

**SYSTEM DESIGN AND PROTOTYPE DEVELOPMENT OF A
GUIDE – GUARD ROBOT
FOR THE DEPARTMENT OF MECHATRONICS ENGINEERING
LABORATORIES**

A MASTER’S THESIS

in

Mechatronics Engineering

Atilim University

by

AMIR NOBAHAR SADEGHI

JULY 2015

**SYSTEM DESIGN AND PROTOTYPE DEVELOPMENT OF A
GUIDE – GUARD ROBOT
FOR THE DEPARTMENT OF MECHATRONICS ENGINEERING
LABORATORIES**

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY
BY
AMIR NOBAHAR SADEGHI**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN**

**THE DEPARTMENT OF MECHATRONICS ENGINEERING
JULY 2015**

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

Prof. Dr. K. Ibrahim AKMAN

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Abdülkadir ERDEN

Head of Department

This is to certify that we have read the thesis “System Design and Prototype Development of a Guide – Guard Robot for the Department of Mechatronics Engineering Laboratories” submitted by “Amir Nobahar Sadeghi” and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Zühal ERDEN

Co-Supervisor

Prof. Dr. Abdülkadir ERDEN

Supervisor

Examining Committee Members:

Asst. Prof. Dr. Yiğit Taşcıoğlu

Asst. Prof. Dr. Hakan Tora

Asst. Prof. Dr. Kutluk Bilge Arıkan

Asst. Prof. Dr. Zühal Erden

Prof. Dr. Abdulkadir Erden

Date: 13.07.2015

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Amir Nobahar Sadeghi

ABSTRACT

SYSTEM DESIGN AND PROTOTYPE DEVELOPMENT OF A

GUIDE – GUARD ROBOT

FOR THE DEPARTMENT OF MECHATRONICS ENGINEERING

LABORATORIES

Nobahar Sadeghi, Amir

M.S., Mechatronics Engineering Department

Supervisor: Prof. Dr. Abdulkadir ERDEN

Co-Supervisor Asst. Prof. Dr. Zuhale ERDEN

July 2015, 146 pages

The goal of this thesis is designing, constructing and manufacturing an autonomous, intelligent, mobile guide - guard robot and enable it to safely navigate through the indoor environments such as the laboratories of the department of Mechatronics Engineering in Atılım University. Based on the main concern of the thesis, an integrated system was designed and implemented.

To accomplish the navigation task, our presented control architecture composes of localization, obstacle avoidance, path planning and robot control for steering the robot from any initial pose to arbitrary assigned pose. Our navigation architecture is a behavior-based and a mapless navigation, which the robot's locations are determined by observing and extracting useful features in the environment based on the onboard sensors.

Keywords: Guide – Guard Robot, Behavior-Based Navigation, Map-Based and Non-Map-Based Navigation, Control Architecture.

ÖZ

MEKATRONİK MÜHENDİSLİĞİ BÖLÜM LABORATUVARLARI İÇİN

REHBER – BEKCI ROBOT

SYSTEM TASARIMI VE PROTOTİP GELİŞTİRİLMESİ

Nobahar Sadeghi, Amir

Yüksek Lisans, Mekatronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Abdülkadir ERDEN

Ortak Tez Yöneticisi: Yrd. Doç. Dr. Zühal ERDEN

Temmuz 2015, 146 sayfa

Bu tez çalışmasının amacı Atılım Üniversitesi Mekatronik Mühendisliği laboratuvarlarında “Rehber-Bekçi Robot” olarak görev yapacak olan akıllı bir hareketli robotun tasarlanması, üretilmesi ve laboratuvar ortamında dolaşmasının sağlanmasıdır. Bu amacı gerçekleştirmek üzere tümleşik bir sistem tasarlanmıştır. Robotun laboratuvarlarda dolaşması için geliştirilen kontrol mimarisi konumlandırma, engelden kaçınma, yol planlaması ve verilen bir başlangıç pozisyonundan rastgele belirlenen bir başka pozisyona gidişin yönetilmesini sağlamaktadır. Bu çalışmada, robota yüklenmiş bir haritaya dayalı olmayan davranış-temelli bir navigasyon algoritması kullanılmıştır. Buna göre robot, içinde bulunduğu laboratuvar ortamının çeşitli özelliklerini, üzerindeki sensörler yardımıyla algılayarak kendi konumunu belirlemekte ve böylece laboratuvar ortamında dolaşmaktadır. Bu çalışma kapsamında robotun tasarımı, imalatı, sistem entegrasyonu gerçekleştirilmiş ve laboratuvar ortamında dolaşması test edilmiştir.

Anahtar Sözcükler: Rehber-Bekçi Robot, Davranış-Temelli Navigasyon, Harita-Temelli ve Harita-Temelli Olmayan Navigasyon, Kontrol Mimarisi

To My Parents, Wife & lovely Son
for their Spiritual and Financial Supports

ACKNOWLEDGMENTS

I express sincere appreciation to my supervisor Prof. Dr. Abdülkadir Erden for his guidance and insight throughout the research. Thanks also go to my cosupervisor Asst. Prof. Dr. Zühal Erden. The technical and mental assistance of Meral Aday, Handan Kara and Cahit Gürel is gratefully acknowledged. Thanks also go to technicians in the machine shop, specially Mehmet Çakmak. To my parents, my wife, and my lovely son, Arian, I offer sincere thanks for their continuous support and patience during this period.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	v
DEDICATION	vi
ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xii
LIST OF FIGURES	xiii
ABBREVIATIONS	xix
NOMENCLATURE.....	xxi
CHAPTER	
1. INTRODUCTION	1
2. LITERATURE SURVEY	7
3. SYSTEM DESIGN	22
3.1 System Structure	22
3.1.1 Body	23
3.1.1.1 Chassis Framework	23
3.1.1.2 Floors, Walls, Sensors Mount.....	24
3.1.2 Wheels.....	24
3.1.3 Motors	27
3.1.4 IR Distance Sensors	28
3.1.5 Ultrasonic Distance Sensors	30
3.1.6 Compass.....	30
3.1.7 Micro-Controller Board	31
3.1.8 Motor Driver Shield	32

3.1.9 Voltage Regulator	33
3.1.10 Handmade Voltage and Data Distributor Board.....	33
3.1.11 Navigation Computer	34
3.1.12 Touch-Screen Notebook	34
3.1.13 Battery	35
3.2 Control Architecture	35
3.2.1 Control of Mobile Robot.....	35
3.2.2 Dynamics of Mobile Robot.....	35
3.2.3 Hard Switches vs. Blending.....	39
3.2.4 Hybrid Automata	40
3.2.5 Navigation using Behaviors.....	40
3.2.5.1 Go-to-Goal Behavior	40
3.2.5.2 Avoid-Obstacle Behavior.....	41
3.2.5.3 Blending	43
3.2.5.4 Wall-Follow Behavior.....	43
3.2.5.5 Hybrid Automata	44
3.2.6 Layered Architecture	45
4. SYSTEM IMPLEMENTATION	49
4.1 Electrical Hardware Assembling.....	49
4.1.1 Voltage Supplying	49
4.1.2 Data and Power Lines Wiring.....	50
4.2 Data Transmission.....	51
4.2.1 IR Range Sensors	51
4.2.2 Ultrasonic Range Sensors	54
4.2.3 Compass	55
4.2.4 Differential Wheel Drive	56

4.2.5 Motor Quadrotor Encoders	57
4.3 Microcontroller Board-MATLAB Simulink Connection	58
4.4 Control and Model of the System	59
4.5 Plant Model	59
4.5.1 Data Acquisition	60
4.5.2 System Identification	61
4.5.3 Controller Design.....	63
4.5.4 Real-Time Testing	65
4.6 Architecture of Differential-Drive Trackers	67
4.7 Odometry	69
4.8 Architecture of Differential-Drive Planners.....	69
4.8.1 Go-to-Goal Behavior	69
4.8.2 Avoid-Obstacles Behavior	70
4.8.3 Blending and Hard Switches.....	72
4.8.4 Follow-Wall Behavior.....	75
4.8.5 All Behaviors in a Single Navigation System	77
4.9 Motor Limitations	79
4.10 Graphical User Interface	80
4.10.1 Design	80
4.10.2 Bridging GUI and Microcontroller	81
5. EXPERIMENT AND RESULTS	83
5.1 Results of the Simulated System.....	83
5.2 Results of the Actual system with No-Load	84
5.3 Results of the Actual system in the Field.....	87
5.3.1 Wheels Situ Spin Controller	88
5.3.2 Straight - Move Controller	88

6. DISCUSSION AND CONCLUSIONS.....	97
References.....	99
Appendices.....	103

LIST OF TABLES

Table 1: Coordinates of the Robot, Intermediary & Final Goal Points	48
Table 2: Analog / Digital Connection Lines between Parts & Microcontroller Boards	50
Table 3: Analog / Digital Connection Lines between MMCB and DS	51
Table 4: A Lookup Table for Interpolating a Distance from the Analog and Digital Output Voltages	53
Table 5: The Results of Plant Model using by Transfer Function, Nonlinear ARX and Hammer-Wiener type Estimators	62
Table 6: Topics of Implemented Tests with Various Conditions in the Field.....	92

LIST OF FIGURES

Figure 1: The Functions of the Guide Robot	22
Figure 2: Physical Structure of the Robot	23
Figure 3: Designed & Built Chassis for the Robot	24
Figure 4: Rear Powered Track-wheels and Front Omni-Wheels.....	25
Figure 5: Data-Transmission Flow between the Subsystems of the Hardware.....	26
Figure 6: Components of a DC Servo Motor.....	27
Figure 7: Used DC Servo Motor and its Specifications.....	28
Figure 8: The Motor's Mount.....	28
Figure 9: Schematic Diagram of the IR Sensors and a Real Prototype.....	29
Figure 10: Used Ultrasonic Sensor and its Specifications	30
Figure 11: Used Compass and its Specifications	31
Figure 12: Arrangement of the IR, Ultrasonic Sensors Motors & Wheels	31
Figure 13: Microcontroller Board	32
Figure 14: Motor Driver Schield	33
Figure 15: Voltage Regulator Board	33

Figure 16: Handmade Distributor Board	34
Figure 17: Touch-Screen Notebook	34
Figure 18: The Battery.....	35
Figure 19: Response by a P-Regulator.....	36
Figure 20: Response by a PI-Regulator	36
Figure 21: Response by a PID-Regulator	37
Figure 22: Taking the Robot to Drive from two Points	37
Figure 23: Heading and Desired Heading	38
Figure 24: Desired Heading in Go-to-Goal Behavior	38
Figure 25: Alternative Desired Headings in Obstacle Avoidance Behavior	39
Figure 26: Control Input Vector	40
Figure 27: Control Input Vector in Go-to-Goal Behavior	41
Figure 28: Control Input Vector vs Error in Go-to-Goal Behavior	41
Figure 29: Control Input Vector in Avoid-Obstacle Behavior	41
Figure 30: Control Input Vector vs Error in Avoid-Obstacle Behavior	42
Figure 31: Control Input Vector of Go-to-Goal and Avoid-Obstacle Behaviors	42

Figure 32: Hybrid Automata	42
Figure 33: Control Input Vector in Blended Behavior	43
Figure 34: Control Input Vector in Fallow-Wall Behavior	43
Figure 35: Wall Fallowing in two different Directions	44
Figure 36: All Behaviors in single Hybrid Automata	45
Figure 37: Intermediary & Final Goal Points on the Map Platform	46
Figure 38: Schematic of the Control Architecture	47
Figure 39: Distance Measuring Characteristics of (a) 10-80 cm (b) 20-150 cm types of IR Range Sensors	52
Figure 40: Timing Diagram of Trigger, Burst & Echo Pulses of Ultrasonic Sensor.....	54
Figure 41: Subsystem Model to obtain the Range from the Echo Signal	55
Figure 42: The Axes of the Compass Board.....	56
Figure 43: MATLAB Simulink and Aruino Mega 2560.....	59
Figure 44: Workflow of the Data-Driven Modeling	60
Figure 45: Data Acquisition Hardware Setup	60
Figure 46: Time Plot of Input and Output Signals	61

Figure 47: Measured and Simulated Model Output Time Signals of (a) Transfer Function 1 (b) Nonlinear H-W 1 Estimators, and Step Response of (c) First Estimator (d) Second Estimator	62
Figure 48: Step Responses of the Two Models in a Open-Loop System	64
Figure 49: Root-Locus of the Open-Loop System	64
Figure 50: Close-Loop System using by Discrete PID	64
Figure 51: Step Responses of the two Models in a Close-Loop System	65
Figure 52: Root-Locus of the Close-Loop System	65
Figure 53: Step Responses of the Real Motor in a Close-Loop System with (a) Design Phase's PID Parameters (b) Mentioned PID Parameters	66
Figure 54: Step Responses of the Real Motor in a Open-Loop System in (a) No-Load (b) Under-Load Conditions	66
Figure 55: Primary Differential-Drive Architecture	67
Figure 56: Picking a New Point	67
Figure 57: Control Input Vector of New Point	68
Figure 58: Final Differential-Drive Architecture	68
Figure 59: Go-to-Goal Vector	69
Figure 60: Avoid-Obstacles Vector	71

Figure 61: Blended Vector	73
Figure 62: Two Vectors in FW Behavior	76
Figure 63: A Screenshot of the Graphical User Interface on the Notebook	81
Figure 64: Data Flow in Hardware Setup from GUI to the Controllers	82
Figure 65: (a) Time Signals of Position & Heading (b) Traversed Trajectory of the Robot to reach the Goal Point at (100, 0) and come back to the Initial Point in the Simulated System	84
Figure 66: (a) Time Signals of Position & Heading (b) Traversed Trajectory of the Robot to reach the Goal Point at (100, 0) and come back to the Initial Point in the Actual System	85
Figure 67: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Point at (469, 1261) through the Intermediary Way Point at (469, 552) in the Actual System.....	86
Figure 68: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Point at (2000, 0) in the Actual System.....	87
Figure 69: New Rear Powered Wheels	88
Figure 70: New Front Omni-Wheels.....	90
Figure 71: New Motor's Mount	90

Figure 72: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Points based on Table 6 in the Actual System96

LIST OF ABBREVIATIONS

AGV	-	Automated Guided Vehicle
UGV	-	Unmanned Ground Vehicle
UAV	-	Unmanned Aerial Vehicle
AUV	-	Autonomous Underwater Vehicle
IR	-	InfraRed
CCD	-	Charge Coupled Device
DC	-	Direct Current
PSD	-	Position Sensitive Detector
IREDD	-	Infrared Emitting Diode
AHRS	-	Attitude and Heading Reference System
IMU	-	Inertial Measurement Unit
PWM	-	Pulse Width Modulation
PID	-	Proportional Integral Derivative
GTG	-	Go To Goal
AO	-	Avoid Obstacle
FW	-	Fallow Wall
AI	-	Artificial Intelligence
GUI	-	Graphical User Interface

IRS - Infrared Sensor

US - Ultrasonic Sensor

CP - Compass

E - Encoder

M - Motor

MMCB - Master Microcontroller Board

SMCB - Slave Microcontroller Board

DS - Driver Shield

DB - Distributor Board

A - Analog

D - Digital

ADC - Analog to Digital Convertor

CPR - Count Per Revolution

TPR - Tick Per Revolution

CPS - Count Per Second

QEI - Quadrature Encoder Interface

I/O - Input/Output

FSM - Finite State Machine

IPS - Indoor Positioning System

NOMENCLATURE

F - Force

u - Control Input Vector

u_x - Control Input in x Direction

u_y - Control Input in y Direction

x - Posiotion of the Robot in x Direction

y - Posiotion of the Robot in y Direction

x_g - Posiotion of the Goal Point in x Direction

y_g - Posiotion of the Goal Point in y Direction

x_o - Posiotion of the Obstacle Point in x Direction

y_o - Posiotion of the Obstacle Point in y Direction

$x_{initial}$ - Posiotion of the Initial Point in x Direction

$y_{initial}$ - Posiotion of the Initial Point in y Direction

φ - Heading of the Robot

φ_d - Desired Heading

φ_{goal} - Heading of the Goal Point

φ_{obst} - Heading of the Obstacle Point

v - Linear Velocity of the Robot

v_r - Angular Velocity of Right Motor

v_l - Angular Velocity of Left Motor

ω - Angular Velocity of the Robot

M - Mass of the Robot

R - Radius of the Wheels

L - Distance between the Wheels

g - Gravitational Constant

Γ - Torque

\Re - Real Numbers

u_{GTG} - Control Input Vector in Go-to-Goal Behavior

u_{AO} - Control Input Vector in Avoid-Obstacle Behavior

u_{AO-GTG} - Control Input Vector in Blended Behavior

u_{FW} - Control Input Vector in Follow-Wall Behavior

u^c_{FW} - Control Input Vector in Follow-Wall Behavior in Clockwise

u^{cc}_{FW} - Control Input Vector in Follow-Wall Behavior in Counter-Clockwise

t - Time

t_r - Rising Time

τ - Time of Last Switch

t_t - Trigger Pulse Width

t_e - Echo Pulse Width

Δ - Specified Distance

V - Voltage

CHAPTER 1

INTRODUCTION

Recently, since the quality of life is improving constantly in advanced countries, automation techniques are requested and developed widely in many fields. Among them, to design assistant robots for service people has become an emergent field that the governments of various countries and research institutions invest actively particularly. Generally speaking, assistant or service robots are mostly autonomous robots, which does not usually emphasize the accuracy and performance of operation in industrial robot, always focuses on giving assistance and providing a comfortable service. These mobile robots are evolving to serve and assist people at office, supermarket, museum, public place and so on. Whatever the context is, one of the basic requirements of such future robots is to guide a person from one place to another.

In general, all of the existing guide robots have two major functional modules, namely navigation and interaction. However, how to implement these two modules differs from one guide robot to another. Good navigation methods normally require accurate localization and reliable obstacle avoidance. However interactive tour guide robots need the ‘object approaching’ behaviors to serve visitors instead of avoiding them. Most of guide robots being deployed so far have no such a function.

Visual based indoor navigation can be categorized as map-based and non-map-based navigation. In map-based navigation, the robot has a known map being built manually before its operation, and locates itself by matching the current sensor data with the map data. Global localization must construct a match between the observations and the entire database. The uncertainties and the ambiguities are

main problems. Incremental localization algorithms use the information of initial position and odometers, and reduce the uncertainties by matching the current observations and expect observations. In mapless navigation, the robot has no map being built and its locations are determined by observing and extracting useful features in the environment based on the onboard sensors.

In recent years, much research has been paid to design and implementation of guide robots, in order to provide them with some innovative functions, features and appearances. Thanks to those efforts, guide robot have become more popular at various places or other important commercial areas. In addition to attracting attention, guide robots can guide visitors who trying to get to a certain place and explain them so that they can acquire knowledge of there and enjoy visiting. From the system design of view, there are two critical research issues inside these world wide guide robots. One is concerning with how to achieve safe navigation in dynamic, cluttered and crowded public environments, and the other is regarding how to design effective and interesting human-robot interactions between the visitor and the robot. Navigation in large-scale, dynamic, cluttered and crowded public environments without collision is a very fundamental but challenging task for guide robots. To accomplish the task, researchers have presented many complete integrated navigation sytems, composed of mapping, localization, robot security, obstacle avoidance, path planning and robot control for steering the guide robot from any initial pose to arbitrary desire pose.

This section is started with description of some basic concepts like robot and robotics, and then conducted to the concept of the guide robot, one important branch of intelligent service robots, and finally explained the navigation problem of guide robots, that is the main concern of the thesis.

The word “robot” is used to refer to a wide range of machines, the common feature is that they are all capable of movement and can be used to perform physical tasks. The word robot was first introduced by a Czech dramatist, *Karel Capek* in his 1921 play “*Rossum's Universal Robots*” [1]. He was referring to a perfect and tireless worker performing manual labor jobs for human beings. Then famous science fiction

writer Isaac Asimov coined the word “robotics” as the science of the study of robots in his science fiction stories about robots in 1940s [2]. In Webster's New World Dictionary, robotics is defined as “the science or technology of robots, their design, manufacture, application, use etc” [3].

In order to fulfill the desired tasks independently and automatically by a robot, autonomous robots have been raised. Autonomous robots are robots which can perform desired tasks in unstructured environments without continuous human guidance. Before the autonomous robots were invented, there were only ordinary robots. Those robots were all depending on human control. Besides, those robots also did not have any self avoiding systems toward obstacles as their avoiding systems were totally controlled by human. But due to the lack of technologies at that time, the circuits of those autonomous robots were complexes directly increased the cost of the robot. Those autonomous robots were invented to replace human in doing hazardous works such as denoting bomb and exploring unknown places. Lately, autonomous robots were also utilized as guides to blind man.

Autonomous robots increasingly have the potential to interact with people in daily life. It is believed that, based on this ability, they will play an essential role in human society in the not-so-distant future. Intelligent service robots for guiding, public building service, personal entertainment, military service and etc. have been developed and the limit of applications are widely spreading.

From 1990 onwards the intelligent service robots were focused on development of navigation system including map building, obstacle avoidance and so on. Because performance of navigation system has been improved based on improvement of sensors and also algorithms, nowadays development of indoor guide robot has been focused on integration with navigation and other high intelligent system as voice communication, face recognition, etc.

A mobile robot is an autonomous robot that is capable of movement in any given environment. Mobile robots have the capability to move around in their environment and are not fixed to one physical location.

An automated or automatic guided vehicle (AGV) is a mobile robot that follows markers or wires in the floor, or uses vision or lasers. They are most often used in industrial applications to move materials around a manufacturing facility or a warehouse. Application of the automatic guided vehicle has broadened during the late 20th century.

Mobile robots are a major focus of current research and almost every major university has one or more labs that focus on mobile robot research. Mobile robots are also found in industry, military and security environments. Domestic robots are consumer products, including entertainment robots and those that perform certain household tasks such as vacuuming or gardening.

Mobile robots may be classified by:

1. The environment in which they travel:

- Land or home robots are usually referred to as Unmanned Ground Vehicles (*UGVs*). They are most commonly wheeled or tracked, but also include legged robots with two or more legs (Humanoid, or resembling animals or insects).
- Aerial robots are usually referred to as Unmanned Aerial Vehicles (*UAVs*).
- Underwater robots are usually called autonomous underwater vehicles (*AUVs*).
- Polar robots, designed to navigate icy, crevasse filled environments.

2. The device they use to move, mainly:

- Legged robot : human-like legs or animal-like legs.
- Wheeled robot.
- Tracks

In the following the problem of robot navigation is introduced comprehensively. Moving from one place to another is an obvious task for humans. One decides how to move in a split second. For a robot such an elementary and basic task is a major

challenge. In autonomous robotics motion planning is one of the most significant challenges. There is a fundamental need to specify a task in a high-level language that is automatically translated into low-level descriptions of how the robot should move. The typical problem is to find a motion for a robot, whether it is a vacuum cleaning robot, a robotic arm, a flying object, or a guide robot from a starting position to a goal position whilst safely avoiding any obstacles in its way.

For any mobile device, the ability to navigate in its environment is important. Avoiding dangerous situations such as collisions and unsafe conditions (temperature, radiation, exposure to weather, etc.) comes first, but if the robot has a purpose that relates to specific places in the robot environment, it must find those places. Robot navigation means the robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location. In order to navigate in its environment, the robot or any other mobility device requires representation, i.e. a map of the environment and the ability to interpret that representation[4].

Navigation can be defined as the combination of the three fundamental competences:

1. Self-localization
2. Path planning
3. Map-building and map interpretation

Robot localization denotes the robot's ability to establish its own position and orientation within the frame of reference. Path planning is effectively an extension of localization, in that it requires determination of the robot's current position and a position of a goal location, both within the same frame of reference or coordinates. Map building can be in the shape of a metric map or any notation describing locations in the robot frame of reference.

Vision-Based Navigation uses optical sensors include IR proximity sensors, laser-based range finder and photometric cameras using *CCD* arrays to extract the visual features required to the localization in the surrounding environment. However, there are a range of techniques for navigation and localization using vision information, the main components of each technique are:

- Representations of the environment.
- Sensing models.
- Localization algorithms.

In order to give an overview of vision-based navigation and its techniques, these techniques have been classified under indoor navigation and outdoor navigation.

The easiest way of making a robot go to a goal location is simply to guide it to this location. This guidance can be done in different ways: burying an inductive loop or magnets in the floor, painting lines on the floor, or by placing beacons, markers, bar codes etc. in the environment. There are a very wider variety of indoor navigation systems [5].

An autonomously guided robot knows at least some information about where it is and how to reach various goals and or waypoints along the way. "Localization" or knowledge of its current location is calculated by one or more means, using sensors such as motor encoders, vision, lasers and global positioning systems. Positioning systems often use triangulation, relative position to determine the location and orientation of the platform, from which it can plan a path to its next waypoint or goal. It can gather sensor readings that are time- and location-stamped, so that a hospital, for instance, can know exactly when and where radiation levels exceeded permissible levels. Such robots are often part of the wireless enterprise network, interfaced with other sensing and control systems in the building.

With the introduction of these concepts, now we can have a better percipience of the goal of this thesis: design, construct and manufacture of an autonomous, intelligent, mobile, guide robot and enable it to navigate through the environments such as indoor shopping malls, airports and etc. The test and operation scenario will be implemented in the corridors of laboratories of Mechatronics Engineering Department in Atilim University as the work space of our robot.

CHAPTER 2

LITERATURE SURVEY

Mobile autonomous guide robots are gradually leaving the laboratories and entering the real world with extensive application fields at the current state of technology. Nowadays, one of the requirements in robotics is to develop partner robots for assisting people in a living or working environment. Such robots that can guide people to destinations while communicating appropriately with them have been extensively researched. Over the last decade, various types of autonomous guide robots have been introduced and their effectiveness assessed.

Mobile autonomous guide robots intended for large-scale environments pose a unique trans-disciplinary research. They require the integration of sensing, acting, planning and communicating within a single system.

The autonomous guide robot task can be decoupled in two main separate issues: navigation and interaction.

- **Navigation:** A limited number of researchers have demonstrated autonomous navigation in shopping malls, exhibitions or museums. There have been many related and successful guide robots that autonomously navigate with obstacle sensing capability in the indoor or outdoor environment. [9, 11-13, 16, 18-19, 21-35]
- **Interaction:** An important aspect of the guide robot is its interactive component. Human-robot interfaces are of great importance for robots that are to interact with ordinary people. Human-centered and social interactive robotics is a comparatively young field in mobile robotic

research. However some experiences where untrained people and robots meet each other are available. [9-10, 12-15, 17, 20, 24, 28, 37]

A number of survived literatures have been addressed the issue of design methodology and system architecture in autonomous robots. *Archila, John Faber, and Marcelo Becker* has presented design and mathematical models of an AGV as part of *FMS* (flexible manufacturing systems) [6]. Their literature proposes a mechatronics design methodology to the areas of mechanics, electronic and control using *CAD* and *CAE* tools. It is applicable to general mobile robots. The article starts with the kinematics and dynamics models, continues with the mechanical, electronic and control system development, and finally the assembly of robot and test results are shown. System architecture for intelligent building guide robot has presented in the other article [7]. This paper introduce a building guide robot, which has software components, including voice communication, face recognition, navigation, touch pad based *GUI*, database and hardware components including cameras, range sensors, micro-processors and power supplying system. For effective control, *MS* (Master Slave) method is employed for hardware system and *PSMR* (*PIRO* Service Mobile Robot) is used for all software components. To integrate complex hardware systems, it is necessary that well defined hardware architecture which is Master Slave architecture, including architecture of control modules and Sensors/Interfaces. To control resources effectively each software component is needed to be controlled creation, execution, deletion and pause. For this reason *PSMR* (*PIRO* Service Mobile Robot) based on *PBCM* (Port Based Component Model) is applied. Building a fully autonomous tour guide robot is another issue which some members of the *BlueBotics SA and Autonomous System Lab, EPFL* in Lausanne have presented in their article [8]. This paper presents the effort that has been undertaken in designing and building both hardware and software for a fully autonomous navigating vehicle for a tour guide application. The goal of the project is to maximize the autonomy and interactivity of the mobile platform while ensuring high robustness, reliability and performance. The robot task is decoupled to two separate issues: navigation and interaction, and then each issue are presented. *Sasai, Takuya, et al.* has developed a guide robot interacting with the user using information projection [9]. In their basic

system, the robot can detect multiple persons around it and choose one as a user. This is realized using an Omni directional camera and a laser range finder. The robot also has a projector with a pan-tilt mechanism. It can project information anywhere in the environment and guide a person. The user can input commands by simple gestures using the foot on the dialog box projected on the floor. After destination decision, the robot guides him/her by its motion and information projected onto the floor/wall/ceiling. Design methodologies are considered in order to develop voice-enabled interfaces for tour-guide robots to be deployed at the *Robotics Exposition of the Swiss National Exhibition* [10]. Human-robot voice communication presents new challenges for design of fully autonomous mobile robots, in that interactivity must be robot-initiated in conservation and within a dynamic adverse environment. The authors have approached these general problems for a voice enabled interface, tailored to limited computational resources of one on-board processor, when integrating smart speech signal acquisition, automatic speech recognition and synthesis, as well as dialogue system into the multi-modal, multi-sensor interface for the guide robot. They also focus on particular issues that need to be addressed in voice-based interaction when planning specific tasks and research experiments. Design methodology is addressed in a literature and applied on a security and patrol robot system [11]. The paper describes how to construct a security and patrol robot system based on modular and object oriented approach. Designing an autonomous robot requires the integration of many sensors and actuators on a physical base to give the robot the capacity to interact with their environment and fulfill their tasks. Even if sensors and actuators do impose limits on robot performance and autonomy, another critical factor is on-board processing capability required for controlling the robot. On-board processing imposes important influences on the electrical and structural requirements of the robot. The robots shape plays an important role in maintaining the public security. Theoretically, many shapes allow the robot to perform the tasks, but an appropriate shape is convenient to installing of the sensors and reducing the complexity of the whole control system. The authors have applied a novel controller running on real time windows operation system to the robot design. The robot equipped with stereovision and friendly man-robot interface can finish tasks such as navigation and maintaining the public security.

Development of the robots for various applications is the other major issue, which some literatures have presented. Eleven researchers from *Carnegie Mellon* and *Bonn* universities presented an interactive tour-guide robot, which was successfully exhibited in a *Smithsonian museum* [12]. During its two weeks of operation, the robot interacted with thousands of people, traveling more than 44 km distance, at speeds of up to 163 cm/sec. By use of this robot, the approach specifically addresses issues such as safe navigation in unmodified and dynamic environments, and short-term human-robot interaction. It uses learning pervasively at all levels of the software architecture. *Simmons, Reid, et al.* has presented *Xavier*, an autonomous indoor building guide robot which has been running an experiment in web-based interaction [13]. The robot, can accept commands to travel to different offices in the building, broadcasting camera images as it travels. Their article describes the autonomous robot system, the web-based interfaces, and how they communicate with the robot. It highlights lessons learned during this experiment in web-based robotics and includes recommendations for putting future mobile robots on the web. To explore possible robot tasks in daily life, a guide robot is developed for a shopping mall and conducted a field trial by Japanese researchers in *ATR Intelligent Robotics and Communication Laboratory* [14]. The robot was designed to interact naturally with customers and to affectively provide shopping information. It was also designed to repeatedly interact with people to build a rapport; since a shopping mall is a place people repeatedly visit, it provides the chance to explicitly design a robot for multiple interactions. For this capability, *RFID* tags have been used for person identification. The robot is semi-autonomous, partially controlled by a human operator, to cope with the difficulty of speech recognition in a real environment and to handle unexpected situation. A project namely *NEEL*, an intelligent shopping guide, and its portal were envisaged to provide the authors with crucial insights into the commercialization of service robots [15]. They describe their system and propose an approach to develop an interactive conversational agent which can serve shopping needs of the visitors in a shopping mall. Some key objectives were set forth in their project; justify how a rich social interaction with a robot can increase retailer sales and also ensure buyer satisfaction, to find out how can a real intelligent agent enhance social networking for an intelligent virtual agent, collect data from field

trials to improve the recommendation engine the project web. In another study a complete image processing and analysis system is developed for a special purpose mobile robot [16]. The aim of this mobile robot is to recognize the open doors and move through the open door with narrow clearance. The output of image processing is used as input data file to a *NN* based pattern recognition software. Further process is to interpret the door status by using some knowledge based algorithms. If the door is open, next process determines position of the mobile robot with respect to the open door. This paper includes related image processing algorithms for image analysis stages. *Shieh, Ming-Yuan, et al.* propose design concepts of a vision-based shopping assistant robot which can serve people in a mall [17]. By using such robots, a mall could not only save human resources effectively but also improve the quality of information service. In general a mall always provides three kinds of customer services as guiding, communication and accompanying. In their paper, a fuzzy collision-free guiding controller and a fuzzy following controller are proposed for guiding and accompanying services respectively. For collision-free controls, a named ‘warning area’ will be preplanned in the image plane to check whether there are objectives and /or obstacles in front of the robot. Besides, the eigenblock of objectives could be determined by using color image processing methods. The color recognition and size filter are used to search for proper objective image blocks and their relative locations. Based on a fuzzy following controller, the robot will accomplish the mobility of following the specified customer. The development of a new outdoor tour guide robot, on the campus of *National Taiwan University* is presented by *Chiang, Kuo-Hung, et al* [18]. In order to fulfill reliability and safety of outdoor navigation, data acquired via several sensing technologies, such as differential global positioning system (*DGPS*), dead reckoning, and digital compass are fused by the way of the well known extended Kalman filtering (*EKF*) technique. Furthermore, a shortest path planning and obstacle avoidance algorithm is also implemented. For the latter, it is deployed twelve ultrasonic sensors around the body to detect the nearby object within three meters. Once the robot encounters an unexpected obstacle, the proposed hierarchical navigation strategy will drive the robot to avoid collision. The author has endeavored to construct an outdoor robot which can autonomously guide visitors around the campus and provide introduction

of the visited sites. Encouraging people to listen to a guide robot is an interesting issue which is presented by some members of *ATR Intelligent Robotics and Communication Laboratory* [19]. Tour guidance is a common task of social robots. Such a robot must be able to encourage the participation of people who are not directly interacting with it. The author is particularly interested in encouraging people to overhear its interaction with others, since it has often been observed that even people who hesitate to interact with a robot are willing to observe its activity. To encourage such participation as bystanders, they developed a robot that walks backwards based on observations of human tour guides. Their developed system uses a robust human tracking system that enables a robot to guide people by walking forward/backward and allows them to scrutinize people's behavior after the experiment. Another literature presents a framework for a mobile robot guide, which provides the human with the flexibility to decide upon the way he wants to be guided [20]. For a robot to behave socially, it should not expect that the human will always follow the exact trajectory of the robot or will always maintain a fixed distance with robot or always support the guiding. In this paper the authors present a framework of monitoring and adapting to the human commitment on the joint task, and carrying out appropriate and goal oriented re-engagement attempts, if required, from the view point of guiding. *NCCU Security Warrior* is the name of an intelligent security robot which has been developed by *National Chung Cheng University* [21]. The robot consists of six systems including vision, motion, robot arms, power estimation, remote supervise and sensory systems. The vision system is used to carry out human detection and tracking. The motion system is built by using embedded systems and used to achieve motion planning in real time. In order to avoid the robot shut down suddenly, the power estimation system has been employed. For the robot, the sensory system is one of the important parts. With the fire detection sensor, the security robot can detect fire alarm in the environment. Path planning is accomplished by using the multi-sensor fusion. Body sensors can detect intruders. Combining the sensory and remote supervising systems, the owner can get notice whether there are any situations occurred through the *PDA* and *GSM* module. Design of a remote control based hybrid-structure robot for home security applications is described by *Kuo, C. H., et al.* researchers from *Chang Gung University* in Taiwan [22]. It is presented a

hybrid-structure robot with humanoid and vehicle types to perform home security tasks. To achieve home security issues, the smoke and temperature detection sensors are mounted on the robot. At the same time, the *CCD* camera is mounted on the head to capture the guarded videos and to assist remote manipulations. The proposed hybrid-structure robot behaves vehicle type in most of operation time to perform stable and fast movements and to reduce energy consumptions. When the robot enters humpy grounds or crosses small doorsills, the robot structure is changed as humanoid type to pass the non-flat grounds. The development of a Multi-Sensor based intelligent security robot is designed and implemented in the above mentioned university [23]. The author describes in the paper, an intelligent multi-sensor based security robot that can detect abnormal and hazardous situation and notify. The function of security robot contains six parts, including software development system, obstacle avoidance and motion planning system, image system, sensor system, remote supervisory system. They use touch panel to display system state, and design a general user interface (*GUI*) on the robot. It can moves by *X* axis and rotates by *Z* axis, and guards the security robot easily to connect using sensor based method. It can detect fire event and intruder, and transmit the message of the detection result to the cellular phone using *GSM*, or to client computer through internet. Finally from point view of application, the above mentioned article [8], has presented tour guide application of a fully autonomous navigating vehicle.

The issue of navigation system in autonomous robots has been presented by some other literatures. Autonomous navigation of an indoor tour guide robot is described in an article presented by *Chinese Electrical Technology Research and Development and Association* [24]. This paper develops methodologies and techniques for autonomous navigation of a tour guide robot with a human robot interaction system. The designed navigation system includes global localization, dynamic path planning, local goal-seeking, safe obstacle avoidance, behavior fusion, and autonomous robot control. The *RFID* module for global pose initialization is presented based on the *RSSI* measurements, the proposed calibration method and least square method. With the gross fusion the *RFID* data and laser scanning measurements utilizing an extended Kalman filter (*EKF*). The global path is generated by dynamic

programming method. The *Petri-net* model is employed to construct the event-driven logic control sequence from the present states of the robot and assigned tour missions. *Mizobuchi, Yoshinobu, et al*, present trajectory planning method of guide robots for achieving the guidance [25]. In their paper, firstly, the advanced guidance knowledge is formulated using production rules based on linguistic variables. Secondly, the trajectory planning of guide robot is implemented by the quantified knowledge and Distance-Type Fuzzy Reasoning method. The Distance-Type Fuzzy Reasoning method is a fuzzy reasoning method by considering the distance value between two fuzzy sets, and is effective even when the common set between an antecedent and a fact is an empty set. Smooth and efficient obstacle avoidance for a tour guide robot is the subject of an interesting literature [26]. The paper presents an implementation of path planning and obstacle avoidance for an autonomous tour guide robot. In view of the authors a tour guide robot faces certain requirements. The collision risk must be low and the eventual effects of a collision must be harmless. Smooth motion is important, as visitors anticipate movement when they follow the guide. The obstacle avoidance control loop should be fast in order to not only run in real-time, but also leave enough processing resources to other modules such as localization, sensor acquisition, web server and motor control. Enabling mobile robots to navigate through crowded environments such as indoor shopping malls, airports, or downtown sidewalks is the goal of another research [27]. This approach uses inverse reinforcement learning (*IRL*) to learn human-like navigation behavior based on example paths. Since robots have only limited sensing, the author extends existing *IRL* methods to the case of partially observable environments. *Shen, Jiali, and Huosheng Hu* presents an approach to visual navigation of a museum guide robot, which can detect visitors nearby and interact with them via voice and touching screen [28]. The software architecture of the robot is presented to show how this complex system is organized. Furthermore, a fast and robust multi-sensor based navigation is explained. Implementation of a tour guide robot via shape recognition and path planning is the subject of a paper presented by some Taiwanese academic researchers [29]. A tour guide robot with image system is implemented in their paper. The principal aim is the application of a shape recognition method so that the robot can move autonomously. In addition, to abandon the conventional rail-type

tracking control, they intend a fixed path planning to let the robot move autonomously along the desired path. *Gorry, Benjamin, et al.* describes a new computer vision algorithm that has been developed to track moving objects as part of a long-term study into the design of semi-autonomous vehicles [30]. The basic of the work is the mean shift object-tracking algorithm, for a moving target, it is usual to define a rectangular target window in an initial frame, and then process the data within that window to separate the tracked object from the background by the mean shift segmentation algorithm. Rather than use the standard, *Epanechnikov kernel*, they have used a kernel weighted by the *Chamfer* distance transform to improve the accuracy of target representation and localization, minimizing the distance between the two distributions in RGB color space using the Bhattacharya coefficient. Moving obstacle avoidance of a mobile robot using a single camera is the issue of a research by *Kim, Jeongdae, and Yongtae Do*, from *Daegu University* in south Korea [31]. Their paper presents some preliminary results of the detection of moving obstacles by the use of a single camera attached to a mobile robot. When a camera moves, simple moving object detection techniques for a stationary camera, such as background subtraction or image differencing, cannot be employed. They thus detect objects that move near the robot by block-based motion estimation. In the method an image is firstly divided into small blocks, and then the motion of each block is searched by comparing two consecutive images. If the motion between matching blocks is significantly large, the block in the current image is classified as belonging to moving objects. Another literature presents recognition of moving objects by image processing and its applications to a guide robot [32]. As the guide robot estimates the direction of the movement of approaching objects and emits a warning about its own movement, the user and the guide robot can safely pass by moving objects such as pedestrians. This paper improved the safety of the guide robot and proposed a method of recognizing moving objects by image processing. They have presented a recognition algorithm for both moving objects and safe-zone to moving. Japanese researchers from *Saitama University* present the tracking visitors with sensor poles for robot's museum tour [33]. It is proposed in their article a robot system which can take visitors on guided tours. The robot consists of simple devices such as multiple laser range finder attached to a pole. By just placing the sensor

poles, they could track the location and orientation of the robot and visitors at the same time. Then they conducted experiments to confirm the effectiveness and accuracy of the system. In a resembling paper, a simple Personal Identification (*SPI*) method using Dress Color Information (*DCI*) as a kind of image processing for a guide robot is proposed [34]. The *DCI* is a small number of color information that is only calculated at narrow areas around a user's joint positions obtained via *KINECT* on a mobile robot. The *SPI* method includes not only the person's skeletal information but also the *DCI*. This method can identify the specific user in real time. As a result if the mobile robot loses the user temporarily when there are many people present, it can find the user properly and promptly. Visual navigation of museum guide robot is another topic in this issue using by image processing [35]. The paper presents a novel approach to visual based navigation of a robot which can detect visitors nearby and interact with them via voice and a touching screen. The software architecture of the robot is presented to show how this complex system is organized. Furthermore, a fast and robust multi-sensor based navigation is explained. *Elnagar, Ashraf, and Leena Lulu* from department of computer science, *university of Sharjah* employs a visual tool for computer supported learning: as the robot motion planning example [36]. They introduce an effective computer aided learning visual tool (*CALVT*) to teach graph-based applications. In the robot motion planning, *CALVT* enables users to setup the working environment by creating obstacles and a robot of different shapes, specifying starting and goal positions, and setting other path or environment parameters from a user-friendly interface. Localization problem using geometric features from a 360° laser range finder and monocular vision system is described in another paper [37]. It's practicability under conditions of continuous localization during motion in real time is investigated in large-scale experiments. The features are infinite horizontal lines for the laser and vertical lines for the camera. They are extracted using physically well-grounded models for all sensors and passed to a *Kalman* filter for fusion and position estimation. Positioning accuracy close to sub-centimeter has been achieved with an environment model requiring 30 byte/m². Already with a moderate number of matched features, the vision information was found to further increase this precision, particularly in the orientation.

All of the intelligent service robots which their specifications have been extracted from the above literatures are listed below. Capability and specification of each one is described briefly. In appendix 1 all of these service robots have been summarized as a table and presented in a short description, showed a picture of them and where have been used.

MINERVA [12] is an interactive tour-guide robot, which was successfully exhibited in a *Smithsonian museum*. During its two weeks of operation, the robot interacted with thousands of people, traversing more than 44 km at speeds of up to 163 cm/sec. *XAVIER* [13] has the identity of a building guide robot which can accept commands to travel to different offices in the building, broadcasting camera images as it travels. A number of intelligent service robots have been developed to guide a building for visitors. The robot named *PHOPE* [7] is made for guidance at indoor environment. All software components including face recognition, voice communication, navigation, touch-pad based *GUI*, database and *TTS* systems are integrated into one guide system based on *PIRO* Service Mobile Robot (*PSMR*). And hardware components including sensors, actuators, micro-processors and power-supply system are also integrated into the robot. *ROBOX* [26] has the role of an interactive moving machine which can operate in human environments and interact by seeing humans, talking to and looking at them, showing icons and asking them to answer its questions. *RHINO* [28] is another autonomous, interactive tour-guide robot. It is deployed for a period of six days in a densely populated museum. The empirical results demonstrate reliable operation in public environments. The robot successfully raised the museum's attendance. In addition, thousands of people all over the world controlled the robot through the Web. *PYGMALION* [37] is a fully autonomous self-contained robot which has the capability of localization using geometric features from a 360° laser range finder and a monocular vision system. A test bed for the localization system was an annual computer tradeshow in *Lausanne, Switzerland*, where during 4 days visitors could give high-level navigation commands to the robot via a web interface. *SAGE* [8] has the role of a guide robot that has been installed at the *Carnegie Museum of Natural History* as a full-time autonomous member of the staff. Its goal is to provide educational content to museum visitors in order to

augment their museum experience. Sage's topological navigation system is based on color vision and odometric information. The robot has also the ability to conduct automatic long-term parameter adjustment. *MASTIFF-I* [11] was constructed as part of a project to build inexpensive, autonomous robot for public security and patrol building, which operated under dynamic and unknown environment. A mobile robot platform ready for the real world! *NTU-I* [18] is an outdoor tour guide robot, which is constructed to provide autonomous guiding services on the campus of *National Taiwan University*. It has twelve ultrasonic sensors around the body to detect the nearby object within three meters. Once the robot encounters an unexpected obstacle, the proposed hierarchical navigation strategy will drive the robot to avoid collision. *SECURITY WARRIOR* [21] is an intelligent security robot. The robot has developed by *National Chung Cheng University (NCCU)*. It consists of six systems including vision, motion, robot arms, power estimation, remote supervise and sensory systems. The vision system is used to carry out human detection and tracking. The motion system is built by using embedded systems and used to achieve motion planning in real time. The robot arms are the useful devices for the robot. Equipped with the arms, the robot can perform more tasks such as gesture expression or grasp functions. In order to avoid the robot shutdown suddenly, the power estimation system has been employed. For the robot, the sensory system is one of the important parts. With the fire detection sensor, the security robot can detect fire alarm in the environment. Path planning is accomplished by using the multi-sensor fusion. Body sensors can detect intruders. Combining the sensory and remote supervising systems, the owner can get notice whether there are any situations occurred. *ROBOVI* [18] is an autonomous robot which is able to form relationships with children and that children might learn from robots as they learn from other children. Using wireless identification tags and sensors, these robots identified and interacted with children who came near them. The robots gestured and spoke English with the children, using a vocabulary of about 300 sentences for speaking and 50 words for recognition. The robot has been also developed for a shopping mall and conducted a field trial with it. The robot was designed to interact naturally with customers and to affectively provide shopping information. *ATLAS* [28] has the role of a museum guide robot. The visual navigation system of this robot can detect visitors nearby and interact with them via voice and a

touching screen. *PIONEER 3-AT* [19] is a small four-wheel, four motor skid-steer robot ideal for all-terrain operation or laboratory experimentation. Pioneer research robots are the world's most popular intelligent mobile robots for education and research. *MODROB-C & MODROB-D* [16] robots are technology demonstrative mobile robots. The aim of these robots is to recognize open doors and move through the open door with narrow clearance. *TT-JOY* [9] has the identity of a mobile guide robot, which can detect multiple persons around it and choose the closest one as a user. This is realized using an Omni-directional camera and a laser range finder. The robot also has a projector with a pan-tilt mechanism. It can project information anywhere in the environment and guide a person. The user can input commands by simple gestures using the foot on the dialog box projected onto the floor. After destination decision, the robot guides him/her by its motion and information projected onto the floor/wall/ceiling. The displayed information is changed corresponding to his/her position. *VBSAR* [17] is a vision-based shopping assistant mobile robot which can serve people in a mall. By using such robots, a mall could not only save human resources effectively but also improve the quality of information service. In general, a mall always provides three kinds of customer services as guiding, communication and accompanying. Finally *NEEL* [15] is an intelligent shopping guide robot. The project *Myneel* was envisaged to provide with crucial insights into the commercialization of service robots. It has an interactive conversational agent which can serve shopping needs of the visitors in a shopping mall.

The service robotics market is expected to reach \$19.41 Billion by 2020 at a CAGR of 21.5% from 2014 to 2020 around the world [38]. According to another statistics published by the United Nations Economic Commission for Europe (*UNECE*) [39], there are over 20,000 professional service robots in use today valued at an estimated \$2.4 billion. If personal entertainment robots and domestic robots like vacuum cleaners are included, this number is well over \$3.5 billion. These economical statistics affirm the importance of our prospect, if we can actualize our research and be hopefully a portion of this market. Below we have looked up a couple of service robots in the market.

Panasonic's *HOSPI* automatic medication delivery robot has already made it into hospitals in Japan and other countries where it is used to sort and transport medications to nurse stations [40]. The *HOSPI-Rimo* employs the same self guiding technology and high-definition visual communications technology found in *HOSPI*, but is tasked with serving as an intermediary to provide communication between people who are bedridden or have limited mobility to communicate and others, such as an attending doctor somewhere else in the hospital or far flung friends and family, as if they were interacting face to face. *SCITOS A5* is a versatile mobile service-guide robot, which pleasantly communicates with people via speech or a touch screen at any location [41]. Driving and approaching people autonomously, it certainly works well implemented at the “Points-of-Sale”, an exhibition booth or inside an office building. *SCITOS G6 Transporter*, a mobile transporter that is able to fully autonomously carry out flexible intra-logistic tasks [41]. It is perfect for fast food restaurants and cafeterias. The transporter is able to detect when it is full of trays with dirty dishes and then fully autonomously brings these dishes into the kitchen in order to be cleaned. The advantage this robot brings is the fact that they need to be. Employees no longer have to constantly check how full the tray racks are and can then focus more on the preparation of food and serving customers. This robot can be built to fit any situation and environment. Due to its navigation software there are no environmental changes as well as external sensors or markers necessary. With a small built sensor the robot is able to operate kitchen doors and eliminate the need for employee help. *SCITOS G5* is a professional platform, combining the advantages of an industrial robot with the flexibility of research robot [41]. The robot combines the advantages of industrial robots, such as robustness and longevity, with the mobility and flexibility of a research robot, which are necessary for the development of mobile and interactive robotic applications. The drive system is able to move the 60 kg platform at speed of up to 1.4 m/s and handles payloads of up to 50 kg without any difficulties. The standard *SCITOS G5* houses a ring of 24 ultrasonic range finders to obtain an image of the robot’s environment. Additionally, it includes a closed bumper as a security system. Further sensors can be added optionally, such as laser-range-finder or cameras. *SCITOS G5* has two upgraded version, *SCITOS G5 Monitoring* and *SCITOS G5 Manipulator*. *SCITOS G5*

Monitoring, an autonomous mobile service robot, can help us to measure the environmental parameters in industrial plants[40]. The second robot is *SCITOS G5* attached with a robotic arm [41]. It is perfect for precise transportation needs. With the robotic arm the robot is able to transport fragile objects to specific locations with an astonishing accuracy of 2-3 cm. And finally *SCITOS G3* is a highly interactive home-care robot system for the support of persons in home environments, nursing homes, or hospitals [41].

As mentioned in the previous chapter, the main goal of this thesis is designing and manufacturing of an autonomous robot namely as Guide – Guard robot and then enabling it to safely navigate in our indoor environment.

By surviving all of above literatures, and studying the various control architectures, we decided to accomplish the navigation task, using by a non-map behavior-based architecture. After designing and manufacturing of the physical structure and then implementing of the architecture on it, the advantages and disadvantages of such an architecture will be studied against map-based navigation system.

CHAPTER 3

SYSTEM DESIGN

System design of the guide – guard robot first requires classification of all functions which the robot should perform. Such a classification has been represented in Figure 1. The main purpose of this thesis as described in chapter 1, has been classified as navigation in this diagram. In order to perform the navigation task, an integrated system is designed. The system design is described comprehensively in this section.

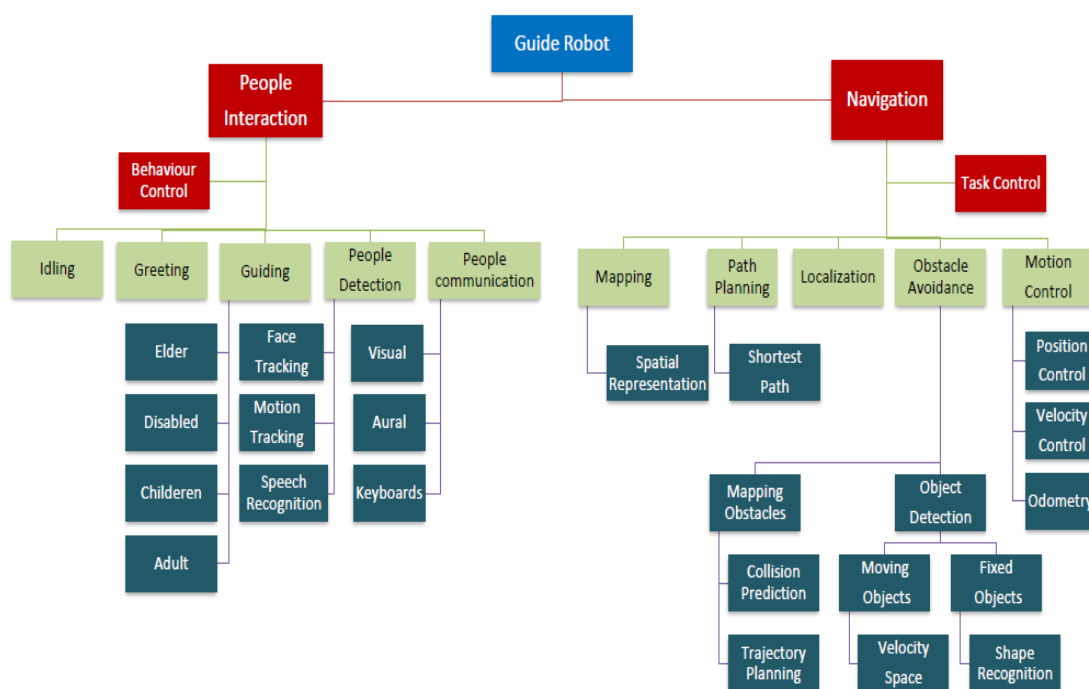


Figure 1: The Functions of the Guide Robot

3.1 System Structure

The physical structure of our guide robot is shown in Figure 2. The designed mobile platform is composed of a body (chassis framework, floors and walls), a differential

driving mechanism with two driving and two free-rotating wheels, two DC servo motors, twelve IR proximity sensors, four ultrasonic range sensor, a compass, a navigation computer, two microcontroller boards, a motor driver shield and a voltage regulator. All of these are described below in two main groups; Mechanical Hardware and Electronic Hardware.

- **Mechanical Hardware**

3.1.1 Body

Generally the body of the robot is in a proportional form with the task work of it, preferably in the form of human body. Human body form is more appropriate in guide robots because its design is for functional purposes, such as interacting with human tools and environments, for experimental purposes, or for other purposes. The designed body for our purpose is consisting of two frameworks; chassis and floors - walls.



Figure 2: Physical Structure of the Robot

3.1.1.1 Chassis Framework

This structural subsystem of the robot is responsible for physical support. It holds everything in place, and is, in effect, the durable “skeleton” of the robot to which all the other subsystems are attached. The structure and motion subsystems are very tightly integrated to form the chassis of the robot. The chassis is probably the largest

part of the robot, so we should make sure it is made of a light weight rigid material such as iron, steel or aluminum. Figure 3 shows our designed and built chassis for the purpose. First the needed parts for the chassis were designed in *SOLIDWORKS* environment and then assembled as shown in the figure 3. Appendix 2 shows the drawing of the chassis and four main components of it. It is worth mentioning that the primary prototype of the chasis had been designed and manufactured in our department some years ago [42].

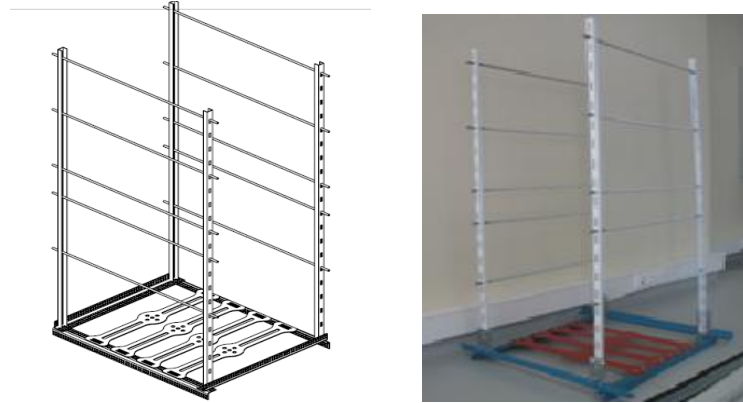


Figure 3: Designed & Built Chassis for the Robot

3.1.1.2 Floors, Walls and Sensors Mount

We decided to cover the chassis by walls and separate it to five floors. Each one of these floors will be allocated to the electronic hardware of the main functions of the robot. In order to be light, the material of the walls, floors and sensor mounts were chosen to be *Plexy-Glass*. This material is transparent and this cause we feel more to stand against a demonstration model. Similar to the design of the chassis, all of these floors, walls and sensor mounts were designed in *SOLIDWORKS* and then cut using by the Laser-cutter in our department and then assembled on the chassis. Appendix 3 shows the drawings of these parts.

3.1.2 Wheels

Wheeled robots are robots that navigate around the ground using motorized wheels to propel them. This design is simpler than using treads or legs and by using wheels they are easier to design, build, and program for movement in flat, not-so-rugged

terrain. They are also better controlled than other types of robots. Disadvantages of wheeled robots are that they cannot navigate well over obstacles, such as rocky terrain, sharp declines, or areas with low friction. Wheeled robots are most popular among the consumer market; their differential steering provides low cost and simplicity. Robots can have any number of wheels, but three wheels are sufficient for static and dynamic balance. Additional wheels can add to balance; however, additional mechanisms will be required to keep all the wheels in the ground, when the terrain is not flat.

Most wheeled robots use differential steering, which uses separately driven wheels for movement. They can change direction by rotating each wheel at a different speed. There may be additional wheels that are not driven by a motor these extra wheels help keep it balanced.

Our design is based on a four-wheeled robot, two powered, and two free rotating. For free rotating ones we preferred to use *Omni*-wheels. Figure 4 shows our used 100 mm *Track*-wheels in the rear and *Omni*-wheels in the front of the robot.



Figure 4: Rear Powered Track-Wheels & Front Omni-Wheels

An *Omni*-wheel is like many smaller wheels making up a large one, the smaller ones have axis perpendicular to the axis of the core wheel. This allows the wheels to move in two directions, and the ability to move holonomically, which means, it can instantaneously move in any direction. *Omni*-wheeled robots can move in at any angle in any direction, without rotating beforehand. The disadvantages of using *Omni*-wheels is that they have poor efficiency due to not all the wheels rotating in the direction of movement, which also causes loss from friction.

- **Electronic Hardware**

The electronic hardware of the robot consists of two DC servo motor, two encoders, twelve IR proximity sensors, four ultrasonic range sensor, a compass, two microcontroller boards (Master and Slave), one motor driver shield, one voltage regulator, one navigation computer, and one 15 inch touch screen notebook. Figure 5 shows each subsystem block diagram of the overall electronic system structure and data transmission flow between the subsystems.

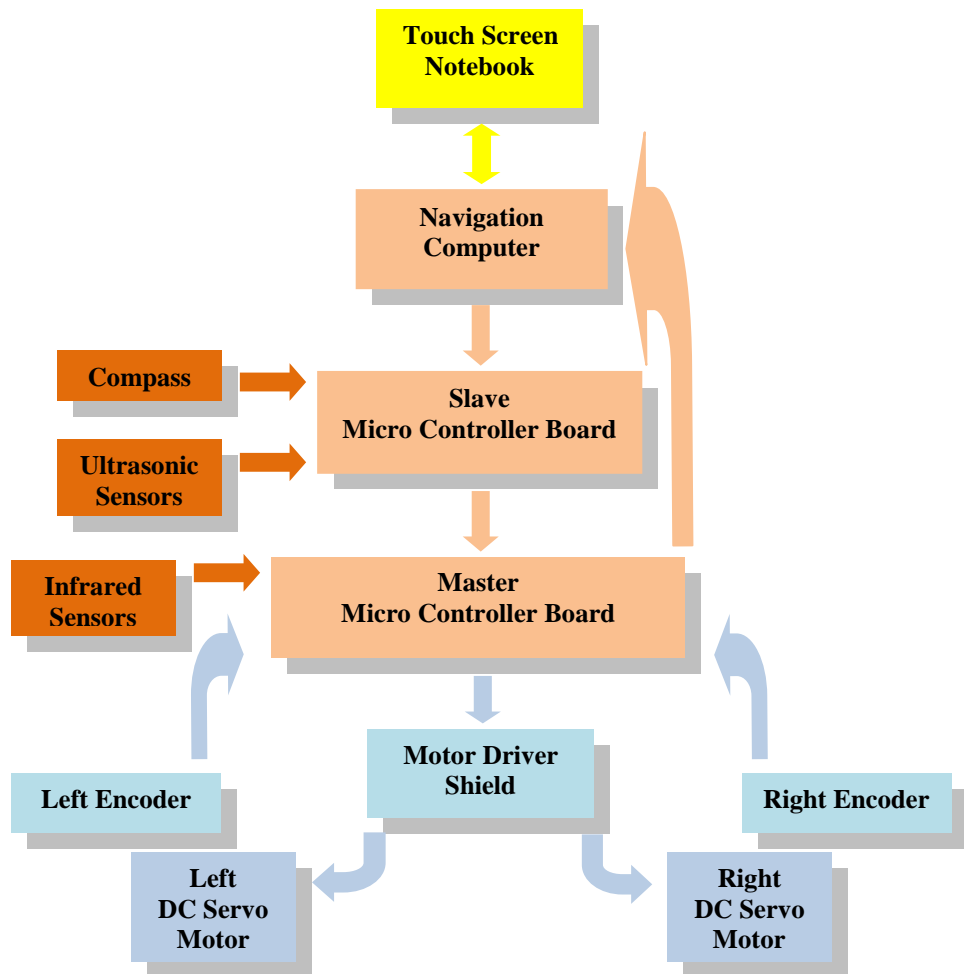


Figure 5: Data-Transmission Flow between the Subsystems of the Hardware

3.1.3 Motors

Among all kinds of DC motors, a servo kind was preferred to use in the robot, because of some effective specifications. Servo motors are standard DC or brushless motors with an encoder feedback loop. The computer reads the position of the motor and controls the power applied to the motor. Servo motors are faster moving point to point and are better at accelerating very heavy loads. It's optimized for smooth precise slower speed moment. That means more poles (brushes), more commutator segments, skewed rotor often internal mounting for an encoder, and just generally designed around a lower speed high torque use. The servo can also be wound differently, with different gauge wires to improve thermal properties. Unlike a motor a servo is expected to handle full load at zero speed, a motor will normally over heat in the same application. Figure 6 shows a prototype DC servo motor with integrated encoder and it's including parts.

In order to select a proper motor and calculate the needed torque, which should beget by each motor, the below procedure was followed:

1. The motion profile was determined and maximum velocity required to make the desired move was calculated.
2. The mechanical drive mechanism was selected and then the load torque was calculated.
3. The required motor torque for our application was determined.
4. The proper motor and driver based on their speed-torque characteristics was selected.



Figure 6: Components of a DC Servo Motor

It is worth mentioning, that the total weight of the loads (robot's weight), which our two motors should be able to carry, was estimated around 20 kg. Appendix 4 shows

the weight of all parts of the robot and the total. Since the visitors should be able to follow the robot easily and based on the human's normal walking speed (1.2 m/s) our needed motor's rpm and torque were computed as follow;

$$rpm = \frac{1.2 \times 60}{2\pi \times 0.05} = 229 \quad a = \frac{80 \times 2\pi \times 0.05}{60 \times 0.2} = 2.1m/s^2$$

$$\Gamma = \frac{M \cdot a \cdot R}{2_{motors}} = \frac{20 \times 2.1 \times 0.05}{2} = 1.05N.m = 10.7kg.cm$$

Our selected motor is an integrated encoder, powerful brushed DC servo motor with metal gearbox intended for operation at 12 V manufactured by *Pololu Robotics & Electronics* [43]. Figure 7 shows this motor and its important specifications.

- Gear Ratio: 131:1
- Stall Torque: 18 Kg.cm
- No-Load Speed: 80 rpm
- Stall Current: 5A (@12V)



Figure 7: Used DC Servo Motor and its Specifications

In order to assemble these motors on the chassis of the robot, a proper mount was designed and built. This mount which connected to the bracket, helps to firm and strong linkage of the motors to the chassis. Figure 8 shows the built mount.



Figure 8: The Motor's Mount

3.1.4 IR Distance Sensors

An infrared sensor is a device (usually with supporting circuitry) that can detect infrared light (which is below the optical spectrum) for use to purpose. Most of the remote controls for TVs and other entertainment equipment use infrared energy as

the transmission medium to carry information between the control and the equipment to be operated. Infrared sensors also have important scientific, military, security and rescue applications since they can “see” the “radiant heat energy” which is infrared radiation. This electromagnetic energy is in the wavelengths from 750 nm, which is the lower end of the optical spectrum, to well over 10,000 nm, deep in the infrared. The heart of the system is a photo detector or photo sensor. It does its task based on black body radiation, which is the emission of energy based on the temperature of the object. As the radiant energy is a direct function of temperature, even the slightest difference in temperature results in the radiation of a slightly different wavelength of infrared light. Figure 9 shows the schematic diagram of the IR sensors and a real prototype.

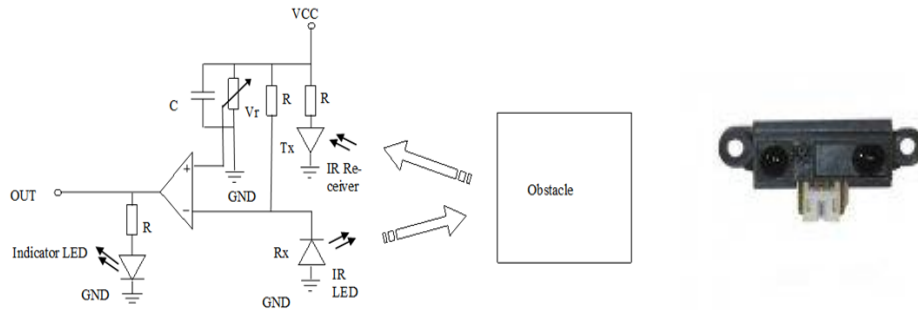


Figure 9: Schematic Diagram of the IR Sensors and a Real Prototype

This general purpose proximity IR sensor is used in our system for collision detection or obstacle detection. Our two types selected IR sensors are distance measuring sensor unit, composed of an integrated combination of *PSD* (position sensitive detector), *IRE*D (infrared emitting diode) and signal processing circuit manufactured by *Sharp* Company [44]. The long distance finder (20-150 cm) were selected to assembly on the front of the robot and short ones (10-80 cm) on the sides of the robot. Other important specifications of these two-type sensors are:

- Distance measuring range: 10-80 cm and 20-150 cm
- Consumption current : Typ. 30 mA
- Package size : 29.5×13×13.5 mm
- Supply voltage : 4.5 to 5.5 V

3.1.5 Ultrasonic Distance Sensors

Ultrasonic sensors work on a principle similar to radar or sonar, which evaluate attributes of a target by interpreting the echoes from radio or sound waves respectively. Active ultrasonic sensors generate high frequency sound waves and evaluate the echo which is received back by the sensor, measuring the time interval between sending the signal and receiving the echo to determine the distance to an object. Our used ultrasonic ranging module *HC - SR04* provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm [45]. The modules includes ultrasonic transmitters, receiver and control circuit. Figure 10 shows this sensor and its important specifications.

- Working Voltage & Current: DC 5 V, 15 mA
- Working Frequency: 40 Hz
- Working Range: 2 – 400 cm
- Measuring Angle: 15 degree



Figure 10: Used Ultrasonic Sensor and its Specifications

3.1.6 Compass

A compass is an instrument used for navigation and orientation that shows direction relative to the geographic cardinal directions. The nine independent rotation, acceleration, and magnetic readings provide all the data needed to make an attitude and heading reference system (*AHRS*). What we need in our system, is just a reference to have an accurate measurement of the robot's heading. The Pololu *MinIMU-9 v3*, which we preferred to use in our system, is an inertial measurement unit (*IMU*) that packs an *L3GD20H* 3-axis gyro and an *LSM303D* 3-axis accelerometer and 3-axis magnetometer onto a tiny $0.8'' \times 0.5''$ board [46]. Figure 11 shows this sensor and its important specifications.

- Output format (I²C):
 - Gyro: one 16-bit reading per axis

- Accelerometer: one 16-bit reading per axis
- Magnetometer: one 16-bit reading per axis
- Operating voltage: 2.5 V to 5.5 V
- Supply current: 6 mA



Figure 11: Used Compass and its Specifications

Figure 12 demonstrate the arrangement of above mentioned, IR and ultrasonic sensors, motors and wheels around the body of the robot.

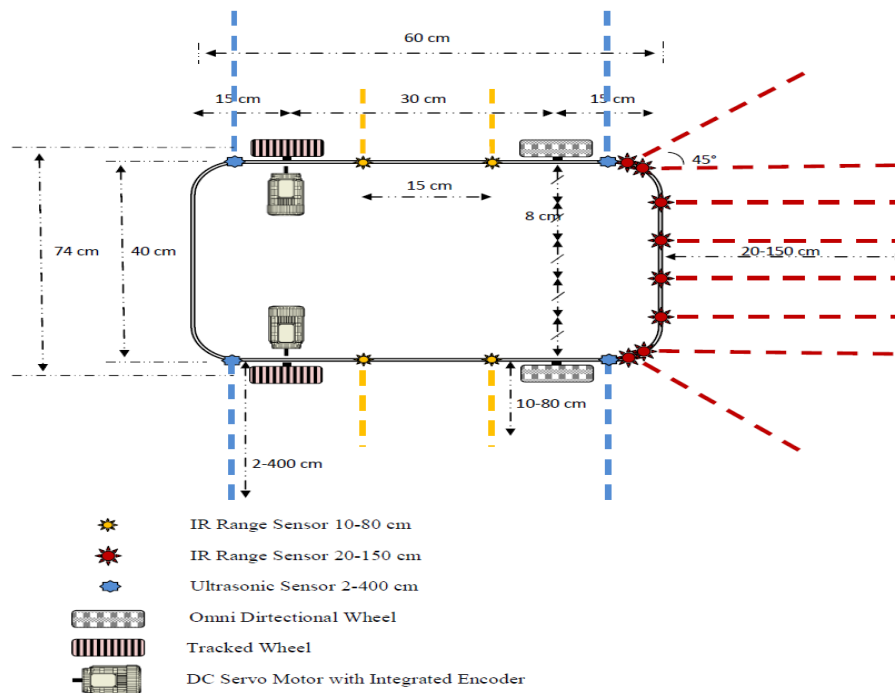


Figure 12: Arrangement of the IR, Ultrasonic Sensors, Motors & Wheels

3.1.7 Micro-Controller Board

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications in automatically controlled

products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, power tools, toys, robots and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes.

Arduino Mega 2560 which is a microcontroller board based on the *ATmega2560* manufactured by *Arduino* [47] is preferred in this work. It has 54 digital input/output pins (of which 15 can be used as *PWM* outputs), 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values), 4 *UARTs* (hardware serial ports), a 16 MHz crystal oscillator, a *USB* connection, a power jack, an *ICSP* header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a *USB* cable or power it with an AC-to-DC adapter or battery to get started. Figure 13 shows this microcontroller board, which we used two of them as master and slave boards.

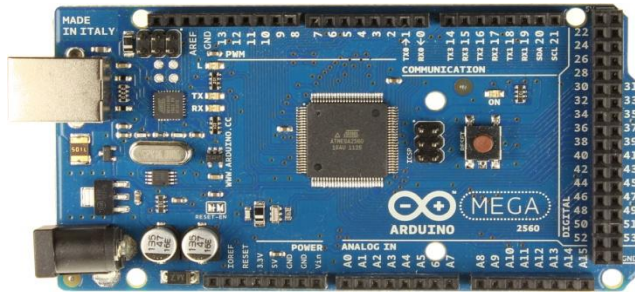


Figure 13: Microcontroller Board

3.1.8 Motor Driver Shield

This shield manufactured by *Pololu Robotics & Electronics*, makes it easy to control two high-power DC motors with our Arduino microcontroller board [48]. Its dual robust *VNH5019* motor drivers operate from 5.5 to 24 V and can deliver a continuous 12 A (30 A peak) per motor, or a continuous 24 A (60 A peak) to a single motor connected to both channels. These great drivers also offer current-sense feedback and accept ultrasonic *PWM* frequencies for quieter operation. The Arduino pin mappings can all be customized if the defaults are not convenient, and the motor driver control

lines are broken out along the left side of the shield for general-purpose use without an Arduino. Figure 14 shows this driver shield.

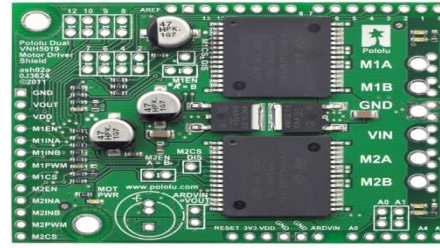


Figure 14: Motor Driver Shield

3.1.9 Voltage Regulator

We have to use a voltage regulator to provide a buffer, eliminating the loading effect. The voltage follower makes it so the voltage at one point is precisely equal to the voltage at another point, regardless of what the resistance, capacitance or inductances are. The compact switching step-down voltage regulator, manufactured by *Pololu Robotics & Electronics*, takes an input voltage between 4.5 V and 42 V and efficiently reduces it to a lower, user-adjustable voltage [49]. It has an output voltage range of 4 V to 25 V and a maximum output current of 600 mA. Figure 15 shows this regulator.



Figure 15: Voltage Regulator Board

3.1.10 Handmade Voltage and Data Distributor Board

In order to supply the voltages of twelve IR sensors, four ultrasonic sensors and two encoders we need +5v which is available by the regulator. On the other hand, the motors are supplied by a voltage source of +12v, which can be supplied directly from the battery. With the aim of supplying these mentioned elements and data transmission between them, regularly and neatly, we designed and made an electronic supply distributor board and mounted the regulator on it. Figure 16 shows our handmade board.

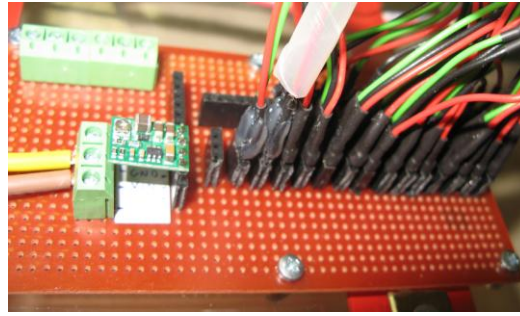


Figure 16: Handmade Distributor Board

3.1.11 Navigation Computer

Embedded computers deliver with the help of their processor, enabling hobbyists, students and innovators to bring a project to development fast. Designed with community inputs in mind, this open hardware design brings the previous generations' laptop-like performance and expandability to the next level, while adhering to hand-held power levels.

In our robot, an embedded computer will be allocated to the navigation system as well as the interaction system. For the time being, the embedded computer's task is assigned to the existing microcontroller board and a 15" Touch-Screen Notebook.

3.1.12 Touch-Screen Notebook

As mentioned above, it was decided to taking advantages of the processor of a Notebook to fulfill the embedded computer's task. It will also use from the Notebook's screen for importing the input's commands and displaying some information as the outputs. The main technical specifications of this Notebook are, 15" Display, Intel CPU Z2760 1.80 GHz Processor, 2 GB RAM, 32 bit Operating System x86 based processor, Microsoft Windows 8 version 2012.



Figure 17: Touch-Screen Notebook

3.1.13 Battery

Based on the needed supplying voltages and loading current of the motors, IR sensors and electronic boards, it's decided to use a battery with the voltage, 11.1v and power, 3050mAh. Figure 18 shows the used battery.



Figure 18: The Battery

3.2 Control Architecture

In order to design control architecture for our system, we should and ought to peruse a critical issue: how we can control a mobile robot.

3.2.1 Control of a Mobile Robot

The main question is: how we should make mobile robots move in effective, safe, predictable, and collaborative ways using modern control theory? The basic control theory determines how we should pick the control signal? And the important objectives in this process are; stability, tracking, robustness, disturbance rejection, and optimality.

3.2.2 Dynamics of the Mobile Robot

In order to make a mobile drive at a desired reference speed (r), if we take our state as the velocity (X) of the robot, and the input as the motor voltage (u), based on Newton's second law, we will have the equation 1, in which c is the electro-mechanical transmission coefficient and m is the weight of the robot;

$$F = cu = m\dot{X} \Rightarrow \dot{X} = \frac{c}{m}u \quad (1)$$

Now, assume that we want to measure the velocity ($y = \dot{X}$). With taking a simple P-regulator, we will reach the equation (2), which means exactly tracking! Figure 19 shows the response of the system to a simple step input.

$$u = ke \Rightarrow \dot{X} = 0 = \frac{c}{m} k(r - X) \Rightarrow X = r \quad (2)$$

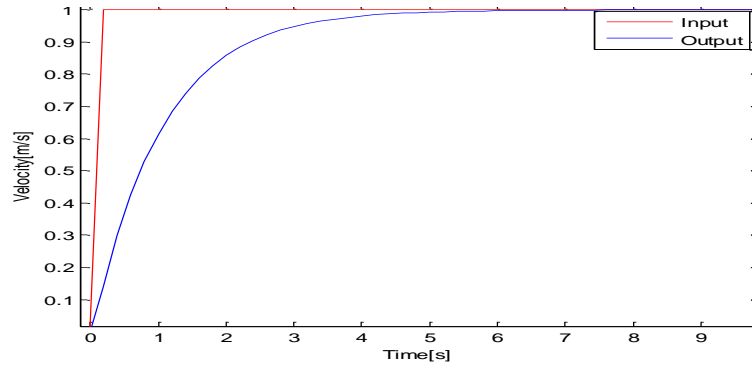


Figure 19: Response by a P-Regulator

With taking a PI-Regulator, it will result the response given in Figure 20.

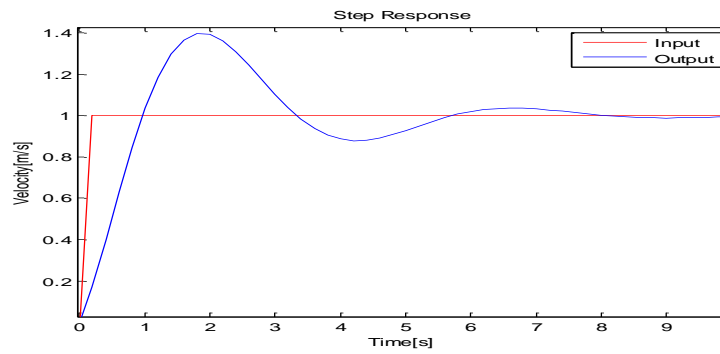


Figure 20: Response by a PI-Regulator

And a PID-Regulator will cause to have the response shown in Figure 21.

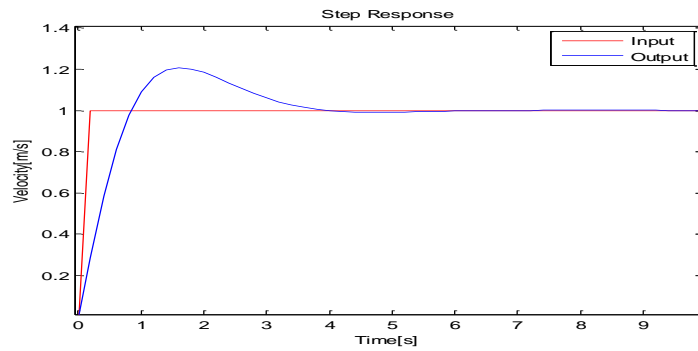


Figure 21: Response by a PID-Regulator

Appendix 5 represents a simple model to simulate the step responses of the system as mentioned above.

By the aim of driving a mobile robot around, as shown in Figure 22, the main problem will be taking it to drive from point A to point B?

The controller must be able to respond to dynamic and unknown environmental conditions. So instead of building one complicated controller, it's better to divide and conquer the whole control task, and consider wellknown behaviors such as: *Go-to-Goal*, *Avoid-Obstacles*, *Follow-Wall*, and *Track-Target*.

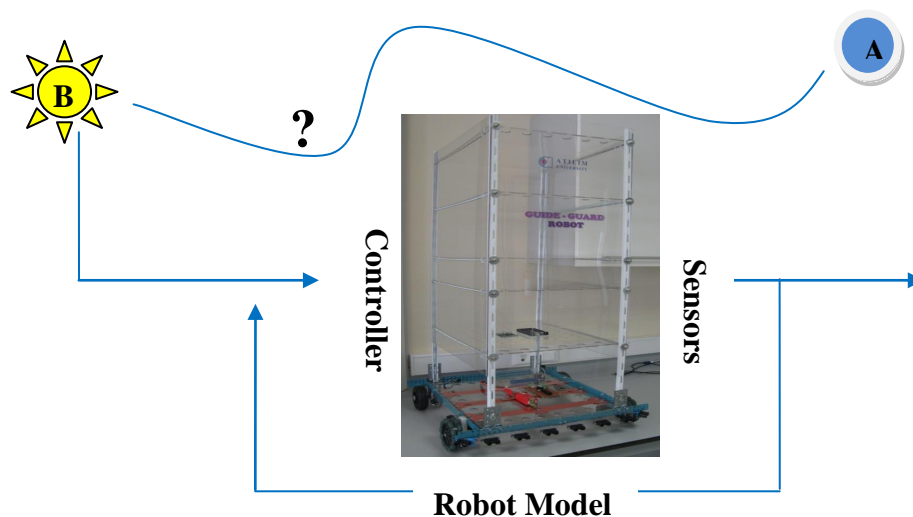


Figure 22: Taking the Robot to Drive from two Points

Switching among controllers in response to the environmental changes is simpler than have a complicated controller. This is *Behavior-Based Robotics* [50][51].

To build a behavior, assume we have a differential-drive, wheeled mobile robot driving at a constant speed, and therefore we have the equations (3).

$$\begin{cases} \dot{x} = v_0 \cdot \cos \varphi \\ \dot{y} = v_0 \cdot \sin \varphi \\ \dot{\varphi} = \omega \end{cases} \quad (3)$$

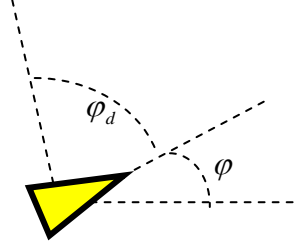


Figure 23: Heading and Desired Heading

If we want to drive in a desired heading, how ω should be calculated? We have a reference, a model, a control input, and a tracking error as equations (4).

$$r = \varphi_d, \quad e = \varphi_d - \varphi, \quad \dot{\varphi} = \omega \quad (4)$$

And by using a *PID*, we will have: $\omega = PID(e) = k_p e + k_I \int_0^t e d\tau + k_D \dot{e}$ (5)

In Go-to-Goal behavior, as it is clear from the following figure, the desired heading will be as the equation (6).

$$\varphi_d = \arctan\left(\frac{y_g - y}{x_g - x}\right) \quad (6)$$

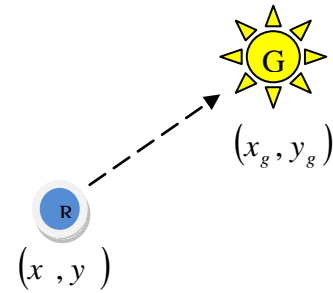


Figure 24: Desired Heading in Go-to-Goal Behavior

We can use the same idea in the Obstacle-Avoidance Behavior. Equations 7, 8, 9 and 10 present some alternatives to this behavior.

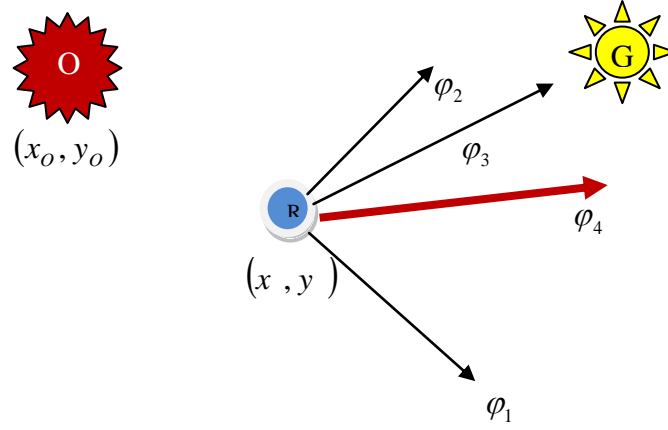


Figure 25: Alternative Desired Headings in Obstacle Avoidance Behavior

$$\varphi_1 = \varphi_{obst} + \pi \quad \text{'Pure Avoidance'} \quad (7)$$

$$\varphi_2 = \varphi_{obst} \pm \frac{\pi}{2} \quad (8)$$

$$\varphi_3 = \varphi_{goal} \quad \text{'Pure Go-to-Goal'} \quad (9)$$

Unlike there is no obvious “correct” choice between the above hard behaviors, but it seems, a combined behavior (blended) be more logical.

$$\varphi_4 = F(\varphi_{obst}, \varphi_{goal}) \quad \text{'Blended'} \quad (10)$$

3.2.3 Hard Switches vs. Blending

Given two different behaviors, they can be combined in two options:

- Hard Switches
- Blended Behaviors

Each of these options has its own pros and cons. The pro of hard switches is performance guarantees and con of it is bumpy ride. Against the pro of blended behaviors is smooth ride and con of it is no guarantees.

3.2.4 Hybrid Automata

We need to be able to describe system that contains both the continuous dynamics and discrete switch logic. *Hybrid Automata* means, finite discrete logic on continuous dynamics. In this issue, there are some important punch lines.

- By combining stable modes, the resulting hybrid system may be unstable!
- By combining unstable modes, the resulting hybrid system may be stable!
- Design stable modes but be aware that this is a risk one may face!

As it is mentioned, stable subsystems do not guarantee a stable hybrid system. In other words, it is possible to destabilize stable subsystems by an unfortunate series of switches.

3.2.5 Navigation using Behaviors

For the purpose of planning, let's start simply. If we write our dynamic equation as (11), and take the controllability matrix on it as equation (12), we find out, our system is completely controllable.

$$\dot{x} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times u \quad (11)$$

$$\text{rank}(\begin{bmatrix} B & AB \end{bmatrix}) = \begin{bmatrix} I & 0 \end{bmatrix} = 2 \quad (12)$$

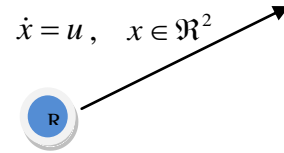


Figure 26: Control Input Vector

The two bread-and-butter behaviors are go-to-goal and avoid-obstacle.

3.2.5.1 Go-to-Goal Behavior

This behavior drives the robot towards the goal.

The error function and control inputs will be as the equation (13).

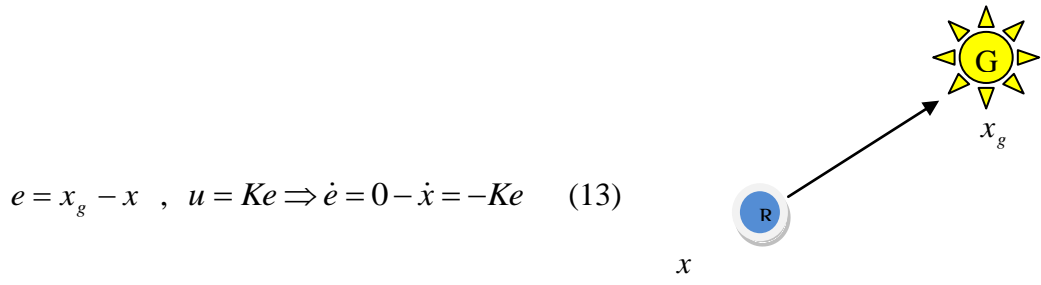


Figure 27: Control Input Vector in Go-to-Goal Behavior

The system is asymptotically stable. That means if: $k \succ 0$ ($\text{eig}[K] \succ 0$), $e \rightarrow 0$

But a linear controller means the robot goes faster the further away the goal is. The solution in practice is making the gain K a function of e like equations (14).

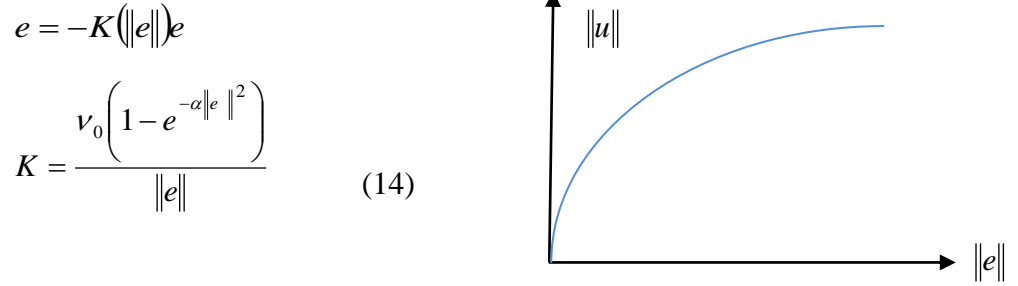


Figure 28: Control Input Vector vs. Error in Go-to-Goal Behavior

3.2.5.2 Avoid-Obstacle Behavior

This behavior means, don't slam into things. In order to get the direction, let's just flip the sign in the go-to-goal behavior as shown in equation (15).

$$e = x_o - x \quad (15)$$

$$u = Ke \Rightarrow \dot{e} = Ke$$

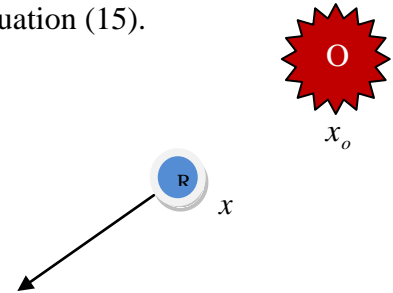


Figure 29: Control Input Vector in Avoid-Obstacle Behavior

Now the system is unstable. That means the robot drives off to infinity. And also we care less the closer we get. The solution is making K dependent on e like equation (16).

$$K = \frac{1}{\|e\|} \left(\frac{c}{\|e\|^2 + \varepsilon} \right) \quad (16)$$

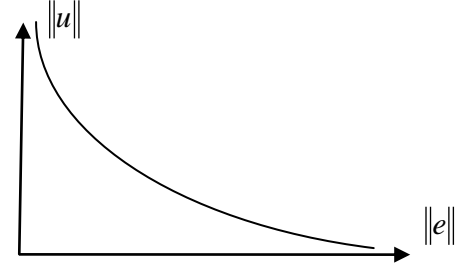


Figure 30: Control Input Vector vs. Error in Avoid-Obstacle Behaviors

Now we are ready to switch between behaviors, as shown in figure 32. The Hybrid Automata switches between two behaviors as presented in equations (17).

$$\begin{aligned} u_{AO} &= K_{AO} \cdot (x - x_o) \\ u_{GTG} &= K_{GTG} \cdot (x_g - x) \end{aligned} \quad (17)$$

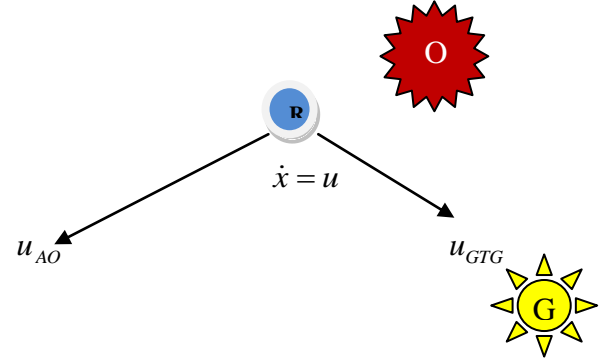


Figure 31: Control Input Vectors of Go-to-Goal and Avoid-Obstacle Behaviors

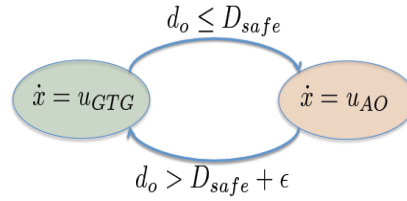


Figure 32: Hybrid Automata

3.2.5.3 Blending

The blending Function $\sigma(d_o) \in [0,1]$ can be defined as the equation (18).

$$\dot{x} = \sigma(d_o) \cdot u_{GTG} + (1 - \sigma(d_o)) \cdot u_{AO} \quad (18)$$

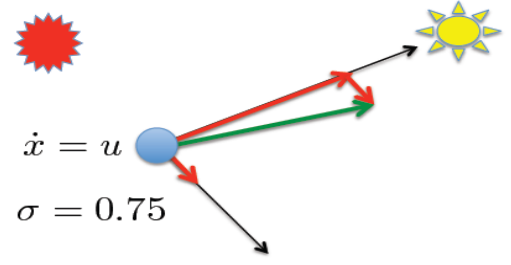


Figure 33: Control Input Vector in Blended Behavior

3.2.5.4 Wall Following Behavior

We really need at least one more behavior to be able to deal with these obstacle worlds. A really useful behavior is one that makes the robot follow the boundary of an obstacle/wall.

The follow-wall behavior should maintain a constant distance to the obstacle/wall it is if following as displayed in equation (19).

$$u_{FW} = \alpha \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \times u_{AO} = \alpha \cdot R\left(-\frac{\pi}{2}\right) \times u_{AO} \quad (19)$$

Figure 34: Control Input Vector in Follow-Wall Behavior

But we can really move in two different directions along a wall as shown equations (20) and (21).

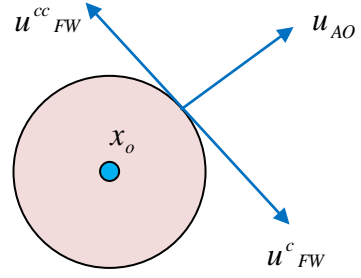


Figure 35: Wall Following in two different Directions

$$u^c_{FW} = \alpha \cdot R\left(-\pi/2\right) \times u_{AO} = \alpha \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \times u_{AO} \quad (20)$$

$$u^{cc}_{FW} = \alpha \cdot R\left(\pi/2\right) \times u_{AO} = \alpha \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \times u_{AO} \quad (21)$$

But now which direction should we choose? Although there is no obvious answer to that but let the goal-to-goal direction determines. Equation (22) describes this situation.

$$\begin{cases} \langle u_{GTG}, u^c_{FW} \rangle > 0 \Rightarrow u^c_{FW} \\ \langle u_{GTG}, u^{cc}_{FW} \rangle > 0 \Rightarrow u^{cc}_{FW} \end{cases} \quad (22)$$

We should stop the wall-following behavior, when enough progress has been made and we have a “clear” shot to the goal! These conditions are presented in below equations.

$$\begin{aligned} \tau &= \text{time of last switch} \\ \text{progress} : \|x - x_g\| < \|x(\tau) - x_g\| \\ \text{clearshot} : \langle u_{AO}, u_{GTG} \rangle > 0 \end{aligned} \quad (23)$$

3.2.5.5 Hybrid Automata

Now if we put all behaviors together in a plan, named *Hybrid Automata*, we will achieve a single plan as shown in figure 36.

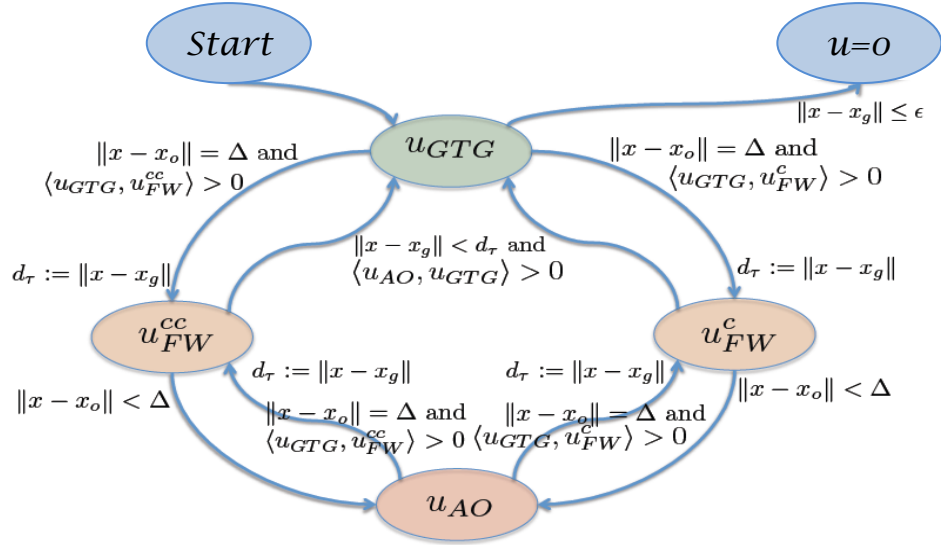


Figure 36: All Behaviors in single Hybrid Automata

3.2.6 Layered architecture

We have a problem: even with a simple robot model, the navigation architecture becomes rather involved. The layered architecture makes the navigation problem easier by separating it into a planning phase and tracking phase [52].

Standard navigation systems are typically decoupled along three different levels of abstraction:

Strategic Level: Where to go (high-level, long-term)?

Operational Level: Where to go (low-level, short-term)?

Tactical level: How to go there?

Or slightly less militaristic:

High-Level Planning: Where should the (intermediary) goal points be?

Low-Level Planning: Which “direction” to move in-between goal points?

Execution: How make the robot move in those directions?

There are many Artificial Intelligence (AI) methods (e.g., *Dijkstra Algorithm* [53], *Dynamic Programming* [54], *PRT* [55], etc) for doing high-level planning. Based on

the platform of the laboratories of our department, which is shown in appendix 6, all of the intermediary goal points and final goal points in the space work of the robot were extracted. Figure 37 shows these points on the map, followed by the coordinates of each one which is shown in Table 1.

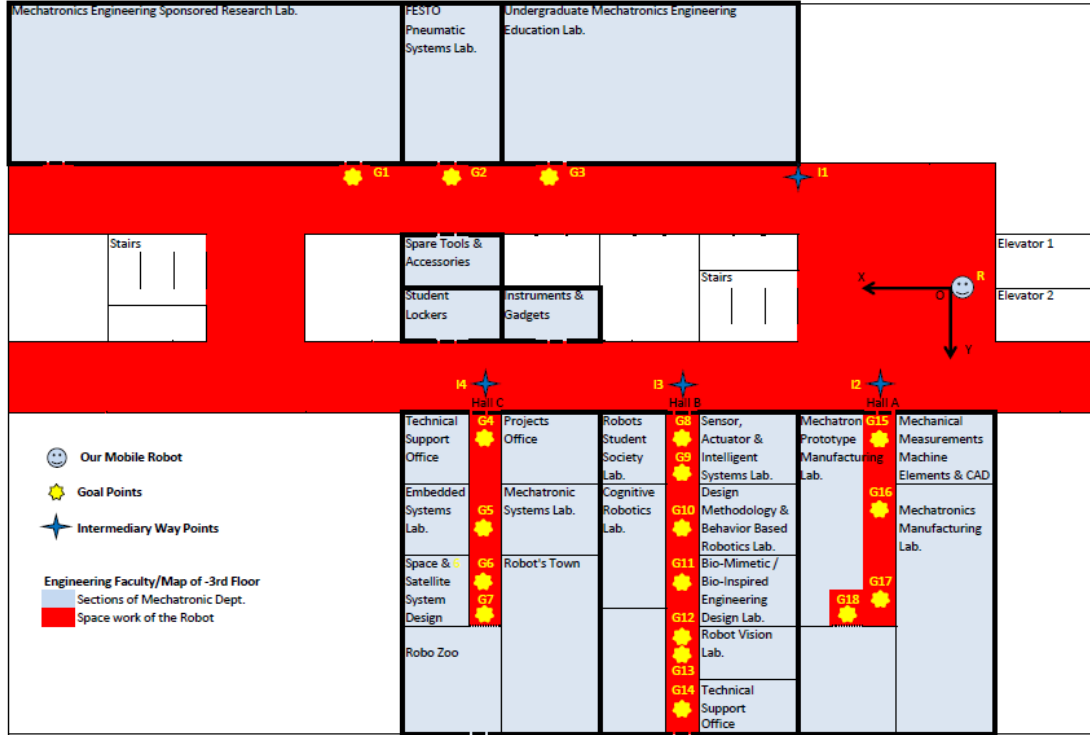


Figure 37: Intermediary & Final Goal Points on the Map Platform

We already know to do low-level planning. By assuming $\dot{x} = u$, the output is a desired direction (and magnitude) of travel. The execution-level will be described in the next section, comprehensively and practically.

Now we are ready to design control architecture to cover our requirements. The control architecture is responsible to implement two main functions to move and sense. These functions will be implemented by *MATLAB* software on the navigation computer.

1. Transform the outputs of our controllers to the control inputs of the robot
2. Keep track of where the robot is located

In order to do these functions, the robot has to have data transmission between two motors, twelve IR sensors, four ultrasonic sensors, two encoders and a compass through a supervisor, as shown in Figure 38. The inputs will be entered by selecting and touching the *Push-Bottoms* on the screen of the notebook and some useful information as the outputs will be displayed on it at the goal points. These graphical interfaces will be provided by the help of the *MATLAB GUI* (Graphical User Interface), which will be completely described in the next chapter.

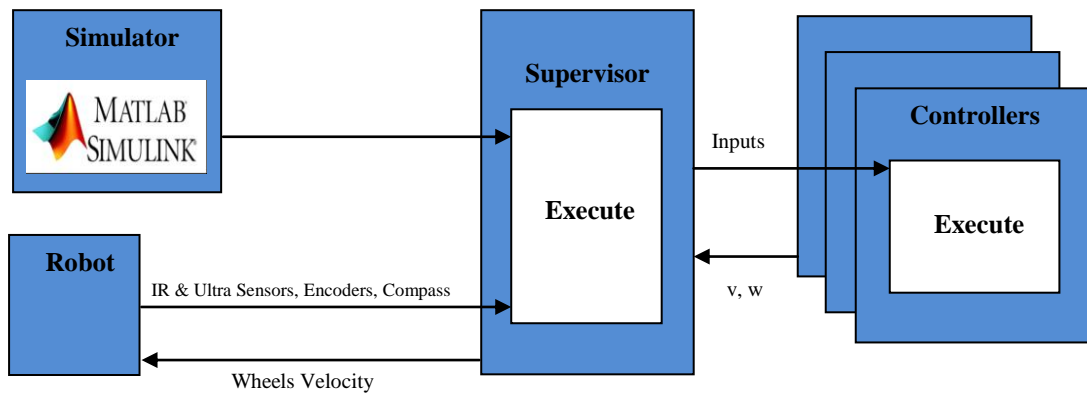


Figure 38: Schematic of the Control Architecture

Table 1: Coordinates of the Robot, Intermediary & Final Goal Points

	Points	X	Y	Laboratory
Goal	G1	3028	-300	Mechatronics Engineering Sponsored Research
	G2	2718	-300	FESTO Pneumatic Systems
	G3	2230	-300	Undergraduate Mechatronics Engineering Education
	G4	2310	1108	Technical Support Office / Projects Office
	G5	2310	1828	Embedded Systems / Mechatronic Systems
	G6	2310	2010	Space & Satellite System Design / Robots Town
	G7	2310	2508	Robo Zoo
	G8	1280	1112	Sensor, Actuator & Intelligent Systems
	G9	1280	1298	Robots Student Society
	G10	1280	1698	Design Methodology & Behavior Based Robotics
	G11	1280	1877	Cognitive Robotics / Bio-Mimetic Bio-Inspired Design
	G12	1280	2280	Empty
	G13	1280	2465	Robot Vision
	G14	1280	2645	Technical Support Office
	G15	354	1372	Mechanical Measurements Machine Elements & CAD
	G16	354	965	Mechatronic Prototype Manufacturing
	G17	354	1781	Mechatronics Manufacturing
	G18	444	1982	Empty
Intermediary	I1	845	-450	Right Corridor
	I2	404	655	Hall A
	I3	1360	830	Hall B
	I4	2360	830	Hall C
Robot	R	0	0	Initial Point

CHAPTER 4

SYSTEM IMPLEMENTATION

The system design was described comprehensively in the last chapter. The system structure, including of the mechanical and electronic hardware, and the software architecture were designed based on our needs. The chapter was closed by the assembling of the mechanical parts. The system implementation is getting started by the electronic hardware assembling, voltage supplying, data lines wiring, and continued by the data transmission methods from IR sensors, ultrasonic sensors, compass, and encoders to the microcontroller, and control data from there to the motors.

4.1 Electronic Hardware Assembling

As it mentioned in the last chapter, in order to transmit data and supply the voltages of twelve IR sensors, four ultrasonic sensors, two encoders, and also our microcontroller board and motor driver shield, an electronic distribution board was designed. Our regulator board was mounted on it, and the voltage-supply and data lines were wired between IR and ultrasonic sensors and motors, and the microcontroller board through it. Appendix 7 shows various shots of the assembling and wiring steps.

4.1.1 Voltage Supplying

Our twelve IR sensors, four ultrasonic sensors, compass, two encoders and microcontroller boards are supplied by a voltage source of +5v, which is available by the regulator mounted on the distributor board. On the other hand, the motors are supplied by a voltage source of +12v, which can be supplied by the driver shield. This shield can be supplied by a +12v source, directly from the battery.

4.1.2 Data and Power Lines Wiring

Table 2: Analog/Digital Connection Lines between Parts & Microcontroller Borads

Part	No.	Pin	Power Line		Data Line
			+	-	
IRS	1	-	DB(+)	DB(-)	A2-MMCB
	2	-	DB(+)	DB(-)	A3-MMCB
	3	-	DB(+)	DB(-)	A4-MMCB
	4	-	DB(+)	DB(-)	A5-MMCB
	5	-	DB(+)	DB(-)	A6-MMCB
	6	-	DB(+)	DB(-)	A7-MMCB
	7	-	DB(+)	DB(-)	A8-MMCB
	8	-	DB(+)	DB(-)	A9-MMCB
	9	-	DB(+)	DB(-)	A10-MMCB
	10	-	DB(+)	DB(-)	A11-MMCB
	11	-	DB(+)	DB(-)	A12-MMCB
	12	-	DB(+)	DB(-)	A13-MMCB
US	1	T	DB(+)	DB(-)	D13-SMCB
		E			D12-SMCB
	2	T	DB(+)	DB(-)	D7-SMCB
		E			D6-SMCB
	3	T	DB(+)	DB(-)	D9-SMCB
		E			D8-SMCB
	4	T	DB(+)	DB(-)	D11-SMCB
		E			D10-SMCB
M	1	-	M1A-DS	M1B-DS	-
	2	-	M2A-DS	M2B-DS	-
CP	-	SDA	5V-SMCB	GND-SMCB	SDA-SMCB
		SCL			SCL-SMCB
E	1	A	DB(+)	DB(-)	TX1-MMCB
		B	DB(+)	DB(-)	RX1-MMCB
	2	A	DB(+)	DB(-)	SDA-MMCB
		B	DB(+)	DB(-)	SCL-MMCB
SMCB	-	TX0	USB	GND-MMCB	RX0-MMCB
		TX2			RX2-MMCB
		TX3			RX3-MMCB
DS	-	-	Battery(+)	Battery(-)	-

The data and power transmission between IR Sensors (*IRS*), Ultrasonic Sensors (*US* (T; trigger, E; echo)), Compass (*CP*), Encoders (*E* (pin A, pin B)), Motors (*M*), Microcontroller Boards (Master; *MMCB*, Slave; *SMCB*) and Driver Shield (*DS*) is done regularly through our handmade Distributor Board (*DB*). The above Table 2 demonstrates these analog and digital connection lines between the whole of hardware. Table 3 shows digital connection lines between *MMCB* and *DS*.

Table 3: Analog/Digital Connection Lines between MMCB and DS

Part	Pin	DS
MMCB	D2	M1INA
	D4	M1INB
	D6	M1EN/DIAG
	D7	M2INA
	D8	M2INB
	D9	M1PWM
	D10	M2PWM
	D12	M2EN/DIAG
	A0	M1CS
	A1	M2CS

4.2 Data Transmission

The robot is equipped with twelve infrared range sensors, of which eight are located in the front and four are located on its sides. Four ultrasonic sensors are located on sides of the robot. It has a two-wheel differential drive system (two wheels, two motors) with a wheel encoder for each wheel and one compass. Data transmission, from sensors and encoders to the microcontroller, and control data from there to the motors, are decrypting comprehensively here. The related flowchart was demonstated in Figure 5 of chapter 3.

4.2.1 IR Range Sensors

The orientation (relative to the body of the robot, as shown in figure 10 in section 3.1.4) of IR sensors 1 through 12 is 90^0 , 90^0 , 45^0 , 0^0 , 0^0 , 0^0 , 0^0 , 0^0 , 0^0 , -45^0 , -90^0 ,

-90°, respectively. Our IR range sensors are effective in the range of 10-80 cm and 20-150 cm only. However, the IR sensors return raw values in the range of [0.4, 2.75] and [0.4, 3.25] volt, instead of the measured distances, Figure 39 (a) and (b) demonstrate the function that maps these sensors values to distances.

To complicate matters slightly, the *Arduino Mega 2560* digitizes the analog output voltage using a voltage divider and a 12-bit analog-to-digital converter (ADC). Table 4 is a look-up table to demonstrate the relationship between the ADC output, the analog voltage from the IR proximity sensor, and the approximate distance that corresponds to this voltage for the both front and side ones.

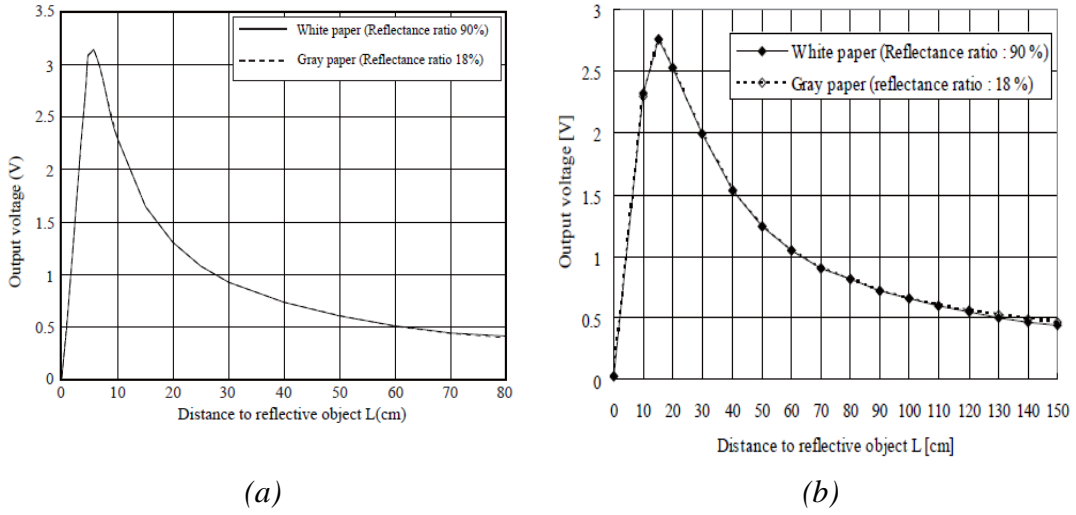


Figure 39: Distance Measuring Characteristics of (a) 10-80 cm (b) 20-150 cm types of IR Range Sensors

As it may be elicited from this table, the conversion from ADC output to analog output voltage is simply;

$$V_{ADC} = \frac{2^{10} \cdot V_{Analog}}{V_s} = \frac{1024 \cdot V_{Analog}}{5} \quad (24)$$

Converting from the the analog output voltage to a distance is a little bit more complicated, because a) the relationships between analog output voltage and distance is not linear, and b) the look-up table provides a coarse sample of points on the curve in Figure 39. *MATLAB* has a *polyfit* function to fit a curve to the values in the look-

up table and a *polyval* function to interpolate a point on that fitted curve. The combination of these two functions can be use to approximate a distance based on the analog output voltage.

Table 4: A Lookup Table for Interpolating a Distance from the Analog and Digital Output Voltages

Distance (cm)	Front Sensors		Side Sensors	
	Voltage (V)	ADC Out	Voltage (V)	ADC Out
10	2.40	480	---	---
20	1.32	276	2.50	515
30	0.92	192	1.95	392
40	0.71	152	1.50	300
50	0.58	132	1.20	242
60	0.52	105	1.02	202
70	0.45	88	0.85	185
80	0.42	80	0.75	170
90	---	---	0.65	152
100	---	---	0.58	135
110	---	---	0.54	120
120	---	---	0.50	110
130	---	---	0.48	100
140	---	---	0.46	91
150	---	---	0.42	87

It is important to note that the IR proximity sensor on the actual robot will be influenced by ambient lighting and other sources of interference. For example, under different ambient lighting conditions, the same analog output voltage may correspond to different distances of an object from the IR proximity sensor. This effect of ambient lighting (and other sources of noise) is not modelled in our *Simulink* model, but will be apparent on the actual hardware.

4.2.2 Ultrasonic Range Sensors

The orientation (relative to the body of the robot, as shown in figure 10 in section 3.1.4) of ultrasonic sensors 1 through 4 is 90^0 , 90^0 , -90^0 , -90^0 , respectively. Our ultrasonic range sensors are effective in the range of 2 - 400 cm only with the resolution of 3 mm.

As shown in timing diagram (Figure 40), it only needs to supply a short $10\ \mu s$ pulse to the *trigger* input to start the ranging, and then the module will send out an 8 cycle *burst* of ultrasound at 40 kHz and raise its echo. The echo is a distance object that is pulse width and the range in proportion. We can calculate the range through the time interval between sending *trigger* signal and receiving *echo* signal. On the other hand, in order to prevent trigger signal to the echo signal, it is better to use over 60 ms measurement cycle. The speed of ultrasonic is 343 m/s. To measure a distance from 1 meter, it's need round about 6 ms (actually there are two meters, one meter to the destination and one meter back). So we need a special time for one measurement cycle and we are not able to make as many measurements per second as we want.

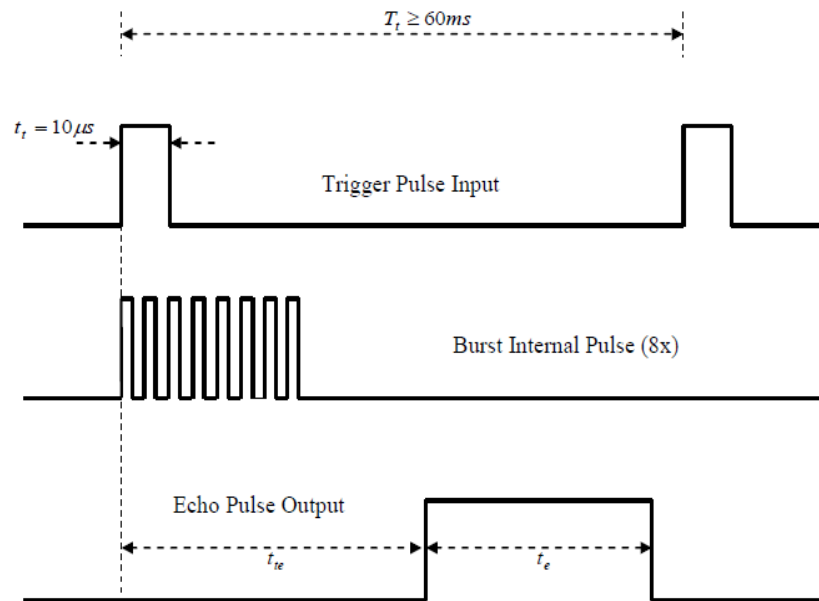


Figure 40: Timing Diagram of Trigger, Burst & Echo Pulses of Ultrasonic Sensor

Now in order to calculate the range, we should measure the time interval between sending *trigger* signal and receiving *echo* signal (t_{te}) or it's equivalent (t_e). The needed range in millimeters will be calculated simply as; $range = \frac{t_e}{5.82}$, when t_e is in microseconds. We can calculate the range as a subsystem in our *Simulink* model, which has been illustrated in Figure 41. The received *echo* signal triggers the *Digital Clock* on both falling and rising edges.

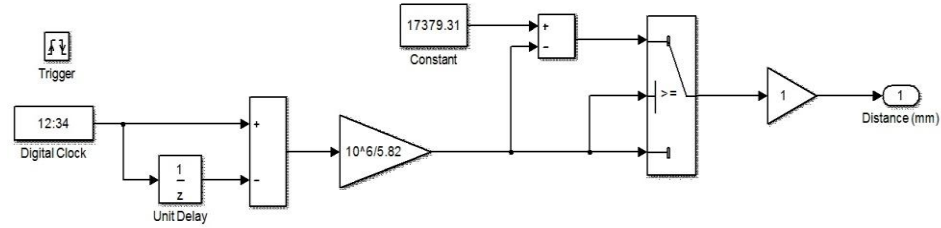


Figure 41: Subsystem Model to obtain the Range from the Echo Signal

As it mentioned above, our trigger signal has a pulse width of 10 microseconds. There is no problem alone, but this pulse width causes a major difficulty in the whole of the model. The fundamental sample time of the model drops out from 10 milliseconds to 10 microseconds, and it means the model freezes indeed. Both trigger and echo signals should be applied and measured as our solution, by the codes which are uploded on the slave microcontroller board (*SCMB*). Part 2 of appendix 8 includes this code. As it is shown in table 2, the needed data is transfetred from TX2 on *SCMB* to RX2 on *MMCB*.

4.2.3 Compass

The *LSM303D*, ultra compact high performance e-Compass, 3D accelerometer and 3D magnetometer module, of the *MinIMU-9* board has an I²C interface which accesses nine independent rotation, acceleration, and magnetic measurements that can be used to calculate the sensor's absolute orientation. As shown in Figure 42, the respective axes of the two chips are aligned on the board to facilitate these sensor fusion calculations.

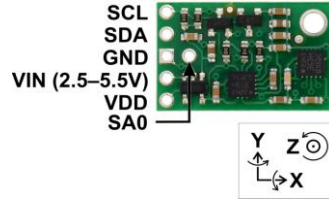


Figure 42: The Axes of the Compass Board

What we need to use in the robot as heading, is just the orientation along the z axis. Part 3 of appendix 8 includes a program which uses readings from the accelerometer and magnetometer to calculate a tilt-compensated compass heading in degrees relative to a default vector. As it is shown in Table 2 of this chapter, the proper data is transferred from TX3 on *SMCB* to RX3 on *MMCB*. In order to calibrate the heading, it has been used another program as illustrated in appendix 9. This program gives us the minimum and maximum of the readings from each magnetometer axis, which have been later entered in the related part.

4.2.4 Differential Wheel Drive

Since the robot has a differential wheel drive (i.e., is not a unicycle), it has to be controlled by specifying the angular velocities of the right and left wheel (v_r, v_l), instead of the linear and angular velocities of a unicycle (v, ω). These velocities are computed by a transformation from (v, ω) to (v_r, v_l). Recall that the dynamics of the unicycle are defined as;

$$\begin{aligned}\dot{x} &= v \cdot \cos(\varphi) \\ \dot{y} &= v \cdot \sin(\varphi) \\ \dot{\varphi} &= \omega\end{aligned}\quad (25)$$

The dynamics of the differential drive are defined as;

$$\begin{aligned}\dot{x} &= \frac{R}{2} \cdot (v_r + v_l) \cdot \cos(\varphi) \\ \dot{y} &= \frac{R}{2} \cdot (v_r + v_l) \cdot \sin(\varphi) \\ \dot{\varphi} &= \frac{R}{L} \cdot (v_r - v_l)\end{aligned}\quad (26)$$

where R is the radius of the wheels and L is the distance between the wheels.

It is important to note that if the robot is controlled or move at maximum linear velocity, it is not possible to achieve any angular velocity, because the angular velocity of the wheel will have been maximized. Therefore, there exists a tradeoff between the linear and angular velocity of the Robot; *the faster the robot should turn, the slower it has to move forward.*

4.2.5 Motor Quadrature Encoders

Each of the motors is integrated with a quadrature encoder that increments or decrements a tick counter depending on whether it is moving forward or backwards, respectively. The encoders may be used to infer the relative pose of the robot. This inference is called *Odometry*. The relevant information needed for odometry is the radius of the wheel (5 cm), the distance between the wheels (47 cm), and the number of count per revolution (*CPR*) of the wheel, which is calculated using by the encoders tick per revolution (*TPR*=16).

$$CPR = 4 \cdot TPR \cdot GearRatio = 4 \times 16 \times 131.25 = 8400 \quad (27)$$

As it has been written in the motor's document by the producer, it's no-load speed is 80 rpm. But actually, the motor speeds up to 86 rpm under load! So the number of count per second (*CPS*) will be;

$$CPS = CPR \cdot \frac{rpm}{60sec} = 8400 \times \frac{86}{60} = 12040 \quad (28)$$

In order to handling the encoder input, initially it was used the standard *DigitalRead* block of *Arduino* in *Simulink* environment. Everything worked fine when the motors turned manually, the interrupts kicked in and the counters were correctly incremented / decremented. However, when the motors powered up the *Arduino* block froze. The main problem is this; The quadrature encoders are too fast for *Arduino*. In other words, the amount of *CPS* is too big to be counted by *DigitalRead* block.

Others suffered from the same experience [56]. One suggested solution is the use of dedicated quadrature decoder chips well known as *Quadrature Encoder Interface*

(*QEI*) like the *LS7266R1*. More information about its usage in a robot project can be found in the thesis *Design of a Wireless Control System for a Laboratory Planetary Rover* [57]. Another student, named *Sertaç Emre Kara*, in our university has solved this problem using by a high performance signal controller, *dsPIC 30F4011*, and presented it in his thesis [58]. Actually this microchip has a *QEI* module, which provides the interface to incremental encoders for obtaining mechanical position data. If we solved our problem using by this microchip, we had to use two of them.

Fortunately no dedicated microchip hardware is necessary, at least in such cases. It turns out that the *Arduino* function *digitalRead* comes with a lot of overhead and is correspondingly slow. We could solve this problem and avoid the use of *digitalRead* block, by directly reading from *Atmel* (the Microprocessor of the *Arduino Mega 2560*) ports rather than going through the *Arduino* library.

Two encoders were connected to the *External Interrupts* pins on the *Arduino Mega 2560*. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. The allowed pins are 2, 3, 18, 19, 20 and 21 on the board. A *MATLAB Function Block* attaches at initialization time, the interrupt service routines, written in *C* code as illustrated in appendix 10, to the two pins to which the encoder is connected. After that, when the encoder rotates the interrupts service routines update the encoder position. The encoder position is then set as output of the block every time (sample time) the block is executed.

4.3 Microcontroller Board– MATLAB Simulink Connection

In order to facilitate the design, test, and verify our system that combines hardware components and software algorithms, it was preferred to connect *MATLAB Simulink* with the microprocessor *Arduino Mega 2560* (Figure 43). *MATLAB Simulink* supports data-driven control design and provides an environment for: Data Acquisition, System Identification, Control Design and Real-time Testing.

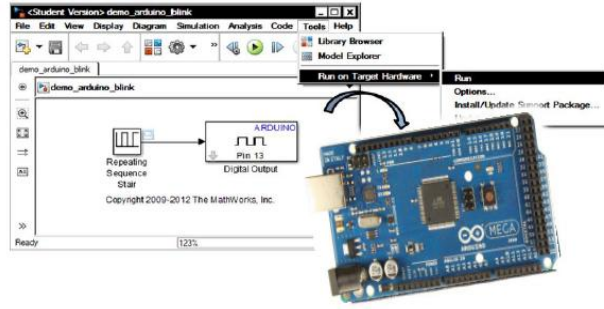


Figure 43: MATLAB Simulink and Aruino Mega 2560

The *Simulink* built-in support for the *Arduino* platform includes:

- Library of *Simulink* blocks that connect to *Arduino* I/O, such as digital input and output, analog input and output, serial receive and transmit, and servo read and write.
- Interactive parameter tuning and signal monitoring of applications running on the *Arduino Mega 2560*.
-

4.4 Control and Model of the System

As it mentioned in section 4.2.2 of this chapter, the robot has to be controlled by specifying the angular velocities of the right and left motor (v_r, v_l). However in practice, we can not get the expected results as it is calculated theoritically. For this purpose, we have to employ the controllers to minimize between the practical and theoritically results. In our case, by using a *PID* controller, we intend to make the actual motor speed match the desired motor speed.

PID algorithm calculates, necessary speed changes to get the desired speed. This is done by creating a cycle where the motor speed is continously being checked against the desired speed. The speed power level is always set based on what is needed to achieve the correct results. In order to start the speed control tasks, we ought first to model our motors.

4.5 Plant Model

Generally we have two approaches to modeling our plant; first-principle modeling which requires knowledge of math and dynamic of the DC motors and physical

structure, and data-driven modeling which is based on the input-output data measurement [59]. The dynamics of a DC servo motor consists of the dynamics of mechanical and electrical parts. In addition, although it is feasible to compute some needed parameters like electromotive force constant (K_b) or motor torque constant (K_t), but measuring of some other parameters like moment of inertia of the rotor (J) and or motor viscous friction constant (b), is a time-consuming process and out of the sketch of this thesis. So we decided to apply the data driven modeling, because the first principle can practically get challenging. The workflow of data driven modeling for our plant is presented in Figure 44.

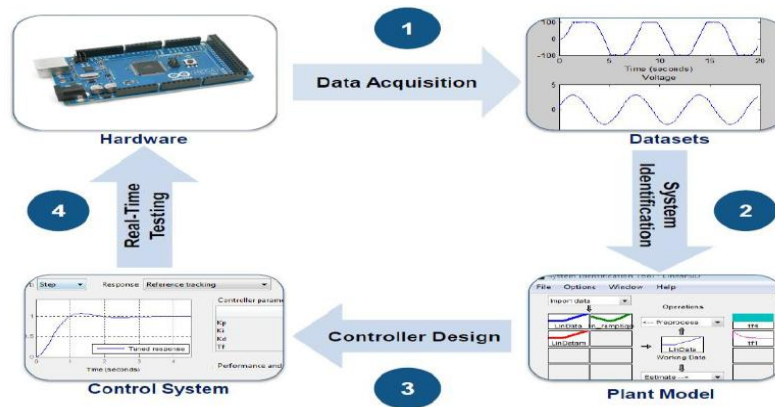


Figure 44: Workflow of the Data-Driven Modeling

4.5.1 Data Acquisition

To acquire data from the plant, it should setup a hardware as shown in Figure 45. Then the executable *Simulink* model is run on target hardware. But in order to validate the measured data, and close up the model and real plant, we prefer to measure the outputs when the motors are under load. With this work, in addition to modeling of morors, we have modeled the whole system.

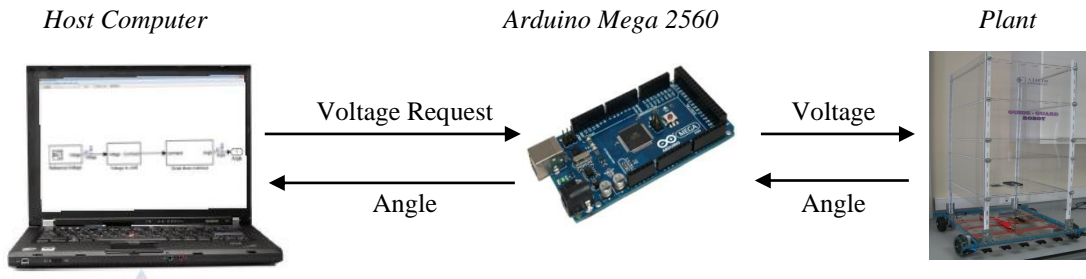


Figure 45: Data Acquisition Hardware Setup

As it is shown in Figure 46, the input signal, is set as the desired speed and the position (angle in degree) of the motor is measured as the output. All of these data is saved in the workspace as the time domain signals and used to system identification.

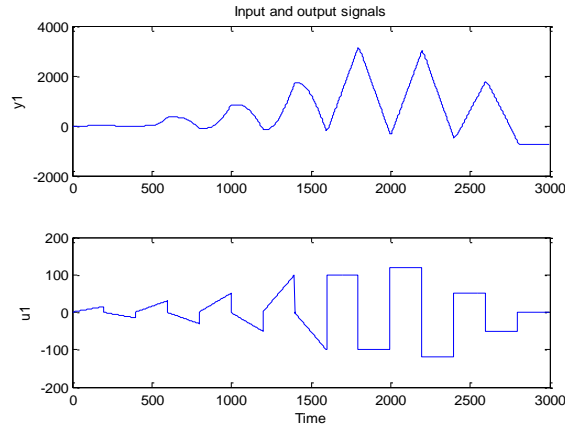


Figure 46: Time Plot of Input and Output Signals

4.5.2 System Identification

Using by the system identification toolbox in *MATLAB Simulink* environment, we tried to construct mathematical model of dynamic system from the above mentioned measured input-output data. We used time-domain input-output data to identify discrete-time transfer function and non-linear models. Table 5 demonstrates five transfer function models, four Nonlinear *ARX* type models and two Nonlinear *Hammerstein-Wiener* type models, which have been estimated.

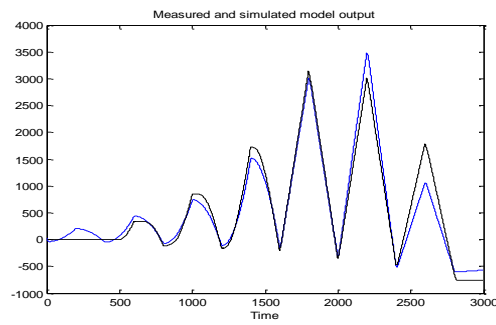
Considering, fit to estimation data, the *Transfer function1* and *Nonlinear H-W1* models were chosen in order to design a proper controller. The *Transfer Function1* has a discrete function form as;

$$TransferFunction1 = \frac{Output}{Input} = \frac{0.02638(\pm 0.003287)}{z^2 - 1.843(\pm 0.01973)z + 0.8431(\pm 0.01973)} \quad (29)$$

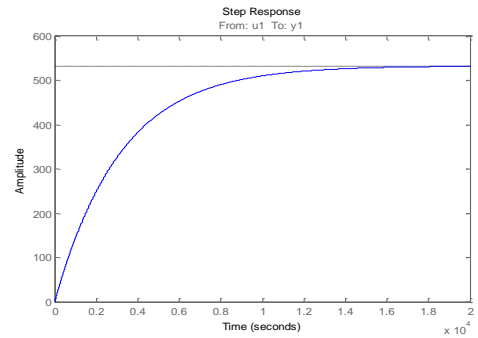
This transfer function can be easily implemented to control purposes using by a discrete type transfer function in *Simulink* environment. With the same goal, the second nonlinear *Hammerstein-Wiener* type model can be employed by an *IDNLHV* model block, which simulate *Hammerstein-Wiener* model for time-domain input and output data in *Simulink* software. Figure 47 demonstrate measured and simulated model output time signals, and step response of these two estimators.

Table 5: The Results of Plant Model using by Transfer Function, Nonlinear ARX and Hammer-Wiener type Estimators

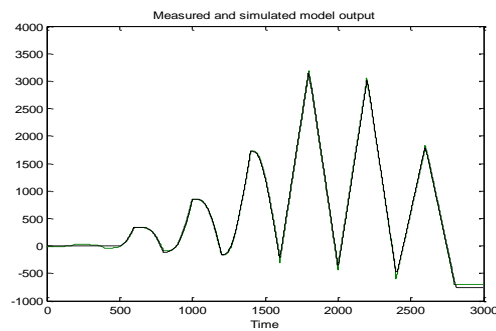
Model	Number of Poles	Number of Zeros	Fit to Estimation Data
Transfer Function 1	2	1	76.31%
Transfer Function 2	2	2	76.4%
Transfer Function 3	3	1	76.38%
Transfer Function 4	3	2	76.39%
Transfer Function 5	3	3	76.4%
	Number of Input Terms	Number of Output Terms	Fit to Estimation Data
Nonlinear ARX 1	2	2	32%
Nonlinear ARX 2	4	2	72%
Nonlinear ARX 3	3	3	13%
	Number of Input Units	Number of Output Units	Fit to Estimation Data
Nonlinear H-W 1	10	10	97.37%
Nonlinear H-W 2	12	12	96.59%



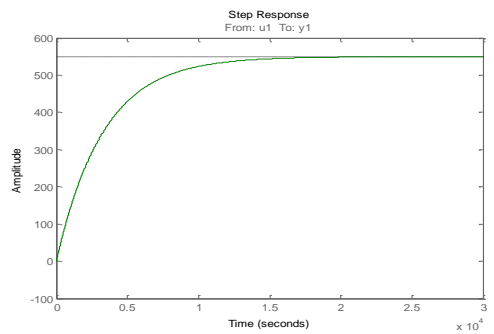
(a)



(b)



(c)



(d)

Figure 47: Measured and Simulated Model Output Time Signals of (a) Transfer Function1 (b) Nonlinear H-W1 Estimators, and Step Response of (c) First Estimator (d) Second Estimator

4.5.3 Controller Design

With employing the two above mentioned models, we start to design a proper velocity *PID* controller. The aim in using the *PID* controller is to make the actual motor speed match the desired motor speed. This controller is more feasible, when it is compared with other controllers. To design a discrete *PID* control, there are some critical points, which we ought to consider them carefully [60]. The most critical point is the choice of the sampling period. Since the nature consists of analog signals, most plant transfer function are modeled in continuous time. In order to implement a discrete *PID* controller, the designer should take samples from the continuous time error signal. However, these samples should be taken frequently enough in order not to miss system dynamics.

The TransferFunction 1 as it has been calculated in equation (29), can be written in this format;

$$TransferFunction1 = \frac{0.02638}{(z - 0.8362) \cdot (z - 1.007)} \quad (30)$$

The rise time for the output will approximately be;

$$t_r = \frac{4}{|Dominant Pole Location|} = \frac{4}{0.8362} = 4.78 \text{sec} \quad (31)$$

Our sampling period should be much less than this rising time. 0.1 sec, is a reasonable sampling time, which is picked up.

First without any controller, the step responses of the models are obtained as demonstrated in Figure 48. The *Root-Locus* of the open-loop system using by the first model, is illustrated in Figure 49.

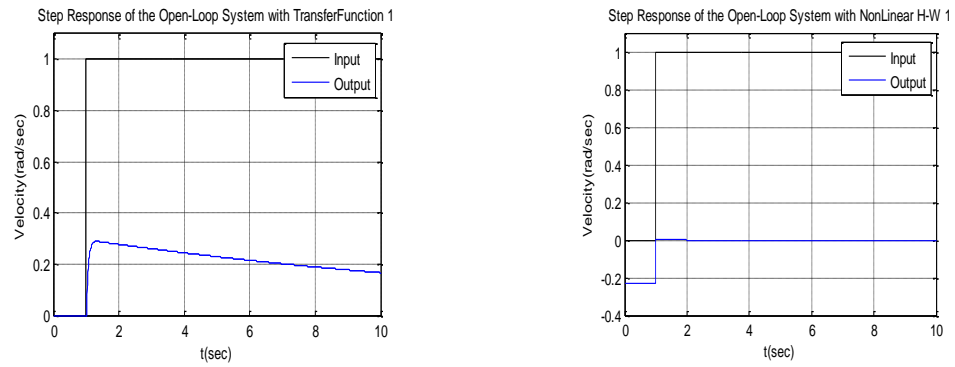


Figure 48: Step Responses of the Two Models in a Open-Loop System

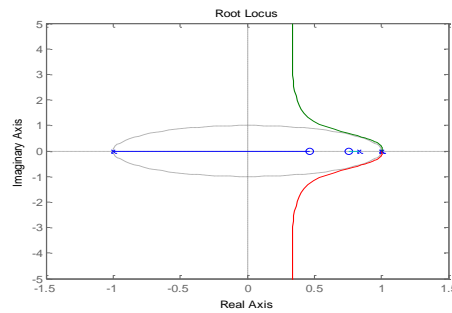


Figure 49: Root-Locus of the Open-Loop System

Then a closed-loop system is constituted using by a discrete *PID* as illustrated in Figure 50. The tuning of *PID* was worked out automatically in the block. Just, in order to improve the shape of response, some tuning was manually done in response time and transient behavior parameters. The final tuned response was caught by this values; $P = 73.7$, $I = 112.9$, $D = -0.8$

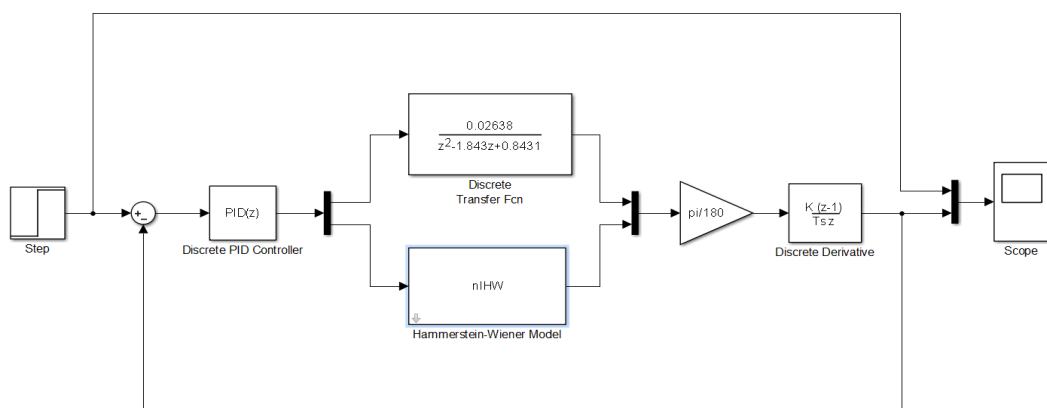


Figure 50: Close-Loop System using by Discrete PID

The step responses of the two models are obtained as demonstrated in Figure 51, and the Root-Locus of the close-loop system, is illustrated in Figure 52.

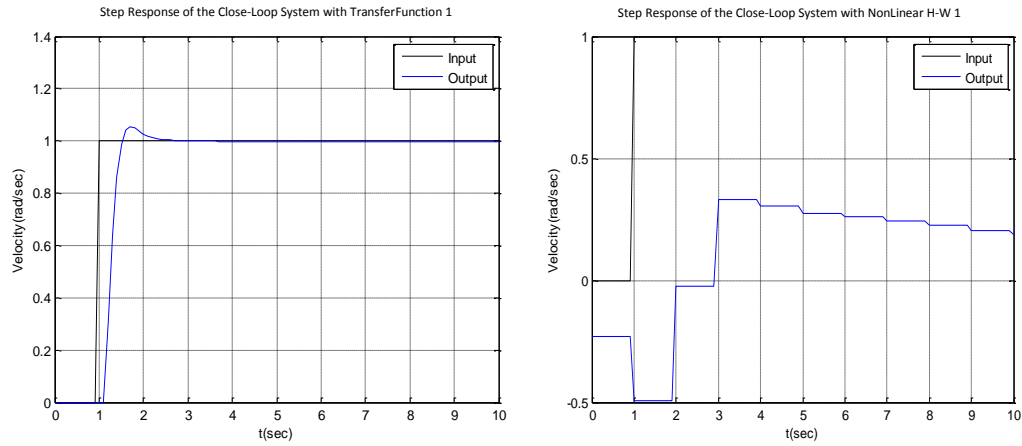


Figure 51: Step Responses of the two Models in a Close-Loop System

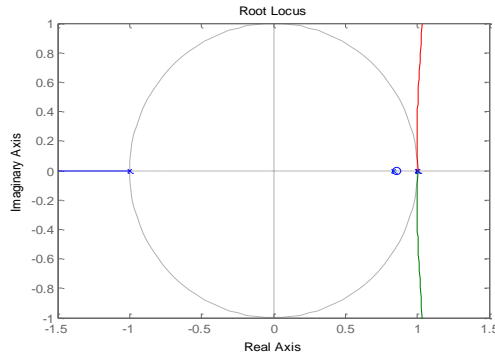


Figure 52: Root-Locus of the Close-Loop System

Despite of the first model, the second model does not behave as well as the first one. The reason is the unstability of the close-loop system, employing with *Hammerstein-Wiener* model. As it clearly be seen from the above figure, the pole at -1 goes to infinity, which is the reason of instability. To solve it, pole placement should be done. In real plant, the addition of pole can be done by adding a proper capacitor at the end of the *PID* controller.

4.5.4 Real-Time Testing

Now that's turn to test the close-loop system in real-time. Using by the same hardware setup, which was employed to data acquisition described in section 4.5.1, the real-time testing on the real motor was done. In the first case, the test was done

employing by the same *PID*, which was tuned in the design phase. Figure 53 (a) demonstrates the step response in this case. Then it is tried to catch better response shape, which was lead to a response as illustrated in Figure 53 (b). The value of *PID* parameters in this case was; $P=1.7$, $I=0.5$, $D=0.005$.

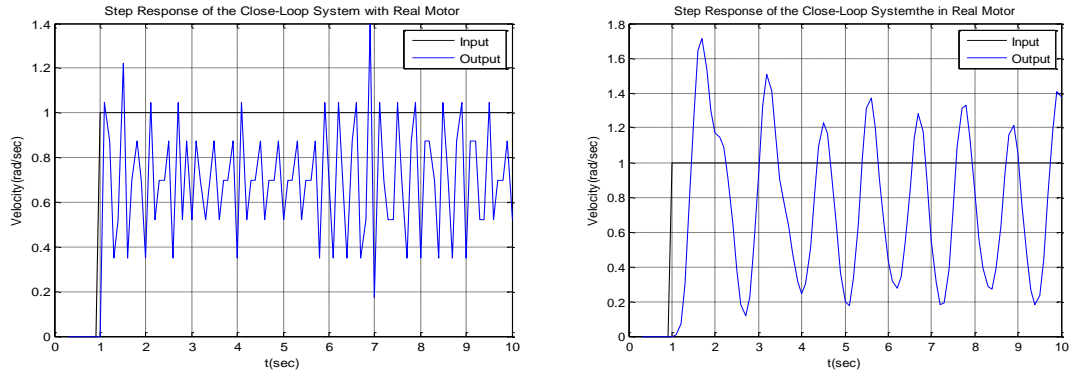


Figure 53: Step Responses of the Real Motor in a Close-Loop System with (a) Design Phase's PID Parameters (b) Mentioned PID Parameters

Then we got the step response of the open-loop system with the real motor in two conditions; no-load and under-load. As they can clearly be seen from Figure 54, our open-loop system behaves well as required. We will employ an open-loop system to velocity control of the motors, and let them to be controlled by a strong position controller.

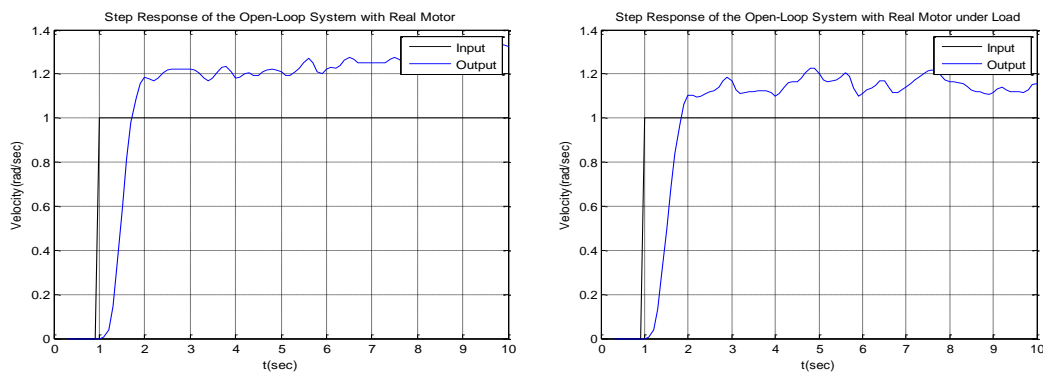


Figure 54: Step Responses of the Real Motor in an Open-Loop System in (a) No-Load (b) Under-Load Conditions

4.6 Architecture of Differential-Drive Trackers

In order to design a tracker for our differential-drive robot, recall the single plan, described in section 3.2.5.5 of chapter 3. Let the output from the planner be u . This reference trajectory is applied to the tracker and the angular velocities of the right and left motor are achieved from it.

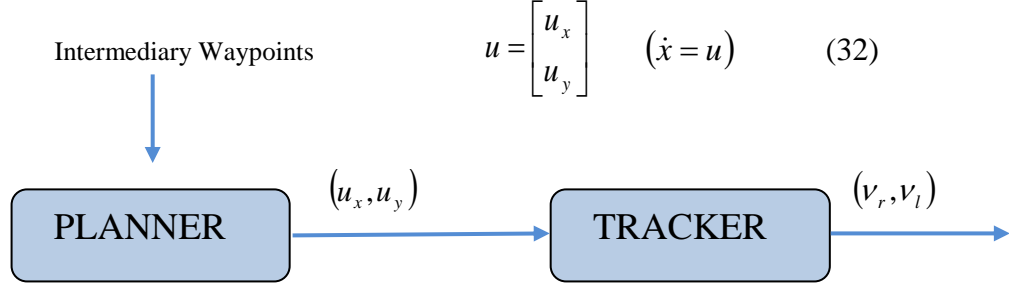


Figure 55: Primary Differential-Drive Architectur

$$\begin{aligned} \varphi_d &= a \tan\left(\frac{u_y}{u_x}\right) & v_r &= \frac{2v + \omega L}{2R} \\ \omega &= PID(\varphi_d - \varphi) & v_l &= \frac{2v - \omega L}{2R} \\ v &= \sqrt{u_x^2 + u_y^2} & & \end{aligned} \quad (33)$$

Now what if we ignored the orientation and picked a different point (\tilde{x}, \tilde{y}) , on the robot as the point we care about?

$$\text{Dynamics of old point} \quad \begin{cases} \dot{x} = v \cdot \cos \varphi \\ \dot{y} = v \cdot \sin \varphi \\ \dot{\varphi} = \omega \end{cases} \quad (34)$$

$$\text{Position of new point} \quad \begin{cases} \tilde{x} = x + l \cdot \cos \varphi \\ \tilde{y} = y + l \cdot \sin \varphi \end{cases} \quad (35)$$

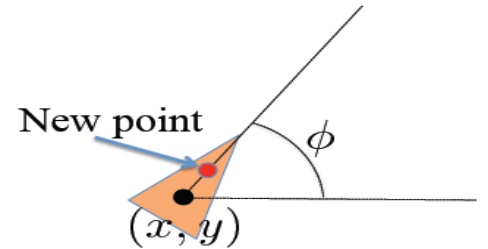


Figure 56: Picking a New Point

$$\text{Dynamics of new point} \begin{cases} \dot{\tilde{x}} = v \cdot \cos \varphi - l \cdot \omega \cdot \sin \varphi \\ \dot{\tilde{y}} = v \cdot \sin \varphi + l \cdot \omega \cdot \cos \varphi \end{cases} \quad (36)$$

Now let's assume that we can control the new point directly;

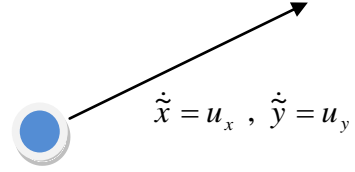


Figure 57: Control Input Vector of New Point

$$\begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} \times \begin{bmatrix} v \\ l\omega \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1/l \end{bmatrix} \times R(-\varphi) \times \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad (37)$$

So the differential-drive architecture will be:

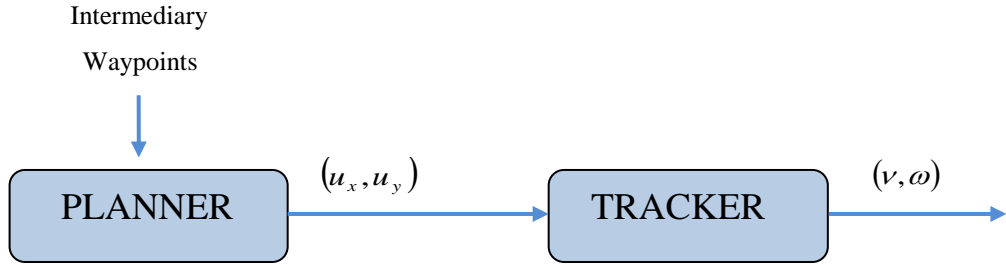


Figure 58: Final Differential-Drive Architecture

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1/l \end{bmatrix} \times R(-\varphi) \times \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad (38)$$

4.7 Odometry

The odometry for the robot is implemented such that as the robot moves around, its pose, (x, y, φ) is estimated based on how far each of the wheels have turned. The general idea behind odometry is to use wheel encoders to measure the distance the wheels have turned over a small period of time, and use this information to approximate the change in pose of the robot.

The pose of the robot is composed of its position (x, y) and its orientation φ , on a two-dimensional plane. The currently estimated pose is stored in a variable, which bundles x , y and φ . If D_r and D_l is the distance which right and left wheels have been turned;

$$\begin{cases} x = x_{initial} + \frac{D_r + D_l}{2} \cdot \cos \varphi \\ y = y_{initial} + \frac{D_r + D_l}{2} \cdot \sin \varphi \\ \varphi = \varphi_{initial} + \frac{D_r - D_l}{L} \end{cases} \quad (39)$$

4.8 Architecture of Differential-Drive Planners

Now it is time to handle the differential-drive planner more practical. As it is clear in Figure 43, the intermediary waypoints enter to the single plan (*Hybrid Automata*), consisting of all behaviors, as inputs and the reference trajectory are produced as outputs (u).

Here we study carefully, how the reference trajectory is computed in each behavior.

4.8.1 Go-to-Goal Behavior (GTG)

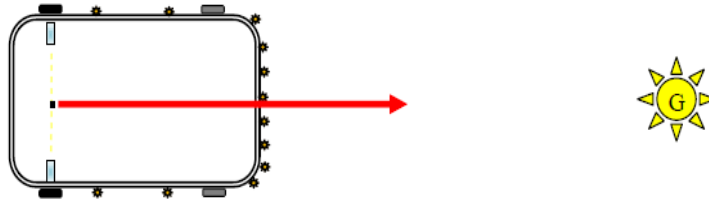


Figure 59: Go-to-Goal Vector

The strategy for this behavior that is used is as follows;

1. Calculate the heading (angle), φ_g , to the goal location (x_g, y_g) , Let u be the vector from the robot located at (x, y) to the goal located at (x_g, y_g) , then φ_g is the angle u makes with the x-axis (positive φ_g is in the counterclockwise direction). The vector u can be expressed in terms of its x-component, u_x , and its y-component u_y . Use these two components and the *atan2* function to compute the angle to the goal, φ_g .
2. Calculate the error e_φ between φ_g and the current heading of the robot, φ . Make sure to use *atan2* and or other functions to keep the error between $[-\pi, \pi]$.
3. Calculate the proportional, integral, and derivative terms for the *PID* regulator that steers the robot to the goal. Now, we need to tune our *PID* gains to get a fast settle time (φ matches φ_g within 10% in three seconds or less) and there should be little overshoot (maximum φ should not increase beyond 10% of the reference value φ_g).
4. Ensure that the robot steers with an angular velocity ω , even if the combination of v and ω exceeds the maximum angular velocity of the robots motors. This issue will be comprehensively described in section 4.9.

Since our *PID* controllers focus more on steering than on controlling the linear velocity, we want to prioritize ω over v in situations, where we cannot satisfy ω with the motors. In fact, we will simply reduce v until we have sufficient headroom to achieve ω with the robot.

4.8.2 Avoid-Obstacles Behavior (AO)

To implement this behavior, the IR sensors allow us to measure the distance to obstacles in the environment, but we need to compute the points in the world to which these distances correspond.

Strategy for obstacle avoidance that we will use is as follows;

1. Transform the IR distances to points in the world.
2. Compute a vector to each point from the robot; u_1, u_2, \dots, u_{12} .
3. Weigh each vector according to their importance, $\alpha_1 \cdot u_1, \alpha_2 \cdot u_2, \dots, \alpha_{12} \cdot u_{12}$. For example, the front sensors are typically more important, for obstacle avoidance while moving forward.

4. Sum the weighted vectors to form a single vector,

$$u_{AO} = \alpha_1 \cdot u_1 + \alpha_2 \cdot u_2 + \dots + \alpha_{12} \cdot u_{12}$$

5. Use this vector to compute a heading and steer the robot to this angle.

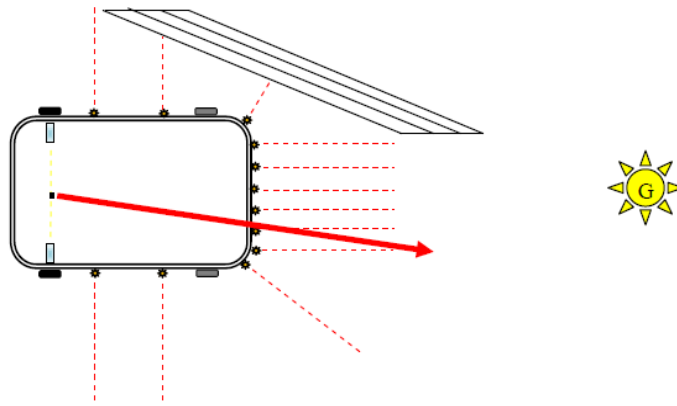


Figure 60: Avoid-Obstacles Vector

This strategy will steer the robot in a direction with the most free space (i.e. it is a direction away from obstacles). For this strategy to work, we will need to implement three crucial parts of the strategy for the obstacle avoidance behavior;

- Transform the IR distance measured by each sensor to a point in the reference frame of the robot.

A point p_i that is measured to be d_i meters away by sensor i can be written

as the vector $v_i = \begin{bmatrix} d_i \\ 0 \end{bmatrix}$ in the reference frame of sensor i . We first need to

transform this point to be in the reference frame of the robot. To do this

transformation, we need to use the pose (location and orientation) of the

sensor in the reference frame of the robot, $(x_{s_i}, y_{s_i}, \theta_{s_i})$. The transformation is

defined as $v'_i = R(x_{s_i}, y_{s_i}, \theta_{s_i}) \times \begin{bmatrix} v_i \\ 1 \end{bmatrix}$, where R is known as the transformation

matrix that applies a translation by (x, y) and a rotation by θ ;

$$R_{(x,y,\theta)} = \begin{bmatrix} \cos \theta & -\sin \theta & x \\ \sin \theta & \cos \theta & y \\ 0 & 0 & 1 \end{bmatrix} \quad (40)$$

- Transform the point in the robot's reference frame to the world's reference frame. A second transformation is needed to determine where a point p_i is located in the world that is measured by sensor i . We need to use the pose of the robot, (x, y, φ) to transform the robot from the robot's reference frame to the world's reference frame. This transformation is defined as:

$$v''_i = R(x, y, \varphi) \times v'_i \quad (41)$$

- Use the set of transformed points to compute a vector that points away from the obstacle. The robot will steer in the direction of this vector and successfully avoid the obstacle.

4.8.3 Blending and Hard Switches (AO-GTG)

Now we are ready to test two arbitration mechanisms: Blending and Hard Switches. This arbitration between the two controllers will allow the robot to drive to a goal, while not colliding with any obstacles on the way.

1. Implement a simple control for the linear velocity v , as a function of the angular velocity ω . So far, we have implemented controllers that either steer the robot towards a goal location, or steer the robot away from an obstacle. In both cases, we have set the linear velocity, v , to a constant value. While this approach works, it certainly leaves plenty of room for improvement. Now we want to improve the performance of both the go-to-goal and avoid-obstacles behavior by dynamically adjusting the linear velocity based on the angular velocity of the robot.

As mentioned in section 4.2.4, with a differential drive robot, we cannot, for example, drive the robot at the maximum linear and angular velocities. Each motor

has a maximum and minimum angular velocity; therefore, there must be a trade-off between linear and angular velocities: linear velocity has to decrease in some cases for angular velocity to increase, and vice versa.

However, one could improve the above strategy by letting the linear velocity be a function of the angular velocity and the distance to the goal (or distance to the nearest obstacle). For example, the linear velocity in the go-to-goal controller could be scaled by ω and the distance to the goal, such that the robot slows down as it reaches the goal.

2. Combine the go-to-goal and avoid-obstacle controllers into a single controller that blends the two behaviors (*Blending*). But, one important piece is missing. u_{GTG} is a vector pointing to the goal from the robot, and u_{AO} is a vector pointing from the robot to a point in space away from obstacles. These two vectors need to be combined (blended) in some way into the vector u_{GTG} , which should be a vector that points the robot both away from obstacles and towards the goal.

The combination of the two vectors into u_{AO-GTG} should result in the robot driving to a goal without colliding with any obstacles in the way. To do this, weigh each vector according to their importance, and then linearly combine the two vectors into a single vector, u_{AO-GTG} . For example;

$$\alpha = 0.75$$

$$u_{AO-GTG} = \alpha \cdot u_{GTG} + (1 - \alpha) \cdot u_{AO} \quad (42)$$

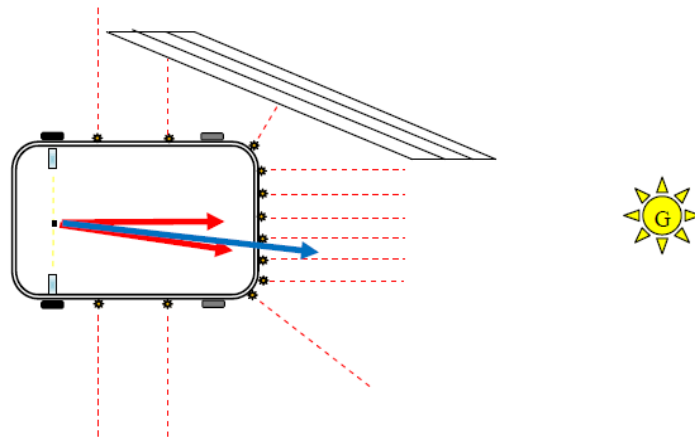


Figure 61: Blended Vector

As you see, the *GTG* behavior is stronger than the *AO* behavior, but that may not be the best strategy. α needs to be carefully tuned (or a different weighted linear combination needs to be designed) to get the best balance between *GTG* and *AO*. To make life easier, we consider using the normalized versions of u_{AO} and u_{GTG} .

3. Implement the switching logic that switches between the *GTG* controller and the *AO* controller, such that the robot avoids any nearby obstacles and drives to the goal when clear of any obstacles. In other words, instead of executing both *AO* and *GTG* simultaneously, we can only execute one controller at a time, but switch between the two controllers whenever a certain condition is satisfied.

In order to switch between different controllers (or states), we can define a set of events. These events can be checked to see if they are true or false. The idea is to start of in some state (wich runs a certain controller), check if a particular event has occurred, and if so, switch to a new controller.

The tools that we should will need to implement the switching logic;

- Four events can be checked, where name is the name of the state:
 - ‘*at-obstacle*’ checks to see if any of front sensors detect an obstacle at a distance less than $d_{obstacle}$.
 - ‘*at-goal*’ checks to see if the robot is within d_{stop} of the goal location.
 - ‘*unsafe*’ checks to see if any of the front sensors detect an obstacle at a distance less than d_{unsafe} .
 - ‘*Eunsafe*’ checks to see if any of the front and side sensors detect an obstacle at a distance less than $d_{Eunsafe}$.
- A function switches between the states/controllers. There currently are four possible values that name can be:
 - ‘*go-to-goal*’ for the *GTG* controller.
 - ‘*avoid-obstacles*’ for the *AO* controller.
 - ‘*ao-and-gtg*’ for the blending (*AO-GTG*) controller.

Implement the logic for switching to *AO*, when *at-obstacle* is true, switching to *GTG* when *obstacle-cleared* is true, and switching to stop when *at-goal* is true.

4. Improve the switching arbitration by using the blended controller as an intermediary between the *GTG* and *AO* controller.

The blending controllers advantage is that it (hopefully) smoothly blends *GTG* and *AO* together. However, when there are no obstacle around, it is better to purely use *GTG*, and when the robot gets dangerously close, it is better to only use *AO*. The switching logic performs better in those kinds of situations, but jitters between *GTG* and *AO* when close to a goal. A solution is to squeeze the blending controller in between the *GTG* and *AO* controller.

Implement the logic for switching to *AO-GTG*, when *at-obstacle* is true, switching to *GTG* when *obstacle-cleared* is true, switching to *AO* when *unsafe* is true, and switching to stop when *at-goal* and or *Eunsafe* is true.

4.8.4 Follow-Wall Behavior (FW)

The strategy for this behavior, which will aid the robot in navigating around obstacles, is as follows;

1. Compute a vector, u_{FW_t} that estimates a section of the obstacle (wall) next to the robot using the robot's right or left IR sensors. In Figure 62, this vector, u_{FW_t} is illustrated in red.

Suppose we want to follow an obstacle to the left of the robot, then we would use the left set of IR sensors (1-4). If we are following the wall, then at all times there should be at least one sensor that can detect the obstacle. So, we need to pick a second sensor and use the points corresponding to the measurements from these two sensors to form a line that estimates a section of the obstacle. In the Figure 62, sensors 2 and 3 are used to roughly approximate the edge of the obstacle. But what about corners?

Corners are trickier, because typically only a single sensor will be able to detect the wall. The estimate is off as one can see in the Figure, but as long as the robot isn't following the wall too closely, it will be ok.

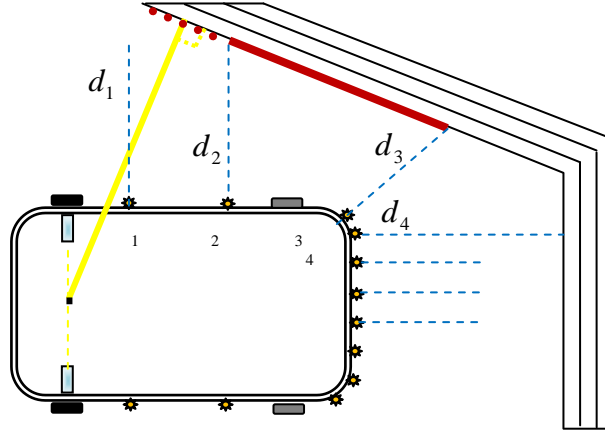


Figure 62: Two vectors in FW Behavior

An example strategy for estimating a section of the wall, is to pick the two sensors (from IR sensors 1-3) with the smallest measured distance. Suppose sensors 2 and 3 returned the smallest values, then let $p_2 = d_2$ and $p_3 = d_3$. A vector that estimates a section of the obstacle is $u_{FWt} = p_3 - p_2$. It is important that the sensor with smaller distance (in the example, sensor 3) is assigned to p_3 , and the sensor with the larger distance (in the example, sensor 2) is assigned to p_2 , because we want that the vector points in the direction that the robot should travel. The figures correspond to the above example strategy, but we may want to experiment with different strategies for computing u_{FWt} . A better estimate would make wall following safer and smoother, when the robot navigates around the corners of obstacles.

2. Compute a vector, u_{FWp} that points from the robot to the closest point on u_{FWt} . This vector, u_{FWp} is visualized as yellow line in the Figure 62 and can be computed using a little bit of linear algebra;

$$u'_{FWt} = \frac{u_{FWt}}{\|u_{FWt}\|}, u_p = \begin{bmatrix} x \\ y \end{bmatrix}, u_a = p_2$$

$$u_{FWp} = (u_a - u_p) - ((u_a - u_p)' \times u'_{FWt}) \times u'_{FWt} \quad (43)$$

A small technicality is that we are computing u_{FWp} as the vector pointing from the robot, to the closest point on u_{FWt} , as if u_{FWt} were infinitely long.

3. Combine the two vectors u_{FWt} and u_{FWp} , such that it can be used as a heading vector for a *PID* controller that will follow the wall to the right (or left) at some distance d_{FW} . u_{FWt} will ensure that the robot drives in a direction that is parallel to an edge on the obstacle, while u_{FWp} needs to be used to maintain a distance d_{FW} from the obstacle. One way to achieve this is;

$$u'_{FWp} = u_{FWp} - d_{FW} \cdot \frac{u_{FWp}}{\|u_{FWp}\|} \quad (44)$$

where u'_{FWp} is now a vector points towards the obstacle when the distance to the obstacle, $d \succ d_{FW}$ is near zero when the robot is d_{FW} away from the obstacle, and points away from the obstacle when $d \prec d_{FW}$.

All that is left, is to linearly combine u'_{FWt} and u'_{FWp} into a single vector u_{FW} that can be used with the *PID* controller to steer the robot along the obstacle at the distance d_{FW} .

$$u_{FW} = d_{FW} \cdot u'_{FWt} + (u_{FWp} - d_{FW} \cdot u'_{FWp}) \quad (45)$$

4.8.5 All Behaviors in a Single Navigation System

Now we are ready to combine the *GTG*, *AO*, and *FW* controllers into a full navigation system for the robot, which is done using by *State Flow* Toolbox in *MATLAB*. The robot will be able to navigate around a cluttered, complex environment without colliding with any obstacles and reaching the goal location successfully.

1. Implement the *progress-made* event, that will determine whether the robot is making any progress towards the goal.

By default, the robot is set up to switch between *AO* and *GTG* to navigate the environment. However, notice that the robot cannot escape the larger obstacle as it tries to reach the goal located.

The robot needs a better strategy for navigation. This strategy needs to realize that the robot is not making any forward progress and switch to *FW* to navigate out of the obstacle.

This event has been occurred till the following condition is true;

$$\left\| \begin{bmatrix} x - x_g \\ y - y_g \end{bmatrix} \right\| < d_{progress} - \varepsilon \quad (46)$$

Where ε gives a little bit of slack, and $d_{progress}$ is the closest the robot has progressed towards the goal.

2. Implement the *sliding-left* and *sliding-right* events, that will serve as a criterion for whether the robot should continue to *FW* or switch back to the *GTG* behavior.

While the lack of *progress-made* will trigger the navigation system into a *FW* behavior, we need to check whether the robot should stay in the *FW* behavior, or switch back *GTG*. We can check whether we need to be in the sliding mode (*FW*) by testing if $\sigma_1 > 0$ and $\sigma_2 > 0$, where;

$$\begin{bmatrix} u_{GTG} & u_{AO} \end{bmatrix} \times \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = u_{FW} \quad (47)$$

3. Implement the finite state machine that will navigate the robot to the goal located at (x_g, y_g) without colliding with any of the obstacles in the environment.

Now, we are ready to implement a *Finite State Machine (FSM)* that solves the full navigation problem. A *FSM* is nothing but a set of if/elseif/else statements that first check which state (or behavior) the robot is in, then based on whether an event (condition) is satisfied, the *FSM* switches to another state or stays in the same state.

The finite state machines, named here as *Supervisor*, switches between controllers and handles their inputs and outputs. Some of the logic that should be part of the *Supervisor* is;

- If *at-obstacle* is not true, then switch to state *GTG*.
- If *at-obstacle* is true, then switch to state *AO-GTG*.
- If *at-goal* and or *Eunsafe* is true, then switch to *Stop*.

- If *unsafe* is true, then switch to state *AO*.
- If in state *AO-GTG* and *at-obstacle* is true, and *progress-made* is not true, then check whether the robot needs to slide-left or slide-right. If so set *progress-point*, and switch to state *FW*.
- If in state *FW*, check whether *progress-made* is true and the robot does not need to *sliding-left* or *sliding-right*. If so, switch to state *AO-GTG*, otherwise keep *FW*.

4.9 Motor Limitations

Similar to all DC motors, our used motors have two critical limitations. The first limitation is that the motors have a maximum angular velocity, vel_{\max} , and the seconde limitation is that they stall at low speeds, vel_{\max} .

Suppose that we pick a linear velocity v that requires the motors to spin at 90% power. Then, we want to change ω from 0 to some value that requires 20% more power from the right motor, and 20% less power from the left motor. This is not an issue for the left motor, but the right motor cannot turn at a capacity greater than 100%. The results is that the robot cannot turn with the ω specified by our controller.

On the other hand, it is also true that the motors have a minimum speed before the robot starts moving. If not enough power is applied to the motors, the angular velocity of the motors remains at 0. Once enough power is applied, the motors spin at a speed vel_{\min} . We measured the motor's maximum and minimum forward angular velocity on the under-load real motor; $vel_{\min} = 0.7rad/sec$, $vel_{\max} = 9rad/sec$.

Small (v, ω) may not be achievable on the robot, so we have to scale up v to make ω possible. Similarly, if (v, ω) are both large, we have to scale down v to make ω possible.

It is maybe interesting how one deals with physical limitations on a mobile robot, like ours. This particular approach has an interesting consequence, which is that if $v > 0$ then v_r and v_l are both positive (and vise versa, if $v < 0$). Therefore, we often

have to increase or decrease v significantly to ensure ω even if it were better to make small adjustments to both v and ω . Our programming assignments to consider all of these limitations was coded in a *MATLAB* function, as illustrated in appendix 11. As with most of the components in these programming assignments, there are alternative designs with their own advantages and disadvantages.

4.10 Graphical User Interface

Graphical user interfaces (also known as *GUIs*) provide point-and-click control of software applications, eliminating the need to learn a language or type commands in order to run the application. A user interface is a graphical display in one or more windows containing controls, called components, that enable a user to perform interactive tasks. The user does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user does not need to understand the details of how the tasks are performed. Typically, *GUIs* wait for a user to manipulate a control, and then respond to each user action in turn.

4.10.1 Design

Based on our needs, and by the help of *Guide* in *MATLAB*, a graphical user interface was designed and created. Figure 63 shows a screenshot of our graphical user interface on the Notebook and the pertinent Matlab code has been illustrated in appendix 12.

As it may be cleared from the figure, we have an initial point, four intermediary and twenty two final goal points, which have been introduced in brown, blue and yellow *Push-Buttons* respectively. The user select the intended location by pushing the related bottom as listed in the displayed menu. By pushing it the selected location is displayed on the top of the menu. Then press the *Connect & GO* bottoms respectively. At the end the *STOP* bottom should be pressed.



Figure 63: A Screenshot of the Graphical User Interface on the Notebook

4.10.2 Bridging GUI and Microcontroller

In order to data transfer from *GUI* to the microcontroller and run there, the robot's controllers (as illustrated in Figure 38, chapter 3), a serial communication is needed. First we had ideated; this communication is simply done by *USB* connection between notebook and microcontroller. In other words, the data is serially transmitted by *GUI* and a receiver catches the data. The *Arduino's Serial Receive* block gets one byte of data per sample period from the receive buffer of the specified serial port. But this idea does not really work!

Others may suffered from the same experience [61] [62]. Actually the *Serial* block does not work with Simulink Coder. It works just during simulation phase but after compiling to the execution file it doesn't perform it's task. One suggested solution is, use of a *byte pack block* before the *Arduino Serial Block*. But in order to overcome this problem, we preferred to use our slave microcontroller board (*SMCB*).

As illustrated in Figure 64, the *SMCB* has perched between the *Notebook* and the master microcontroller board (*MMCB*). It receives data by serial *USB* from *GUI* and transmits it again serially to the *MMCB*. The pertinent code uploaded on *SMCB*, has been illustrated in part 1 of appendix 8. Now the *Serial Receive* block in *MMCB* is

capable to catch the serial data. This hardware connection is between the serial port 0 of *SMCB* (TX0) and *MMCB* (RX0).

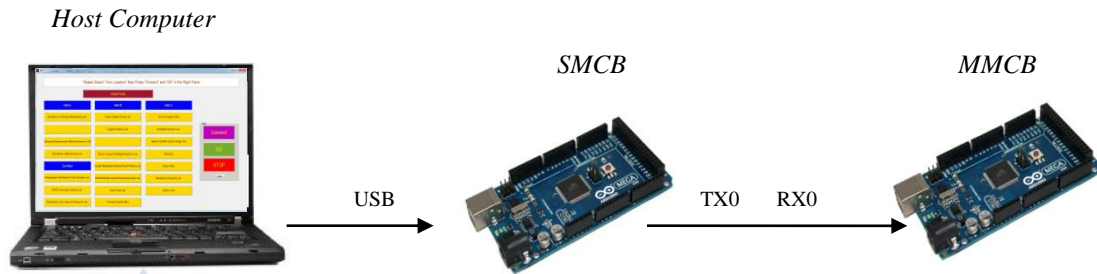


Figure 64: Data Flow in Hardware Setup from GUI to the Controllers

At the end of this chapter, it is worth mentioning that all of mentioned controllers, supervisor, planner, tracker, events and the other functions, are uploaded by *Simulink* coder on the *MMCB*. Appendix 13 illustrates all of these subsystems as a part of full navigation system.

CHAPTER 5

EXPERIMENTS AND RESULTS

Three cycles of experiment were performed to test and validate the designed and implemented navigation system. First cycle of experiment was performed on the simulated system. The second cycle was implemented on the actual system but not under load, and the third cycle of experiments were conducted in the field on the actual robot.

The analysis for the efficiency of the controllers, especially the *GTG* controller, is done preliminary based on the results of the first cycle of experiments. The efficiency of the *GTG*, *AO* and *AO-GTG* controllers are analyzed in the second cycle of experiments. Then the third cycle of experiments were conducted for dozens of times to complete and improve the efficiency of the *GTG*, *AO*, *AO-GTG* and *FW* controllers. Mechanical behavior of the components and performance of whole of the system were checked out in this cycle. The results of the experiments are presented and discussed in the following pages.

5.1 Results of the Simulated System

A *Simulink* model was formed using by the plant model, achieved in the section 4.5 of chapter 4. Appendix 14 illustrates this simulated model.

In this cycle we checked a *GTG* behavior, which drives the robot to a goal location and stops. Figure 65 shows the time signals of heading and position of the robot when it tries to reach the goal location at; $(x_g, y_g) = (100,0)$. Picking proper gains in the *PID* controller leads to have such a graph, with a reasonable settle time and little overshoot and undershoot. The traversed trajectory of the robot, when it moves to the

goal point and tries to come back to the initial point, is illustrated in the second graph.

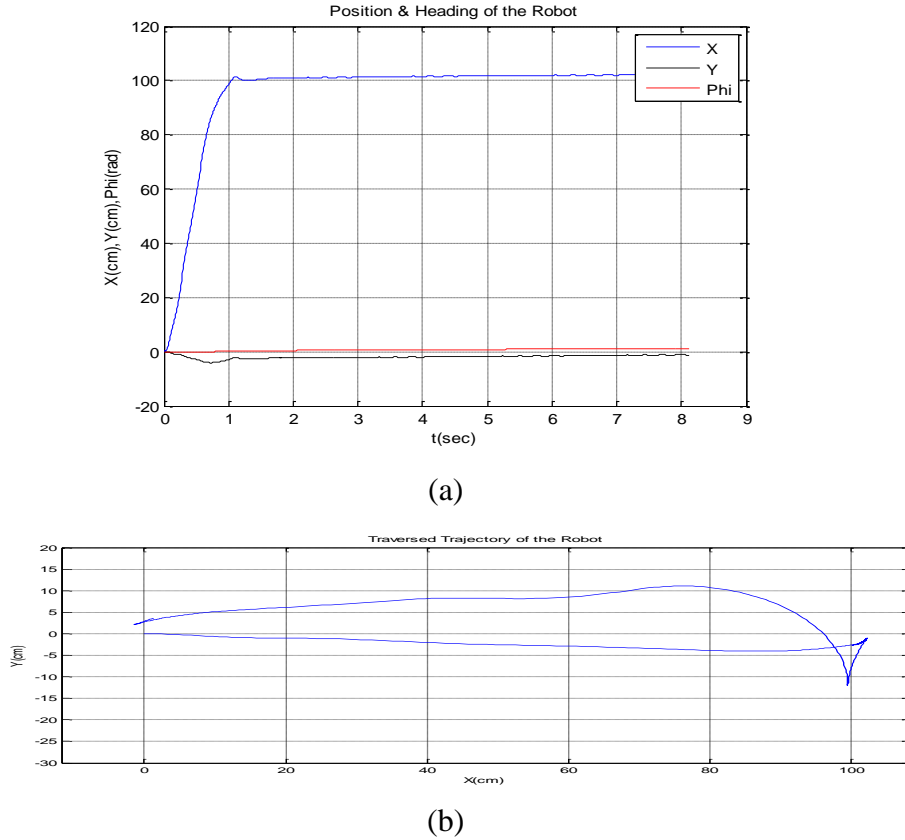
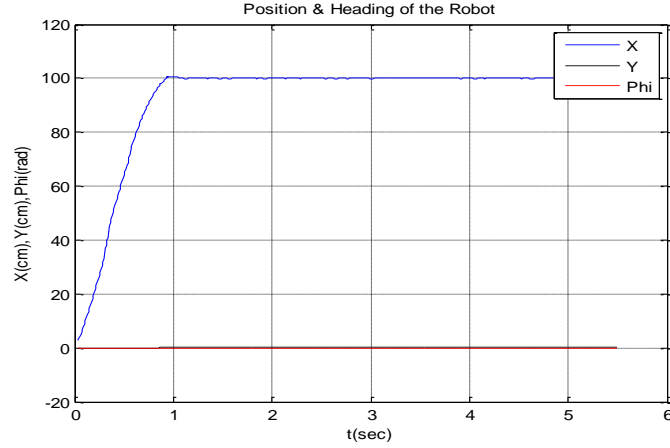


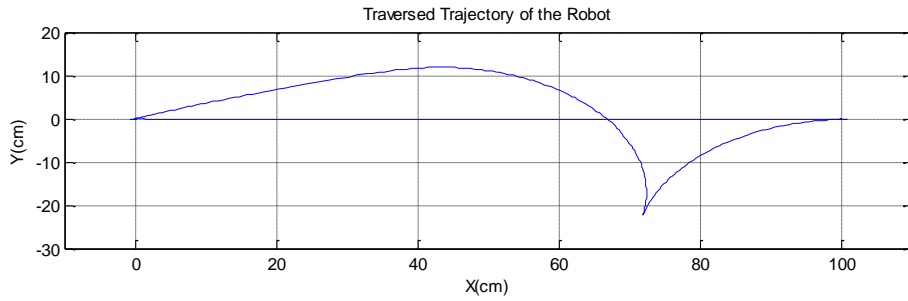
Figure 65: (a) Time Signals of Position & Heading (b) Traversed Trajectory of the Robot to reach the Goal Point at (100,0) and come back to the Initial Point in the Simulated System

5.2 Results of the Actual System with No-Load

In this cycle we repeated the above test on the actual robot. Performance of the actual navigation system, as illustrated in appendix 13, including of all it's subsystems, was tested. This test implemented in no-load condition, in other words the powered wheels were not on the ground. Samely a *GTG* behavior was checked and then *AO* and *AO-GTG* behaviors were tested. It is worth mentioning that the heading of the robot is computed through the odometry in this cycle. Figure 66 shows the time signals of heading and position of the robot in this case. Picking the same gains in the *PID* controller leads to have a better situation. We had roughly the same settle time but very little overshoot and undershoot. The traversed trajectory of the robot, as it is displayed in the second graph, was more accurate and pleasant.



(a)



(b)

Figure 66: (a) Time Signals of Position & Heading (b) Traversed Trajectory of the Robot to reach the Goal Point at (100,0) and come back to the Initial Point in the Actual System

In the second test, we adopted another goal point, but this time there is an intermediary way point. The robot tries to reach the goal location (*Mechatronics Prototype Manufacturing Lab.*) at; $(x_g, y_g) = (354, 965)$ through the way point (*Entrance of Hall A*) at; $(x_i, y_i) = (404, 655)$. Figure 67 shows the time signals of heading and position of the robot in two separated graphs, which is followed by the traversed trajectory of the robot in third graph. Whole of the navigation system fulfill its duty efficiently.

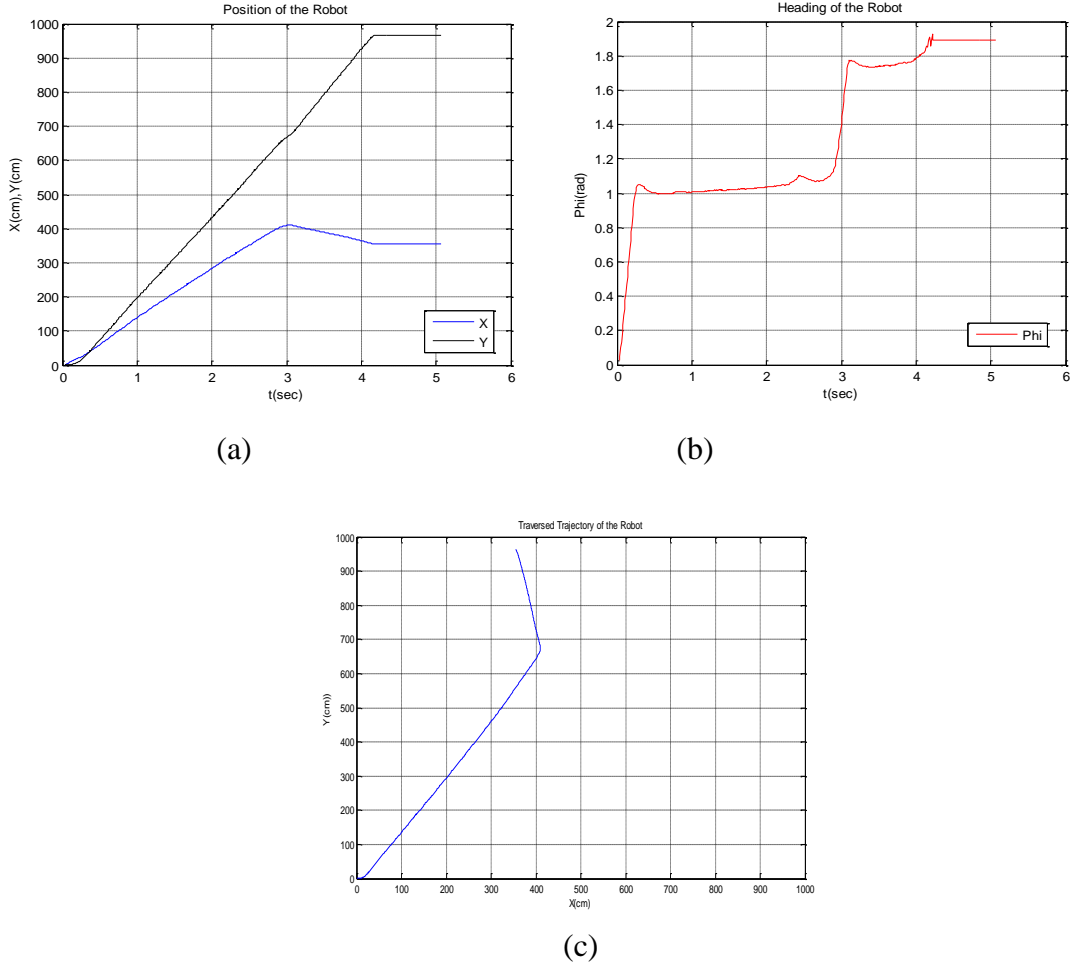


Figure 67: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Point at (469,1261) through the Intermediary Way Point at (469,552) in the Actual System

As the third test of this cycle, our goal point is at; $(x_g, y_g) = (2000, 0)$, but there is an obstacle on the way. The performance of the *AO* and *AO-GTG* controllers are checked in this test. Figure 68 shows the time signals of heading and position of the robot and the traversed trajectory of the robot. Again the navigation system fulfill its duty efficiently.

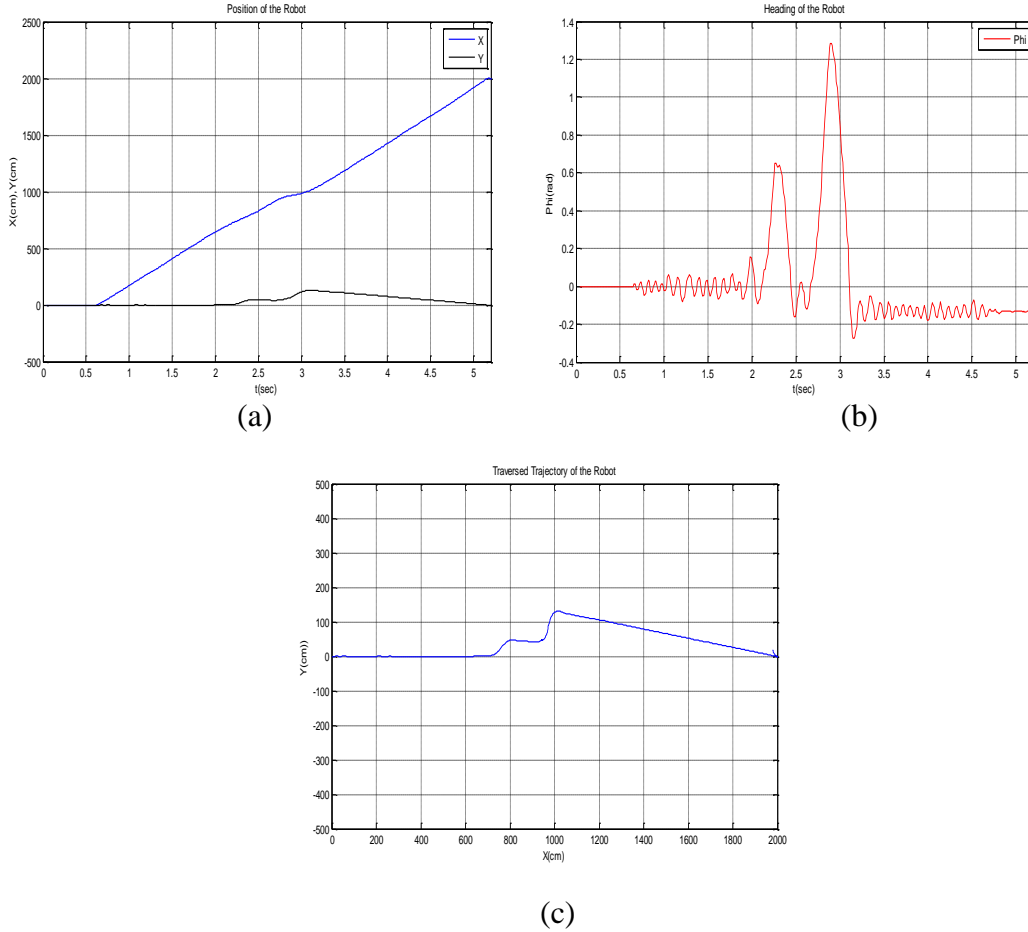


Figure 68: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Point at (2000,0) in the Actual System

5.3 Results of the Actual System in the Field

Eventually it was time to harvesting whatever we had planted! The actual robot should perform what it was designed and manufactured for. After a couple of tests in the field, we found out, the electronic hardwares including sensors, encoders, motors, microcontroller and driver boards act and react properly to the behavior controllers. But two unpleasant physical problem come to pass; first, the powered wheels turn situ at start time and second, the robot dose not move straight. The first problem stirs up all calculations which cancludes to reach a wrong final position and the second one drive the robot lopsided which cancludes similarly to have a big final error at goal points. With the aim of overcoming these unpleasant problems and drive the robot correctly in desired path, we designed and implemented two controllers; *Wheels Situ Spin Controller* and *Straight-Move controller*.

5.3.1 Wheels Situ Spin Controller

A situ spin occurs when the force delivered to the wheel tread exceeds that of available tread-to-surface friction and one or both wheels lose traction. This problem occurs in our robot, when it starts to move. In a moment, the speed power of the motors increases or decreases from zero to a specified amount and causes this undesirable problem. To solve simply this problem, the power speed of the motors should change smoothly. In other words, they should have ramp shape instead of step.

We employ the speed of the motors to compare up with the speeds, which have been arised by the controllers. According to this comparision, the *Wheels Situ Spin* controller performs its task. The pertinent *Matlab Function* has been illustrated in appendix 15.

On the other hand, type of the powered wheels were changed to the other type which is shown in Figure 69. This wheels are made from anti-slip material which helps to have less slip.

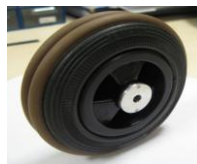


Figure 69: New Rear Powered Wheels

5.3.2 Straight - Move Controller

The crooked-drive may arise from various physical problems. In a vehicle using independent wheel control, applying the same power to each wheel generally does not result in the vehicle moving straight. This is caused by mechanical and surface differences experienced by each of the wheels. Basically, both sides of our robot are different. Our motors have a slightly different length of wire winding, our gearboxes have different amounts of friction, our wheels are slightly different diameters and have different amounts of friction with the ground, one wheel might encounter more dust than the other one... etc.

To reduce deviation in the robot heading, a better approach is to use a closed-loop, which adjusts the power applied to two motors based on the difference in their rates of rotation. A *PID* controller minimizes the difference between the measured and the desired value of a chosen system variable by adjusting the system control inputs. A key point is that, this controller ought not to apply on the robot when it is turning.

In our *Straight - Move Controller*, a *PID* controller uses the difference between two motors speed to calculate the angular speed difference that should be applied to the robot controller. We named this controller, *Motor's Speed Controller*, which will be a portion of our *Straight - Move Controller*. By the help of this controller, we could make sure the two wheels are going at the same speed, which overcomes the differences in the system up to the point they meet the wheels, but the wheels might still be slightly different diameters, and or;

- The gravity centre is not in the middle! ie maybe one of the robot's parts is placed more on a side than on the other. etc.
- The contact point of the dragging on the rear wheels is not on the middle! If the piece of the case that touches the ground is on one side, it can affect direction greatly.
- The robot is not symmetrical! ie maybe it's longer, wider or heavier on the one side.
- Both of pairs rear and front wheels or one of them are not parallel with each other or with the robot's side.

Because of all above mentioned problems, there may still be little deviation. So we tried to optimize the above mechanical problems insofar as it was possible.

First the type of front wheels was changed to the other type of omni-wheels, which is shown in Figure 70. The main reason of this change was that the old types were not rotate as freely as we expected.



Figure 70: New Front Omni-Wheels

The motor's mount shown in Figure 8 at chapter 3, which were assembled on each motor separately, seem not to be parallel with each other and with the side of the robot. They were tuned to be parallel but after a couple of robot's movement, their parallelism was suspicious! We decided to construct and assemble the motors on another mount seamlessly. This mount shown in Figure 71, keeps the rear wheels in parallel with each other and with the side of the robot at any time.



Figure 71: New Motor's Mount

It was also tried to perch the gravity centre in the middle. But we can do nothing to optimize the other difficulties. What could we do to the wheel's diameter and or the contact point of the dragging!

To optimize the situation and detect any other crooked drive, as a first solution, we employed four ultrasonic sensors and try to eliminate the effects of the problems and keep the robot move straight. Two sensors on both sides measure the distance from the robot to the parallel wall and then the controller calculates the heading of the robot. We named this controller, *Ultrasonic Sensor Controller* which is a portion of our *Straight - Move Controller*. Same to the *Motor's Speed controller*, this controller ought to apply on the robot when it is not turning. The output of the *Straight-Move controller*, *WD*, is used to improve the robot angular speed.

But what if there is no wall where the robot would move in parallel with it! The safe and sustainable solution is having the heading of the robot completely independent. To perform this task, we employ the *Compass*. In other words, the data achieved by the *Compass* is used as the heading of the robot instead of the data achieved by the odometry. Applying the compass has another benefit for us. Although we applied a controller to overcome wheels situ spin, but if there is even this difficulty, the heading of the robot will not be calculated wrong.

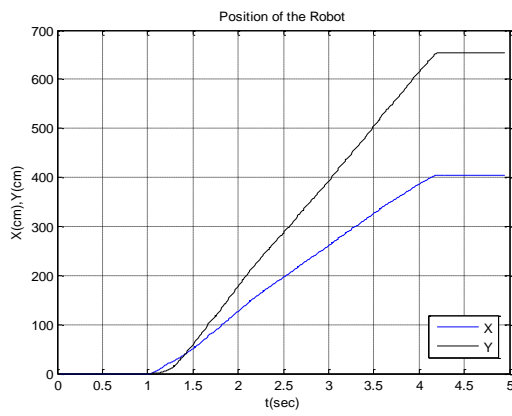
After designing and adding the *Wheels Situ Spin* and *Straight - Move Controller* to the controller architecture of the navigation system, and also using of the achieved data from *Compass* as the heading, we continued to test the robot in the field as the third cycle of experiments.

The tests were performed in various conditions for various goal points with and without intermediary way points. Table 6 illustrates the topics and conditions of these implemented tests. Figures 72 (1-8) show the time signals of heading and position of the robot and the traversed trajectory of the robot for each test. Again the navigation system fulfill its duty efficiently.

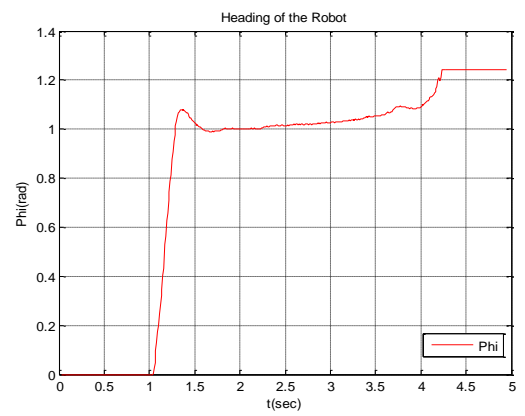
During the tests it is turned out that the final location of the robot changes slightly in each of them. In other words, the robot stops at goal point with a small difference compared to the last one. In order to final tuning of the robot's location, there are some solutions. The easiest way is simply to guide the robot to this location. This final guidance can be done in different ways: placing beacons, markers, bar codes, painting lines, burying an inductive loop or magnets, or by etc. in the environment. Performing one of these solutions will be simply implemented by the vision system in the People-Interaction phase. Another way which is more sustainable and accurate, is the using of the *Indoor Positioning Systems* (IPS) [63]. In these systems the exact position of a mobile device is determined using by radio waves, magnetic fields, acoustic signals, or other sensory information.

Table 6: Topics of Implemented Tests with Various Conditions in the Field

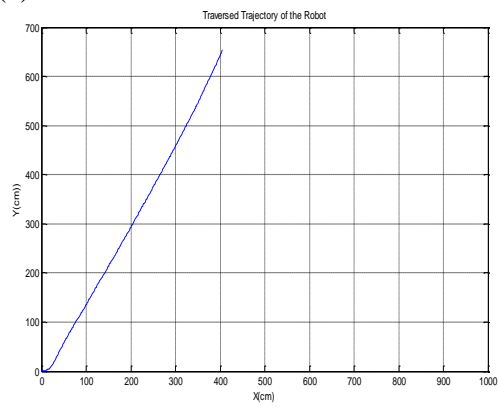
Test No.	Goal Point	Intermediary Way Point	Fixed Obstacle	Movable Obstacle	Wall
1	Entrance of Hall A	-	-	-	-
2	Entrance of Hall A	-	X	-	-
3	Entrance of Hall A	-	-	X	-
4	Mechatronics Prototype Manufacturing Lab.	Entrance of Hall A	-	-	X
5	Entrance of Hall B	-	X	X	X
6	Robots Student Society Lab.	Entrance of Hall B	X	-	X
7	Right Corridor	-	-	-	-
8	Undergraduate Mech. Eng. Education Lab.	Right Corridor	X	-	X



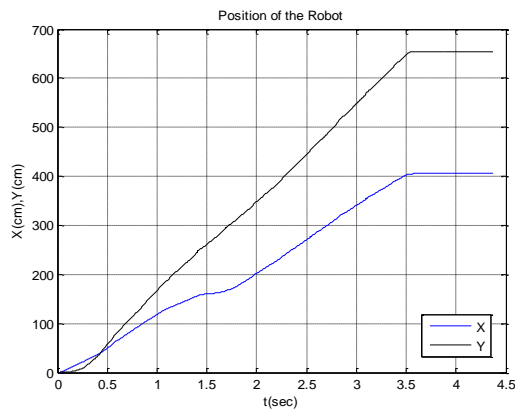
1 (a)



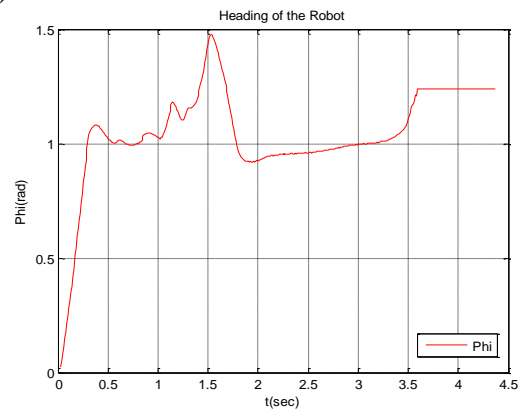
1 (b)



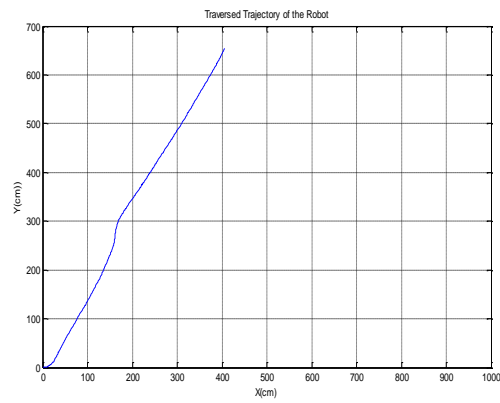
1 (c)



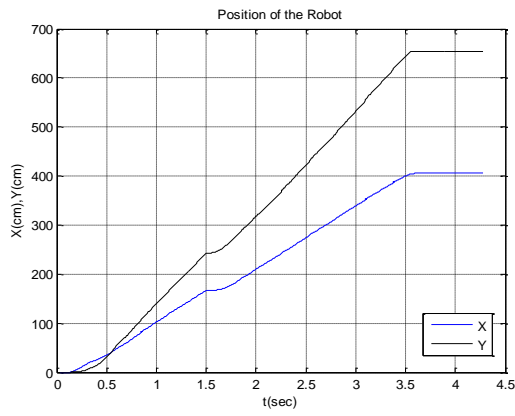
2 (a)



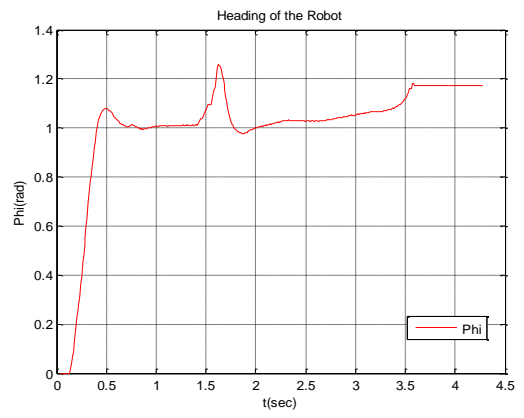
2 (b)



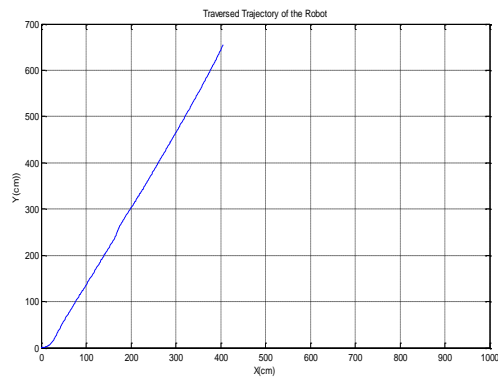
2 (c)



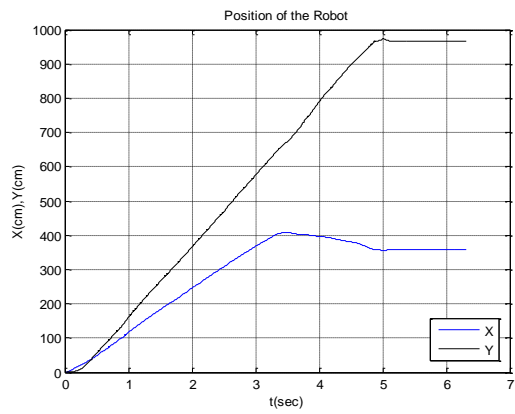
3 (a)



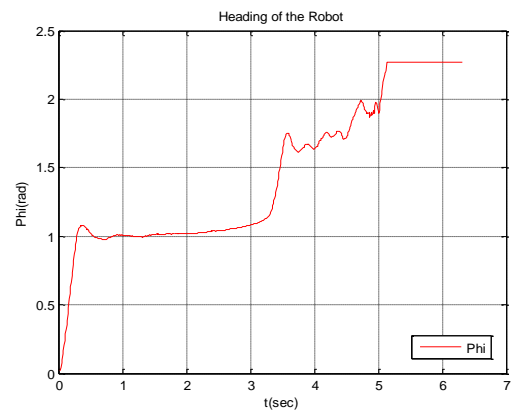
3(b)



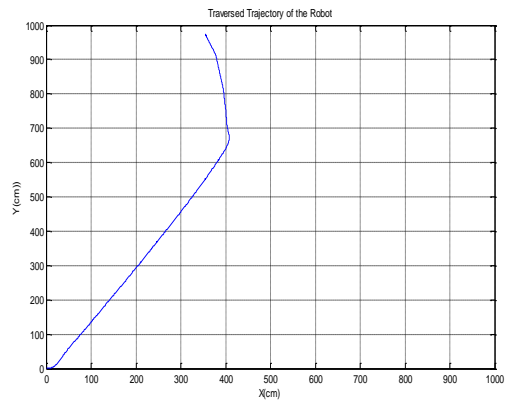
3 (c)



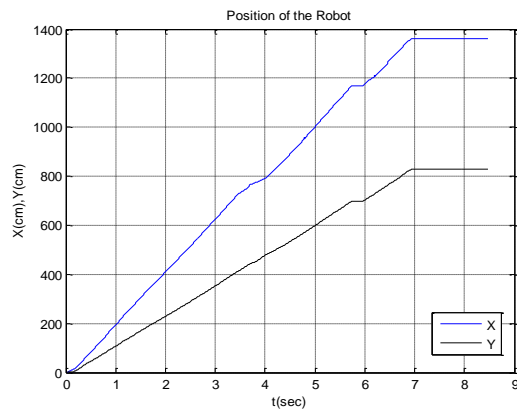
4 (a)



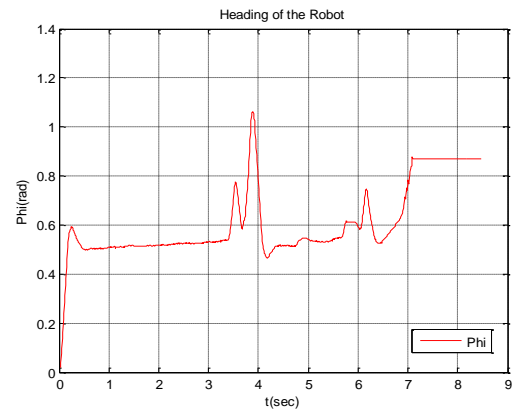
4 (b)



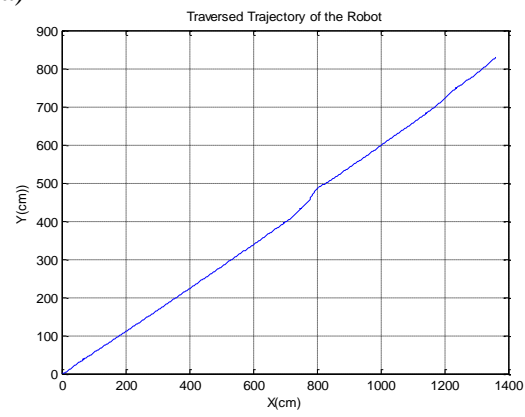
4 (c)



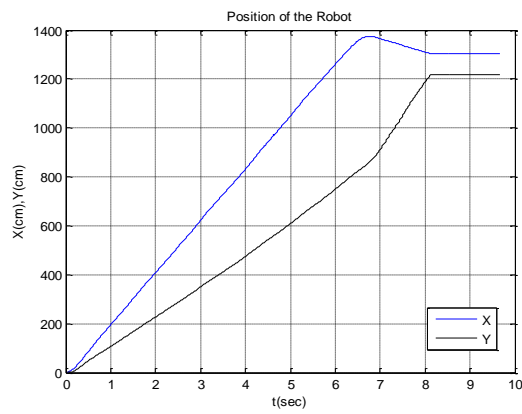
5 (a)



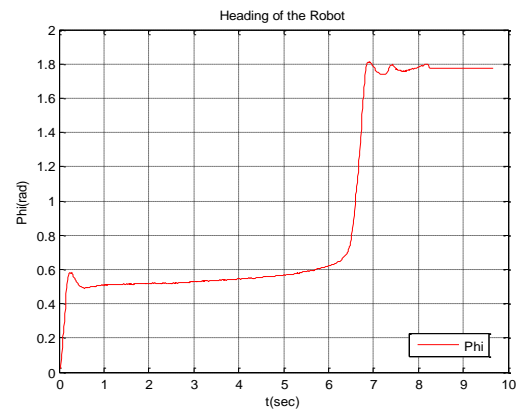
5 (b)



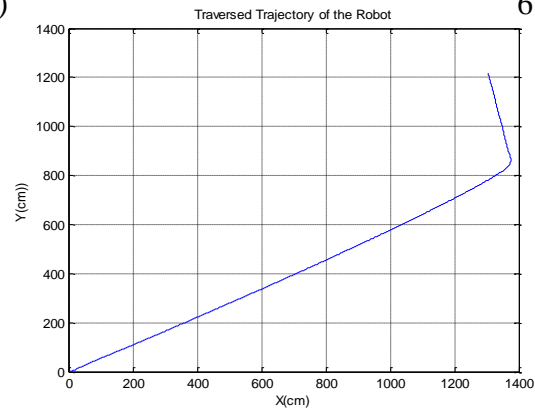
5 (c)



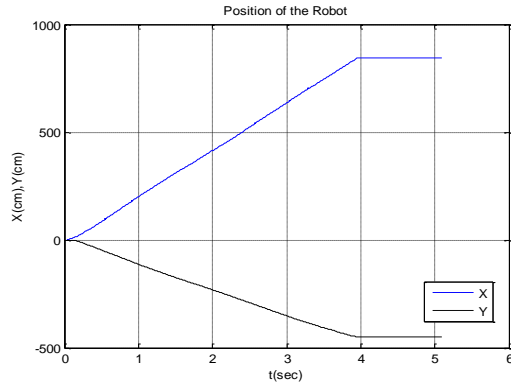
6 (a)



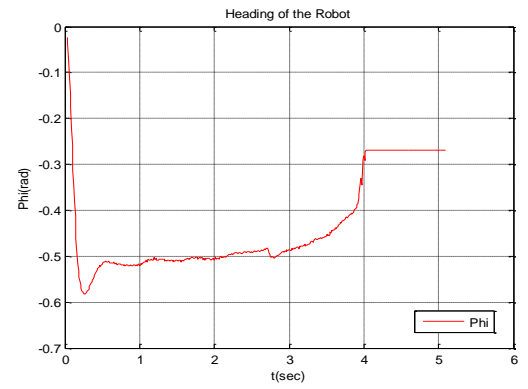
6 (b)



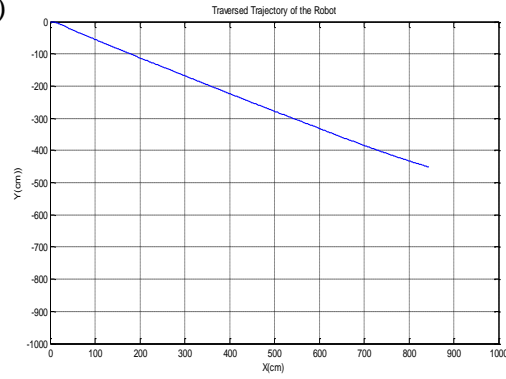
6 (c)



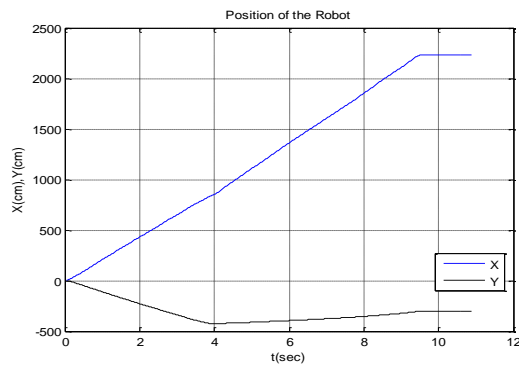
7 (a)



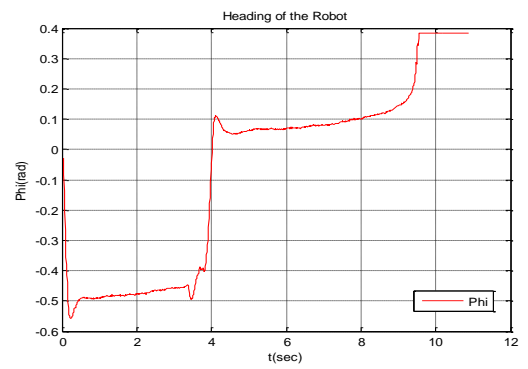
7 (b)



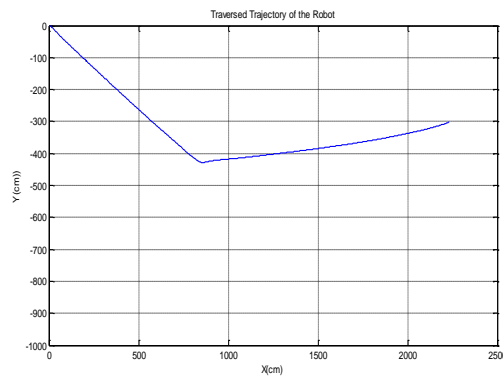
7 (c)



8 (a)



8 (b)



8 (c)

Figure 72: (a) Time Signals of Position (b) Heading (c) Traversed Trajectory of the Robot to reach the Goal Points based on Table 6 in the Actual System

CHAPTER 6

DISCUSSION AND CONCLUSION

This study is focused on the navigation of a mobile robot based on the behaviors architecture. After designing, constructing and implementing of an integrated system, the performance of the robot to safe navigate through the indoor environments are studied. The research also aimed to identify the pros and cons of the behavior- based maples against map-based navigation.

In the presented behavior-based control architecture which composes of localization, obstacle avoidance, path planning and robot control for steering, the onboard sensors are the main role to localize the robot. Such a vision-based navigation has some advantages respect to the map-based navigation.

The main pro is no need for any map to navigate. The robot is able to steer to any arbitrary desire pose in a maples unknown environment. In other words, it is able to establish easy navigation in places without pricey infrastructure or pre-made maps. In addition, it has some other advantages. One of which is, it's higher efficiency and quickness faced with moving obstacles. Despite these benefits, maples navigation also has certain drawbacks. For instance, the control architecture is more complicated. Moreover, while the map-based navigation requires less hardware, it is relatively expensive.

To test and validate the navigation system, some cycles of experiment were performed. The analysis for the efficiency of the behavior-based controllers is done based on the results of these experiments. Meanwhile the mechanical behavior of the components and performance of whole of the system were checked out during these tests.

Based from the results of the experiments, it is figured out that the major difficulty to have a flawless navigation, originates from physical problems. The stuff of the powered wheels, the gravity centre of the robot, the contact point of the dragging on the wheels, the robot symmetry, and the parallel wheels, are some of the critical points which we ought to pay more attention in designing and manufacturing steps. Then, besides our presented perfect controller architecture, we can have a flawless navigation system.

In the future developments, the capabilities of the robot will upgrade, so that it will be able to guide visitors who trying to get to a certain place and explain them so that they can acquire knowledge of there and enjoy visiting. To realize such a task, in addition to a safe navigation in dynamic, cluttered and crowded public environments, the system ought to have an effective and interesting human-robot interactions.

REFERENCES


- [1] Capek, Karel, et al. "*Rossum's universal robots*." Prague, CZ (1920).
- [2] http://en.wikipedia.org/wiki/Robot_series_Asimov.27s_robots_on_screen, Visited Date; 20.01.2014
- [3] <http://websters.yourdictionary.com>, Visited Date; 20.01.2014
- [4] <http://en.wikipedia.org/>, Visited Date; 20.01.2014
- [5] Fezari, Mohamed, S. Lemboub, and M. S. Boumaza. "*VR-Stamp with DSP-TM320C6711 for hand-free voice-driven monitoring robots navigation*." International Conference on Information Technology and e-Services (ICITeS), IEEE, 2012.
- [6] Archila, John Faber, and Marcelo Becker. "*Mathematical models and design of an AGV (Automated Guided Vehicle)*." 8th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2013.
- [7] Oh, SeungSub, et al. "*A system architecture for intelligent building guide robot PHOPE*." 4th International Conference on Autonomous Robots and Agents, ICARA, IEEE, 2009.
- [8] Tomatis, N., et al. "*Building a fully autonomous tour guide robot: Where academic research meets industry*." Proc. Int. Symp. on Robotics. 2002.
- [9] Sasai, Takuya, et al. "*Development of a guide robot interacting with the user using information projection—Basic system*." IEEE International Conference on Mechatronics and Automation (ICMA), 2011.
- [10] Prodanov, Plamen J., et al. "*Voice enabled interface for interactive tour-guide robots*." IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 2. IEEE, 2002.
- [11] Dehuai, Zeng, Xie Cunxi, and Li Xuemei. "*Design and Implementation of a security and patrol robot system*." IEEE Mechatronics and Automation, International Conference, Vol. 4, 2005.
- [12] Thrun, Sebastian, et al. "*MINERVA: A second-generation museum tour-guide robot*." IEEE International Conference on Robotics and Automation Proceedings, Vol. 3, 1999.
- [13] Simmons, Reid, et al. "*Lessons learned from Xavier*." Robotics & Automation Magazine, IEEE 7.2 (2000): 33-39.
- [14] Kanda, Takayuki, et al. "*An affective guide robot in a shopping mall*." Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction, 2009.
- [15] J Datta, Chandan, and Ritukar Vijay. "*Neel: an intelligent shopping guide-using web data for rich interactions*." Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction, Press, 2010.
- [16] Cokal, Erhan, and Abdulkadir Erden. "*Development of an image processing system for a special purpose mobile robot navigation*." IEEE Fourth Annual Conference on Mechatronics and Machine Vision in Practice, 1997.








- [17] Shieh, Ming-Yuan, et al. "*Design and Implementation of a Vision-Based Shopping Assistant Robot.*" IEEE International Conference on Systems, Man and Cybernetics, SMC'06. Vol. 6, 2006.
- [18] Chiang, Kuo-Hung, et al. "*Multisensor-based outdoor tour guide robot NTUI.*" SICE Annual Conference, IEEE, 2008.
- [19] I Shiomi, Masahiro, et al. "*A larger audience, please!: encouraging people to listen to a guide robot.*" Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction. IEEE Press, 2010.
- [20] Pandey, Amit Kumar, and Rachid Alami. "*A step towards a sociable robot guide which monitors and adapts to the person's activities.*" IEEE International Conference on Advanced Robotics, ICAR, 2009.
- [21] Luo, R. C., et al. "*NCCU security warrior: an intelligent security robot system.*" IEEE 33rd Annual Conference of the Industrial Electronics Society, IECON, 2007.
- [22] Kuo, C. H., et al. "*Remote control based hybrid-structure robot design for home security applications.*" International Conference on Intelligent Robots and Systems, IEEE/RSJ, 2006.
- [23] Elnagar, Ashraf, and Leena Lulu. "*A visual tool for computer supported learning: The robot motion planning example.*" Computers & Education 49.2 (2007): 269-283.
- [24] Tsai, Ching-Chih, et al. "*Autonomous navigation of an indoor tour guide robot.*" Workshop on Advanced robotics and Its Social Impacts, ARSO IEEE, 2008.
- [25] Mizobuchi, Yoshinobu, et al. "*Trajectory planning method of guide robots for a achieving the guidance.*" IEEE International Conference on Robotics and Biomimetics (ROBIO). 2005
- [26] Philippsen, Roland, and Roland Siegwart. "*Smooth and efficient obstacle avoidance for a tour guide robot.*" ICRA. 2003.
- [27] Henry, Peter, et al. "*Learning to navigate through crowded environments.*" IEEE International Conference on Robotics and Automation (ICRA), 2010.
- [28] Shen, Jiali, and Huosheng Hu. "*Visual navigation of a museum guide robot.*" The Sixth World Congress on Intelligent Control and Automation, WCICA, Vol. 2. IEEE, 2006.
- [29] Wang, Chun-Chieh, Kuo-Lan Su, and Chih-Teng Shen. "*Implementation of Tour Guide Robot via Shape Recognition and Path Planning.*" Fourth International Conference on Innovative Computing, Information and Control (ICICIC), IEEE, 2009.
- [30] Gorry, Benjamin, et al. "*Using Mean-Shift Tracking Algorithms for Real-Time Tracking of Moving Images on an Autonomous Vehicle Testbed Platform.*" International Journal of Computer Science & Engineering 1.3 (2007).
- [31] Kim, Jeongdae, and Yongtae Do. "*Moving obstacle avoidance of a mobile robot using a single camera.*" Procedia Engineering 41 (2012): 911-916.
- [32] Moriwaki, Katsumi. "*Recognition of moving objects by image processing and its applications to a guide robot.*" IEEE/SICE International Symposium on System Integration (SII), IEEE, 2011.



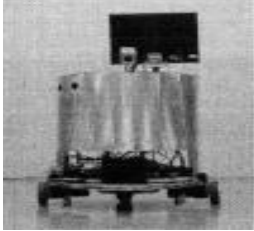




- [33] Oyama, Takaya, et al. "*Tracking visitors with sensor poles for robot's museum guide tour.*" The 6th International Conference on Human System Interaction (HSI), IEEE, 2013.
- [34] Sugiyama, Seiji, Kouhei Baba, and Tsuneo Yoshikawa. "*Guide robot with personal identification method using dress color information via KINECT.*" IEEE International Conference on Robotics and Biomimetics (ROBIO), 2012.
- [35] Shen, Jiali, and Huosheng Hu. "*Visual navigation of a museum guide robot.*" The Sixth World Congress on Intelligent Control and Automation, WCICA, Vol. 2. IEEE, 2006
- [36] Luo, R. C., T. Y. Hsu, and K. L. Su. "*The development of a multisensor based intelligent security robot: Chung Cheng# 1.*" IEEE International Conference on Mechatronics, ICM'05, 2005.
- [37] Arras, Kai O., et al. "*Multisensor on-the-fly localization: Precision and reliability for applications.*" Robotics and Autonomous Systems 34.2 (2001): 131-143.
- [38] <http://marketsandmarkets.com/ServiceRobotics>, Visited Date; 10.02.2015
- [39] <http://wtec.org/robotics/report/05-Industrial.pdf>, Download Date; 10.02.2015
- [40] <http://news.panasonic.com/press/news/official.data>, Visited Date; 10.02.2015
- [41] <http://metralabs.com>, Visited Date; 10.02.2015
- [42] G. Çelik, E.Erdem, K.Kok, D. Oksay, C. Sevimli, Z. Erden. " Mekatronik Mühendisliği Bölümü Laboratuvarları İçin Bekçi/Rehber Robot Tasarımı." 3. Mekatronik Mühendisliği Öğrenci Kongresi (MeMÖK2012) Bildiri Kitabı, Sayfa 47-56, 8 Haziran 2012, Atılım Üniversitesi, Ankara..
- [43] <http://www.pololu.com/product/1107>, Visited Date; 01.05.2015
- [44] http://www.sharpsma.com/webfm_send/1487, Downloaded Date; 01.05.2015
- [45] <http://www.micropik.com/PDF/HCSR04.pdf> , Downloaded Date; 01.05.2015
- [46] <https://www.pololu.com/product/2468>, Visited Date; 01.05.2015
- [47] <http://arduino.cc/en/Main/arduinoBoardMega2560B>, Visited Date; 01.05.2015
- [48] <http://www.pololu.com/product/2502>, Visited Date; 01.05.2015
- [49] <http://www.pololu.com/product/2104>, Visited Date; 20.01.2015
- [50] Arkin, Ronald C. "Behavior-based robotics." MIT press, 1998.
- [51] <http://gritslab.gatech.edu>, Visited Date; 01.05.2015
- [52] Simmons, Reid, et al. "*A layered architecture for coordination of mobile robots.*" Multi-robot systems: from swarms to intelligent automata. Springer Netherlands, 2002. 103-112.
- [53] Yan, Melissa. "Dijkstra's Algorithm." <http://www-math.mit.edu>
- [54] Bertsekas, Dimitri P., and Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Vol. 1. No. 2. Belmont, MA: Athena Scientific, 1995.
- [55] https://www.advancedtransit.net/files/Muller1_Presentation, Visited Date; 05.03.2014
- [56] <http://forum.arduino.cc/index.php/topic>, missing counting encoder, Visited Date; 15.11.2014
- [57] Wilhelm, Eric Jamesson. "*Design of a wireless control system for a laboratory planetary rover*". Diss. Massachusetts Institute of Technology, 1999
- [58] Kara, Sertac Emre. " *Control of two wheel self stabilizing mobile robot with a simple arm* ". Mechatronics Engineering, Atılım University, October 2014.


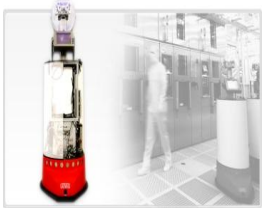


- [59] <http://www.mathworks.com/company/newsletters/articles/data-sets-for-motor-control-with-arduino.html>, Visited Date; 20.12.2014
- [60] S.Temel, S.Yagli, S.Goren. "*Discrete time control systems recitation*". Electrical and Electronics Engineering, Middle East Technical University.
- [61] <http://www.mathworks.com/matlabcentral/answers/116783>, Visited Date; 05.02.2015
- [62] <http://stackoverflow.com/questions/21883900>, Visited Date; 05.02.2015
- [63] http://en.wikipedia.org/wiki/Indoor_positioning_system, Visited Date; 26.06.2015

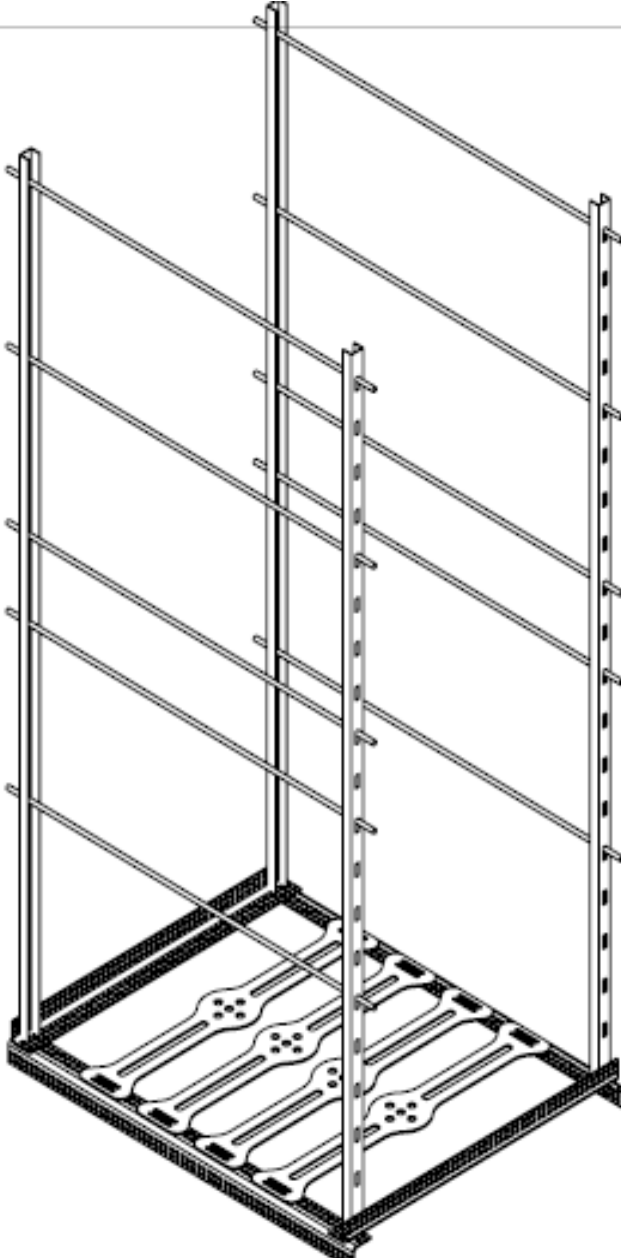
Appendix 1: Table of Service Robots

	Name of the Robot	Description	Place	Picture
1	MINERVA	Museum Tour Guide Robot	Smithsonian's National Museum of American History, USA	
2	XAVIER	Building Guide Robot	Carnegie Mellon University, USA	
3	PHOPE	Building Guide Robot	South Korea	
4	ROBOX	Interact with Human Robot	Switzerland	
5	RHINO	Autonomous, Interactive Tour Guide Robot	Germany	
6	PYGMALION	Fully Autonomous Self-Contained Robot	Switzerland	

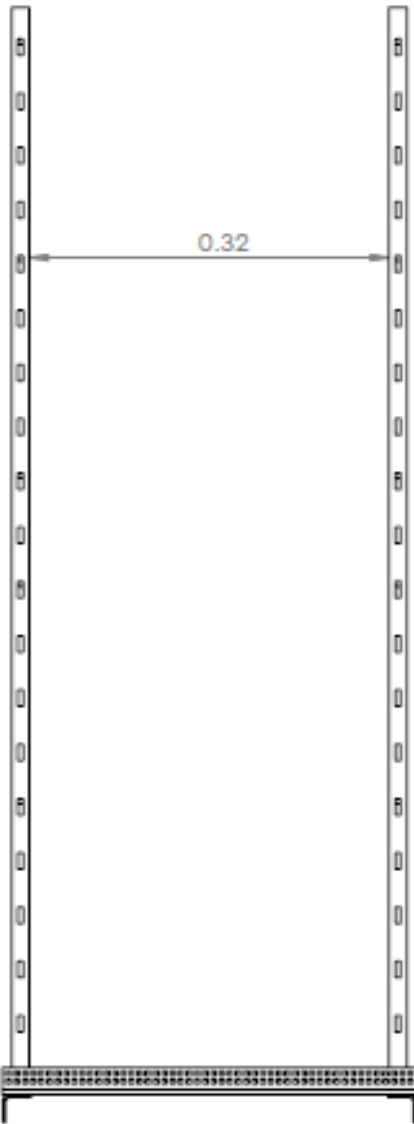
7	SAGE	Guide Robot	Carnegie Museum of Natural History, USA	
8	MASTIFF-I	Security & Patrol Robot	China	
9	NTU-I	Tour Guide Robot	National Taiwan University, Taiwan	
10	SECURITY WARRIOR	Intelligent Security Robot	Taiwan	
11	ROBOVI	Autonomous, Interactive Tour Guide Robot	Japan	
12	ATLAS	Museum Guide Robot	United Kingdom	
13	PIONEER3-AT	Small Prepared Robot	General	

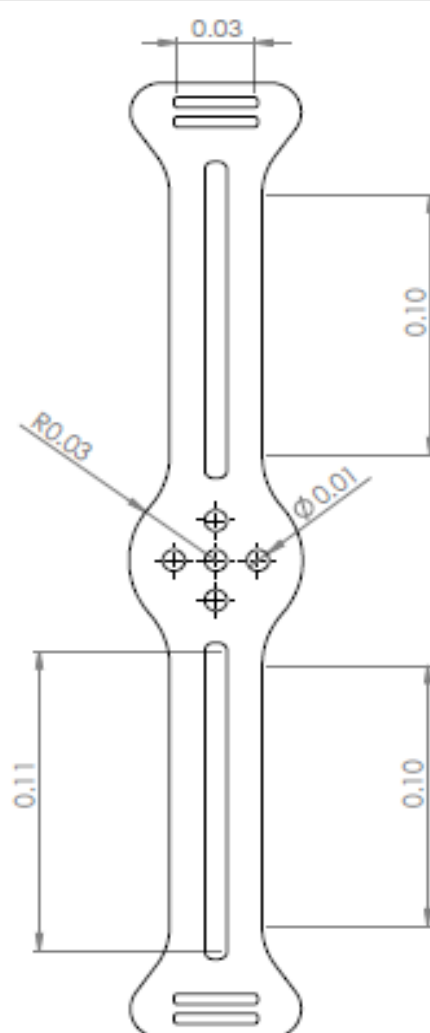
14	MODROB-C & D	Technology Demonstrative Mobile Robots	METU, Turkey	
15	TT-JOY	Mobile Guide Robot	Japan	
16	VBSAR	Shopping Assistant Mobile Robot	Taiwan	
17	NEEL	Shopping Guide Robot	University of Auckland, New Zealand	
18	HOSPI	Autonomous Medication Delivery Robot	MetraLabs, Germany	
19	SCITOS A5	Service Guide Robot	MetraLabs, Germany	
20	SCITOS G6 Transporter	Autonomous Transporter Robot	MetraLabs, Germany	

21	SCITOS G5	Professional Platform Robot	MetraLabs, Germany	
22	SCITOS G5 Monitoring	Autonomous Service Robot	MetraLabs, Germany	
23	SCITOS G5 Manipulator	Autonomous Service Robot	MetraLabs, Germany	
24	SCITOS G3	Interactive Home-Care Robot	MetraLabs, Germany	



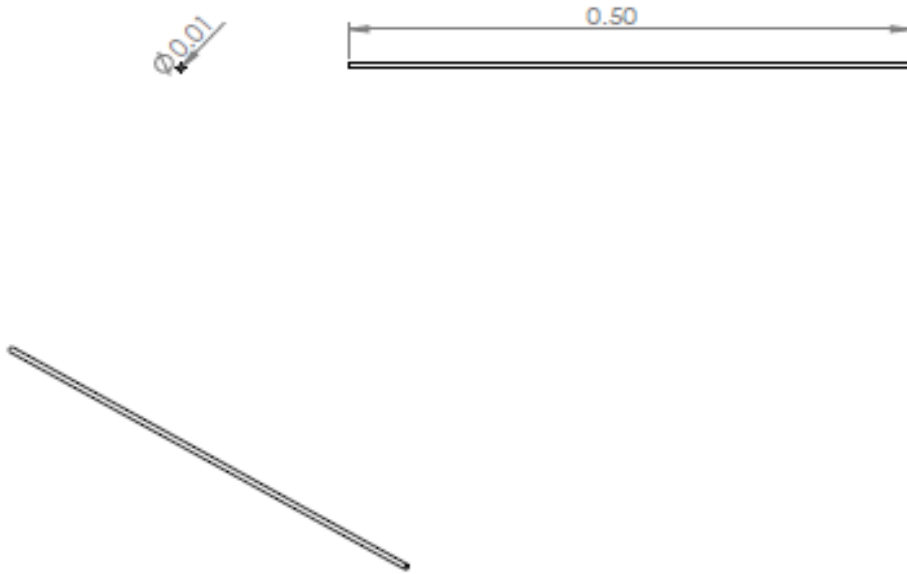
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:	ORDER AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
SCALE 1 : 5					
				TITLE:	
DRAWN	NAME	SIGNATURE	DATE		
CHECKED					
APPROVED					
MFG					
Q.A.					
				DWG NO.	
				Appendix2-Chasis^{A4}	
				SCALE: 1:10	
				SHEET 1 OF 1	

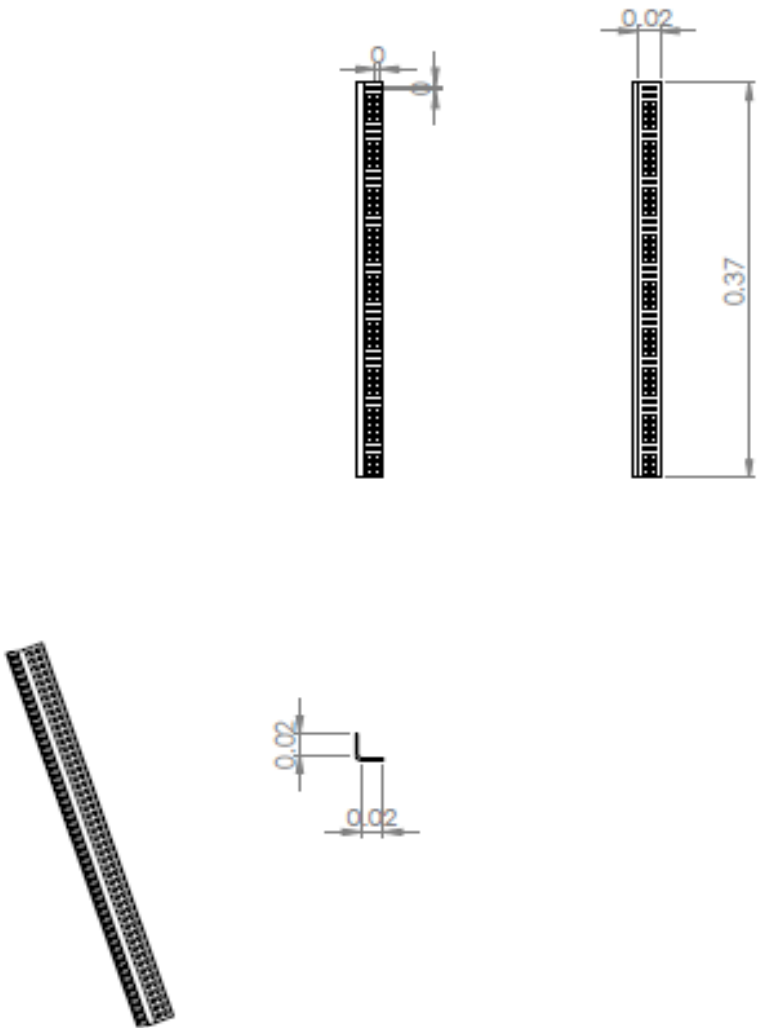
																																													
SCALE 1 : 5																																													
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		ORDER AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION																																			
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">NAME</th> <th style="width: 20%;">SIGNATURE</th> <th style="width: 10%;">DATE</th> <th style="width: 10%;"></th> <th style="width: 10%;"></th> <th style="width: 10%;"></th> </tr> </thead> <tbody> <tr> <td>DESIGN</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPROV</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>MFG</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>QA</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>				NAME	SIGNATURE	DATE				DESIGN						CHECK						APPROV						MFG						QA						MATERIAL:		TITLE:			
NAME	SIGNATURE	DATE																																											
DESIGN																																													
CHECK																																													
APPROV																																													
MFG																																													
QA																																													
WEIGHT:				SCALE: 1:10		SHEET 1 OF 1		Appendix2-Chasis Front, Back																																					



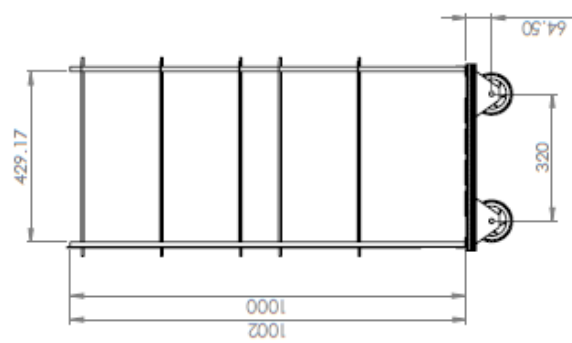
SCALE 1 : 2

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		PICK:		ORDER AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
NAME		SIGNATURE		DATE		TITLE	
DRAWN							
CHECKED							
APPROVED							
MFG							
Q.A.				MATERIAL:		DWG NO.	
						Appendix2-Red Arm ⁴	
				WEIGHT:		SCALE 1:2	
						SHEET 1 OF 1	

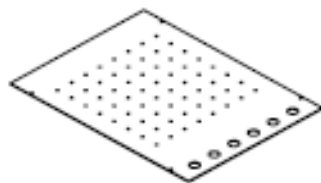
					
<small>UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:</small>		<small>FINISH:</small>	<small>DRILL AND BEND SHARP EDGES</small>	<small>DO NOT SCALE DRAWING</small>	<small>REVISION</small>
<small>DRAWN</small>	<small>NAME</small>	<small>SIGNATURE</small>	<small>DATE</small>	<small>TITLE</small>	
<small>CHECKED</small>				<h1 style="margin: 0;">Appendix2-Long Screw</h1>	
<small>APPROVED</small>					
<small>INQ</small>					
<small>Q.A.</small>					
<small>MATERIAL:</small>		<small>DWG. NO.</small>		<small>SCALE</small>	
<small>WEIGHT:</small>		<small>SCALE 1:1</small>		<small>SHEET 1 OF 1</small>	



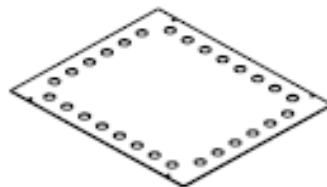
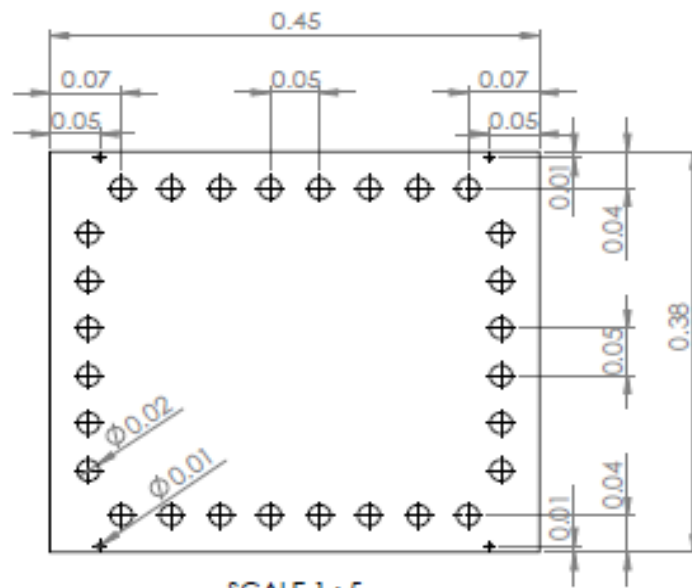
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: UNITS: ANGLES:				FINISH:		ORDER AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
DRAWN		NAME		SIGNATURE		DATE		TITLE:			
CHECKED											
APPROVED											
MFG											
Q.A.						MATERIAL:		DWG NO.		Appendix2-L bar A4	
						WEIGHT:		SCALE: 1:2		SHEET 1 OF 1	



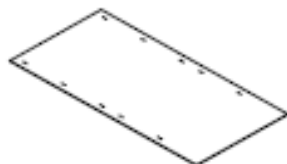
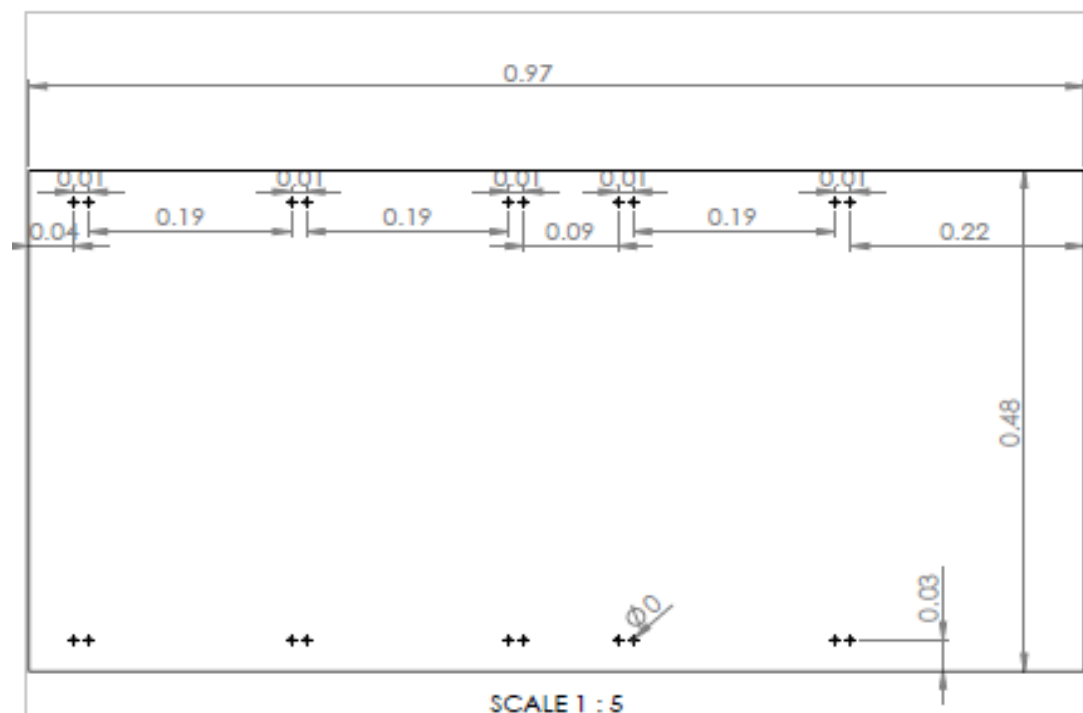
		<p>0.45</p> <p>0.05</p> <p>0.05</p> <p>0.02</p> <p>0.05</p> <p>0.05</p> <p>0.34</p> <p>SCALE 1 : 5</p>																			
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH: DRILL AND BREAK SHARP EDGES																			
		DO NOT SCALE DRAWING																			
		REVISION																			
<table border="1"> <tr> <th>NAME</th> <th>SIGNATURE</th> <th>DATE</th> </tr> <tr> <td>DESIGN</td> <td></td> <td></td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> </tr> <tr> <td>APPROVE</td> <td></td> <td></td> </tr> <tr> <td>MFG</td> <td></td> <td></td> </tr> <tr> <td>Q.A.</td> <td></td> <td></td> </tr> </table>		NAME	SIGNATURE	DATE	DESIGN			CHECK			APPROVE			MFG			Q.A.			TITLE:	
NAME	SIGNATURE	DATE																			
DESIGN																					
CHECK																					
APPROVE																					
MFG																					
Q.A.																					
MATERIAL:		DWG NO.																			
WEIGHT:		SCALE: 1:10																			
		SHEET 1 OF 1																			



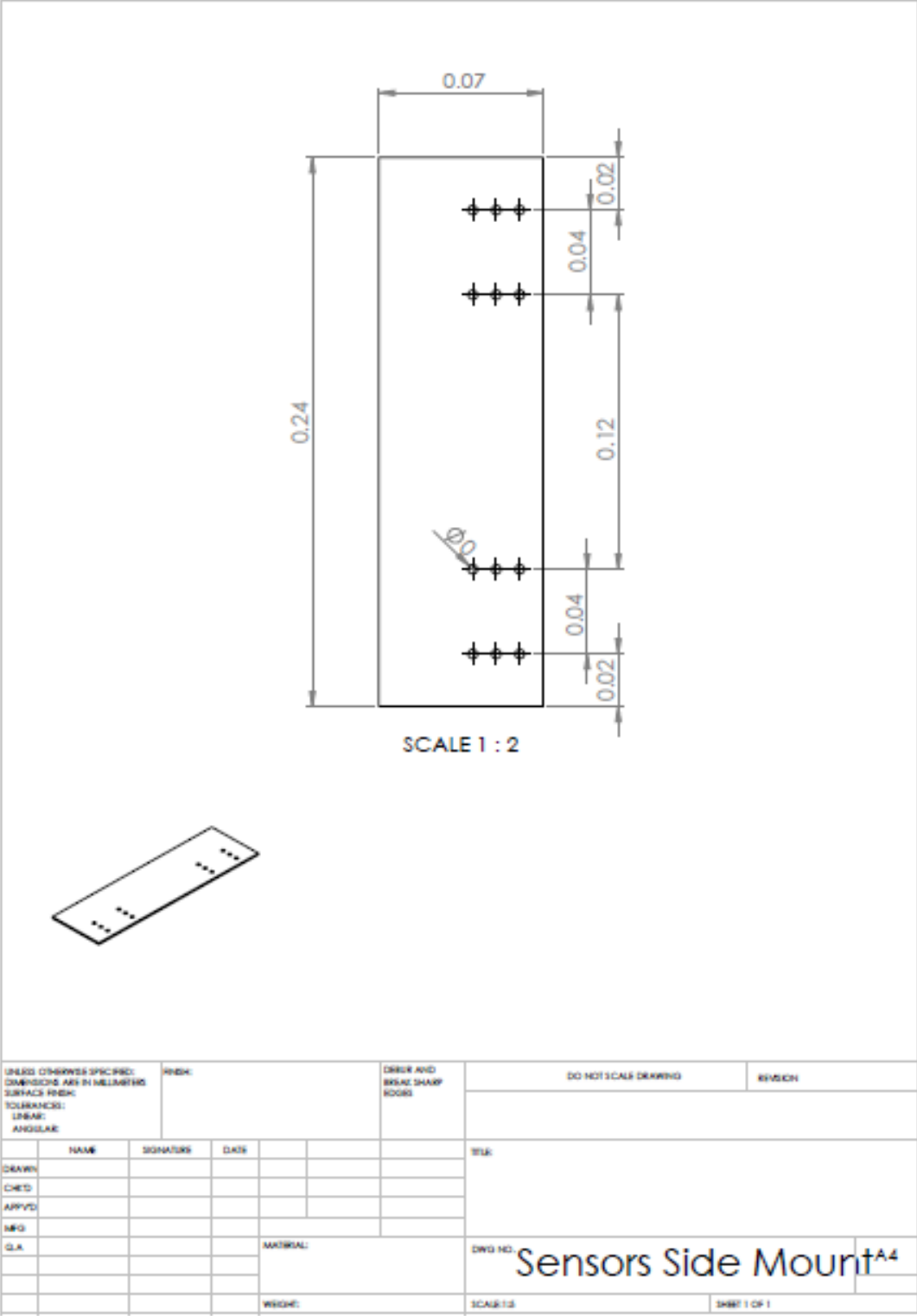
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		ORDER AND BRUSH SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
NAME		SIGNATURE		DATE				TITLE:			
DRAWN											
CHECKED											
APPROVED											
MRG											
G.A.				MATRICAL:		DWG NO. Appendix3-Floor 2-5⁴					
				WROTE:		SCALE: 1:10		SHEET 1 OF 1			

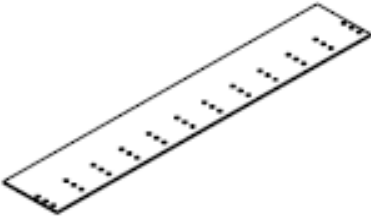
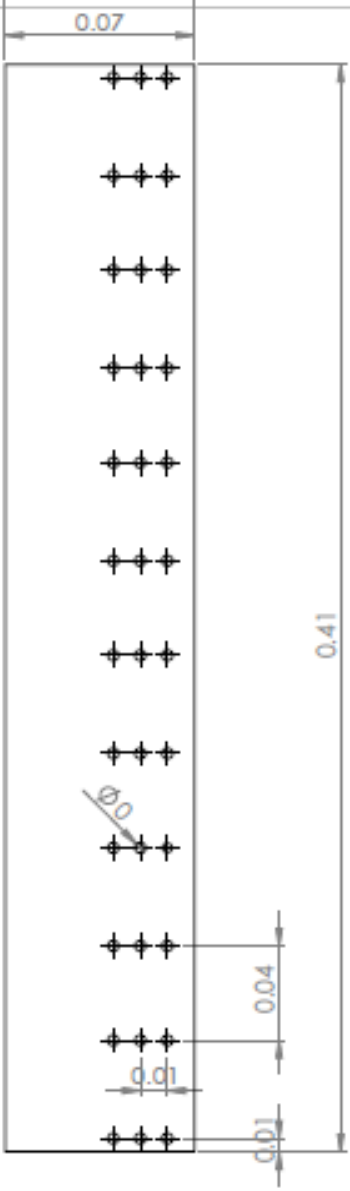


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACES FINISH TOLERANCES: LINEAR: ANGULAR:		FINISH:		ORDER AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
DRAWN		NAME		SIGNATURE		DATE		TITLE	
CHECKED									
APPROVED									
MFG									
Q.A.								DWG NO. Appendix3-Floor 6 ^{A4}	
								SCALE: 1:10	
								SHEET 1 OF 1	



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: LINEAR: ANGULAR:				RISK:		DESIGN AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
NAME		SIGNATURE		DATE				TITLE			
DRAWN								<div> DWG 100 Appendix3-Side Walks </div>			
CHECKED											
APPROVED											
MFG											
Q.A.				MATERIAL:				<div> DWG 100 Appendix3-Side Walks </div>			
				WEIGHT:				SCALE: 1:20		SHEET 1 OF 1	

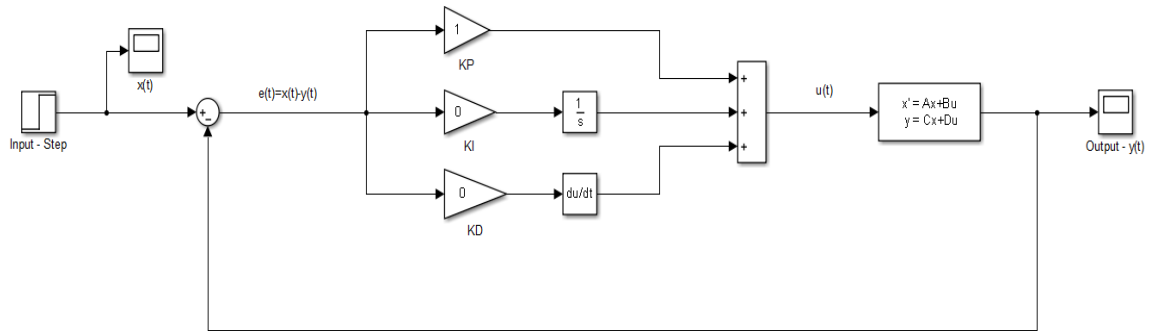


																																														
					SCALE 1 : 2																																									
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:					FINISH:		DRESS AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION																																			
<table border="1"> <thead> <tr> <th></th> <th>NAME</th> <th>SIGNATURE</th> <th>DATE</th> <th></th> <th></th> </tr> </thead> <tbody> <tr> <td>DRAWN</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>CHECKED</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPROVED</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>ENG</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>QA</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>						NAME	SIGNATURE	DATE			DRAWN						CHECKED						APPROVED						ENG						QA						TITLE:		DWG NO:		SHEET 1 OF 1	
	NAME	SIGNATURE	DATE																																											
DRAWN																																														
CHECKED																																														
APPROVED																																														
ENG																																														
QA																																														
MATERIAL:					SENSORS FRONT MOUNT#4		SCALE: 1:2		SHEET 1 OF 1																																					
WEIGHT:					SCALE: 1:2		SHEET 1 OF 1		SHEET 1 OF 1																																					

**Appendix 4: Weights of the Parts and
the Computations of needed Torque and RPM**

Unit	Hardware	Qty.	Weight (grams)	
			Unit	Total
Cognitive	Single Board Computer	1		
Motion Mechanism	Microcontroller Board	2	50	100
	Chassis	1	5000	5000
	Wheels	4	200	800
	Servo DC Motor	2	350	700
	IR Sensor	12	20	240
	Compass	1	20	20
	Ultrasonic Distance Sensor	4	20	80
	Battery	1	200	200
Vision System	Digital Webcam	1		
	Laser Scanner	1		
	Embedded Computer	1		
Audition System	Microphone	1		
	Embedded Computer	1		
Environmental Sensors	IR	1		
	Gas / Smoke	1		
	Temperature	1		
LCD Touch Screen		1	200	200
Wires and Others		1	600	600
Body		1	10000	10000
			Total	17940

Appendix 5: Simulink Model of the Robot's Dynamics

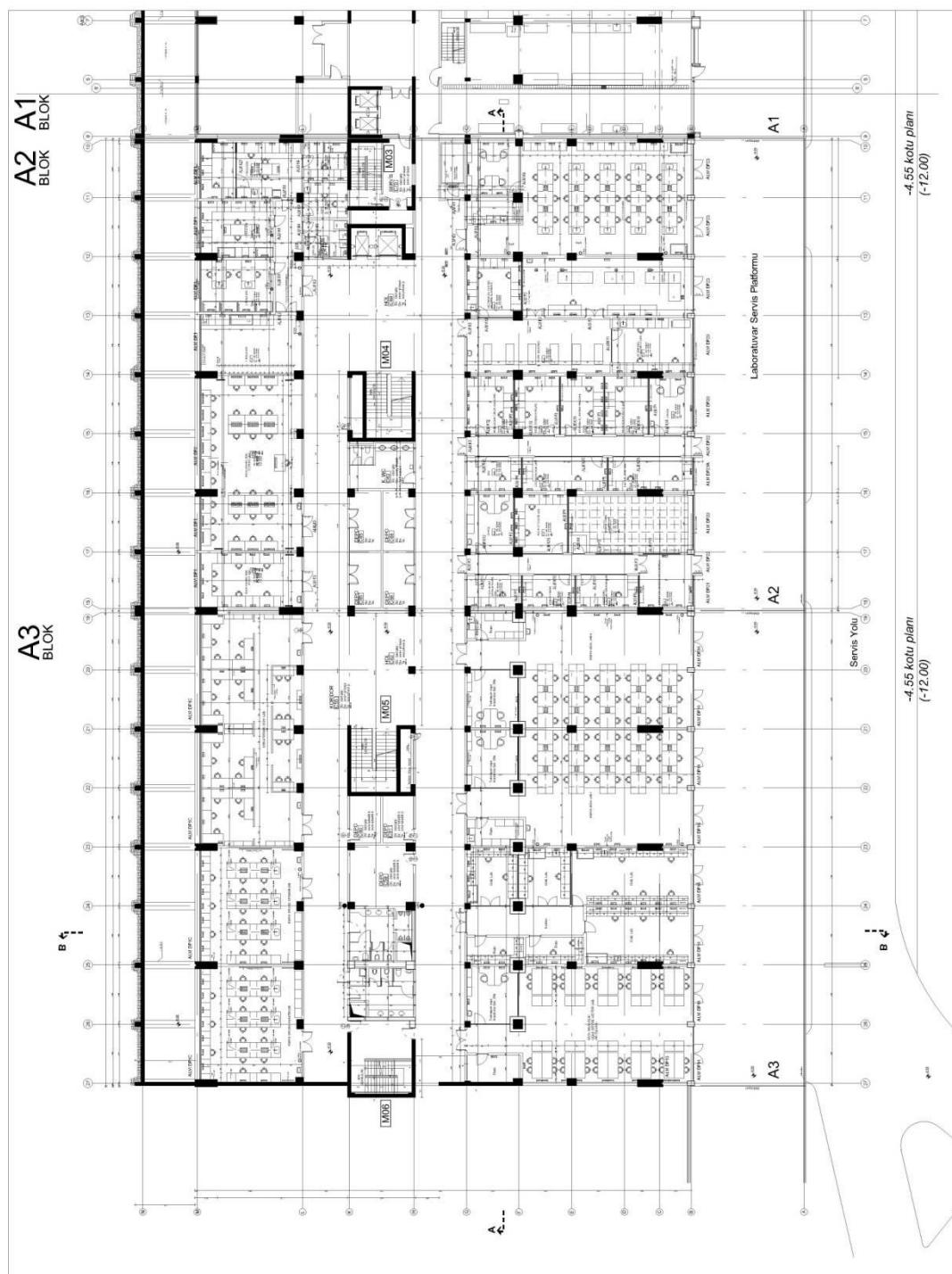


$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ c/m \end{bmatrix} \quad C = [0 \quad 1] \quad D = [0]$$

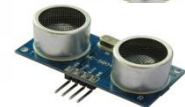
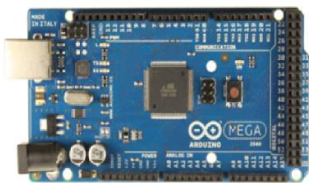
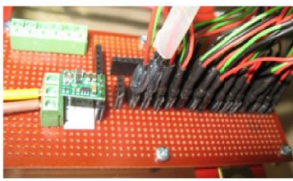
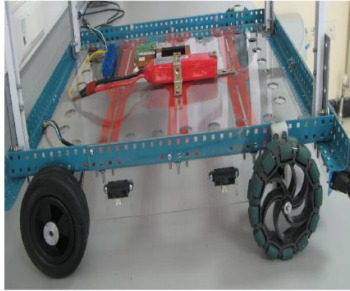
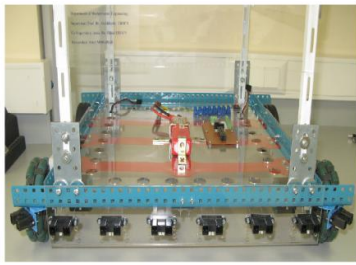
c: electro-mechanical transmission coefficient

m: weight of the robot

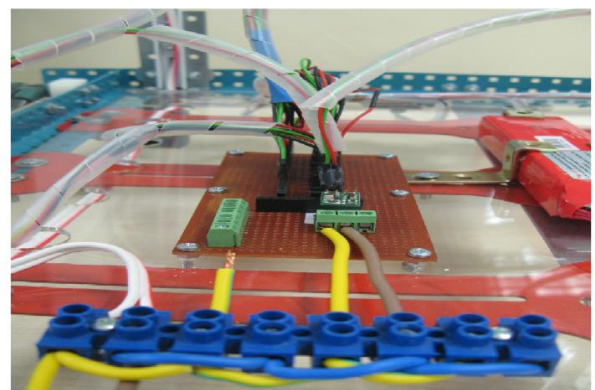
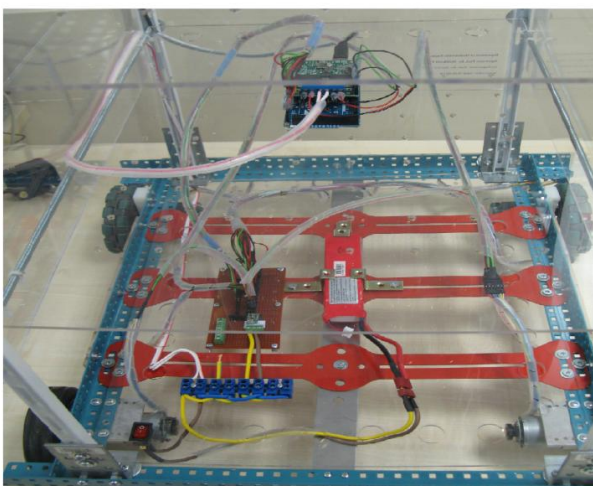
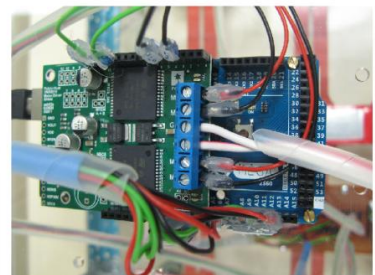
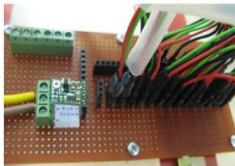
Appendix 6: Platform of the Laboratories of our Department

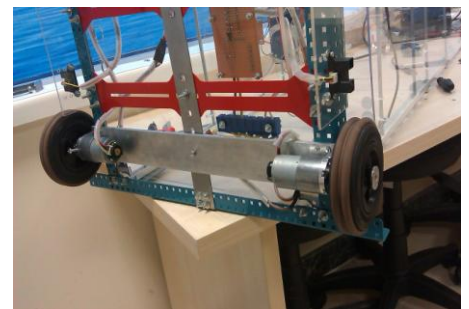
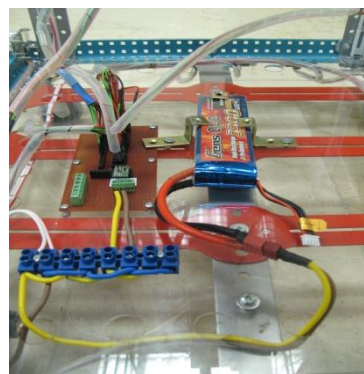
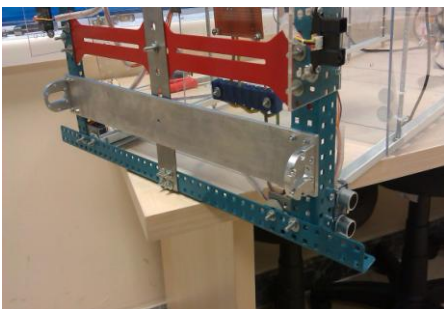
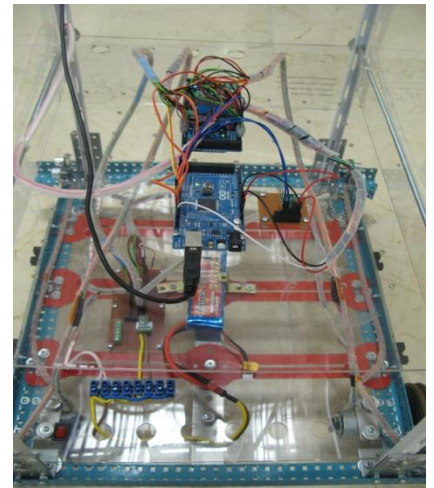
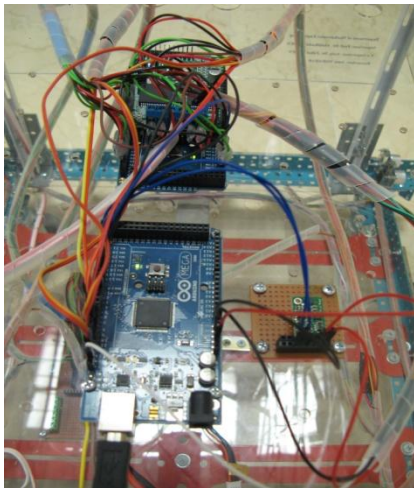
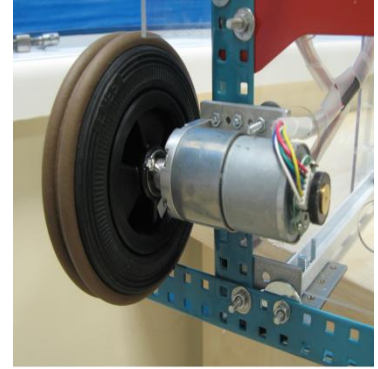
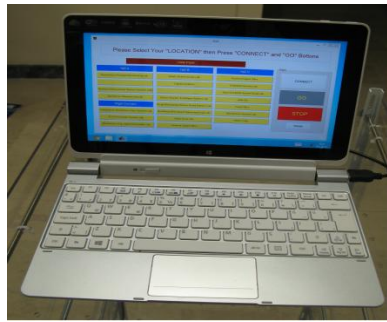
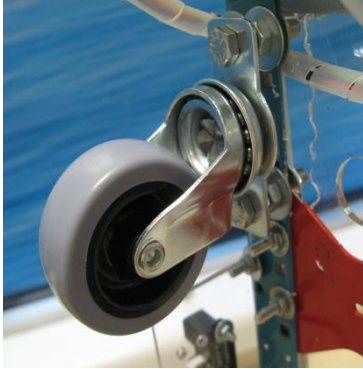


Appendix 7: Various Shots of the Assembling and Wiring Steps.



Department of Mechatronics Engineering
Supervisor: Prof. Dr. Abdülkadir ERDEN
Co-Supervisor: Assis. Dr. Zühal ERDEN
Researcher: Amir NOBAHAR





Appendix 8

```
// This program is uploaded on SMCB to handle Data from
// GUI, Ultrasonicsensors and the Compass and then send
// them to MNCB in 3 parts.

#include <Wire.h>
#include <LSM303.h>

int trigPin1 = 6;    //Trigger Signal of Sensor 1
int echoPin1 = 7;    //Echo Signal of Sensor 1
int trigPin2 = 8;    //Trigger Signal of Sensor 2
int echoPin2 = 9;    //Echo Signal of Sensor 2
int trigPin3 = 10;   //Trigger Signal of Sensor 3
int echoPin3 = 11;   //Echo Signal of Sensor 3
int trigPin4 = 12;   //Trigger Signal of Sensor 4
int echoPin4 = 13;   //Echo Signal of Sensor 4
long duration1, duration2, duration3, duration4;
float mm1, mm2, mm3, mm4, m1, m2, m3, m4, Wd1, Wd2, Wd;

int P;

LSM303 compass;

void setup() {
  Serial.begin (9600);
  Serial2.begin (9600);
  Serial3.begin(9600);

  pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);
  pinMode(trigPin3, OUTPUT);
  pinMode(echoPin3, INPUT);
  pinMode(trigPin4, OUTPUT);
  pinMode(echoPin4, INPUT);

  Wire.begin();
  compass.init();
  compass.enableDefault();
  compass.m_min = (LSM303::vector<int16_t>){-987, -1321, -2294};
  compass.m_max = (LSM303::vector<int16_t>){+2166, +1879, -1981};
  // the compass.m_min & compass.m_max data is achieved by the
  //Calibration program
}

void loop()
{

  // Part 1: Receive Data from GUI and send it to MNCB

  if(Serial.available()>0)
  {
    P=Serial.read();
    if (P==1||P==2||P==3||P==4||P==5||P== 6||
        P==7||P==8||P==9||P==11||P==12||P==13
        ||P==14||P==15||P==16||P==17||P==18||
```



```

        P==19||P==20||P==21||P==22||P==23||
        P==24||P==25||P==26||P==27||P == 28)
    {
Serial.write(P);
    }
}

// Part 2: Handling the Ultrasonicsensors and send the
// computed data to MNCB

digitalWrite(trigPin1, LOW);
delayMicroseconds(5);
digitalWrite(trigPin1, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin1, LOW);
pinMode(echoPin1, INPUT);
duration1 = pulseIn(echoPin1, HIGH);
mm1 = duration1 / 5.82;
if (mm1<3000) m1=mm1;
else m1=0;

digitalWrite(trigPin2, LOW);
delayMicroseconds(5);
digitalWrite(trigPin2, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin2, LOW);
pinMode(echoPin2, INPUT);
duration2 = pulseIn(echoPin2, HIGH);
mm2 = duration2 / 5.82;
if (mm2<3000) m2=mm2;
else m2=0;

digitalWrite(trigPin3, LOW);
delayMicroseconds(5);
digitalWrite(trigPin3, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin3, LOW);
pinMode(echoPin3, INPUT);
duration3 = pulseIn(echoPin3, HIGH);
mm3 = duration3 / 5.82;
if (mm3<3000) m3=mm3;
else m3=0;

digitalWrite(trigPin4, LOW);
delayMicroseconds(5);
digitalWrite(trigPin4, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin4, LOW);
pinMode(echoPin4, INPUT);
duration4 = pulseIn(echoPin4, HIGH);
mm4 = duration4 / 5.82;
if (mm4<3000) m4=mm4;
else m4=0;

Wd1=(asin((m1-m2)/445));
Wd2=(asin((m3-m4)/445));

if (mm2<mm3)
    Wd=Wd1;

```

```
else
    Wd=Wd2;

int Wdd=Wd*180/3.1416;

Serial2.write(Wdd);

// Part 3: Handling the Compass and send the needed data to MMCB

compass.read();
int heading = compass.heading();
Serial3.write(270-heading);
delay(100);
}
```

Appendix 9

```
// This calibration program gives the compass.m_min & compass.m_max
// data to use in the Compass program. After running, change the
// heading of the Robot from minimum to maximum manually.

#include <Wire.h>
#include <LSM303.h>

LSM303 compass;
LSM303::vector<int16_t> running_min = {32767, 32767, 32767},
running_max = {-32768, -32768, -32768};

char report[80];

void setup() {
  Serial.begin(9600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
}

void loop() {
  compass.read();

  running_min.x = min(running_min.x, compass.m.x);
  running_min.y = min(running_min.y, compass.m.y);
  running_min.z = min(running_min.z, compass.m.z);

  running_max.x = max(running_max.x, compass.m.x);
  running_max.y = max(running_max.y, compass.m.y);
  running_max.z = max(running_max.z, compass.m.z);

  snprintf(report, sizeof(report), "min: {%+6d, %+6d, %+6d}    max:
{%+6d, %+6d, %+6d}",
    running_min.x, running_min.y, running_min.z,
    running_max.x, running_max.y, running_max.z);
  Serial.println(report);

  delay(100);
}
```

Appendix 10

```
/* These interrupt service routines, written in C code, update */
/* the position of our two encoders at each sample time.

#include <Arduino.h>

typedef struct { int pinA; int pinB; double pos; int del;} Encoder;
volatile Encoder Enc[3] = {{0,0,0,0}, {0,0,0,0}, {0,0,0,0}};

/* auxiliary function to handle encoder attachment */
int getIntNum(int pin) {
/* returns the interrupt number for a given interrupt pin */
    switch(pin) {
        case 21:
            return 2;
        case 20:
            return 3;
        case 19:
            return 4;
        case 18:
            return 5;
        default:
            return -1;
    }
}

/* auxiliary debouncing function */
void debounce(int del) {
    for (int k=0;k<del;k++) {
        /* can't use delay in the ISR so need to waste some time
           performing operations */
        k = k +0.0 +0.0 -0.0 +3.0 -3.0;
    }
}

/* Interrupt Service Routine: change on pin A for Encoder 1 */
void irsPinAEn1(){

    /* read pin B right away */
    int drB = digitalRead(Enc[1].pinB);

    /* possibly wait before reading pin A, then read it */
    debounce(Enc[1].del);
    int drA = digitalRead(Enc[1].pinA);

    /* this updates the counter */
    if (drA == HIGH) {

        if (drB == LOW) { /* check pin B */
            Enc[1].pos++; /* going clockwise: increment */
        } else {
            Enc[1].pos--; /* going counterclockwise: decrement */
        }
    }
}
```

```

    } else { /* must be high to low on A */

        if (drB == HIGH) { /* check pin B */
            Enc[1].pos++; /* going clockwise: increment */
        } else {
            Enc[1].pos--; /* going counterclockwise: decrement */
        }

    } /* end counter update */

} /* end ISR pin A Encoder 1 */

/* Interrupt Service Routine: change on pin B for Encoder 1 */
void isrPinBEn1() {

    /* read pin A right away */
    int drA = digitalRead(Enc[1].pinA);

    /* possibly wait before reading pin B, then read it */
    debounce(Enc[1].del);
    int drB = digitalRead(Enc[1].pinB);

    /* this updates the counter */
    if (drB == HIGH) {

        if (drA == HIGH) { /* check pin A */
            Enc[1].pos++; /* going clockwise: increment */
        } else {
            Enc[1].pos--; /* going counterclockwise: decrement */
        }

    } else { /* must be high to low on B */

        if (drA == LOW) { /* check pin A */
            Enc[1].pos++; /* going clockwise: increment */
        } else {
            Enc[1].pos--; /* going counterclockwise: decrement */
        }

    } /* end counter update */

} /* end ISR pin B Encoder 1 */

/* Interrupt Service Routine: change on pin A for Encoder 2 */
void irsPinAEn2() {

    /* read pin B right away */
    int drB = digitalRead(Enc[2].pinB);

    /* possibly wait before reading pin A, then read it */
    debounce(Enc[2].del);
    int drA = digitalRead(Enc[2].pinA);

    /* this updates the counter */

```

```

if (drA == HIGH) {

    if (drB == LOW) { /* check pin B */
        Enc[2].pos++; /* going clockwise: increment */
    } else {
        Enc[2].pos--; /* going counterclockwise: decrement */
    }

} else { /* must be high to low on A */

    if (drB == HIGH) { /* check pin B */
        Enc[2].pos++; /* going clockwise: increment */
    } else {
        Enc[2].pos--; /* going counterclockwise: decrement */
    }

} /* end counter update */

} /* end ISR pin A Encoder 2 */

/* Interrupt Service Routine: change on pin B for Encoder 2 */
void isrPinBEn2(){

    /* read pin A right away */
    int drA = digitalRead(Enc[2].pinA);

    /* possibly wait before reading pin B, then read it */
    debounce(Enc[2].del);
    int drB = digitalRead(Enc[2].pinB);

    /* this updates the counter */
    if (drB == HIGH) {
        if (drA == HIGH) { /* check pin A */
            Enc[2].pos++; /* going clockwise: increment */
        } else {
            Enc[2].pos--; /* going counterclockwise: decrement */
        }

    } else { /* must be high to low on B */

        if (drA == LOW) { /* check pin A */
            Enc[2].pos++; /* going clockwise: increment */
        } else {
            Enc[2].pos--; /* going counterclockwise: decrement */
        }

    } /* end counter update */

} /* end ISR pin B Encoder 2 */

void enc_init(int enc, int pinA, int pinB) {

    /* enc is the encoder number and it can be 1 or 2 */
    /* store pinA and pinB in global encoder structure Enc */
    /* they will be needed later by the interrupt routine */
    /* that will not be able to access s-function parameters */

```

```

Enc[enc].pinA=pinA;      /* set pin A */
Enc[enc].pinB=pinB;      /* set pin B */

/* set encoder pins as inputs */
pinMode(Enc[enc].pinA, INPUT);
pinMode(Enc[enc].pinB, INPUT);

/* attach interrupts */
switch(enc) {

    case 1:
        attachInterrupt(getIntNum(Enc[1].pinA), irsPinAEn1,
CHANGE);
        attachInterrupt(getIntNum(Enc[1].pinB), isrPinBEn1,
CHANGE);
        break;
    case 2:
        attachInterrupt(getIntNum(Enc[2].pinA), irsPinAEn2,
CHANGE);
        attachInterrupt(getIntNum(Enc[2].pinB), isrPinBEn2,
CHANGE);
        break;
}

}

int enc_output(int enc) {
    return ((double)Enc[enc].pos);
}

```

Appendix 11

```
% This function ensures that w (Robot's Angular Velocity) is
% respected as best as possible by scaling v (Robot's Linear
% Velocity.

function [v,w]= fcn(vi,wi)

vel_max=9;          % A motor's maximum forward angular velocity
(rad/sec)
vel_min=0.7;        % A motor's minimum forward angular velocity
(rad/sec)

R=5;                % Radius of the wheels
L=47;               % Distance between the wheels

if abs(vi)>0
    % Limit v,w to be possible in the range
    % [vel_min,vel_max], avoid stalling or exceeding motor
    limits

    v_lim=max(min(abs(vi), (R/2)*(2*vel_max)), (R/2)*(2*vel_min));
    w_lim=max(min(abs(wi), (R/L)*(vel_max-vel_min)), 0);

    % Compute the desired curvature of the robot's motion

    v_rd=(2*v_lim+L*w_lim)/(2*R);
    v_ld=(2*v_lim-L*w_lim)/(2*R);

    % Find the max and min v_rd & v_ld

    v_rl_max= max(v_rd,v_ld);
    v_rl_min= min(v_rd,v_ld);

    % Shift v_r & v_l if they exceed max/min level

    if v_rl_max>vel_max
        v_r=v_rd-(v_rl_max-vel_max);
        v_l=v_ld-(v_rl_max-vel_max);

    elseif v_rl_min<vel_min
        v_r=v_rd+(vel_min-v_rl_min);
        v_l=v_ld+(vel_min-v_rl_min);
    else
        v_r=v_rd;
        v_l=v_ld;
    end

    % Fix signs (Always either both positive or negative)

    v_sh=(R/2)*(v_r+v_l);
    w_sh=(R/L)*(v_r-v_l);
    v=sign(vi)*v_sh;
    w=sign(wi)*w_sh;
```



```

else
    % Robot is stationary, so we can either not rotate,
    % or rotate with some minimum/maximum angular
    % velocity

    w_min=(R/L)*2*vel_min;
    w_max=(R/L)*2*vel_max;

    if abs(wi)>w_min
        w=sign(wi)*max(min(abs(wi),w_max),w_min);
        v=0;
    else
        w=0;
        v=0;
    end
end
end

```

Appendix 12

```
% This function creates our designed GUI and allocates a number to
% each point. Then send the selected point serially to the MMCB.

function varargout = GUI(varargin)

% Last Modified by GUIDE v2.5 26-Mar-2015 12:55:24

% Begin initialization code

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',   @GUI7_OpeningFcn, ...
                  'gui_OutputFcn',    @GUI7_OutputFcn, ...
                  'gui_LayoutFcn',    [] , ...
                  'gui_Callback',     []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code

% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function SelectYourLocation_Callback(hObject, eventdata, handles)

function SelectYourLocation_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```

% --- Executes on button press in InitialPoint.
function InitialPoint_Callback(hObject, eventdata, handles)
global P
P=1;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Initial Point'));

% --- Executes on button press in HallA.
function HallA_Callback(hObject, eventdata, handles)
global P
P=2;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Entrance of Hall A'));

% --- Executes on button press in MechatronicsPrototype.
function MechatronicsPrototype_Callback(hObject, eventdata, handles)
global P
P=3;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Mechatronics Prototype Manufacturing Laboratory in Hall
A'));

% --- Executes on button press in AEmpty.
function AEmpty_Callback(hObject, eventdata, handles)
global P
P=4;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Empty in Hall A'));

% --- Executes on button press in MechanicalMeasurements.
function MechanicalMeasurements_Callback(hObject, eventdata,
handles)
global P
P=5;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Mechanical Measurements, Machine Elements and CAD
Laboratory in Hall A'));

% --- Executes on button press in MechatronicsManufacturing.
function MechatronicsManufacturing_Callback(hObject, eventdata,
handles)
global P
P=6;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Mechatronics Manufacturing Laboratory in Hall A'));

% --- Executes on button press in HallB.
function HallB_Callback(hObject, eventdata, handles)
global P
P=7;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Entrance of Hall B'));

% --- Executes on button press in RobotsStudent.
function RobotsStudent_Callback(hObject, eventdata, handles)
global P
P=8;

```

```

set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Robots Student Society Laboratory in Hall B'));

% --- Executes on button press in CognitiveRobotics.
function CognitiveRobotics_Callback(hObject, eventdata, handles)
global P
P=9;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Cognitive Robotics Laboratory in Hall B'));

% --- Executes on button press in BEmpty.
function BEmpty_Callback(hObject, eventdata, handles)
global P
P=11;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Empty in Hall B'));

% --- Executes on button press in SensorActuator.
function SensorActuator_Callback(hObject, eventdata, handles)
global P
P=12;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Sensor, Actuator and Intelligent Systems Laboratory in
Hall B'));

% --- Executes on button press in DesignMethodology.
function DesignMethodology_Callback(hObject, eventdata, handles)
global P
P=13;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Design Methodology and Behavior Based Robotics Laboratory
in Hall B'));

% --- Executes on button press in BioMimetic.
function BioMimetic_Callback(hObject, eventdata, handles)
global P
P=14;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Bio-Mimetic / Bio-Inspired Engineering Design Laboratory
in Hall B'));

% --- Executes on button press in RobotVision.
function RobotVision_Callback(hObject, eventdata, handles)
global P
P=15;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Robot Vision Laboratory in Hall B'));

% --- Executes on button press in TechnicalSupportB.
function TechnicalSupportB_Callback(hObject, eventdata, handles)
global P
P=16;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Technical Support Office in Hall B'));

% --- Executes on button press in HallC.
function HallC_Callback(hObject, eventdata, handles)
global P

```

```

P=17;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Entrance of Hall C'));

% --- Executes on button press in TechnicalSupportC.
function TechnicalSupportC_Callback(hObject, eventdata, handles)
global P
P=18;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Technical Support Office in Hall C'));

% --- Executes on button press in EmbeddedSystems.
function EmbeddedSystems_Callback(hObject, eventdata, handles)
global P
P=19;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Embedded Systems Laboratory in Hall C'));

% --- Executes on button press in SpaseSatellite.
function SpaseSatellite_Callback(hObject, eventdata, handles)
global P
P=20;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Space and satellite System Design Laboratory in Hall C'));

% --- Executes on button press in RoboZoo.
function RoboZoo_Callback(hObject, eventdata, handles)
global P
P=21;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: RoboZoo in Hall C'));

% --- Executes on button press in ProjectOffice.
function ProjectOffice_Callback(hObject, eventdata, handles)
global P
P=22;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Projects Office in Hall C'));

% --- Executes on button press in MechatronicsSystems.
function MechatronicsSystems_Callback(hObject, eventdata, handles)
global P
P=23;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Mechatronics Systems Laboratory in Hall C'));

% --- Executes on button press in RobotsTown.
function RobotsTown_Callback(hObject, eventdata, handles)
global P
P=24;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Robots Town in Hall C'));

% --- Executes on button press in Corridor.
function Corridor_Callback(hObject, eventdata, handles)
global P
P=25;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Right Corridor'));

```

```

% --- Executes on button press in UndergraduateMechatronics.
function UndergraduateMechatronics_Callback(hObject, eventdata, handles)
global P
P=26;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Undergraduate Mechatronics Engineering Education
Laboratory'));

% --- Executes on button press in FESTO.
function FESTO_Callback(hObject, eventdata, handles)
global P
P=27;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: FESTO Pneumatic Systems Laboratory'));

% --- Executes on button press in MechatronicsEngSponsored.
function MechatronicsEngSponsored_Callback(hObject, eventdata, handles)
global P
P=28;
set(handles.SelectYourLocation,'String',num2str('Your Selected
Location: Mechatronics Engineering Sponsored Research Laboratory'));

% --- Executes on button press in connect.
function connect_Callback(hObject, eventdata, handles)
handles.output = hObject;
global s
handles.s=s;
handles.s=serial('COM4','BAUD', 9600);
fopen(handles.s);
guidata(hObject, handles);

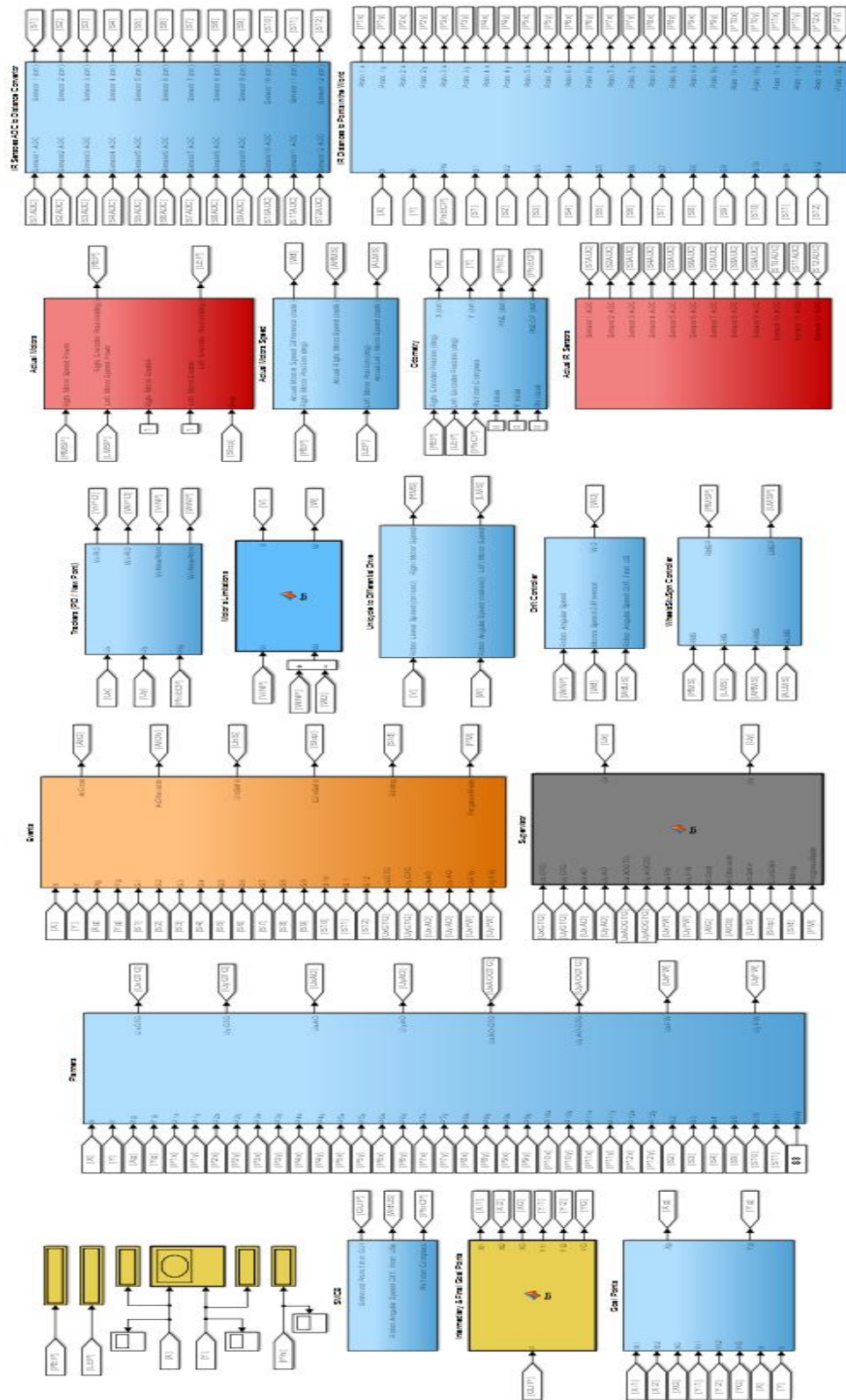
% --- Executes on button press in Go.
function Go_Callback(hObject, eventdata, handles)
handles.output = hObject;
global P
fprintf(handles.s,P);
guidata(hObject, handles);

% --- Executes on button press in Stop.
function Stop_Callback(hObject, eventdata, handles)
global s
handles.s=s;
handles.s=serial('COM4','BAUD', 9600);
fclose(handles.s);

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
close all

```

Appendix 13



145



Appendix 15

% This function performs the task of the Wheels Situ Spin Controller

```
function [RMSP,LMSP]= fcn(RMS,LMS,ARMS,ALMS)
```

```
if RMS>=0
    if ARMS<RMS
        if ARMS<0.8
            RMSR=0.8;
        else
            RMSR=ARMS+0.3333;
        end
    else
        RMSR=RMS;
    end
else
    if ARMS>RMS
        if ARMS>-0.8
            RMSR=-0.8;
        else
            RMSR=ARMS-0.3333;
        end
    else
        RMSR=RMS;
    end
end
RMSP=RMSR*100/9;

if LMS>=0
    if ALMS<LMS
        if ALMS<0.8
            LMSR=0.8;
        else
            LMSR=ALMS+0.3333;
        end
    else
        LMSR=LMS;
    end
else
    if ALMS>LMS
        if ALMS>-0.8
            LMSR=-0.8;
        else
            LMSR=ALMS-0.3333;
        end
    else
        LMSR=LMS;
    end
end
LMSP=LMSR*100/9;

end
```