

DEVELOPMENT OF A MOBILE ROBOT PLATFORM TO BE USED IN MOBILE ROBOT
RESEARCH

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUHAMMET KASIM GÖNÜLLÜ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

FEBRUARY 2013

Approval of the thesis:

DEVELOPMENT OF A MOBILE ROBOT PLATFORM TO BE USED IN MOBILE ROBOT RESEARCH

submitted by **MUHAMMET KASIM GÖNÜLLÜ** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Suha Oral
Head of the Department, **Mechanical Engineering**

Asst. Prof. Dr. A. Buğra Koku
Supervisor, Mechanical Engineering Dept., METU

Assoc. Prof. Dr. E. İlhan Konukseven
Co-Supervisor, Mechanical Engineering Dept., METU

Examining Committee Members:

Prof. Dr. Tuna Balkan
Mechanical Engineering Dept., METU

Asst. Prof. Dr. A. Buğra Koku
Mechanical Engineering Dept., METU

Assoc. Prof. Dr. E. İlhan Konukseven
Mechanical Engineering Dept., METU

Asst. Prof. Dr. Yiğit Yazıcıoğlu
Mechanical Engineering Dept., METU

Dr. Refik TOKSÖZ
Industrial Design Dept., METU

Date: February 1th, 2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Muhammet Kasım Gönüllü

Signature :

ABSTRACT

DEVELOPMENT OF A MOBILE ROBOT PLATFORM TO BE USED IN MOBILE ROBOT RESEARCH

Gönüllü, Muhammet Kasım

M.Sc., Department of Mechanical Engineering

Supervisor : Asst. Prof. Dr. A. Buğra Koku

Co-Supervisor : Assoc. Prof. Dr. E. İlhan Konukseven

February 2013, 143 pages

Robotics is an interdisciplinary subject and combines mechanical, computer and electrical engineering components together to solve different kinds of problems. In order to build robotic systems, these disciplines should be integrated. Therefore, mobile robots can be used as a tool in education for teaching engineering concepts. They can be employed to be used in undergraduate, graduate and doctorate research. Hands on experience on a mobile robot increase motivation of the students on the topic and give them precious practical knowledge. It also delivers students new skills like teamwork, problem solving, creativity, by executing robotic exercises. To be able to fulfill these outcomes, universities and research centers need mobile robot platforms that are modular, easy to build, cheap and flexible. However it should be also powerful and capable of being used in different research studies and hence be customizable depending on the requirements of these topics.

This thesis aims at building an indoor mobile robot that can be used as a platform for developing algorithms involving various sensors incorporated onto a mobile platform. More precisely, it can be used as a base for indoor navigation and localization algorithms, as well as it can be used as platform for developing algorithms for larger autonomous mobile robots. The thesis work involves the design and manufacturing of a mobile robot platform that can potentially facilitate mobile robotics research that involves use of various hardware to develop and test different perception and navigation algorithms.

Keywords: Modular Mobile Robotic Platform, Educational Robotics, Autonomous Mobile Robots, Robot Toolboxes and Kits, Robot Platforms, Localization, Navigation.

ÖZ

MOBİL ROBOT ARAŞTIRMALARINDA KULLANILMAK ÜZERE BİR MOBİL ROBOT PLATFORMU GELİŞTİRMESİ

Gönüllü, Muhammet Kasım

Yüksek Lisans, Makine Mühendisliği Bölümü

Tez yöneticisi : Yrd. Doç. Dr. A. Buğra Koku

Ortak tez yöneticisi : Doç. Dr. E. İlhan Konukseven

Şubat 2013, 143 sayfa

Robotik disiplinler arası bir konudur. Mekanik, bilgisayar ve elektrik mühendisliği parçalarından bir araya gelerek farklı soru türlerinin çözülmesini sağlar. Robotik sistemi inşa etmek için, bu disiplinler birleştirilmiş(entegre) olmalıdır. Bu yüzden, mobil robotlar mühendis kavramlarını öğretmeye yarayan eğitim aracı olarak kullanılabilir, lisans, yüksek lisans ve doktora araştırma konuları kullanılmak üzere istihdam edilebilir. Mobil robot üzerinde teoriden ziyade pratik bazı çalışmalarla tecrübe edinen öğrencilerin konu üzerindeki motivasyonları daha da artacak ve önemli bilgi ediniceklerdir. Ayrıca robotik egzersizleri uygulamak; araştırmacılara takım çalışması, problem çözme, ve yaratıcılık gibi yeni beceriler sunar. Bunun için, üniversiteler ve araştırma merkezlerinin kolay elde edilebilir, ucuz, esnek ve modüler mobil robot platformlarına ihtiyacı bulunmaktadır. Ancak bu platform araştırma konularının gerekliliklerine göre de güçlü ve farklı araştırma konularını yürütebilecek ve özelleştirilebilir olmalıdır.

Bu tez, akademik araştırmalarda algoritma geliştirmek için kullanılabilen, çeşitli sensörlerden faydalanan bir mobil robot geliştirmeyi amaçlar. Bir başka deyişle, geliştirilecek robot platformu iç ortam lokalizasyon ve navigasyon problemlerinin çözümünde algoritma geliştirme için kullanılabilir, bunun yanısıra daha büyük otonom mobil robotlara algoritma geliştirmek için bir platform olarak değerlendirilebilir. Bu tez çeşitli donanımları kullanarak değişik algılama ve navigasyon algoritmalarının geliştirilmesi ve testlerini içeren robotik araştırmaları kolaylaştıracak bir mobil robot platformunun tasarımını ve üretilmesini içerir.

Anahtar Kelimeler: Modüler Robotik Mobil Platformu, Eğitsel Robotik, Otonom Mobil Robot, Robot Takım ve Kitleri, Robot Platformu, Lokalizasyon, Navigasyon.

To My Lovely Family

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere appreciation to my supervisors Asst. Prof. Dr. A. Buğra Koku and Assoc. Prof. Dr. E. İlhan Konukseven for their invaluable guidance, advice, and criticism and support that made this study possible.

I would like to thank my colleagues for their support and help.

Also, I would like to thank my lovely family, especially my father Ahmet Gönüllü, for his invaluable efforts when I felt hopeless and weak in solving problems.

Finally, I would like to send my special thanks to my love and wife Betül for her understanding, help, encouragement and faith in me.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vi
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES.....	xiv
CHAPTERS	
INTRODUCTION.....	1
STATE OF THE ART	3
DESIGN PROCEDURE	11
3.1 Modular Frame Design	12
3.1.1 Modularity	14
3.2 Moving Mechanisms.....	14
3.2.1 Traction Mechanism	14
3.2.2 Shaft.....	14
3.2.3 Coupling	15
3.2.4 Bearing House	15
3.3 Hardware.....	16
3.3.1 Power Requirement Calculations	16
3.3.2 Motor	18
3.3.3 Energy Supply	19
3.3.4 Microcontroller.....	20
3.3.6 Motor Driver Board.....	21
3.3.7 Ultrasonic Ranging Module.....	22
3.3.8 Kinect	23
3.3.9 Digital Compass	23
ROBOT PLATFORM CONFIGURATION AND DESIGN PARAMETERS	27
CONTROL ARCHITECTURE DESIGN	31
5.1 Data Format	33
5.1.1 PC Side Commands	35
5.1.2 Microcontroller Side Commands.....	47
5.2 The Serial Packet Handling Algorithm.....	58
5.3 High Level Functions.....	58
5.3.1 Set Linear Speed.....	58
5.3.2 Turn Angle (Point Turn).....	59
5.3.3 Set Distance	62

5.4 Kinect Library.....	62
5.5 Robot Library	63
RESULTS AND DISCUSSIONS	67
6.1 Set Linear Speed Test	69
6.2 Turn Angle Test.....	78
6.3 Turn To Angle Test	79
6.4 Set Distance Test	81
6.5 Obstacle Avoidance Implementation.....	83
SUMMARY, CONCLUSIONS AND FUTURE WORK	87
REFERENCES	89
APPENDICES	
SERIAL PORT HANDLING ALGORITHM	93
SPEED TEST CALCULATION TABLE	95
LIBRARY REFERENCE (EDUROB.NAMESPACE)	97
Events	97
Methods.....	97
Enumerations and Constants	98
Example Usage of the Library.....	99
Library Code	102
Microcontroller Code	124

LIST OF TABLES

TABLES

Table 1 – Comparison of Similar Robots	9
Table 2 - Estimated coefficient of rolling resistance table [20].....	17
Table 3 - Encoder Cable Functions	19
Table 4 - PC Side Commands	33
Table 5 - Microcontroller Side Commands	34
Table 6 - Data Packet from PC.....	34
Table 7 - Data Packet from Microcontroller	34
Table 8 - Set Direction Command (PC Side)	35
Table 9 - Example Set Direction Data Packet (PC Side)	35
Table 10 - Get Direction Command (PC Side)	36
Table 11 - Example Get Direction Data Packet (PC Side).....	36
Table 12 - Set PWM Command (PC Side).....	36
Table 13 - Example Set PWM Data Packet (PC Side).....	37
Table 14 - Get PWM Command (PC Side).....	37
Table 15 - Example Get PWM Data Packet (PC Side)	37
Table 16 - Get Encoder Command (PC Side)	38
Table 17 - Example Get Encoder Data Packet (PC Side).....	38
Table 18 - Get Current Command (PC Side)	38
Table 19 - Example Get Current Data Packet (PC Side).....	38
Table 20 - Get Sonar Data Command (PC Side).....	39
Table 21 - Example Get Sonar Data Packet	39
Table 22 – Set Emergency Stop Command (PC Side)	39
Table 23 - Example Set Emergency Stop Data Packet.....	39
Table 24 – Reset Encoder Command (PC Side)	40
Table 25 - Example Reset Encoder Data Packet	40
Table 26 - Set RPM Command (PC Side).....	40
Table 27 - Example Set RPM Data Packet (PC Side)	40
Table 28 - Get RPM Command (PC Side)	41
Table 29 - Example Get RPM Data Packet (PC Side)	41
Table 30 - Get Heading Command (PC Side)	41
Table 31 - Example Get Accelerometer Data Packet (PC Side)	41
Table 32 - Get Accelerometer Command (PC Side)	42
Table 33 - Example Get Accelerometer Data Packet (PC Side)	42
Table 34 – Heartbeat Command (PC Side)	42
Table 35 - Example Heartbeat Data Packet.....	42
Table 36 - Motor Encoder Update Command (PC Side).....	43
Table 37 - Example Motor Encoder Update Data Packet (PC Side).....	43
Table 38 - Motor RPM Update Command (PC Side)	43
Table 39 - Example Motor RPM Update Data Packet (PC Side).....	44
Table 40 - Motor Current Update Command (PC Side).....	44
Table 41 - Example Motor Current Update Data Packet (PC Side)	44
Table 42 - Sonar Update Command (PC Side)	45
Table 43 - Example Sonar Update Data Packet (PC Side).....	45
Table 44 - Heading Update Command (PC Side)	45
Table 45 - Example Heading Update Data Packet (PC Side).....	45

Table 46 - Accelerometer Update Command (PC Side)	46
Table 47 - Example Accelerometer Update Data Packet (PC Side).....	46
Table 48 – Set Heartbeat Interval Command (PC Side).....	47
Table 49 - Example – Set Heartbeat Interval Data Packet (PC Side).....	47
Table 50 - Get Direction Command (Microcontroller Side)	47
Table 51 - Example Get Direction Data Packet (Microcontroller Side)	48
Table 52 - Get PWM Command (Microcontroller Side).....	48
Table 53 - Example Get PWM Data Packet (Microcontroller Side)	48
Table 54 - Get Encoder Command (Microcontroller Side).....	49
Table 55 - Example Get Encoder Data Packet (Microcontroller Side)	49
Table 56 - Get Current Command (Microcontroller Side)	50
Table 57 - Example Get Current Data Packet (Microcontroller Side)	50
Table 58 - Get Sonar Data Command (Microcontroller Side)	51
Table 59 - Get Sonar Data Packet (Microcontroller Side)	51
Table 60 - Get RPM Command (Microcontroller Side).....	52
Table 61 - Example Get RPM Data Packet (Microcontroller Side)	52
Table 62 - Get Heading Command (Microcontroller Side).....	52
Table 63 - Example Get Heading Data Packet (Microcontroller Side)	53
Table 64 - Get Accelerometer Command (Microcontroller Side).....	53
Table 65 - Example Get Accelerometer Data Packet (Microcontroller Side)	53
Table 66 – Heartbeat Command (Microcontroller Side).....	54
Table 67 - Example Heartbeat Data Packet (Microcontroller Side).....	54
Table 68 - Motor Encoder Update Command (Microcontroller Side)	54
Table 69 - Example Motor Encoder Update Data Packet (Microcontroller Side).....	54
Table 70 - Motor RPM Update Command (Microcontroller Side)	55
Table 71 - Example Motor RPM Update Data Packet (Microcontroller Side)	55
Table 72 - Motor Current Update Command (Microcontroller Side)	55
Table 73 - Example Motor Current Update Data Packet (Microcontroller Side).....	55
Table 74 - Sonar Update Command (Microcontroller Side)	56
Table 75 - Example Sonar Update Data Packet (Microcontroller Side)	56
Table 76 - Heading Update Command (Microcontroller Side).....	56
Table 77 - Example Heading Update Data Packet (Microcontroller Side)	56
Table 78 - Accelerometer Update Command (Microcontroller Side)	57
Table 79 - Example Accelerometer Update Data Packet (Microcontroller Side)	57
Table 80 - Low Level Functions	58
Table 81 - Speed Test 1 - 0.2 m/s.....	69
Table 82 - Speed Test 2 - 0.2 m/s.....	70
Table 83 - Speed Test 3 - 0.2 m/s.....	70
Table 84 - 0.2 m/s Test Results	71
Table 85 - Speed Test 4 - 0.4 m/s.....	71
Table 86 - Speed Test 5 - 0.4 m/s.....	72
Table 87 - Speed Test 6 - 0.4 m/s.....	72
Table 88 - 0.4 m/s Test Results	73
Table 89 - Speed Test 7 - 0.6 m/s.....	73
Table 90 - Speed Test 8 - 0.6 m/s.....	74
Table 91 - Speed Test 9 - 0.6 m/s.....	74
Table 92 - 0.6 m/s Test Results	75
Table 93 - Speed Test 10 - 0.8 m/s.....	75
Table 94 - Speed Test 11 - 0.8 m/s.....	76
Table 95 - Speed Test 12 - 0.8 m/s.....	76

Table 96 - 0.8 m/s Test Results	77
Table 97 - Speed Test Result.....	78
Table 98 – Turn Angle Test (90 degrees) Results	79
Table 99 – Turn Angle Test (45 degrees) Results	79
Table 100 – Turn to Angle Test (Turn To 20 degree) Results	80
Table 101 – Turn to Angle Test (Turn To 100 degree) Results	80
Table 102 – Turn to Angle Test (Turn To 100 degree) Results	81
Table 103 - Set Distance Test (1 m) Results	82
Table 104 - Set Distance Test (1 m) Results	83
Table 105 - Set Distance Test (1 m) Results	83
Table 106 - Speed Calculation Table	95
Table 107 - Events	97
Table 108 - Methods	97

LIST OF FIGURES

FIGURES

Figure 1 - Pioneer 3-AT [1]	3
Figure 2 - Koala II [2].....	3
Figure 3 - Gaia 2 [3].....	3
Figure 4 - Parallax Eddie Robot Platform [4]	4
Figure 5 - Calliope iRobot Create Netbook Development Platform [5].....	4
Figure 6 - Nexus Robot 4WD Mecanum Robot [6]	5
Figure 7 - Dr. Robot Sentinel3 Wi-Fi Mobile Development Platform [7]	5
Figure 8 - Dr. Robot Jaguar Tracked Mobile Platform [8].....	6
Figure 9 - InspectorBots Mega Bot Wireless 4WD Robot Platform [9]	6
Figure 10 - CoroWare CoroBot CB-W Robot Development Platform [10].....	7
Figure 11 - Seekur Jr [11]	7
Figure 12 - Tetra-DS IV Mobile Robot Platform [12]	7
Figure 13 – Volksbot [13].....	8
Figure 14 - MoRob Scalable Processing Box [14].....	8
Figure 15 - Chassis designed with aluminum profiles	13
Figure 16 - Chassis designed with sheet metal.....	13
Figure 17 - Chassis designed with sigma extrusions.....	14
Figure 18 – Motor Shaft Design.....	15
Figure 19 – Coupling	15
Figure 20 - Pillow Bearing.....	15
Figure 21 - Wheel Block.....	15
Figure 22 - Forces acting on the Robot Platform	16
Figure 23 - Pololu 37D 100rpm DC Motor [15]	18
Figure 24 - 37D 64 CPR Encoder [15].....	19
Figure 25 - 12V dry accumulator	20
Figure 26 - Arduino Mega module [16].....	21
Figure 27 - Rover 5 Motor Driver Board [17].....	22
Figure 28 - HC - SR04 Ultrasonic Ranging Module [18]	23
Figure 29 - Kinect Sensor [19].....	23
Figure 30 - Roll, Pitch and Heading angles [21].....	24
Figure 31 - Heading angle Calculation [21].....	24
Figure 32 - Tilted Position [21].....	25
Figure 33 - LSM303 Tilt Compensated Compass [22]	25
Figure 34 – Block diagram of electronic compass system [21]	25
Figure 35 - LSM303 Calibration Process Screenshot	26
Figure 36 - 2 wheeled configuration	27
Figure 37 - 4 wheeled configuration	27
Figure 38 - 6 wheeled configuration	28
Figure 39 - A mainboard configuration with a laser and ultrasonic sensor.....	28
Figure 40 - A notebook configuration with Kinect and ultrasonic sensors	29
Figure 41 - Motor Controller and Microcontroller Schematic	31
Figure 42 - Functional Block Diagram	32
Figure 43 - Motor Setpoint Data Flow	57
Figure 44 - Differential Drive kinematics [23]	59
Figure 45 - Forward Kinematics for Differential Robot	60
Figure 46 – Robot User Control in the Toolbox of MS Visual Studio.....	63

Figure 47 - User Control Screenshot.....	64
Figure 48 - Property Window of the User Control.....	65
Figure 49 - Example Usage of the User Control.....	66
Figure 50 - Robot Platform.....	67
Figure 51 - Inside of the Robot.....	68
Figure 52 - User Interface.....	68
Figure 53 - Test Setup of the Speed Test.....	69
Figure 54 - Speed Test 1 - 0.2 m/s.....	70
Figure 55 - Speed Test 2 - 0.2 m/s.....	70
Figure 56 - Speed Test 3 - 0.2 m/s.....	71
Figure 57 - Speed Test 4 - 0.4 m/s.....	72
Figure 58 - Speed Test 5 - 0.4 m/s.....	72
Figure 59 - Speed Test 6 - 0.4 m/s.....	73
Figure 60 - Speed Test 7 - 0.6 m/s.....	74
Figure 61 - Speed Test 8 - 0.6 m/s.....	74
Figure 62 - Speed Test 9 - 0.6 m/s.....	75
Figure 63 - Speed Test 10 - 0.8 m/s.....	76
Figure 64 - Speed Test 11 - 0.8 m/s.....	76
Figure 65 - Speed Test 12 - 0.8 m/s.....	77
Figure 66 - Sample Screenshot of Speed Test.....	77
Figure 67 - Sample Screenshot of Angle Before The Test.....	78
Figure 68 - Sample Screenshot of Angle After The Test.....	78
Figure 69 - Schematic of TurnAngle Tests.....	79
Figure 70 - Sample Screenshot of Angle Before The TurnToAngle Test.....	80
Figure 71 - Sample Screenshot of Angle After The TurnToAngle Test.....	80
Figure 72 - Sample SetDistance 1 m Test Screenshot.....	81
Figure 73 - Sample SetDistance 2 m Test Screenshot.....	82
Figure 74 - Sample SetDistance 3 m Test Screenshot.....	82
Figure 75 - Obstacle Avoidance Setup on the Robot.....	83
Figure 76 - Depth Map Divided into Three Windows.....	84
Figure 77 - Sample Obstacle Avoidance Screenshots.....	85
Figure 78 - Serial Port Handling Algorithm.....	93

CHAPTER 1

INTRODUCTION

From the invention of robots to present day, robots have been involved increasingly in our daily life. As a result of accelerated development of technology in the past several years, the use of robot increased exponentially and even more it will increase to a greater extent in the future.

Robots are used in a wide variety of fields. Robots are used in industry in every area since they are accurate, fast and productive, for example automotive industry. Also in medical industry, robots that can perform surgery are used greatly because in some surgery a small amount of hand shake will risk the patient. In addition, robots are used for removing landmines, defusing bombs, exploring dangerous areas where human life cannot be risked. Even more, robots are used in space, for example Moon and Mars, where humans cannot work or cannot work for long time. On the other hand, robots can also be used for simple tasks. To give an example, floor cleaning or vacuuming robots are this type of robots. Also, there are robot hobbyists where the only purpose is entertainment and fun. So, robots are in everywhere now in our life.

One of the hot topics in robotics is autonomous mobile robotics where robot is working without requiring human supervision. In this field, solutions for navigation, planning, localization, and similar problems are sought.

Many research institutions, universities and industry are investing a lot of resources and time on research in this field. They are trying to attract students and researchers to study in this field because of the potentials and they know the importance of the topic.

A typical observation was that various robotic applications always start from scratch. That is remarkable since most of the projects deal with similar problems. The development of mobile robotic systems is a demanding task regarding its complexity, required resources and skills in multiple fields such as software development, artificial intelligence, mechanical design, electrical engineering, signal processing, sensor technology or control theory. Current mobile robot systems often are monolithic, highly integrated prototypes that took long time to develop, they are costly and hard to maintain. Also robots tend to grow old quite quickly when used frequently or used under highly physical stress or when the robots hardware simply gets outdated after a few years. Fluctuation of people combined with long training times for new team members and loss of knowledge are other difficulties we frequently experienced.

The aim of this thesis is to cope with such diverse difficulties by using modular, component-oriented design approaches for mobile robot prototyping. The approach should enable developers to focus on their specific domain, still being able to have a clear comprehension of the entire system by help of different levels of abstraction and well defined interfaces to hardware and software modules. Furthermore, (re-usability) should be maximized by having well documented, system components of manageable size.

1.1 Problem Description

Most universities and other academic institutions use low-cost and commercially available robot platforms to introduce their students to robotics and to perform on them simple robot applications. These robot platforms are also the type usually purchased by private robot enthusiasts due to their low cost, ease of use, and availability. The main problem with these robot platforms is that they cannot be used for advanced robot applications, because they do not fulfill requirements such as:

- Processing power – needed for processing-intensive applications such as running algorithms in the field of machine learning, and image recognition.
- Flexibility
- Extendibility

- Power consumption – important for mobile robots, since they usually get their power supply from batteries.

Because of these restrictions, low-cost robot platforms are mostly only used for robotic introductory courses and in simple robotic applications by the academics, and by robotic enthusiasts who can afford them because of their low cost. A popular low-cost and commercially available robot platform that is widely used in the academics is the LEGO™ Mindstorms. For more advanced robotics applications, different classes of robot platforms are used. There are not many types of these platforms available commercially, or at least they are not easily available. This is probably due to the high cost connected with purchasing them, they often do not fulfill the specific requirements wanted, and because universities and other academic institutions usually want to gain new experiences and knowledge by designing and building their own robot platform. However, developing a new robot platform costs a lot of time and money, and besides the actual designing and implementing, the documentation has to be created, bugs and upgrades have to be continuously be made to it. Therefore, developing your own robot platform is an interesting choice when it is going to be reproduced in a large quantity, or if new experiences and know-how should be gained.

1.2 Thesis Objectives

The aim of this thesis is to build a mobile robot platform to be used for mobile robot research. The platform should enable securely mounting / carrying of different sensors and hardware. Also, the robot should safely navigate over common indoor obstacles such as carpets, doorsteps etc. And the robot should have odometry as well as a digital compass as basic sensors.

Basic sensors and actuators should be accessible to other platforms on the robot (such as a laptop, tablet pc etc.) through a standardized interface. Hence, basic communication protocols should be defined and implemented.

1.3 Overview of Thesis

The outline of the thesis is as follows. The next chapter “Literature Survey” is presented performed on the related subjects and robot platforms. After the literature survey, in chapter 3, the overall design procedure of the thesis is described. In chapter 4 “Robot Platform Configuration and Design Parameters”, the overall design configuration and parameters are introduced and described. Later, the control system design is described in chapter 5. The performance of the robot platform is tested in the chapter 6 “Results and Discussion”. And in the last chapter, chapter 7 “Conclusion and Future Works”, the conclusion and possible future work is studied.

CHAPTER 2

STATE OF THE ART

In this chapter the results of conducted literature survey are presented. Survey is based on the requirements and objectives stated at the introduction which revealed platforms for education and research. Several example platforms, found at the survey, are analyzed.

In the survey, Pioneer 3-AT robot is the first wheeled small sized mobile robot. Mobile Robots Inc. developed the Pioneer 3-AT robot which can be seen in Figure 1. It costs about \$4,000. The robot's locomotion system is formed as four wheel differential drive. This platform is able to move with speed up to 1.2m/s and able to climb up to 20° slopes. Its dimensions are 50cm by 49cm by 26cm. The platform can operate up to 4 hours on a single charge.



Figure 1 - Pioneer 3-AT [1]

Koala II is another popular wheeled robot platform (Figure 2). The K-TEAM Corporation developed the robot. Koala II is able to climb up to 43°. Also, the robot's maximum speed is 0.6m/s. In addition, base weight of the robot is 4kg with the batteries; maximum payload capacity is 3kg which can be mounted to the robot such as custom built accessories can be used in expansion modules. It consists of 32cm by 32cm by 20cm dimensions. It can operate about 4 hours.



Figure 2 - Koala II [2]

Another robot found within the scope of our survey is Gaia-2 (Figure 3). AAI Canada Inc. produces the robot. The robot's highest speed is 0.8m/s. Base weight of the robot is 40kg, also it can carry 20kg. Its dimensions are 49cm by 53cm by 26.5cm dimensions. It has ability to go over 15cm high obstacles. It can operate about 5 hours.



Figure 3 - Gaia 2 [3]

A laptop and a Microsoft Kinect sensor are used together on the Eddie robot platform in Figure 4. The laptop helps to control main unit of the robot, while Kinect is a robot sensing input device. Two 12 VDC electric motors energize the platform, each actuating a single wheel and blind spots are prevented by the IR and ultrasonic distance sensors which are not covered by the Kinect sensor.

The USB interface connects the platform and the laptop which is the only probable connection. The Parallax's own Propeller P8X32A microcontroller is the controller of the platform, which has 8 32-bit cores. It has an 8-channel 10-bit ADC and numerous I/O ports. 12V, and 14,4 AH gel-cell batteries create power, to sustain up to 7 hours activity the kit includes a charger. It costs approximately \$1.200.



Figure 4 - Parallax Eddie Robot Platform [4]

Carnegie Mellon University has developed the Tekkotsu-based robot. The robot is designed to have several developing application such as real-time motion control, forward and inverse kinematics, remote monitoring, teleoperation and localization. iRobot Create 4400 platform carries on Asus Eee PC 1000-series netbook which is essential kit for the Calliope iRobot seen in Figure 5. Its price tag is around \$1.000.

The Ubuntu Linux and the Tekkotsu software suite are pre-installed in the laptop. Written C++ it also makes use of third party libraries such as NEWMAT, libjpeg, libpng, libxml2 or zlib. An extensive set of commands and the documentation are being readily available. With the help of a USB, a 3 Ah NiMH battery and a charger connect the laptop and the platform's communication.



Figure 5 - Calliope iRobot Create Netbook Development Platform [5]

Nexus 4WD Mecanum robot (Figure 6) is an Omni wheeled 4- wheel drive robot platform. Its control is based on Arduino microcontroller.

Swedish wheel type has been quite useful for this platform because it enables the robot to move holonomically, i.e., the platform can move backward, forward, sideways or turn around itself where the speed and rotation direction of each wheel can be separately controlled. Its sturdy platform is made of aluminum alloy which is around 4 kilograms. There is a pivoting suspension mechanism in it to guarantee effective surface contact of each wheel.

Four 12V DC motors with encoders actuates the wheels, 4 ultrasonic sensors, 4 IR sensors and an Arduino I/O expansion board are presented. There is also a NiMh battery and a charger in it. The robot is nearly \$1.500.



Figure 6 - Nexus Robot 4WD Mecanum Robot [6]

The Dr. Robot Sentinel3 has several features such as navigation, recharging itself, and teleoperation. The Dr. Robot Sentinel3 base, which can be seen in Figure 7, weights 6kg, has a sturdy aluminum chassis and can lift up to 15 kg. Twin 12 VDC motors and integrated optical encoders actuate the platform. It has ability to connect to a standard 802.11b/g Wi-Fi network.

It has a 704x480 resolution camera, 3 ultrasonic sensors and 2 pyro electric sensors. It has 2 12 V 3.8 Ah NiMH batteries. It has a joystick for teleoperation and software for monitoring. The value for this platform is 12.000 USD.



Figure 7 - Dr. Robot Sentinel3 Wi-Fi Mobile Development Platform [7]

The Jaguar mobile platform (Figure 8) is the product of Dr. Robot; it is suitable for such condition such as extreme terrains. It has tracked rolling chassis with two 360 degree articulated arms, independent, and depending on customer specification, with the help of these features, the platform becomes versatile. In order to actuate the traction system 24VDC motors is used, it helps to climb slopes up to 45 degrees, and also it can surpass stairs and can reach the top speed of 7 km/h.

The robot has an extremely sturdy chassis; the base weight of the robot is around 22kg. The robot has ability to navigate autonomously and it can also connect to 802.11G/N Wi-Fi network. It has wide variety of sensors such as camera, IMU, GPS. This platform has a complete SDK with data protocols,

sample codes and support for MRDS, MS Visual Studio, and Matlab etc. It is powered by 22.2 V 10 Ah LiPo and the price for this robot is 11.000 USD.



Figure 8 - Dr. Robot Jaguar Tracked Mobile Platform [8]

A more affordable, yet sturdy and customizable platform are available from InspectorBots, in the form of the MegaBot 4WD robotic platform (Figure 9). It can operate indoor as well as outdoor and can be fitted with a wide range of auxiliary equipment. This is a heavy-duty platform, equipped with high-torque electric motors that provide a 900 kg towing capacity and can even carry a human. It can be used for entertainment purposes as well in rescue or security applications. The platform has a weight of about 80 kg and can tackle slopes up to 30 degrees. The chassis is made of steel and is water resistant.

It is a modular platform so that any hardware can be installed onto it, according to the requirements and applications. It has a price tag of around 7.000 US dollars.



Figure 9 - InspectorBots Mega Bot Wireless 4WD Robot Platform [9]

The CoroWare CoroBot CB-W Robot Development Platform is a capable, expandable and affordable robotic platform that comes fully assembled with an application to teleoperate right out of the box. The teleoperation software allows the user to remotely control the robot and read sensors. Complete source code is included.

CoroBot (Figure 10) was created to minimize the complexity of robot development. By combining a powerful PC-class platform with a robust, object-oriented software development system, the CoroBot is ready to rapidly deploy and develop robotics solutions. The CoroBot also assists the hardware developer with additional physical mounting space, ports, sensors and communication devices.



Figure 10 - CoroWare CoroBot CB-W Robot Development Platform [10]

Seekur Jr (Figure 11) is a skid steer, all-weather robot platform for research, security and inspection use. Seekur Jr's powerful drive motors allow movement up steep slopes and over rough terrain. Manipulation, sensing, additional computing and wireless communication can be added to fit any application.

Seekur Jr brings the advanced technology architecture of the larger Seekur robot to into smaller and more affordable package, and provides a much more robust and capable option than the Pioneer AT.



Figure 11 - Seekur Jr [11]

The Tetra-DS IV Mobile Robot Platform (Figure 12) is an advanced indoor robot platform used for developing and testing autonomous robot locomotion technology and programming. It is composed of the body and control boards. Modular design of the control boards allow for easy maintenance and future upgrade.

Use of differential drive and high performance AC servo motor provides superior speed and payload handling to the platform. Mounting holes are provided at top of the platform to conveniently attach various sensors and devices utilized for developing autonomous movement software.



Figure 12 - Tetra-DS IV Mobile Robot Platform [12]

The comparison of similar robots in the literature is given in Table 1.

So far the analyzed robot platforms are not modular as this thesis indicates. There are two examples in the literature where the objectives of this thesis are in parallel. They are described below.

First one, the Volksbot robot which is a flexible component based mobile robot system. Volksbot is a flexible and modular mobile robot construction kit. The component based approach offers a plug-in architecture with open interfaces in mechanics, electronic hardware and software. Quick integration of own modules combined with reuse of existing ones foster application-specific but effective system development.



Figure 13 – Volksbot [13]

The VolksBot construction kit addresses the rising demand for reusability in software, electronic hardware and mechanics by offering open and clearly defined interfaces as well as standardized components in all three fields.

The second one is MoRob toolbox. The core of MoRob is the Scalable Processing Box (SPB), a standardized processing platform for easy experimentation with any kind of robotic platform. Key characteristics are scalable performance, modular setup to facilitate tailoring to individual courses, flexible interfaces and easy configuration. Furthermore, MoRob aims to provide a standard set of control modules and teaching units, with a particular focus on project-based learning.



Figure 14 - MoRob Scalable Processing Box [14]

It is possible to rearrange SPBs in a similar way to Lego building blocks to fit different scenarios. The software architecture (Linux Real-time Environment - LiRE) offers a real time framework for easy use of interfaces and algorithms. All software is publicly available on a website (open source approach); therefore the toolbox is open for extensions and improvements by the educators and robotics community, as well as students.

Table 1 – Comparison of Similar Robots




			
	Mobile Robots Pioneer 3DX ^[1]	Pioneer 3-AT	Dongbu Robot Tetra-DS III ^[3]
Dimensions (LxWxH) [mm]	450x400x245	500x490x260	522x456x295
Weight [kg]	12	12	<20
Battery	12V, 7Ah	12V, 7Ah	24V 20Ah Li-PB
RunTime (hour)	8 (max.)	8 (max.)	8 (max.)
Charging Time [saat]	2,4 - 12	2,4 - 12	1,5
Driving System	2 wheel differential drive, one caster at rear	4 WD	2 wheel differential drive, one caster at rear
Wheel Dia [mm]	200	200	240
Drives	2 x Geared DC Motor	4x Geared DC Motor, 100 PPR Encoder	2 x Servo Motor
Max. Speed [m/s]	1,2	0,8	1,5
Max. Tilt Angle [%]	25	35	-
Max. Payload Capacity [kg]	23 (düz yüzey) 14 (%13 eğim)	15	80
Sensors	8 ultrasonic range finders at rear and front	8 ultrasonic range finders at rear and front Laser scanner DGPS	7 adet ultrasonic IR Sensor
Optional Sensor	Laser scanner 6DOF IMU GPS Camera	Gripper Stereo Range Finders Compass	Laser scanner GPS Pan/Tilt camera Sterovision camera Gyroscope
Mainboard	EBX Form Factor 3 PC/104+ can be inserted Versallogic Cobra SBC	EBX Form Factor 3 PC/104+ can be inserted Versallogic Cobra SBC	Nano ITX Form Factor VIA EP1A N800 32G SSD
Ek özellikler	Motion Control Button	Motion Control Button	Emergency Button Charge status led Speaker
Operating System/Software	Linux/Windows	Linux/Windows	Linux Ubuntu
Fiyat	10500 \$	10500 \$	13500\$

Table 1 (continued) - Comparison of Similar Robots

			
	Dr. Robot i90 ^[2]	Dr. Robot Jaguar ^[5]	Seekur Jr, Mobile Robots
Dimensions (LxWxH) [mm]	380x430x300	570x530x255	1198x835x494
Weight [kg]	5	19,5	77
Battery	2 x 3800mAh	22,2V 10Ah LiPo	3x24V NiMMH
RunTime (hour)	3 (max.)	2 (max.)	3,5 (max.)
Charging Time [saat]	-	-	3
Driving System	2 wheel differential drive, one caster at rear	4 WD	4WD
Wheel Dia [mm]	178	-	406
Drives	2x 12V Geared DC Motor, 800 PPR Encoder	4 x DC motor	
Max. Speed [m/s]	0,75	4,15	1,2
Max. Tilt Angle [%]	-	45	75
Max. Payload Capacity [kg]	15	30 (taşıma) 50 (çekme)	40
Sensors	740x480, 30 fps 10x dijital zoom camera 2 Pyroelectric Human Sensor 3 sonar sensor 7 IR range sensor	640x480, 30 fps color camera 5Hz GPS 9DOF IMU temperature sensor	Phytec MPC-565 Laser Range Finders Stereo Vision 6DOF IMU
Optional Sensor	RFID Accelerometer Laser scanner Temperature sensor	Laser scanner	Robotic Arm GPS
Mainboard	-	-	-
Ek özellikler	320x240 Touchscreen	Water and dust proof Resists 1.2 m falls	IP54
Operating System/Software	Microsoft Robotic Studio	Microsoft Robotic Studio	ARIA
Fiyat	6500\$	8000\$	8000\$

To sum up, although example robot platforms so far observed during the literature survey can carry different kinds of payload with respect to the different missions and even can change its locomotion type, they cannot be qualified as truly modular in the context of this thesis due to their lack of ability to have a flexible design capacity for the size and shape of their bodies which is one of the goals of the robot platform described in this thesis.

CHAPTER 3

DESIGN PROCEDURE

With the aim to develop an approach for multi-purpose robot prototyping, several design goals in hardware and software can be defined which are illustrated in the following. The goals are labeled in brackets (G1-G12) for later reference.

One of the major goals is to reduce costs, time and resources needed to conduct mobile robotic projects (G1). This should motivate more research groups from various backgrounds to start or continue activities related to mobile robotics in education and research. Also it should help to generate interest and open the market for new robot applications with more companies being willing to invest in robot technology and prototyping projects.

The complexity of recent robotic systems grows constantly with the complexity of the applications they are designed for. Modern mobile robots usually require a variety of different sensors, actuators and controllers but also algorithms and methods for signal processing, sensor data fusion, planning, localization, navigation and control of the robot, especially when being used in real world environments. The approach therefore should support the developers to manage this constantly growing system complexity (G2).

The system should allow the exchange and reuse of existing components in hardware and software (G3). For example it should not be necessary to start a system development from scratch every time a new robot needs to be built.

Also, an already existing robot platform should be easily reconfigurable and extendable by use of these hardware and software components (G4).

Reconfiguration and maintenance of the platform should be efficient and should not require special tools or machinery (G5). This way, developers are independent from having access to special facilities and experts and have more time to spend on research and development.

Often, groups already have worked in the domain of mobile robotics in the past. Therefore they should be able to efficiently integrate already existing technology into the system (G6).

The approach should help to foster the exchange and distribution of knowledge (G7). The design of such systems usually requires the interplay of many different individual skills which are distributed over a group or multiple groups of people.

The mechanics of the kit should be robust and scalable and allow for high payloads and high dynamics (G8).

To offer a wide range of possible applications, the kit should allow for diverse robot variants for different scenarios (G9).

The training periods for new users should be short (G10), so that they can produce results more quickly.

Here is a summarization of the design goals:

- G1. Reduce costs, time and resources in mobile robotics projects
- G2. Be able to manage system complexity
- G3. Allow exchange and reuse of existing components
- G4. Allow easy reconfiguration and extension of the systems
- G5. Allow simple and efficient maintenance
- G6. Allow efficient integration of existing technology
- G7. Foster exchange of knowledge

- G8. Robust and scalable mechanical design
- G9. Allow for a wide range of robot variants and applications
- G10. Allow for short training periods of new users

From these goals, we derived various design criteria for the construction kit in hardware, software and mechanics.

To reduce the costs and efforts for manufacturing and design of special components, standardized, available industrial components should be used if applicable (D1).

To keep the system complexity low and to be able to maintain the construction kit, the amount of components should be kept minimal, yet offering a high grade of re-configurability. (D2)

Components should possess a fine granularity and should be universal to ensure reuse (D3).

A comprehensive mechanical component library should be built up using standard CAD software tools (D4). Before actually building the robot, a complete design and simulation should be done in CAD avoiding major design errors and allowing fast iterations during the design phase.

The same holds true for software development, where a software library should be built up using state-of-the-art software development standards regarding architecture, documentation and coding conventions (D5).

Besides development of own software, existing software and frameworks should be used and integrated into the approach (D6).

When developing a component in hardware or software, documentation standards for developers and users should be applied (D7).

Different layers of abstraction should be provided during system integration and development in hardware and software (D8). This should help to reduce training times and allow a wide range of people from different technical background to work with the system. Clear interface definitions for hardware and software components have to be defined and maintained (D9).

Furthermore to keep the number of possible variants high and the system complexity low, dependencies between components should be avoided (D10).

Here is a summarization of the design criteria:

- D1. Extensive use of standardized, industrial components
- D2. Small number of different components with high reconfigurability
- D3. Fine granularity of modules to ensure reuse
- D4. Build up mechanical component library in CAD
- D5. Build up software library with documentation and coding standards
- D6. Use and integrate existing software and frameworks
- D7. Apply documentation standards for components
- D8. Introduce multiple abstraction layers
- D9. Clear interface definitions for hardware and software components
- D10. Avoid dependencies between components

3.1 Modular Frame Design

Several ways to produce the chassis is compared in this section. In order to meet the design requirements aluminum material has to be used in the robot chassis because of its weight advantage over steel.

Aluminum is the best option for chassis material because of its low density with respect to steel and ease of machining. However, steel is also seems an appropriate choice but its density is about three

times aluminum so it increases the weight of the robot. Other polymeric materials are not appropriate because of their low flexibility, low ductility and inferior machinability. Additionally, to access non-metallic materials is more difficult than metals. To sum up, after all considerations, aluminum will be the right choice.

Using aluminum profiles and welding a chassis can be constructed. An example construction is given in Figure 15. This technique can be used by using standard square aluminum profiles. Then, after constructing the frame the profiles should be welded to build the chassis. This construct is rigid, however, since the design requirements states that the platform should be modular, low cost and easy to use this technique is eliminated. Because welding process is high cost and time consuming.

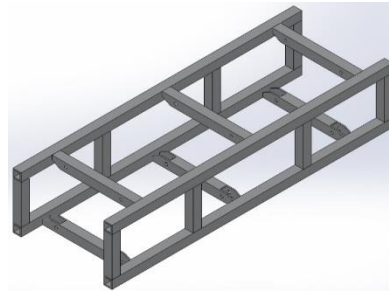


Figure 15 - Chassis designed with aluminum profiles

Another technique to construct the chassis is using sheet metals. The design is done using sheet metals. Then the parts are produced using laser cutters. After that, the parts should be bend according to designs. And later, the assembly of the chassis is realized. An example design is given in Figure 16. Just like the previous example, this technique is also not suitable for our design requirements.

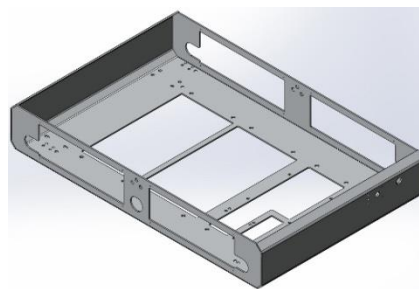


Figure 16 - Chassis designed with sheet metal

On the other hand, it is decided to use standard aluminum sigma extrusions (X-beams). They provide high rigidity, are light weight and offer a variety of different connections. Also, all the surface of these profiles can be used as connection surface. This feature is very important for our robot platform and design requirements because it satisfies and gives us the opportunity to be modular. An example design is given in Figure 17.

Size and shape of the robot's main frame can be adapted individually to the needs by simple mechanical processing, i.e. cutting and screwing. By using pluggable t-nut-connectors, it is possible to establish new connections without having to decompose the frame. All sides of the X-beams can be used to connect to additional elements. In our design, all hardware components are connected to the main-frame. Therefore only geometrical dependencies between the component and the main-frame occur, not between the components themselves. All components like batteries, motor controller, drive units and sensors are connected to a rectangular single layered main frame. With this, the repositioning of components and scaling of the platform can be easily done.

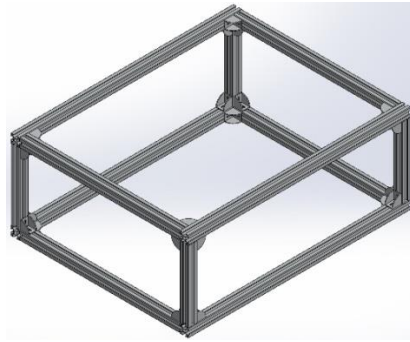


Figure 17 - Chassis designed with sigma extrusions

3.1.1 Modularity

We decide to separate the robot into two parts. The lower part of the robot which will become the mechanical platform and is worked out further in this chapter. This part will include the chassis and basic components of the platform, like motor, motor driver, battery, and controller. The upper part of the robot will be the platform which can contain the equipment needed to fulfill a desired robot task. In this way modularity is introduced in the design phase. The upper platform holds the computer, sensors and specific electronics and is mechanically coupled to the chassis.

3.2 Moving Mechanisms

The motor block of the robot is intended to be modular so that it can be separated from one platform and can be mounted to another combination of a platform. Therefore the analysis and the design are done accordingly.

3.2.1 Traction Mechanism

Pallet provides better traction performance in rough surfaces, but they have higher cost and harder to assemble. Wheels are easier to assemble but their traction is less than the pallet's. However, we plan to use the robot in indoor environment mostly. Therefore, we will use wheels because of its ease of assemble and low cost. In addition since the platform is modular, pallet can also be used according to the needs.

In design requirements it is specified that the robot should be modular and can be configurable according to the needs of the problem. Therefore, the driving unit should be designed accordingly. Whether wheel or pallet is used, it should work directly, without depending on another module. Therefore, it is not necessary to specify a number of wheels for the robot. It is defined by the problem that robot will be used for. So, a standard bearing housing is used as a connection to the chassis. The main structural element of the driving unit will be this bearing housing. It can be selected also according to the needs of the problem. After selecting it, one part –shaft- should be produced in order to connect the motor and the wheels. This part should be designed and manufactured. However, this part can also be designed modular so that different motors can be used. Only the coupling between the shaft and the motor should be selected accordingly. On the other side, the wheel size and diameter also can be selected differently and shaft should be designed so that the different diameter wheels can be used.

3.2.2 Shaft

The motor shaft which holds the wheel one side and motor other side is one part that is machined. It is designed to be used with different dimension wheel and different types of motors (however in this thesis we use one type of motor). The shaft is designed for wheel to be threaded to it, so that it can be used with different wheel dimension. This way the machining process is also decreased. The wheel will be secured by using contra nuts. The bearing is also secured with nuts. The coupling is mounted to connect to motor. In Figure 18, the design motor shaft can be seen.

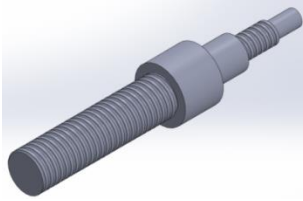


Figure 18 – Motor Shaft Design

3.2.3 Coupling

The coupling (Figure 19) will be the part that will connect the shaft to the motor. It is selected as a flexible coupling so that different types of motor can be used.

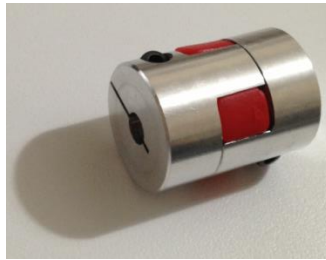


Figure 19 – Coupling

3.2.4 Bearing House

The bearing house (Figure 20) is also selected from the market, not designed and produced. The pillow bearing is suitable for our application. Different dimensions can be selected according to the needs of the application. We have selected the housing with the diameter of the bearing is 12mm.



Figure 20 - Pillow Bearing

Therefore, after the design and production the motor block of the platform is assembled and can be seen in Figure 21.

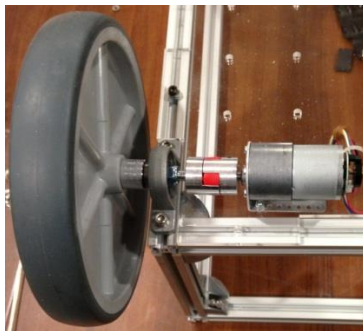


Figure 21 - Wheel Block

3.3 Hardware

In this section, we will explain the details about the hardware elements of the prototype. All of the hardware is selected so that it will be cost efficient and easy to get.

First minimum power required for this robot platform is calculated, then by using this value an appropriate motor and battery is selected, and then detailed design of the wheel motor system and main body is presented.

3.3.1 Power Requirement Calculations

For calculation of the minimum required power, maximum estimated total force and the maximum desired speed must be known. The maximum estimated total force is equal to the sum of gradient resistance force, air drag force (which can be assumed as zero), rolling resistance force and inertial force, which is equal to mass times acceleration.

$$F_{Total} = F_{gradient} + F_{air} + F_{rolling} + M_m \times a_{max} \quad (3.1)$$

Where:

F_{Total} : Maximum estimated total force

$F_{gradient}$: Gradient resistance force

F_{air} : Air resistance force

$F_{rolling}$: Rolling resistance force

M_m : Total mass of the robot platform

a_{max} : Maximum acceleration of the robot platform

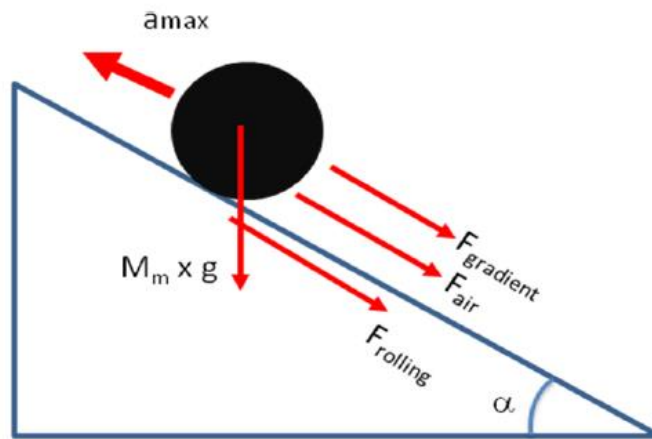


Figure 22 - Forces acting on the Robot Platform

The maximum total force calculation is performed for a four wheel configuration. The forces acting on the robot platform is shown in Figure 22. Calculations regarding these forces can be seen at below.

The first component that is calculated is the Gradient resistance force. Gradient resistance force is the component of the weight of the module that is parallel to the road. Maximum gradient force occurs when the robot module climbs a slope with the maximum designated angle which is determined as 20° . The formula of the gradient resistance is:

$$F_{gradient} = M_m \times \sin \alpha \times g \quad (3.2)$$

Where:

α : Angle of the slope that robot module is climbing up

The result of the (3.2) for 15 kilograms of estimated robot platform mass and 20° slope (because the robot will be mainly used indoor) is:

$$F_{gradient} = 15 \times \sin 20^\circ \times 9.81 \cong 50.33 \text{ N} \quad (3.3)$$

The third component, rolling resistance occurs mainly due to the deformation on the road and the tire surfaces and it can be calculated with the formula below:

$$F_{rolling} = f_r \times W \quad (3.4)$$

$$W = M_m \times \cos \alpha \times g \quad (3.5)$$

Where:

f_r : Coefficient of rolling resistance

W : Platform tire load

Maximum estimated total force is assumed to be occurring at robot moving on a slope and indoor. For this reason f_r is chosen as 0.020 from Table 2. By using this at formula (3.4) for a robot platform with 15 kg mass, the estimated rolling resistance is calculated as:

$$F_{rolling} = f_r \times W = f_r \times M_m \times \cos \alpha \times g = 0.020 \times 15 \times \cos 20^\circ \times 9.81 \cong 2.77 \text{ N} \quad (3.6)$$

Table 2 - Estimated coefficient of rolling resistance table [20]

Road Surface	f_r
Very good concrete	0.008-0.010
Average concrete	0.010-0.015
Concrete in poor condition	0.020
Very good tarmac	0.010-0.0125
Average tarmac	0.018
Tarmac in poor condition	0.023
Very good macadam	0.013-0.016
Average macadam	0.018-0.023
Dusty macadam	0.023-0.028
Good Stone paving	0.033-0.055
Stone paving in poor condition	0.033-0.055
Snow(50mm layer)	0.025
Snow(100mm layer)	0.037
Unmaintained natural road	0.080-0.160
Sand	0.150-0.300

The last component of the estimated total force comes from the acceleration. It is assumed that the robot has an acceleration of 0.5 m/s^2 at the maximum power need conditions. The estimated maximum total force can be found when the results of the equations (3.3), (3.6) and the maximum acceleration and determined mass of the module are put into the equation (3.1).

$$F_{Total} = F_{gradient} + F_{air} + F_{rolling} + M_m \times a_{max} = 50.33 + 0 + 2.77 + 15 \times 0.5$$

$$F_{Total} = 60.60 \text{ N} \tag{3.7}$$

After the calculation of the maximum total force, minimum required power output of the motor can be calculated by multiplying the maximum total force with the desired maximum operational speed. From here the power is calculated as:

$$P_M = F_{total} \times V_{max} \tag{3.8}$$

Where:

P_M : Minimum required power output of the motor

V_{max} : Determined maximum speed of the robot platform

Maximum desired speed for the robot platform is determined to be in the range of 0.6 to 1.2 m/s. For this calculation maximum speed is assumed to be 1.2 m/s. By putting the values of the variables of equation (3.8) minimum required power is found as:

$$P_M = F_{total} \times V_{max} = 60.60 \times 1.2 = 72.72 \text{ W} \tag{3.9}$$

It should be noted that calculated minimum estimated power requirement is for 4 wheeled robot platforms the power requirement of a single wheel is 18.18 W. When criteria such as procurement time, size, power output, price and aftermarket are taken into consideration, most eligible and affordable motor will be the Pololu 37D 12 V motors. The 100 rpm selection of this motor will give us roughly 23 W of power and this satisfies our power requirement.

3.3.2 Motor

This $2.71" \times 1.45" \times 1.45"$ gear motor is a powerful 12V brushed DC motor with a 102.083:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 64 counts per revolution of the motor shaft, which corresponds to 6533 counts per revolution of the gearbox's output shaft. These units have a 0.61"-long, 6 mm-diameter D-shaped output shaft. Key specs at 12 V: 100 RPM and 300 mA free-run, 220 oz.-in (16 kg-cm) and 5 A stall. The dimensions of the motor is given in Figure 23.

The face plate has six mounting holes evenly spaced around the outer edge threaded for M3 screws. These mounting holes form a regular hexagon and the centers of neighboring holes are 15.5 mm apart.

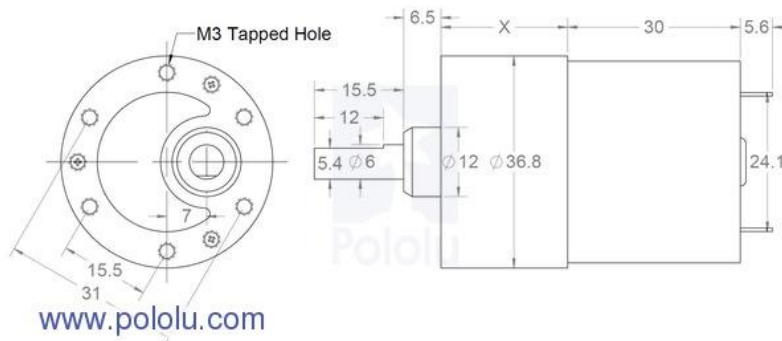


Figure 23 - Pololu 37D 100rpm DC Motor [15]

A two-channel Hall Effect encoder (Figure 24) is used to sense the rotation of a magnetic disk on a rear protrusion of the motor shaft. The quadrature encoder provides a resolution of 64 counts per

revolution of the motor shaft. To compute the counts per revolution of the gearbox output, multiply the gear ratio by 64. The motor/encoder has six color-coded, 11" (28 cm) leads:



Figure 24 - 37D 64 CPR Encoder [15]

The cabling of the motor and encoder is given in Table 3.

Table 3 - Encoder Cable Functions

Color	Function
Red	Motor power (connects to one motor terminal)
Black	Motor power (connects to the other motor terminal)
Green	Encoder GND
Blue	Encoder Vcc (3.5 – 20 V)
Yellow	Encoder A output
White	Encoder B output

The Hall sensor requires an input voltage, Vcc, between 3.5 and 20 V and draws a maximum of 10 mA. The A and B outputs are square waves from 0 V to Vcc approximately 90° out of phase. The frequency of the transitions tells you the speed of the motor, and the order of the transitions tells you the direction.

By counting both the rising and falling edges of both the A and B outputs, it is possible to get 64 counts per revolution of the motor shaft. Using just a single edge of one channel results in 16 counts per revolution of the motor shaft, so the frequency of the A output in the above oscilloscope capture is 16 times the motor rotation frequency.

3.3.3 Energy Supply

This platform needs an energy supply to be able to perform tasks autonomously. Choosing a battery configuration is a key feature in the design process. Actually this is an iterative process. The dimensions, voltage, weight and recharge method of the battery configuration form restrictions on the entire design process and determine directly the autonomy of the robot.

To achieve our requirements on energy supply we decide to use dry accumulator (Figure 25). The choices were lithium polymer batteries, dry accumulator or battery packs. In terms of weight, the dry accumulator is the heaviest one amongst these types. However, we would like our robot to be suitable for self-charging - yet this thesis work does not need to demonstrate autonomous self-charging. Therefore, choosing a lithium polymer battery would increase the difficulty of the self-charging process since the charging method for those batteries needs proper care and a special control unit is required. Charging, discharging, and storage all affect the lifespan of batteries. Also, LiPo batteries are still expensive compared to NiCad and NiMH, but coming down in price all the time. In addition, LiPo's don't last that long, perhaps only 300-400 charge cycles. Therefore, we eliminate the LiPo batteries.

On the other hand, a battery pack is a set of any number of (preferably) identical batteries or individual battery cells. They may be configured in a series, parallel or a mixture of both to deliver the desired voltage, capacity, or power density. An advantage of a battery pack is the ease with which it can be swapped into or out of a device. This allows multiple packs to deliver extended runtimes, freeing up the device for continued use while charging the removed pack separately. However, we want our platform suitable for self-charging and this choice is also not proper for the job. Also, compared to dry accumulator they are expensive.



Figure 25 - 12V dry accumulator

Nowadays a popular voltage seems to be 12V, because most of the popular electronics use 12V as power input.

The Robot will have output sockets for 12V and 5V.

3.3.4 Microcontroller

Arduino has become a popular open-source single-board microcontroller among electronic hobbyists, and it is gaining acceptance as a quick prototyping tool for engineering and educational projects also.

Suitable for educational purpose because;

- Low cost: The components must be affordable.
- Easy to assembly due to the constitution of the modules.
- Able to run on different platforms: the overall system can operate in different operating systems.
- Open hardware and open source: This means that the hardware and the software used has a public access. Anyone can use it and improve it.

Arduino offers several models with different characteristics. The main differences between the modules are the number of inputs and outputs, the type of microcontroller and the capacity of the Flash memory. Taking into account the requirements, the best option is the Mega module because it has the most input output pins.

As it can see in Figure 26, the Mega module provides different inputs and outputs, either analogs or digitals. Moreover, also provides PWM outputs. The main features of the Mega module are the following:

- Microcontroller : ATmega2560
- Operating Voltage : 5V
- Input Voltage (recommended) : 7-12V
- Input Voltage (limits) : 6-20V
- Digital I/O Pins : 54 (of which 15 provide PWM output)
- Analog Input Pins : 16
- DC Current per I/O Pin : 40 mA
- DC Current for 3.3V Pin : 50 mA
- Flash Memory : 256 KB of which 8 KB used by boot loader
- SRAM : 8 KB
- EEPROM : 4 KB

- Clock Speed : 16 MHz

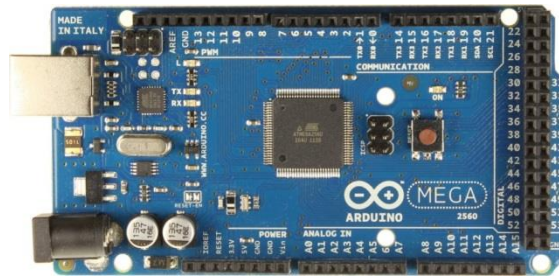


Figure 26 - Arduino Mega module [16]

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 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. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino
- Open source and extensible software- The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based.

Some example shields for Arduino are as follows:

- Wireless communication shield (Xbee)
- Motor Driver Shields
- LCD Shield
- Ethernet Shield, etc.

3.3.6 Motor Driver Board

Rover 5 Motor Driver Board (Figure 27) is ideal for any small 4-wheel drive robotic vehicle. With four motor outputs, four encoder inputs and current sensing for each motor.

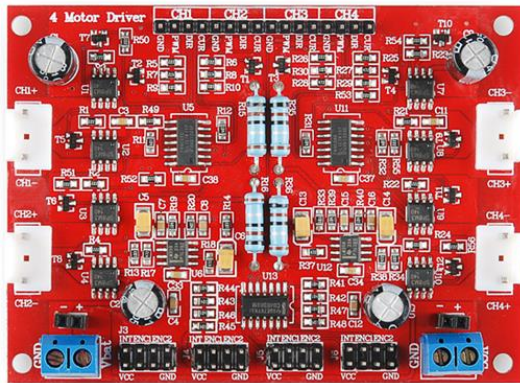


Figure 27 - Rover 5 Motor Driver Board [17]

The motor drivers can be controlled by simply applying logic 0 or 1 to the direction pin for that motor and a PWM signal to the speed pin. In this way, the speed and direction of four separate motors can be controlled independently from only 8 GPIO pins. The encoder inputs on the driver board mix each pair of encoder inputs using an XOR gate making it possible to read both inputs from a quadrature encoder using only one interrupt pin.

Reading the current sensor output is easy, each current sensor pin will output about 1V for each Amp of current drawn by the associated motor up to 5V. Connect the current sensor pin to the analog input of your controller and you'll be able to detect stalls and other motor problems.

There are two power connectors on board. One is for 5V logic and the other is the motor supply. Be sure to turn on your logic supply before applying the power source for your motors. The board is rated for a maximum motor supply voltage of 12V.

Features:

- 4 x Low Resistance FET “H” Bridges
- Each Channel Rated for 4A Stall Current
- Easy-to-Use Control Logic
- Current Monitoring for Each Channel.
- Quadrature Encoder Mixing Circuitry

Includes:

- 4 Channel Motor Driver Board
- Mounting Hardware

3.3.7 Ultrasonic Ranging Module

Ultrasonic ranging module HC - SR04 (Figure 28) provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time × velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground



Figure 28 - HC - SR04 Ultrasonic Ranging Module [18]

Electric Parameter

- Working Voltage : DC 5 V
- Working Current : 15mA
- Working Frequency : 40Hz
- Max Range : 4m
- Min Range : 2cm
- Measuring Angle : 15 degree
- Trigger Input Signal : 10uS TTL pulse
- Echo Output Signal Input : TTL lever signal and the range in proportion
- Dimension : 45*20*15mm

3.3.8 Kinect

Kinect is based on software technology developed internally by Rare, a subsidiary of Microsoft Game Studios owned by Microsoft, and on range camera technology by PrimeSense, which interprets 3D scene information from a continuously projected infrared structured light. The Kinect is equipped with two sensors - a near infra-red camera used for depth detection, and a color camera. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions.

The Kinect sensor (Figure 29) outputs video at a frame rate of 30 Hz. The RGB video stream uses 8-bit VGA resolution (640 × 480 pixels) with a Bayer color filter, while the monochrome depth sensing video stream is in VGA resolution (640 × 480 pixels) with 11-bit depth, which provides 2,048 levels of sensitivity. The sensor has an angular field of view of 57 horizontally and 43 vertically. The Kinect sensor has a practical ranging limit of 1.2 - 3.5 m distance when used with the Xbox software. Figure 11 shows the device and some annotation indicating the cameras and projector.



Figure 29 - Kinect Sensor [19]

3.3.9 Digital Compass

The robot will have a digital compass to be able to get its heading angle. The strength of the earth's magnetic field is about 0.5 to 0.6 gauss and has a component parallel to the earth's surface that always points toward the magnetic north pole. In the northern hemisphere, this field points down. At the equator, it points horizontally and in the southern hemisphere, it points up. This angle between the

earth's magnetic field and the horizontal plane is defined as an inclination angle. Another angle between the earth's magnetic north and geographic north is defined as a declination angle in the range of $\pm 20^\circ$ depending on the geographic location.

A tilt compensated electronic compass system requires a 3-axis magnetic sensor and a 3-axis accelerometer sensor. The accelerometer is used to measure the tilt angles of pitch and roll for tilt compensation. And the magnetic sensor is used to measure the earth's magnetic field and then to determine the heading angle with respect to the magnetic north (Figure 30). If the heading with respect to the geographic north is required, the declination angle at the current geographic location should be compensated to the magnetic heading.



Figure 30 - Roll, Pitch and Heading angles [21]

We will be using heading angle mostly in our robot platform.

When the device is at a leveled position, pitch and roll angles are 0° . Then the heading angle can be determined as shown in Figure 31.

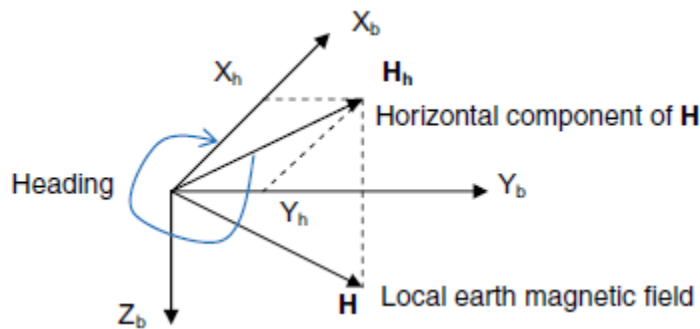


Figure 31 - Heading angle Calculation [21]

Local earth magnetic field H has a fixed component H_h on the horizontal plane pointing to the earth's magnetic north. This component can be measured by the magnetic sensor sensing axes X_M and Y_M that are named as X_h and Y_h . Then the heading angle is calculated as:

$$\text{Heading} = \arctan(Y_h/X_h)$$

In Figure 31, when the device body X_b axis is parallel to H_h which is pointing to the magnetic north, then $X_h = \max$ and $Y_h = 0$ so that heading = 0° . Rotating the device clockwise on the horizontal plane, the heading increases. When $X_h = 0$ and $Y_h = \min$, then heading = 90° . Keep rotating until $X_h = \min$ and $Y_h = 0$, then heading = 180° . And so on. After a full round 360° rotation, the user sees a centered circle if plotting X_h and Y_h values coming from the magnetic sensor measurements.

If the handheld device is tilted, then the pitch and roll angles are not equal to 0° as shown in Figure 32, where the pitch and roll can be measured by a 3-axis accelerometer. Therefore, the magnetic

sensor measurements X_M , Y_M and Z_M need to be compensated to obtain X_h and Y_h as shown in Equation 2. And then apply Equation 1 for the heading calculation.

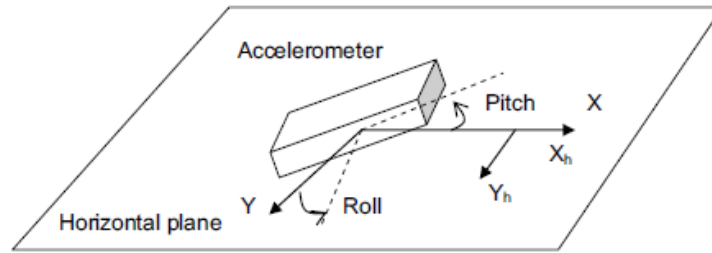


Figure 32 - Tilted Position [21]

$$X_h = X_M \cos(Pitch) + Z_M \sin(Pitch)$$

$$Y_h = X_M \sin(Roll) \sin(Pitch) + Y_M \cos(Roll) - Z_M \sin(Roll) \cos(Pitch)$$

Where X_M , Y_M and Z_M are magnetic sensor measurements.

We will use LSM303 Breakout Board - Tilt Compensated Compass (Figure 33) for this purpose. This board is cost effective compared to other products and has tilt compensation on it.



Figure 33 - LSM303 Tilt Compensated Compass [22]

Figure 34 below shows the block diagram of an electronic compass system. Arduino is used to collect the 3-axis accelerometer raw data for the pitch and roll calculation and collect the 3-axis magnetic sensor raw data for the heading calculation. The following is the procedure for building a working electronic compass system.

- Hardware design to make sure the MCU can get clean raw data from the accelerometer and the magnetic sensor
- Accelerometer calibration to obtain parameters to convert accelerometer raw data to normalized values for pitch and roll calculation
- Magnetic sensor calibration to obtain parameters to convert magnetic sensor raw data to normalized values for the heading calculation
- Test the performance of the electronic compass system.

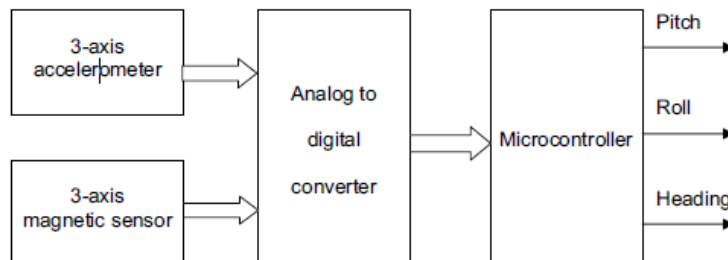
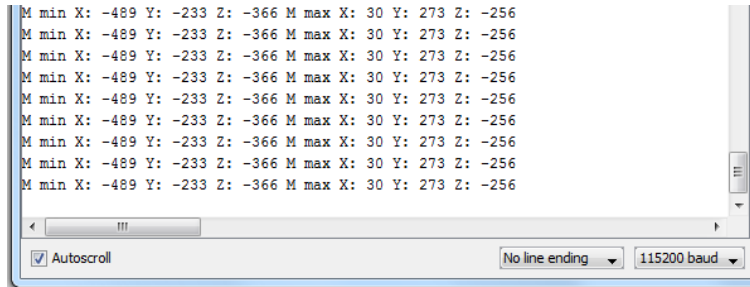


Figure 34 – Block diagram of electronic compass system [21]

Therefore, after implementing this compass to the system we will have a heading angle of our robot platform.

The calibration is run on the Arduino and the results for the calibration are given below. The calibration is done using the open source calibration library of the LSM303 Compass.



```
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
M min X: -489 Y: -233 Z: -366 M max X: 30 Y: 273 Z: -256
```

Figure 35 - LSM303 Calibration Process Screenshot

The calibration process prints a running minimum and maximum of the readings from each magnetometer axis. The final values for the minimum and maximum values are obtained and will be used in calculating the heading after moving the LSM303 through every possible orientation.

With all these sensors and hardware, most of the robotic research algorithms can be run on the developed robot platform. The next section will describe the control system, and communication protocols of the robot, and how to start developing an algorithm.

CHAPTER 4

ROBOT PLATFORM CONFIGURATION AND DESIGN PARAMETERS

In chapter 3, the aluminum sigma profiles are chosen to build the robot platform chassis. It was stated that the sigma profiles are modular, easy to assemble and the production time of the profiles are very little compared to other methods. In this chapter some possible configuration of the robot platform will be shown using this sigma profiles.

The simplest robot configuration is a two wheeled differential drive platform, which is illustrated in Figure 36. The stability of the robot platform can be maintained by using internal balancing systems or simply mounting caster wheel to the front and back of the robot platform.

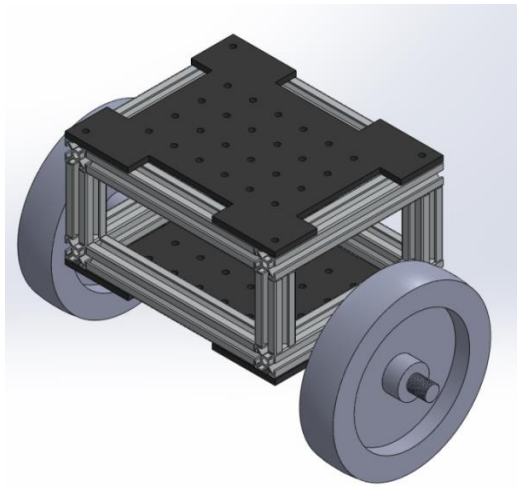


Figure 36 - 2 wheeled configuration

Another configuration for the robot is 4 wheeled which will be the prototype we will be producing can be seen in Figure 37.

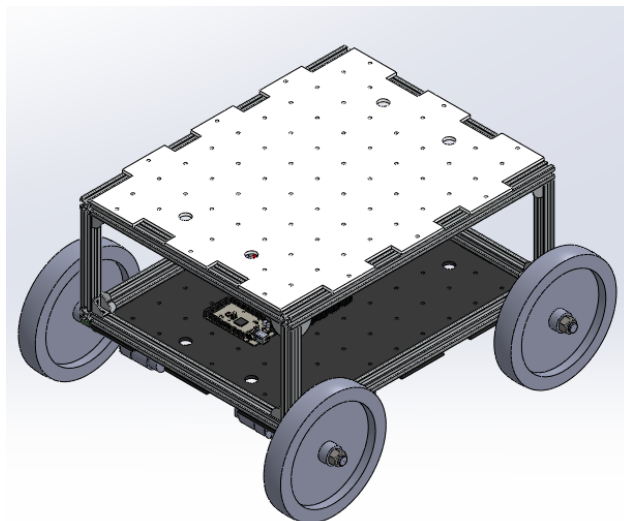


Figure 37 - 4 wheeled configuration

Another configuration for the robot is 6 wheeled (Figure 38). This and any other configuration is made only by defining the length, width and height of the robot and producing the chassis according to these values. It is a very easy process because the sigma profiles are produced only by giving the lengths. After that the assembly is done by just connecting the profiles with special connection parts. Then the chassis is ready. After this the number of motor block should be decided and they also connected to the chassis by 2 screws. Then the configuration is ready.

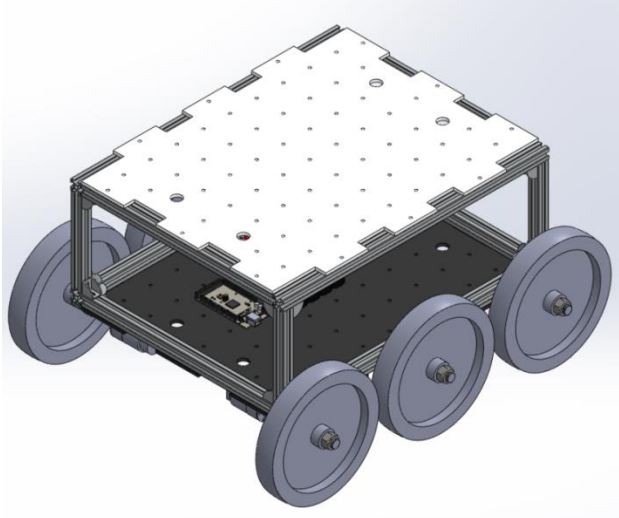


Figure 38 - 6 wheeled configuration

The dimensions and the motor number of the robot platform can be selected like mentioned above. The sensors of the platform should also be selected to the specific needs. First of the brain of the robot should be selected. A mainboard can be selected to be the processing unit of the robot as well as a laptop can also be selected. A configuration with mainboard and a laser scanner is given in Figure 39.

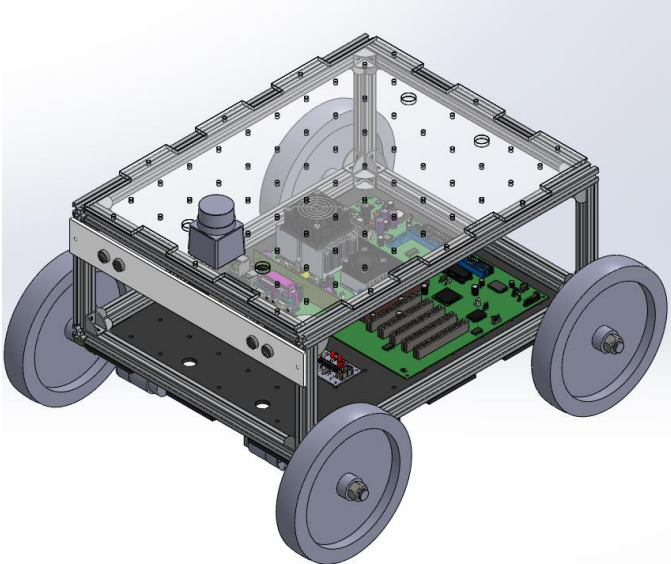


Figure 39 - A mainboard configuration with a laser and ultrasonic sensor

Another alternative for the sensors, which we will be constructing for demo prototype, is given in Figure 40.

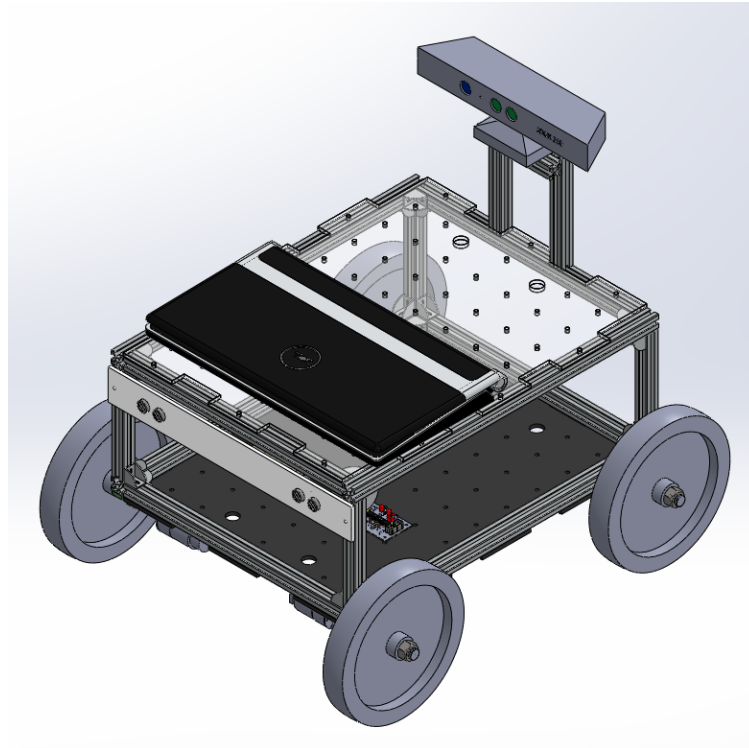


Figure 40 - A notebook configuration with Kinect and ultrasonic sensors

The top and bottom layer of the robot is produced by using Plexiglas. The reason for that is firstly, its production is easy. Secondly, after the production the manipulation of the Plexiglas is easy, if you need additional mounting holes, it is very easy to operate on it. Thirdly, it is not conductive so there is no risk of short circuits at the bottom part. Lastly, it will give a nice looking because it is transparent.

The ultrasonic sensor for both configurations will be used for blind spots where Kinect cannot get readings.

As it can be seen from previous configurations a robot platform for specific needs can be constructed very easily. A toolbox for the sigma profiles can be prepared for a lab and students can construct their robots using this toolbox. The toolbox will be their standard components and everyone can construct different dimension robots and can use different type of sensor for their research.

CHAPTER 5

CONTROL ARCHITECTURE DESIGN

In the scope of this thesis a simple controller system for the lower part of the robot platform is implemented in order to perform low level actions like move forward, backward or turn.

The control architecture mainly consists of 2 microcontroller cards namely Arduino Mega and motor controller card (Rover 5 Motor Controller Card). The motor controller card can drive up to 4 motor simultaneously. The schematic of the microcontroller card and motor controller is given in Figure 41. One Arduino will be used to control the motor and other sensor implementation while one Arduino will be used just for reading the ultrasonic sensors. This is because of that the robot platform has 10 ultrasonic sensors. If there is no obstacle in front of a sensor (this means that the sensor will read a maximum distance) the reading will approximately take 30 milliseconds to be performed. Therefore 10 ultrasonic sensor means a maximum of 300 ms in reading. The test for just using one microcontroller for all of the work done in low level will not be suitable due to the explained reasons. The other microcontroller card takes the encoder inputs of the motors into the system and drives them by giving PWM to the motor controller card. The motor controller card also takes direction input and gives current output of the motors. This way the currents of the motors are monitored and the power consumption can be calculated as well as current limitations can be done.

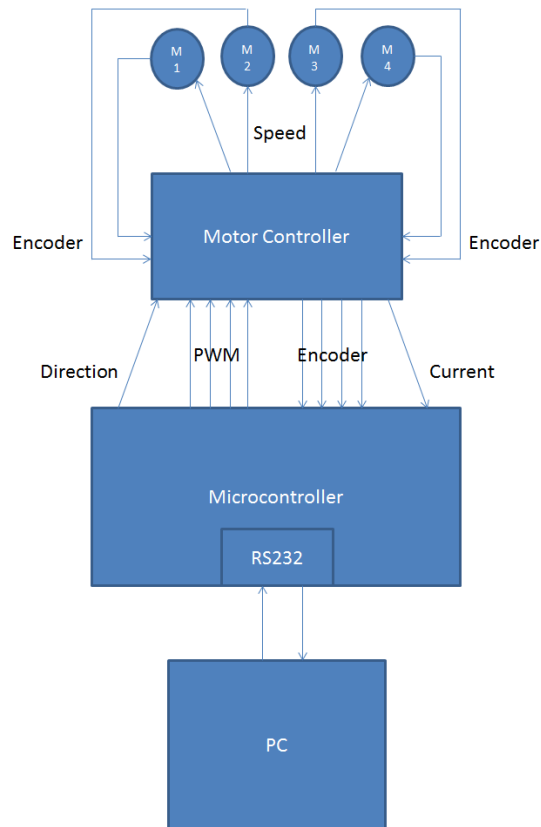


Figure 41 - Motor Controller and Microcontroller Schematic

The information exchange with the computer is accomplished through RS232 protocol. The microcontroller card can be connected to the computer via USB. Since we plan to use laptop on the robot platform for our test and demos, we will use this connection. However, Arduino boards can easily be adapted to Xbee Pro modules which can manage wireless transmission. These modules can also be used if it is necessary. Therefore the microcontroller unit is the low level controller for the

robot platform. It waits commands from a high level controller. A functional block diagram is also shown in Figure 42.

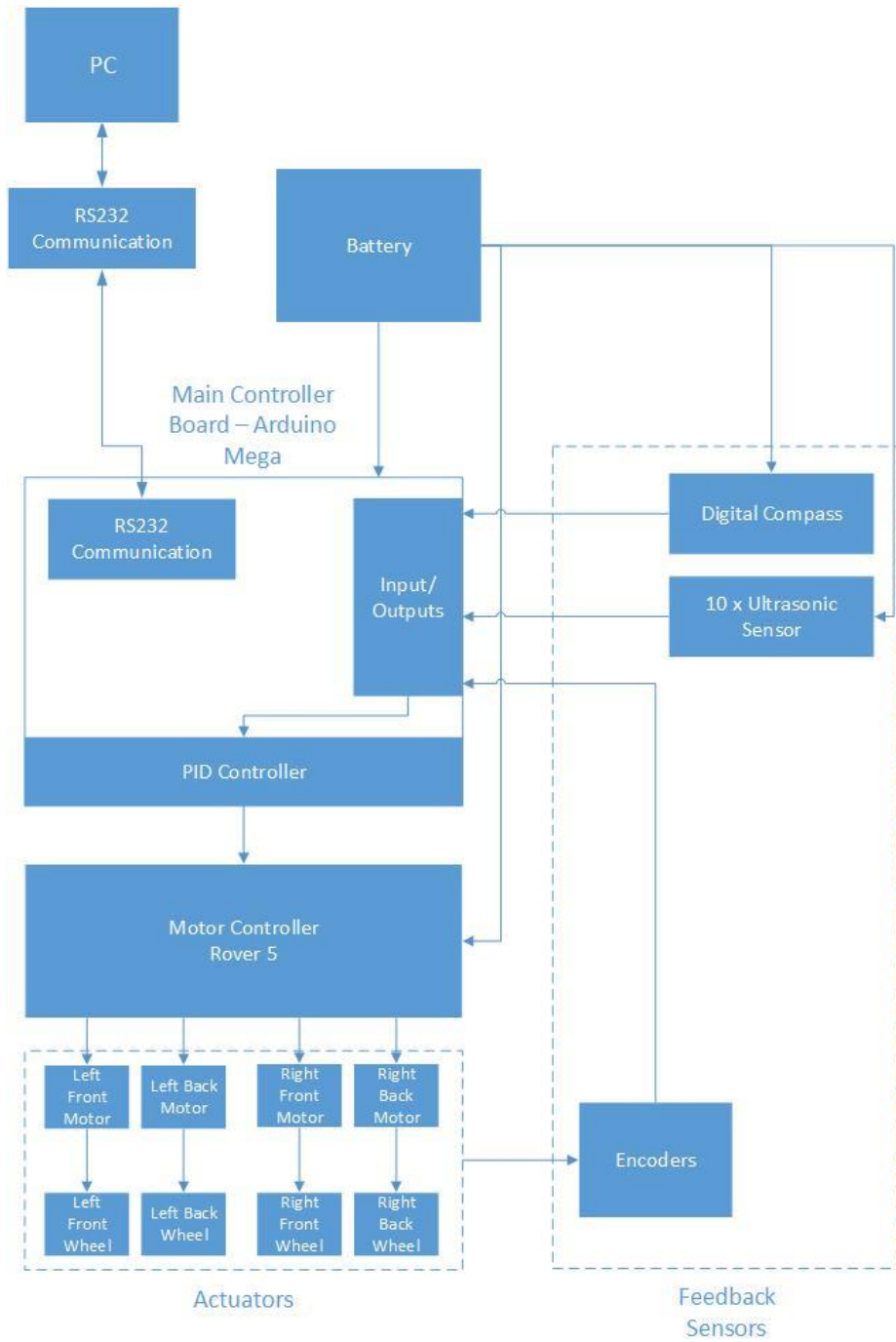


Figure 42 - Functional Block Diagram

A serial protocol is designed and implemented. In this part, this protocol is explained in detail.

The serial RS232 communication protocol was developed for transmitting and receiving data over the RS232 serial port of the Arduino microcontroller. The protocol is defined for a point-to-point communication based on the EIA-RS232 standard.

The protocol can be used to implement the command set defined for the Arduino. For a high degree of reliability in an electrically noisy environment it is designed with a checksum.

Before going into detail in the serial protocol, the heartbeat and broadcasting features of the robot is explained.

The heartbeat is a signal for the robot to continue its operations. The robot waits for the heartbeat signal to start to operate. If no heartbeat signal is received, the robot will not move. So, to use the robot a heartbeat signal should send to the robot in an interval. The interval can also be customizable. While receiving the heartbeat, the robot will operate as the user wants. If the communication is lost between the robot and PC the heartbeat signal will also be lost. Therefore, when the robot did not receive a heartbeat signal, it will immediately stop its operation and wait for a new heartbeat signal.

The broadcasting features of the robot includes all the sensors on the robot. When a user wants a sensor data from the robot, the broadcasting mode of the related sensor should be first activated. From the PC side, the user should specify which sensor to be broadcasted and the user should also set the interval of the update. Therefore, the user will only receive the sensor updates that he/she wanted.

The detailed use and explanation of the functions are explained in the next parts.

5.1 Data Format

Data is transmitted in an asynchronous way, that means each data packet is transmitted individually with its own start and stop bit.

First the number of commands and their data lengths is defined. We define the commands for use with the sensors we have, however we developed the data packet format as it can be used with any other sensor.

The commands and data lengths of the pc side is created and given in Table 4, they will explained in detail in the later of this chapter:

Table 4 - PC Side Commands

COMMAND NAME	DATA LENGTH
SET DIRECTION	4 bytes
GET DIRECTION	-
SET PWM	4 bytes
GET PWM	-
GET ENCODER	-
GET CURRENT	-
GET SONAR DATA	-
EMERGENCY STOP	-
RESET ENCODER	-
SET RPM	4 bytes
GET RPM	-
GET HEADING	-
GET ACCELEROMETER	-
HEARTBEAT	-
MOTOR ENCODER UPDATE	2 bytes
MOTOR RPM UPDATE	2 bytes
MOTOR CURRENT UPDATE	2 bytes
SONAR UPDATE	2 bytes
HEADING UPDATE	2 bytes
ACCELEROMETER UPDATE	2 bytes

The commands and data lengths of the microcontroller side are created and given in Table 5:

Table 5 - Microcontroller Side Commands

COMMAND NAME	DATA LENGTH
GET DIRECTION	4 bytes
GET PWM	4 bytes
GET ENCODER	16 bytes
GET CURRENT	8 bytes
GET SONAR DATA	8 bytes
GET RPM	4 bytes
GET HEADING	4 bytes
GET ACCELEROMETER	12 bytes
HEARTBEAT	-
MOTOR ENCODER UPDATE	-
MOTOR RPM UPDATE	-
MOTOR CURRENT UPDATE	-
SONAR UPDATE	-
HEADING UPDATE	-
ACCELEROMETER UPDATE	-

The data packet will include a start of text byte, a command byte, data bytes, a block check character byte and an end of text bytes. With all these concatenated the data packet is formed.

A command packet for the pc side is defined and given in Table 6:

Table 6 - Data Packet from PC

1 byte	1 byte	1 byte	n bytes	1 byte	1 byte
STX	CMD	DATA LENGTH	DATA	BCC	ETX

and for microcontroller side it is given in Table 7:

Table 7 - Data Packet from Microcontroller

1 byte	1 byte	1 byte	n bytes	1 byte	1 byte
STX	CMD	DATA LENGTH	DATA	BCC	ETX

Where:

STX: Start of Text Byte

CMD: A unique command byte assigned to commands

DATA LENGTH: The length of the data in bytes

DATA: The data of the the packet

BCC: Block Check Character of the data

ETX: End of Text Byte

The STX byte of the data packet will be constant in all packets and it will be 0x02.

The ETX byte of the data packet will also be constant in all packets and it will be 0x03. These two bytes will be defining the start and end of the data packet and will help the algorithm to find a data packet in a buffer. Therefore no packet will be escape.

The BCC is added to the data packet because by implementing this we will secure our packet from electrical noises, and if the BCC is different we will discard that packet. The BCC function is both sides is the same and by starting the command byte, it subtracts the next byte and it iterates to the end of the data bytes of the packets. It does not use STX and ETX bytes of the data packet because they are constants in all packets. By this method, it calculates the BCC byte and both sides use the same function and check when a packet arrives.

By implementing a robust serial protocol the data packets are received and transmitted very efficiently. The algorithm will be explained later in this chapter.

At the PC side a class for controlling the robot is created by using visual studio and high level control functions that runs the commands for the microcontroller is implemented. The corresponding function and its example use will be explained in the specified command detail below.

There is no front or back in this robot platform since it is symmetrical, however, for the sake of simplicity in the protocol, the side where the electronic cards (microcontroller and motor controller boards) are located is considered as the front side of the robot platform.

5.1.1 PC Side Commands

In this part the PC Side Commands of the protocol is described.

1. SET DIRECTION

The description and details of the command is given in Table 8.

Table 8 - Set Direction Command (PC Side)

Command Name	SET DIRECTION
Description	This command sets the direction of the individual motors in the robot platform
Command Byte	0x41
Data	The data packet of this command is 4 bytes. Each byte will set the direction of individual motors in the robot platform. The order of the bytes corresponding to motors in our configuration are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

An example data packet for this command is (which gives forward direction for all motors) given in Table 9:

Table 9 - Example Set Direction Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x41	0x04	0x49	0x49	0x49	0x49	0x41	0x03

This command will return GET DIRECTION command for the microcontroller side automatically. This will ensure that the data packet is received at the microcontroller side and immediately returns the set values. Therefore, this feature is just like a handshake.

The function and example use in the pc side for this command is given below.

```
SetDirection(int directionLF, int directionLB, int directionRF, int directionRB);
```

Two constants are defined at the top of the class for directions. These are;

Forward Direction:

```
MOTOR_DIRECTION_FRONT = 'T';
```

Backward Direction:

```
MOTOR_DIRECTION_BACK = 'G';
```

Therefore an example command for forward direction of all motors will be:

```
SetDirection(MOTOR_DIRECTION_FRONT, MOTOR_DIRECTION_FRONT,
MOTOR_DIRECTION_FRONT, MOTOR_DIRECTION_FRONT);
```

2. GET DIRECTION

The description and details of the command is given in Table 10.

Table 10 - Get Direction Command (PC Side)

Command Name	GET DIRECTION
Description	This command gets the current direction of the individual motors in the robot platform
Command Byte	0x42
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 11:

Table 11 - Example Get Direction Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x42	0x00	0x42	0x03

The function and example use in the pc side for this command is given below.

```
GetDirection();
```

The return for this function is the current directions of the motor. The packet of the return will be explained in the microcontroller command section.

3. SET PWM

The description and details of the command is given in Table 12

Table 12 - Set PWM Command (PC Side)

Command Name	SET PWM
Description	This command sets the pwm of the individual motors in the robot platform

Command Byte	0x43
Data	The data packet of this command is 4 bytes. Each byte will set the pwm of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

An example data packet for this command is (which gives 100 pwm value for all motors) is given in Table 13:

Table 13 - Example Set PWM Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x43	0x04	0x64	0x64	0x64	0x64	0x43	0x03

This command will return GET PWM command for the microcontroller side automatically. This will ensure that the data packet is received at the microcontroller side and immediately returns the set values. Therefore, this feature is just like a handshake.

The function and example use in the pc side for this command is given below.

SetPWM(int pwmMotorLF, int pwmMotorLB, int pwmMotorRF, int pwmMotorRB)

Therefore an example command for forward direction of all motors will be:

SetPWM(100, 100, 100, 100);

4. GET PWM

The description and details of the command is given in Table 14

Table 14 - Get PWM Command (PC Side)

Command Name	GET PWM
Description	This command gets the current pwm of the individual motors in the robot platform
Command Byte	0x44
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 15:

Table 15 - Example Get PWM Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x44	0x00	0x44	0x03

The function and example use in the pc side for this command is given below.

GetPWM();

The return for this function is the current pwms of the motors. The packet of the return will be explained in the microcontroller command section.

5. GET ENCODER

The description and details of the command is given in Table 16

Table 16 - Get Encoder Command (PC Side)

Command Name	GET ENCODER
Description	This command gets the current encoder ticks of the individual motors in the robot platform
Command Byte	0x46
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 17:

Table 17 - Example Get Encoder Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x46	0x00	0x46	0x03

The function and example use in the pc side for this command is given below.

GetEncoder();

The return for this function is the current encoder ticks of the motors. The packet of the return will be explained in the microcontroller command section.

6. GET CURRENT

The description and details of the command is given in Table 18.

Table 18 - Get Current Command (PC Side)

Command Name	GET CURRENT
Description	This command gets the current currents of the individual motors in the robot platform
Command Byte	0x47
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 19:

Table 19 - Example Get Current Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x47	0x00	0x47	0x03

The function and example use in the pc side for this command is given below.

GetCurrent();

The return for this function is the current currents of the motors. The packet of the return will be explained in the microcontroller command section.

7. GET SONAR DATA

The description and details of the command is given in Table 20.

Table 20 - Get Sonar Data Command (PC Side)

Command Name	GET SONAR DATA
Description	This command gets the current sonar data of the sonar sensors in the robot platform
Command Byte	0x45
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 21:

Table 21 - Example Get Sonar Data Packet

STX	CMD	DL	BCC	ETX
0x02	0x45	0x00	0x45	0x03

The function and example use in the pc side for this command is given below.

```
GetSonarData();
```

The return for this function is the current sonar data of the sonar sensors. The packet of the return will be explained in the microcontroller command section.

8. SET EMERGENCY STOP

The description and details of the command is given in Table 22.

Table 22 – Set Emergency Stop Command (PC Side)

Command Name	SET EMERGENCY STOP
Description	This command is used as the emergency stop. The microcontroller does not evaluate any data with this command. When it receives this command, the motors are shut down immediately.
Command Byte	0x48
Data	No data packet for the command
Packet Size	5 bytes

An example data packet for this command is given in Table 23:

Table 23 - Example Set Emergency Stop Data Packet

STX	CMD	DL	BCC	ETX
0x02	0x48	0x00	0x48	0x03

The function and example use in the pc side for this command is given below.

```
SetEmergencyStop();
```

The return for this function is the pwm values of the motors (GET PWM Command). The packet of the return will be explained in the microcontroller command section.

9. RESET ENCODER

The description and details of the command is given in Table 24.

Table 24 – Reset Encoder Command (PC Side)

Command Name	SET EMERGENCY STOP
Description	This command resets the encoder ticks in the microcontroller.
Command Byte	0x49
Data	No data packet for the command
Packet Size	5 bytes

An example data packet for this command is given in Table 25:

Table 25 - Example Reset Encoder Data Packet

STX	CMD	DL	BCC	ETX
0x02	0x49	0x00	0x49	0x03

The function and example use in the pc side for this command is given below.

```
ResetEncoder();
```

The return for this function is the encoder values of the motors (GET ENCODER Command). The packet of the return will be explained in the microcontroller command section.

10. SET RPM

The description and details of the command is given in Table 26.

Table 26 - Set RPM Command (PC Side)

Command Name	SET RPM
Description	This command sets the rpm of the individual motors in the robot platform
Command Byte	0x51
Data	The data packet of this command is 4 bytes. Each byte will set the rpm of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

An example data packet for this command is (which gives 100 rpm value for all motors) given in Table 27:

Table 27 - Example Set RPM Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x50	0x04	0x64	0x64	0x64	0x64	0x50	0x03

This command will return GET RPM command for the microcontroller side automatically. This will ensure that the data packet is received at the microcontroller side and immediately returns the set values. Therefore, this feature is just like a handshake.

The function and example use in the pc side for this command is given below.

SetRPM(int rpmMotorLF, int rpmMotorLB, int rpmMotorRF, int rpmMotorRB)

Therefore an example command for forward direction of all motors will be:

SetRPM(100, 100, 100, 100);

11. GET RPM

The description and details of the command is given in Table 28.

Table 28 - Get RPM Command (PC Side)

Command Name	GET RPM
Description	This command gets the current rpm of the individual motors in the robot platform
Command Byte	0x51
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 29:

Table 29 - Example Get RPM Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x51	0x00	0x51	0x03

The function and example use in the pc side for this command is given below.

GetRPM();

The return for this function is the current rpms of the motors. The packet of the return will be explained in the microcontroller command section.

12. GET HEADING

The description and details of the command is given in Table 30.

Table 30 - Get Heading Command (PC Side)

Command Name	GET HEADING
Description	This command gets the current heading of the robot platform
Command Byte	0x52
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 31:

Table 31 - Example Get Accelerometer Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x52	0x00	0x52	0x03

The function and example use in the pc side for this command is given below.

GetHeading();

The return for this function is the current heading of the robot platform. The packet of the return will be explained in the microcontroller command section.

13. GET ACCELEROMETER

The description and details of the command is given in Table 32.

Table 32 - Get Accelerometer Command (PC Side)

Command Name	GET ACCELEROMETER
Description	This command gets the current x,y,z accelerometer data of the robot platform
Command Byte	0x53
Data	No data packet for the get commands.
Packet Size	5 bytes

An example data packet for this command is given in Table 33:

Table 33 - Example Get Accelerometer Data Packet (PC Side)

STX	CMD	DL	BCC	ETX
0x02	0x53	0x00	0x53	0x03

The function and example use in the pc side for this command is given below.

GetAccelerometer();

The return for this function is the current x,y,z accelerometer data of the robot platform. The packet of the return will be explained in the microcontroller command section.

14. HEARTBEAT

The description and details of the command is given in Table 34.

Table 34 – Heartbeat Command (PC Side)

Command Name	HEARTBEAT
Description	This command sends a heartbeat to the microcontroller
Command Byte	0x54
Data	No data packet for the command
Packet Size	5 bytes

An example data packet for this command is given in Table 35:

Table 35 - Example Heartbeat Data Packet

STX	CMD	DL	BCC	ETX
0x02	0x54	0x00	0x54	0x03

This command is connected to a timer in the PC side. It is configured before the connection to the microcontroller by setting the interval time of the timer. After that in every timer tick the heartbeat command is sent to the microcontroller. This is for not losing the control over the robot if the PC is

crashes or the serial communication is lost. If it happens the PC will not send a heartbeat in the timer therefore the microcontroller will understand that the communication is lost and it will shut down the motor. The microcontroller will begin to wait for a new heartbeat to continue operation.

The return for this function is also a heartbeat. This will ensure that the microcontroller is responding to the heartbeat. The packet of the return will be explained in the microcontroller command section.

15. MOTOR ENCODER UPDATE

The description and details of the command is given in Table 36.

Table 36 - Motor Encoder Update Command (PC Side)

Command Name	MOTOR ENCODER UPDATE
Description	This command sets if the user wants the motor encoder data to be broadcasted and the update interval for the motor encoder data.
Command Byte	0x55
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 37:

Table 37 - Example Motor Encoder Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x55	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.MotorEncoderUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the motor encoder parameters in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

16. MOTOR RPM UPDATE

The description and details of the command is given in Table 38.

Table 38 - Motor RPM Update Command (PC Side)

Command Name	MOTOR RPM UPDATE
Description	This command sets if the user wants the motor rpm data to be broadcasted and the update interval for the motor rpm data.
Command Byte	0x56
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 39:

Table 39 - Example Motor RPM Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x56	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.MotorRPMUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the motor rpm parameters in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

17. MOTOR CURRENT UPDATE

The description and details of the command is given in Table 40.

Table 40 - Motor Current Update Command (PC Side)

Command Name	MOTOR CURRENT UPDATE
Description	This command sets if the user wants the motor current data to be broadcasted and the update interval for the motor current data.
Command Byte	0x57
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 41:

Table 41 - Example Motor Current Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x57	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.MotorCurrentUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the motor current parameters in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

18. SONAR UPDATE

The description and details of the command is given in Table 42.

Table 42 - Sonar Update Command (PC Side)

Command Name	SONAR UPDATE
Description	This command sets if the user wants the sonar data to be broadcasted and the update interval for the sonar data.
Command Byte	0x58
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 43:

Table 43 - Example Sonar Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x58	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.SonarUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the sonar parameters in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

19. HEADING UPDATE

The description and details of the command is given in Table 44.

Table 44 - Heading Update Command (PC Side)

Command Name	HEADING UPDATE
Description	This command sets if the user wants the heading data to be broadcasted and the update interval for the heading data.
Command Byte	0x59
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 45:

Table 45 - Example Heading Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x59	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.HeadingUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the heading parameter in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

20. ACCELEROMETER UPDATE

The description and details of the command is given in Table 46.

Table 46 - Accelerometer Update Command (PC Side)

Command Name	ACCELEROMETER UPDATE
Description	This command sets if the user wants the accelerometer data to be broadcasted and the update interval for the accelerometer data.
Command Byte	0x60
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 47:

Table 47 - Example Accelerometer Update Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	BCC	ETX
0x02	0x60	0x03	0x01	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

```
SetUpdateParameterInterval(EduRobot.UpdateParameter.AccelerometerUpdate,
EduRobot.UpdateResponse.YES, 100);
```

This will make the robot to send the accelerometer parameters in every 100 ms.

The return for this function is a data packet with the same command but with no data. The packet of the return will be explained in the microcontroller command section.

21. SET HEARTBEAT INTERVAL

The description and details of the command is given in Table 48.

Table 48 – Set Heartbeat Interval Command (PC Side)

Command Name	ACCELEROMETER UPDATE
Description	This command sets the heartbeat interval of the microcontroller.
Command Byte	0x61
Data	1 st and 2 nd byte: The heartbeat interval in milliseconds
Packet Size	7 bytes

An example data packet for this command is given in Table 49:

Table 49 - Example – Set Heartbeat Interval Data Packet (PC Side)

STX	CMD	DL	DATA 1	DATA 2	BCC	ETX
0x02	0x61	0x03	0x00	0x64	0x3F	0x03

The function and example use in the pc side for this command is given below.

SetHeartBeat(2000)

This will set the heartbeat interval of the robot to 2000 ms.

The return for this function is a heartbeat. The packet of the return will be explained in the microcontroller command section.

5.1.2 Microcontroller Side Commands

In this part the Microcontroller Side Commands of the protocol is described.

1. GET DIRECTION

The description and details of the command is given in Table 50.

Table 50 - Get Direction Command (Microcontroller Side)

Command Name	GET DIRECTION
Description	This command sends the current direction of the individual motors in the robot platform to the pc
Command Byte	0x42
Data	The data packet of this command is 4 bytes. Each byte will send the direction of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

The direction information of the motor can be represented as 1 byte so the data is 4 bytes.

An example data packet for this command is given in Table 51:

Table 51 - Example Get Direction Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x42	0x04	0x49	0x49	0x49	0x49	0x42	0x03

The function and example use in the pc side for this command is given below.

```
GetDirection();
```

The return for this function is the current directions of the motors.

2. GET PWM

The description and details of the command is given in Table 52.

Table 52 - Get PWM Command (Microcontroller Side)

Command Name	GET PWM
Description	This command sends the current pwms of the individual motors in the robot platform to the pc
Command Byte	0x44
Data	The data packet of this command is 4 bytes. Each byte will send the pwm of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

The pwm information of the motor (0 to 255) can be represented as 1 byte so the data is 4 bytes.

An example data packet for this command is given in Table 53:

Table 53 - Example Get PWM Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x42	0x04	0x64	0x64	0x64	0x64	0x42	0x03

The function and example use in the pc side for this command is given below.

```
GetPWM();
```

The return for this function is the current pwms of the motors.

3. GET ENCODER

The description and details of the command is given in Table 54.

Table 54 - Get Encoder Command (Microcontroller Side)

Command Name	GET ENCODER
Description	This command sends the current encoder ticks of the individual motors in the robot platform to the pc
Command Byte	0x46
Data	The data packet of this command is 16 bytes. Each 4 byte will send the encoder ticks of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1-4 byte : Left Front Motor 5-8 byte: Left Back Motor 9-12 byte: Right Front Motor 13-16 byte: Right Back Motor
Packet Size	21 bytes

The encoder information of the motor is a long int number (32 bit) therefore 4 byte is needed to represent each encoder data. So, the data is 16 bytes.

An example data packet for this command is given in Table 55:

Table 55 - Example Get Encoder Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	DATA 8
0x02	0x46	0x15	0x00	0x04	0x0b	0x13	0x00	0x03	0x4f	0xf6
DATA 9	DATA 10		DATA 11	DATA 12	DATA 13	DATA 14	DATA 15	DATA 16	BCC	ETX
0x00	0x03		0xe0	0x52	0x00	0x03	0xcb	0x2a	0xb3	0x03

The corresponding encoder ticks for example:

Encoder LF: 0x00040b13 = 264979

Encoder LB: 0x00034ff6 = 217078

Encoder RF: 0x0003e052 = 254034

Encoder RB: 0x0003cb2a = 248618

The function and example use in the pc side for this command is given below.

```
GetEncoder();
```

The return for this function is the current encoder ticks of the motors.

4. GET CURRENT

The description and details of the command is given in Table 56.

Table 56 - Get Current Command (Microcontroller Side)

Command Name	GET CURRENT
Description	This command sends the current currents of the individual motors in the robot platform to the pc
Command Byte	0x47
Data	The data packet of this command is 8 bytes. Each 2 byte will send the currents of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1-2 byte : Left Front Motor 3-4 byte: Left Back Motor 5-6 byte: Right Front Motor 7-8 byte: Right Back Motor
Packet Size	13 bytes

The current information of the motor is an int number and can take values up to ~5000 therefore 2 byte is needed to represent each current data. So, the data is 8 bytes.

An example data packet for this command is given in Table 57:

Table 57 - Example Get Current Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7
0x02	0x47	0x08	0x00	0x93	0x00	0x96	0x00	0xaf	0x00
DATA 8	BCC	ETX							
0x94	0x79	0x03							

The corresponding currents for example:

Current LF: 147

Current LB: 150

Current RF: 175

Current RB: 148

The function and example use in the pc side for this command is given below.

```
GetCurrent();
```

The return for this function is the current currents of the motors.

5. GET SONAR DATA

The description and details of the command is given in Table 58.

Table 58 - Get Sonar Data Command (Microcontroller Side)

Command Name	GET SONAR DATA
Description	This command sends the current sonar data of the sonar sensors in the robot platform to the pc
Command Byte	0x45
Data	The data packet of this command is 8 bytes. Each 2 byte will send the sonar data of individual sonar sensors in the robot platform. The order of the bytes corresponding to sensors are: 1-2 byte : Front Left Sensor 3-4 byte: Front Right Sensor 5-6 byte: Back Left Sensor 7-8 byte: Back Right Sensor
Packet Size	13 bytes

The sonar data information of the sonar sensors is an int number and can take values up to ~500 therefore 2 byte is needed to represent each sonar data. So, the data is 8 bytes.

An example data packet for this command is given in Table 59:

Table 59 - Get Sonar Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7
0x02	0x45	0x08	0x00	0x16	0x00	0x18	0x00	0x87	0x00
DATA 8	BCC	ETX							
0x00	0xcc	0x03							

The corresponding sonar data for example:

Sonar Data FL: 22

Sonar Data FR: 24

Sonar Data BL: 135

Sonar Data BR: 0

The function and example use in the pc side for this command is given below.

```
GetSonarData();
```

The return for this function is the current sonar data of the sonar sensors.

6. GET RPM

The description and details of the command is given in Table 60.

Table 60 - Get RPM Command (Microcontroller Side)

Command Name	GET PWM
Description	This command sends the current rpms of the individual motors in the robot platform to the pc
Command Byte	0x51
Data	The data packet of this command is 4 bytes. Each byte will send the rpm of individual motors in the robot platform. The order of the bytes corresponding to motors are: 1st byte : Left Front Motor 2nd byte: Left Back Motor 3rd byte: Right Front Motor 4th byte: Right Back Motor
Packet Size	9 bytes

The pwm information of the motor (0 to 255) can be represented as 1 byte so the data is 4 bytes.

An example data packet for this command is given in Table 61:

Table 61 - Example Get RPM Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x51	0x04	0x64	0x64	0x64	0x64	0x51	0x03

The function and example use in the pc side for this command is given below.

```
GetRPM();
```

The return for this function is the current rpms of the motors.

7. GET HEADING

The description and details of the command is given in Table 62.

Table 62 - Get Heading Command (Microcontroller Side)

Command Name	GET HEADING
Description	This command sends the current heading of the robot platform to the pc
Command Byte	0x52
Data	The data packet of this command is 4 bytes since the heading of the robot is a float number and can be represented by 4 bytes in the microcontroller side.
Packet Size	9 bytes

An example data packet for this command is given in Table 63:

Table 63 - Example Get Heading Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	BCC	ETX
0x02	0x52	0x04	0x64	0x64	0x64	0x64	0x52	0x03

The function and example use in the pc side for this command is given below.

GetHeading();

The return for this function is the current heading of the robot platform.

8. GET ACCELEROMETER

The description and details of the command is given in Table 64.

Table 64 - Get Accelerometer Command (Microcontroller Side)

Command Name	GET ACCELEROMETER
Description	This command sends the current x,y,z accelerometer data of the robot platform to the pc.
Command Byte	0x53
Data	The data packet of this command is 12 bytes since the each accelerometer data of the robot is a float number and can be represented as 4 bytes in the microcontroller side.
Packet Size	17 bytes

An example data packet for this command is given in Table 65:

Table 65 - Example Get Accelerometer Data Packet (Microcontroller Side)

STX	CMD	DL	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7
0x02	0x46	0x0c	0x00	0x04	0x0b	0x13	0x00	0x03	0x4f
DATA 8	DATA 9	DATA 10	DATA 11	BCC	ETX				
0x00	0x03	0xe0	0x52	0xb3	0x03				

The function and example use in the pc side for this command is given below.

GetAccelerometer();

The return for this function is the current x,y,z accelerometer data of the robot platform.

9. HEARTBEAT

The description and details of the command is given in Table 66.

Table 66 – Heartbeat Command (Microcontroller Side)

Command Name	HEARTBEAT
Description	This command sends a heartbeat to the PC
Command Byte	0x54
Data	No data packet for the command
Packet Size	5 bytes

An example data packet for this command is given in Table 67:

Table 67 - Example Heartbeat Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x54	0x00	0x54	0x03

This packet is created when a heartbeat from the PC is received as explained in the PC side.

10. MOTOR ENCODER UPDATE

The description and details of the command is given in Table 68.

Table 68 - Motor Encoder Update Command (Microcontroller Side)

Command Name	MOTOR ENCODER UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x55
Data	-
Packet Size	5 bytes

An example data packet for this command is given in Table 69:

Table 69 - Example Motor Encoder Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x55	0x00	0x55	0x03

11. MOTOR RPM UPDATE

The description and details of the command is given in Table 70.

Table 70 - Motor RPM Update Command (Microcontroller Side)

Command Name	MOTOR RPM UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x56
Data	-
Packet Size	5bytes

An example data packet for this command is given in Table 71:

Table 71 - Example Motor RPM Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x56	0x00	0x56	0x03

12. MOTOR CURRENT UPDATE

The description and details of the command is given in Table 72.

Table 72 - Motor Current Update Command (Microcontroller Side)

Command Name	MOTOR CURRENT UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x57
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 73:

Table 73 - Example Motor Current Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x57	0x00	0x57	0x03

13. SONAR UPDATE

The description and details of the command is given in Table 74.

Table 74 - Sonar Update Command (Microcontroller Side)

Command Name	SONAR UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x58
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 75:

Table 75 - Example Sonar Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x58	0x00	0x58	0x03

14. HEADING UPDATE

The description and details of the command is given in Table 76.

Table 76 - Heading Update Command (Microcontroller Side)

Command Name	HEADING UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x59
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 77:

Table 77 - Example Heading Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x59	0x00	0x59	0x03

15. ACCELEROMETER UPDATE

The description and details of the command is given in Table 78.

Table 78 - Accelerometer Update Command (Microcontroller Side)

Command Name	ACCELEROMETER UPDATE
Description	This command sends a data packet to inform the PC that the microcontroller received the update information.
Command Byte	0x60
Data	1 st byte: Sets a Boolean value for whether or not to send the parameters 2 nd and 3 rd byte: The update interval in milliseconds
Packet Size	8 bytes

An example data packet for this command is given in Table 79:

Table 79 - Example Accelerometer Update Data Packet (Microcontroller Side)

STX	CMD	DL	BCC	ETX
0x02	0x60	0x00	0x60	0x03

An example motor set point data flow schematic is given in Figure 43.

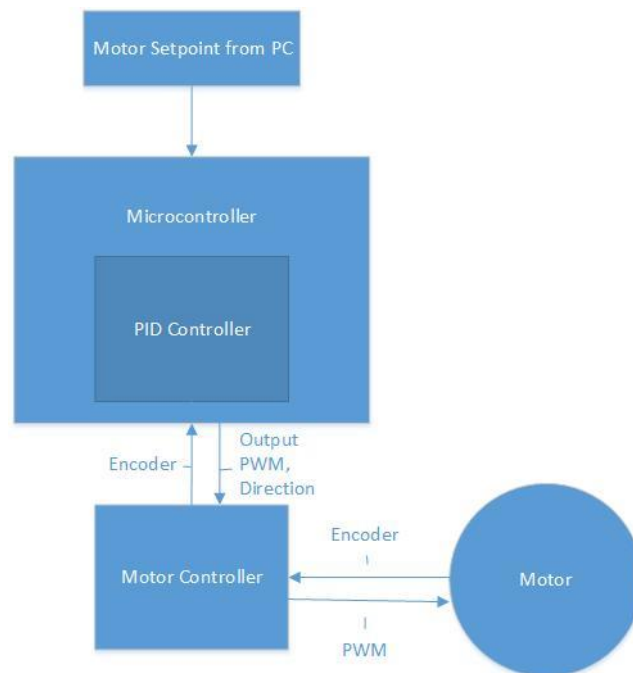


Figure 43 - Motor Setpoint Data Flow

The commands for both sides can be added if there is a need for new one. It is just needed to satisfy the data packet rules, using STX, CMD, DATA LENGTH, DATA, BCC, ETX bytes. And you should only implement the software what to do with the new packet.

5.2 The Serial Packet Handling Algorithm

In both PC and microcontroller the same algorithm runs for the serial packet handling. This is a robust serial handling. This algorithm ensures that every packet is evaluated. The code for the algorithm is given below.

After the packet is constructed byte by byte, it is sent to another method that evaluates this packet. It looks for the command byte for which command this packet is sent from and extracts the data from the packet. After that, the data is ready to use.

The codes for the serial packet handling algorithm can be found at Appendix A. In Appendix A the serial protocol and the algorithm is explained via the comment lines. Also, below the library functions for driving the robot platform is explained.

The list of the function in the library which is explained in the previous part is given in Table 80.

Table 80 - Low Level Functions

FUNCTION NAME
SetDirection(int directionLF, int directionLB, int directionRF, int directionRB);
GetDirection();
SetPWM(int pwmMotorLF, int pwmMotorLB, int pwmMotorRF, int pwmMotorRB);
GetPWM();
GetEncoder();
GetCurrent();
GetSonarData();
SetEmergencyStop();
ResetEncoder();
SetRPM(int rpmMotorLF, int rpmMotorLB, int rpmMotorRF, int rpmMotorRB);

5.3 High Level Functions

These are the primitive function to drive the robot platform. However, we implemented some high level function to drive the robot. This is for simplifying the use of the robot. One of the functions is for driving the robot with a linear velocity. This method can take different parameters like rpm, rps or m/s. Another method is giving a robot an angular velocity. Since the robot is a differential driven robot, this method is used for turning the robot for a specific heading. This method also can take degrees to turn. Another method is driving the robot with some distance value. These functions will use the same commands explained above, however they will have some algorithms in their function blocks and translate the high level functions to low level functions.

A robust motor controller algorithm is implemented to the microcontroller side. This algorithm takes the set values of the motors (rpms) and controls the motor automatically inside the microcontroller. There is no need to track this in the pc side of the software.

In this part we divide the motors into 2 sides, Left side and Right Side. Therefore we can have different speed for different sides.

5.3.1 Set Linear Speed

The linear speed of the robot can be set by using this method. The function gets the direction of the motion and the speed for this motion. The motor rpm values are calculated as given below;

$$RPM = \frac{V}{\pi \times D} \times 60$$

where

RPM : The revolution per minute value for motors

V : The linear velocity

D : Diameter of the wheels

By using this equation the motor rpms are calculated and the command for setting direction and rpm values for left and right sides are given to microcontroller.

5.3.2 Turn Angle (Point Turn)

The angular velocity of the robot can be set by using this method. The function gets the direction of the turn and the angle to be turned.

However, in order to understand the algorithm first we should look at the kinematics of the differential drive. While we can vary the velocity of each wheel, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC - Instantaneous Center of Curvature (see Figure 44).

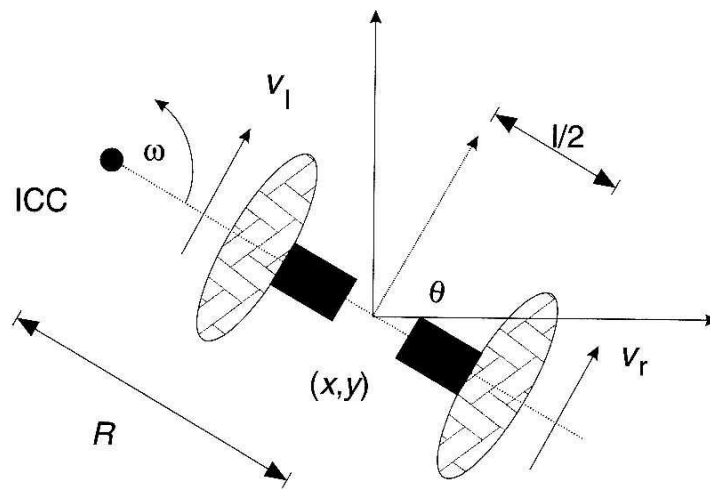


Figure 44 - Differential Drive kinematics [23]

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. Because the rate of rotation ω about the ICC must be the same for both wheels, we can write the following equations:

$$\omega(R + l/2) = V_r \quad (1)$$

$$\omega(R - l/2) = V_l \quad (2)$$

where l is the distance between the centers of the two wheels, V_r and V_l are the right and left wheel velocities along the ground, and R is the signed distance from the ICC to the midpoint between the wheels. At any instance in time we can solve for R and ω :

$$R = \frac{l(V_l + V_r)}{2(V_r - V_l)} \quad \text{and} \quad \omega = \frac{V_r - V_l}{l} \quad (3)$$

There are three interesting cases with these kinds of drives.

1. If $V_l = V_r$, then we have forward linear motion in a straight line. R becomes infinite, and there is effectively no rotation, means ω is zero.

2. If $V_l = -V_r$, then $R = 0$, and we have rotation about the midpoint of the wheel axis - we rotate in place.

3. If $V_l = 0$, then we have rotation about the left wheel. In this case $R = \frac{l}{2}$. Same is true if $V_r = 0$.

Note that a differential drive robot cannot move in the direction along the axis - this is a singularity. Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect the robot trajectory.

Forward Kinematics for Differential Drive Robots

In Figure 44, assume the robot is at some position (x, y) , headed in a direction making an angle θ with X axis. We assume the robot is centered at a point midway along the wheel axle. By manipulating the control parameters V_l, V_r , we can get the robot to move to different positions and orientations.

Knowing velocities V_l, V_r and using equation 3, we can find the ICC location:

$$ICC = [x - R\sin(\theta), y + R\cos(\theta)] \quad (4)$$

And at time $t + \delta t$ the robot's pose will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \quad (5)$$

This equation simply describes the motion of a robot rotating a distance R about its ICC with an angular velocity of ω .

Another way to understand this is that the motion of the robot is equivalent to 1) translating the ICC to the origin of the coordinate system, 2) rotating about the origin by an angular amount $\omega\delta t$, and 3) translating back to the ICC. Refer to Figure 45

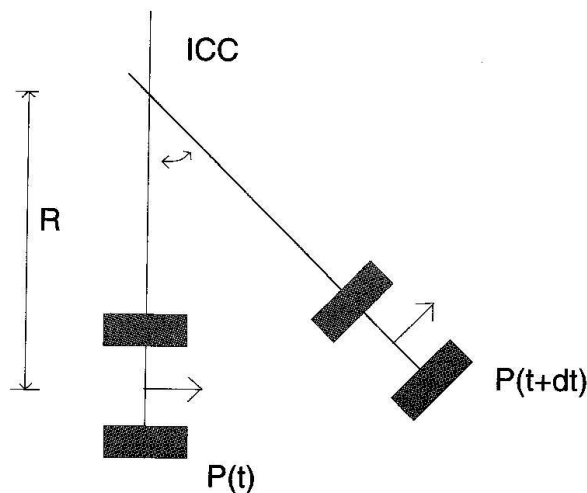


Figure 45 - Forward Kinematics for Differential Robot

Inverse Kinematics of a Mobile Robot

In general, we can describe the position of a robot capable of moving in a particular direction θ_t at a given velocity $V(t)$ as:

$$x(t) = \int_0^t V(t) \cos[\theta(t)] dt \quad (6)$$

$$y(t) = \int_0^t V(t) \sin[\theta(t)] dt \quad (7)$$

$$\theta(t) = \int_0^t \omega(t) dt \quad (8)$$

For our robot platform the equations become;

$$x(t) = \frac{1}{2} \int_0^t [v_r(t) + v_l(t)] \cos[\theta(t)] dt \quad (9)$$

$$y(t) = \frac{1}{2} \int_0^t [v_r(t) + v_l(t)] \sin[\theta(t)] dt \quad (10)$$

$$\theta(t) = \frac{1}{l} \int_0^t [v_r(t) - v_l(t)] dt \quad (11)$$

A related question is: How can we control the robot to reach a given configuration (x, y, θ) – this is known as the inverse kinematics problem.

Unfortunately, a differential drive robot imposes what are called non-holonomic constraints on establishing its position. For example, the robot cannot move laterally along its axle. A similar nonholonomic constraint is a car that can only turn its front wheels. It cannot move directly sidewise, as parallel parking a car requires a more complicated set of steering maneuvers. So we cannot simply specify an arbitrary robot pose (x, y, θ) and find the velocities that will get us there.

For the special cases of $v_r = v_l = v$ (robot moving in a straight line) the motion equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v \cos(\theta) \delta t \\ y + v \sin(\theta) \delta t \\ \theta \end{bmatrix} \quad (12)$$

If $v_r = -v_l = v$, then the robot rotates in place and the equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v \delta t / l \end{bmatrix} \quad (13)$$

This motivates a strategy of moving the robot in a straight line, then rotating for a turn in place, and then moving straight again as a navigation strategy for differential drive robots.

We define the following terms: r_{wheel} : wheel radius. D_{robot} : length of the differential drive wheel axle. V_{wheel} : magnitude of wheel velocity measured in rpm.

To calculate an angular rotation \emptyset of the robot, we need to find an equation for the amount of time we need to turn the robot an angle of \emptyset degrees using a wheel velocity of V_{wheel} . In this example, we assume that the wheels are turning in the opposite direction at the same velocity (robot is turning in place).

Given a time, the wheel will turn:

$$Distance_{wheel} = V_{wheel} \times r_{wheel} \times t \quad (14)$$

To determine the time to turn the robot a specified angle in place, we note that the entire circumference C of the robot when it turns 360° is πD_{robot} . We can turn an angle of ϕ in time t using the equation [24]:

$$\frac{Distance_{wheel}}{C} = \frac{\phi}{2\pi} \quad (15)$$

$$\frac{V_{wheel} \times r_{wheel} \times t}{C} = \frac{\phi}{2\pi} \quad (16)$$

$$t = \frac{\phi C}{2\pi \times V_{wheel} \times r_{wheel}} \quad (17)$$

where ϕ in radians.

Therefore, by tracking one of the parameters, distance or time, one can turn the robot by a specific angle. Also the digital compass in the robot can be used in the turning algorithm.

By using these equations the time or distance for the motor turns are calculated and the command for setting direction and rpm values are given to microcontroller.

5.3.3 Set Distance

The distance to move the robot can be set by using this method. The function gets the direction of the motion and distance to move and velocity of the motion. The equation for distance is;

$$Distance_{wheel} = V_{wheel} \times r_{wheel} \times t$$

where

V_{wheel} : The angular velocity of the wheels

r_{wheel} : The radius of the wheels

t : Time of motion

Therefore, by tracking one of the parameters, distance (calculated by encoder ticks) or time, one can move the robot with the specified distance value.

By using this equation the time for motion is calculated and the command for setting direction and rpm values are given to microcontroller.

The microcontroller will return a done message for this function, in order to give us a feedback for the motion.

5.4 Kinect Library

Another library for Microsoft Kinect is also prepared for the robot platform. The RGB and Depth sensor data of the Microsoft Kinect is extracted by using Kinect for Windows SDK 1.5 and ready for user as Bitmaps with the following variables:

`_bitmapRGB` for RGB Sensor Data

`_bitmapDepth` for Depth Sensor Data

These variables are global variables in the library and they are connected to events that are fired as the RGB and Depths Sensor Frames are updated. This is approximately ~30fps.

At this point, a researcher can take the robot platform and start developing algorithms. Since, he/she has the library to drive the robot and a robust serial protocol; the user can focus and gives all attention to develop robotic algorithms.

Actions done so far can be summarized as the realization steps of the robot platform. In this chapter required motor, battery and gearbox for this platform is selected, and then detailed design of the chassis and the body of the robot is described and finally the schematics of the control system explained. Low level and high level functions of the robot is explained in detail with the help of the serial protocol. The robot platform is manufactured with respect to the defined specification at this chapter and test results of this platform are discussed at the next chapter.

5.5 Robot Library

A class is constructed for the high and low level functions of the robot. The user will use this class to reach to the robot. With this class, the robot will be an object to the user and the high level functions of the robot will be the functions of the class. The architecture of this class will be explained in this part. First the low level functions and necessary implementation for communication are constructed. These are the serial port handling, the low level functions explained in the previous parts and sensor update events. The serial port handling algorithm is also explained in the previous part and is embedded in this library. The sensor update events are for users to be informed if a new sensor data is received. Every sensor in the robot has an event that user can sign up to. Therefore, if a user wants a sensor data, first the broadcasting mode for that sensor should be opened. Then, the user should register to the related event for the sensor. After that, the user will receive the event in every new sensor data which was broadcasted from the robot with the user specified interval.

The high level functions are also accessible to the user. The user can use these high level functions to control the robot. If a user wants to move the robot with a specified speed, the user will only use the necessary function with the necessary parameters, and the library will manage the rest.

The library reference and example use of the library can be found in the Appendix C.

In addition to this robot library, a simple user control for the robot is created. This will allow a new user to just drag and drop the robot control from the toolbox of the Visual Studio (Figure 46) and simply adjust some properties like serial port name and sensor parameter update interval. Then the user will have 3 outputs from robot shown in the form (ultrasonic sensors, speed and heading angle) and the user will see the heartbeat of the robot and can control the robot using the virtual joystick. This will allow the user just by simply adding the control to a new project, the robot will be accessibly and can be easily teleoperated to see that it works well.

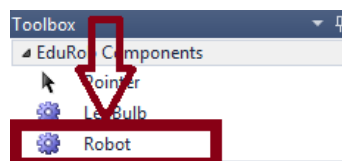


Figure 46 – Robot User Control in the Toolbox of MS Visual Studio

By using the user control just drag and drop the robot control to the newly created project. And the user control will include the connect button, the robot's picture from top and a virtual joystick in the center, heading sensor gauge, speed gauge, heartbeat led, and ultrasonic sensor readings will be drawn to the control when new reading is obtained. After dropping the control to the newly created project a sample screenshot is given in Figure 47.

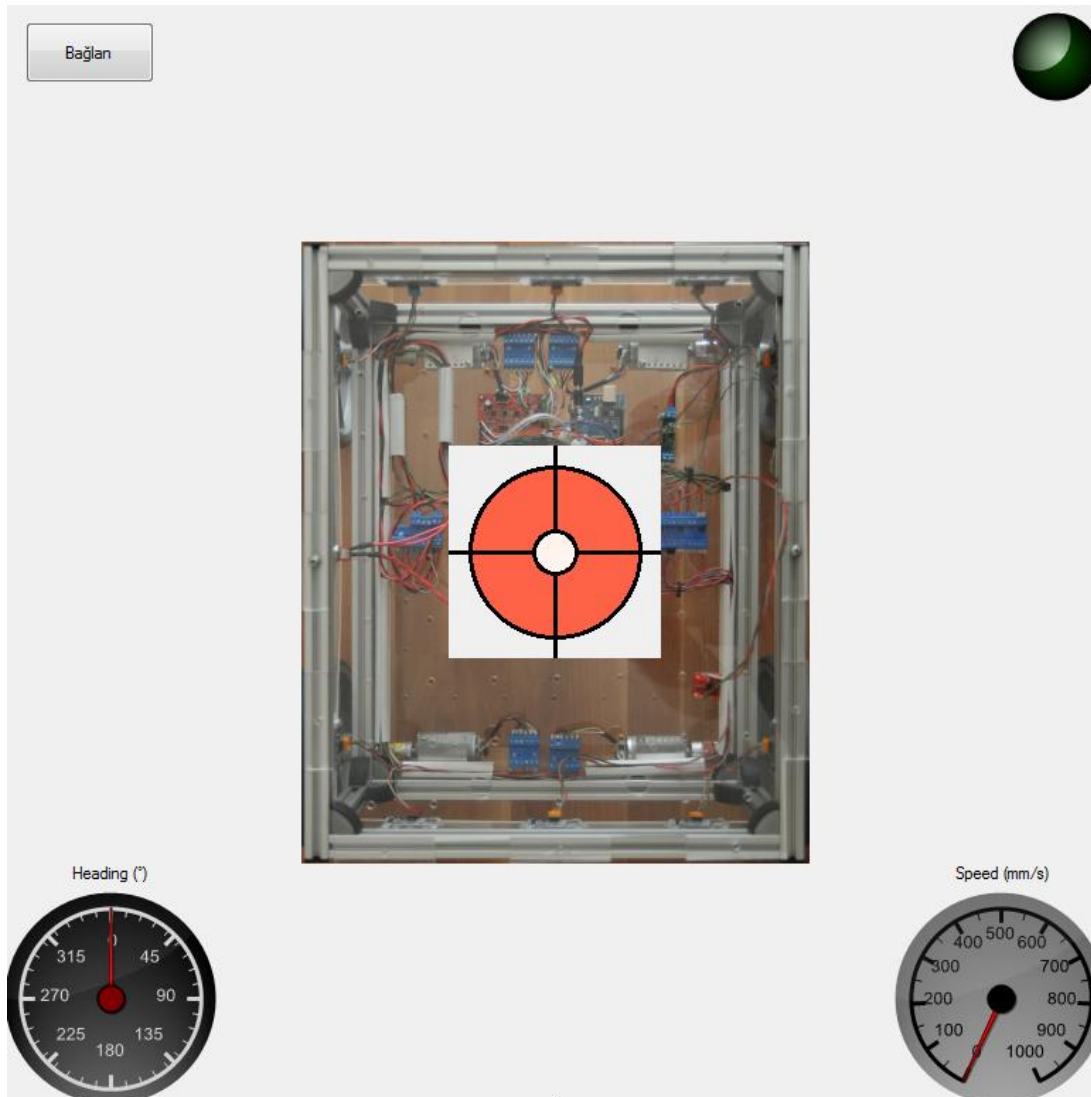


Figure 47 - User Control Screenshot

The properties window will look like Figure 48. The user can change the properties as needed and should set the communication ports to the robot's Arduino cards. The robot port property will set the Arduino port which will be the main controller of the robot. The robot port sonar property will set the Arduino port which is the ultrasonic sensor microcontroller card.

The robot control sample screenshot while in use is shown in Figure 49. In this screenshot the front of the robot is clear, and the other sides are not clear. In this configuration for example robot should be moved to forward. As it can be seen from the Figure 49, the robot can easily be teleoperated by using this user control.

▷ (ApplicationSettings)	
▷ (DataBindings)	
(Name)	robot1
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoScroll	False
▷ AutoScrollMargin	0; 0
▷ AutoScrollMinSize	0; 0
AutoSize	False
AutoSizeMode	GrowOnly
AutoValidate	EnablePreventFocusChange
BackColor	<input type="checkbox"/> Control
BackgroundImage	<input type="checkbox"/> (none)
BackgroundImageLayout	Tile
BorderStyle	None
BroadcastHeading	True
BroadcastingInterval	250
BroadcastSpeed	True
BroadcastUltrasonicSens	True
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
Dock	None
Enabled	True
▷ Font	Microsoft Sans Serif; 8,25pt
ForeColor	<input type="checkbox"/> ControlText
GenerateMember	True
ImeMode	NoControl
▷ Location	12; 12
Locked	False
▷ Margin	3; 3; 3; 3
▷ MaximumSize	0; 0
▷ MinimumSize	0; 0
Modifiers	Private
▷ Padding	0; 0; 0; 0
RightToLeft	No
RobotPort	COM8
RobotPortSonar	COM6
▷ Size	779; 779
TabIndex	2
TabStop	True
Tag	

Figure 48 - Property Window of the User Control

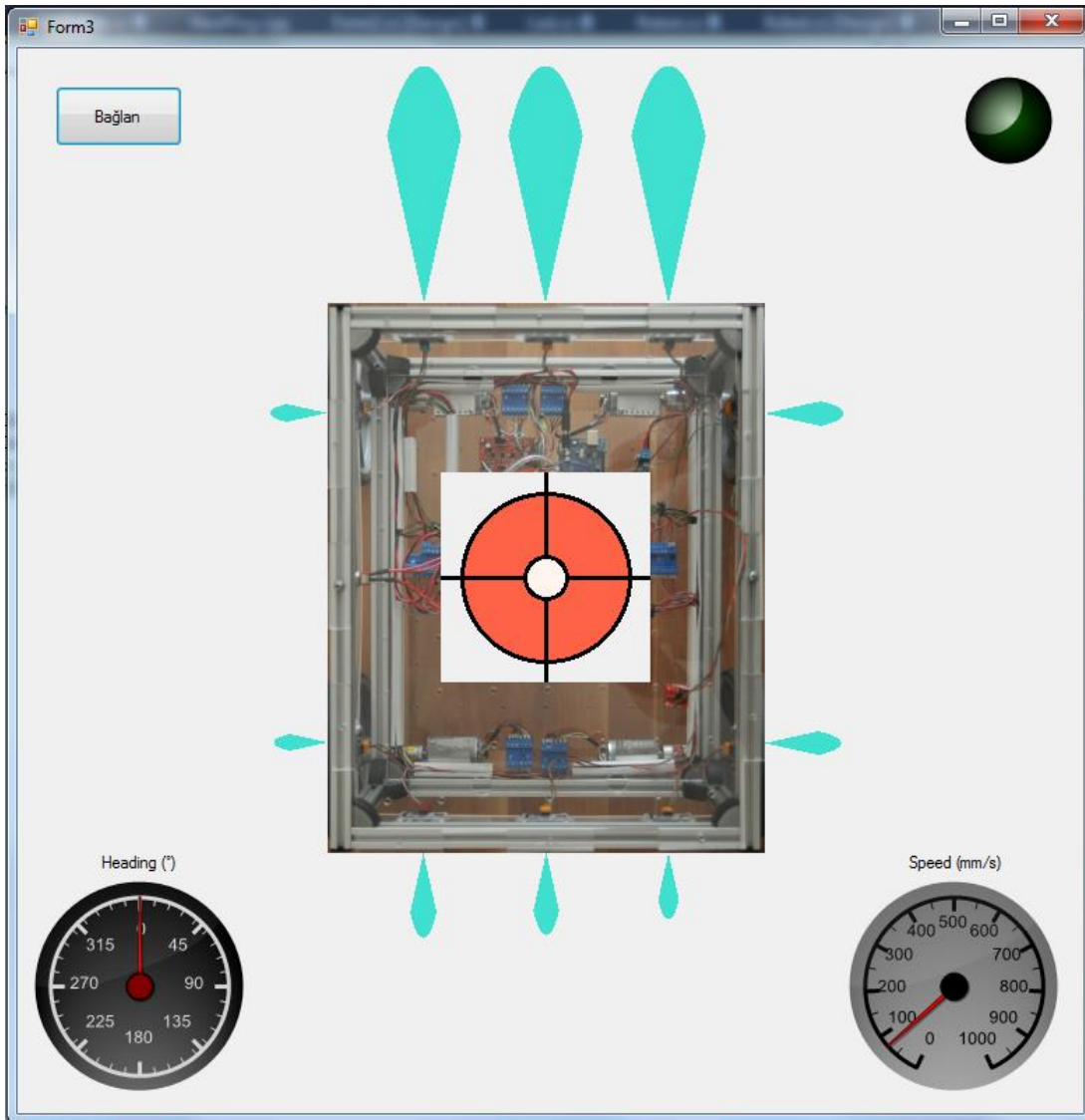


Figure 49 - Example Usage of the User Control

In the next chapter, performed tests and their results can be found.

CHAPTER 6

RESULTS AND DISCUSSIONS

In this chapter the test of the manufactured robot platform is discussed. The tests are done with the manufactured robot platform according to specified design configuration. A sample picture of the manufactured robot platform can be seen in Figure 50.



Figure 50 - Robot Platform

We will use a laptop as the brain of the robot. The manufactured robot has 4 wheels and each one has a motor as described in the previous part. The microcontroller and the motor driver board are implemented on the robot. All the other hardware mentioned in the previous part is implemented. The robot has 10 ultrasonic sensors on it. The front and back side (We will refer to the part of where the microcontroller and motor driver board is close as the front side of the robot for simplicity, however there is no front or back side in the robot platform as it is symmetrical) of the robot has 3 sensors and the sides has 2 sensors. The tilt compensated heading sensor is also implemented. The top view (inside view) of the robot platform can be seen in Figure 51.

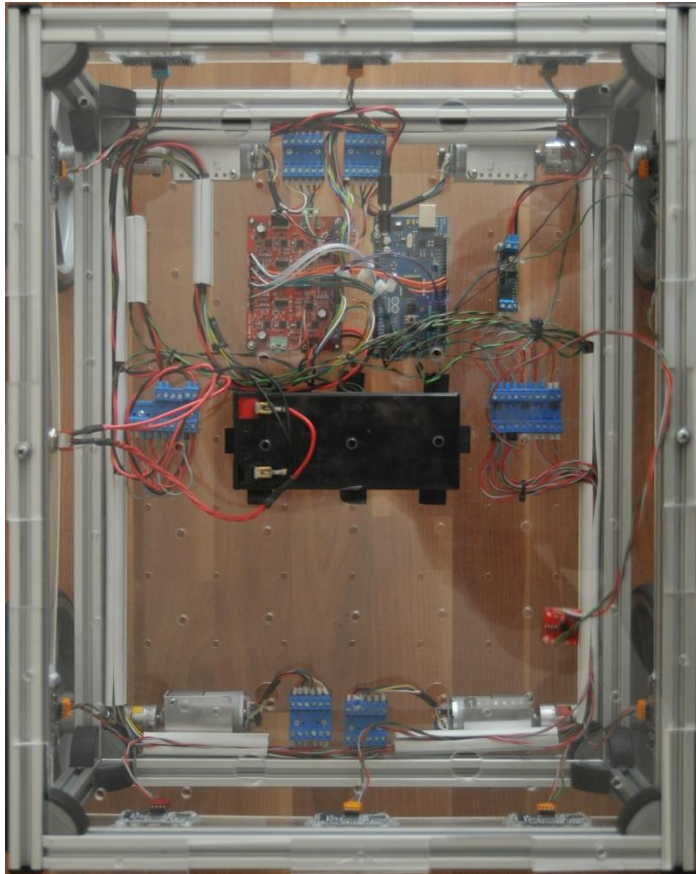


Figure 51 - Inside of the Robot

A user interface is prepared for seeing the low level implementation, the sensor outputs and give commands to the robot (Figure 52).

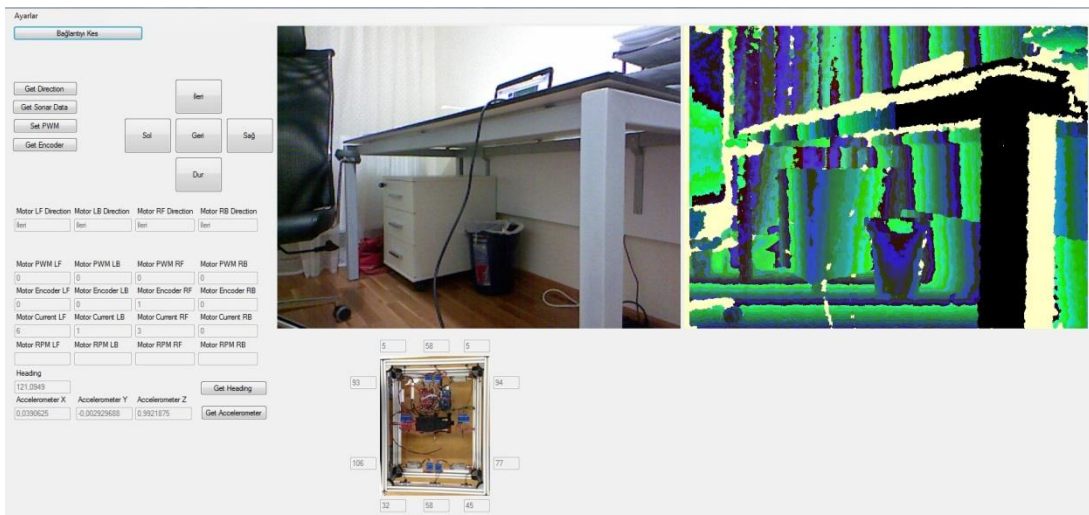


Figure 52 - User Interface

The tests will include the high level functions. We will test how the robot will react to those functions and the accuracy that we have in the robot.

6.1 Set Linear Speed Test

First test is the set linear speed. We will give the robot some test value speeds and drive it. After the robot start to move and the speed of the robot become constant, it will pass the first proximity sensor which will start a timer. The test setup will have 4 proximity sensors, and the distance between 1st and 2nd sensor is 33 cm. The distance between 1st and 3rd one is 100 cm and the distance between 3rd and 4th one is 33 cm also. The sketch of the test setup is given in the Figure 53. An algorithm for the test is developed such that all proximity sensors will save 2 values, the first entering of the robot in the sensor field and leaving of the sensor. Therefore, in each test we will have 8 time values, from which we will calculate the speed values. After getting the time values and calculating the speed, it will be compared with the given speed.

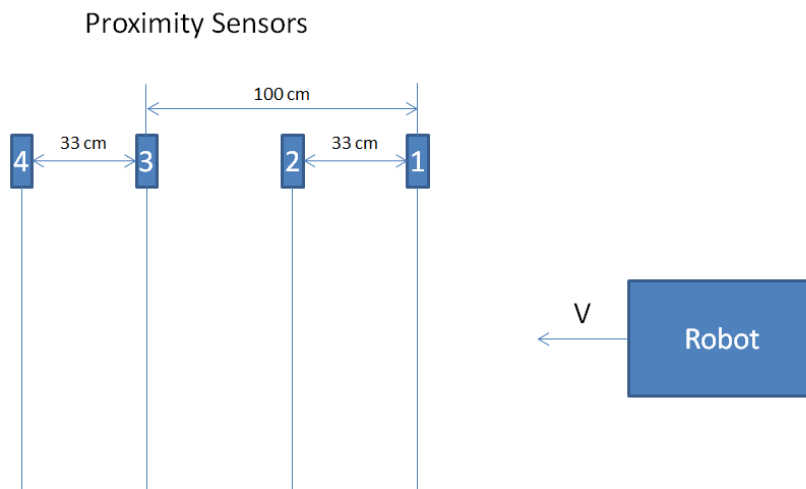


Figure 53 - Test Setup of the Speed Test

Therefore, after completion of the test we will calculate 7 speed value from the test results. First, the speed is calculated between 1st and 2nd proximity sensor time elapsed and from their distance. Also, every proximity in and out time values are used to calculate the speed by using the robot length 50 cm. The table for calculating speed values is given in Appendix B.

For speed tests 4 different speed values are used. These are 0.2 m/s, 0.4 m/s, 0.6 m/s and 0.8 m/s. In every speed 3 tests are run and their average value is calculated. The test results for 0.2 m/s is given below.

Table 81 - Speed Test 1 - 0.2 m/s

Test 1: 0.2 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 1786 ms	1786	0,18
[1]Out: 2720 ms	2720	0,18
[2]Out: 4370 ms	4370	0,19
[3]In: 5168 ms	5168	0,19
[4]In: 6908 ms	6908	0,19
[3]Out: 7858 ms	7858	0,19
[4]Out: 9528 ms	9528	0,19
	Avarage	0,19

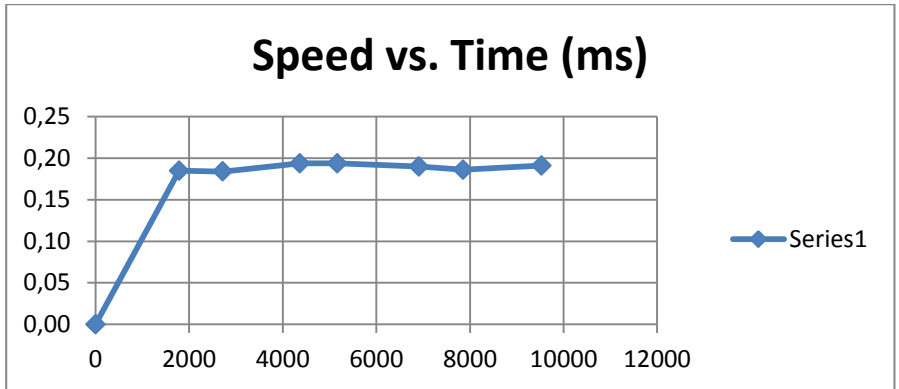


Figure 54 - Speed Test 1 - 0.2 m/s

The second test results are given below.

Table 82 - Speed Test 2 - 0.2 m/s

Test 2: 0.2 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 1680 ms	1680	0,20
[1]Out: 2554 ms	2554	0,20
[2]Out: 4128 ms	4128	0,20
[3]In: 4928 ms	4928	0,20
[4]In: 6532 ms	6532	0,21
[3]Out: 7486 ms	7486	0,20
[4]Out: 9112 ms	9112	0,19
	Avarage	0,20

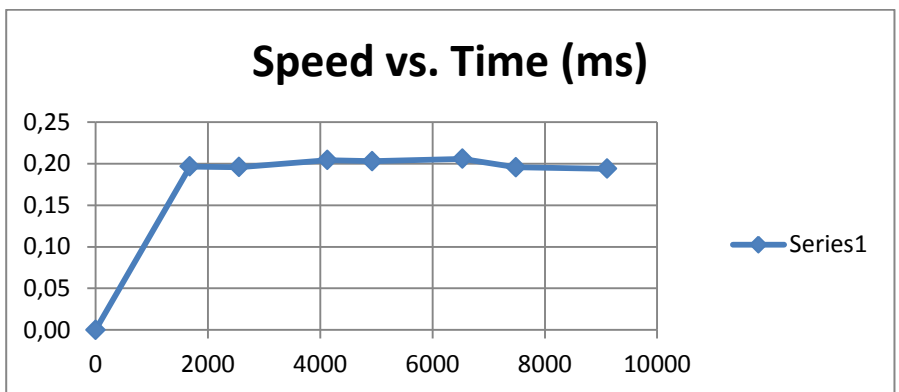


Figure 55 - Speed Test 2 - 0.2 m/s

The third test results are given below.

Table 83 - Speed Test 3 - 0.2 m/s

Test 3: 0.2 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00

[2]In: 1646 ms	1646	0,20
[1]Out: 2512 ms	2512	0,20
[2]Out: 4062 ms	4062	0,21
[3]In: 4836 ms	4836	0,21
[4]In: 6416 ms	6416	0,21
[3]Out: 7358 ms	7358	0,20
[4]Out: 8956 ms	8956	0,20
	Avarage	0,20

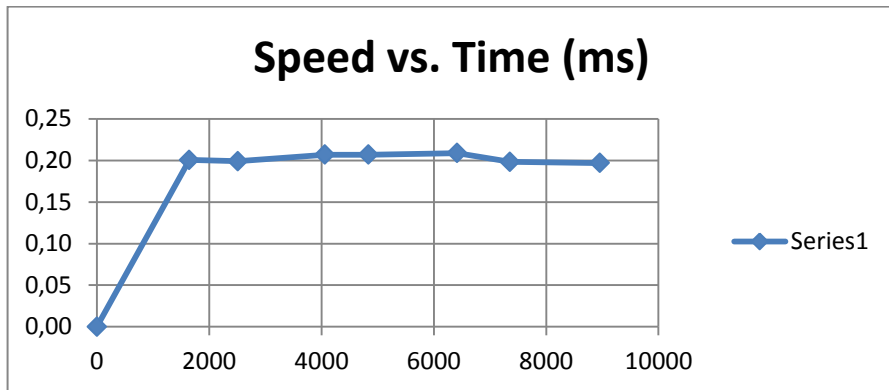


Figure 56 - Speed Test 3 - 0.2 m/s

The average of the 3 tests is given below.

Table 84 - 0.2 m/s Test Results

Test 1	0,19
Test 2	0,20
Test 3	0,20
Avarage	0,20

The result is very satisfactory.

The test results for 0.4 m/s is given below.

Table 85 - Speed Test 4 - 0.4 m/s

Test 4: 0.4 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 738 ms	738	0,45
[1]Out: 1156 ms	1156	0,43
[2]Out: 1882 ms	1882	0,44
[3]In: 2204 ms	2204	0,45
[4]In: 2900 ms	2900	0,47
[3]Out: 3362 ms	3362	0,43
[4]Out: 4090 ms	4090	0,42
	Avarage	0,44

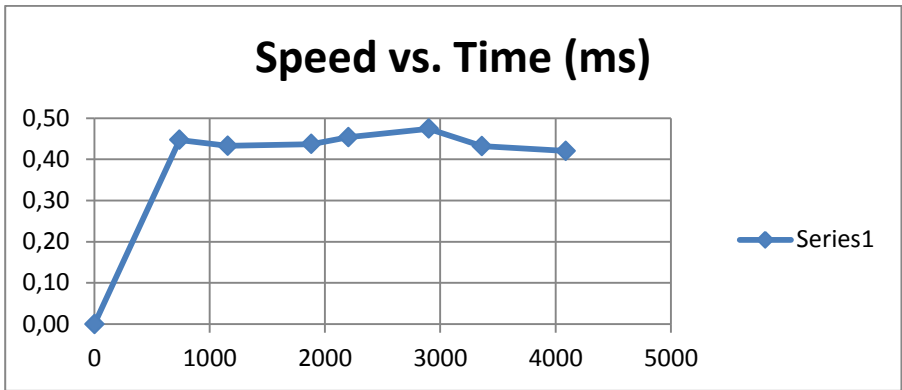


Figure 57 - Speed Test 4 - 0.4 m/s

The second test results are given below.

Table 86 - Speed Test 5 - 0.4 m/s

Test 5: 0.4 m/s		
	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 788 ms	788	0,42
[1]Out: 1210 ms	1210	0,41
[2]Out: 1958 ms	1958	0,43
[3]In: 2318 ms	2318	0,43
[4]In: 3086 ms	3086	0,43
[3]Out: 3546 ms	3546	0,41
[4]Out: 4324 ms	4324	0,40
	Avarage	0,42

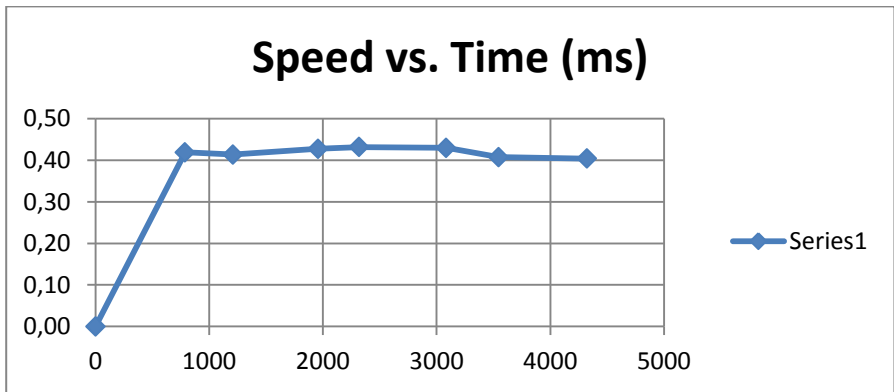


Figure 58 - Speed Test 5 - 0.4 m/s

The third test results are given below.

Table 87 - Speed Test 6 - 0.4 m/s

Test 6: 0.4 m/s		
	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 820 ms	820	0,40
[1]Out: 1250 ms	1250	0,40

[2]Out: 2016 ms	2016	0,42
[3]In: 2390 ms	2390	0,42
[4]In: 3172 ms	3172	0,42
[3]Out: 3634 ms	3634	0,40
[4]Out: 4440 ms	4440	0,39
	Avarage	0,41

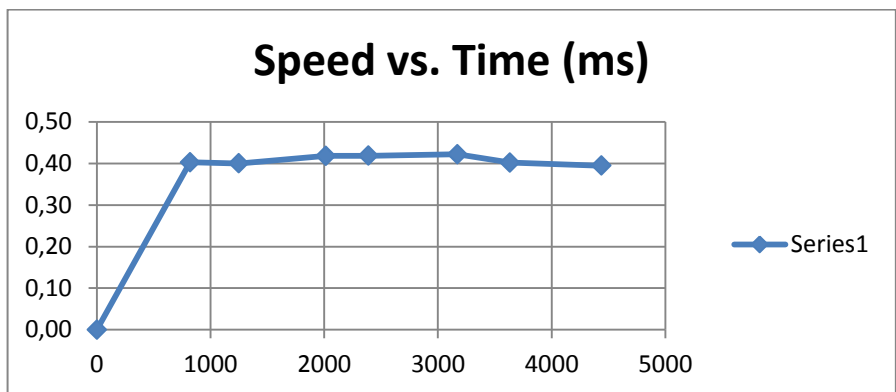


Figure 59 - Speed Test 6 - 0.4 m/s

The average of the 3 tests is given below.

Table 88 - 0.4 m/s Test Results

Test 1	0,44
Test 2	0,42
Test 3	0,41
Avarage	0,42

The result is very satisfactory.

The test results for 0.6 m/s is given below.

Table 89 - Speed Test 7 - 0.6 m/s

Test 7: 0.6 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 580 ms	580	0,57
[1]Out: 878 ms	878	0,57
[2]Out: 1434 ms	1434	0,59
[3]In: 1706 ms	1706	0,59
[4]In: 2274 ms	2274	0,58
[3]Out: 2578 ms	2578	0,57
[4]Out: 3136 ms	3136	0,58
	Avarage	0,58

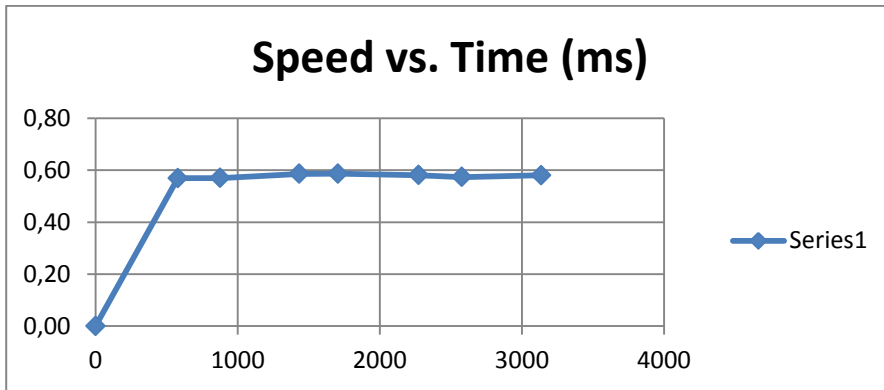


Figure 60 - Speed Test 7 - 0.6 m/s

The second test results are given below.

Table 90 - Speed Test 8 - 0.6 m/s

Test 8: 0.6 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 588 ms	588	0,56
[1]Out: 892 ms	892	0,56
[2]Out: 1444 ms	1444	0,58
[3]In: 1724 ms	1724	0,58
[4]In: 2302 ms	2302	0,57
[3]Out: 2620 ms	2620	0,56
[4]Out: 3186 ms	3186	0,57
	Avarage	0,57

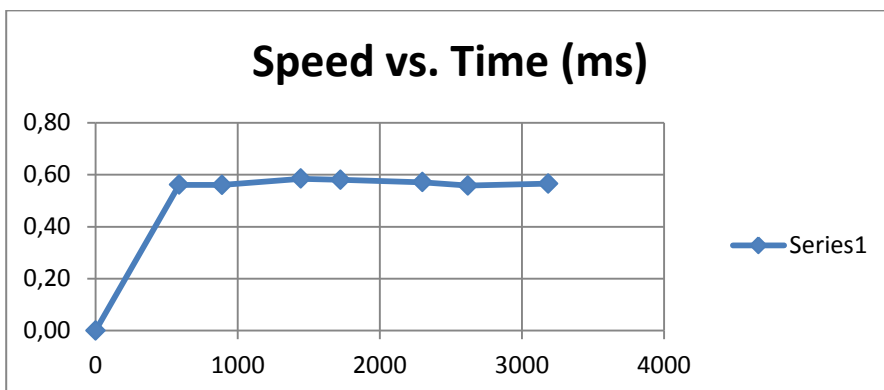


Figure 61 - Speed Test 8 - 0.6 m/s

The third test results are given below.

Table 91 - Speed Test 9 - 0.6 m/s

Test 9: 0.6 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 590 ms	590	0,56
[1]Out: 898 ms	898	0,56

[2]Out: 1460 ms	1460	0,57
[3]In: 1738 ms	1738	0,58
[4]In: 2304 ms	2304	0,58
[3]Out: 2640 ms	2640	0,55
[4]Out: 3206 ms	3206	0,55
	Avarage	0,57

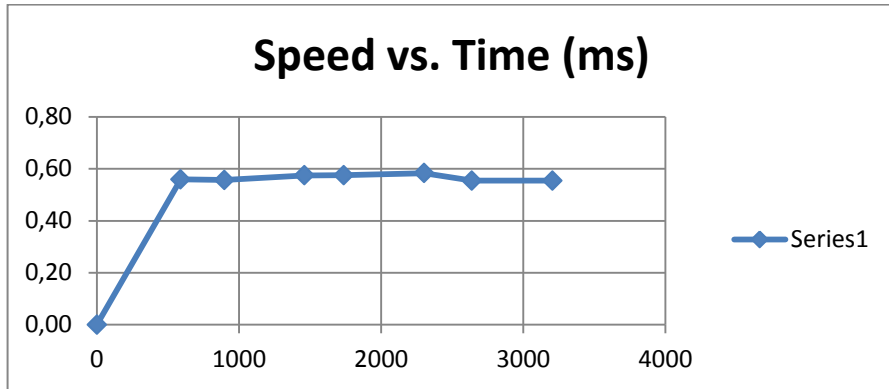


Figure 62 - Speed Test 9 - 0.6 m/s

The average of the 3 tests is given below.

Table 92 - 0.6 m/s Test Results

Test 1	0,58
Test 2	0,57
Test 3	0,57
Avarage	0,57

The result is very satisfactory.

The test results for 0.8 m/s is given below.

Table 93 - Speed Test 10 - 0.8 m/s

Test 10: 0.8 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 450 ms	450	0,73
[1]Out: 684 ms	684	0,73
[2]Out: 1116 ms	1116	0,75
[3]In: 1326 ms	1326	0,75
[4]In: 1768 ms	1768	0,75
[3]Out: 2014 ms	2014	0,73
[4]Out: 2452 ms	2452	0,73
	Avarage	0,74

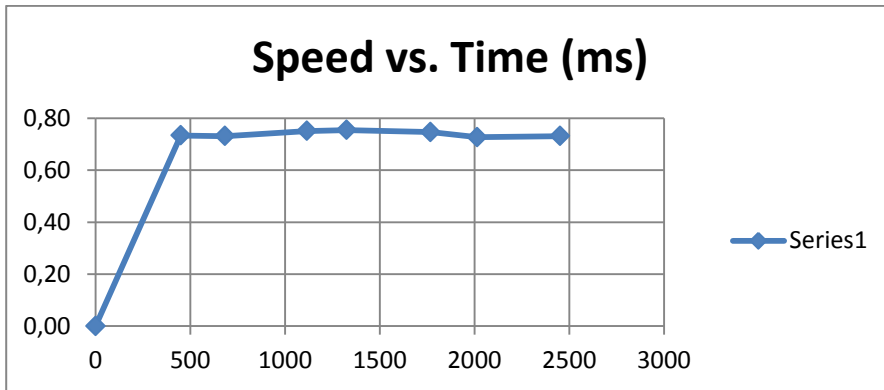


Figure 63 - Speed Test 10 - 0.8 m/s

The second test results are given below.

Table 94 - Speed Test 11 - 0.8 m/s

Test 11: 0.8 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 454 ms	454	0,73
[1]Out: 696 ms	696	0,72
[2]Out: 1130 ms	1130	0,74
[3]In: 1344 ms	1344	0,74
[4]In: 1774 ms	1774	0,77
[3]Out: 2044 ms	2044	0,71
[4]Out: 2484 ms	2484	0,70
	Avarage	0,73

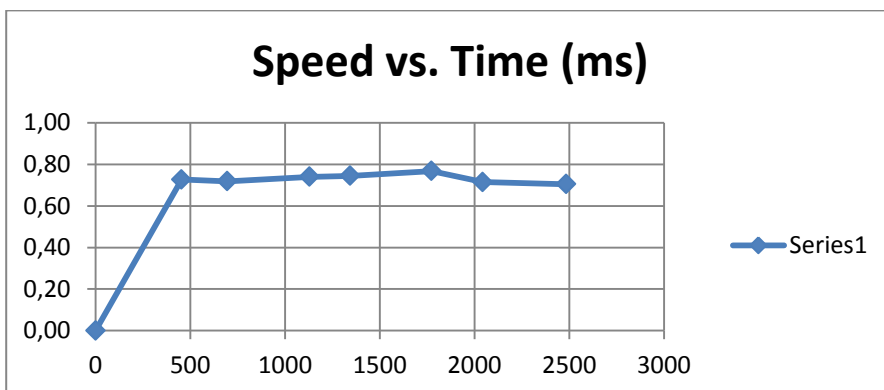


Figure 64 - Speed Test 11 - 0.8 m/s

The third test results are given below.

Table 95 - Speed Test 12 - 0.8 m/s

Test 12: 0.8 m/s	Time Value (ms)	Speed (m/s)
[1]In: 0 ms	0	0,00
[2]In: 456 ms	456	0,72
[1]Out: 698 ms	698	0,72

[2]Out: 1128 ms	1128	0,74
[3]In: 1342 ms	1342	0,75
[4]In: 1784 ms	1784	0,75
[3]Out: 2044 ms	2044	0,71
[4]Out: 2486 ms	2486	0,71
	Avarage	0,73

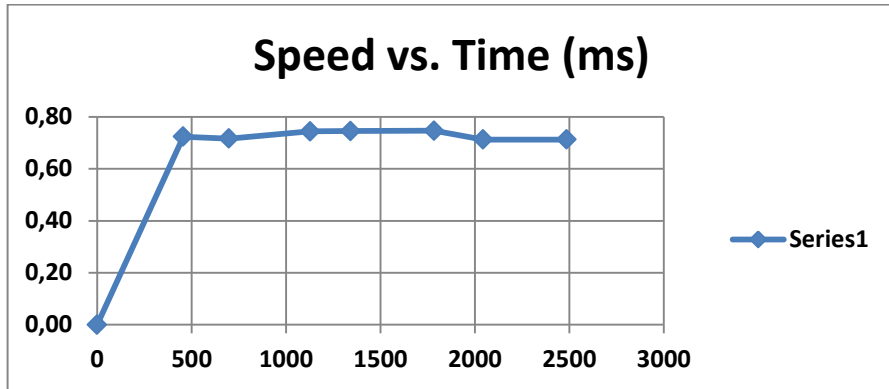


Figure 65 - Speed Test 12 - 0.8 m/s

The average of the 3 tests is given below.

Table 96 - 0.8 m/s Test Results

Test 1	0,74
Test 2	0,73
Test 3	0,73
Avarage	0,73

In this test it can be seen that the robot has difficulty to reach the setpoint. It is seen that the results are the top speed of the robot.

A screenshot from one of the tests is given in Figure 66.

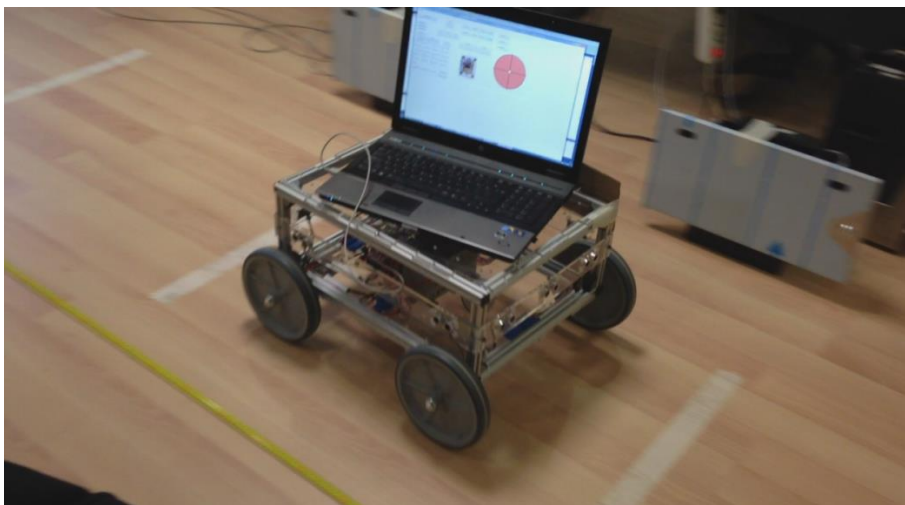


Figure 66 - Sample Screenshot of Speed Test

In conclusion, the results for the speed test are given in Table 97.

Table 97 - Speed Test Result

0.2 m/s Speed Test	0,20
0.4 m/s Speed Test	0,42
0.6 m/s Speed Test	0,57
0.8 m/s Speed Test	0,73

6.2 Turn Angle Test

The second test is the Turn Angle (Point Turn). We will give the robot some test value angles and drive it. The angle from the start of the test (Figure 67) and the angle at the end of the test (Figure 68) is measured using an iPhone digital compass. The robot will use the heading sensor to turn an angle. Then we will compare the results.

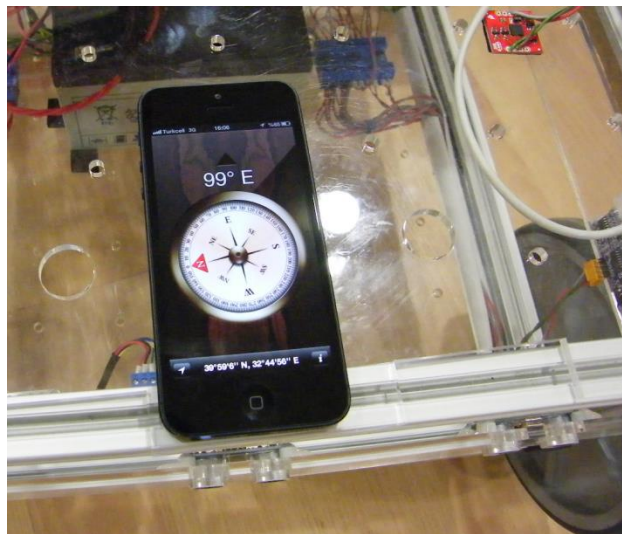


Figure 67 - Sample Screenshot of Angle Before The Test

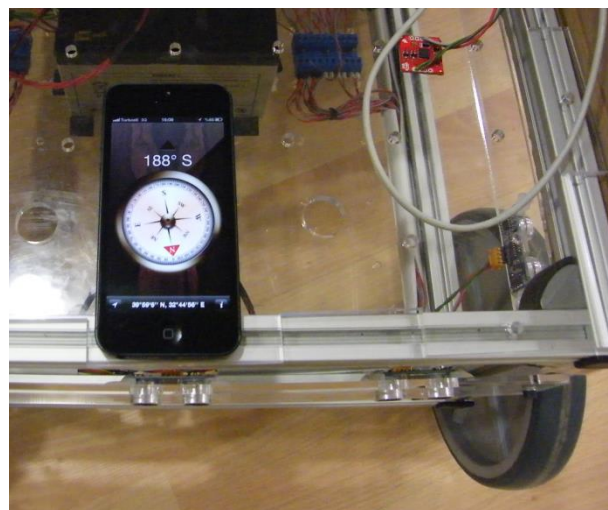


Figure 68 - Sample Screenshot of Angle After The Test

2 different angles have been given to the robot for this test. The schematic of the test can be seen in Figure 69. The robot TurnAngle function is tested. The robot will turn the given angle with a given turn direction from its position.

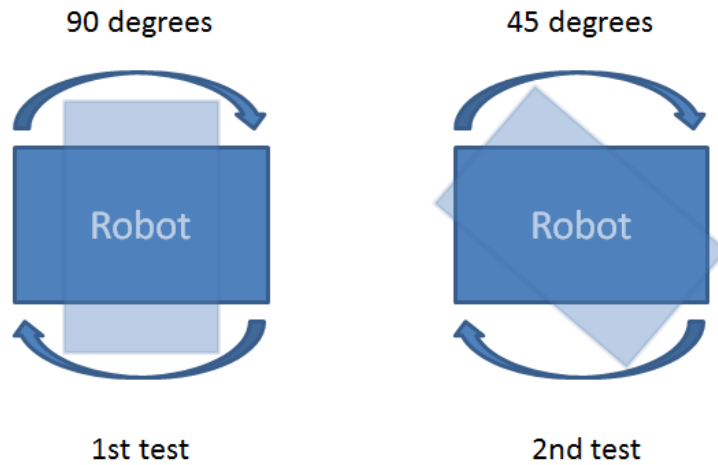


Figure 69 - Schematic of TurnAngle Tests

The test results for 90 degree test are given below. The test is conducted 5 times.

Table 98 – Turn Angle Test (90 degrees) Results

Start Angle	End Angle	Angle Turned
90	181	91
258	352	94
99	188	89
188	268	80
101	193	92

The test results for 45 degree test are given below. The test is conducted 5 times.

Table 99 – Turn Angle Test (45 degrees) Results

Start Angle	End Angle	Angle Turned
348	28	40
28	73	45
73	123	50
123	169	46
169	214	45

In conclusion, the turn angle test is satisfactory. It can be seen that the robot is turned with the desired angle in an approximately %3-4 error. This result is normal considering the heading sensor error.

6.3 Turn To Angle Test

The third test is the Turn to Angle function. We will give the robot some test value angles and drive it. An iPhone digital compass is used to obtain the start (Figure 70) and end angle (Figure 71) of the robot during the test. The robot will use the heading sensor to turn an angle. Then we will compare the results.

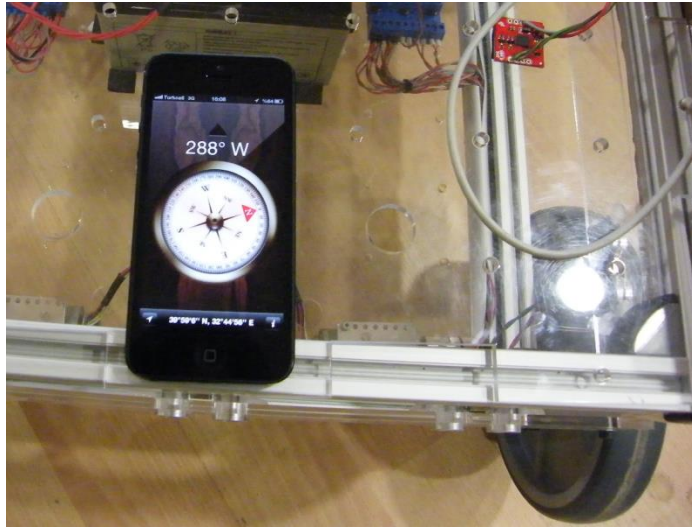


Figure 70 - Sample Screenshot of Angle Before The TurnToAngle Test

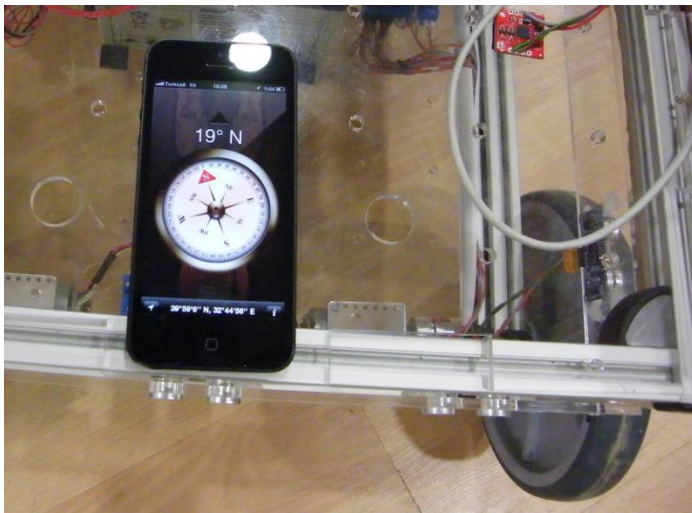


Figure 71 - Sample Screenshot of Angle After The TurnToAngle Test

3 different angles have been given to the robot for this test. The robot TurnToAngle function is tested. The robot will turn to the given angle with a given turn direction from its position.

The test results for 20 degree test are given below. The test is conducted 2 times.

Table 100 – Turn to Angle Test (Turn To 20 degree) Results

Start Angle	End Angle
193	19
288	19

The test results for 100 degree test are given below. The test is conducted 2 times.

Table 101 – Turn to Angle Test (Turn To 100 degree) Results

Start Angle	End Angle
21	105
250	102

The test results for 250 degree test are given below. The test is conducted 2 times.

Table 102 – Turn to Angle Test (Turn To 100 degree) Results

Start Angle	End Angle
127	245
345	248

In conclusion, the turn to angle test is satisfactory. It can be seen that the robot is turned with the desired angle in an approximately %3-4 error. This result is normal considering the heading sensor error.

6.4 Set Distance Test

The last test is the set distance. We will give the robot some test value distance and drive it. Then we will measure the distance that the robot travelled with a meter (Figure 72, Figure 73, Figure 74). Then we will compare the results with the given set point.

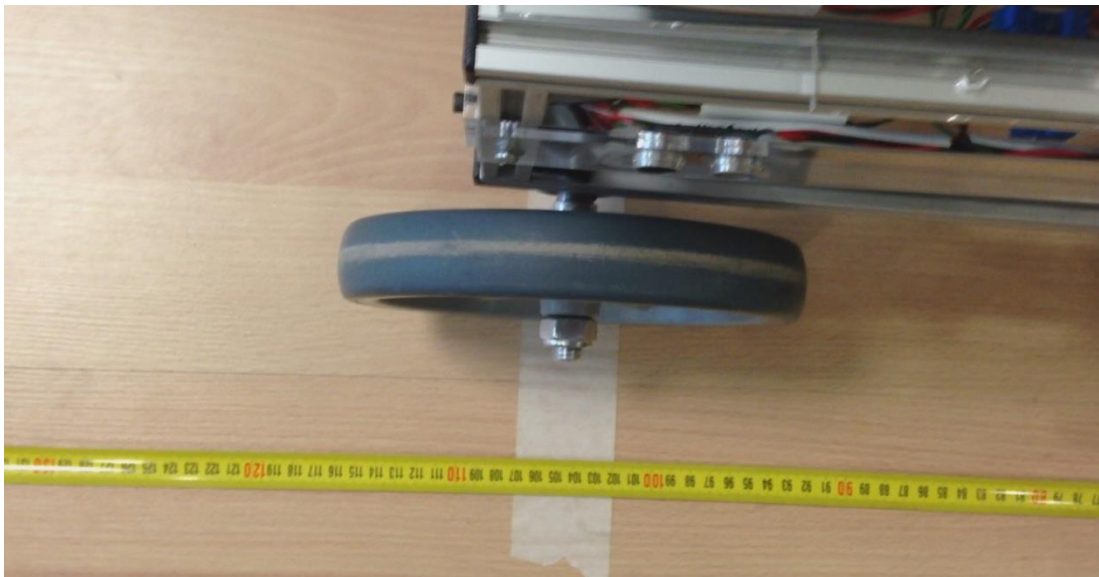


Figure 72 - Sample SetDistance 1 m Test Screenshot



Figure 73 - Sample SetDistance 2 m Test Screenshot

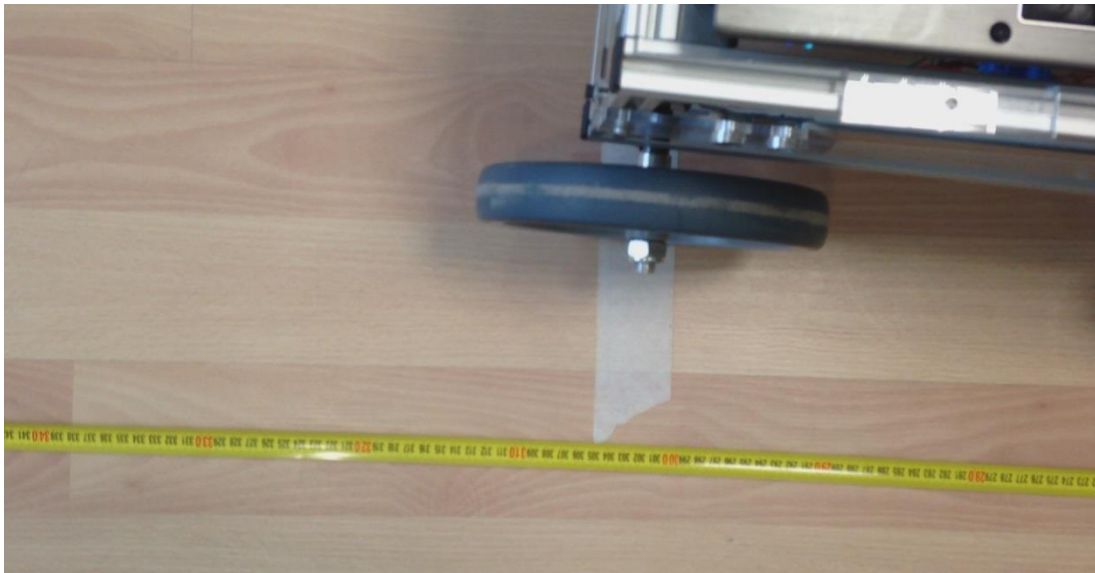


Figure 74 - Sample SetDistance 3 m Test Screenshot

3 different distance have been given to the robot for this test. The robot SetDistance function is tested. The robot will move to the specified distance with 0.2 m/s speed.

The test results for 1 m test are given below. The test is conducted 4 times.

Table 103 - Set Distance Test (1 m) Results

Measured Distance From The Start Point
107
103
104
104

The test results for 2 m test are given below. The test is conducted 4 times.

Table 104 - Set Distance Test (1 m) Results

Measured Distance From The Start Point
205
206
208
204

The test results for 2 m test are given below. The test is conducted 4 times.

Table 105 - Set Distance Test (1 m) Results

Measured Distance From The Start Point
307
301
307
306

It can be seen that the robot moved all the distance with a maximum of %3-4 error. This result is very satisfactory.

6.5 Obstacle Avoidance Implementation

After these test, we implemented some basic robotic algorithms to show that the developed robot platform is capable of running robotic algorithms. For this purpose, an obstacle avoidance algorithm is implemented using Kinect. The picture of the robot platform with the Kinect is given in Figure 75.

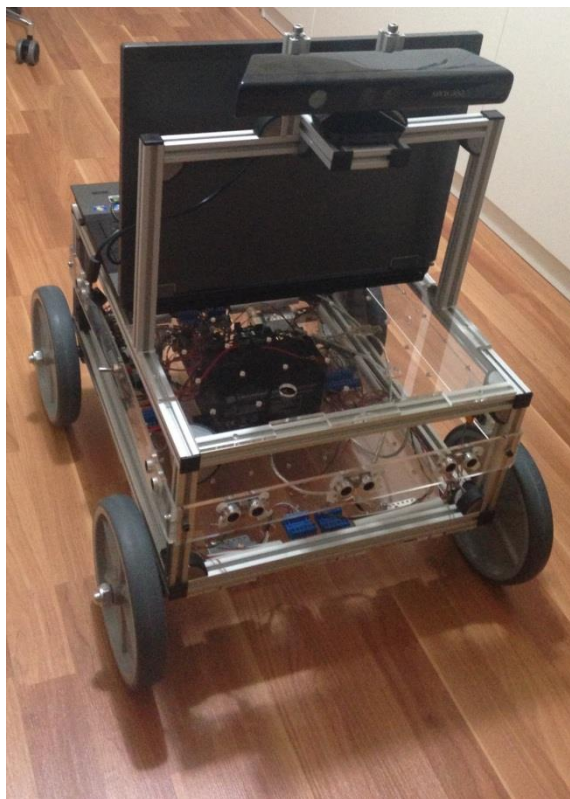


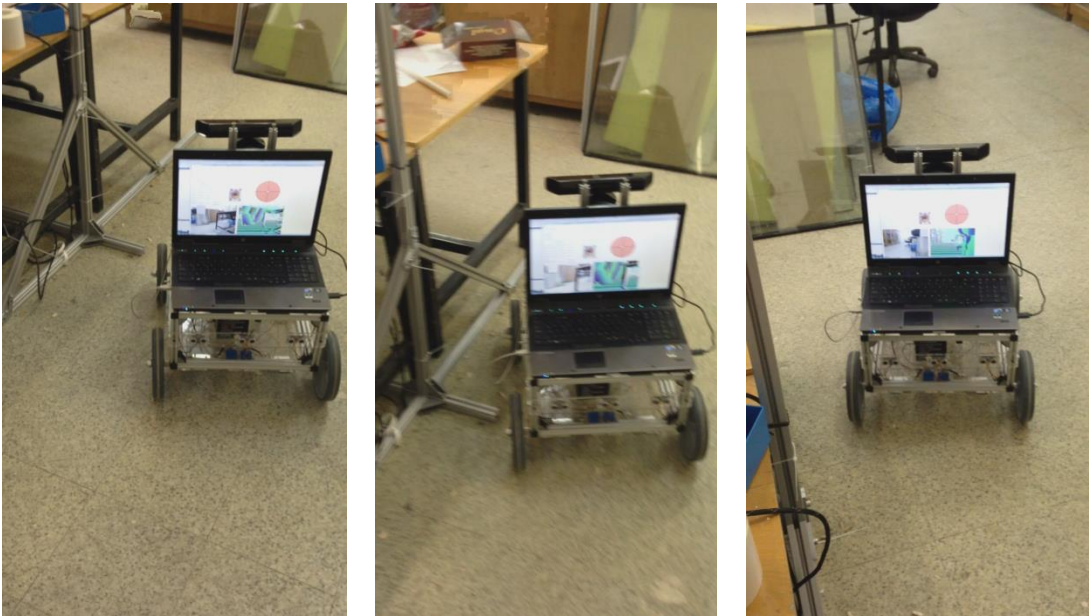
Figure 75 - Obstacle Avoidance Setup on the Robot

The Kinect depth map is used in the navigation of the robot. A complex obstacle avoidance algorithm can be implemented however, the purpose of implementing the obstacle avoidance is to show that the developed robot platform can run robotic research algorithm. Therefore, we implement a simple decision making algorithm that focuses on computational efficiency and simple. The goal of this algorithm is to find the most appropriate direction from the possible directions of movement, i.e. forward, left, and right. In order to achieve that, the depth map of the Kinect is divided into three equally sized windows as shown in the Figure 76.



Figure 76 - Depth Map Divided into Three Windows

The boundary regions of the depth map are excluded in constructing because such regions are often problematic. In each window, the pixels whose depth value is greater than a defined threshold are enumerated. Then the enumeration results are normalized toward the window's pixels population and then examined. The smaller value points out that the direction is the most appropriate direction to move. If all of the values are greater than some predefined threshold value, the robot will move backward to obtain a wide angle. The sample screenshot from one of the obstacle avoidance algorithm test is given in Figure 77.



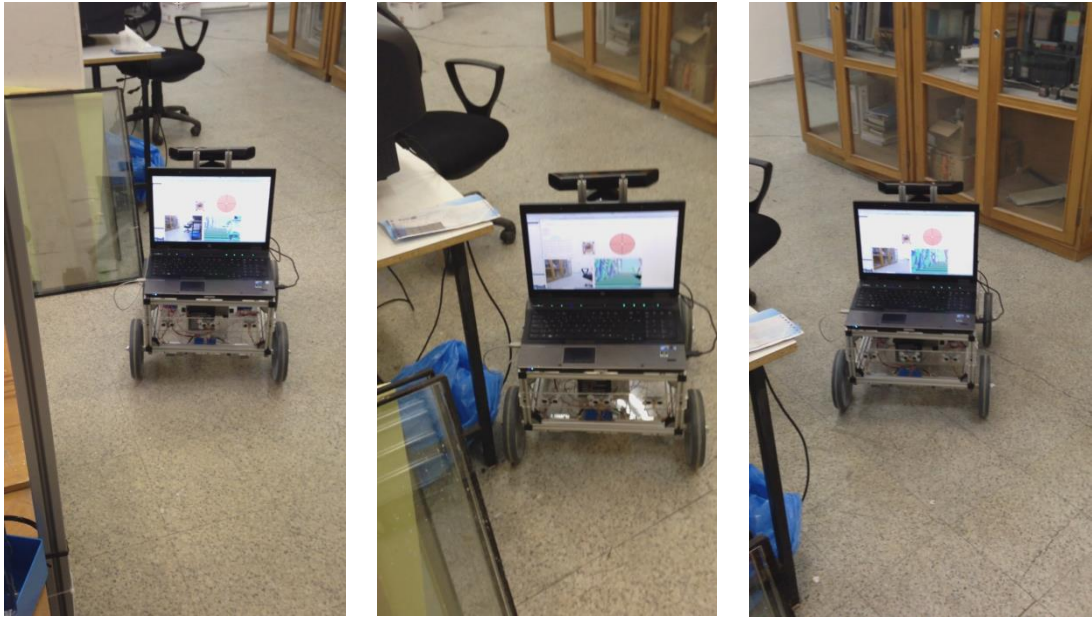


Figure 77 - Sample Obstacle Avoidance Screenshots

The performed tests show that the developed robot platform, control architecture and robot library is fulfills the design requirements and tasks. By implementing the obstacle avoidance algorithm, it is shown that this platform can be ready to be used for developing robotic algorithms. A user can easily get started to develop such tasks and concentrate on algorithm development.

CHAPTER 7

SUMMARY, CONCLUSIONS AND FUTURE WORK

In this thesis, a modular and cost efficient robot platform for educational research purposes is designed and manufactured. The robot base is designed so that it can be configured as the specific application needs, so that the motor number or dimension or the sensors that the robot has can be changed. Electronic infrastructure of the robot platform is designed and implemented. 2 Arduino Mega microcontroller are used to control the robot platform. One is only for reading the ultrasonic readings and the other one is used for motor controller and other jobs. A serial communication protocol is developed and implemented to both PC and microcontroller side. In addition, code is developed for reading the digital compass, and also a pid controller is implemented to drive the motors. On the other hand, a library is created in Microsoft Visual Studio using C#. This library is used to communicate with the robot platform and to use it. In addition to the library, a user control is created to quickly attach to the robot platform and start to use it. The modular structure of the robot allows the user to attach any kind of sensor to the platform. At design stage, the expected performance of the robot is configured, and at the test the modularity is shown. A researcher will no longer spend time on low level implementation of the modules. He/She will use the implemented serial protocol and high level function library and spend time on developing robotic algorithms.

The high level functions of the robot platform is tested. These tests include speed test, turn angle test, turn to angle test and distance test. All of the test results are satisfactory. Also, a simple obstacle avoidance algorithm is implemented on the robot platform. The high level functions are used in this algorithm. It has shown that the developed robot platform is capable of running robotic algorithms and suitable for developing research algorithms.

In the near future, a self-charging algorithm for the robot will be implemented. This will allow the robot for continuous operation. Also, a suspension system will be implemented to the wheels so that the robot platform can also be used at outdoor use. It is also intended that several of these robots can be connected with each other for specific tasks.

REFERENCES

1. Mobile Robots Company product datasheet web page, Last accessed in January, 2013 [Online]. <http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT-RevA.sflb.ashx>.
2. K-TEAM corporation Koala II robot specifications web page, Last accessed in January, 2013 [Online]. <http://www.k-team.com/mobile-robotics-products/koala>
3. AAI Canada Inc. Gaia-2 robot platform product web page, Last accessed in January, 2013 [Online]. http://www.aai.ca/robots/gaia_2.html.
4. Parallax Eddie robot platform product web page, Last accessed in January, 2013 [Online]. <http://www.parallax.com/eddie>
5. RoPro Design Calliope iRobot Create Netbook Development Platform product web page, Last accessed in January, 2013 [Online]. <http://chiara-robot.org/Calliope/>
6. Nexus Robot 4WD Mecanum robot product web page, Last accessed in January, 2013 [Online]. http://www.nexusrobot.com/product.php?id_product=67
7. Dr. Robot Sentinel3 WiFi mobile development platform product web page, Last accessed in January, 2013 [Online]. http://www.drrobot.com/products_item.asp?itemNumber=sentinel3
8. Dr. Robot Jaguar tracked mobile platform product web page, Last accessed in January, 2013 [Online]. http://jaguar.drrobot.com/specification_lite.asp
9. InspectorBots Mega Bot wireless 4WD robot platform product web page, Last accessed in January, 2013 [Online]. http://www.inspectorbots.com/Mega_Bot.html
10. CoroWare CoroBot CB-W Robot Development Platform product user guide, Last accessed in January, 2013 [Online]. <http://www.robotshop.com/PDF/corobot-user-guide.pdf>
11. Seekur Jr robot platform product web page, Last accessed in January, 2013 [Online]. <http://www.mobilerobots.com/ResearchRobots/SeekurJr.aspx>
12. Dongbu Robot Tetra-DS IV Mobile Robot Platform product web page, Last accessed in January, 2013 [Online]. <http://www.dongburobot.com/jsp/cms/view.jsp?code=100792>
13. Wisspeintner, Thomas, and Abheek Bose. "The VolksBot Concept – Rapid Prototyping for Real-life Applications in Mobile Robotics (Das VolksBot Konzept – Ein Baukastensystem Für Anwendungsnahes Rapid-Prototyping Mobiler Roboter)." *It - Information Technology*, vol. 47, pp. 274-281, May 2005.
14. Gerecke, U. ,Hohmann, P. , Wagner, B., "Solutions to Meet the Requirements of Educational Robotics", *International Conference on Engineering Education and Research "Progress Through Partnership"*, VŠB-TUO, Ostrava, 2004.
15. Pololu 37D 100 rpm Motor Product Page, Last accessed in January, 2013 [Online]. <http://www.pololu.com/catalog/product/1446>
16. Arduino Mega Product Page, Last accessed in January, 2013 [Online]. <http://arduino.cc/en/Main/arduinoBoardMega>
17. Rover 5 Motor Driver Board Product Page, Last accessed in January, 2013 [Online]. <https://www.sparkfun.com/products/11593?>

18. HC-SR04 Ultrasonic Sensor Distance Measuring Module Product Page, Last accessed in January, 2013 [Online]. <http://dx.com/p/hc-sr04-ultrasonic-sensor-distance-measuring-module-133696>
19. Microsoft Kinect Product Page, Last accessed in January, 2013 [Online]. <http://www.xbox.com/en-US/KINECT>
20. Genta, G., *Motor Vehicle Dynamics*, World Scientific, 1997.
21. LSM303DLH Tilt Compensated Electronic Compass Application Notes, Last accessed in January, 2013 [Online]. <http://www.sparkfun.com/datasheets/Sensors/Magneto/Tilt%20Compensated%20Compass.pdf>
22. LSM303DLH Tilt Compensated Electronic Compass Product Web Page, Last accessed in January, 2013 [Online]. <https://www.sparkfun.com/products/9810>
23. Dudek, G., Michael, J., *Computational Principles of Mobile Robotics*. New York: Cambridge UP, 2010.
24. Differential Drive Kinematics, Last accessed in January, 2013 [Online]. <http://chess.eecs.berkeley.edu/eecs149/documentation/differentialDrive.pdf>
25. Al-Eryani, J., “A Biologically Inspired Modular Autonomous Mobile Robot Platform”, *Artificial Life and Robotics*, vol. 13, Issue 1, pp 22-26, December 2008.
26. Bayar, G., Koku, A. B., Konukseven, E. İ., “Design of a Configurable All Terrain Mobile Robot Platform”. *International Journal Of Mathematical Models And Methods In Applied Sciences*, vol. 3, Issue 4, 2009.
27. Csaba, G. and Vámosy, Z. “Fuzzy Based Obstacle Avoidance for Mobil Robots with Kinect Sensor”, *4th IEEE International Symposium on Logistics and Industrial Informatics*, 2012.
28. Gerecke, U., Wagner B., Hohmann P., “A Modular Educational Robotic Toolbox to Support University Teaching Efforts in Engineering”, *International Conference on Engineering Education*, 2003.
29. Goris, K., “Autonomous Mobile Robot Mechanical Design”, *Master’s Thesis*, Vrije Universiteit Brussel, 2005.
30. Hacinecipoglu, A., “Development Of Electrical And Control System Of An Unmanned Ground Vehicle For Force Feedback Teleoperation”, *Master’s Thesis*, Middle East Technical University, 2012.
31. Hofbauer, M. and Jantscher, S., “Modular Re-Configurable Robot Drives”, *Robotics Automation and Mechatronics*, 2010.
32. Jeong, H. and Kim, S. “Development of Autonomous Robot System for Indoor Messy Environments”, *ICROS-SICE International Joint Conference*, 2009.
33. Bayar, G., “Configurable Robot Base Design for Mixed Terrain Applications”, *Master’s Thesis*, Middle East Technical University, 2005.
34. Kucsera, P., “Industrial Component-based Sample Mobile Robot System”, *Acta Polytechnica Hungarica*, vol.4, 2007.
35. Kul, M. “Design, Development And Manufacturing Of An All Terrain Modular Robot Platform”, *Master’s Thesis*, Middle East Technical University, 2010.

36. Nalpantidis, L., Gasteratos, A., "Stereovision-Based Fuzzy Obstacle Avoidance Method", *International Journal Of Humanoid Robotics*, vol.8, 2011.
37. Granado Navarro, M. Á., "Arduino Based Acquisition System For Control Applications.", Master's Thesis, Universitat Politècnica De Catalunya, 2012.
38. Rusu, C., Birou, I., "Obstacle Avoidance Fuzzy System For Mobile Robot With Ir Sensors", *10th International Conference On Development And Application Systems*, 2010.
39. Surmann, H., Bredenfeld, A., "The Volksbot", *International Conference On Simulation, Modeling And Programming For Autonomous Robots*, 2008.
40. Winchenbach, S., Segee, D., "Cost-Effective Mobile Robot Platform Using Commercially Available Components", *Journal Of Computer And Information Technology*, vol.1, 2011.

APPENDIX A

SERIAL PORT HANDLING ALGORITHM

```
void getSerial()
{
    int numberOfBytes = spArduino.BytesToRead; //Get number of bytes to read
    if (numberOfBytes == 0) //If number of bytes is 0 return
        return;
    while (numberOfBytes != 0) //While there is bytes to read, loop through
    {
        myRS232_Data.readValue =
            Convert.ToByte(spArduino.ReadByte()); //Read one byte
        numberOfBytes--;

        if (myRS232_Data.packetState == doWaitSTX) //Whether we wait
        {
            //the first byte of a packet
            myRS232_Data.length = 0; //Initialize the length of the data
            if (myRS232_Data.readValue != STX) //If the byte is not a stx continue
            {
                // to get the bytes
                continue;
            }
            myRS232_Data.buff[ADR_STX] = STX; //If STX put this into buffer array
            myRS232_Data.packetState = doWaitData; //So the next bytes are data packet
            myRS232_Data.length = 1; //The data packet length is now 1
            continue;
        }
        else if (myRS232_Data.packetState == doWaitData) //Next byte comes to this
        {
            myRS232_Data.buff[myRS232_Data.length++] =
                myRS232_Data.readValue; //Take byte
            if (myRS232_Data.length < 20) //and put it to buffer array
            {
                //until the buffer array size is 20
                continue;
            }

            if (myRS232_Data.buff[myRS232_Data.length - 2] !=
                calculateBCC(myRS232_Data.buff, 1, myRS232_Data.length - 3))
            {
                // Check for BCC values in the packet
                myRS232_Data.packetState = doWaitSTX; // If it is corrupt packet
                myRS232_Data.length = 0; //discard this packet and wait for the next one
                break;
            }

            RobotResponse(); //Packet is valid. Evaluate the Packet
            break; //Continue if serial port buffer has bytes to read.
        }
        else
        {
            myRS232_Data.packetState = doWaitSTX;
            continue;
        }
    }
}
```

Figure 78 - Serial Port Handling Algorithm

APPENDIX B

SPEED TEST CALCULATION TABLE

Table 106 - Speed Calculation Table

Test Point	Test Time	Speed Calculation
Proximity 1 In	t_0	-
Proximity 2 In	t_1	$0.33/(t_1 - t_0)$
Proximity 1 Out	t_2	$0.50/(t_2 - t_0)$
Proximity 2 Out	t_3	$0.50/(t_3 - t_1)$
Proximity 3 In	t_4	$1.00/(t_4 - t_0)$
Proximity 4 In	t_5	$0.33/(t_5 - t_4)$
Proximity 3 Out	t_6	$0.50/(t_6 - t_4)$
Proximity 4 Out	t_7	$0.50/(t_7 - t_5)$

APPENDIX C

LIBRARY REFERENCE (EDUROB.NAMESPACE)

In this section the library of the robot will be described.

Events

The events are used for new data delegates. All of the events explained below are raised when new data is received. The user can register to an event to monitor the robot properties.

Table 107 - Events

Event Name	Description
GetDirectionData	This event is raised when new direction data is received.
GetSonarData	This event is raised when new sonar data is received.
GetPWMData	This event is raised when new pwm data is received.
GetEncoderData	This event is raised when new encoder data is received.
GetCurrentData	This event is raised when new current data is received.
GetRPMData	This event is raised when new rpm data is received.
GetHeadingData	This event is raised when new heading data is received.
GetAccelerometerData	This event is raised when new accelerometer data is received.
HeartBeat	This event is raised when new heartbeat is received.
MotorEncoderUpdate	This event is raised when new motor encoder update response is received.
MotorRPMUpdate	This event is raised when new motor rpm update response is received.
MotorCurrentUpdate	This event is raised when new motor current update response is received.
SonarUpdate	This event is raised when new sonar update response is received.
HeadingUpdate	This event is raised when new heading update response is received.
AccelerometerUpdate	This event is raised when new accelerometer update response is received.

Methods

The methods are explained below.

Table 108 - Methods

Method Name	Description
SetHeartBeat(Int16 heartBeat)	This function sets the heartbeat interval for the robot. The function takes interval as milliseconds. Default heartbeat interval is 2000 ms.
SerialConnect(string serialPortName)	This function opens a connection to robot with the given port name. Returns the value indicating the open or closed value of the serial port.
CheckSerialConnection()	This function returns the value indicating the open or closed value of the serial port.
SerialDisconnect()	This function closes the connection of the robot. Returns the value indicating the open or closed value of the serial port.
SetDirection(int directionLF, int	This function sets the direction of the

<code>directionLB, int directionRF, int directionRB)</code>	individual motors in the robot platform.
<code>SetRPM(int rpmMotorLF, int rpmMotorLB, int rpmMotorRF, int rpmMotorRB)</code>	This function sets the rpm of the individual motors in the robot platform.
<code>GetSonar()</code>	This function sends a command to robot platform for getting the ultrasonic sensor data.
<code>GetEncoder()</code>	This function sends a command to robot platform for getting the encoder data.
<code>GetMotorDirection()</code>	This function sends a command to robot platform for getting the motor direction values.
<code>GetHeading()</code>	This function sends a command to robot platform for getting the heading data.
<code>GetAccelerometer()</code>	This function sends a command to robot platform for getting the accelerometer data.
<code>SetPWM(int pwmMotorLF, int pwmMotorLB, int pwmMotorRF, int pwmMotorRB)</code>	This function sets the pwm of the individual motors in the robot platform.
<code>GetRPM()</code>	This function sends a command to robot platform for getting the rpm data.
<code>GetPWM()</code>	This function sends a command to robot platform for getting the pwm data.
<code>GetCurrent()</code>	This function sends a command to robot platform for getting the current data.
<code>SetRPS(double rpsLF, double rpsLB, double rpsRF, double rpsRB)</code>	This function sets the rps of the individual motors in the robot platform.
<code>SetSpeed(double speed)</code>	This function sets the speed of the robot platform.
<code>SetDistance(double distance, double speed)</code>	This function drives the robot platform to specified distance with the specified speed.
<code>TurnAngle(int angle, RotationDirection direction)</code>	This function rotates the robot platform with specified angle and the specified direction.
<code>TurnToAngle(int angle, RotationDirection direction)</code>	This function rotates the robot platform to the specified angle with the specified direction.
<code>ResetEncoder()</code>	This function resets the encoder ticks of the motors.
<code>SetUpdateParameterInterval(UpdateParameter parameter, UpdateResponse response, Int16 UpdateInterval)</code>	This function sets which parameter to be updated. UpdateInterval is in milliseconds.

Enumerations and Constants

Enumerations and constants are given below.

```
public enum UpdateResponse : int
{
    YES = 1,
    NO = 0
}
```

```

public enum UpdateParameter : int
{
    Sonar,
    Heading,
    Accelerometer,
    MotorEncoder,
    MotorRPM,
    MotorCurrent
}

public enum RotationDirection : int
{
    Right,
    Left
}

public static class MotorDirection
{
    public const int Forward = 'I';
    public const int Backward = 'G';
}

```

Example Usage of the Library

First determine the serial port to connect to the robot. This can be through USB cable to Arduino or a wireless serial communication can also be used by using XBEE modules. In the code first add the EduRob.dll to the project. Then, insert the below code to the using statements of the project.

```
using EduRob;
```

This will allow the user to create and use the EduRob Library.

Create a new instance of the library.

```
EduRobot myRobot = new EduRobot();
```

Then first thing is to connect to the robot.

```
myRobot.SerialConnect("COM1");
```

This function will also start the heartbeat of the robot. Then the user should specify the update intervals of the sensors that are going to be used. Note that if no sensor update interval is specified the robot will not broadcast any sensor data. The robot will only broadcast if the user sends the command to the robot.

```
myRobot.SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.YES, 100);
```

```
myRobot.SetUpdateParameterInterval(EduRobot.UpdateParameter.Sonar,
EduRobot.UpdateResponse.YES, 200);
```

```
myRobot.SetUpdateParameterInterval(EduRobot.UpdateParameter.MotorEncoder,
EduRobot.UpdateResponse.YES, 100);
```

```
myRobot.SetUpdateParameterInterval(EduRobot.UpdateParameter.MotorCurrent,
EduRobot.UpdateResponse.YES, 100);
```

In this code, heading sensor, ultrasonic sensors, motor encoders and motor currents are demanded from the robot with the specified intervals.

Then, the user should register to the related events to get updated when new data comes.

```
myRobot.GetHeadingData += new EduRobot.delGetHeading(myRobot_GetHeadingData);
myRobot.GetSonarData += new EduRobot.delGetSonar(myRobot_GetSonarData);
myRobot.GetEncoderData += new EduRobot.delGetEncoder(myRobot_GetEncoderData);
myRobot.GetCurrentData += new EduRobot.delGetCurrent(myRobot_GetCurrentData);
```

The events are registered. Then the methods for the events should be implemented. In the below functions the new data from robot is written to the related global variables and they are ready for later use.

```
void myRobot_GetHeadingData(int heading)
{
    this.heading = heading;
    headingDataRecieved = true;
}
```

```
void myRobot_GetSonarData(int sonarFrontLeftInCm, int sonarFrontMiddleInCm,
int sonarFrontRightInCm, int sonarRightFrontInCm, int sonarRightBackInCm, int
sonarBackRightInCm, int sonarBackMiddleInCm, int sonarBackLeftInCm, int
sonarLeftBackInCm, int sonarLeftFrontInCm)
{
    this.sonarFrontLeftInCm = sonarFrontLeftInCm;
    this.sonarFrontMiddleInCm = sonarFrontMiddleInCm;
    this.sonarFrontRightInCm = sonarFrontRightInCm;
    this.sonarRightFrontInCm = sonarRightFrontInCm;
    this.sonarRightBackInCm = sonarRightBackInCm;
    this.sonarBackRightInCm = sonarBackRightInCm;
    this.sonarBackMiddleInCm = sonarBackMiddleInCm;
    this.sonarBackLeftInCm = sonarBackLeftInCm;
    this.sonarLeftBackInCm = sonarLeftBackInCm;
    this.sonarLeftFrontInCm = sonarLeftFrontInCm;
    sonarDataRecieved = true;
}
```

```
void myRobot_GetEncoderData(int positionForEncoderLF, int
positionForEncoderLB, int positionForEncoderRF, int positionForEncoderRB)
{
    this.positionForEncoderLF = positionForEncoderLF;
    this.positionForEncoderLB = positionForEncoderLB;
    this.positionForEncoderRF = positionForEncoderRF;
    this.positionForEncoderRB = positionForEncoderRB;
    encoderDataRecieved = true;
}
```

```
void myRobot_GetCurrentData(int currentMotorLF, int currentMotorLB, int
currentMotorRF, int currentMotorRB)
{
    this.currentMotorLF = currentMotorLF;
    this.currentMotorLB = currentMotorLB;
    this.currentMotorRF = currentMotorRF;
    this.currentMotorRB = currentMotorRB;
    currentDataRecieved = true;
}
```

Now we have global variables for the specified sensors. We can use this data in an algorithm and use the high level functions. The example for this is given below. But first the example usages of the functions are given.

```
myRobot.SetDirection(EduRobot.MotorDirection.Forward,
EduRobot.MotorDirection.Forward, EduRobot.MotorDirection.Forward,
EduRobot.MotorDirection.Forward);
myRobot.SetRPM(30, 30, 30, 30);
```

First the directions of the motors are set and then rpm values are given for the motors.

```
myRobot.GetSonar();
myRobot.GetEncoder();
myRobot.GetMotorDirection();
myRobot.GetHeading();
myRobot.GetAccelerometer();
```

These methods are used for getting the data for just one time. To get the data in an interval (broadcasting mode) the interval for the specified sensor should be set just like mentioned in the previous page.

```
myRobot.SetSpeed(0.2);
```

This line sets the speed of the robot 0.2 m/s.

```
myRobot.SetDistance(1, 0.2);
```

This line moves the robot 1 m with a speed of 0.2 m/s.

```
myRobot.TurnAngle(90, EduRobot.RotationDirection.Right);
```

This line turns the robot in the right direction (Clockwise) by 90 degrees.

```
myRobot.TurnToAngle(20, EduRobot.RotationDirection.Right);
```

This line turns the robot to 20 degrees relative to Magnetic North in the right direction (Clockwise).

```
myRobot.ResetEncoder();
```

This line will reset the encoder variable in the microcontroller side.

The below lines of code is a simple wandering algorithm. The robot will move forward if there is no obstacle in front. If an obstacle in the front, the robot will first look right, turn if right is empty. If not, the robot will turn left, if left is empty. If all the ways are closed, the robot will move backwards.

```
if (depthTotalCenter > forwardThreshold)
{
    if (depthTotalRight < sideThreshold)
        myRobot.TurnAngle(10, EduRobot.RotationDirection.Left);
    else if (depthTotalLeft < sideThreshold)
        myRobot.TurnAngle(10, EduRobot.RotationDirection.Right);
    else
        myRobot.SetSpeed(0.2);
}
else if (depthTotalRight > forwardThreshold)
{
    myRobot.TurnAngle(10, EduRobot.RotationDirection.Right);}
else if (depthTotalLeft > forwardThreshold)
{
    myRobot.TurnAngle(10, EduRobot.RotationDirection.Left);
}
else
{
```

```

        myRobot.SetDirection(EduRobot.MotorDirection.Backward,
EduRobot.MotorDirection. Backward, EduRobot.MotorDirection. Backward,
EduRobot.MotorDirection. Backward);
        myRobot.SetSpeed(0.2);
    }

```

Library Code

The whole code of the library is given below. The code includes the serial port algorithm and private functions sets to handle the high level functions.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.Timers;

namespace EduRob
{
    public partial class EduRobot
    {
        public delegate void delGetDirection(int motorDirectionLF, int motorDirectionLB, int
motorDirectionRF, int motorDirectionRB);
        /// <summary>
        /// This event is raised when new direction data is recieved.
        /// </summary>
        public event delGetDirection GetDirectionData;

        public delegate void delGetSonar(int sonarFrontLeftInCm, int sonarFrontMiddleInCm,
int sonarFrontRightInCm, int sonarRightFrontInCm,
int sonarRightBackInCm, int sonarBackRightInCm, int
sonarBackMiddleInCm, int sonarBackLeftInCm, int sonarLeftBackInCm, int sonarLeftFrontInCm);
        /// <summary>
        /// This event is raised when new sonar data is recieved.
        /// </summary>
        public event delGetSonar GetSonarData;

        public delegate void delGetPWM(int pwmLF, int pwmLB, int pwmRF, int pwmRB);
        /// <summary>
        /// This event is raised when new pwm data is recieved.
        /// </summary>
        public event delGetPWM GetPWMData;

        public delegate void delGetEncoder(int positionForEncoderLF, int
positionForEncoderLB, int positionForEncoderRF, int positionForEncoderRB);
        /// <summary>
        /// This event is raised when new encoder data is recieved.
        /// </summary>
        public event delGetEncoder GetEncoderData;

        public delegate void delGetCurrent(int currentMotorLF, int currentMotorLB, int
currentMotorRF, int currentMotorRB);
        /// <summary>
        /// This event is raised when new current data is recieved.
        /// </summary>
        public event delGetCurrent GetCurrentData;

        public delegate void delGetRPM(int rpmLF, int rpmLB, int rpmRF, int rpmRB);
        /// <summary>
        /// This event is raised when new rpm data is recieved.
        /// </summary>
        public event delGetRPM GetRPMData;

        public delegate void delGetHeading(int heading);
        /// <summary>

```

```

    /// This event is raised when new heading data is recieved.
    /// </summary>
    public event delGetHeading GetHeadingData;

    public delegate void delGetAccelerometer(float accelerometerX, float accelerometerY,
float accelerometerZ);
    /// <summary>
    /// This event is raised when new accelerometer data is recieved.
    /// </summary>
    public event delGetAccelerometer GetAccelerometerData;

    public delegate void delHeartBeat(bool heartBeat);
    /// <summary>
    /// This event is raised when new heartbeat is recieved.
    /// </summary>
    public event delHeartBeat HeartBeat;

    public delegate void delMotorEncoderUpdate(bool motorEncoderUpdate);
    /// <summary>
    /// This event is raised when new motor encoder update response is recieved.
    /// </summary>
    public event delMotorEncoderUpdate MotorEncoderUpdate;

    public delegate void delMotorRPMUpdate(bool motorRPMUpdate);
    /// <summary>
    /// This event is raised when new motor rpm update response is recieved.
    /// </summary>
    public event delMotorRPMUpdate MotorRPMUpdate;

    public delegate void delMotorCurrentUpdate(bool motorCurrentUpdate);
    /// <summary>
    /// This event is raised when new motor current update response is recieved.
    /// </summary>
    public event delMotorCurrentUpdate MotorCurrentUpdate;

    public delegate void delSonarUpdate(bool sonarUpdate);
    /// <summary>
    /// This event is raised when new sonar update response is recieved.
    /// </summary>
    public event delSonarUpdate SonarUpdate;

    public delegate void delHeadingUpdate(bool headingUpdate);
    /// <summary>
    /// This event is raised when new heading update response is recieved.
    /// </summary>
    public event delHeadingUpdate HeadingUpdate;

    public delegate void delAccelerometerUpdate(bool accelerometerUpdate);
    /// <summary>
    /// This event is raised when new accelerometer update response is recieved.
    /// </summary>
    public event delAccelerometerUpdate AccelerometerUpdate;

    private SerialPort spArduino;

    private int motorDirectionLF;
    private int motorDirectionLB;
    private int motorDirectionRF;
    private int motorDirectionRB;

    private int responseMotorDirectionLF;
    private int responseMotorDirectionLB;
    private int responseMotorDirectionRF;
    private int responseMotorDirectionRB;

    private int pwmLF = 0;
    private int pwmLB = 0;
    private int pwmRF = 0;
    private int pwmRB = 0;

    private int responsePwmLF = 0;

```

```

private int responsePwmLB = 0;
private int responsePwmRF = 0;
private int responsePwmRB = 0;

private int positionForEncoderLF;
private int positionForEncoderLB;
private int positionForEncoderRF;
private int positionForEncoderRB;

private int currentMotorLF;
private int currentMotorLB;
private int currentMotorRF;
private int currentMotorRB;

private int sonarFrontLeftInCm;
private int sonarFrontMiddleInCm;
private int sonarFrontRightInCm;
private int sonarRightFrontInCm;
private int sonarRightBackInCm;
private int sonarBackRightInCm;
private int sonarBackMiddleInCm;
private int sonarBackLeftInCm;
private int sonarLeftBackInCm;
private int sonarLeftFrontInCm;

private int rpmLF = 0;
private int rpmLB = 0;
private int rpmRF = 0;
private int rpmRB = 0;

private int responseRpmLF = 0;
private int responseRpmLB = 0;
private int responseRpmRF = 0;
private int responseRpmRB = 0;

private int heading;

private float accelerometerX;
private float accelerometerY;
private float accelerometerZ;

struct myRS232_Data
{
    public static byte[] buff = new byte[32];
    public static int length;
    public static byte readValue;
    public static byte packetState;
};

public static byte[] sendBuffer = new byte[32];

//For SetDistance Method
private DateTime testTime;
private double timeSeconds;
private Timer timerDistance;

private int distanceStartEncoderLF;
private int distanceStartEncoderLB;
private int distanceStartEncoderRF;
private int distanceStartEncoderRB;

private int distanceEncoderTicks;

//For TurnAngle Method
private Timer timerRotation;
private int angleToTurn;
private int angleResolution = 2;
private bool robotStartedHeadingUpdate = false;

private Timer timerHeartBeat;

```



```

private int sendSonar = 0;
private int sendEncoder = 0;
private int sendRPM = 0;
private int sendHeading = 0;
private int sendAccelerometer = 0;
private int sendCurrent = 0;
private Int16 motorEncoderUpdateInterval = 0;
private Int16 motorRPMUpdateInterval = 0;
private Int16 motorCurrentUpdateInterval = 0;
private Int16 sonarUpdateInterval = 0;
private Int16 headingUpdateInterval = 0;
private Int16 accelerometerUpdateInterval = 0;

private Timer timerTurnToAngle;
private int firstAngleTurn;
private Int16 heartBeat = 2000;

/// <summary>
/// This function opens a connection to robot with the given port name.
/// Returns the value indicating the open or closed value of the serial port.
/// </summary>
public bool SerialConnect(string serialPortName)
{
    spArduino = new SerialPort();
    spArduino.DataBits = 8;
    spArduino.DiscardNull = false;
    spArduino.DtrEnable = false;
    spArduino.Handshake = Handshake.None;
    spArduino.Parity = Parity.None;
    spArduino.ReadTimeout = -1;
    spArduino.ReceivedBytesThreshold = 1;
    spArduino.RtsEnable = false;
    spArduino.StopBits = StopBits.One;

    spArduino.PortName = serialPortName;
    spArduino.BaudRate = 115200;

    if (spArduino.IsOpen == false)
        spArduino.Open();

    spArduino.DataReceived += new
SerialDataReceivedEventHandler(spArduino_DataReceived);

    timerHeartBeat = new Timer();
    timerHeartBeat.Interval = heartBeat;
    timerHeartBeat.Elapsed += new ElapsedEventHandler(timerHeartBeat_Elapsed);
    timerHeartBeat.Enabled = true;

    InitializeParameters();

    return spArduino.IsOpen;
}

/// <summary>
/// This function sets the heartbeat interval for the robot. The function takes
interval as milliseconds
/// Default heartbeat interval is 2000 ms.
/// </summary>
public void SetHeartBeat(Int16 heartBeat)
{
    this.heartBeat = heartBeat;
    SendCMD(Commands.SETHEARTBEATINTERVAL);
}

void timerHeartBeat_Elapsed(object sender, ElapsedEventArgs e)
{
    SendCMD(Commands.HEARTBEAT);
}

void spArduino_DataReceived(object sender, SerialDataReceivedEventArgs e)
{

```

```

    getSerial();
}

private void getSerial()
{
    int numberOfBytes = spArduino.BytesToRead; //Get number of bytes to read
    if (numberOfBytes == 0) //If number of bytes is 0 return
        return;
    while (numberOfBytes != 0) //While there is bytes to read, loop through
    {
        myRS232_Data.readValue =
            Convert.ToByte(spArduino.ReadByte()); //Read one byte
        numberOfBytes--;

        if (myRS232_Data.packetState == SerialRecieveConstants.doWaitSTX) //Whether
we wait
        {
            //the first byte of a packet
            myRS232_Data.length = 0; //Initialize the length of the data
            if (myRS232_Data.readValue != ByteConstants.STX) //If the byte is not a
stx continue
            {
                // to get the bytes
                continue;
            }
            myRS232_Data.buff[SerialAddressConstants.STX] = ByteConstants.STX; //If
STX put this into buffer array
            myRS232_Data.packetState = SerialRecieveConstants.doWaitDataLength; //So
the next bytes are data packet
            myRS232_Data.length = 1; //The data packet length is now 1
            continue;
        }
        else if (myRS232_Data.packetState == SerialRecieveConstants.doWaitDataLength)
        {
            myRS232_Data.buff[myRS232_Data.length++] = myRS232_Data.readValue; //Take
command byte
            myRS232_Data.packetState = SerialRecieveConstants.doWaitData;
            continue;
        }
        else if (myRS232_Data.packetState == SerialRecieveConstants.doWaitData)
//Next byte comes to this
        {
            myRS232_Data.buff[myRS232_Data.length++] = myRS232_Data.readValue;
//Take data length byte

            if (myRS232_Data.length < (myRS232_Data.buff[2] + 5)) //and put
it to buffer array
            {
                //until the buffer array size is 20
                continue;
            }

            if (myRS232_Data.buff[myRS232_Data.length - 2] !=
                calculateBCC(myRS232_Data.buff, 1, myRS232_Data.length - 3))
            {
                // Check for BCC values in the packet
                myRS232_Data.packetState = SerialRecieveConstants.doWaitSTX; // If
it is corrupt packet
                myRS232_Data.length = 0; //discard this packet and wait for the next
one
                break;
            }

            RobotResponse(); //Packet is valid. Evaluate the Packet
            break; //Continue if serial port buffer has bytes to read.
        }
        else
        {
            myRS232_Data.packetState = SerialRecieveConstants.doWaitSTX;
            continue;
        }
    }
}
}

```

```

private byte calculateBCC(byte[] buffer, int startIndex, int stopIndex)
{
    byte tempByte;
    int tempByteIndex;
    tempByte = buffer[startIndex];
    for (tempByteIndex = startIndex + 1; tempByteIndex <= stopIndex; tempByteIndex++)
    {
        tempByte ^= buffer[tempByteIndex];
    }
    return tempByte;
}

private void RobotResponse()
{
    switch (myRS232_Data.buff[SerialAddressConstants.COMD])
    {
        //----- // ASCII
        case 0x00:
            break;
        //-----
        case Commands.GETDIRECTION:
            responseMotorDirectionLF =
myRS232_Data.buff[SerialAddressConstants.DATA];
            responseMotorDirectionLB = myRS232_Data.buff[SerialAddressConstants.DATA
+ 1];
            responseMotorDirectionRF = myRS232_Data.buff[SerialAddressConstants.DATA
+ 2];
            responseMotorDirectionRB = myRS232_Data.buff[SerialAddressConstants.DATA
+ 3];

            if (GetDirectionData != null)
            {
                GetDirectionData(responseMotorDirectionLF, responseMotorDirectionLB,
responseMotorDirectionRF, responseMotorDirectionRB);
            }

            break;
        //-----
        case Commands.GETSONARDATA:

            sonarFrontLeftInCm = 0;
            sonarFrontLeftInCm = myRS232_Data.buff[SerialAddressConstants.DATA];
            sonarFrontLeftInCm = (sonarFrontLeftInCm << 8);
            sonarFrontLeftInCm = sonarFrontLeftInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 1];

            sonarFrontMiddleInCm = 0;
            sonarFrontMiddleInCm = myRS232_Data.buff[SerialAddressConstants.DATA +
2];
            sonarFrontMiddleInCm = (sonarFrontMiddleInCm << 8);
            sonarFrontMiddleInCm = sonarFrontMiddleInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 3];

            sonarFrontRightInCm = 0;
            sonarFrontRightInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 4];
            sonarFrontRightInCm = (sonarFrontRightInCm << 8);
            sonarFrontRightInCm = sonarFrontRightInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 5];

            sonarRightFrontInCm = 0;
            sonarRightFrontInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 6];
            sonarRightFrontInCm = (sonarRightFrontInCm << 8);
            sonarRightFrontInCm = sonarRightFrontInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 7];

            sonarRightBackInCm = 0;
            sonarRightBackInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 8];
            sonarRightBackInCm = (sonarRightBackInCm << 8);
            sonarRightBackInCm = sonarRightBackInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 9];
    }
}

```

```

        sonarBackRightInCm = 0;
        sonarBackRightInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 10];
        sonarBackRightInCm = (sonarBackRightInCm << 8);
        sonarBackRightInCm = sonarBackRightInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 11];

        sonarBackMiddleInCm = 0;
        sonarBackMiddleInCm = myRS232_Data.buff[SerialAddressConstants.DATA +
12];

        sonarBackMiddleInCm = (sonarBackMiddleInCm << 8);
        sonarBackMiddleInCm = sonarBackMiddleInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 13];

        sonarBackLeftInCm = 0;
        sonarBackLeftInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 14];
        sonarBackLeftInCm = (sonarBackLeftInCm << 8);
        sonarBackLeftInCm = sonarBackLeftInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 15];

        sonarLeftBackInCm = 0;
        sonarLeftBackInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 16];
        sonarLeftBackInCm = (sonarLeftBackInCm << 8);
        sonarLeftBackInCm = sonarLeftBackInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 17];

        sonarLeftFrontInCm = 0;
        sonarLeftFrontInCm = myRS232_Data.buff[SerialAddressConstants.DATA + 18];
        sonarLeftFrontInCm = (sonarLeftFrontInCm << 8);
        sonarLeftFrontInCm = sonarLeftFrontInCm |
myRS232_Data.buff[SerialAddressConstants.DATA + 19];

        if (GetSonarData != null)
        {
            GetSonarData(sonarFrontLeftInCm, sonarFrontMiddleInCm,
sonarFrontRightInCm, sonarRightFrontInCm, sonarRightBackInCm, sonarBackRightInCm,
sonarBackMiddleInCm, sonarBackLeftInCm, sonarLeftBackInCm, sonarLeftFrontInCm);
        }

        break;
//-----
case Commands.GETPWM:

        responsePwmLF = myRS232_Data.buff[SerialAddressConstants.DATA];
        responsePwmLB = myRS232_Data.buff[SerialAddressConstants.DATA + 1];
        responsePwmRF = myRS232_Data.buff[SerialAddressConstants.DATA + 2];
        responsePwmRB = myRS232_Data.buff[SerialAddressConstants.DATA + 3];

        if (GetPWMDData != null)
        {
            GetPWMDData(responsePwmLF, responsePwmLB, responsePwmRF,
responsePwmRB);
        }

        break;
//-----
case Commands.GETENCODER:

        positionForEncoderLF = 0;
        positionForEncoderLF = myRS232_Data.buff[SerialAddressConstants.DATA];
        positionForEncoderLF = (positionForEncoderLF << 8);
        positionForEncoderLF = positionForEncoderLF |
myRS232_Data.buff[SerialAddressConstants.DATA + 1];
        positionForEncoderLF = (positionForEncoderLF << 8);
        positionForEncoderLF = positionForEncoderLF |
myRS232_Data.buff[SerialAddressConstants.DATA + 2];
        positionForEncoderLF = (positionForEncoderLF << 8);
        positionForEncoderLF = positionForEncoderLF |
myRS232_Data.buff[SerialAddressConstants.DATA + 3];

        positionForEncoderLB = 0;

```

```

4];
    positionForEncoderLB = myRS232_Data.buff[SerialAddressConstants.DATA +
    positionForEncoderLB = (positionForEncoderLB << 8);
    positionForEncoderLB = positionForEncoderLB |
myRS232_Data.buff[SerialAddressConstants.DATA + 5];
    positionForEncoderLB = (positionForEncoderLB << 8);
    positionForEncoderLB = positionForEncoderLB |
myRS232_Data.buff[SerialAddressConstants.DATA + 6];
    positionForEncoderLB = (positionForEncoderLB << 8);
    positionForEncoderLB = positionForEncoderLB |
myRS232_Data.buff[SerialAddressConstants.DATA + 7];

    positionForEncoderRF = 0;
    positionForEncoderRF = myRS232_Data.buff[SerialAddressConstants.DATA +
8];
    positionForEncoderRF = (positionForEncoderRF << 8);
    positionForEncoderRF = positionForEncoderRF |
myRS232_Data.buff[SerialAddressConstants.DATA + 9];
    positionForEncoderRF = (positionForEncoderRF << 8);
    positionForEncoderRF = positionForEncoderRF |
myRS232_Data.buff[SerialAddressConstants.DATA + 10];
    positionForEncoderRF = (positionForEncoderRF << 8);
    positionForEncoderRF = positionForEncoderRF |
myRS232_Data.buff[SerialAddressConstants.DATA + 11];

    positionForEncoderRB = 0;
    positionForEncoderRB = myRS232_Data.buff[SerialAddressConstants.DATA +
12];
    positionForEncoderRB = (positionForEncoderRB << 8);
    positionForEncoderRB = positionForEncoderRB |
myRS232_Data.buff[SerialAddressConstants.DATA + 13];
    positionForEncoderRB = (positionForEncoderRB << 8);
    positionForEncoderRB = positionForEncoderRB |
myRS232_Data.buff[SerialAddressConstants.DATA + 14];
    positionForEncoderRB = (positionForEncoderRB << 8);
    positionForEncoderRB = positionForEncoderRB |
myRS232_Data.buff[SerialAddressConstants.DATA + 15];

    if (GetEncoderData != null)
    {
        GetEncoderData(positionForEncoderLF, positionForEncoderLB,
positionForEncoderRF, positionForEncoderRB);
    }

    break;
//-----
case Commands.GETCURRENT:

    currentMotorLF = 0;
    currentMotorLF = myRS232_Data.buff[SerialAddressConstants.DATA];
    currentMotorLF = (currentMotorLF << 8);
    currentMotorLF = currentMotorLF |
myRS232_Data.buff[SerialAddressConstants.DATA + 1];

    currentMotorLB = 0;
    currentMotorLB = myRS232_Data.buff[SerialAddressConstants.DATA + 2];
    currentMotorLB = (currentMotorLB << 8);
    currentMotorLB = currentMotorLB |
myRS232_Data.buff[SerialAddressConstants.DATA + 3];

    currentMotorRF = 0;
    currentMotorRF = myRS232_Data.buff[SerialAddressConstants.DATA + 4];
    currentMotorRF = (currentMotorRF << 8);
    currentMotorRF = currentMotorRF |
myRS232_Data.buff[SerialAddressConstants.DATA + 5];

    currentMotorRB = 0;
    currentMotorRB = myRS232_Data.buff[SerialAddressConstants.DATA + 6];
    currentMotorRB = (currentMotorRB << 8);
    currentMotorRB = currentMotorRB |
myRS232_Data.buff[SerialAddressConstants.DATA + 7];

```

```

        if (GetCurrentData != null)
        {
            GetCurrentData(currentMotorLF, currentMotorLB, currentMotorRF,
currentMotorRB);
        }

        break;
//-----
case Commands.GETRPM:

    responseRpmLF = myRS232_Data.buff[SerialAddressConstants.DATA];
    responseRpmLB = myRS232_Data.buff[SerialAddressConstants.DATA + 1];
    responseRpmRF = myRS232_Data.buff[SerialAddressConstants.DATA + 2];
    responseRpmRB = myRS232_Data.buff[SerialAddressConstants.DATA + 3];

    if (GetRPMDData != null)
    {
        GetRPMDData(responseRpmLF, responseRpmLB, responseRpmRF,
responseRpmRB);
    }

    break;
//-----
case Commands.GETHEADING:

    //heading = System.BitConverter.ToSingle(myRS232_Data.buff,
SerialAddressConstants.DATA);
    heading = 0;
    heading = myRS232_Data.buff[SerialAddressConstants.DATA];
    heading = (heading << 8);
    heading = heading | myRS232_Data.buff[SerialAddressConstants.DATA + 1];

    if (GetHeadingData != null)
    {
        GetHeadingData(heading);
    }

    break;
//-----
case Commands.GETACCELEROMETER:

    accelerometerX = System.BitConverter.ToSingle(myRS232_Data.buff,
SerialAddressConstants.DATA);

    accelerometerY = System.BitConverter.ToSingle(myRS232_Data.buff,
SerialAddressConstants.DATA + 4);

    accelerometerZ = System.BitConverter.ToSingle(myRS232_Data.buff,
SerialAddressConstants.DATA + 8);

    if (GetAccelerometerData != null)
    {
        GetAccelerometerData(accelerometerX, accelerometerY, accelerometerZ);
    }

    break;
//-----
case Commands.HEARTBEAT:

    if (HeartBeat != null)
    {
        HeartBeat(true);
    }

    break;
//-----
case Commands.MOTORCURRENTUPDATE:

    if (MotorCurrentUpdate != null)
    {

```

```

        MotorCurrentUpdate(true);
    }

    break;
//-----
case Commands.MOTORENCODERUPDATE:

    if (MotorEncoderUpdate != null)
    {
        MotorEncoderUpdate(true);
    }

    break;
//-----
case Commands.MOTORRPMUPDATE:

    if (MotorRPMUpdate != null)
    {
        MotorRPMUpdate(true);
    }

    break;
//-----
case Commands.SONARUPDATE:

    if (SonarUpdate != null)
    {
        SonarUpdate(true);
    }

    break;
//-----
case Commands.HEADINGUPDATE:

    if (HeadingUpdate != null)
    {
        HeadingUpdate(true);
    }

    break;
//-----
case Commands.ACCELEROMETERUPDATE:

    if (AccelerometerUpdate != null)
    {
        AccelerometerUpdate(true);
    }

    break;
//-----
default:
    break;
}

myRS232_Data.packetState = SerialRecieveConstants.doWaitSTX;
}

/// <summary>
/// This function returns the value indicating the open or closed value of the serial
port.
/// </summary>
public bool CheckSerialConnection()
{
    return spArduino.IsOpen;
}

/// <summary>
/// This function closes the connection of the robot.
/// Returns the value indicating the open or closed value of the serial port.
/// </summary>
public bool SerialDisconnect()

```

```

    {
        timerHeartBeat.Enabled = false;

        spArduino.DataReceived -= new
System.IO.Ports.SerialDataReceivedEventHandler(spArduino_DataReceived);
        spArduino.ReadExisting();

        if (spArduino.IsOpen == true)
            spArduino.Close();

        return !spArduino.IsOpen;
    }

    /// <summary>
    /// This function initializes the parameters for serial port, and motor controller.
    /// </summary>
    private void InitializeParameters()
    {
        myRS232_Data.packetState = SerialRecieveConstants.doWaitSTX;

        pwmLF = 0;
        pwmLB = 0;
        pwmRF = 0;
        pwmRB = 0;

        motorDirectionLF = MotorDirection.Forward;
        motorDirectionLB = MotorDirection.Forward;
        motorDirectionRF = MotorDirection.Forward;
        motorDirectionRB = MotorDirection.Forward;
    }

    private void SendCMD(byte myCmd)
    {
        Array.Clear(sendBuffer, 0, sendBuffer.Length);

        int ADR_BCC = 0;
        int ADR_ETX = 0;

        sendBuffer[SerialAddressConstants.STX] = ByteConstants.STX;

        switch (myCmd)
        {
            case Commands.SETDIRECTION:
                sendBuffer[SerialAddressConstants.CMD] = myCmd;

                sendBuffer[SerialAddressConstants.DL] = DataLengths.SETDIRECTION;
                ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
                ADR_ETX = ADR_BCC + 1;
                sendBuffer[ADR_ETX] = ByteConstants.ETX;

                sendBuffer[SerialAddressConstants.DATA] =
Convert.ToByte(motorDirectionLF); //Left Front

                sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(motorDirectionLB); //Left Back

                sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(motorDirectionRF); //Right Front

                sendBuffer[SerialAddressConstants.DATA + 3] =
Convert.ToByte(motorDirectionRB); //Right Back

                sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

                spArduino.Write(sendBuffer, 0, 9);
                break;
            case Commands.GETSONARDATA:
                sendBuffer[SerialAddressConstants.CMD] = myCmd;

```



```

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETSONARDATA;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.SETPWM:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.SETPWM;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(pwmLF); //Left
Front

        sendBuffer[SerialAddressConstants.DATA + 1] = Convert.ToByte(pwmLB);
//Left Back

        sendBuffer[SerialAddressConstants.DATA + 2] = Convert.ToByte(pwmRF);
//Right Front

        sendBuffer[SerialAddressConstants.DATA + 3] = Convert.ToByte(pwmRB);
//Right Back

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 9);

        break;
    case Commands.GETENCODER:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETENCODER;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.GETDIRECTION:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETDIRECTION;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.GETPWM:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETPWM;

```

```

        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.GETCURRENT:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETCURRENT;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.EMERGENCYSTOP:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.EMERGENCYSTOP;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.RESETENCODER:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.RESETENCODER;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.SETRPM:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.SETRPM;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(rpmLF); //Left
Front

        sendBuffer[SerialAddressConstants.DATA + 1] = Convert.ToByte(rpmLB);
//Left Back

        sendBuffer[SerialAddressConstants.DATA + 2] = Convert.ToByte(rpmRF);
//Right Front

```

```

        sendBuffer[SerialAddressConstants.DATA + 3] = Convert.ToByte(rpmRB);
//Right Back
        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 9);
        break;
    case Commands.GETRPM:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETRPM;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.GETHEADING:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETHEADING;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.GETACCELEROMETER:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.GETACCELEROMETER;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.HEARTBEAT:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.HEARTBEAT;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 5);
        break;
    case Commands.MOTORCURRENTUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.MOTORCURRENTUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;

```

```

        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(sendCurrent);

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(motorCurrentUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(motorCurrentUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.MOTORENCODERUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.MOTORENCODERUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(sendEncoder);

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(motorEncoderUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(motorEncoderUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.MOTORRPMUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.MOTORRPMUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(sendRPM);

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(motorRPMUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(motorRPMUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.SONARUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.SONARUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(sendSonar);

```

```

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(sonarUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(sonarUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.HEADINGUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.HEADINGUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(sendHeading);

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(headingUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(headingUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.ACCELEROMETERUPDATE:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.ACCELEROMETERUPDATE;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] =
Convert.ToByte(sendAccelerometer);

        sendBuffer[SerialAddressConstants.DATA + 1] =
Convert.ToByte(accelerometerUpdateInterval>>8);

        sendBuffer[SerialAddressConstants.DATA + 2] =
Convert.ToByte(accelerometerUpdateInterval);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

        spArduino.Write(sendBuffer, 0, 8);
        break;
    case Commands.SETHEARTBEATINTERVAL:
        sendBuffer[SerialAddressConstants.CMD] = myCmd;

        sendBuffer[SerialAddressConstants.DL] = DataLengths.SETHEARTBEATINTERVAL;
        ADR_BCC = SerialAddressConstants.DL +
sendBuffer[SerialAddressConstants.DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        sendBuffer[ADR_ETX] = ByteConstants.ETX;

        sendBuffer[SerialAddressConstants.DATA] = Convert.ToByte(heartBeat >> 8);

        sendBuffer[SerialAddressConstants.DATA + 1] = Convert.ToByte(heartBeat);

        sendBuffer[ADR_BCC] = calculateBCC(sendBuffer, 1,
sendBuffer[SerialAddressConstants.DL] + 2);

```

```

        spArduino.Write(sendBuffer, 0, 7);
        break;
    default:
        break;
    }

}

/// <summary>
/// This function sets the direction of the individual motors in the robot platform.
/// </summary>
public void SetDirection(int directionLF, int directionLB, int directionRF, int
directionRB)
{
    motorDirectionLF = directionLF;
    motorDirectionLB = directionLB;
    motorDirectionRF = directionRF;
    motorDirectionRB = directionRB;

    SendCMD(Commands.SETDIRECTION);
}

/// <summary>
/// This function sets the rpm of the individual motors in the robot platform.
/// </summary>
public void SetRPM(int rpmMotorLF, int rpmMotorLB, int rpmMotorRF, int rpmMotorRB)
{
    rpmLF = rpmMotorLF;
    rpmLB = rpmMotorLB;
    rpmRF = rpmMotorRF;
    rpmRB = rpmMotorRB;

    SendCMD(Commands.SETRPM);
}

/// <summary>
/// This function sends a command to robot platform for getting the ultrasonic sensor
data.
/// </summary>
public void GetSonar()
{
    SendCMD(Commands.GETSONARDATA);
}

/// <summary>
/// This function sends a command to robot platform for getting the encoder data.
/// </summary>
public void GetEncoder()
{
    SendCMD(Commands.GETENCODER);
}

/// <summary>
/// This function sends a command to robot platform for getting the motor direction
values.
/// </summary>
public void GetMotorDirection()
{
    SendCMD(Commands.GETDIRECTION);
}

/// <summary>
/// This function sends a command to robot platform for getting the heading data.
/// </summary>
public void GetHeading()
{
    SendCMD(Commands.GETHEADING);
}

```

```

data.
    /// <summary>
    /// This function sends a command to robot platform for getting the accelerometer
    /// </summary>
    public void GetAccelerometer()
    {
        SendCMD(Commands.GETACCELEROMETER);
    }

    /// <summary>
    /// This function sets the pwm of the individual motors in the robot platform.
    /// </summary>
    public void SetPWM(int pwmMotorLF, int pwmMotorLB, int pwmMotorRF, int pwmMotorRB)
    {
        pwmLF = pwmMotorLF;
        pwmLB = pwmMotorLB;
        pwmRF = pwmMotorRF;
        pwmRB = pwmMotorRB;

        SendCMD(Commands.SETPWM);
    }

    /// <summary>
    /// This function sends a command to robot platform for getting the rpm data.
    /// </summary>
    public void GetRPM()
    {
        SendCMD(Commands.GETRPM);
    }

    /// <summary>
    /// This function sends a command to robot platform for getting the pwm data.
    /// </summary>
    public void GetPWM()
    {
        SendCMD(Commands.GETPWM);
    }

    /// <summary>
    /// This function sends a command to robot platform for getting the current data.
    /// </summary>
    public void GetCurrent()
    {
        SendCMD(Commands.GETCURRENT);
    }

    /// <summary>
    /// This function sets the rps of the individual motors in the robot platform.
    /// </summary>
    public void SetRPS(double rpsLF, double rpsLB, double rpsRF, double rpsRB)
    {
        SetRPM((int)(rpsLF * 60), (int)(rpsLB * 60), (int)(rpsRF * 60), (int)(rpsRB *
60));
    }

    /// <summary>
    /// This function sets the speed of the robot platform.
    /// </summary>
    public void SetSpeed(double speed)
    {
        double oneTurn = 3.14 * 0.2; //bir turda hareket m
        double rps = speed / oneTurn;
        SetRPS(rps, rps, rps, rps);
    }

    /// <summary>
    /// This function drives the robot platform to specified distance with the specified
    speed.
    /// </summary>
    public void SetDistance(double distance, double speed)

```

```

{
    testTime = DateTime.Now;
    timeSeconds = distance / speed;
    double oneTurn = 3.14 * 0.2; //bir turda hareket m
    double howManyTurns = distance / oneTurn;
    distanceEncoderTicks = (int)(howManyTurns * 6533);

    if (sendEncoder == 1)
    {
        distanceStartEncoderLF = positionForEncoderLF;
        distanceStartEncoderLB = positionForEncoderLB;
        distanceStartEncoderRF = positionForEncoderRF;
        distanceStartEncoderRB = positionForEncoderRB;
    }

    SetSpeed(speed);
    timerDistance = new Timer();
    timerDistance.Interval = 100;
    timerDistance.Elapsed += new ElapsedEventHandler(timerDistance_Elapsed);
    timerDistance.Enabled = true;
}

void timerDistance_Elapsed(object sender, ElapsedEventArgs e)
{
    if (sendEncoder == 1)
    {
        int avarageEncoderLF = Math.Abs(positionForEncoderLF -
distanceStartEncoderLF);
        int avarageEncoderLB = Math.Abs(positionForEncoderLB -
distanceStartEncoderLB);
        int avarageEncoderRF = Math.Abs(positionForEncoderRF -
distanceStartEncoderRF);
        int avarageEncoderRB = Math.Abs(positionForEncoderRB -
distanceStartEncoderRB);
        int avarageEncoder = (int)((avarageEncoderLF + avarageEncoderLB +
avarageEncoderRF + avarageEncoderRB) / 4);
        if (avarageEncoder > distanceEncoderTicks)
        {
            SetRPM(0, 0, 0, 0);
            timerDistance.Enabled = false;
        }
    }
    else
    {
        TimeSpan elapsedTime = DateTime.Now.Subtract(testTime);
        if (elapsedTime.Seconds > timeSeconds)
        {
            SetRPM(0, 0, 0, 0);
            timerDistance.Enabled = false;
        }
    }
}
}

/// <summary>
/// This function sets which parameter to be updated.
/// UpdateInterval is in milliseconds. Example use;
/// SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.YES, 100);
/// </summary>
public void SetUpdateParameterInterval(UpdateParameter parameter, UpdateResponse
response, Int16 UpdateInterval)
{
    switch (parameter)
    {
        case UpdateParameter.Accelerometer:
            sendAccelerometer = (int)response;
            accelerometerUpdateInterval = UpdateInterval;
            SendCMD(Commands.ACCELEROMETERUPDATE);
            break;
        case UpdateParameter.Heading:
            sendHeading = (int)response;

```



```

        headingUpdateInterval = UpdateInterval;
        SendCMD(Commands.HEADINGUPDATE);
        break;
    case UpdateParameter.MotorCurrent:
        sendCurrent = (int)response;
        motorCurrentUpdateInterval = UpdateInterval;
        SendCMD(Commands.MOTORCURRENTUPDATE);
        break;
    case UpdateParameter.MotorEncoder:
        sendEncoder = (int)response;
        motorEncoderUpdateInterval = UpdateInterval;
        SendCMD(Commands.MOTORENCODERUPDATE);
        break;
    case UpdateParameter.MotorRPM:
        sendRPM = (int)response;
        motorRPMUpdateInterval = UpdateInterval;
        SendCMD(Commands.MOTORRPMUPDATE);
        break;
    case UpdateParameter.Sonar:
        sendSonar = (int)response;
        sonarUpdateInterval = UpdateInterval;
        SendCMD(Commands.SONARUPDATE);
        break;
    default:
        break;
    }
}

/// <summary>
/// This function rotates the robot platform with specified angle and the specified
direction.
/// </summary>
public void TurnAngle(int angle, RotationDirection direction)
{
    if (sendHeading == 1)
    {
        int angleFirst = heading;
        timerRotation = new Timer();
        timerRotation.Interval = 100;
        timerRotation.Elapsed += new ElapsedEventHandler(timerRotation_Elapsed);
        timerRotation.Enabled = true;

        if (direction == RotationDirection.Left)
        {
            angleToTurn = angleFirst - angle;
            if (angleToTurn < 0)
                angleToTurn = angleToTurn + 360;
            SetDirection(MotorDirection.Backward, MotorDirection.Backward,
MotorDirection.Forward, MotorDirection.Forward);
            SetRPM(20, 20, 20, 20);
        }
        else if (direction == RotationDirection.Right)
        {
            angleToTurn = angleFirst + angle;
            if (angleToTurn > 359)
                angleToTurn = angleToTurn - 360;
            SetDirection(MotorDirection.Forward, MotorDirection.Forward,
MotorDirection.Backward, MotorDirection.Backward);
            SetRPM(20, 20, 20, 20);
        }
    }
    else
    {
        robotStartedHeadingUpdate = true;
        GetHeading();
        SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.YES, 10);
        TurnAngle(angle, direction);
    }
}
}

```

```

        void timerRotation_Elapsed(object sender, ElapsedEventArgs e)
        {
            if (heading < (angleToTurn + angleResolution) && heading > (angleToTurn -
angleResolution))
            {
                if (robotStartedHeadingUpdate)
                {
                    SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.NO, 10);
                    robotStartedHeadingUpdate = false;
                }
                SetRPM(0, 0, 0, 0);
                timerRotation.Enabled = false;
            }
        }

        /// <summary>
        /// This function resets the encoder ticks of the motors.
        /// </summary>
        public void ResetEncoder()
        {
            SendCMD(Commands.RESETENCODER);
        }

        /// <summary>
        /// This function rotates the robot platform to the specified angle with the
specified direction.
        /// </summary>
        public void TurnToAngle(int angle, RotationDirection direction)
        {
            if (sendHeading == 1)
            {
                firstAngleTurn = heading;
                timerTurnToAngle = new Timer();
                timerTurnToAngle.Interval = 100;
                timerTurnToAngle.Elapsed += new
ElapsedEventHandler(timerTurnToAngle_Elapsed);
                timerTurnToAngle.Enabled = true;

                if (direction == RotationDirection.Left)
                {
                    angleToTurn = angle;

                    SetDirection(MotorDirection.Backward, MotorDirection.Backward,
MotorDirection.Forward, MotorDirection.Forward);
                    SetRPM(20, 20, 20, 20);
                }
                else if (direction == RotationDirection.Right)
                {
                    angleToTurn = angle;

                    SetDirection(MotorDirection.Forward, MotorDirection.Forward,
MotorDirection.Backward, MotorDirection.Backward);
                    SetRPM(20, 20, 20, 20);
                }
            }
            else
            {
                robotStartedHeadingUpdate = true;
                GetHeading();
                SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.YES, 10);
                TurnAngle(angle, direction);
            }
        }

        void timerTurnToAngle_Elapsed(object sender, ElapsedEventArgs e)
        {
            if (heading < (angleToTurn + angleResolution) && heading > (angleToTurn -
angleResolution))
            {

```

```
        if (robotStartedHeadingUpdate)
        {
            SetUpdateParameterInterval(EduRobot.UpdateParameter.Heading,
EduRobot.UpdateResponse.NO, 10);
            robotStartedHeadingUpdate = false;
        }
        SetRPM(0, 0, 0, 0);
        timerTurnToAngle.Enabled = false;
    }
}
}
```

Microcontroller Code

```
#include "Arduino.h"
#include <HardwareSerial.h>
#include <QuadratureEncoder.h>
#include <NewPing.h>
#include <PID_v1.h>
#include <Wire.h>
#include <math.h>
#include <LSM303.h>

LSM303 compass;

double SetpointLF, InputLF, OutputLF;
double SetpointLB, InputLB, OutputLB;
double SetpointRF, InputRF, OutputRF;
double SetpointRB, InputRB, OutputRB;

//For Ultrasonic Sensor
#define TRIGGER_PIN_FRONT_LEFT 30 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_FRONT_LEFT 31 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define TRIGGER_PIN_FRONT_MIDDLE 24 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_FRONT_MIDDLE 25 // Arduino pin tied to echo pin on the ultrasonic
sensor.
#define TRIGGER_PIN_FRONT_RIGHT 26 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_FRONT_RIGHT 27 // Arduino pin tied to echo pin on the ultrasonic
sensor.
#define TRIGGER_PIN_RIGHT_FRONT 36 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_RIGHT_FRONT 37 // Arduino pin tied to echo pin on the ultrasonic
sensor.
#define TRIGGER_PIN_RIGHT_BACK 38 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_RIGHT_BACK 39 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define TRIGGER_PIN_BACK_RIGHT 28 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_BACK_RIGHT 29 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define TRIGGER_PIN_BACK_MIDDLE 40 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_BACK_MIDDLE 41 // Arduino pin tied to echo pin on the ultrasonic
sensor.
#define TRIGGER_PIN_BACK_LEFT 42 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_BACK_LEFT 43 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define TRIGGER_PIN_LEFT_BACK 32 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_LEFT_BACK 33 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define TRIGGER_PIN_LEFT_FRONT 34 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN_LEFT_FRONT 35 // Arduino pin tied to echo pin on the ultrasonic sensor.

#define MAX_DISTANCE 250 // Maximum distance we want to ping for (in centimeters). Maximum
sensor distance is rated at 400-500cm.
#define PING_INTERVAL 33 // Milliseconds between pings

unsigned long pingTimer[10]; // When each pings.
unsigned int cm[10]; // Store ping distances.
uint8_t currentSensor = 0; // Which sensor is active.

NewPing sonar[10] = {
NewPing(TRIGGER_PIN_FRONT_LEFT, ECHO_PIN_FRONT_LEFT, MAX_DISTANCE), // NewPing setup of pins
and maximum distance.
NewPing(TRIGGER_PIN_FRONT_MIDDLE, ECHO_PIN_FRONT_MIDDLE, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_FRONT_RIGHT, ECHO_PIN_FRONT_RIGHT, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_RIGHT_FRONT, ECHO_PIN_RIGHT_FRONT, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_RIGHT_BACK, ECHO_PIN_RIGHT_BACK, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_BACK_RIGHT, ECHO_PIN_RIGHT_BACK, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_BACK_MIDDLE, ECHO_PIN_BACK_MIDDLE, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_BACK_LEFT, ECHO_PIN_BACK_LEFT, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_LEFT_BACK, ECHO_PIN_LEFT_BACK, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_LEFT_FRONT, ECHO_PIN_LEFT_FRONT, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
};
```

```

NewPing(TRIGGER_PIN_RIGHT_BACK, ECHO_PIN_RIGHT_BACK, MAX_DISTANCE), // NewPing setup of pins
and maximum distance.
NewPing(TRIGGER_PIN_BACK_RIGHT, ECHO_PIN_BACK_RIGHT, MAX_DISTANCE), // NewPing setup of pins
and maximum distance.
NewPing(TRIGGER_PIN_BACK_MIDDLE, ECHO_PIN_BACK_MIDDLE, MAX_DISTANCE), // NewPing setup of
pins and maximum distance.
NewPing(TRIGGER_PIN_BACK_LEFT, ECHO_PIN_BACK_LEFT, MAX_DISTANCE), // NewPing setup of pins
and maximum distance.
NewPing(TRIGGER_PIN_LEFT_BACK, ECHO_PIN_LEFT_BACK, MAX_DISTANCE), // NewPing setup of pins
and maximum distance.
NewPing(TRIGGER_PIN_LEFT_FRONT, ECHO_PIN_LEFT_FRONT, MAX_DISTANCE) // NewPing setup of pins
and maximum distance.
};

int pwmOutPinMotorLB = 4;
int pwmOutPinMotorRB = 5;
int pwmOutPinMotorLF = 6;
int pwmOutPinMotorRF = 7;

int directionPinMotorLB = 52;
int directionPinMotorRB = 50;
int directionPinMotorLF = 48;
int directionPinMotorRF = 46;

int currentReadPinMotorLB = 3;
int currentReadPinMotorRB = 2;
int currentReadPinMotorLF = 1;
int currentReadPinMotorRF = 0;

int encoderInterruptPinMotorLB = 1; //Pin 3
int encoderInterruptPinMotorRB = 0; //Pin 2
int encoderInterruptPinMotorLF = 4; //Pin 19
int encoderInterruptPinMotorRF = 5; //Pin 18

bool encoderSetMotorLF;
bool encoderSetMotorLB;
bool encoderSetMotorRF;
bool encoderSetMotorRB;

signed long int positionForEncoderLF;
signed long int positionForEncoderLB;
signed long int positionForEncoderRF;
signed long int positionForEncoderRB;

int heading;
float accX;
float accY;
float accZ;

int pwmLF = 0;
int pwmLB = 0;
int pwmRF = 0;
int pwmRB = 0;

int rpmLF = 0;
int rpmLB = 0;
int rpmRF = 0;
int rpmRB = 0;
int deneme = 0;
int actualRpmLF = 0;
int actualRpmLB = 0;
int actualRpmRF = 0;
int actualRpmRB = 0;

int motorDirectionLF;
int motorDirectionLB;
int motorDirectionRF;
int motorDirectionRB;

unsigned int sonarFrontLeftInCm;
unsigned int sonarFrontMiddleInCm;

```

```

unsigned int sonarFrontRightInCm;
unsigned int sonarRightFrontInCm;
unsigned int sonarRightBackInCm;
unsigned int sonarBackRightInCm;
unsigned int sonarBackMiddleInCm;
unsigned int sonarBackLeftInCm;
unsigned int sonarLeftBackInCm;
unsigned int sonarLeftFrontInCm;

unsigned int currentLF;
unsigned int currentLB;
unsigned int currentRF;
unsigned int currentRB;

#define MotorDirectionToForward 1
#define MotorDirectionToBackward 0

#define RS232_DATA_SIZE 32
#define RS232_SEND_SIZE 32
#define doWaitSTX 0
#define doWaitDataLength 1
#define doWaitData 2

#define STX 0x02
#define ETX 0x03
//Address
#define ADR_STX 0
#define ADR_CMD 1
#define ADR_DL 2
#define ADR_PAR 3

//Commands
#define CMD_SETDIRECTION 0x41
#define CMD_GETDIRECTION 0x42
#define CMD_SETPWM 0x43
#define CMD_GETPWM 0x44
#define CMD_GETSONARDATA 0x45
#define CMD_GETENCODER 0x46
#define CMD_GETCURRENT 0x47
#define CMD_EMERGENCYSTOP 0x48
#define CMD_RESETEncoder 0x49
#define CMD_SETRPM 0x50
#define CMD_GETRPM 0x51
#define CMD_GETHEADING 0x52
#define CMD_GETACCELEROMETER 0x53
#define CMD_HEARTBEAT 0x54
#define CMD_MOTORENCODERUPDATE 0x55
#define CMD_MOTORRPMUPDATE 0x56
#define CMD_MOTORCURRENTUPDATE 0x57
#define CMD_SONARUPDATE 0x58
#define CMD_HEADINGUPDATE 0x59
#define CMD_ACCELEROMETERUPDATE 0x60
#define CMD_SETHEARTBEATINTERVAL 0x61

#define DL_SETDIRECTION 0x04
#define DL_GETDIRECTION 0x04
#define DL_SETPWM 0x04
#define DL_GETPWM 0x04
#define DL_GETSONARDATA 0x14
#define DL_GETENCODER 0x10
#define DL_GETCURRENT 0x08
#define DL_EMERGENCYSTOP 0x00
#define DL_RESETEncoder 0x00
#define DL_SETRPM 0x04
#define DL_GETRPM 0x04
#define DL_GETHEADING 0x02
#define DL_GETACCELEROMETER 0x0C
#define DL_HEARTBEAT 0x00
#define DL_MOTORENCODERUPDATE 0x00
#define DL_MOTORRPMUPDATE 0x00
#define DL_MOTORCURRENTUPDATE 0x00

```

```

#define DL_SONARUPDATE          0x00
#define DL_HEADINGUPDATE      0x00
#define DL_ACCELEROMETERUPDATE 0x00

unsigned long lastHeartBeat = 0;
int heartBeatSeconds = 2500;

unsigned long previousMilliseconds = 0;
unsigned long previousMillisecondsUpdate = 0;
signed long int positionForEncoderLFTemp = 0;
signed long int positionForEncoderLBTemp = 0;
signed long int positionForEncoderRFTemp = 0;
signed long int positionForEncoderRBTemp = 0;

int sendSonar = 0;
int sendEncoder = 0;
int sendRPM = 0;
int sendHeading = 0;
int sendAccelerometer = 0;
int sendCurrent = 0;

unsigned long motorEncoderUpdate = 0;
unsigned long motorRPMUpdate = 0;
unsigned long motorCurrentUpdate = 0;
unsigned long sonarUpdate = 0;
unsigned long headingUpdate = 0;
unsigned long accelerometerUpdate = 0;

long motorEncoderUpdateInterval = 250;
long motorRPMUpdateInterval = 250;
long motorCurrentUpdateInterval = 250;
long sonarUpdateInterval = 750;
long headingUpdateInterval = 750;
long accelerometerUpdateInterval = 750;

struct{
    unsigned char buff[RS232_DATA_SIZE];
    unsigned char length;
    signed int readValue;
    unsigned char packetState;
} myRS232_Data;

struct{
    unsigned char buff[RS232_SEND_SIZE];
} myRS232_Send;

void setup()
{
    pwmLF = 0;
    pwmLB = 0;
    pwmRF = 0;
    pwmRB = 0;

    rpmLF = 0;
    rpmLB = 0;
    rpmRF = 0;
    rpmRB = 0;

    analogWrite(pwmOutPinMotorLF, pwmLF); // Stop the motor left 1
    analogWrite(pwmOutPinMotorLB, pwmLB); // Stop the motor left 1
    analogWrite(pwmOutPinMotorRF, pwmRF); // Stop the motor left 1
    analogWrite(pwmOutPinMotorRB, pwmRB); // Stop the motor left 1

    pinMode(directionPinMotorLF, OUTPUT); // Motor Left 1 Direction Pin, Defined as Output
    pinMode(directionPinMotorLB, OUTPUT); // Motor Left 1 Direction Pin, Defined as
Output
    pinMode(directionPinMotorRF, OUTPUT); // Motor Left 1 Direction Pin, Defined as
Output
    pinMode(directionPinMotorRB, OUTPUT); // Motor Left 1 Direction Pin, Defined as
Output
}

```

```

    motorDirectionLF = MotorDirectionToForward;
    motorDirectionLB = MotorDirectionToForward;
    motorDirectionRF = MotorDirectionToForward;
    motorDirectionRB = MotorDirectionToForward;

    digitalWrite(directionPinMotorLF, motorDirectionLF); // Set default direction for motor
left front
    digitalWrite(directionPinMotorLB, motorDirectionLB); // Set default direction for
motor left back
    digitalWrite(directionPinMotorRF, motorDirectionRF); // Set default direction for
motor right front
    digitalWrite(directionPinMotorRB, motorDirectionRB); // Set default direction for
motor right back

    pinMode(encoderInterruptPinMotorLF, INPUT); // sets pin A as input
    pinMode(encoderInterruptPinMotorLB, INPUT); // sets pin A as input
    pinMode(encoderInterruptPinMotorRF, INPUT); // sets pin A as input
    pinMode(encoderInterruptPinMotorRB, INPUT); // sets pin A as input

    encoderSetMotorLF = digitalRead(encoderInterruptPinMotorLF);
    encoderSetMotorLB = digitalRead(encoderInterruptPinMotorLB);
    encoderSetMotorRF = digitalRead(encoderInterruptPinMotorRF);
    encoderSetMotorRB = digitalRead(encoderInterruptPinMotorRB);

    positionForEncoderLF = 0;
    positionForEncoderLB = 0;
    positionForEncoderRF = 0;
    positionForEncoderRB = 0;

    attachInterrupt(encoderInterruptPinMotorLF, HandleMotorLF, CHANGE); // Motor Left 1
Encoder interrupt A, Pin 2
    attachInterrupt(encoderInterruptPinMotorLB, HandleMotorLB, CHANGE); // Motor Left 1
Encoder interrupt B, Pin 3
    attachInterrupt(encoderInterruptPinMotorRF, HandleMotorRF, CHANGE); // Motor Left 1
Encoder interrupt A, Pin 2
    attachInterrupt(encoderInterruptPinMotorRB, HandleMotorRB, CHANGE); // Motor Left 1
Encoder interrupt B, Pin 3

    sonarFrontLeftInCm = 0;
    sonarFrontMiddleInCm = 0;
    sonarFrontRightInCm = 0;
    sonarRightFrontInCm = 0;
    sonarRightBackInCm = 0;
    sonarBackRightInCm = 0;
    sonarBackMiddleInCm = 0;
    sonarBackLeftInCm = 0;
    sonarLeftBackInCm = 0;
    sonarLeftFrontInCm = 0;

    currentLF = 0;
    currentLB = 0;
    currentRF = 0;
    currentRB = 0;

    InputLF = 0;
    SetpointLF = 0;

    InputLB = 0;
    SetpointLB = 0;

    InputRF = 0;
    SetpointRF = 0;

    InputRB = 0;
    SetpointRB = 0;

    myRS232_Data.packetState = dowaitSTX;

Serial.begin(115200);

Wire.begin();

```



```

compass.init();
compass.enableDefault();

// Calibration values. Use the Calibrate example program to get the values for
// your compass.
compass.m_min.x = -471; compass.m_min.y = -259; compass.m_min.z = -549;
compass.m_max.x = +108; compass.m_max.y = +258; compass.m_max.z = -481;

pingTimer[0] = millis() + 75; // First ping start in ms.
for (uint8_t i = 1; i < 10; i++)
    pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;
}

void HandleMotorLF()
{
    if(motorDirectionLF == MotorDirectionToForward)
        positionForEncoderLF++;
    else if(motorDirectionLF == MotorDirectionToBackward)
        positionForEncoderLF--;
}

void HandleMotorLB()
{
    if(motorDirectionLB == MotorDirectionToForward)
        positionForEncoderLB++;
    else if(motorDirectionLB == MotorDirectionToBackward)
        positionForEncoderLB--;
}

void HandleMotorRF()
{
    if(motorDirectionRF == MotorDirectionToForward)
        positionForEncoderRF++;
    else if(motorDirectionRF == MotorDirectionToBackward)
        positionForEncoderRF--;
}

void HandleMotorRB()
{
    if(motorDirectionRB == MotorDirectionToForward)
        positionForEncoderRB++;
    else if(motorDirectionRB == MotorDirectionToBackward)
        positionForEncoderRB--;
}

void getSerial()
{
    int numberOfBytes = Serial.available();

    if(numberOfBytes == 0)
        return;

    while(numberOfBytes){
        // count 0 olana degin
        myRS232_Data.readValue = Serial.read();
        // bir bayt veriyi al
        numberOfBytes--;

        if(myRS232_Data.readValue == -1)
            return;

        if(myRS232_Data.packetState == doWaitSTX){
            myRS232_Data.length = 0;
            if(myRS232_Data.readValue != STX){
                continue;
            }
            myRS232_Data.buff[ADR_STX] = STX;
            myRS232_Data.packetState= doWaitDataLength;
            myRS232_Data.length = 1;
            continue;
        }
    }
}

```

```

    }
    else if(myRS232_Data.packetState == doWaitDataLength)
    {
        myRS232_Data.buff[myRS232_Data.length++] = myRS232_Data.readValue;
        myRS232_Data.packetState = doWaitData;
    }
    else if(myRS232_Data.packetState == doWaitData){
        myRS232_Data.buff[myRS232_Data.length++] = myRS232_Data.readValue;
        if(myRS232_Data.length < (myRS232_Data.buff[2] + 5) ){ //
paket tamamlanmadı
                continue;
            }

            if(myRS232_Data.buff[myRS232_Data.length-2] !=
fnCalcBCC(myRS232_Data.buff,1,myRS232_Data.length-3)){//Paket BCC değeri bozuk ise
                myRS232_Data.packetState= doWaitSTX;
                myRS232_Data.length = 0;

                break;
            }

            fnPCSoftware();
// Paket BCC OK, her seferde tek bir
paket process edelim
                break;
            }
            else{
                myRS232_Data.packetState = doWaitSTX;
                continue;
            }
        }
    }

//-----
// İşlev Adı: fnCalcBCC(...)
// Tanımı: Gönderilen bir dizinin BCC değerinin hesaplanması
// Girdiler: myBCCArray : Verilerin tutulduğu dizin
//           chStartPnt : Başlangıç noktası
//           chStopPnt : Bitiş noktası
// Dönüş Değeri: BCC değeri
//-----
unsigned char fnCalcBCC(unsigned char myBCCArray[], unsigned char chStartPnt, unsigned char
chStopPnt){

    unsigned char chTempBCC, chTempIndex;
    chTempBCC = myBCCArray[chStartPnt];
    for(chTempIndex=chStartPnt+1; chTempIndex<=chStopPnt; chTempIndex++){
        chTempBCC ^= myBCCArray[chTempIndex];
    }
    return chTempBCC;
}

//-----
// İşlev fnPCSoftware() :
// Aldığı veri :
// Döndüğü veri :
//-----
void fnPCSoftware(){

    switch(myRS232_Data.buff[ADR_CMD]){
        // ASCII
        //-----
        case 0x00:
            break;
        //-----
        case CMD_SETDIRECTION:
            if(myRS232_Data.buff[ADR_PAR] == 'I')
                motorDirectionLF = MotorDirectionToForward;
            else if(myRS232_Data.buff[ADR_PAR] == 'G')
                motorDirectionLF = MotorDirectionToBackward;
    }
}

```

```

        digitalWrite(directionPinMotorLF, motorDirectionLF); // Set default
direction for motor left front

        if(myRS232_Data.buff[ADR_PAR + 1] == 'I')
            motorDirectionLB = MotorDirectionToForward;
        else if(myRS232_Data.buff[ADR_PAR + 1] == 'G')
            motorDirectionLB = MotorDirectionToBackward;
        digitalWrite(directionPinMotorLB, motorDirectionLB); // Set default
direction for motor left back

        if(myRS232_Data.buff[ADR_PAR + 2] == 'I')
            motorDirectionRF = MotorDirectionToForward;
        else if(myRS232_Data.buff[ADR_PAR + 2] == 'G')
            motorDirectionRF = MotorDirectionToBackward;
        digitalWrite(directionPinMotorRF, motorDirectionRF); // Set default
direction for motor right front

        if(myRS232_Data.buff[ADR_PAR + 3] == 'I')
            motorDirectionRB = MotorDirectionToForward;
        else if(myRS232_Data.buff[ADR_PAR + 3] == 'G')
            motorDirectionRB = MotorDirectionToBackward;
        digitalWrite(directionPinMotorRB, motorDirectionRB); // Set default
direction for motor right back

        fnSendCmd(CMD_GETDIRECTION);
        break;
//-----

        case CMD_GETSONARDATA:
            fnSendCmd(CMD_GETSONARDATA);
            break;
//-----

        case CMD_SETPWM:
            pwmLF = myRS232_Data.buff[ADR_PAR];
            analogWrite(pwmOutPinMotorLF, pwmLF); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

            pwmLB = myRS232_Data.buff[ADR_PAR + 1];
            analogWrite(pwmOutPinMotorLB, pwmLB); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

            pwmRF = myRS232_Data.buff[ADR_PAR + 2];
            analogWrite(pwmOutPinMotorRF, pwmRF); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

            pwmRB = myRS232_Data.buff[ADR_PAR + 3];
            analogWrite(pwmOutPinMotorRB, pwmRB); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

            fnSendCmd(CMD_GETPWM);
            break;
//-----

        case CMD_GETENCODER:
            fnSendCmd(CMD_GETENCODER);
            break;
//-----

        case CMD_GETDIRECTION:
            fnSendCmd(CMD_GETDIRECTION);
            break;
//-----

        case CMD_GETPWM:
            fnSendCmd(CMD_GETPWM);
            break;
//-----

        case CMD_GETCURRENT:
            fnSendCmd(CMD_GETCURRENT);
            break;
//-----

        case CMD_EMERGENCYSTOP:
            pwmLF = 0;

```

```

        analogWrite(pwmOutPinMotorLF, pwmLF); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

        pwmLB = 0;
        analogWrite(pwmOutPinMotorLB, pwmLB); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

        pwmRF = 0;
        analogWrite(pwmOutPinMotorRF, pwmRF); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

        pwmRB = 0;
        analogWrite(pwmOutPinMotorRB, pwmRB); // analogRead values go from 0
to 1023, analogWrite values from 0 to 255

        fnSendCmd(CMD_GETPWM);
    break;
//-----
    case CMD_RESETEncoder:
        positionForEncoderLF = 0;
        positionForEncoderLB = 0;
        positionForEncoderRF = 0;
        positionForEncoderRB = 0;
        positionForEncoderLFTemp = 0;
        positionForEncoderLBTemp = 0;
        positionForEncoderRFTemp = 0;
        positionForEncoderRBTemp = 0;
        fnSendCmd(CMD_GETENCODER);
    break;
//-----
    case CMD_SETRPM:
        rpmLF = myRS232_Data.buff[ADR_PAR];
        SetpointLF = rpmLF;

        //analogWrite(pwmOutPinMotorLF, rpmLF); // analogRead values go from
0 to 1023, analogWrite values from 0 to 255

        rpmLB = myRS232_Data.buff[ADR_PAR + 1];
        SetpointLB = rpmLB;
        //analogWrite(pwmOutPinMotorLB, rpmLB); // analogRead values go from
0 to 1023, analogWrite values from 0 to 255

        rpmRF = myRS232_Data.buff[ADR_PAR + 2];
        SetpointRF = rpmRF;
        //analogWrite(pwmOutPinMotorRF, rpmRF); // analogRead values go from
0 to 1023, analogWrite values from 0 to 255

        rpmRB = myRS232_Data.buff[ADR_PAR + 3];
        SetpointRB = rpmRB;
        //analogWrite(pwmOutPinMotorRB, rpmRB); // analogRead values go from
0 to 1023, analogWrite values from 0 to 255

        fnSendCmd(CMD_GETRPM);
    break;
//-----
    case CMD_GETRPM:
        fnSendCmd(CMD_GETRPM);
    break;
//-----
    case CMD_GETHEADING:
        fnSendCmd(CMD_GETHEADING);
    break;
//-----
    case CMD_GETACCELEROMETER:
        fnSendCmd(CMD_GETACCELEROMETER);
    break;
//-----
    case CMD_HEARTBEAT:
        lastHeartBeat = millis();
        fnSendCmd(CMD_HEARTBEAT);
    break;

```

```

//-----
case CMD_MOTORENCODERUPDATE:
    sendEncoder = myRS232_Data.buff[ADR_PAR];
    motorEncoderUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    motorEncoderUpdateInterval = (motorEncoderUpdateInterval<<8);
    motorEncoderUpdateInterval = motorEncoderUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_MOTORENCODERUPDATE);
break;
//-----
case CMD_MOTORRPMUPDATE:
    sendRPM = myRS232_Data.buff[ADR_PAR];
    motorRPMUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    motorRPMUpdateInterval = (motorRPMUpdateInterval<<8);
    motorRPMUpdateInterval = motorRPMUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_MOTORRPMUPDATE);
break;
//-----
case CMD_MOTORCURRENTUPDATE:
    sendCurrent = myRS232_Data.buff[ADR_PAR];
    motorCurrentUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    motorCurrentUpdateInterval = (motorCurrentUpdateInterval<<8);
    motorCurrentUpdateInterval = motorCurrentUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_MOTORCURRENTUPDATE);
break;
//-----
case CMD_SONARUPDATE:
    sendSonar = myRS232_Data.buff[ADR_PAR];
    sonarUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    sonarUpdateInterval = (sonarUpdateInterval<<8);
    sonarUpdateInterval = sonarUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_SONARUPDATE);
break;
//-----
case CMD_HEADINGUPDATE:
    sendHeading = myRS232_Data.buff[ADR_PAR];
    headingUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    headingUpdateInterval = (headingUpdateInterval<<8);
    headingUpdateInterval = headingUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_HEADINGUPDATE);
break;
//-----
case CMD_ACCELEROMETERUPDATE:
    sendAccelerometer = myRS232_Data.buff[ADR_PAR];
    accelerometerUpdateInterval = myRS232_Data.buff[ADR_PAR + 1];
    accelerometerUpdateInterval = (accelerometerUpdateInterval<<8);
    accelerometerUpdateInterval = accelerometerUpdateInterval |
myRS232_Data.buff[ADR_PAR + 2];

    fnSendCmd(CMD_ACCELEROMETERUPDATE);
break;
//-----
case CMD_SETHEARTBEATINTERVAL:
    heartBeatSeconds = myRS232_Data.buff[ADR_PAR];
    heartBeatSeconds = (heartBeatSeconds<<8);
    heartBeatSeconds = heartBeatSeconds | myRS232_Data.buff[ADR_PAR +
1];

    fnSendCmd(CMD_HEARTBEAT);
break;
//-----
default:

```

```

        break;
    }

    myRS232_Data.packetState = dowaitSTX;
}

//-----
// İşlev fnSendBinary() :
// Aldığı veri : ---
// Döndüğü veri : ---
//
//-----
void fnSendCmd(unsigned char myCmd){
    memset(myRS232_Send.buff, 0x00, RS232_SEND_SIZE);
    myRS232_Send.buff[ADR_STX] = STX;

    byte * bX = 0;
    byte * bY = 0;
    byte * bZ = 0;
    byte * b = 0;

    int ADR_BCC = 0;
    int ADR_ETX = 0;

    switch(myCmd){

        //-----
        case CMD_GETDIRECTION:
            myRS232_Send.buff[ADR_CMD] = myCmd;

            myRS232_Send.buff[ADR_DL] = DL_GETDIRECTION;
            ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
            ADR_ETX = ADR_BCC + 1;
            myRS232_Send.buff[ADR_ETX] = ETX;

            if(motorDirectionLF == MotorDirectionToForward)
                myRS232_Send.buff[ADR_PAR] = 'I';
            else if(motorDirectionLF == MotorDirectionToBackward)
                myRS232_Send.buff[ADR_PAR] = 'G';

            if(motorDirectionLB == MotorDirectionToForward)
                myRS232_Send.buff[ADR_PAR + 1] = 'I';
            else if(motorDirectionLB == MotorDirectionToBackward)
                myRS232_Send.buff[ADR_PAR + 1] = 'G';

            if(motorDirectionRF == MotorDirectionToForward)
                myRS232_Send.buff[ADR_PAR + 2] = 'I';
            else if(motorDirectionRF == MotorDirectionToBackward)
                myRS232_Send.buff[ADR_PAR + 2] = 'G';

            if(motorDirectionRB == MotorDirectionToForward)
                myRS232_Send.buff[ADR_PAR + 3] = 'I';
            else if(motorDirectionRB == MotorDirectionToBackward)
                myRS232_Send.buff[ADR_PAR + 3] = 'G';

            myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

            break;

        //-----
        case CMD_GETSONARDATA:

            /*sonarFrontLeftInCm = sonarFrontLeft.ping_cm();
            sonarFrontMiddleInCm = sonarFrontMiddle.ping_cm();
            sonarFrontRightInCm = sonarFrontRight.ping_cm();
            sonarRightFrontInCm = sonarRightFront.ping_cm();
            sonarRightBackInCm = sonarRightBack.ping_cm();
            sonarBackRightInCm = sonarBackRight.ping_cm();
            sonarBackMiddleInCm = sonarBackMiddle.ping_cm();
            sonarBackLeftInCm = sonarBackLeft.ping_cm();
            sonarLeftBackInCm = sonarLeftBack.ping_cm();

```

```

        sonarLeftFrontInCm = sonarLeftFront.ping_cm();*/

        myRS232_Send.buff[ADR_CMD] = myCmd;

        myRS232_Send.buff[ADR_DL] = DL_GETSONARDATA;
        ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        myRS232_Send.buff[ADR_ETX] = ETX;

        myRS232_Send.buff[ADR_PAR] = (sonarFrontLeftInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 1] = sonarFrontLeftInCm;

        myRS232_Send.buff[ADR_PAR + 2] = (sonarFrontMiddleInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 3] = sonarFrontMiddleInCm;

        myRS232_Send.buff[ADR_PAR + 4] = (sonarFrontRightInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 5] = sonarFrontRightInCm;

        myRS232_Send.buff[ADR_PAR + 6] = (sonarRightFrontInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 7] = sonarRightFrontInCm;

        myRS232_Send.buff[ADR_PAR + 8] = (sonarRightBackInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 9] = sonarRightBackInCm;

        myRS232_Send.buff[ADR_PAR + 10] = (sonarBackRightInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 11] = sonarBackRightInCm;

        myRS232_Send.buff[ADR_PAR + 12] = (sonarBackMiddleInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 13] = sonarBackMiddleInCm;

        myRS232_Send.buff[ADR_PAR + 14] = (sonarBackLeftInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 15] = sonarBackLeftInCm;

        myRS232_Send.buff[ADR_PAR + 16] = (sonarLeftBackInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 17] = sonarLeftBackInCm;

        myRS232_Send.buff[ADR_PAR + 18] = (sonarLeftFrontInCm >> 8);
        myRS232_Send.buff[ADR_PAR + 19] = sonarLeftFrontInCm;

        myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

        break;
//-----
case CMD_GETPWM:
        myRS232_Send.buff[ADR_CMD] = myCmd;

        myRS232_Send.buff[ADR_DL] = DL_GETPWM;
        ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        myRS232_Send.buff[ADR_ETX] = ETX;

        myRS232_Send.buff[ADR_PAR] = pwmLF;
        myRS232_Send.buff[ADR_PAR + 1] = pwmLB;
        myRS232_Send.buff[ADR_PAR + 2] = pwmRF;
        myRS232_Send.buff[ADR_PAR + 3] = pwmRB;

        myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

        break;
//-----
case CMD_GETENCODER:

        myRS232_Send.buff[ADR_CMD] = myCmd;

        myRS232_Send.buff[ADR_DL] = DL_GETENCODER;
        ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        myRS232_Send.buff[ADR_ETX] = ETX;

```

```

myRS232_Send.buf[ADR_PAR] = (positionForEncoderLF >> 24);
myRS232_Send.buf[ADR_PAR + 1] = (positionForEncoderLF >> 16);
myRS232_Send.buf[ADR_PAR + 2] = (positionForEncoderLF >> 8);
myRS232_Send.buf[ADR_PAR + 3] = positionForEncoderLF;

myRS232_Send.buf[ADR_PAR + 4] = (positionForEncoderLB >> 24);
myRS232_Send.buf[ADR_PAR + 5] = (positionForEncoderLB >> 16);
myRS232_Send.buf[ADR_PAR + 6] = (positionForEncoderLB >> 8);
myRS232_Send.buf[ADR_PAR + 7] = positionForEncoderLB;

myRS232_Send.buf[ADR_PAR + 8] = (positionForEncoderRF >> 24);
myRS232_Send.buf[ADR_PAR + 9] = (positionForEncoderRF >> 16);
myRS232_Send.buf[ADR_PAR + 10] = (positionForEncoderRF >> 8);
myRS232_Send.buf[ADR_PAR + 11] = positionForEncoderRF;

myRS232_Send.buf[ADR_PAR + 12] = (positionForEncoderRB >> 24);
myRS232_Send.buf[ADR_PAR + 13] = (positionForEncoderRB >> 16);
myRS232_Send.buf[ADR_PAR + 14] = (positionForEncoderRB >> 8);
myRS232_Send.buf[ADR_PAR + 15] = positionForEncoderRB;

myRS232_Send.buf[ADR_BCC] = fnCalcBCC(myRS232_Send.buf, 1,
myRS232_Send.buf[ADR_DL] + 2);

break;
//-----
case CMD_GETCURRENT:

currentLF = analogRead(currentReadPinMotorLF);
currentLB = analogRead(currentReadPinMotorLB);
currentRF = analogRead(currentReadPinMotorRF);
currentRB = analogRead(currentReadPinMotorRB);

myRS232_Send.buf[ADR_CMD] = myCmd;

myRS232_Send.buf[ADR_DL] = DL_GETENCODER;
ADR_BCC = ADR_DL + myRS232_Send.buf[ADR_DL] + 1;
ADR_ETX = ADR_BCC + 1;
myRS232_Send.buf[ADR_ETX] = ETX;

myRS232_Send.buf[ADR_PAR] = (currentLF >> 8);
myRS232_Send.buf[ADR_PAR + 1] = currentLF;

myRS232_Send.buf[ADR_PAR + 2] = (currentLB >> 8);
myRS232_Send.buf[ADR_PAR + 3] = currentLB;

myRS232_Send.buf[ADR_PAR + 4] = (currentRF >> 8);
myRS232_Send.buf[ADR_PAR + 5] = currentRF;

myRS232_Send.buf[ADR_PAR + 6] = (currentRB >> 8);
myRS232_Send.buf[ADR_PAR + 7] = currentRB;

myRS232_Send.buf[ADR_BCC] = fnCalcBCC(myRS232_Send.buf, 1,
myRS232_Send.buf[ADR_DL] + 2);

break;
//-----
case CMD_GETRPM:

myRS232_Send.buf[ADR_CMD] = myCmd;

myRS232_Send.buf[ADR_DL] = DL_GETRPM;
ADR_BCC = ADR_DL + myRS232_Send.buf[ADR_DL] + 1;
ADR_ETX = ADR_BCC + 1;
myRS232_Send.buf[ADR_ETX] = ETX;

myRS232_Send.buf[ADR_PAR] = actualRpmLF;
myRS232_Send.buf[ADR_PAR + 1] = actualRpmLB;
myRS232_Send.buf[ADR_PAR + 2] = actualRpmRF;
myRS232_Send.buf[ADR_PAR + 3] = actualRpmRB;

```



```

        myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

        break;
//-----
case CMD_GETHEADING:
    compass.read();
    heading = compass.heading((LSM303::vector){0,-1,0});

    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_GETHEADING;
ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
ADR_ETX = ADR_BCC + 1;
myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_PAR] = (heading >> 8);
    myRS232_Send.buff[ADR_PAR + 1] = heading;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_GETACCELEROMETER:

    compass.read();

    accX = compass.a.x;
    accY = compass.a.y;
    accZ = compass.a.z;

    bX = (byte *) &accX;
    bY = (byte *) &accY;
    bZ = (byte *) &accZ;

    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_GETACCELEROMETER;
ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
ADR_ETX = ADR_BCC + 1;
myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_PAR] = bX[0];
    myRS232_Send.buff[ADR_PAR + 1] = bX[1];
    myRS232_Send.buff[ADR_PAR + 2] = bX[2];
    myRS232_Send.buff[ADR_PAR + 3] = bX[3];

    myRS232_Send.buff[ADR_PAR + 4] = bY[0];
    myRS232_Send.buff[ADR_PAR + 5] = bY[1];
    myRS232_Send.buff[ADR_PAR + 6] = bY[2];
    myRS232_Send.buff[ADR_PAR + 7] = bY[3];

    myRS232_Send.buff[ADR_PAR + 8] = bZ[0];
    myRS232_Send.buff[ADR_PAR + 9] = bZ[1];
    myRS232_Send.buff[ADR_PAR + 10] = bZ[2];
    myRS232_Send.buff[ADR_PAR + 11] = bZ[3];

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_HEARTBEAT:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_HEARTBEAT;
ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
ADR_ETX = ADR_BCC + 1;
myRS232_Send.buff[ADR_ETX] = ETX;

```

```

        myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

        break;
//-----
case CMD_MOTORENCODERUPDATE:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_MOTORENCODERUPDATE;
    ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
    ADR_ETX = ADR_BCC + 1;
    myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_MOTORRPMUPDATE:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_MOTORRPMUPDATE;
    ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
    ADR_ETX = ADR_BCC + 1;
    myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_MOTORCURRENTUPDATE:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_MOTORCURRENTUPDATE;
    ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
    ADR_ETX = ADR_BCC + 1;
    myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_SONARUPDATE:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_SONARUPDATE;
    ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
    ADR_ETX = ADR_BCC + 1;
    myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_HEADINGUPDATE:
    myRS232_Send.buff[ADR_CMD] = myCmd;

    myRS232_Send.buff[ADR_DL] = DL_HEADINGUPDATE;
    ADR_BCC = ADR_DL + myRS232_Send.buff[ADR_DL] + 1;
    ADR_ETX = ADR_BCC + 1;
    myRS232_Send.buff[ADR_ETX] = ETX;

    myRS232_Send.buff[ADR_BCC] = fnCalcBCC(myRS232_Send.buff, 1,
myRS232_Send.buff[ADR_DL] + 2);

    break;
//-----
case CMD_ACCELEROMETERUPDATE:

```

```

        myRS232_Send.buf[ADR_CMD] = myCmd;

        myRS232_Send.buf[ADR_DL] = DL_ACCELEROMETERUPDATE;
        ADR_BCC = ADR_DL + myRS232_Send.buf[ADR_DL] + 1;
        ADR_ETX = ADR_BCC + 1;
        myRS232_Send.buf[ADR_ETX] = ETX;

        myRS232_Send.buf[ADR_BCC] = fnCalcBCC(myRS232_Send.buf, 1,
myRS232_Send.buf[ADR_DL] + 2);

        break;
//-----
default:
        break;
    }

    Serial.write(myRS232_Send.buf, myRS232_Send.buf[ADR_DL] + 5);
}
int i = 0;
void loop()
{
    getSerial();

    unsigned long currentLoopMillis = millis();

    if((currentLoopMillis - lastHeartBeat) > heartBeatSeconds)
    {
        SetpointLF = 0;
        SetpointLB = 0;
        SetpointRF = 0;
        SetpointRB = 0;
    }

    if((currentLoopMillis - motorEncoderUpdate) > motorEncoderUpdateInterval)
    {
        motorEncoderUpdate = currentLoopMillis;
        if(sendEncoder)
        {
            fnSendCmd(CMD_GETENCODER);
        }
    }
    if((currentLoopMillis - motorRPMUpdate) > motorRPMUpdateInterval)
    {
        motorRPMUpdate = currentLoopMillis;
        if(sendRPM)
        {
            fnSendCmd(CMD_GETRPM);
        }
    }
    if((currentLoopMillis - motorCurrentUpdate) > motorCurrentUpdateInterval)
    {
        motorCurrentUpdate = currentLoopMillis;
        if(sendCurrent)
        {
            fnSendCmd(CMD_GETCURRENT);
        }
    }
    if((currentLoopMillis - sonarUpdate) > sonarUpdateInterval)
    {
        sonarUpdate = currentLoopMillis;
        if(sendSonar)
        {
            fnSendCmd(CMD_GETSONARDATA);
        }
    }
    if((currentLoopMillis - headingUpdate) > headingUpdateInterval)
    {
        headingUpdate = currentLoopMillis;
        if(sendHeading)
        {
            fnSendCmd(CMD_GETHEADING);
        }
    }
}

```

```

    }
}
if((currentLoopMillis - accelerometerUpdate) > accelerometerUpdateInterval)
{
    accelerometerUpdate = currentLoopMillis;
    if(sendAccelerometer)
    {
        fnSendCmd(CMD_GETACCELEROMETER);
    }
}

if (currentLoopMillis >= pingTimer[i]) {
    pingTimer[i] += PING_INTERVAL * 10;
    sonar[currentSensor].timer_stop();
    currentSensor = i;
    sonar[currentSensor].ping_timer(echoCheck);
    i ++ ;
    if(i == 10)
        i=0;
}

pid();
}

void echoCheck() { // If ping echo, set distance to array.
    if (sonar[currentSensor].check_timer())
    {
        cm[currentSensor] = sonar[currentSensor].ping_result / US_ROUNDTRIP_CM;

        if(currentSensor == 0)
            sonarFrontLeftInCm = cm[currentSensor];
        else if(currentSensor == 1)
            sonarFrontMiddleInCm = cm[currentSensor];
        else if(currentSensor == 2)
            sonarFrontRightInCm = cm[currentSensor];
        else if(currentSensor == 3)
            sonarRightFrontInCm = cm[currentSensor];
        else if(currentSensor == 4)
            sonarRightBackInCm = cm[currentSensor];
        else if(currentSensor == 5)
            sonarBackRightInCm = cm[currentSensor];
        else if(currentSensor == 6)
            sonarBackMiddleInCm = cm[currentSensor];
        else if(currentSensor == 7)
            sonarBackLeftInCm = cm[currentSensor];
        else if(currentSensor == 8)
            sonarLeftBackInCm = cm[currentSensor];
        else if(currentSensor == 9)
            sonarLeftFrontInCm = cm[currentSensor];
    }
}

void oneSensorCycle() { // Do something with the results.
    for (uint8_t i = 0; i < 10; i++) {
        Serial.print(i);
        Serial.print("=");
        Serial.print(cm[i]);
        Serial.print("cm ");
    }
    Serial.println();
}

double kP = 85;
double kI = 0.005;
double kD = 5;
double errSumLF, errSumLB, errSumRF, errSumRB;
double lastErrLF, lastErrLB, lastErrRF, lastErrRB;
void pid()
{
    unsigned long currentMilliseconds = millis();
    unsigned long milliSecsSinceLastUpdate = currentMilliseconds - previousMilliseconds;
}

```

```

if(milliSecsSinceLastUpdate > 5)
{
    float fActualRpmLF = (((float)positionForEncoderLF-
(float)positionForEncoderLFTemp)/(float)milliSecsSinceLastUpdate)/(float)6533)*(float)1000*(f
loat)60;
    InputLF = abs((double) fActualRpmLF);
    actualRpmLF = (int) InputLF;

    double errorLF = SetpointLF - InputLF;
    errSumLF += (errorLF * milliSecsSinceLastUpdate);
    double dErrLF = (errorLF - lastErrLF) / milliSecsSinceLastUpdate;

    /*Compute PID Output*/
    OutputLF = kP * errorLF + kI * errSumLF + kD * dErrLF;
    if(OutputLF > 255)
        OutputLF = 255;
    if(OutputLF < 0)
        OutputLF = 0;

    currentLF = analogRead(currentReadPinMotorLF);
    if(currentLF>3500)
    {
        OutputLF = OutputLF/2;
    }
    /*Remember some variables for next time*/
    lastErrLF = errorLF;

    //PIDLF.Compute();

    int pwmLFOutput =(int) (OutputLF);

    if(SetpointLF != 0)
    {
        analogWrite(pwmOutPinMotorLF, pwmLFOutput);
    }
    else
    {
        analogWrite(pwmOutPinMotorLF, 0);
    }
    positionForEncoderLFTemp = positionForEncoderLF;

    float fActualRpmLB = (((float)positionForEncoderLB-
(float)positionForEncoderLBTemp)/(float)milliSecsSinceLastUpdate)/(float)6533)*(float)1000*(f
loat)60;
    InputLB = abs((double) fActualRpmLB);
    actualRpmLB = (int) InputLB;
    //PIDLB.Compute();

    double errorLB = SetpointLB - InputLB;
    errSumLB += (errorLB * milliSecsSinceLastUpdate);
    double dErrLB = (errorLB - lastErrLB) / milliSecsSinceLastUpdate;

    /*Compute PID Output*/
    OutputLB = kP * errorLB + kI * errSumLB + kD * dErrLB;
    if(OutputLB > 255)
        OutputLB = 255;
    if(OutputLB < 0)
        OutputLB = 0;

    currentLB = analogRead(currentReadPinMotorLB);
    if(currentLB>3500)
    {
        OutputLB = OutputLB/2;
    }
    /*Remember some variables for next time*/
    lastErrLB = errorLB;

```

```

int pwmLBOOutput =(int) (OutputLB);
if(SetpointLB != 0)
{
    analogWrite(pwmOutPinMotorLB, pwmLBOOutput);
}
else
{
    analogWrite(pwmOutPinMotorLB, 0);
}
positionForEncoderLBTemp = positionForEncoderLB;

float fActualRpmRF = (((float)positionForEncoderRF-
(float)positionForEncoderRFTemp)/(float)milliSecsSinceLastUpdate)/(float)6533)*(float)1000*(f
loat)60;
InputRF = abs((double) fActualRpmRF);
actualRpmRF = (int) InputRF;

double errorRF = SetpointRF - InputRF;
errSumRF += (errorRF * milliSecsSinceLastUpdate);
double dErrRF = (errorRF - lastErrRF) / milliSecsSinceLastUpdate;

/*Compute PID Output*/
OutputRF = kP * errorRF + kI * errSumRF + kD * dErrRF;
if(OutputRF > 255)
    OutputRF = 255;
if(OutputRF < 0)
    OutputRF = 0;

currentRF = analogRead(currentReadPinMotorRF);
if(currentRF>3500)
{
    OutputRF = OutputRF/2;
}
/*Remember some variables for next time*/
lastErrRF = errorRF;

//PIDRF.Compute();

int pwmRFOutput =(int) (OutputRF);
deneme = pwmRFOutput;
if(SetpointRF != 0)
{
    analogWrite(pwmOutPinMotorRF, pwmRFOutput);
}
else
{
    analogWrite(pwmOutPinMotorRF, 0);
}
positionForEncoderRFTemp = positionForEncoderRF;

float fActualRpmRB = (((float)positionForEncoderRB-
(float)positionForEncoderRBTemp)/(float)milliSecsSinceLastUpdate)/(float)6533)*(float)1000*(f
loat)60;
InputRB = abs((double) fActualRpmRB);
actualRpmRB = (int) InputRB;
//PIDRB.Compute();

double errorRB = SetpointRB - InputRB;
errSumRB += (errorRB * milliSecsSinceLastUpdate);
double dErrRB = (errorRB - lastErrRB) / milliSecsSinceLastUpdate;

/*Compute PID Output*/
OutputRB = kP * errorRB + kI * errSumRB + kD * dErrRB;
if(OutputRB > 255)
    OutputRB = 255;
if(OutputRB < 0)
    OutputRB = 0;

```

```

currentRB = analogRead(currentReadPinMotorRB);
if(currentRB>3500)
{
    OutputRB = OutputRB/2;
}
/*Remember some variables for next time*/
lastErrRB = errorRB;

int pwmRBOoutput =(int) (OutputRB);
if(SetpointRB != 0)
{
    analogWrite(pwmOutPinMotorRB, pwmRBOoutput);
}
else
{
    analogWrite(pwmOutPinMotorRB, 0);
}
positionForEncoderRBTemp = positionForEncoderRB;

previousMilliseconds = currentMilliseconds;
}
}

```