_u \dot{\theta} + \dfrac{F_{uL} + F_{uR}}{m_T} \\ \ddot{\theta} = \dfrac{L}{J_T}(F_{uR} - F_{uL}) + \dfrac{d}{J_T}(F_{wR} - F_{wL}) \end{cases} \tag{2.48}$$

While the kinematic model of the mobile robot has been developed in section 2.1.2, it is accepted that there is no sliding in lateral direction and a pure rolling in longitudinal direction. Consequently, the lateral velocity of point A in local frame can be accepted as below equation instead of Equation (2.44).

$$v_w = d\dot{\theta} \tag{2.49}$$

Then, substituting equation (2.49) into equations of (2.48) and manipulating them, it is possible to get the below equations that gives the relationships of the affecting torques with the linear and angular speed of the mobile robot.

$$\begin{cases} (m_T \dot{v}_u \dot{\theta} - \ m_R d \dot{\theta}^2 \ = \dfrac{1}{R}\ (\tau_R + \tau_L) \\ (m_T d^2 + J_T)\ddot{\theta} - \ m_T d v_u \dot{\theta} \ = \dfrac{L}{R}\ (\tau_R - \tau_L) \end{cases} \tag{2.50}$$

Comparing equation (2.43) which comes from Lagrangian dynamics approach with equation (2.50) which comes from Newton-Euler dynamics approach, it can be seen that the resulting dynamic model equations are pretty similar.

2.2.7 Transfer Function of the Open Loop Mobile Robot Model

A speed-based control for each wheel is developed in this thesis. To accomplish this purpose a Salem proposed an open loop transfer function separately for each wheel involving the dynamics of the DC motor, and the dynamics of the system [27]. This open loop system accepts the controller system voltage input as the input and angular wheel speed as the output. The open loop transfer function is given by following equation.

$$\begin{cases} Z = (r^2 M L_a + 2b_T R_a + r^2 M R_a + c_r L_a + 2J_T L_a) \\ G(s) = \dfrac{2K_{tac}K_t}{2b_T L_a s^2 + Zs + 2K_T K_t + c_r R_a + J_t R_a} \end{cases} \tag{2.51}$$

According to the proposed model of Salem's study, the robot model is developed in this study. In Section 2.3 the developed model will be widely introduced.

2.2.8 Controller Selection and Design

Miscellaneous controlling methodologies for mobile robots have been introduced so far. Bashir introduced a mobile robot model and control methodology using deadbeat response [31]. Asensio introduced a speed-based DC motor control for this case [32].

Since the main focus of this study is on benchmarking different RL algorithms, for the low-level control it is decided to go with Asensio's methodology. Therefore, a speed-based DC motor control by two different PID controllers for each DC motor on the right and left wheel has been introduced in this study. The open loop transfer function of the controller is given by following equation.

$$G_{PID}(s) = K_P + K_I \frac{1}{s} + K_D \frac{N}{1 + N\frac{1}{s}} \tag{2.52}$$

N stands for the filter coefficient, K_P stands for proportional, K_I stands for integral and K_D stands for the derivative part of the controller. Parameters of the PID controller are tuned by MATLAB's auto tuner algorithm which is by default choosing the crossover frequency based on the plant dynamics and designs for a target phase margin.

Accordingly changing the response time, bandwidth, transient response or phase margin using the interface, the algorithm computes the new gains. The gain values of this PID controller is given in Appendix A. The tuned step response of the system is given by Figure 2.11. The performance parameters for the tuned system are given as: rise time 0.00184, settling time 0.0376, overshoot 14.5%.

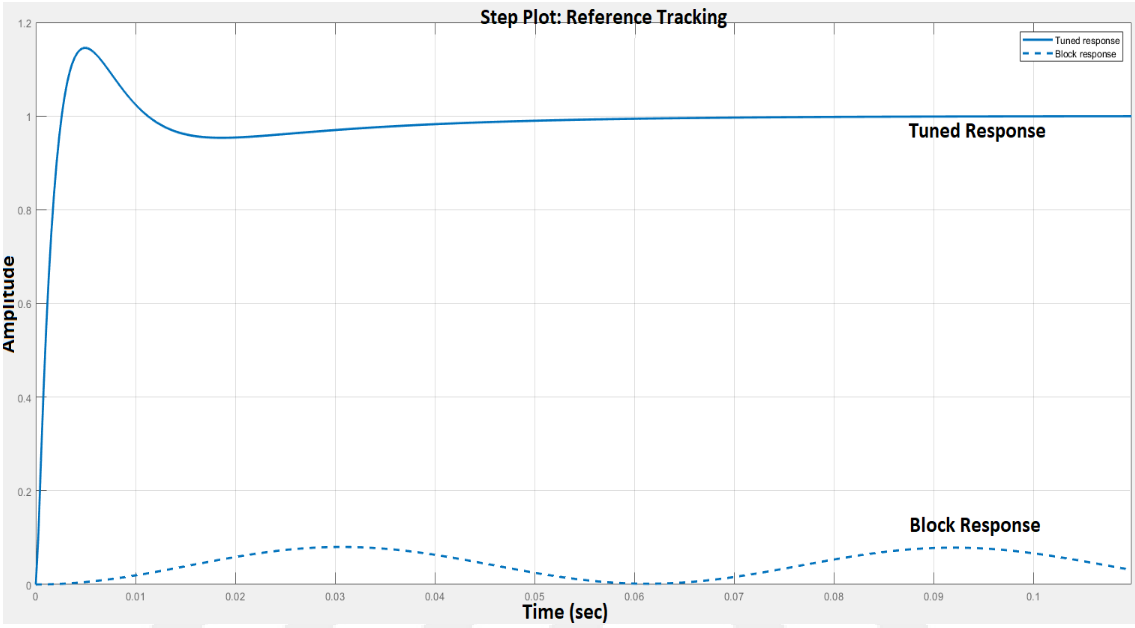


Figure 2. 11 Tuned Step Response of the System

According to the set-point value for heading angle, medium-level controller creates the linear speed setpoint values for the DC motor drivers with respect to the orientation feedback that is presented in Section 2.2.5.3. On the high-level side of control, RL algorithms which use eight proximity sensors as multiple feedbacks, determine heading angle for the next motion of the mobile robot. This operation and the simulation results are explained deeply in Chapter 4.

2.3 Simulink Model of Mobile Robot

The model that is shown in this section involves the low-level controller and its interface between mobile robot dynamics and kinematics. The modeled parts of the system under this model are DC motors, PID controllers, dynamics, and kinematics of the robot platform and its parameters that are given by Appendix A.

.

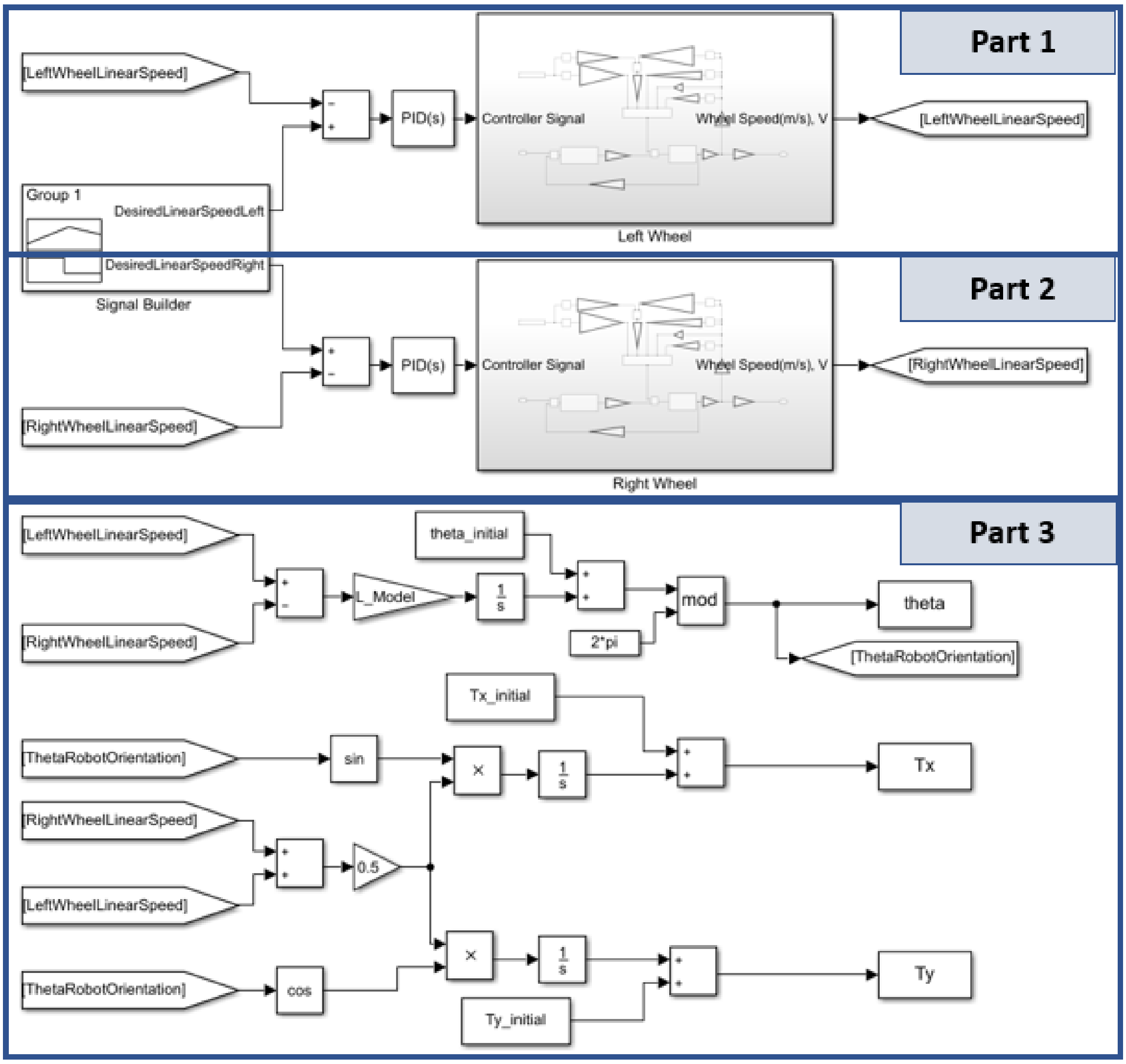


Figure 2. 12 Simulink Model of Mobile Robot

Simulink model consists of three parts as it is also given by Figure 2.12. Part 1 and Part 2 are structurally identical. Part 1 represents the DC motor and the controller of left wheel including robot dynamics. Part 2 represents the same for right wheel. The source of the linear speed inputs for each wheel are different and for the initial testing purposes both have customized signals that comes from signal builder block. Part 1 outputs the linear speed of left wheel and Part 2 does it for right wheel. These linear speed outputs are both on the robot frame. Part 3 converts the robot frame velocity values to the inertial frame and outputs the inertial orientation of the robot, translational position on x and y axis. Initially from the right and left wheel velocities the orientation of the robot on inertial frame is calculated through Equation (2.10).

Then, using the orientation information with the velocities it is possible to calculate the translational displacement by Equation (2.11).

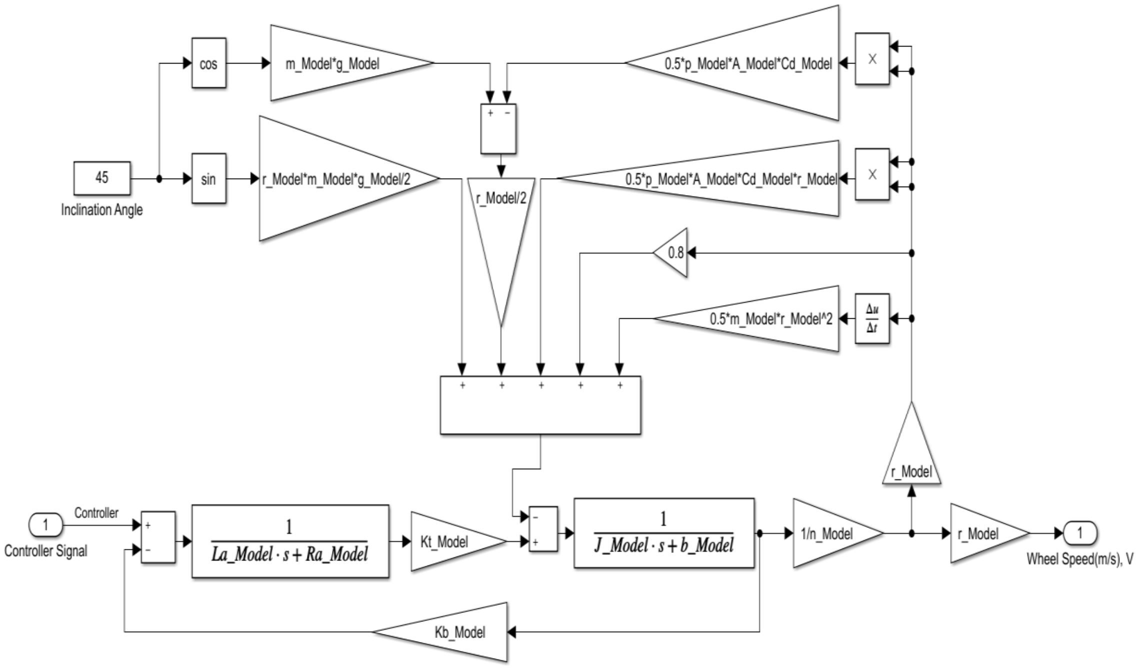


Figure 2. 13 Accurate Simulink Model of Mobile Robot Platform for Each Wheel

Figure 2.13 is the closer look of the models which are introduced both under Part 1 and 2. Considering all dynamics that also include electric actuator and feedback speed sensor are previously represented by Equation (2.51), are also shown here.

2.3.1 Testing and Results

In order to validate and test the developed mobile robot model and its controller functionality, two different customized signals have been generated through Signal Builder block and fed these signals as desired linear speed for left and right wheel. The signals are given by Figure 2.14.

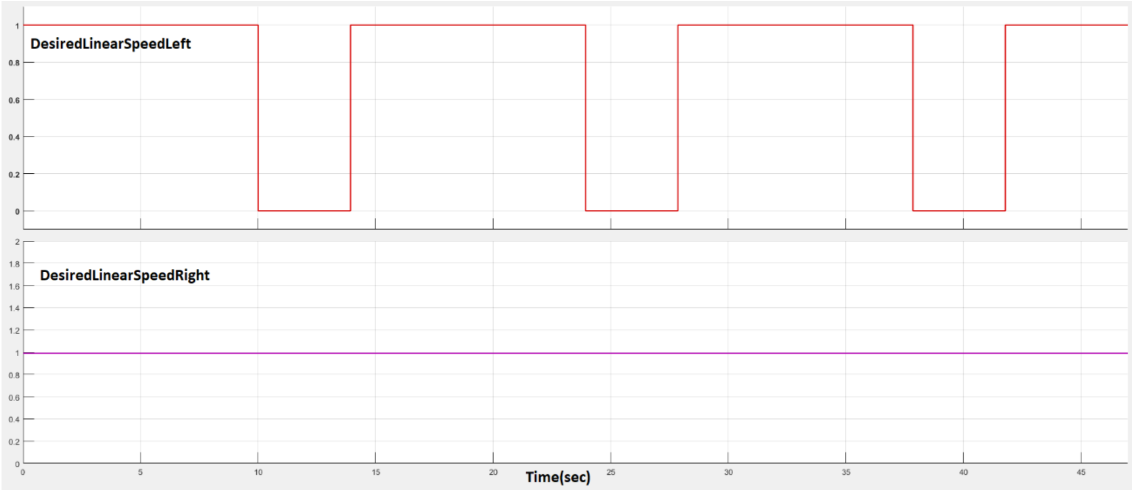


Figure 2. 14 Desired Linear Speed Signals to be Fed into Simulation

From the Equation (2.11), it is easy to determine the time for changing the orientation of the mobile robot for 90 degrees while the desired linear speed of left wheel is 0 and the right one is 1 with respect to the constraints of pure rolling, no slipping, and the robot model parameters in Appendix A.

While building these signals, it is intended to have a rectangular shape in terms of mobile robot trace on inertial frame. Thus, the mobile robot trace by the given signals, can be seen in Figure 2.15.

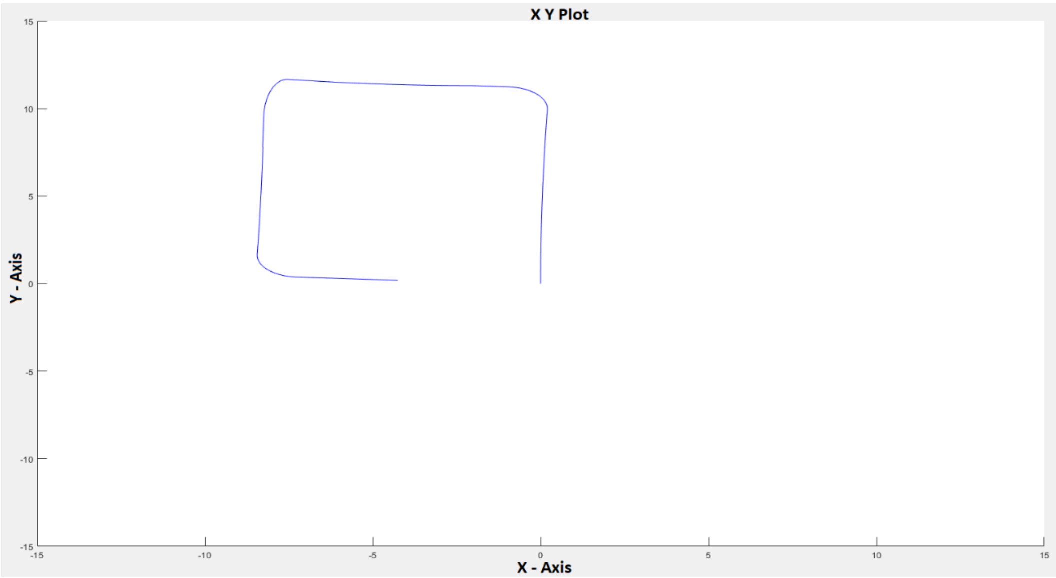


Figure 2. 15 Mobile Robot Trace during Model Validation Test

As a result, we currently have a valid mobile robot model that consists of kinematics, dynamics, tachometer, and the orientation sensor models and is driven by a well calibrated PID controller. As also introduced by Section 1.4, the model of internal state sensors which are the tachometer and the orientation sensor for this study, are developed by Simulink model which is introduced so far. As an external state sensor of the system: the proximity sensors are the interface of the mobile robot with the environment. Thus, they are combined with the model which is introduced so far after the linearization process.

2.4 Linearized Model of Mobile Robot

A solid and valid mobile robot model is developed in Simulink environment so far. This dissertation is not only focused on developing a valid model. The main focus is to simulate and benchmark different RL algorithms on a mobile robot model which is able to interact with simulated environment. RL algorithms consist of various motion steps to train itself and get the knowledge of the environment according to the size of the territory that is desired to be explored. RL controlled system simulations involve lots of episodes which are consisted of many steps. Even for MATLAB which has powerful solvers, it takes time to simulate this kind of maze solving problems with fully accurate mobile robot models.

For the purpose of speeding up the simulations to have a benchmark data between different reinforcement algorithms with same model and simulation environment, it is necessary to shorten the calculation size of each motion step. In order to accomplish this, it is necessary to have a linearization of the system. At this point, MATLAB's Linearization Toolbox helped us out. Figure 2.16 shows the success of this linearization process.

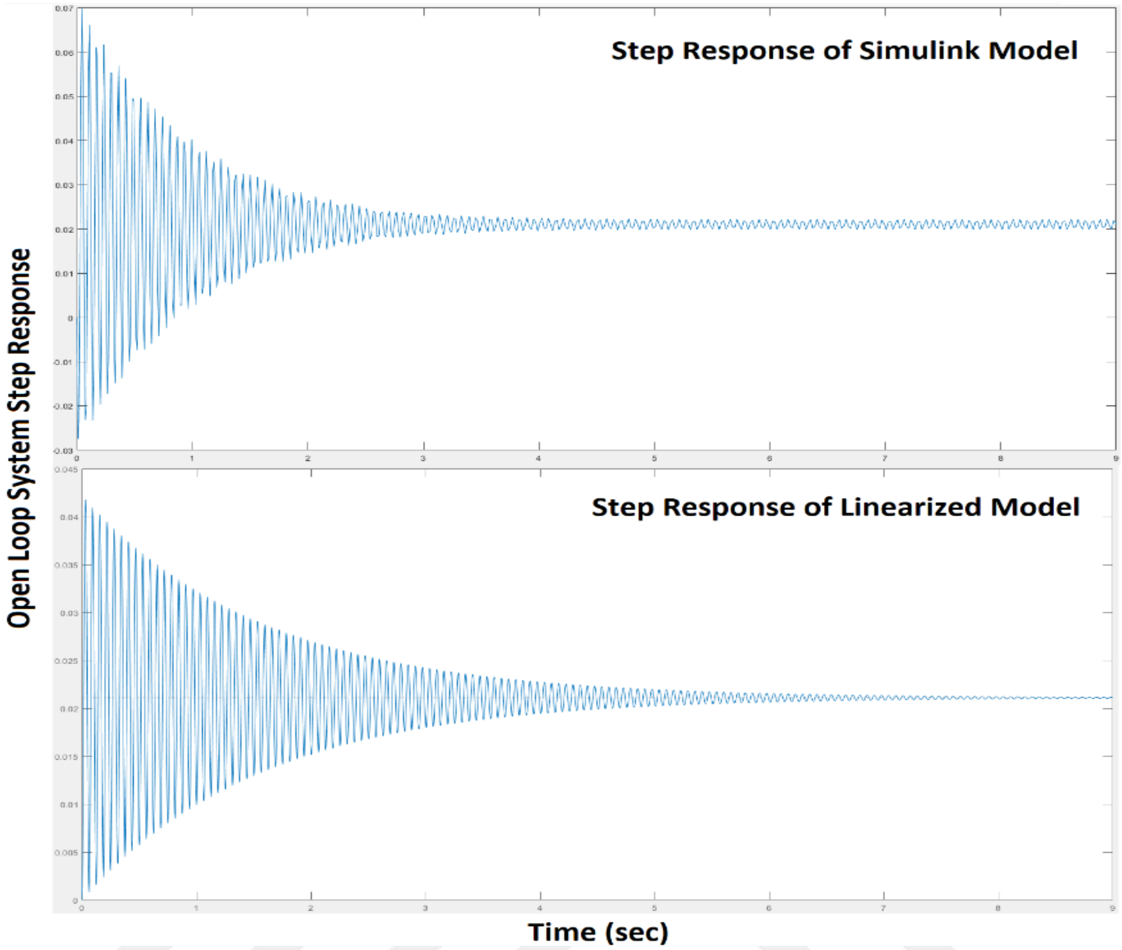


Figure 2. 16 Step Responses of Open Loop Models

The identical open loop models of right and left wheel are given by Figure 2.13 before. By this process the model in Figure 2.13 which takes controller signal as input and actual linear speed as output, is linearized. When you compare the step response of original model, and the linearized one by Figure 2.16, it is seen that there is a minor difference which is not a big deal for our study. Thus, the model is expressed in terms of state space representation which is given in Appendix D.

2.4.1 Testing and Results

In this section, for testing and validating purposes, the linearized model is placed in the two-dimensional simulation environment which is surrounded by four walls. Mobile robot model is driven as step by step through this study and is represented by the green object in a bird's-eye view which is given by Figure 2.17. So, for the testing purpose, it moves for 34 separated motions which are also observable by the given figure below.

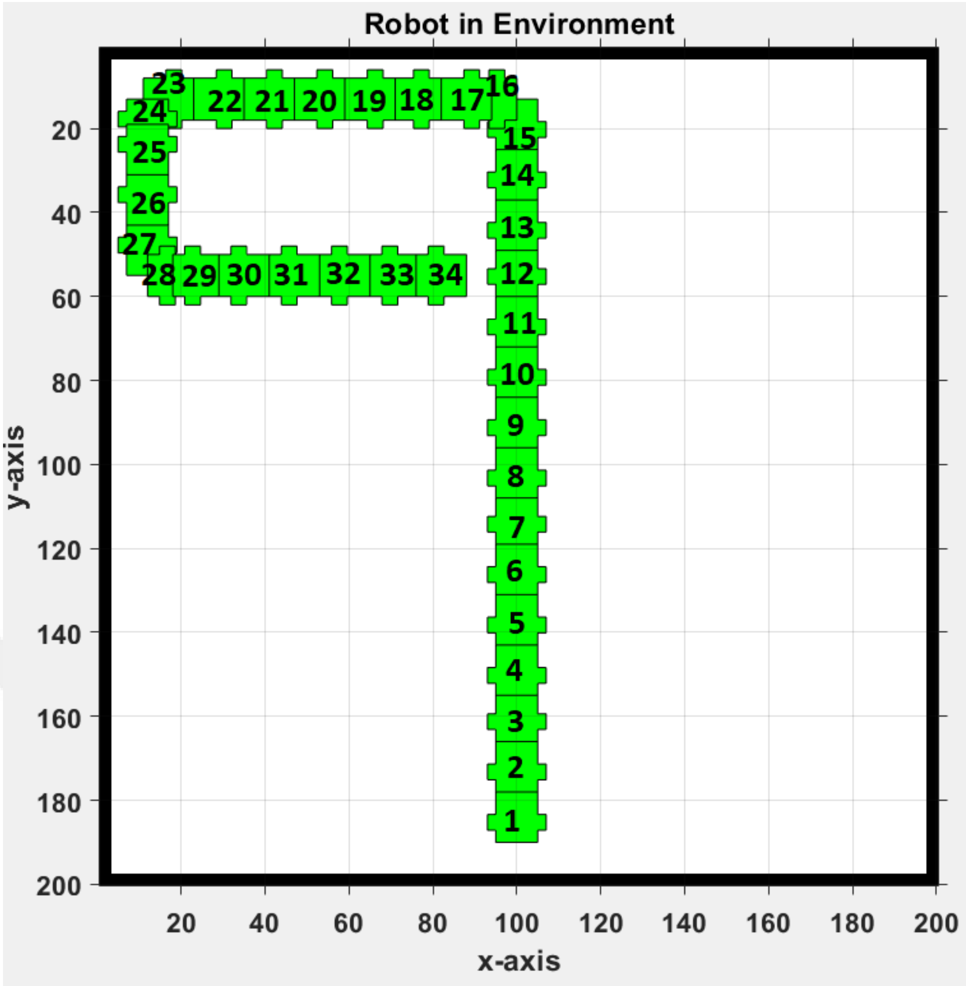


Figure 2. 17 Linearized Robot Model in Environment for Test Drive

During the test drive, robot heads upwards for 15 motions, then rotates counter clockwise for 90 degrees and continues in that direction for 7 more motions until one more counter clockwise rotation for 90 degrees, then continues for 4 more motions until the last counter clockwise rotation for 90 degrees and finally it heads to right side for 6 more motions.

During the test drive it is possible to collect the orientation data to validate the success of the simulation. So, the orientation of the robot during this test drive is given by the Figure 2.18.

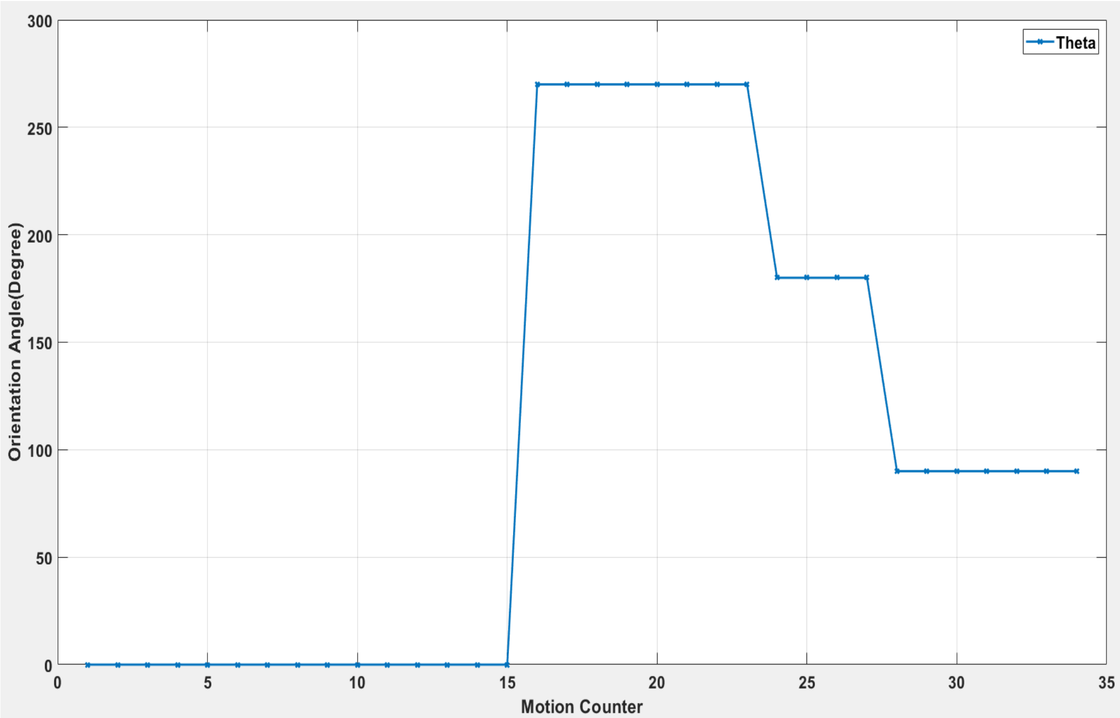


Figure 2. 18 Orientation of the Robot During Test Drive

2.5 Interaction of the Mobile Robot Model with Environment

Types of sensors has been introduced before in Section 1.4 and classified under two categories as internal state and external state. External state sensors are the ones which make it possible for the system to interact with environment. In this study proximity sensors are being used for this aim. The proximity sensors and their location on the robot platform has been introduced in Section 2.2.5.1 and as a reminder the datasheet of the sensor is given by Appendix B.

For the purpose of increasing the performance of the simulation, the sensor data is post-processed as all-or-nothing sensor which gives Boolean output. If any object in its range is detected, the sensor provides a Boolean '1' and if not it provides a Boolean '0'. In Section 2.4.1 a test drive has been performed with the mobile robot. During the test drive not only the orientation data but also the data of proximity sensors are collected. The data which is measured during the test drive is given by Figure 2.19.

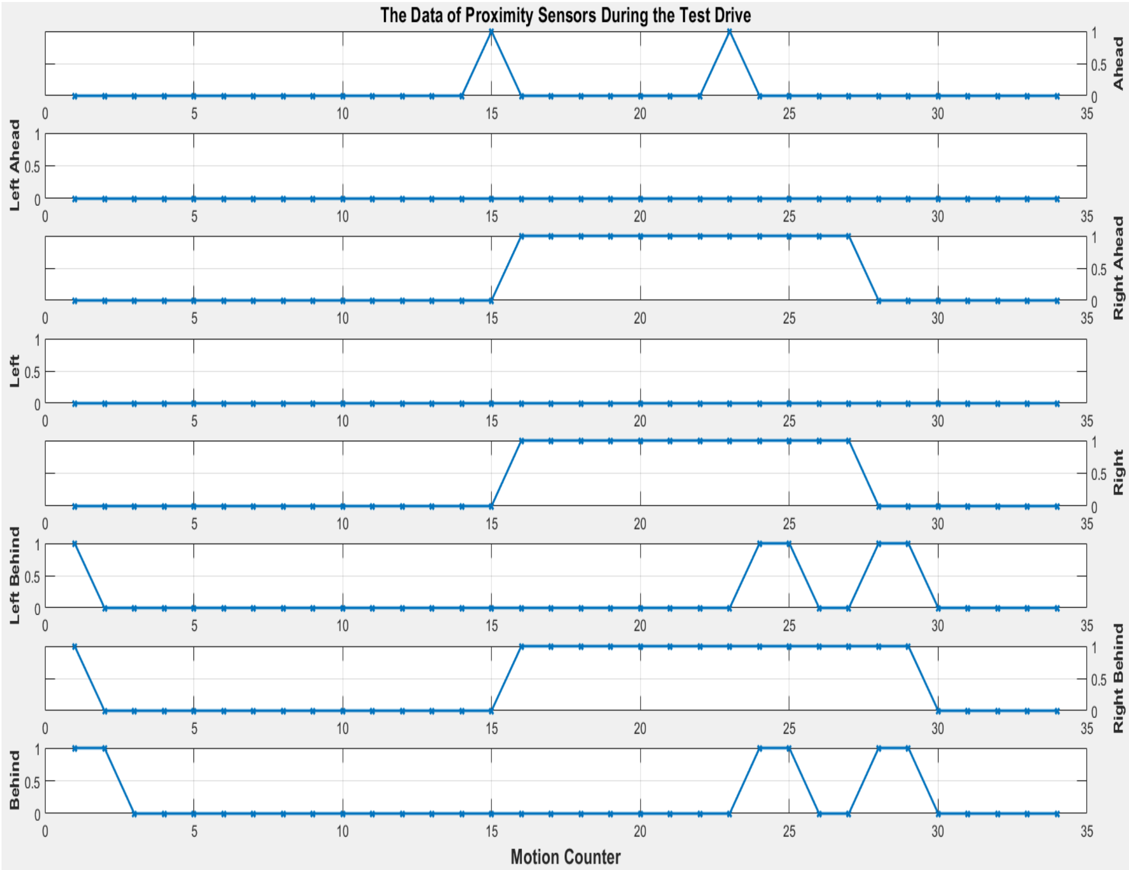


Figure 2. 19 The Data of Proximity Sensors During Test Drive

As a brief summary, for the first two motions behind sensor sees an object, at the 15th and the 23rd motions ahead sensor sees object, all the right sensors are seeing objects from 15th to 27th motions, and left sensor has no trigger during the test drive since it does not face any object through. These data are consistent if you consider the test drive which is given by Figure 2.17 before.

2.6 Conclusion

In this chapter, a mobile robot model has been developed for the simulation purpose of this dissertation. Initially a mathematical model which consists of the electrical actuators, proximity sensors, orientation sensor, tachometer, robot kinematics and dynamics, has been introduced. Later, for the aim of speeding up the simulations to have a benchmark data between different reinforcement algorithms, the Simulink model has been linearized and so simplified. By the test drive the robot's interaction with the environment is validated. The model that has been successfully developed, is going to be used during the simulations of next chapters.

3
Implementation

3.1 Introduction to Reinforcement Learning

RL is being operated in a setting of iterations where the agent is the decision-making learner that takes actions for the purpose of interacting within its surrounding. This interaction results by a penalty or reward. The typical operation mechanism of RL is shown in Figure 3.1. The agent interacts with its surrounding and receives reward as feedback and calculates the action which results as a transition into the new state. The iterative interaction of the agent with the environment is shown in the current time: *t*. This whole process repeats itself in each action step until the target is reached.

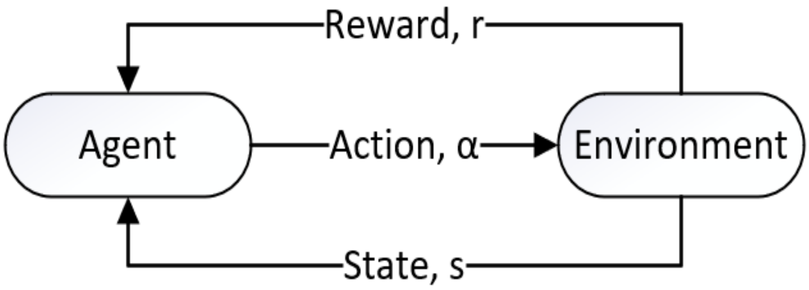


Figure 3. 1 RL Typical Mechanism

Rather than just defining agent and the environment, it is necessary to represent some additional components into this process for explaining the reinforcement algorithms. These are transition model, reward, value, and policy functions[2].

Transition model or transition probability function is a prediction how the environment will respond to the actions of the agent by specifying the possibilities of the outcomes for each action with its each corresponding state. For a standing state, the model has more than one next state and next reward due to the different probabilities of next possible actions. $T(s, a, s')$ refers to the transition model where in the current state: $(s||S_t) \in S$ by choosing action: $(a||A_t) \in A$, denotes the probability state: $(s'||S_{t+1}) \in S$. S is the set of states and A stands for the set of actions.

Reward function is a bounded absolute value between each state-action pair with the single number called reward: $r(s,a) \in R$. R stands for the set of rewards. Rewards show the agent what to achieve not how to achieve. Thus, the rewards are computed as a part of environment for precluding the agent to affect the rewards. The agent is only rewarded if it reaches to the final aim. Defining sub-goals might let agent to ignore the main goal thus they are not recommended having.

Value function determines what is beneficial for the agent in long term while reward function is focused on short term benefit. The term of 'how beneficial' is described in terms of future rewards that can be expected or to be certain in expectations. The value of a state is the total reward of expected state sequences, so it depends directly on the policy. For example, a low-rewarded state can have a high value since it is pursued by states that have high rewards. As a summary it can be said that while making decisions and choosing actions, values are more precise and effective than rewards. Rewards are the outcome of the interaction of the robot with its environment, however values are recalculated after gathering this outcome for each state-action pair. Thus, value iteration method is the most important part of all of RL algorithms. Value function is defined for a given policy as $V^{\pi}: S \rightarrow R$ where π stands for the policy.

Policy function defines how the agent should act according to the recalculated value function from the perceived rewards with respect to the current state-action pair. It can be shortly predicated that the next action is selected by the policy. Complexity of the policy is dependent to the requirements of the system. A deterministic stationary policy is concentrated on a single action of the all states where $s \in S$. A stochastic stationary policy maps each state to the probable different actions. In the deterministic case, taking a given action in a given state always results in the same next state; while in the stochastic case, the next state is a random variable[1]. The closer to optimal policy used by the agent is, the closer to the goal the agent approaches. Optimal policy is represented by π^*.

RL in robotics is characteristically formulated as Markov Decision Process (MDP). The structure of MDP will be introduced in following section.

3.1.1 Markov Decision Process

Putterman defines MDP as, it is the base algorithm for sequential decision making problems which an agent chooses the sequence of actions that maximizes some reward based optimization criterion [33]. Formally MDP is a tuple *where* $M= \{S, A, T, r, \gamma\}$. γ is the discount factor and defined as the effect of future rewards over value of current state and valued between *[0, 1)*. When discount factor is zero, the agent concerns to maximize only the next reward not future rewards, and the system becomes episodic not sequential and when it approaches to one, then future rewards get more important to decide next action for the agent. Figure 1.4 shows the application of a policy. Initially the starting state S_0 is defined. Then, the policy π defines the action $A_0 = \pi\ S_0$ to be performed. Based on the current transition probability function which is defined by $T\ (S_0 A_0 S_1)$, a reward function $R_1 = r\ (S_0, A_0, S_1)$ is received by action A_0. Transition probability function involves the distribution of the different possible actions for each state and maps each state with each possible action.

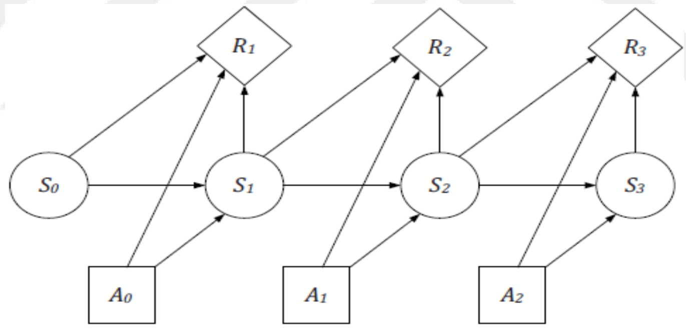


Figure 3. 2 Decision Network of finite MDP [1]

The generalized form of this process is given by below equation where V is the value function and V^{π} is the value function of an state-action pair for a given policy.

$$V\pi(s) = E\pi\, [\textstyle\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s]\ , \forall s \in S \qquad (3.1)$$

E_π is the expected value for policy and discount factor. Similarly, the action-value function $Q^{\pi}: S \times A \rightarrow R$ which is named as Q-value is defined by following equation.

$$\mathrm{Q}\,\pi(\mathrm{s}) = \mathrm{E}\pi\, [\textstyle\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s, A_0 = a] \qquad (3.2)$$

Finally, the advantage function which is corresponded with the policy function is defined by following equation.

$$A^{\pi}(s) = Q^{\pi}(s,a) - V^{\pi}(s) \qquad (3.3)$$

The optimal action-value function Q^* and optimal value function V^* are identified by following equation which is also satisfying the Bellman optimality equations [33].

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \epsilon S} P(s'|s,a)V(s') \qquad (3.4)$$

3.1.2 Partially Observable MDP

For the real use cases unlike MDP, it is not always possible for agent to fully observe the environment. The assumption of MDP might not be practical, insufficient or unnecessary for some cases. For these cases Partially Observable Markov Decision Processes (POMDP) was introduced [34]. POMDP is a tuple $\{S, A, T, R, O, Z\}$ where O is a set of observations and Z is an observation function. The probability of observing (belief state) $Z(o, s, a)$ of the current state-action pair where $o \epsilon O$ is the current observation. POMDP uses belief state as probability distribution all over the possible future states to detect according to the current state that agent is likely to be in. The better the agent knows where it is the more successfully it determines its future actions.

Not only the same observation can be done in different states but also different observations can be done in same state. Therefore, the state of the system cannot be identified from the observations, it only can be predicted. The agent's belief in this prediction is given by the following equation:

$$b'(s') = aO(s',o) \sum_{s \epsilon S} T(s,a,s')b(s) \qquad (3.5)$$

The decision cycle is that depending on the belief state, the agent decides the action a = π(b), and passes to the next state s′, then new belief state b′ is calculated. Russel and Norvig introduced solving the POMDP problem on a real case can be reduced to MDP problem on a corresponding belief state space by using $\tau(b, a, b')$ and $\rho(b,a)$ instead of T(s, a, s′) and R(s,a) [20].

$$\begin{cases} \tau(b,a,b') = \sum_{o \in O} T(b'|b,a,o)T(o|a,b) \\ r(b,a) = \sum_{x \in S} b(s)R(s,a) \end{cases} \tag{3.6}$$

τ is the belief state transition function. If $T(b'|o,a,b)$ equals to 1, b' is equal to the next belief state of the agent after POMDP decision cycle and vice versa if it is zero. The schema of POMDP is given by Figure 1.5.

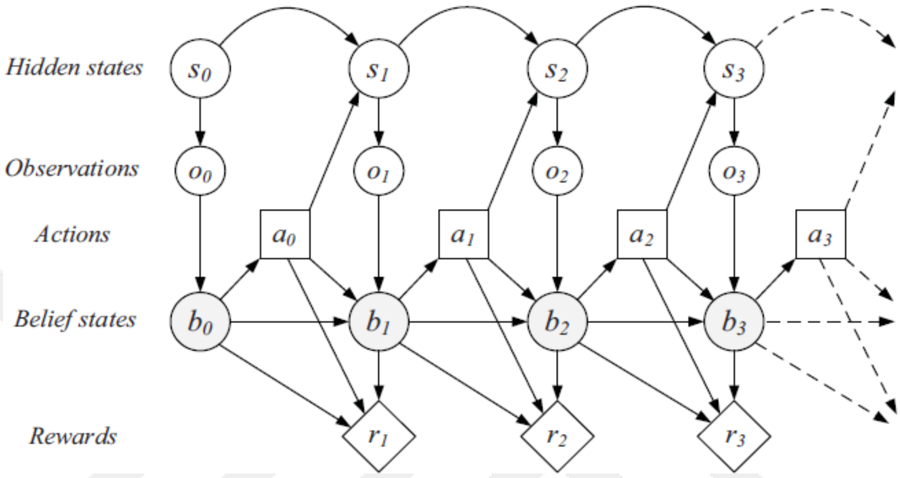
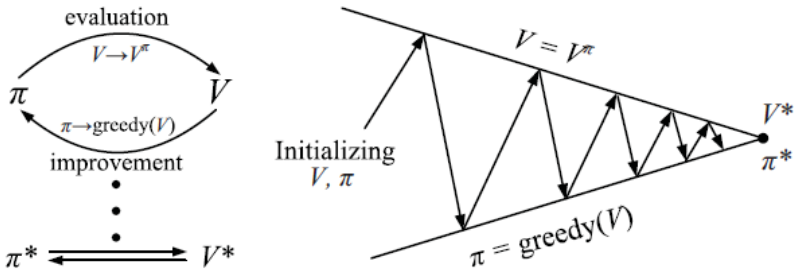


Figure 3. 3 Decision Network of POMDP[1]

3.1.3 Model-Based Algorithms

Dynamic programming(DP) is a solution method for computing an optimal policy for solving a given MDP. DP assumes the full knowledge of MDP therefore it is categorized under model-based learning algorithms. In DP estimation update of value for a state depends on value estimate of its successor states. This is called bootstrapping. Model based algorithms requires the dynamics of the environment, and the reward function. Sutton and Barto indicates that model based algorithms has the generalized policy iteration(GPI) principle [35]. Which is given by Figure 1.6. Policy evaluation is the initial step for this principle. This consists of the estimation of the utility of current policy π and computing the value V^{π}. The second and the last step is collecting the information from the first step and computing the step named policy improvement. The values of the actions are evaluated for every state, for finding the possible improvements. This step computes an improved policy π' from the current policy π using the information in V^{π}. The main aim is to converge to the optimal value function and an optimal policy and the geometric metaphor for convergence is given by Figure 1.6. The weakness of DP is that it requires a fully defined environment.



(a) Interaction of policy evaluation and improvement processes

(b) The convergence of both the value function and the policy to their optimals

Figure 3. 4 Generalized Policy Iteration [35]

Model-based algorithms can be categorized as value iteration (VI) and policy iteration (PI). PI iterates between the two processes of GPI and repeated until converging to an optimal policy. Pseudo code of VI based DP is given in Figure 3.5.

/*Initialization*/

- *t=0, k=0;*
- *∀s ∈S: Initialize $\pi_t(s)$ with an arbitrary action;*
- *∀s ∈S: Initialize $V_k(s)$ with an arbitrary value;*
- *Repeat*

  /*Policy Evaluation*/

  1. *Repeat*
     - *∀s ∈S: $V_{k+1}(s) = r(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s')V_k(s')$;*
     - *$k \longleftarrow k+1$;*
  2. *until: ∀s ∈S: $|V_k(s) - V_k(s)| < \epsilon$;*

  /*Policy Improvement*/

  3. *∀s ∈S: $\pi_{t+1}(s) = \arg\max_{a \in A}[r(s, a) + \gamma \sum_{s' \in S} T(s, a, s')\, V_k(s')]$;*
  4. *$t \longleftarrow t+1$;*
- *until: $\pi_t = \pi_{t-1}$;*
- *$\pi^* = \pi_t$;*

/*Output: An optimal policy */

Figure 3. 5 Pseudo Code of PI based DP [25]

VI consists of overlapping the evaluation and improvement of processes. Instead of completely separating the evaluation and improvement processes, the value iteration approach breaks off evaluation after just one iteration. In fact, it immediately blends the policy improvement step into its iterations, thereby purely focusing on estimating directly the value function[1]. The pseudo code of PI based DP is given in Figure 3.6.

/*Initialization*/

- *k=0;*
- *$\forall s \in S$: Initialize $V_k(s)$ with an arbitrary value;*
- *Repeat*
  1. *$\forall s \in S$: $V_{k+1}(s) = max_{a \in A}[\, r(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s')V_k(s')]$;*
  2. *$k \longleftarrow k+1$;*
- *until: $\forall s \in S$: $|V_k(s) - V_k(s)| < \epsilon$;*
- *$\forall s \in S$: $\pi^*(s) = arg\ max_{a \in A}[r(s, a) + \gamma \sum_{s' \in S} T(s, a, s')\, V_k(s')]$;*

/*Output: An optimal policy */

Figure 3. 6 Pseudo Code of VI based DP [35]

3.1.4 Model-Free Algorithms

In contradistinction to DP that requires a fully defined model of the environment, RL is initially interested in obtaining an optimal policy when a model is unavailable. Therefore, RL is a model-free algorithm. The agent needs to discover the environment by the action-reward pairs for each state in discrete time steps. The aim of RL is to find the optimal policy π^* that links states and observations from the actions for maximizing the cumulative expected reward. At this point it is necessary to determine if the algorithm to stick with well-known actions with relatively high rewards or randomly select new actions for exploring new strategies with an even higher reward. This is identified as exploration-exploitation trade-off.

RL can be categorized as *off-policy* and *on-policy* according to the agent's action choosing style. While the on-policy agent trains itself by learning the value based on its current action which is derived from current policy, the off-policy agent learns the value based on the action that is obtained from another policy. To put it another way, off-policy algorithms are explorative strategies that are different from the desired final policy can be employed during the learning process. Besides, off-policy learning involves behavior policy and estimation policy. Behavior policy is used to make decision, and it needs to ensure all actions have a probability to be selected to explore all possibilities.

Estimation policy is the policy evaluated and improved, and since it does not affect decisions, this policy can be totally greedy. The exploration and the performance of the policy can be resulted in an exploration-exploitation trade-off. As a result, exploration must be built into the policy and determines the speed of the policy improvements.

3.1.4.1 Monte Carlo Methods

Unlike DP, *Monte Carlo Methods*(MCM) do not behoove any knowledge about the environment, and update value of estimation depending to experience instead of estimation of another state. MCM is a bootstrap-free solution method for model-free algorithms and use sampling for estimating the value function and chasing the optimal policy. Pseudo code of On-Policy MCM is given in Figure 3.7.

/*Initialization*/

- *∀s ∈S, ∀a ∈A: Initialize Q(s,a) with an arbitrary action;*
- *∀s ∈S, ∀a ∈A: Returns(s,a) ⟵Empty Reward List*
- *Π ⟵An Arbitray ϵ-soft policy*
- *<u>Repeat</u>*
    1. *Generate an episode using Π;*
    2. *For each pair (s,a) appearing in the episode;*
        a. *R⟵ return following the first occurrence of (s,a)*
        b. *Append R to Returns(s,a);*
        c. *Q(s,a) ⟵average(Returns(s,a));*
    3. *For each s in the episode;*
        a. *a*⟵arg max$_a$(Q(s,a));*
        b. *For ∀a ∈A(s);*
            i. $\Pi(s,a) \longleftarrow \begin{cases} 1-\epsilon+\frac{\epsilon}{|A(s)|} & if\ a = a* \\ \frac{\epsilon}{|A(s)|} & if\ a \neq a* \end{cases}$

/*Output: An optimal action for the defined policy */

Figure 3.7 Pseudo Code of On-Policy Monte Carlo Method [35]

The assumption at this point is that each episode terminates eventually regardless the selected action and after the termination the estimation is updated not after each step. Based on averaging sample returns that consists of episodes, MCM solves RL problems by explicitly estimating how good each action is for state. The weakness of this method is that it is not successful in solving continuing tasks. Pseudo code of Off-Policy MCM is given in Figure 3.8.

/*Initialization*/

- *$\forall s \in S$, $\forall a \in A$: Initialize $Q(s,a)$ with an arbitrary action;*
- *Numerator of $Q(s,a)$: $N(s,a) \longleftarrow 0$;*
- *Denominator of $Q(s,a)$: $D(s,a) \longleftarrow 0$;*
- *$\Pi \longleftarrow$ An Arbitray Deterministic policy*
- *<u>Repeat</u>*
  1. *Select a policy Π' and use it to generate an episode:*
     a. $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
  2. *$T \longleftarrow$ latest time at which $a_T \neq \Pi(s_T)$;*
  3. For each pair s,a appearing in the episode after T:
     a. *$t \longleftarrow$ the time of first occurrence in episode after T:*
     b. $w \longleftarrow \prod_{k=t+1}^{T-1} \frac{1}{\Pi'(s_k,a_k)}$
     c. $N(s,a) \longleftarrow N(s,a)+wR_t$
     d. $D(s,a) \longleftarrow N(s,a)+w$
  4. *For $\forall s \in S$:*
     a. $\Pi(s) \longleftarrow \arg\max_a(Q(s,a))$;

/*Output: An optimal policy */

Figure 3.8 Pseudo Code of On-Policy Monte Carlo Method[35]

3.1.4.2 Temporal Difference Methods

In brief definition, *Temporal difference Methods*(TDM) are the combination of MCM and DP with their good parts. TDM uses sample backups like MCM and bootstrapping like DP. TDM is different from MCM by not having the necessary to wait until an estimate of the return is available in other words until the end of an episode to update the value function. It updates its values on-line after each step.

By using the difference between the old and new estimate of value function, TDM has only to wait for the end of the step. By only taking into account the sampled successor states rather than the complete distributions over successor states, TDM separates itself from DP. However, it bootstraps like DP by updating value estimate with respect to another value estimate as basis. TDM is also model-free like MCM and does not require the model of the environment's dynamics. Formulation of TDM is given by the following equation that uses the received reward and the value for reaching next state to calculate a temporal difference error which is multiplied by a learning rate before the update is executed.

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{3.7}$$

Learning rate is given by α. It is generally valued from 0 to 1. Zero means the learning value is never changed so nothing is learned. Values closer to 1 means the learning is being occurred quickly. Discount factor is given by γ. It is also valued from 0 to 1. This factor decides the worth of future rewards according to immediate rewards. Setting it closer to 0 will enable the algorithm to converge.

Most popular RL algorithms such as Q-learning, SARSA, Deep Q-Network, Fuzzy Rule Interpolation Based Q-Learning etc. are categorized under TDM. It is necessary to attach particular importance to the main focus of this dissertation: RL. For that purpose next section is reserved for RL agents and training them with its popular TDM algortihms.

3.2 Reinforcement Learning Agents and Training Process

Briefly, it is possible to list down the popular RL algorithms that are introduced in previous section and compare them by Table 1.2 in terms of base model type, experience requirements, bootstrapping requirement, estimation update period, policy and operator being used.

Table 3. 1 Comparison of the Methods of Decision-Making Algorithms

Method/Algorithm		Model	Require Experience	Bootstrap	Estimation Update Period	Policy	Operator
Dynamic Programming		Model- Based	No	Yes	Step-By-Step	-	-
Monte Carlo Methods		Model-Free	Yes	No	Episode-By-Episode	Both	-
Temporal Difference Methods	SARSA	Model-Free	Yes	Yes	Step-By-Step	On-Policy	Q-Value
	Q-learning	Model-Free	Yes	Yes	Step-By-Step	Off-Policy	Q-Value
	DQN	Model-Free	Yes	Yes	Step-By-Step	Off-Policy	Q-Value
	FRIQ	Model-Free	Yes	Yes	Step-By-Step	Off-Policy	Q-Value

RL is a computational approach that learns by its agent's interaction with the unknown environment. As a consequence of the observations from series of actions by maximizing the cumulative reward for next decision, it is possible to train the agent. The policy selects the action based on the observation from the environment, and the learning algorithm constantly updates the policy parameters based on this process. Policy consists of tunable parameters like a neural network. Learning algorithm tries to find the optimal policy for reaching the goal.

3.2.1 Q-Learning

Q-learning is an off-policy TD method which is a model-free RL algorithm. Watkins defines Q-learning as it updates action-value function by greedy method, independent of behavior policy after selecting an action for the current state and receiving a reward from that selection by the next state [36]. The following pseudo code in Figure 3.9 is for the training algorithm.

/*Initialization*/

- *∀s ∈S, ∀a ∈A: Initialize Q(s,a);*
- *Repeat (for each episode):*
  1. *Initialize s;*
  2. *Repeat (for each step of episode):*
     a. *Choose a form s using policy derived from Q;*
     b. *Take action a, observe r, s';*
     c. *Q(s,a) ⟵ Q(s,a) +α [r+ γmaxQ(s',a')- Q(s,a)];*
     d. *s⟵s';*
  3. *until s is terminal*
- *until all episodes end.*

/*Output: An optimal policy */

Figure 3.9 Pseudo Code of Temporal Difference Q-Learning [36]

Q-learning generally selects the maximum Q-valued action but even in a random action selection, it learns the optimal policy Q^* independent of the policy being followed. This simplifies the analysis of algorithm and enables early convergence of the proofs.

$$Q(s_t, at) = Q(s_t, at) + \alpha[r_{t+1} + \gamma maxQ(s_{t+1}, a) - Q(s_t, a_t)] \quad (3.8)$$

The idea behind Q-learning is relied on VI. The update equation is replaced with the above formulation thus, it is unnecessary to take care of the transition probability matrix. The action is determined by the maximization of the next state's Q-value instead of following the current policy. Eventually, Q-learning is categorized under the off-policy algorithms.

During training process, agent tries to explore the environment using epsilon-greedy exploration. At each step, agent decides a random action with the probability ϵ or decides an action greedily by the value function with probability *1-*ϵ which also means the greatest value function.

3.2.2 State Action next State next Action

The most popular on-policy TD method is *State Action next State next Action* learning method which is also known as *SARSA*. Rummery defines SARSA as it operates only one policy for evaluation and decision making by using the Q-value as operator [37]. The following pseudo code in Figure 3.10 is for the training algorithm.

/*Initialization*/

- *∀s ∈S, ∀a ∈A: Initialize Q(s,a);*
- *Repeat (for each episode):*
  1. *Initialize s;*
  2. *Choose a from s using policy derived from Q;*
  3. *Repeat (for each step of episode):*
     a. *Take action a, observe r, s';*
     b. *Choose a' from s' using policy derived from Q;*
     c. $Q(s,a) \leftarrow Q(s,a) + \alpha\,[\,r + \gamma Q(s',a') - Q(s,a)\,];$
     d. $s \leftarrow s';\ a \leftarrow a';$
  4. *until s is terminal*
- *until all episodes end.*

Figure 3. 10 Pseudo Code of Temporal Difference SARSA [37]

The difference between SARSA and Q-learning is that the maximum reward for the next state is not necessarily used for updating the Q-values. Instead of this, a new action and reward are selected using the same policy that is already defined the original action. So, it can be said that SARSA learns the new Q-value based on the action performed by the current policy instead of the greedy policy. The formulation of SARSA algorithm is given by following equation.

$$Q(s_t, at) = Q(s_t, at) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (3.9)$$

3.2.3 Deep Q-Network

The main weakness of Q-learning is its lack of generality. Q-learning updates the numbers in a two-dimensional array of action and state. Q-learning agents do not have any ability to estimate the value for unseen states. In order to deal with this issue, *Deep Q-Network(DQN)* is introduced. DQN has the ability to estimate the value for unseen states by introducing Neural Networks and to get rid of the two-dimensional array. The following pseudo code in Figure 3.11 is for the training algorithm.

/*Initialization*/

- *$\forall s \in S$, $\forall a \in A$;*
- *Initialize action-value function Q(s,a) with random weights θ;*
- *Initialize replay memory D to capacity N;*
- *Initialize target action-value function $\bar{Q}$ with weights $\theta^- = \theta$;*
- ***For** episode=1 , M **do***
  1. *Initialize sequence s_1={ x_1} and preprocessed sequence $\Phi_1 = \Phi$ (s_1);*
  2. ***For** t=1, T **do***
     a. *With probability ϵ select a random action a_t;*
     b. *Otherwise select $a_t = \mathrm{argmax}_a\, Q(\Phi(s_1), a; \theta)$;*
     c. *Execute action a_t in emulator and observe reward r_t and image x_{t+1}*
     d. *Set $s_{t+1} = (s_t, a_t, a_{t+1})$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$;*
     e. *Store transition ($\Phi_t, a_t, r_t, \Phi_{t+1}$) in D;*
     f. *Sample random minibatch of transitions ($\Phi_j, a_j, r_j, \Phi_{j+1}$) from D;*
     g. *Set* $y_j \leftarrow \begin{cases} r_j & \text{if episode terminates at step } j+1; \\ r_j + \gamma \ \max_{a'} \bar{Q}(\Phi_{j+1}, a'; \theta^-) & \text{if otherwise;} \end{cases}$
     h. *Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to θ;*
     i. *Every C steps reset $\bar{Q} = Q$;*
  3. ***End For***
- ***End For***

Figure 3. 11 Pseudo Code of Deep Q-Learning [13]

The input for the network is the current, while the output is the corresponding Q-value for each of the action. Training the network is based on the Q-learning update equation by recalling the target Q-value for Q-learning is given by following equation.

$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)}[(r + \gamma maxQ(s',a';\theta_i^-) - Q(s,a;\theta_i))^2] \qquad (3.10)$$

Parametrization of the approximate value function $Q(s,a;\theta_i)$ using deep convolutional neural network in which θ_i are the weights of the neural network at iteration *i*. For performing experience replay it is necessary to store the experiences of agent in $e_t=(s_t,a_t,r_t,s_{t+1})$ at each time-step *t* in a data set $D_t=(e_1,e_2,\ldots,e_t)$. During the learning process, Q-learning updates are applied on samples of experience $(s,a,r,s')\sim U(D)$, drawn uniformly and randomly from the data set of stored samples. The target network parameter θ_i^-, is only updated with the Q-network parameter θ_i in every step and held fixed between individual updates [13].

3.2.4 Fuzzy Rule Interpolation Based Q-Learning

Fuzzy Rule Interpolation Based Q-Learning(FRIQ) uses fuzzy rules to represent the action-value function with the capability of discrete rule base. FRIQ-learning is different with the Q-learning by making derivation of Q-values using fuzzy inference system to update the Q-function. The following pseudo code in Figure 3.12 is for the training algorithm.

/*Initialization*/

- *$\forall s \in S$, $\forall a \in A$;*
- *s_t is the current state at time t;*
- *Derive Q-Values for each state-action pair (s_t, a_t) using fuzzy inference system;*
- *Choose an action a_t on state s_t;*
- *Carry out action a and transit to the next sate s_{t+1};*
- *Derive Q- Values for each state-action pair(s_{t+1}, a_{t+1}) using fuzzy inference system;*
- *Calcuate the update amount of Q-Value (ΔQ);*
- *Update Q-Function or Fuzzy Rules;*
- *Go back to First Step;*

Figure 3. 12 Pseudo Code of FRIQ-learning [38]

Standard Fuzzy Q-learning requires a complete fuzzy rule-base, for every possible case. For FRIQ learning, it is unnecessary to have complete rule-base. Basically FRIQ-Learning extends Fuzzy Q-learning by replacing the underlying fuzzy inference method. Standard Fuzzy Q-learning uses zero order Takagi-Sugeno inference method[9]. Changing this method with Fuzzy Rule Interpolation in Vague Environment (FIVE) method leads to FRIQ-learning.

Most of the controllers require an accurate observation and generates crisp conclusions to those observations. FIVE works by adopting the idea of vague environment (VE) by handling the initial and last fuzzy partitions of the fuzzy rule-base as weighting factors [38]. FIVE turns fuzzy interpolation to crisp interpolation. VE is based on the similarity of rules in the rule base or considered elements. In VE, the fuzzy membership function $\mu A(x)$ is indicating the level of similarity, equivalently, or the degree to which x is indistinguishable from specific element action a that is a representative or prototypical element of the fuzzy set $\mu A(x)$. If s_1 is V_{iS1}, s_2 is V_{iS2} and s_n is V_{iSn} and a is V_{ia} then $Q=f_i(s_1,s_2,\ldots,s_n)$. If s_1 is $S_{k,1}$, s_2 is $S_{k,2}$ and $s_N=s_{k,N}$ and $a=A_k$ then $\bar{Q}(s,a)=q_k$ [38]. Where s_1 is observational state with k^{th} rule in the set of the discrete possible states S and 'aa' is the action at k^t rule in the set of the discrete possible actions A. In addition q_k is the observed value of the k^{th} rule[9].

Fuzzy rules that derive Q values of state-action-value pair are given by following equations where S is the vector of states with n dimensions, R is the vector of reward with R_i is i^{th} reward and V_i is the fuzzy set on variables.

$$q_{i_1,i_2\ldots i_n u}^{k+1} = q_{i_1,i_2\ldots i_n u}^{k} + \Delta\bar{Q}_{i,u}^{k+1} \tag{3.11}$$

Where $q_{i_1,i_2\ldots i_n u}^{k+1}$ is the $(k+1)^t$ iteration of fuzzy rule consequents $q_{i_1,i_2\ldots i_n u}$ with action A_u and the following equation is defined for action values.

$$\Delta\bar{Q}_{i,u}^{k+1} = a_{i,u}^{k}(g_{i,u,j} + \gamma max\bar{Q}_{i,u}^{k+1} - \bar{Q}_{i,u}^{k}) \tag{3.12}$$

The factor $g_{i,u,j}$ is the observed reward from state $S_i \rightarrow S_j$ with action A_u. $a_{i,u}^{k}$ is the step size parameter between range [0, 1].

3.3 Proposed System

For agents to explore new or unknown territories, it is necessary to have smart decision-making algorithms such as RL. However, there is still a need of further control layers to have the system or robot to achieve its tasks. From this perspective, three levels of control has been introduced as: a low-level, medium-level and a high-level control which can also be seen in Figure 3.13.

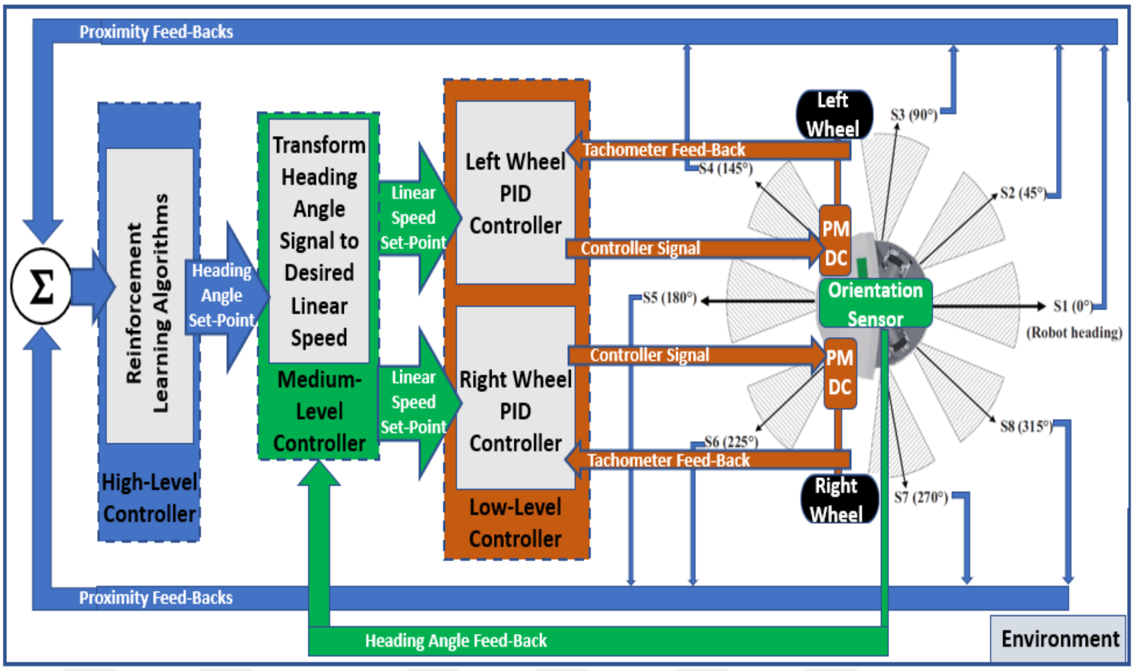


Figure 3. 13 Level-by-Level Controllers of the System

Low-level controller is the simplest control layer of this system which is consisted of the control of PM DC motor based on the tachometer feedback. This control operation is carried out by a PID controller where its parameters are listed in Appendix A and deeply explained in Section 2.2.8. This layer gets the linear speed set-point from medium-level controller layer.

Medium-level controller is based on the orientation of the robot. Collects the heading angle data from the orientation sensor which is deeply explained in Section 2.2.5.3 and validated by the test drive in Section 2.4.1. This layer gets the heading angle set-point from high-level controller layer, then collects the heading angle data from orientation sensor, and accordingly, transforms those signals to linear speed set-point for each wheel. Finally, those set points are being transferred to low-level controller layer as input.

High-level controller is the vital point of this study. This layer is the place where the robot deserves such adjectives as: autonomous, explorer, decision-maker, and learner. High-level controller layer gets the proximity sensor data to get the information for its current state, and according to RL algorithms it selects the next state to go towards. This layer outputs heading angle information and transfers to medium-level controller layer.

4
Experimental Results

A maze is a path or puzzle network of paths that has a goal. It has many obstacles in the path to reach the goal, so it is necessary to cover these obstacles. There is a need of better algorithms and techniques for solving maze for arriving at the goal. The solution to a maze problem, mean finding a route through the maze to arrive at the goal. The Maze can be solved manually or by computing machines. Main motivation is to find goal with the shortest path, less time, less energy and more accuracy.

The view of the selected 50x50 pixel maze with its generic solution on the left-hand side, and the view that the robot is placed at its initial position in this maze on the right-hand side of the Figure 4.1 are given. The goal status for the robot is defined by the red area.

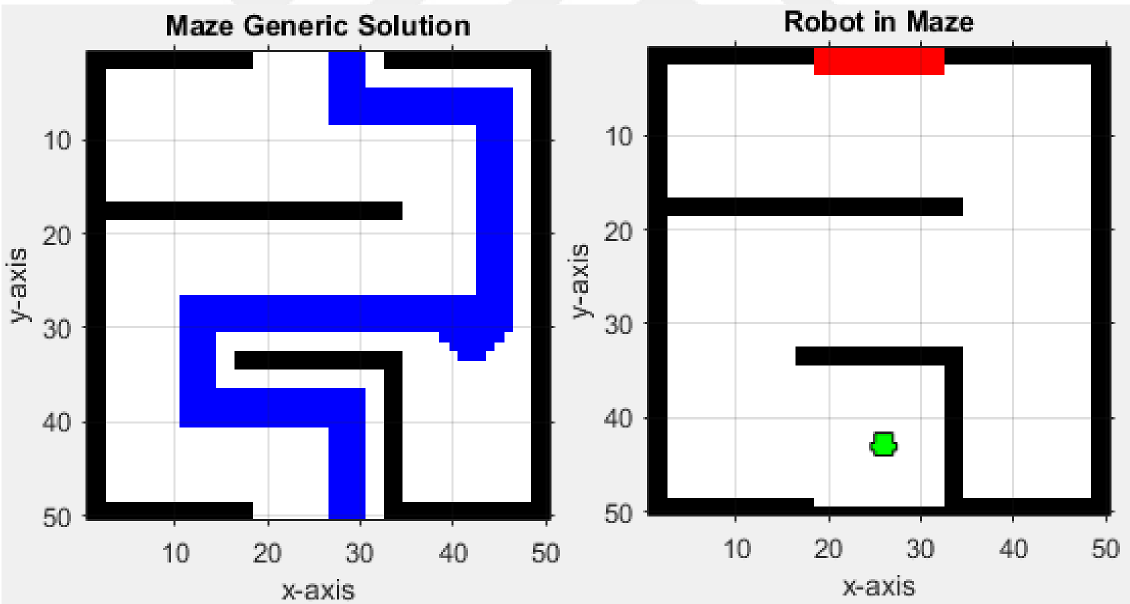


Figure 4. 1 Generic Solution of Maze and Robot in Maze Environment

Eight state-action maze sized matrices which correspond to eight surrounding proximity sensors with 45 degrees around the robot, are defined for each algorithm as already shown in Figure 3.13 before. Two coordinate systems have already been defined in Chapter 2 as robot frame and inertial frame. From the low-level control layer point of view, the robot takes its actions in its own frame. At this point it is necessary to define the possible actions in global inertial frame. Figure 4.2 represents the definition of possible actions in a state and their relations with orientation angle of the robot on inertial frame.

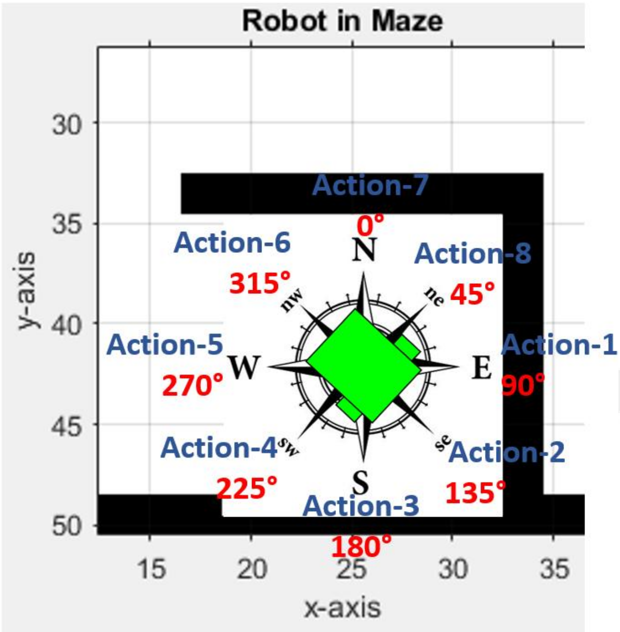


Figure 4. 2 Definition of Actions and Orientation Angles on Inertial Frame

In the figure above, it is seen that robot is heading towards north-west in global frame which means to be 325° orientation angle in inertial frame and has already took the action number 6 in its previous state to reach this state. On this example it can also be seen that frontal proximity sensor is headed north-west, right-front one is headed to north, right one is headed to north-east, right-rear one is headed to east, rear one is headed to south-east, left-rear one is headed to south, left one is headed to south-west, and left-front one. State-action matrices are being feed through this relation with the sensors.

4.1 Simulation Results

The reinforcement learning algorithms that have been introduced in Chapter 3, applied on the same robot model in the same environmental maze with the same initial conditions. Q-learning, SARSA, DQN and FRIQ-learning algorithms are used as high-level controller also known as decision-makers in this study. As already summarized in Table 3.1 that all the algorithms use Q-value as operator but takes decisions according to different mindset. During the simulation maximum step size is defined as 5000 and the maximum episodes are defined as 100. Each episode will finish with if the steps hit to maximum limit or the robot finds the goal. An example in Figure 4.3 shows an implementation of Q-learning agent that finds the goal state after training itself for 37 episodes without finding the goal. This agent has found the goal for the first time in its 4382nd step of 38th episode.

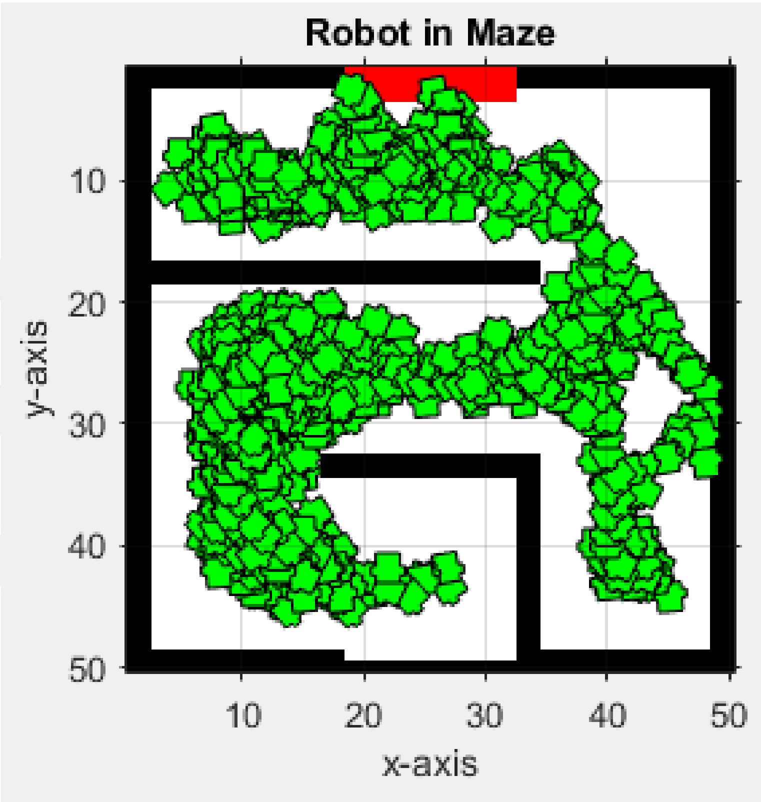


Figure 4. 3 An Example of Episode that Robot Reaches to the Goal

4.1.1 Simulation with Q-learning Algorithm

Q-learning is based on the iterative training of Q-value and this value is the base operator for most of the RL algorithms so that it is the most popular RL algorithm. Simulation results of the algorithm: Q-learning as the decision-making algorithm is given by Figure 4.4.

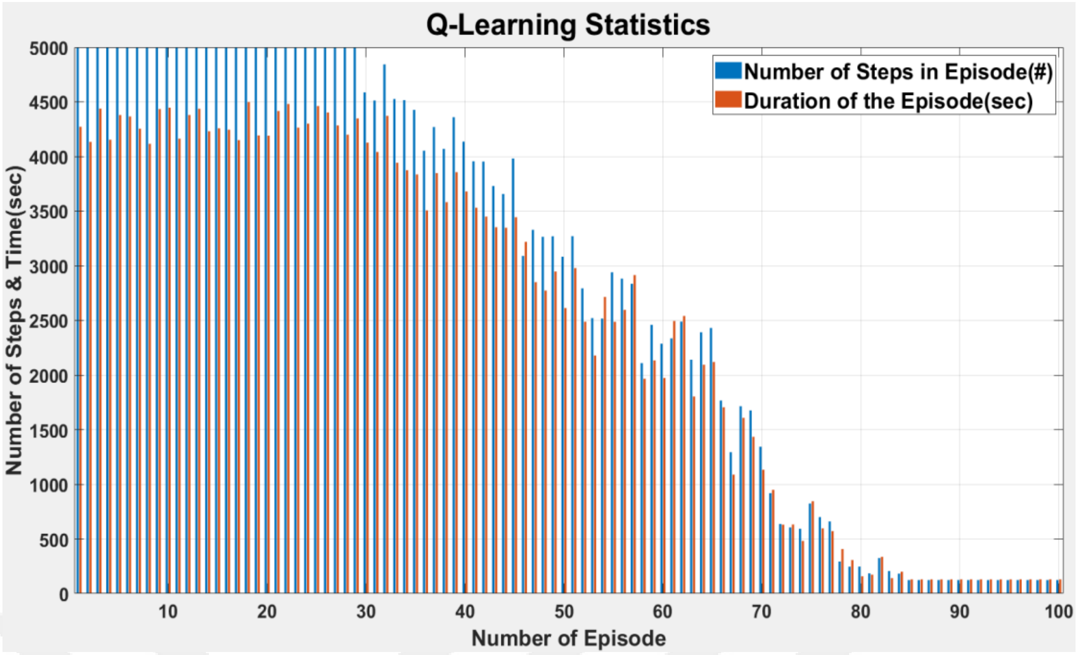


Figure 4. 4 Simulation Statistics of Q-learning

As it can be observed from the statistics that Q-learning agent found the goal for the first time at 30th episode by the step amount of 4587 and have reached to the optimal policy at 85th episode. This means that the training of this agent is neither too fast nor slow. Time and so the consumed energy for solving this maze problem is average when we consider the performance of other agents. It can be said that the performance of the agent's training is not the best one however it steadily got better in each episode by lowering the number of steps to reach to the goal nearly in every episode until it found the optimal policy.

4.1.2 Simulation with SARSA Algorithm

It has been already reviewed in the literary works that SARSA is the weakest algorithm among these four RL algorithm. Simulation results of SARSA as the decision-making algorithm is given by Figure 4.5. As it can be observed from the statistics that SARSA agent found it hard time to reach to the goal until the 35th episode and have reached to the optimal policy at 93rd episode. This means that the training of this agent is slow, and time and so energy consuming for this maze problem. The training performance wasn't also too steady since one episode after another one, the number of steps is not constantly decreasing, the performance of the agent is with ups and downs.

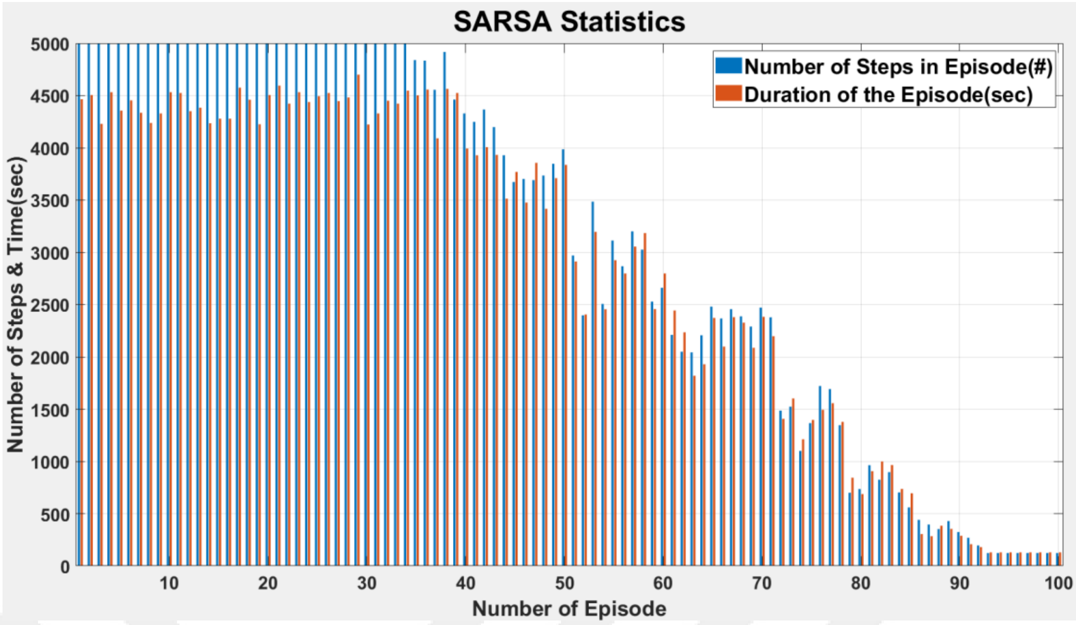


Figure 4. 5 Simulation Statistics of SARSA

4.1.3 Simulation with DQN Algorithm

Simulation results of DQN as the decision-making algorithm is given by Figure 4.6.

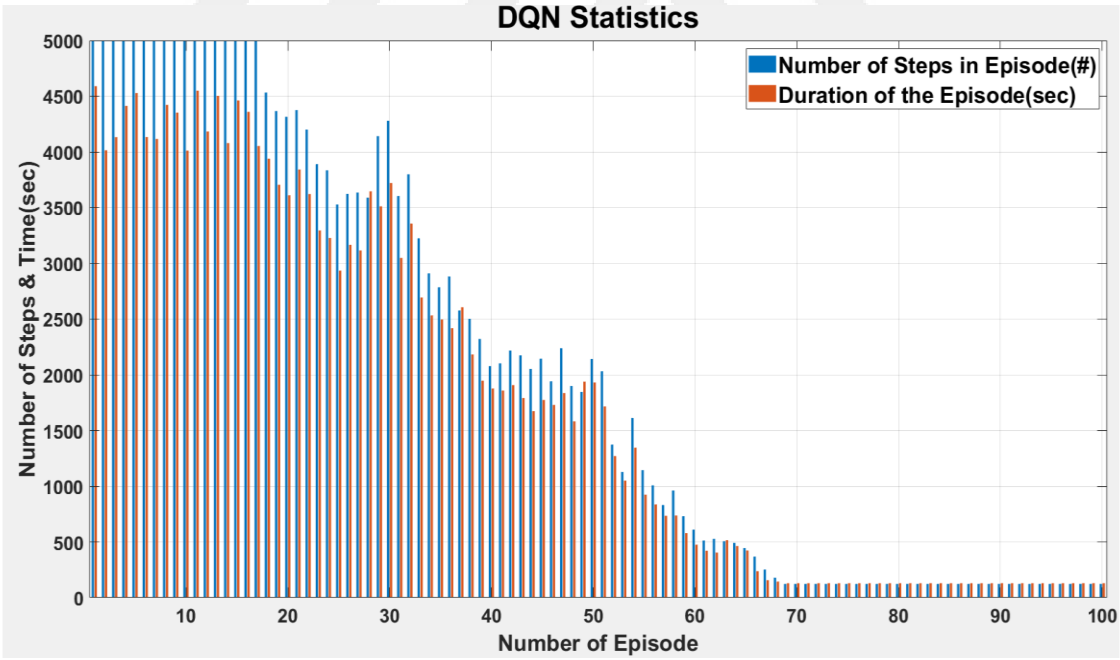


Figure 4. 6 Simulation Statistics of DQN

Here it can be seen that the optimal policy is found at 73rd episode. At the beginning episodes, agent find it hard time to reach to goal since the network has not yet experienced. However, by the episode 50, Q-network is seemed to be well trained and experienced so that starting from 50th episode agent started to lower down the amount of step and the required time to reach the goal.

4.1.4 Simulation with FRIQ-learning Algorithm

FRIQ-learning is an algorithm which is knowledge-based rules that are stated by the experienced human operators. Thus, these rules are human-readable. The rules that are defined for the robot model is given by Table 4.1.

Table 4. 1 Rules for Next Action Selection and Reward for Q-value Calculation

Rule Number	**Rule Operator**	**Right Sensor**	**Right Rear Sensor**	**Rear Sensor**	**Left Rear Sensor**	**Left Sensor**	**Front Left Sensor**	**Front Sensr**	**Front Right Sensor**	**Action**	**Reward**
#1	OR	Goal	Goal	Goal	Goal	Goal	Goal	Goal	Goal	Go To Goal	Positive
#2	AND	**Object Seen**	Not Relevant	Not Relevant	Not Relevant	**Object Seen**	Not Relevant	**Object Seen**	Not Relevant	Turn Back	Negative 1
#3	AND	**Object Seen**	Not Relevant	Not Relevant	Not Relevant	Not Relevant	Not Relevant	**Object Seen**	Not Relevant	Turn Left	Negative 2
#4	AND	Not Relevant	Not Relevant	Not Relevant	Not Relevant	**Object Seen**	Not Relevant	**Object Seen**	Not Relevant	Turn Right	Negative 2
#5	AND	Not Relevant	Not Relevant	**Object Seen**	Not Relevant	Not Relevant	Not Relevant	Not Relevant	Not Relevant	Go Ahead	Neutral
#6	AND	**Object Seen**	Not Relevant	**Object Seen**	Not Relevant	Not Relevant	Not Relevant	Not Relevant	Not Relevant	Go Right or Ahead	Neutral
#7	AND	Not Relevant	Not Relevant	**Object Seen**	Not Relevant	**Object Seen**	Not Relevant	Not Relevant	Not Relevant	Go Left or Ahead	Neutral

There are seven rules, and these are based on robot frame observations that are being fed by eight proximity sensors surrounding the robot. Only the first rule is an *OR* relational operator based, rest of the rules are, *AND* relational operator based. Actions are defined in robot frame. Rewards are categorized under five categories. *Positive* rewards mean 1000, *Negative 1* means -1, *Negative 2* means -0.5, *Neutral* means 0 in terms of computational value.

Simulation results of FRIQ-learning as the decision-making algorithm is given by Figure 4.7. First contact with the goal for the FRIQ agent is at 23rd episode with the step number of 4930 and the optimal policy found is at Statistics show that the training performance is not as steady as DQN however strangely the episode duration is less then number of the steps for most of the episodes. So, this can tell us that knowledge-based fuzzy-rules are good in terms of consumed energy to reach the goal.

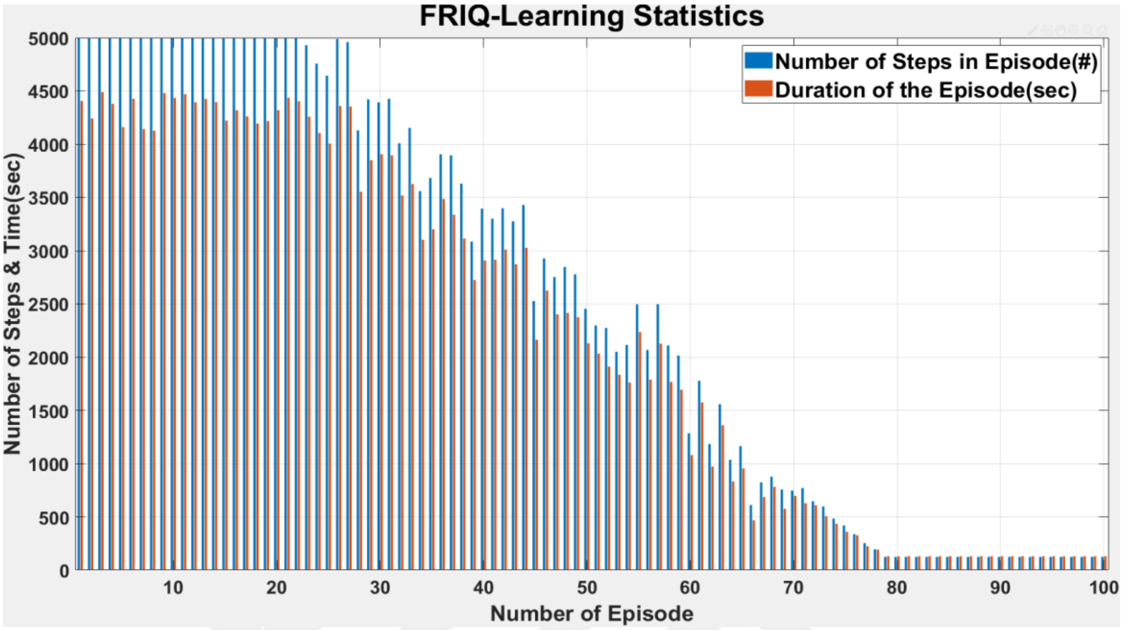


Figure 4. 7 Simulation Statistics of FRIQ-learning

4.2 Comparison of the Simulation Results

The robot model that has been successfully developed in Chapter 2, has been used through the simulations of this chapter by the reinforcement learning algorithms that have been introduced in Chapter 3.

For the purpose of ensuring the robustness of this study, the robot is placed in three different locations as an initial position. The results and graphs that are given by Section 4.1 and its subsections are by the robot's initialization from the first position which is given by Figure 4.1. Other two initialization positions of the robot are given by Figure 4.8 below. Initialization positions of the robot in the maze were as follows: (26,42), (10,25) and (45,45).

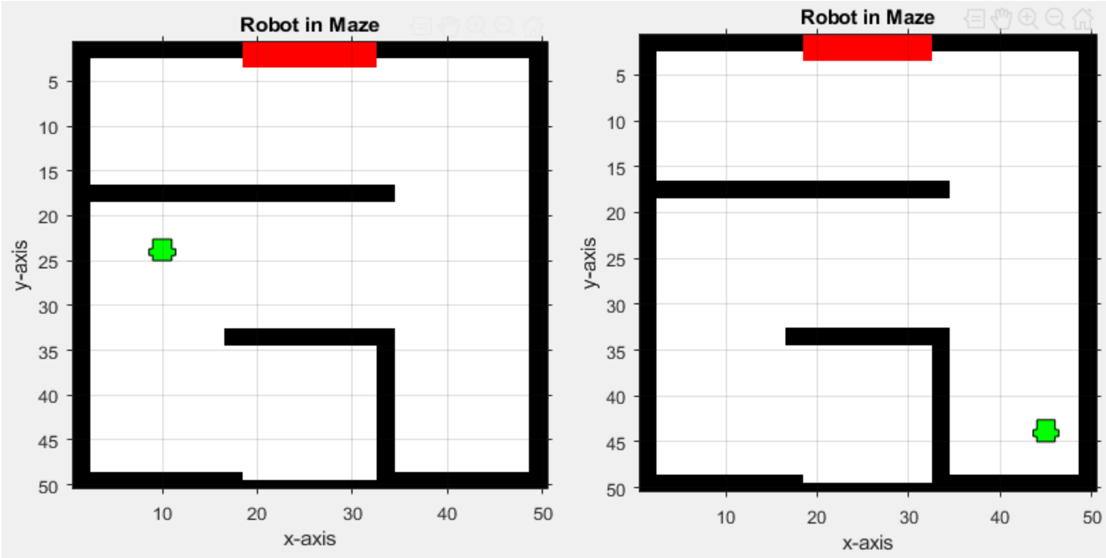


Figure 4. 8 Second and Third Initialization Positions of Robot inside Maze

Four algorithms that are mentioned in previous sections, have been performed with different starting points of the robot in same maze with same initialization conditions. The statistical results are given by Table 4.2.

Table 4. 2 Statistical Comparison of the Algortihms

		Episode Number for Reaching the Goal First Time(#)	Total Consumed Energy During Traning(kW/h)	Mean Consumed Energy per Episode During Training(W/h)	Episode Number of the End of Learning(#)
Robot Initialization Position No:1 (26,42)	SARSA	35	2.74	29.49	93
	Q-Learning	30	2.15	25.89	83
	DQN	**18**	**1.53**	**21.02**	**73**
	FRIQ-Learning	23	1.71	21.70	79
Robot Initialization Position No:2 (10,25)	SARSA	29	2.55	28.98	88
	Q-Learning	26	1.93	24.43	79
	DQN	**11**	**1.3**	20.00	**65**
	FRIQ-Learning	15	1.39	**19.86**	70
Robot Initialization Position No:3 (45,45)	SARSA	30	2.58	27.74	93
	Q-Learning	24	2.01	24.22	83
	DQN	**18**	**1.32**	21.29	**62**
	FRIQ-Learning	23	1.5	**20.83**	72

In the table above, four conditions have taken into consideration. First column shows the number of episodes that represents the robot's reaching of the goal for the first time. Second column is the total energy consumption during the training phase, until robot finds the optimal policy, in other words the fastest route which is also the end of learning phase. Third column represents the average energy consumption during the learning phase for each episode. Fourth column represents the episode number which stands for the end of learning phase in other words finding the fastest route. The table shows that DQN and FRIQ-learning have similar results in terms of mean energy consumption and are far better than rest of them. It is necessary to have a closer look on DQN and FRIQ-learning. DQN is the quickest in terms of finding the goal for the first time, finding the optimal policy. In addition, DQN consumes less energy in total. However, FRIQ-learning is competitive with DQN in terms of the mean energy consumption data.

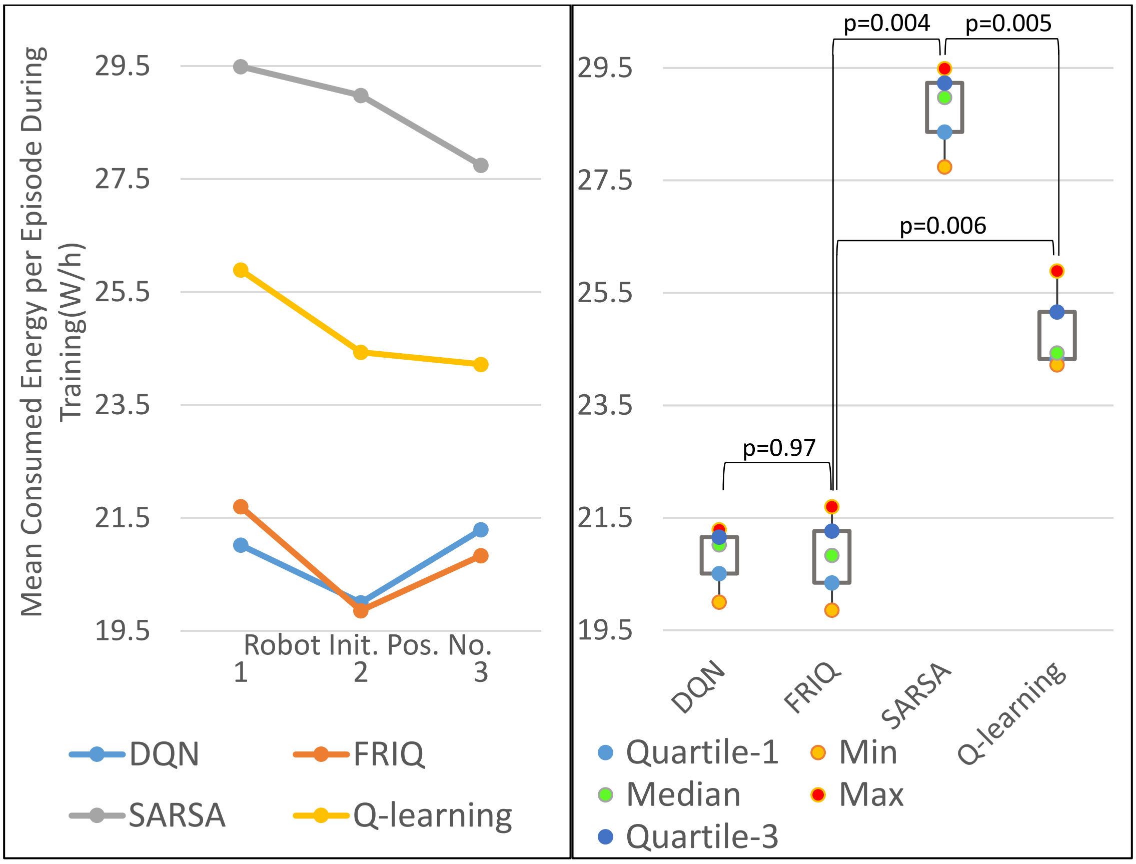


Figure 4. 9 Analysis of Mean Energy Consumption of Different Algorithms

Figure 4.9 provides the information of the mean energy consumption of different algorithms. In the first graph the mean energy consumption per episode is given for different robot initialization positions for each algorithm. The second graph represents the median, quartile, minimum and maximum values for each algorithm’s mean energy consumption data. Two-tailed T-Test has been applied on this data and the level of marginal significance values which are represented by *p* are given in the figure. At this point, t-test outputs that there is no significant difference between mean energy consumption per episode during the learning phase of DQN and FRIQ-learning since the level of marginal significance value is 0.97. However, other algorithm pairs show significant differences which are also given in the figure by the level of marginal significance values.

As already reviewed during literature survey, Q-learning has advantages upon SARSA in maze problems. Thus, considering the comparison results in the previous section, it is seen that DQN and FRIQ-learning have superiority upon Q-learning and SARSA in terms of finding the optimal policy which also means the shortest path for reaching the goal, with a fewer number of episodes. This means these two algorithms are advantageous in terms of time and energy used to find the optimal policy.

DQN and FRIQ-learning can be considered as tied rivals, however DQN has a slightly better performance in terms of finding the optimal policy with a fewer number of episodes since this algorithm enables the future of remembering past experiences with no good results. On the other hand, FRIQ-learning has a slightly better performance on the corners and narrow regions that are surrounded by the walls of the maze since this algorithm has predefined knowledge-based rules for robot to move away from those regions to continue through its way to the goal. This means saving time and energy while finding the goal.

5
Conclusion and Reccommendations

It is aimed to determine the most successful RL algorithms and compare their performances. During the literature survey it is found out that the most powerful algorithms are Q-learning, SARSA, DQN and FRIQ-learning methods. There are comparative performance studies in literature between SARSA and Q-learning, Q-learning and DQN, Q-learning and FRIQ-learning. These studies show DQN and FRIQ-learning have better performance according to other ones however there were no direct comparison work done between these two. This deficiency in the literature is being completed by this thesis.

A mobile robot model has been developed for the simulation purpose of this dissertation. Initially a mathematical model which consists of the electrical actuators, proximity sensors, orientation sensor, tachometer, robot kinematics and dynamics, has been introduced. Later, for the aim of speeding up the simulations to have a benchmark data between different reinforcement algorithms, the Simulink model has been linearized and so simplified. By the test drive the robot's interaction with the environment is validated. Thus, the model that has been successfully developed, is used during the performance comparative simulations.

Considering the comparison results in the previous chapter, it is seen that DQN and FRIQ-learning have superiority upon Q-learning and SARSA in terms of finding the optimal policy which also means the shortest path for reaching the goal, with a fewer number of episodes. This means these two algorithms are advantageous in terms of time and energy used to find the optimal policy.

According to FRIQ-learning, DQN has a slightly better performance in terms of finding the optimal policy with a fewer number of episodes since this algorithm enables the future of remembering past experiences with no good results. On the other hand, FRIQ-learning has a better performance on the corners and narrow regions that are surrounded by the walls of the maze since this algorithm has predefined knowledge-based rules for robot to move away from those regions to continue through its way to the goal. This means saving time and energy with less while finding the goal. Lowering the consumed energy for an autonomous robot is vital since its off-grid batteries will last longer for more task to accomplish.

As a result, these findings show that, according to the case of application, a hybrid algorithm that involves fuzzy rules of FRIQ-learning and DQN might result in a better performance in terms of number of episodes and the consumed energy to find the optimal policy and number of steps.

References

[1] Chen Xia. Intelligent Mobile Robot Learning In Autonomous Navigation. Automatic Control Engineering. Ecole Centrale De Lille, 2015.

[2] Altuntaş, İmal, Öztürk (2016). Reinforcement Learning-Based Mobile Robot Navigation. Turkish Journal Of Electrical Engineering & Computer Sciences.

[3] Zhang, Kaichena, Niroui, Farzad, Ficocelli, Maurizio, Goldie. (2018). Robot Navigation Of Environments With Unknown Rough Terrain Using Deep Reinforcement Learning. 1-7.

[4] Stone, Peter, Sutton, Richard, Kuhlmann, Gregory. (2005). Reinforcement Learning For Robocup Soccer Keepaway. Adaptive Behavior - ADAPT BEHAV. 13.

[5] K. Kopsa, Reinforcement Learning Control For Autorotation Of A Simple Point-Mass Helicopter Model, Master Of Science Dissertation, The Graduate School Of Natural And Applied Science, Middle East Technical University, 2018.

[6] Tutsoy, Onder. (2014). On The Reinforcement Learning Analysis And Learning The Control Of Humanoid Robot Leg.

[7] B. Coors, 'Navigation Of Mobile Robots In Human Environments With Deep Reinforcement Learning', Master Of Science Dissertation, Computer Science And Communication Department, Kth Royal Institute Of Technology, 2016.

[8] H. Yao, Model-Based Reinforcement Learning With State And Action Abstraction, Ph.D. Dissertation, Department Of Comuter Science, University Of Alberta, 2016.

[9] R. L. Kamalapurkar, Model-Based Reinforcement Learning For Online Approximate Optimal Control, Ph.D. Dissertation, University Of Florida, 2014.

[10] Y. Liang, Model-Based Appreticeship Learning For Robotics In High-Dimensional Spaces, Msc. Dissertation, Department Of Computing, Imperial College London, 2014.

[11] Farahmand, Amir-Massoud, Shademan, Azad, Jagersand, Szepesvári, Csaba. (2009). Model-Based And Model-Free Reinforcement Learning For Visual Servoing. Proceedings - IEEE International Conference On Robotics And Automation.

[12] De Bruin. (2016). Improved Deep Reinforcement Learning For Robotics Through Distribution-Based Experience Retention.

[13] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David ,Rusu, Andrei, Demis. (2015). Human-Level Control Through Deep Reinforcement Learning.

[14] Lillicrap. (2015). Continuous Control With Deep Reinforcement Learning. Corr.

[15] Bhagat(2018). Deep Reinforcement Learning For Soft Robotic Applications: Brief Overview With Impending Challenges.

[16] Gu, Shixiang, Holly, Ethan, Lillicrap, Timothy, Levine, Sergey. (2016). Deep Reinforcement Learning For Robotic Manipulation.

[17] Zhang, Jingwei, Tobias Springenberg, Jost, Boedecker, Joschka, Burgard, Wolfram. (2017). Deep Reinforcement Learning With Successor Features For Navigation Across Similar Environments.
[18] Sheoran, Poonam (2017). Comparative Analysis Of Reinforcement Learning Methods For Optimal Solution Of Maze Problems. IJARCS International Journal Of Avanced Research In Computer Science.
[19] Vincze, David. (2017). Fuzzy Rule Interpolation And Reinforcement Learning.
[20] Russell, S.J. And Norvig, T., Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall, New Jersey, 2003.
[21] Knudson, Tumer, K. (2011). Adaptive Navigation For Autonomous Robots. Robotics And Autonomous Systems, 59, 410–420.
[22] O. Mohareri, Mobile Robot Trajectory Tracking Using Neural Networks, Master Of Science Dissertation, University Of Sharjah, 2009.
[23] G. Campion, G. Bastin And B. D'andrea-Novel, "Structural Properties And Classification Of Kinematic And Dynamic Models Of Wheeled Mobile Robots," [1993] Proceedings Ieee International Conference On Robotics And Automation, Atlanta, Ga, Usa, 1993, Pp. 462-469 Vol.1.
[24] Dhaouadi Ahmad Abu Hatab, Rached. (2013). Dynamic Modelling Of Differential-Drive Mobile Robots Using Lagrange And Newton-Euler Methodologies: A Unified Framework. Advances In Robotics & Automation.
[25] Neĭmark JI, Fufaev NA (1972) Dynamics Of Nonholonomic Systems: Translations Of Mathematical Monographs. American Mathematical Society
[26] Julius Maximilian (2003), Kinematics Of A Car-Like Mobile Robot Mobile Robot Basic – Differential Drive'', Institute Of Informatics VII, Experiment Notes
[27] Salem, F.A. (2013). Dynamic And Kinematic Models And Control For Differential Drive Mobile Robots. International Journal Of Current Engineering And Technology.
[28] Gade, K. (2016). The Seven Ways To Find Heading. Journal Of Navigation.
[29] Nguyen, B.M., Fujimoto, H., Hori, Y. (2014). Yaw Angle Control For Autonomous Vehicle Using Kalman Filter Based Disturbance Observer.
[30] Hall, Carrie. (2019). Manufacturing A Calibration Device For Orientation Sensors.
[31] Bashir M. Y. Nouri (2005), Modeling And Control Of Mobile Robot , Proceeding Of The First International Conference On Modeling, Simulation And Applied Optimization, Sharjah, U.A.E. February 1-3.
[32] J. R. Asensio. L. Montano, (2002) A Kinematic And Dynamic Model-Based Motion Controller For Mobile Robot, 15th Triennial World Congress, Barcelona, Spain.
[33] Puterman, M.L. (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st Edn.

[34] Smallwood, Sondik (1973). The optimal control of partially observable markov processes over a finite horizon. Operations Research, 21, 1071–1088.
[35] Sutton, Barto, A.G. (1998). Reinforcement learning: An introduction. MIT press.
[36] Watkins, C.J.C.H. (1989). Learning from delayed rewards. Ph.D. thesis, University of Cambridge England.
[37] Rummery, G.A., Niranjan, M. (1994). On-line q-learning using connectionist systems.
[38] D. Vincze and S. Kovács, “Fuzzy Rule Interpolation-based Q-learning,” pp. 55–60, 2009 IEEE.

A
Parameters of Mobile Robot Model

The followings are the various parameters of the mobile robot model:

Battery Supply Voltage: 12 Volts,

Gear Ratio: 3,

Armature Resistance: 0.1557 Ohms,

Motor Torque Constant: 1.1882 Nm/A,

Armature Inductance: 0.82 MH,

Geared-Motor Inertia: 0.271 kgm^2,

Geared-Motor Viscous Damping: 0.271 Nms,

Motor Back EMF Constant: 1.185 rad/Vs,

Wheel radius: 0.075,

Mobile Robot Height: 0.92 m,

Mobile Robot Width: 0.58 m,

Total Mass of the Mobile Robot: 100 kg,

Total inertia: 0.2752 kgm^2,

Equivalent damping at the armature of the motor: 0.3922 Nms,

Tachometer Constant: 1.8 rad/sec,

Aerodynamic Drag Coefficient: 0.8,

Distance between Center of each Rear Wheel: 0.4 m,

Air Density: 1.25 kg/m^3,

Coefficient of Lift: 0.1,

Inclination Angle: 45 deg,

Proportional Gain Value of the PID Controller: 3417.3153

Integral Gain Value of the PID Controller: 6288.8867

Derivative Gain Value of the PID Controller: 417.8632

Filter Coefficient Value of the PID Controller: 64399278

B
Proximity Sensor Datasheet

SHARP GP2Y0A21YK/GP2Y0D21YK

GP2Y0A21YK/ GP2Y0D21YK

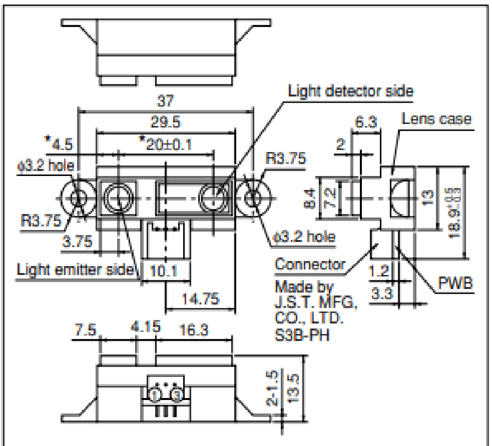
General Purpose Type Distance Measuring Sensors

■ Features

1. Less influence on the color of reflective objects, reflectivity
2. Line-up of distance output/distance judgement type
   Distance output type (analog voltage) : **GP2Y0A21YK**
   Detecting distance : 10 to 80cm
   Distance judgement type : **GP2Y0D21YK**
   Judgement distance : 24cm
   (Adjustable within the range of 10 to 80cm [Optionally available])
3. External control circuit is unnecessary
4. Low cost

■ Outline Dimensions

(Unit : mm)



* The dimensions marked * are described the dimensions of lens center position.
* Unspecified tolerance : ±0.3mm

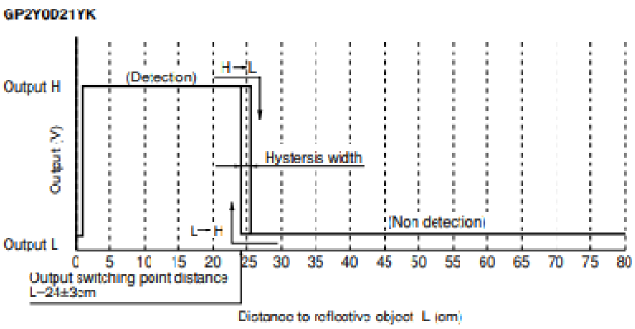
■ Applications

1. TVs
2. Personal computers
3. Cars
4. Copiers

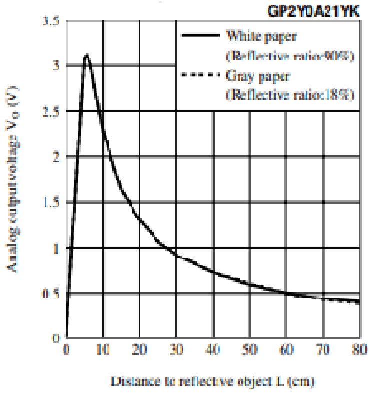
■ Absolute Maximum Ratings (T_a=25°C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	−0.3 to +7	V
Output terminal voltage	V_O	−0.3 to V_{CC} +0.3	V
Operating temperature	T_{opr}	−10 to +60	°C
Storage temperature	T_{stg}	−40 to +70	°C

Distance Characteristics



Analog Output Voltage vs. Distance to Reflective Object



C
Orientation Sensor Datasheet

Technical Note

3DM-GX1®
Orientation Sensor

Timer Ticks, Calculation Cycle and Data Output Rates

Overview

MicroStrain's 3DM-GX1® orientation sensor allows the user to optimize the data output rate of finished data quantities to suit varying applications. Within certain limits, the user may speed up or slow down the number of data quantities delivered per second over the 3DM-GX1®'s default data delivery setting.

- This technical note assumes that the user has a working knowledge of the 3DM-GX1® and has read the *Calculation Cycle and Data Output Rate* section of the 3DM-GX1® Data Communication Protocol manual at: http://www.microstrain.com/manuals/3DM-GX1%20Data%20Communication%20Protocol%203101.pdf.
- **Table 1** describes all of the data quantity commands available for the 3DM-GX1®. The command name and the command byte are given.

Table 1: Commands for Data Quantities

Cmd	Hex
Send Raw Sensor Bits	0x01
Send Gyro-Stabilized Vectors	0x02
Send Instantaneous Vectors	0x03
Send Instantaneous Quaternion	0x04
Send Gyro-Stabilized Quaternion	0x05
Send Instantaneous Orientation Matrix	0x0A
Send Gyro-Stabilized Orientation Matrix	0x0B
Send Gyro-Stabilized Quaternion & Vectors	0x0C
Send Instantaneous Euler Angles	0x0D
Send Gyro-Stabilized Euler Angles	0x0E
Send Gyro-Stabilized Quaternion & Instantaneous Vectors	0x12
Send Gyro-Stabilized Euler Angles & Accel & Rate Vector	0x31

Method

- A series of tests were conducted to determine the effect of differing timer tick intervals on the data output rate. Even timer tick intervals between 1 and 10 milliseconds were tested along with the default factory setting of 6.5536 milliseconds. The baud rate was set to the factory default 38, 400. The 3DM-GX1® was operated in continuous mode. The results are shown in **Table 2**.
- As an example, the blue cell in the table shows that if the timer tick interval is set to 7 and the data quantity *Gyro-Stabilized Orientation Matrix* (command 0x0B) is requested, the 3DM-GX1® requires 2 timer ticks to output a finished data quantity.
- As another example, the red cell in the table shows that if the timer tick interval is set to 10 and the data quantity *Gyro-Stabilized Quaternion & Instantaneous Vectors* (command 0x12) is requested, the 3DM-GX1® requires 1 timer tick to output a finished data quantity.
- The number of timer ticks required for any given timer tick interval will not change, no matter the length of the data acquisition session, except with those settings where we have indicated the letter T, T for transitioning. In these configurations, the number of timer ticks to output a data quantity will vary throughout the session. If the user is counting on a constant output of data, these configurations should be avoided.

Table 2: Timer Ticks required to complete Calculation Cycle

mSecs	6.5536	10.0	9.0	8.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0
Cmd											
0x01	T	1	1	1	1	2	2	2	3	4	7
0x02	2	1	1	1	T	2	2	2	3	4	T
0x03	2	1	1	1	T	2	2	2	3	4	T
0x04	2	1	T	2	2	2	2	3	T	5	T
0x05	2	1	T	2	2	2	2	3	T	5	T
0x0A	2	1	1	2	2	2	2	3	3	5	9
0x0B	2	1	1	2	2	2	2	3	3	5	9
0x0C	2	1	2	2	2	2	2	3	4	5	10
0x0D	2	1	2	2	2	2	2	3	4	5	10
0x0E	2	1	2	2	2	2	2	3	4	5	10
0x12	2	1	2	2	2	2	2	3	4	5	10
0x31	2	1	2	2	2	2	2	3	4	5	10

D
State-Space Representation of Half Robot Model

```
TF =

        212.5

  ------------------------

  s^2 + 1.268 s + 1.007e04

Name: Linearization at model initial condition

Continuous-time transfer function.


A =

   1.0e+04 *

   -0.0001   -1.0070

    0.0001        0

B =

     1

     0

C =

       0  212.5000

D =

     0
```

E
Implementation Codes in MATLAB

```
clear all
close all
clc

% Read the Maze From The Image and Create a Matrix Accordingly
Maze_Image=imread('Maze_Simple.png');
[row,column,z]=size(Maze_Image);
Maze_Grayscale_Matrix = zeros(row,column);
R_Maze_Image = Maze_Image(:,:,1);
G_Maze_Image = Maze_Image(:,:,2);
B_Maze_Image = Maze_Image(:,:,3);
for a=1:row
    for b=1:column
        if R_Maze_Image(a,b)==255
            Maze_Grayscale_Matrix(a,b)=255;
        else
            Maze_Grayscale_Matrix(a,b)=0;
        end
    end
end
for c=1:column
    if Maze_Grayscale_Matrix(row,c)== 255
        x_start=c;
        break
    end
end
for d=x_start:column
    if Maze_Grayscale_Matrix(row,d)== 0
        x_end=d;
        break
    end
end
x_origin = x_start+(x_end-x_start)/2;

for cc=1:column
    if Maze_Grayscale_Matrix(1,cc)== 255
        x_start_1=cc;
        break
    end
end
for dd=x_start_1:column
    if Maze_Grayscale_Matrix(1,dd)== 0
        x_end_1=dd;
        break
    end
```

```
end
x_origin_1 = x_start_1 + (x_end_1-x_start_1)/2;

% Close entrance of maze
Entrance_Closed=[ones(row-1,column) ; ones(1,x_start-1)
zeros(1,x_end-x_start) ones(1,column-x_end+1)];
Maze_Grayscale_Matrix = Maze_Grayscale_Matrix.*Entrance_Closed;

% Set Goal
Goal_Set_G=255*[zeros(3,x_start_1-1) ones(3,x_end_1-x_start_1)
zeros(3,column-x_end_1+1); zeros(row-3,column)];
Goal_Set_B=255*[zeros(3,x_start_1-1) ones(3,x_end_1-x_start_1)
zeros(3,column-x_end_1+1); zeros(row-3,column)];
Maze_Grayscale_Matrix_G = Maze_Grayscale_Matrix-Goal_Set_G;
Maze_Grayscale_Matrix_B = Maze_Grayscale_Matrix-Goal_Set_B;
Maze_GrayScale_Image = cat(3, Maze_Grayscale_Matrix,
Maze_Grayscale_Matrix_G, Maze_Grayscale_Matrix_B);
Maze_Matrix_Total=Maze_Grayscale_Matrix+Maze_Grayscale_Matrix_B+
Maze_Grayscale_Matrix_G;

% Build a state-action matrix by finding all valid states from
maze
Q=zeros(size(Maze_Matrix_Total,1),size(Maze_Matrix_Total,2),8);

% learning rate, discount factor and other initial training
settings
alpha=0.9;
gamma=0.9;
Iteration_Count=0;
[goalX,goalY,val] = find(Maze_Matrix_Total == 255);
Max_Episode_Number=1;
Max_Step_Number=1000;
Count_Episode=0;

%Parameters of the System Model
syms s;
load('SimulinkmodelMAT.mat')
Kp=3417.31529318326;
Ki=6288.88665554317;
Kd=417.863242935618;
N_coeff=6439.92785870288; % filter coefficient
N_coeff_inv=1/N_coeff;
PID=pid(Kp,Ki,Kd,N_coeff_inv);
PID_TF=tf(PID);
LeftWheelTF = tf(LeftWheelModel);
Closed_Loop_Left_Wheel =  feedback(PID_TF*LeftWheelTF,1);
RightWheelTF = tf(RightWheelModel);
Closed_Loop_Right_Wheel =  feedback(PID_TF*RightWheelTF,1);


for k=1:Max_Episode_Number
    Count_Episode=Count_Episode+1;
    DesiredLinearSpeedLeft = 0;
```

```
    DesiredLinearSpeedRight =0;
    Theta_Robot_plus=0;
    y_offset=10;
    Tx_plus=0;
    Ty_plus=y_offset;
    theta_vector(1)=0;
    Tx_vector(1)=0;
    Ty_vector(1)=0;
    Initial_X_Column_Position=round(x_origin);
    Initial_Y_Row_Position=row-Ty_plus;
    time =1;
    time_total=0;
    Count_Step=0;

 %     while
Maze_Matrix_Total(Initial_Y_Row_Position,Initial_X_Column_Positi
on) ~= 255
        for i=1:1:Max_Step_Number
        Count_Step=Count_Step+1;

        %Previous state
        Hold_Y_Row_Position=Initial_Y_Row_Position;
        Hold_X_Column_Position=Initial_X_Column_Position;

        %Linearized Model Operations
        opt_Left =
stepDataOptions('InputOffset',0,'StepAmplitude',DesiredLinearSpe
edLeft);

[LeftWheelLinearSpeed,time_L]=step(Closed_Loop_Left_Wheel,time,o
pt_Left);
        opt_Right =
stepDataOptions('InputOffset',0,'StepAmplitude',DesiredLinearSpe
edRight);

[RightWheelLinearSpeed,time_R]=step(Closed_Loop_Left_Wheel,time,
opt_Right);
        opt_Theta =
stepDataOptions('InputOffset',0,'StepAmplitude',+LeftWheelLinear
Speed(end)-RightWheelLinearSpeed(end));

[ThetaRobotOrientation,time_T]=step(ThetaRobotOrientationCalc,ti
me,opt_Theta);

Theta_Robot_plus=mod(Theta_Robot_plus+ThetaRobotOrientation(end)
,2*pi);
        opt_Tx =
stepDataOptions('InputOffset',0,'StepAmplitude',sin(Theta_Robot_
plus)*(LeftWheelLinearSpeed(end)+RightWheelLinearSpeed(end))/2);
        [Tx,time_Tx]=step(X_DisplacementCalc,time,opt_Tx);
        Tx_plus=Tx_plus+Tx(end);
```

```
        opt_Ty =
stepDataOptions('InputOffset',0,'StepAmplitude',cos(Theta_Robot_
plus)*(LeftWheelLinearSpeed(end)+RightWheelLinearSpeed(end))/2);
        [Ty,time_Ty]=step(Y_DisplacementCalc,time,opt_Ty);
        Ty_plus=Ty_plus+Ty(end);

        %Current State

Initial_X_Column_Position=round(round(x_origin)+Tx_plus);
        Initial_Y_Row_Position=round(row-Ty_plus);
        currentDirection=round(Theta_Robot_plus*180/pi);

        %Location of Proximity Sensors and Range Definition
        th = linspace( 0+Theta_Robot_plus,
2*pi+Theta_Robot_plus, 9); % Number and Position of Proximity
Sensors on Robot
        R = 15;  %Proximity Sensor Range
        x = max(1,min(column,round(R*cos(th) +
Initial_X_Column_Position))); %Max Possible X Range point of
each proximity sensor
        y = max(1,min(row,round(R*sin(th) +
Initial_Y_Row_Position)));%Max Possible Y Range point of each
proximity sensor
        %Proximity Sensor Right Feed-back %While theta=0, X1=0
        Proximity_Vector_Right_x=[x(1)
Initial_X_Column_Position];
        Proximity_Vector_Right_y=[y(1) Initial_Y_Row_Position];

Proximity_Vector_Right=improfile(Maze_Matrix_Total,Proximity_Vec
tor_Right_x,Proximity_Vector_Right_y);
        Obstacle_Right =any(Proximity_Vector_Right (:)<255);
        %Proximity Sensor Right-Behind Feed-back %While theta=0,
X2=45
        Proximity_Vector_Right_Behind_x=[x(2)
Initial_X_Column_Position];
        Proximity_Vector_Right_Behind_y=[y(2)
Initial_Y_Row_Position];

Proximity_Vector_Right_Behind=improfile(Maze_Matrix_Total,Proxim
ity_Vector_Right_Behind_x,Proximity_Vector_Right_Behind_y);
        Obstacle_Right_Behind =any(Proximity_Vector_Right_Behind
(:)<255);
        %Proximity Sensor Behind Feed-back %While theta=0, X3=90
        Proximity_Vector_Behind_x=[x(3)
Initial_X_Column_Position];
        Proximity_Vector_Behind_y=[y(3) Initial_Y_Row_Position];

Proximity_Vector_Behind=improfile(Maze_Matrix_Total,Proximity_Ve
ctor_Behind_x,Proximity_Vector_Behind_y);
        Obstacle_Behind =any(Proximity_Vector_Behind (:)<255);
        %Proximity Sensor Left-Back Feed-back %While theta=0,
X4=135
        Proximity_Vector_Left_Behind_x=[x(4)
Initial_X_Column_Position];
```

```
        Proximity_Vector_Left_Behind_y=[y(4)
Initial_Y_Row_Position];

Proximity_Vector_Left_Behind=improfile(Maze_Matrix_Total,Proximi
ty_Vector_Left_Behind_x,Proximity_Vector_Left_Behind_y);
        Obstacle_Left_Behind =any(Proximity_Vector_Left_Behind
(:)<255);
        %Proximity Sensor Left Feed-back %While theta=0, X5=180
        Proximity_Vector_Left_x=[x(5)
Initial_X_Column_Position];
        Proximity_Vector_Left_y=[y(5) Initial_Y_Row_Position];

Proximity_Vector_Left=improfile(Maze_Matrix_Total,Proximity_Vect
or_Left_x,Proximity_Vector_Left_y);
        Obstacle_Left =any(Proximity_Vector_Left (:)<255);
        %Proximity Sensor Ahead-Left Feed-back %While theta=0,
X6=225
        Proximity_Vector_Ahead_Left_x=[x(6)
Initial_X_Column_Position];
        Proximity_Vector_Ahead_Left_y=[y(6)
Initial_Y_Row_Position];

Proximity_Vector_Ahead_Left=improfile(Maze_Matrix_Total,Proximit
y_Vector_Ahead_Left_x,Proximity_Vector_Ahead_Left_y);
        Obstacle_Ahead_Left =any(Proximity_Vector_Ahead_Left
(:)<255);
        %Proximity Sensor Ahead Feed-back %While theta=0, X7=270
        Proximity_Vector_Ahead_x=[x(7)
Initial_X_Column_Position];
        Proximity_Vector_Ahead_y=[y(7) Initial_Y_Row_Position];

Proximity_Vector_Ahead=improfile(Maze_Matrix_Total,Proximity_Vec
tor_Ahead_x,Proximity_Vector_Ahead_y);
        Obstacle_Ahead =any(Proximity_Vector_Ahead (:)<255);
        %Proximity Sensor Ahead-Right Feed-back %While theta=0,
X8=315
        Proximity_Vector_Ahead_Right_x=[x(8)
Initial_X_Column_Position];
        Proximity_Vector_Ahead_Right_y=[y(8)
Initial_Y_Row_Position];

Proximity_Vector_Ahead_Right=improfile(Maze_Matrix_Total,Proximi
ty_Vector_Ahead_Right_x,Proximity_Vector_Ahead_Right_y);
        Obstacle_Ahead_Right =any(Proximity_Vector_Ahead_Right
(:)<255);

        %Goal Reaching Statement
        if
Maze_Matrix_Total(Initial_Y_Row_Position,Initial_X_Column_Positi
on) == 255
                rewardVal = 1000;
                break
        end
```

```
        %Reward Value Calculation from Environment Observation
Through
        %Heading
        if Obstacle_Ahead == 0 && Obstacle_Ahead_Left==0 &&
Obstacle_Ahead_Right==0
             rewardVal=1;
        else
             rewardVal=-1;
        end
%       Sensor_Feedback=
[Obstacle_Right,Obstacle_Right_Behind,Obstacle_Behind,Obstacle_L
eft_Behind,Obstacle_Left,Obstacle_Ahead_Left,Obstacle_Ahead,Obst
acle_Ahead_Right];
        Global_Directions=[90 135 180 225 270 315 0 45];

        %Observation of the Environment for Reward Value
        if Obstacle_Right==1 %1

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+90),360)))
=Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_
Directions(:)==mod(round(currentDirection+90),360))) -1;
        elseif Obstacle_Right==0 %1

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+90),360)))
=Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_
Directions(:)==mod(round(currentDirection+90),360)))+ 0;
        end
        if Obstacle_Right_Behind==1 %2

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+135),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+135),360)))-1;
        elseif Obstacle_Right_Behind==0 %2

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+135),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+135),360)))+0;
        end
        if Obstacle_Behind==1 %3

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+180),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+180),360)))-1;
        elseif Obstacle_Behind==0 %3

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+180),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+180),360)))+0;
```

```
        end

        if Obstacle_Left_Behind==1 %4

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+225),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+225),360)))-1;
        elseif Obstacle_Left_Behind==0 %4

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+225),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+225),360)))+0;
        end
        if Obstacle_Left==1 %5

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+270),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+270),360)))-1;
        elseif Obstacle_Left==0 %5

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+270),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+270),360)))+0;
        end
        if Obstacle_Ahead_Left==1 %6

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+315),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+315),360)))-1;
        elseif Obstacle_Ahead_Left==0 %6

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+315),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+315),360)))+0;
        end
        if Obstacle_Ahead == 1 %7

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection),360)))-1;
        elseif Obstacle_Ahead == 0 %7

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection),360)))+0;
        end
```

```
        if Obstacle_Ahead_Right==1 %8

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+45),360)))
=Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_
Directions(:)==mod(round(currentDirection+45),360))) -1;
        elseif Obstacle_Ahead_Right==0 %8

Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+45),360))) =
Q(Initial_Y_Row_Position,Initial_X_Column_Position,find(Global_D
irections(:)==mod(round(currentDirection+45),360)))+0;
        end

        %Action Selection
%         Global_Directions=[90 135 180 225 270 315 0 45];
        [val,index] =
max(Q(Initial_Y_Row_Position,Initial_X_Column_Position,:));
        [xx,yy] =
find(Q(Initial_Y_Row_Position,Initial_X_Column_Position,:) ==
val);
        if size(yy,1) > 1
            index = 1+round(rand*(size(yy,1)-1));
            action = yy(index,1);
            Decision=1;
        else
            action = index;
            Decision=2;
        end

        %Action definition in terms of theta
        setpoint=Global_Directions(action);

        % update information for robot in Q for later use
        Q(Hold_Y_Row_Position,Hold_X_Column_Position,action) =
Q(Hold_Y_Row_Position,Hold_X_Column_Position,action) +
alpha*(rewardVal+gamma*max(Q(Initial_Y_Row_Position,Initial_X_Co
lumn_Position,:)) -
Q(Hold_Y_Row_Position,Hold_X_Column_Position,action));
        Q1=Q(:,:,1);
        Q2=Q(:,:,2);
        Q3=Q(:,:,3);
        Q4=Q(:,:,4);
        Q5=Q(:,:,5);
        Q6=Q(:,:,6);
        Q7=Q(:,:,7);
        Q8=Q(:,:,8);
        %         Global_Directions=[90 135 180 225 270 315 0
45];

direction_Code=find(Global_Directions(:)==mod(round(currentDirec
tion),360));
```

```
setpoint_Code=find(Global_Directions(:)==mod(round(setpoint),360
));

        %Motion request
        if round(mod(currentDirection,360))==round(setpoint)
            DesiredLinearSpeedLeft=1;
            DesiredLinearSpeedRight=1;
            time=1;
            time_total=time_total+time;
        elseif setpoint_Code==8 && (direction_Code==1 ||
direction_Code==2 || direction_Code==3||direction_Code==4)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
             time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
             time_total=time_total+time;
        elseif setpoint_Code==7 && (direction_Code==8 ||
direction_Code==1 || direction_Code==2||direction_Code==3)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
             time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
             time_total=time_total+time;
        elseif setpoint_Code==6 && (direction_Code==7 ||
direction_Code==8 || direction_Code==1||direction_Code==2)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
             time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
             time_total=time_total+time;
        elseif setpoint_Code==5 && (direction_Code==6 ||
direction_Code==7 || direction_Code==8||direction_Code==1)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
             time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
             time_total=time_total+time;
        elseif setpoint_Code==4 && (direction_Code==5 ||
direction_Code==6 || direction_Code==7||direction_Code==8)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
             time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
             time_total=time_total+time;
        elseif setpoint_Code==3 && (direction_Code==4 ||
direction_Code==5 || direction_Code==6||direction_Code==7)
             DesiredLinearSpeedLeft=0;
             DesiredLinearSpeedRight=1;
```

```
            time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==2 && (direction_Code==3 ||
direction_Code==4 || direction_Code==5||direction_Code==6)
            DesiredLinearSpeedLeft=0;
            DesiredLinearSpeedRight=1;
            time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==1 && (direction_Code==2 ||
direction_Code==3 || direction_Code==4||direction_Code==5)
            DesiredLinearSpeedLeft=0;
            DesiredLinearSpeedRight=1;
            time =mod((direction_Code-
setpoint_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==8 && (direction_Code==7 ||
direction_Code==6 || direction_Code==5)
            DesiredLinearSpeedLeft=1;
            DesiredLinearSpeedRight=0;
            time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==7 && (direction_Code==6 ||
direction_Code==5 || direction_Code==4)
            DesiredLinearSpeedLeft=1;
            DesiredLinearSpeedRight=0;
            time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==6 && (direction_Code==5 ||
direction_Code==4 || direction_Code==3)
            DesiredLinearSpeedLeft=1;
            DesiredLinearSpeedRight=0;
            time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==5 && (direction_Code==4 ||
direction_Code==3 || direction_Code==2)
            DesiredLinearSpeedLeft=1;
            DesiredLinearSpeedRight=0;
            time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
            time_total=time_total+time;
        elseif setpoint_Code==4 && (direction_Code==3 ||
direction_Code==2 || direction_Code==1)
```

```
                DesiredLinearSpeedLeft=1;
                DesiredLinearSpeedRight=0;
                time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
                time_total=time_total+time;
          elseif setpoint_Code==3 && (direction_Code==2 ||
direction_Code==1 || direction_Code==8)
                DesiredLinearSpeedLeft=1;
                DesiredLinearSpeedRight=0;
                time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
                time_total=time_total+time;
          elseif setpoint_Code==2 && (direction_Code==1 ||
direction_Code==8 || direction_Code==7)
                DesiredLinearSpeedLeft=1;
                DesiredLinearSpeedRight=0;
                time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
                time_total=time_total+time;
          elseif setpoint_Code==1 && (direction_Code==8 ||
direction_Code==7 || direction_Code==6)
                DesiredLinearSpeedLeft=1;
                DesiredLinearSpeedRight=0;
                time =mod((setpoint_Code-
direction_Code),8)*45*0.043633235; %%1 derece cevirmek icin
gerekli sure
                time_total=time_total+time;
          end

          %Saving actions and status for later plottting
          setpoint_vector(Count_Step)=setpoint;
          setpoint_Code_vector(Count_Step)=setpoint_Code;
          currentDirection_vector(Count_Step)=currentDirection;
          direction_Code_vector(Count_Step)=direction_Code;
          theta_vector(Count_Step)=currentDirection;
          theta_deg_vector(Count_Step)=currentDirection*180/pi;

position_row_y_vector(Count_Step)=Initial_Y_Row_Position;

position_column_x_vector(Count_Step)=Initial_X_Column_Position;
          Obstacle_Ahead_vector(Count_Step)=Obstacle_Ahead;
          Obstacle_Left_vector(Count_Step)=Obstacle_Left;

Obstacle_Left_Ahead_vector(Count_Step)=Obstacle_Ahead_Left;
          Obstacle_Right_vector(Count_Step)=Obstacle_Right;

Obstacle_Right_Ahead_vector(Count_Step)=Obstacle_Ahead_Right;
          Obstacle_Behind_vector(Count_Step)=Obstacle_Behind;

Obstacle_Left_Behind_vector(Count_Step)=Obstacle_Left_Behind;
```

```
Obstacle_Right_Behind_vector(Count_Step)=Obstacle_Right_Behind;

StepNumber_Decision_Matrix(Count_Step+1,:)=[Count_Step,Decision]
;

EpisodeNumber_StepNumber_FinishFlag_Position_Action=[Count_Episo
de,Count_Step,first_finish_flag,Initial_Y_Row_Position,Initial_X
_Column_Position,action]

History=[Count_Episode,Count_Step,Decision,direction_Code];
        if Decision==1
            Decision_1_Counter=Decision_1_Counter+1;
        elseif Decision==2
            Decision_2_Counter=Decision_2_Counter+1;
        end
        end
        iterationCount(k,1) = Count_Step;
        TimeStatistic(k,1) = time_total;
        Decision_1_Statistic(k,1)=Decision_1_Counter;
        Decision_2_Statistic(k,1)=Decision_2_Counter;
        Distance_To_Goal(k,1)= sqrt((1-
Initial_Y_Row_Position)*(1-Initial_Y_Row_Position)+
(size(Maze_Matrix_Total,2)/2-
Initial_X_Column_Position)*(size(Maze_Matrix_Total,2)/2-
Initial_X_Column_Position));
        if first_finish_flag == 1
            Q1_First_Finish=Q1;
            Q2_First_Finish=Q2;
            Q3_First_Finish=Q3;
            Q4_First_Finish=Q4;
            Q5_First_Finish=Q5;
            Q6_First_Finish=Q6;
            Q7_First_Finish=Q7;
            Q8_First_Finish=Q8;

position_row_y_vector_First_Finish=position_row_y_vector;

position_column_x_vector_First_Finish=position_column_x_vector;
        end
end

%Place the grayscale matrix on coordinate plane
figure(1)
movegui('northwest')
[grows, gcolumns, gnumberOfColorChannels] =
size(Maze_GrayScale_Image);
imshow(Maze_GrayScale_Image);
title('Robot in Maze')
xlabel('x-axis')
ylabel('y-axis')
axis on;
hp = impixelinfo;
grid on;
```

```
%Define the robot in birds-eye, place it in its initial position
and scale it:
x_shape_scale = [-20 , -20   , -28   , -28 , -20 , -20 ,  0 , 20
,  20 ,  28 ,  28   , 20    ,  20 , 0   ]/4;
% y_shape_scale = [-24 , -12.5 , -12.5 , 2.5 , 2.5 ,  24 , 24 ,
24 , 2.5 , 2.5 , -12.5 , -12.5 , -24 , -24 ]/4;
y_shape_scale = [ 0  , 11.5  , 11.5  , 26.5, 26.5,  48 , 48 , 48
, 26.5, 26.5,  11.5 , 11.5  ,  0   , 0   ]/4;
for i=1:Count_Step

    Rotate=makehgtform('zrotate',theta_vector(i));

Translate=makehgtform('translate',[position_column_x_vector(i)
position_row_y_vector(i) 0]);
    HrectTR=Translate*Rotate;

AxesTransformation=hgtransform('Parent',gca,'matrix',HrectTR);
    Robot_Patch=patch('XData',x_shape_scale,'YData',-
y_shape_scale,'FaceColor','green','Parent',AxesTransformation);
    set(Robot_Patch);
    drawnow
    hold on
end

figure(2)
movegui('southwest')
hold on
plot(setpoint_Code_vector,'-x','MarkerIndices',1:1:Count_Step)
hold on
plot(direction_Code_vector,'-x','MarkerIndices',1:1:Count_Step)
hold on
for i=1:1:Count_Step-2
error(i)=setpoint_vector(i)-currentDirection_vector(i+1);
end
plot(error)
% plot(setpoint_currentDirection_dif_vector,'-
x','MarkerIndices',1:1:Count_Step)
legend('Set Point','Current Direction','error')
display(1:1:Count_Step)

figure(3)
movegui('southeast')
hold on
plot(position_column_x_vector,'-
x','MarkerIndices',1:1:Count_Step)
hold on
plot(position_row_y_vector,'-x','MarkerIndices',1:1:Count_Step)
xlabel('x-axis')
ylabel('y-axis')
legend('Robot X-Column Position on Matrix', 'Robot Y-Row
Position on Matrix')
```

Publications from Thesis

Contact Information: dogukan.engin@gmail.com

Conference Papers

1. D. Engin, A. Yeşildirek. Control of Autonomous Mobile Robots Using Reinforcement Learning. Ereğli International Science and Academic Congress, 2019, Konya, Turkey