İZMİR KATİP ÇELEBİ UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

CONTROL AND SIMULATION OF SWARM MOBILE ROBOTS

M.Sc. THESIS

Hayrettin ŞEN

Department of Mechanical Engineering

Thesis Advisor: Assist. Prof. Dr. Fatih Cemal CAN

JUNE 2016

İZMİR KATİP ÇELEBİ UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

CONTROL AND SIMULATION OF SWARM MOBILE ROBOTS

M.Sc. THESIS

Hayrettin ŞEN
(Y130105007)

Department of Mechanical Engineering

Thesis Advisor: Assist. Prof. Dr. Fatih Cemal CAN

JUNE 2016

İZMİR KATİP ÇELEBİ ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

SÜRÜ MOBİL ROBOTLARIN KONTROLÜ VE SİMÜLASYONU

YÜKSEK LİSANS TEZİ

Hayrettin ŞEN
(Y130105007)

Makine Mühendisliği Bölümü

Tez Danışmanı: Yrd. Doç. Dr. Fatih Cemal CAN

HAZİRAN 2016

Hayrettin ŞEN, a **M.Sc.** student of İzmir Katip Çelebi University **Graduate School of Science and Engineering** student ID **Y130105007**, successfully defended the **thesis** entitled "**CONTROL AND SIMULATION OF SWARM MOBILE ROBOTS**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor: **Assist. Prof. Dr. Fatih Cemal CAN** ....
İzmir Katip Çelebi University

Jury Members: **Prof. Dr. Adnan KAYA** ....
İzmir Katip Çelebi University

Assist. Prof. Dr. Mustafa Berkant SELEK ....
Ege University

Date of Submission: 22 June 2016
Date of Defense   : 28 June 2016

To my spouse,

FOREWORD

First of all, I would like to thank to my supervisor Assist. Prof. Dr. Fatih Cemal Can who helped me very much and taught me many valuable lessons, assisted me in programming and advised me whenever I needed guidance.

I am also grateful to all my professors in Mechatronics Engineering Department for being very kind to me and let me study in the laboratories of the department throughout the research.

I also would like to thank the undergraduate students Murat Hepeyiler and Yunus Durmuş for helping me during the recording test results and the production of the robots.

I would like to thank to my parents who raised and supported me until today.

Finally, I am too much grateful to my wife for being too much patient with me, helping and supporting me during the all steps of my study.

June 2016   Hayrettin ŞEN

TABLE OF CONTENTS

ABBREVIATIONS

ABS	**:** Acrylonitrile Butadiene Styrene
LCD	**:** Liquid Crystal Display
PWM	**:** Pulse Width Modulation
PCB	**:** Printed Circuit Board
PSD	**:** Position Sensitive Detector
IR-LED	**:** Infrared Light Emitting Diode
PAN ID	**:** Personal Area Network Identifier
CH	**:** Channel
DL	**:** Destination Low Address
MY	**:** 16 bit source Address
AT	**:** Transparent Mode
API	**:** Application Programming Interface
RSSI	**:** Received Signal Strength Indication
BL	**:** Body Length

LIST OF TABLES

LIST OF FIGURES

CONTROL AND SIMULATION OF MOBILE ROBOTS

SUMMARY

This thesis presents both control of mobile robots that can move by using collective motion algorithm and simulation of the robots during the motion. The orientation of the robots was controlled remotely by one user during the collective motion. The transmission of orientation data from the remote controller to the robots was done by using XBee modules. The control algorithm of collective motion was developed by using individual-based model. Two modes are considered during the control. These modes are search and swarm mode.

The collective motion was performed by robots that are moving with respect to some pair-wise interactions. The pair-wise interactions between the robots were proposed based on three rules namely attraction, parallel orientation and repulsion fields rules. While the mobile robots try to move toward their neighbors in attraction field, they try to remain close to their neighbors in parallel orientation field. The repulsion field rule avoids the collision with each other during the collective motion.

Since the commercial mobile robots which can be used in swarm robotics are very expensive, the robots used in this study were manufactured in the Prototyping Laboratory, in Izmir Katip Çelebi University. The mechanical parts of mobile robots were designed using SolidWorks and manufactured by using 3D printer technology. Arduino Mega 2560 programmable board was used as control unit of the robots. One electronic circuit, named Arduino Shield Circuit was designed using Proteus 8 Professional. It was produced in order to connect the used electronic components to related pins on Arduino Mega 2560 easily and in a secure way avoiding short circuits.

The simulation code works as a real-time simulation. The code uses the data received from the robots to simulate the motion of the robots. The simulation also saves the all the received data from the robots to one Excel file. Two parameters, polarization and expanse were calculated in order to observe and characterize the motion of the swarm robots by using the saved data.

Lastly the collective motion was tested for a group of two, three, four and five robots. The expanse and polarization values were presented for each test.

SÜRÜ MOBİL ROBOTLARIN KONTROLÜ VE SİMÜLASYONU

ÖZET

Bu tez kolektif hareket algoritmasını kullanarak hareket eden robotların kontrolünü ve hareket halindeki robotların simülasyonunu gösterir. Robotların yönlenme açısı robotların kolektif hareketi süresince bir kullanıcı tarafından uzaktan kontrol edilir. Yönlenme açısının uzaktan kontrolcüden robotlara gönderilmesi XBee modülleri kullanılarak sağlanmıştır. Kolektif hareketin algoritması bireysel tabanlı model kullanılarak geliştirilmiştir. Kontrol sırasında iki mod dikkate alınmıştır. Bunlar arama ve sürü modlarıdır.

Kolektif hareket birbirleri arasındaki ikili etkileşimlere göre hareket eden robotlar tarafından gerçekleştirilmiştir. Robotlar arasındaki bu ikili ilişkiler çekim, paralel yönlenme ve itme alanı kuralları olarak adlandırılan üç kural üzerine tasarlanmıştır. Robotlar çekim alanında komşu robotlara doğru hareket ederlerken, paralel yönlenme alanında ise birbirlerine yakın kalmaya çalışırlar. İtme alanı kuralı kolektif hareket sırasında robotların birbirleriyle çarpışmasını önler.

Sürü robotiğinde kullanılabilecek ticari mobil robotlar çok pahalı olduğundan dolayı, robotlar laboratuvarda üretilmiştir. Robotların mekanik parçaları SolidWorks programında tasarlamış ve üç boyutlu yazıcı teknolojisi kullanılarak üretilmiştir. Arduino Mega 2560 programlanabilir kart robotların kontrol birimi olarak kullanılmıştır. Arduino Shield devresi olarak adlandırılan bir elektronik devre kartı Proteus 8 Professional programında tasarlanmıştır. Daha sonra kullanılan elektronik elemanları Arduino Mega 2560 üzerindeki ilgili bacaklara kolayca ve kısa devre oluşmayacak güvenli bir şekilde bağlamak için üretilmiştir.

Simülasyon gerçek zamanlı simülasyon olarak çalışmaktadır. Simülasyon robotların hareketini simüle etmek için robotlardan alınan verileri kullanır. Simülasyon aynı zamanda robotlardan alınan verileri bir Excel dosyasına kaydeder. Bu kayıt edilen verilen kullanılarak sürü robotların hareketinin karakterizasyonunu incelemek için iki farklı parametre olan kutuplaşma ve yayılma değerleri hesaplanmıştır.

Son olarak kolektif hareket iki, üç, dört ve beş robottan oluşan gruplar için test edilmiştir. Bu testler için kutuplaşma ve yayılma verileri gösterilmiştir.

1. INTRODUCTION

Swarm robotics is a new research field including physical robot body design, construction of robot structure and control of multi-robots systems [1-3]. Swarm robots can be considered as multi mechatronic systems interacting with each other, because, robots can be manufactured using mechanical, computer, electrical and electronics engineering disciplines. Firstly, robot design and construction of robots are directly related to mechanical engineering. Secondly, design of electronic circuits, sensors and batteries are related to electrical and electronics engineering. Thirdly, control of robot behavior via software is related to computer engineering. As a conclusion, a mixture of mentioned engineering disciplines is used to construct and control swarm robots.

Swarm robotics is a research area with potential applications such as rescue missions, constructing buildings, distributed sensing tasks, nanorobotics, micro robotics, mining tasks and agricultural foraging tasks. The most important three tasks which can be performed by swarm robots are rescue missions, mining and agricultural foraging for Turkey. On the other hand, the production of mobile robots for purpose of swarm investigations can be achieved at very small budgets.

Swarm robotics approach gets the its inspiration from the collective movement of social insects such as ants and honey bees and fish which show the three desired parameters to be achieved for multi-robots systems: *robustness, flexibility* and *scalability* [1, 3-6].

Robustness is defined as the need to have a swarm or group work continuity even under abnormal condition such as the presence of disturbances in the working environment of the robots, or the failure of some the robots of swarm. *Flexibility* can be defined as the capability of the swarm robots to find different solutions for different tasks and to adapt to different or changing needs of environment and moment. R*obustness* and *flexibility* seem to have the same meaning, but the difference between these two can be observed in problem level. As the problem changes, the swarm needs to be flexible and solve the new problem by changing the behavior of the swarm. Flexibility can be observed in biological systems, like ant colonies, for instance. They can adapt to different environments and perform different tasks such as foraging and chain formation problems with the same self-organized behavior mechanism. *Scalability* can be defined as the insensitivity of the performance of the swarm robots in terms of

number of the individuals. For example, the desired performance of the swarm should not be related with the individual number in the swarm [1, 3-6].

There are too many problems that are studied in swarm robotics. These problems can be classified into three classes as the problems based on patterns, focusing entities in environment and mixed one of the both. These classification is shown in Table 1.1.

Table 1.1 : Classification of the studied problems in swarm robotics.

The problems based on patterns	Focusing on the entities in environment	Mixed one
Pattern formation[7-9]	Searching for targets[13]	Cooperative transportation
Chain formation[10]	Foraging[14, 15]	Demining[17]
Aggregation[11]	Rescuing[16]	Exploring the planet[18]
Migration		Navigating in large areas
Coordinated movement[12]		

Since the flocking problem is the most studied problem in the swarm robotics, in this thesis flocking problem was studied as a pattern formation problem. This thesis consists of 9 parts. The first chapter of the thesis is introduction and literature review. The second chapter describes the design and manufacturing of both mechanical and electronic parts of the mobile robots. While the third part is related with calibration and configuration of the used components such as sensors, XBee modules and digital compass, in the fourth chapter the control unit and control algorithm were explained. The programming of the mobile robots was explained in detail in chapter 5. The chapter six present both the simulation of mobile robots and the data taker programming. The characterization parameters, polarization and expanse, of the swarm were explained in the chapter 7. In the chapter 8 and 9, the performed test results and conclusion of the study were presented respectively.

1.1 Literature Review

The swarm robotics and swarm simulation studies started to be investigated around 1980s. One of the first swarm simulation was created by Craig Reynolds [19] in 1987. When this computer simulation was created, this type of collective motion was rarely seen in computers. However nowadays, simulations of collective motion are very popular and widely spread.

The collective motion of fish schools was investigated by Inada [20]. The effect of variation of preferred direction was analyzed with his model. His simulation consists of three rules namely attraction, parallel orientation and repulsion.

Strömbom proposed a collective motion model including as a single rule attraction [21]. On the other hand, three different phases were generated by his model. These phases are swarm, undirected mill and moving aligned groups. Model of Strömbom shows that attraction alone can produce many of the patterns which are seen in simulation with alignment. Furthermore, the simulations of collective motion are proposed by using elastic springs between nearby individuals [22, 23].

Oboshi [24] carried out one computer simulation of prey-predator system. He observed the behavior of a fish swarm escaping from a predator. His simulation was compared with the behavior of real swarm of fishes. Two new methods are presented for direction sensing of a robot swarm in order to perform some applications that include landmine detection and firefighting by Venayagamoorthy [25]. The first method indicates an embedded fuzzy logic approach in the particle swarm optimization algorithm. The second one presents a swarm of fuzzy logic controllers.

Castro [26] improved a tool that has strategies for a hunting game between predators and prey by using particle swarm optimization. Based on emergent behavior, this tool was designed in three dimensional environments.

Development of simulations on collective motion has also caused new technological advances as the collective motion of robots, for instance. The collective motion exhibited by animals is applicable to control robotic swarms for specific tasks. There are already many examples of robotic swarms which are controlled by collective motion algorithms and different modeling types.

One of the first swarm robotics study was carried out by Fukuda et al. [27] as a distributed robotic system that had separable mobile robots. These mobile robots were able to communicate with each other. Fukuda et al. experimentally presented that these mobile robots were able to connect and separate with each other automatically to construct a manipulator.

Atyabi et al. [28] designed a robotic swarm which was navigated by a simulation that has two phases, training and testing. The training phase consisted in the participation of agents in survivor rescuing missions as a team. In the test phase, performance of the agents was improved by using the obtained knowledge in training phase.

Swarm robots were controlled by using wireless sensory network and multi mobile robot approach in the study of Lee and Shen [29]. In their study six and twelve individuals were used to simulate swarm behaviors.

3

Fredslund and Mataric [8] investigated motion of four mobile robots as a pattern formation problem using local sensing and minimal communication. In their study the robots were moving without knowing the position or heading of neighbor robots, besides the information regarding one of the neighbor robots.

Ijspeert et al. [30] studied the collaboration of a group of simple reactive robots for stick pulling problem. The task of the robots, which required collaboration of two robots, was to pull a stick out of the ground. In their study 2 to 6 robots were used.

Turgut et al. [31] produced mobile robots named Kobots. The flocking problem using seven mobile robots was investigated. The movement of Kobots was also simulated in computer. The Kobots have two important properties. The first one is short range sensing system that can measure the distances from obstacles and kin robots. The second one is VHS (virtual heading system) that has a digital compass and a wireless communication module for sensing the relative headings of neighboring robots.

Trianni et al. and Trianni et al. [32, 33] investigated the motion of a swarm of robots called *s-bot*. In their study the robots had to explore an area avoiding falling into the holes in the area. The robots avoided falling into the holes due to their ability to connect and disconnect with each other.

Bahçeci and Şahin [7] developed a 3D simulator for an aggregation problem on a swarm robotics system. In the simulator the motion of the simulated robots was studied with different parameter settings.

In the literature there are several methods of control modeling and two types of communications between robots [1, 6].

Table 1.2 : The used modeling types and communication between the robots.

The Modeling Method	Sensor-Based Modeling [7, 11, 34]
	Microscopic Modeling [30, 35, 36]
	Macroscopic Modeling [37-39]
	Cellular Automata Modeling [40]
The Communication Between The Robots	Interaction via Sensing
	Interaction via Communication

Sensor based modeling is the most commonly and the oldest modeling method used in swarm robotics applications. Sensor based modeling method considers the sensors, motors and the objects in the environment as the main components of the system. After modeling these components, interaction of the robots between objects in the environment and each other are modeled [1].

Microscopic modeling is a modelling method which models the interactions both robot to robot and robot to environment individually. In this method all cases for all events are modeled for each robot.

Macroscopic modeling takes the swarm robots as a whole system. Macroscopic modeling models the whole behavior of the swarm directly.

Cellular automata modeling is the simplest mathematical model of swarm robotics system. This model contains discrete lattice of cells in one or two dimensions where each cell in the lattice has finite number of possible states. Each cell interacts only with the neighbor cells and the system dynamics are characterized by the local rules performed locally on the cells in discrete time steps.

The robots which are studied in swarm robotics applications, and specifications of these robots are shown in Table 1.5. In this thesis sensor based modeling method was used in order to program the robots.

In this thesis, firstly Original Arduino Robot was considered to be used for swarm control application. The Original Arduino Robot has no more than one communication port to connect any wireless communication modules. Besides, the commercial mobile robots for swarm applications are very expensive. Because of these two reasons, the robots which were used in this thesis were produced by the author. The specifications and classification of this robot are shown in Table 1.3 and in Table 1.4.

Table 1.3 : Developed mobile robot specifications.

Size (mm) (dia.)	Actuators (differential drive)	Computing capabilities	Sensors	Communication	Relative positioning system	Development /Production Cost
130	Wheeled	AtMega 2560 MCU	8 IR	XBee S1	IR Based	Research/ 250€

Table 1.4 : Classification of swarm robots in this thesis.

Axis	Description	This Thesis
Collective size	Number of robots in the collective	5 (max)
Communication range	Maximum communication range	300 mm
Communication topology	Of the robots in the communication range, those which can be communicated with	IR sharp sensor
Process ability	The computational model used by the robots	Distributed aggregation model
Collective composition	Are the robots homogenous or heterogeneous	Homogenous All robots are completely same

Table 1.5 : The used robots in the swarm robotics [4].

Name	Size(mm) (diam.)or(*l* x *w*)	Actuators	Computing capabilities	Sensors	Communication	Relative positioning system	Development/ Price (If commercial)
Khepera [41]	55	Wheeled (differential drive)	Motorola MC68331	8 IR	RS232 Wired link	-	Research
Khepera III [42]	120	Wheeled (differential drive)	PXA-255 (400 MHz) Linux and dsPICs	11 IR 5 ultra sound	WIFI and Bluetooth	IR based	Research/ 3200 €
e-puck [43]	75	Wheeled (differential drive)	dsPIC	11 IR Contact ring Color camera	Bluetooth	IR based	Research/ 850 €
Alice [44]	20 x 20	Wheeled (differential drive)	Microchip PIC	IR proximity and light Linear camera	Radio (115 kbit/s)	-	Research
Jasmine [45]	23 x 23	Wheeled (differential drive)	2 AtMega microcontrollers	8 IR	IR	IR based	Research
S-Bot [46]	120	Wheeled (differential drive)	XSclae (400 MHz) Linux PICs	15 Proximity OmniCamera Microphone	WIFI	Camera based	Research
Kobot [3, 31, 47]	120	Wheeled (differential drive)	PXA-255 (200 MHz) and PICs	8 IR Colour camera	XBee	IR based	Research
SwarmBot [48]	127 × 127	Wheeled (differential drive)	ARM (40 MHz) and FPGA 200 kgate	IR, light sensors Contact, camera	IR	IR based	Research

2. DESIGN AND MANUFACTURING OF MOBILE SWARM ROBOTS

2.1 Design Criteria and Component Descriptions

Design and manufacturing of robots consists of two parts. The first part is the design and production of mechanical parts. The second part is the design and manufacturing of Arduino shield circuit board. Some criteria were considered during the design of robots. These criteria are the robot speed, the robot size and the ability of robots to perform certain tasks. Design criteria of robots are ordered as follows:

- Robot speed should be 5 cm/s to 10 cm/s.
- Robot size should be 130-200 mm diameter and circular shape.
- The robot is able to detect walls or obstacles while it is moving.
- The robot is able to calculate its rotation angle with respect to North while it is rotating.
- The robot is able to transmit and receive data to/from the other robots.
- The robot is able to operate without stopping for at least one hour.

All the components such as motors, sensors, wheels, ball casters, motor brackets and the other circuit components were chosen using the design criteria of the robots. The mechanical parts of robots were designed according to these components.

2.2 Components

We can divide all the used components and units in to three groups. These groups are mechanical components group, control unit of robots and the other electrical components and sensors group. The mechanical components and their specifications are shown in Table 2.1 while the rest of the components will be shown in electrical design section.

Table 2.1 : Mechanical components.

Component Name	Quantity	Specification	Figures
Electrical DC motor with gearbox (100:1 Micro Metal Gear motor HP (320 Rpm))	2	Transmission ratio 1:100 Speed 320 rpm at 6V DC Current Free run current is 80 mA, Stall current is 1600mA and Stall torque is 1.8 kg-cm.	
Wheel	2	Diameter of wheel is 32mm, thickness of it is 6.5mm	
Plastic Motor Brackets	2	This component was used to attach the motors on the base plate.	
Ball Casters	2	This small ball caster uses a 9mm diameter metal ball. This component was used to balance the robot. The ball casters were placed bottom of the robots symmetrically.	

2.3 Design and Manufacturing of Mechanical Parts

The robots were designed in such a way as to be able to carry all the components required for their motion and control. They consist of three main mechanical parts, the base plate, top plate and battery holder. All the mechanical parts were designed in SolidWorks. The 3D models of the other components were also inserted in the assembly drawing in order to check the compatibility of all the pieces.

After assessing the compatibility of all the components, the mechanical parts of the robots were produced by using 3D printer U-print SE. The material used by the 3D printer is Acrylonitrile butadiene styrene (ABS).
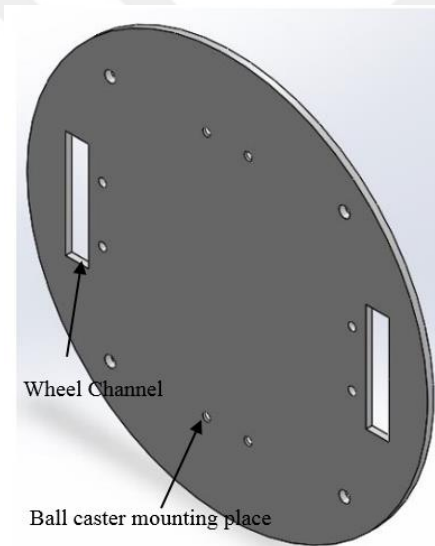
2.3.1 Base plate and battery holders

The base plate (Figure 2.1) has a diameter of 130 mm and 2 mm thickness. Since the biggest component of the robot was battery, robot sizes were determined according to

the battery. On the other hand eight analog distance sensors were needed, therefore the other affecting factor on the robot size is the number of the analog distance sensors.

The base plate can be named as chassis of the robot. It carries the motors, the wheels, the ball casters, top plate and the battery with the battery holders. It has twelve holes for bolts to fix ball casters, motor brackets and plastic rods which hold the top plate. The base plate also has two openings for the wheels. These openings and holes were designed according to the dimensions of the motor brackets, wheels and ball casters.

The battery holders consist of two parts. These two parts have the same size, but one of them has two openings on its corners for the battery cables. The battery holders are designed according to the dimensions of the batteries. The battery holders (Figure 2.2) have two rods on their bottoms in order to fix on the bolts of the ball casters on the base plate.

Figure 2.1 : Base plate.
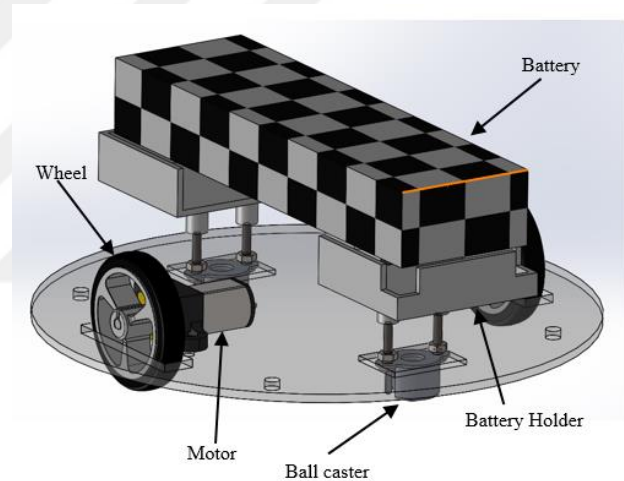
Figure 2.2 : Battery holders.

2.3.2 Top plate

The top plate (Figure 2.3) has same size as the bottom plate. The top plate carries Arduino Mega 2560 with designed Arduino shield circuit. The top plate has eight protrusions on bottom surface in order to connect the analog distance sensors to the robot. These protrusions were placed at 45° with respect to each other. Two openings were provided on the top plate for connection wires to pass. After the connection wires of motors and analog distance sensors pass through these two openings, they are connected to the Arduino Shield circuit. Since no cables pass in front of the analog distance sensors, they give more accurate measurements.

After all the mechanical parts were designed, they were assembled (Figure 2.4) in SolidWorks for checking the compatibility of parts.
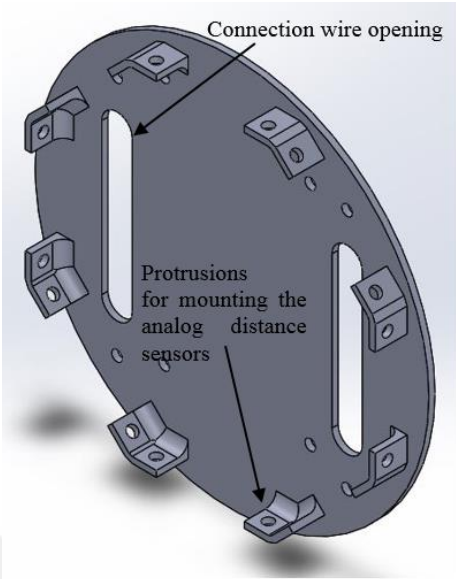
Figure 2.3 : Top plate.
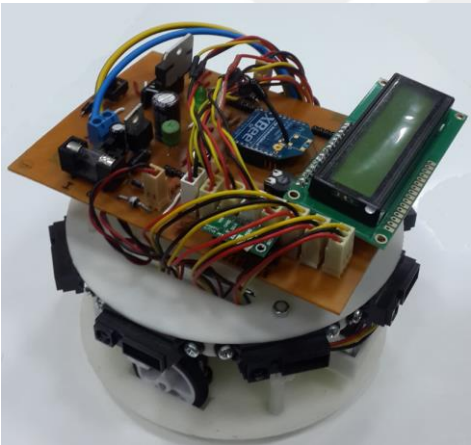
Figure 2.4 : Final assembly.   **Figure 2.5 :** Final CAD assembly.
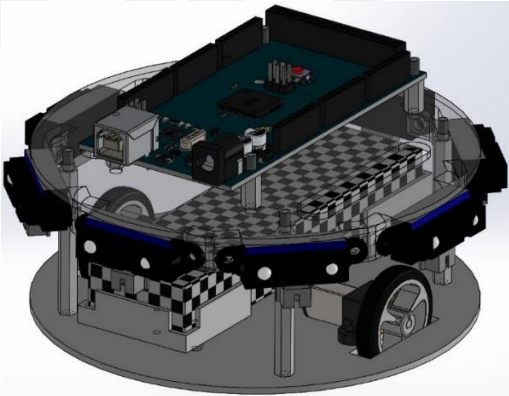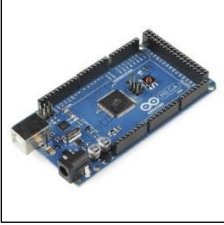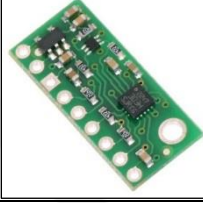
2.4 Design and Manufacturing Arduino Shield Circuit

Although the main control unit of robots is Arduino Mega 2560, it cannot be used without additional electrical circuit. Arduino shield circuit was designed to connect easily all the connection wires of electrical components (Table 2.2) and sensors on Arduino Mega 2560. Also it was designed in such a way as to avoid short circuit and open circuit that can happen accidentally during the movement. Therefore connections will be very stable while the robot is moving.

Arduino shield has four sub-units. These are voltage regulator unit, motor driver unit, wireless communication unit and digital compass module. All these units were tested on the breadboard separately before the whole design of the circuit.

Table 2.2 : Electrical components.

Component Name	Quantity	Specification	Figures
Arduino Mega 2560	1	Control Unit of the robots. It has 16 analog input pins and totally 54 digital I/O pins of which provide 15 PWM output.	
Analog Infrared Distance Sensor (Sharp Sensor)	8	Sensors perform distance measurements in a range 4-30 cm.	
Digital Compass (LSM303D 3D Compass and Accelerometer)	1	It is used to calculate the angle between North and its orientation.	
XBee S1 Wireless Communication Module	1	XBee, 1mW Series 1 Wire Antenna, 2.4 GHz operating frequency, 100 m Communication Range.	
LCD Screen	1	16x2 character LCD (Liquid-crystal display) screen	
Li-Po Battery	1	7.4V and 3050 mAh Li-Po battery is used for the Robots. The Size of battery is 117x32x16mm.	

LM2576 (Figure 2.6) is a voltage regulator that can regulate up to 40V input voltage to 5V. In the shield circuit it regulates from 7.4 to 8.4 Volt of battery voltage to 5 Volt. This regulated +5V was used for the analog distance sensors and LCD screen as an input voltage. The L298N (Figure 2.7) including two H-Bridges was used in the circuit

as a motor driver with 15 pins and two channels. It allows to control motor speed and rotation direction of motor. Operating supply voltage ranges from 5V to 46V and up to 2 Ampere current for each channel. In this study Arduino PWM outputs were used as inputs of L298N motor driver.
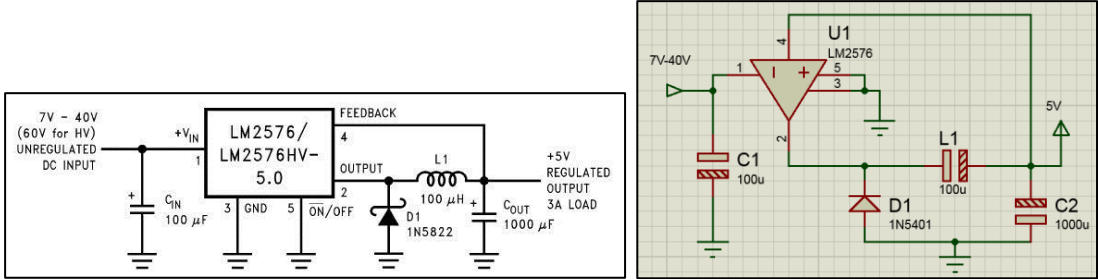
Figure 2.6 : LM2576 fix +5V DC output circuit.
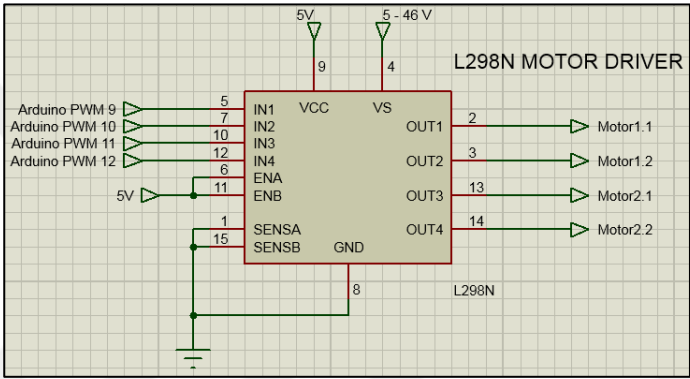
Figure 2.7 : L298N motor driver.

XBee S1 wireless antenna was used for wireless communication among the robot or between the robots and remote controller. Although it has 20 pins, only its 4 pins were used in the circuit. These pins are ground, data in (Rx), data out (Tx) and 3.3V input voltage. In the circuit these four pins were connected (Figure 2.8) to the pins of Arduino Mega 2560 ground, Tx3, Rx3 and output 3.3V respectively. The other pins of XBee are analog and digital input output pins which allow to send analog or digital data to other XBees without any microcontroller.
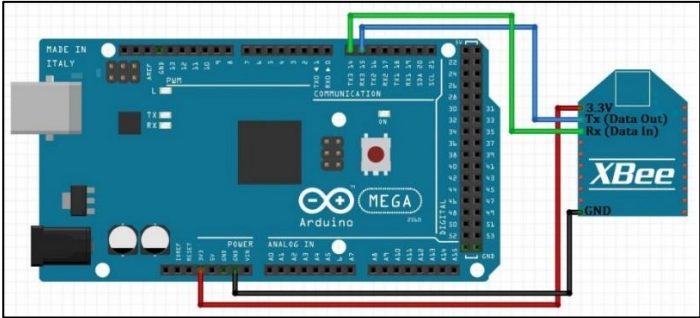
Figure 2.8 : XBee S1 wireless antenna connection to Arduino Mega 2560.

LSM303D is a system-in-package containing a 3D digital linear acceleration sensor and a 3D digital magnetic sensor. The LSM303D digital compass and accelerometer was used in order to sense the angle of orientation with respect to North. In the other words, the output of the LSM303D digital compass gives the orientation of the robots. Although it has 9 pins, only 4 of these 9 pins were used in the circuit to read the compass data (Figure 2.9). The LCD screen which can show 16x2 characters, was used to see the different type of data such as analog distance sensor value, received and transmitted wireless data and compass data.
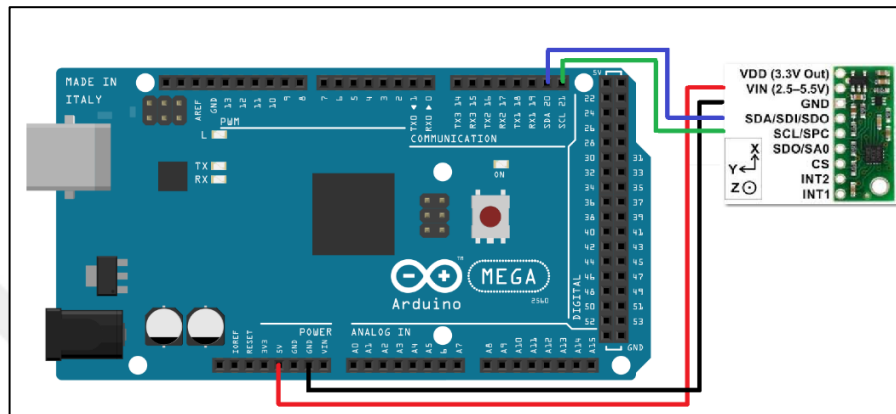
Figure 2.9 : LSM303D connection to Arduino Mega 2560.

The analog distance sensors have only three pins as ground, 5V input and output. These pins were connected to Arduino Mega ground, 5V output of Arduino and analog input of Arduino respectively. It gives output voltage in a range of 3.3V – 0.3V.

According to all these Arduino connections Arduino shield circuit was designed in a circuit schematic design and PCB (printed circuit board) layout drawing software Proteus 8 Professional. Firstly schematic design of the shield (Figure 2.10) was done according to appropriate connections between the pins of electrical components and sensors to Arduino Mega Pins. Then the schematic design PCB layout drawing was performed. Since some of the components such as XBee module and digital compass may interact with each other while they are working, they were placed in the circuit in such a way to prevent interference from happening.
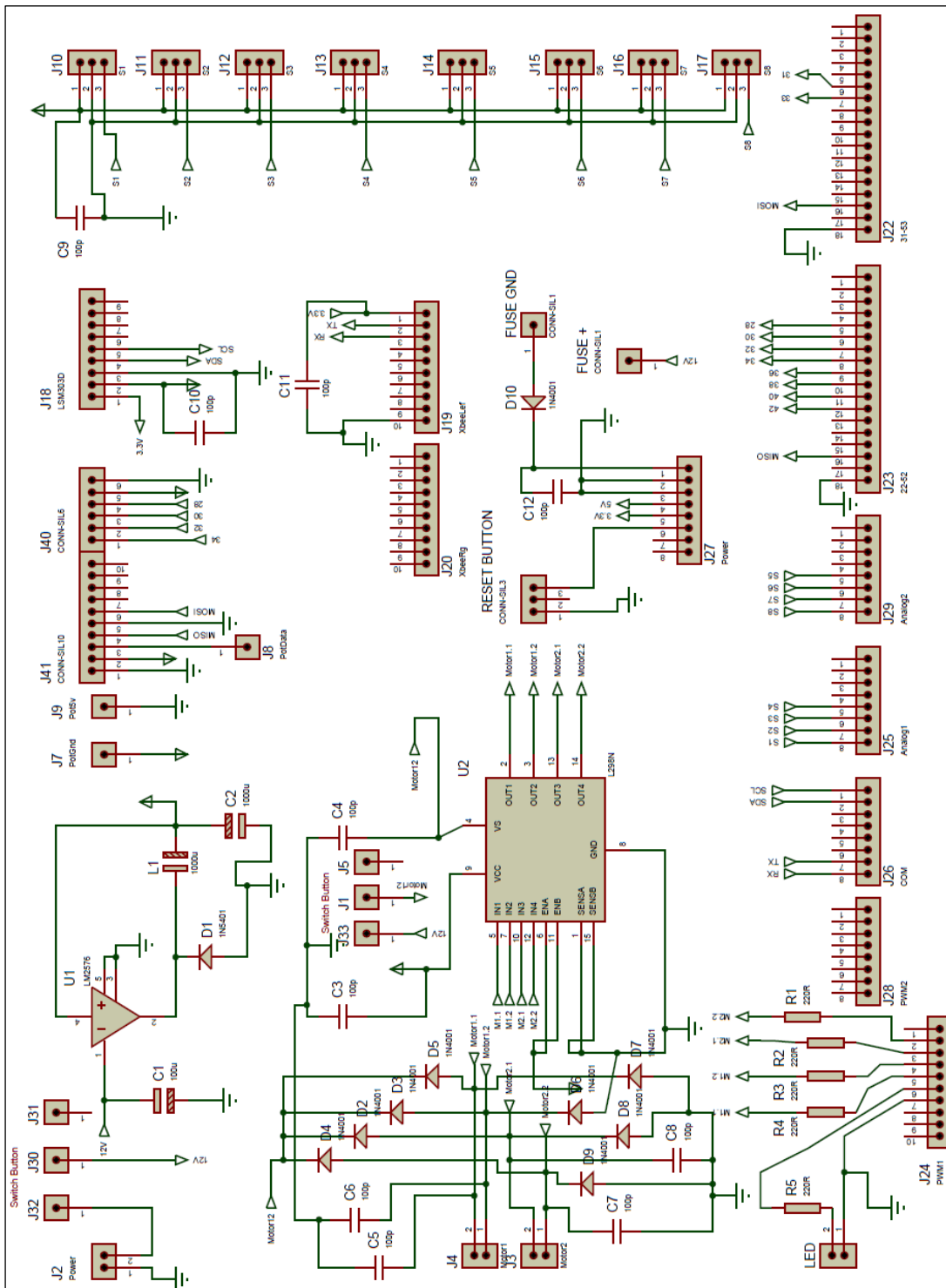
Figure 2.10 : Schematic design of shield.

After the design PCB layout (Figure 2.11) was printed on a special paper. Then the paper was put and ironed on the copper plate to transfer printing from paper to the copper plate. The printed copper plate was put in a solvent which contains HCl and H_2O_2 in order to dissolve the unprinted copper area. After drilling holes on the plate, the electrical components were soldered (Figure 2.12) to the plate.
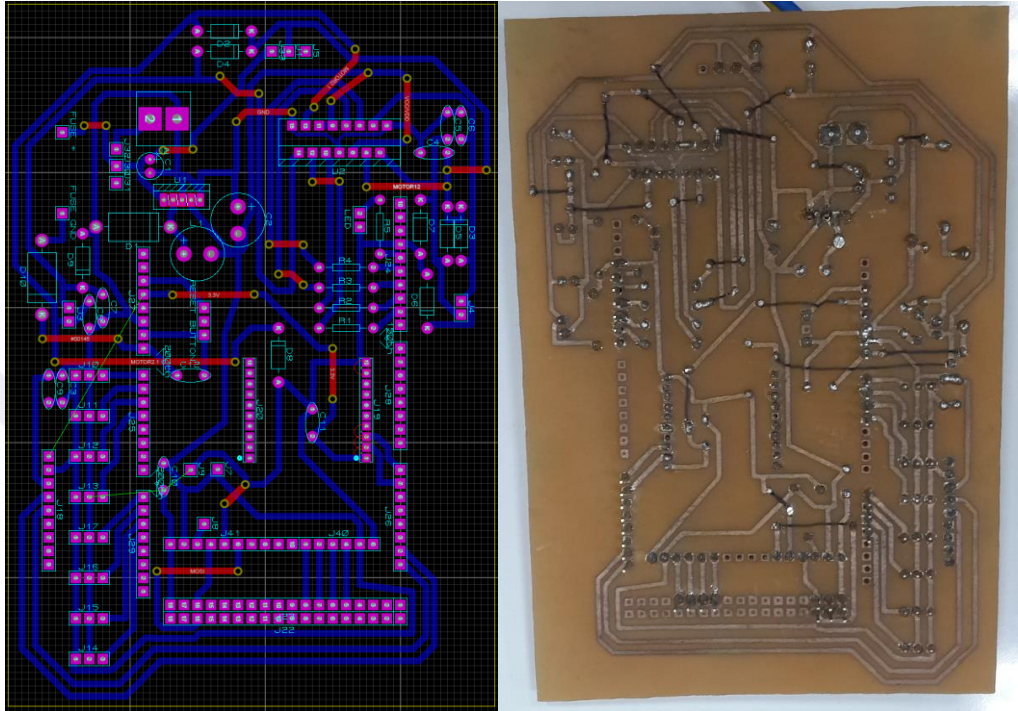
Figure 2.11 : PCB layout of the Arduino Shield.
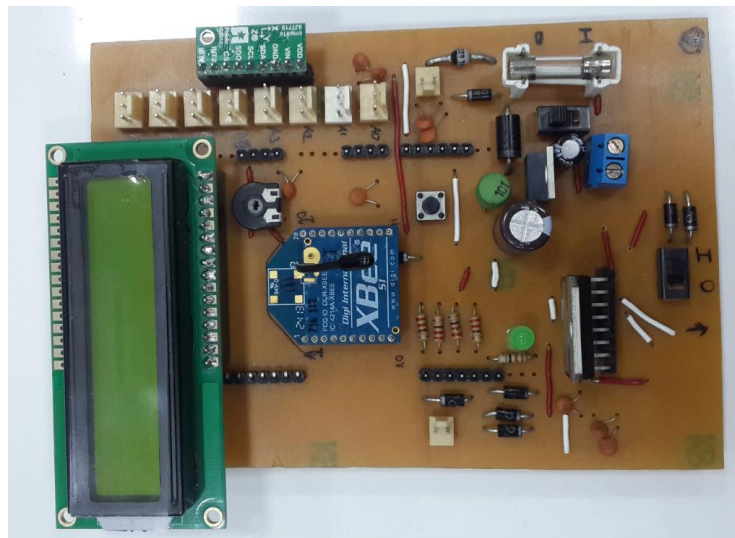
Figure 2.12 : The last view of the Arduino Shield.

3. CALIBRATIONS OF SENSORS AND CONFIGURATIN OF XBEE S

3.1 Calibrations of Sensors

In this study two different type of sensors (the analog distance sensors and digital compass) were used. The analog distance sensors contain an integrated combination of PSD (position sensitive detector), IR-LED (infrared emitting diode) and signal processing circuit. Analog distance sensors send an IR light to the object. After IR light reflects from the object, it reaches a certain place (Figure 3.1) on PSD and sensor gives an output voltage in a range of 3.3V- 0.3V according to place where IR light reaches.
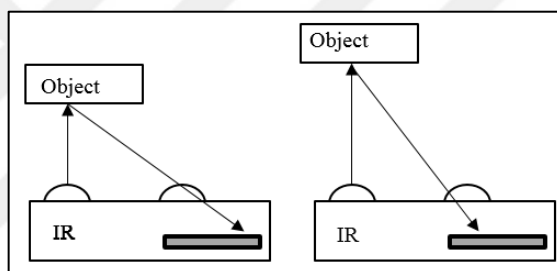
Figure 3.1 : The working principle of the analog distance sensors.

The output pin of the analog distance sensor was attached on one of the analog input pins of Arduino Mega 2560 in order to read sensor output. Arduino analog pins give an analog output value between 0 and 1023 which changes from 0 V to 5V. Since sensor does not give directly the distance output, the output value of sensor should be converted to meaningful distance output. The analog output values were recorded from 40 mm to 250 mm, in every 10 mm in order to convert to analog output values to the real distances in terms of millimeters. These recorded values are given in Table 3.1. The software Mathematica was used to find a curve fit function for the recorded data. This function calculates the distance in terms of millimeters by using the analog output of the sensor as an independent variable. A fifth degree nonlinear function (Figure 3.2) was obtained due to the characteristic of the sensor. The correlation between function and data was very good as shown by the value R^2 as well. This obtained function was used as a sub converting function from analog output to millimeter in written control codes of the robots.

Table 3.1 : Analog output values of analog distance sensor.

Analog Output	Real Distance (mm)	Analog Output	Real Distance (mm)	Analog Output	Real Distance (mm)
520	40	197	120	117	190
438	50	185	130	113	200
382	60	169	140	105	210
329	70	158	150	100	220
290	80	145	160	96	230
263	90	138	170	93	240
236	100	130	180	88	250
216	110				

$$y = 595.121 - 6.392x + 0.0360895x^2 - 0.000109734x^3 + 1.67994 \times 10^{-7}x^4 - 1.01098 \times 10^{-10}x^5$$
$$R^2 = 0.99881$$
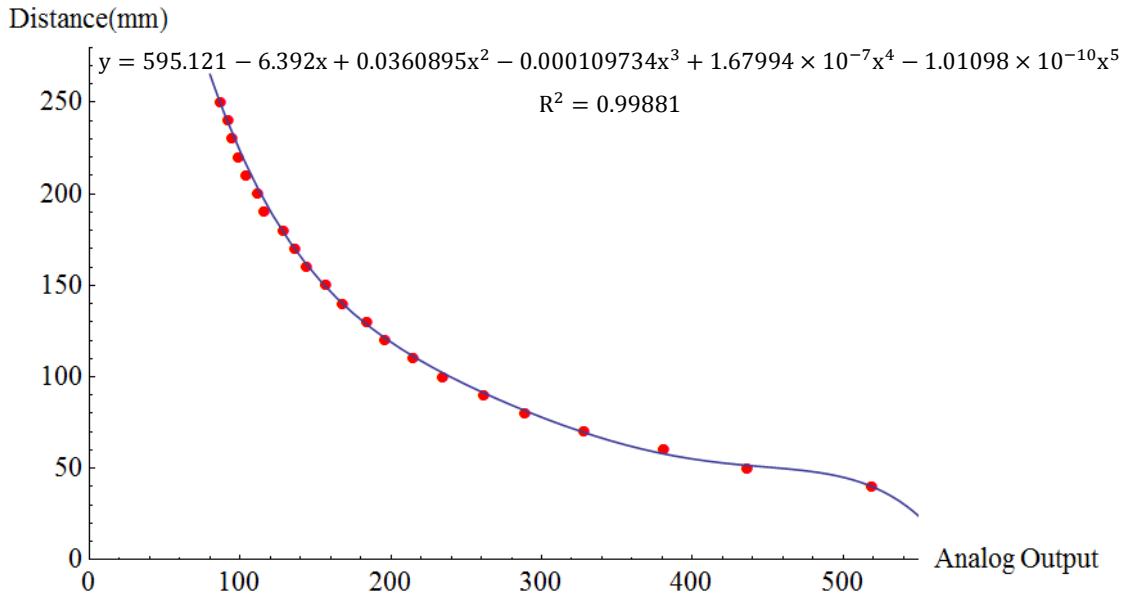
Figure 3.2 : Calibration function.

The LSM303D Arduino library was used in order to read the output data of the sensor. Although Arduino library was used, the digital compasses needed be calibrated. The digital compass gives two different integer type output value for each axis as a maximum and a minimum value of axis between -32767 and +32767 according to the North. The default maximum and minimum values in the used library for each axis are defined +32767 and -32767 respectively. Since magnetic field changes from one location to another location on the Earth, also the digital compass maximum and minimum output values show an alternation according to the location of the compass on the Earth. Therefore the digital compass maximum and minimum output values must be read and replaced the default values. After calibration code (Figure 3.3) was uploaded to Arduino Mega, the digital compass was connected to Arduino Mega.

18

While the digital compass were rotating randomly on the each axis for all angle possibilities, the maximum and minimum output values were observed on the serial monitor (Figure 3.4) of Arduino Mega 2560. The serial monitor of Arduino was observed until the maximum and minimum output values for each axis stop changing. When the output values were stable, these output values were recorded and defined in the control code as maximum and minimum compass values.

Figure 3.3 : Calibration code of digital compass.

(a) During the calibration. (b) After the calibration.

Figure 3.4 : The output of the digital compass

19

3.2 Configuration XBees by Using X-CTU

XBee S1 is a wireless antenna which uses 802.15.4 networking protocol for communication in 2.4 GHz operating frequency. XBees have two type of communication modes, AT mode and API mode. These modes and the other communication settings of XBee are configurable. X-CTU software was used to perform all these configurations. X-CTU is a Windows-based application which allows to change PAN ID (Personal Area Network Identifier), destination low (DL) address, channel (CH), 16 bit source address (MY) and communication mode of XBee. XBee should be connected to the computer in order to perform this configuration. Since XBee cannot be connect to computer directly, a tool which is called XBee adapter (Figure 3.5) is needed for serial communication between XBee and computer.

Figure 3.5 : XBee adapter.

Two XBees should be configured in such a way that they have the same PAN ID and CHANNEL for communication between each other for all communication mode.

AT mode is synonymous with "Transparent" mode. The use of this mode is simpler than API mode. In AT mode, any data sent to the XBee module is immediately sent to the other module identified by the destination low address in memory of the sender XBee. In this mode data package preparation is not needed, only simple send serial data to the Transmitter (Tx) of one XBee and it will be received by the Receiver (Rx) of the destination XBee. Because no packages are created the destination address and type (only-data) are both fixed. In AT mode 16 bit source address (MY) of the first XBee should be configured as destination low address of the second XBee. At the same time 16 bit source address of the second XBee should be configured as destination low address of the first XBee (Table 3.2). AT mode is used to set communication only between two XBees. Therefore AT mode is not an appropriate communication mode for the larger network.

Table 3.2 : XBee configuration for AT mode.

First XBee Parameters		Second XBee parameters	
CH	C	CH	C
PAN ID	1001	PAN ID	1001
DL	5	DL	10
MY	10	MY	5

Another communication mode for the XBee is called API mode, for *application programming interface*. Instead of sending or receiving the data alone, the entire frame is manually constructed for transmission and manually parsed on reception. In API mode data must be formatted in frames with destination information and payload. The frame consists of sender's 16 bit source address, RSSI (Received Signal Strength Indication) level, options, frame IDs, and the data or message itself. In the API mode channels and PAN ID of the XBees must be same as well. But 16 bit source address and destination low address can be configured randomly as distinct from AT mode. As frame has destination address which is the MY address of the receiver XBee, receiver XBee can be identified in the frame without the need for configuration of XBees. Therefore API mode is useful for larger wireless communication network, multiple data receiving and sending. In this study API mode was used for the communication of XBees due to its advantages and flexibility.

In this study, communication Channel and PAN ID of the XBees were selected (Figure 3.6) as 'C' and 1001 respectively. The MY address of the robots was selected as 1. While MY address of the remote controller was defined as 2, MY address of data taker XBee on the computer was configured as 8. These MY addresses of the XBees were selected randomly, because the MY address of the data transmitting XBee can be defined in the prepared data frame in Arduino Mega the by the user.
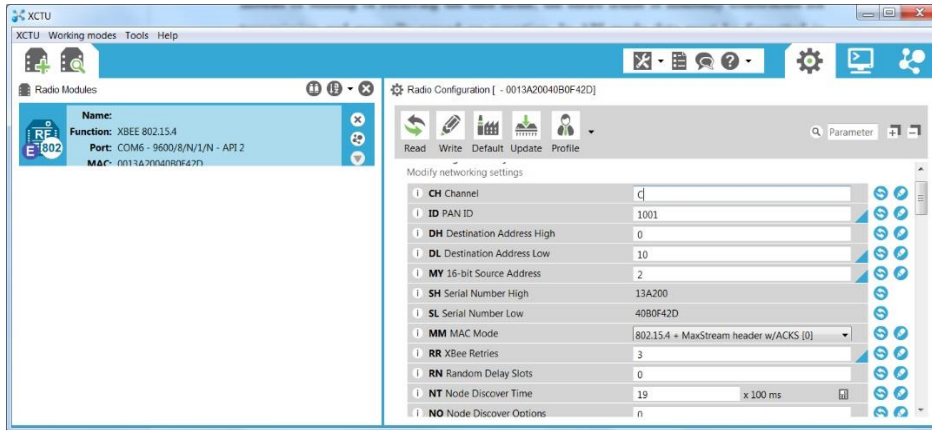
Figure 3.6 : X-CTU XBee configuration window.

3.3 XBee Communication Network

There are three type of communication network topology (Figure 3.7) such as star, tree and mesh topology [49]. In all networks there should be one coordinator which set the network and relay the messages among the other member of the network. The number of the router and end devices of the network can be changed according to size of the network. The routers are responsible for routing traffic between different nodes. End devices do not route the data traffic between the nodes. They can move in the network and rejoin directly the coordinator or another router. End devices send data such as sensor data or any numerical value to the routers or coordinator.

(a) Star	(b) Tree	(c) Mesh
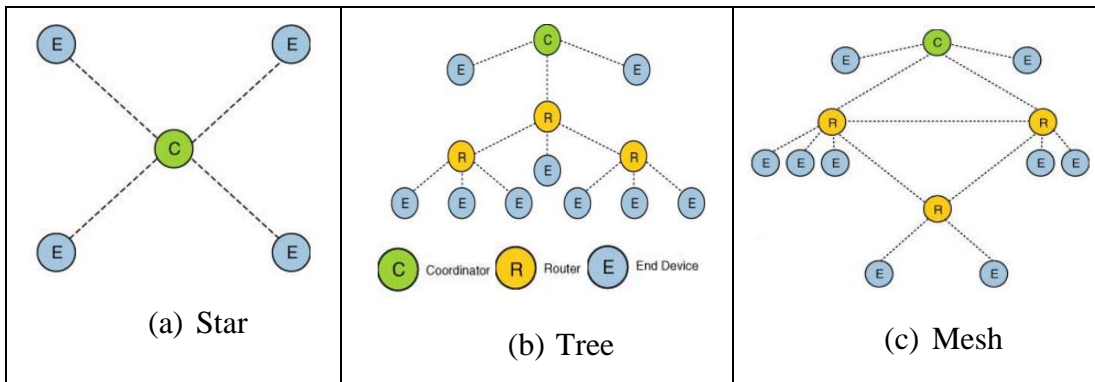
Figure 3.7 : Network mesh topology of XBee [49].

XBee S1 wireless communication modules allow to set up only star topology network. The other network topologies are usable and appropriate for the other XBee or ZigBee wireless communication modules. In this study, while the robots and remote controller were configured as end devices, data taker XBee on PC was configured as a coordinator. In the network MY addresses of the robots, remote controller and data taker were configured as 1, 2, and 8 respectively. The main reason that the same MY

addresses were given to the robots was to avoid the occurring time difference during the robots are receiving data from the remote controller. The orientation data was sent by the remote controller during the movement of the robots. After the robots received the orientation data, they move according to the receiving orientation data and then robots send their own orientation and velocity values to the data taker on PC.

In the API mode, the received data contains the address of the data transmitting XBee. When the data taker receives the data, the addresses of the data are going to be same. Therefore the data taker XBee cannot distinguish the data where it comes from. The solution of this problem are going to be explained in section 5.4.
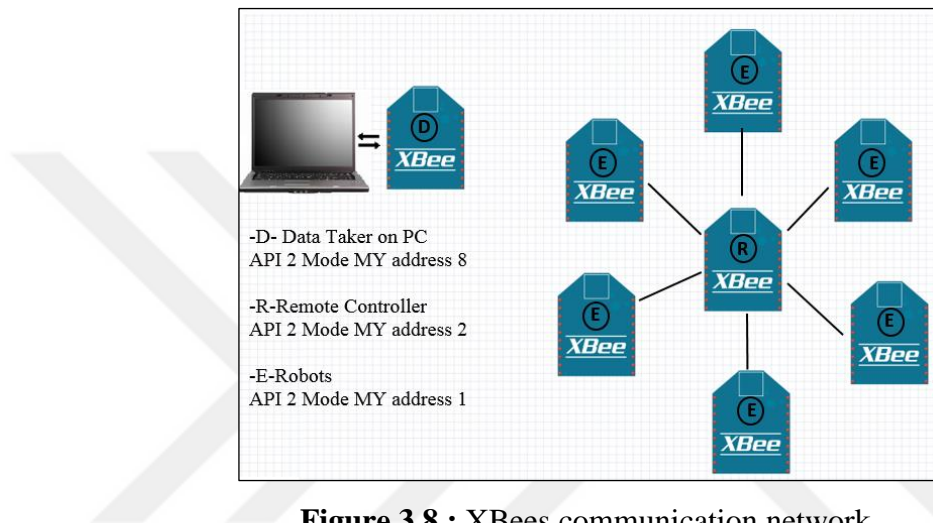
Figure 3.8 : XBees communication network.

In this thesis, two wireless communication line were created between XBee-s. The first communication line was created between the robots and remote controller. The second one was created between the data taker and robots (Figure 3.8). In Figure 3.8 E, R and D represent the mobile robots, remote controller and data taker respectively.

4. CONTROL UNIT AND CONTROL ALGORITHM

4.1 Control Unit of the Robots

The Arduino Mega 2560 is a programmable board based on microcontroller Atmega 2560 chip. Although there are several Arduino programmable board for this study, Arduino Mega 2560 was used as the control unit of the robots due to the fact that Arduino Mega 2560 has too many pins. It has 54 digital input output pins of which 15 can be used as PWM (Pulse Width Modulation) outputs, 16 analog inputs which allow analog reading, 4 serial communication port and a USB connection port. The number of the analog input pins and communication pins were the reason why Arduino Mega was used for controlling of the robots. The eight analog input pins were needed for obtaining analog data from the analog distance sensors. On the other hand it is needed at least one more communication port for the XBee connection besides Tx0 and Rx0 communication port of Arduino because this communication port is used for programming Arduino microcontroller chip. If the zeroth (Tx0 and Rx0) communication port of chip is connected to XBee or any other devices that can be used for serial communication with Arduino, the user cannot upload the program to Arduino chip. If the zeroth communication pins are used for serial communication with any devices, the user have to unplug the device from these pins before uploading the code to the Arduino. The XBee was connected to third communication port (Tx3 and Rx3, Serial3) of Arduino Mega for avoiding this unplugging procedure, since Arduino Mega have 3 more serial communication port besides the zeroth port.

The Arduino Mega can be powered via USB or by any external DC power suppliers. For the external powered there are two options. The first one is that the power supply such as a battery can be plugged into the power jack of board. The second option is that battery positive and negative poles can be inserted in V_{in} (input voltage) and GND (ground) pin headers of Arduino Mega respectively. The recommended supply voltage range for the second powered option is between DC (direct current) 7V and 12V by the producer of Arduino Mega. In this study the second option was used to supply power to Arduino Mega. The used battery voltage which is between 7.4V and 8.4V was supplied to the V_{in} pin of Arduino Mega.

The Arduino Mega board can be programmed by using Arduino Software. The Arduino Software allows to write a code and upload this written code to Arduino Mega

or any type of Arduino products such as Arduino Uno and Arduino Nano. The Arduino Software is based on C programming language. There are several libraries written in C or C++ for sensors and other devices, which can be inserted in the Arduino Software. By using these libraries, reading data from the sensors, or usage of some special other modules (such as XBee and digital compass) becomes an easier task.

4.2 Mechanism of Swarm Behavior

Mechanism of swarm behavior was developed according to pairwise interactions between the robots (Figure 4.1). The pairwise interactions between the robots were created based on three rules, namely attraction, parallel orientation and repulsion field[20, 50]. The robots try to follow some rules during the movement. These rules are:

- In the attraction field, the robots try to get closer until they reach the parallel orientation field.
- In the parallel orientation field the robots try to move in the same direction and keep the distance between each other constant.
- In the repulsion field the robots try to move away from the each other until they reach the parallel orientation field.
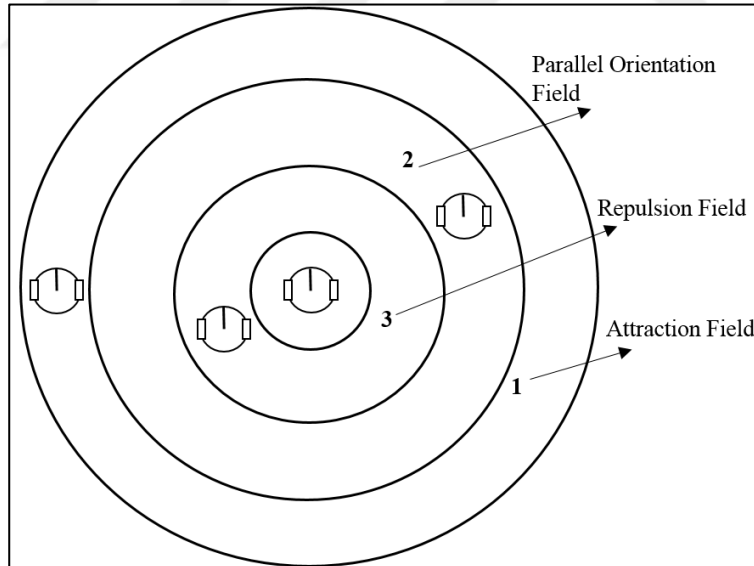
Figure 4.1 : Mechanism of swarm behavior.

The radii of these three fields were defined in the control code of the robots as a parameter. Also these radii values can be changed by the user in the control code. The distances between the robots were measured by the analog distance sensors on the

robot. The robots compare the radii of the fields with the sensing the distances between the robots. After the comparison the robots decide the movement type.

4.3 Control Algorithm

The control algorithm has four steps. These steps are as follows;

- Read the orientation data from the digital compass,
- Get the direction data from the remote controller,
- Read the distances from the analog distance sensors,
- Go to direction or behave according to neighbor robot.

The control algorithm of the robots is based on the control of the motors. The robots get the distance data from the sensors, the orientation data from the digital compass and the direction data from the remote controller by using XBee before signal was sent to the motors.

The control of the motors was performed by sending PWM signal from Arduino Mega to the L298N motor driver. The PWM signals allow to obtain analog output from the digital outputs. Digital control is used to form a square wave signal that can be switched on (5V) and off (0V). In order to obtain a voltage value between 5V and 0V this on and off pattern can be simulated by changing the portion of time the signal spends on (5V) during the period of the square wave signal. The time duration of 5V is named pulse width. By changing the pulse width, varying analog output values can be obtained.

In the Arduino, the period of the square wave signals is 2 millisecond. In order to get analog output from the PWM output pins *analogWrite(x)* command was used on the range between 0-255 for "*x*". For example the analogWrite(127) is a 50% pulse width which gives 2.5V output voltage (Figure 4.2).

The obtained analog outputs were used as input for the L298N motor driver. The L298N motor driver gives the analog voltages as outputs to the motors according to the input voltages. Therefore the speed of the motors can be changed during the movement by the using PWM outputs.
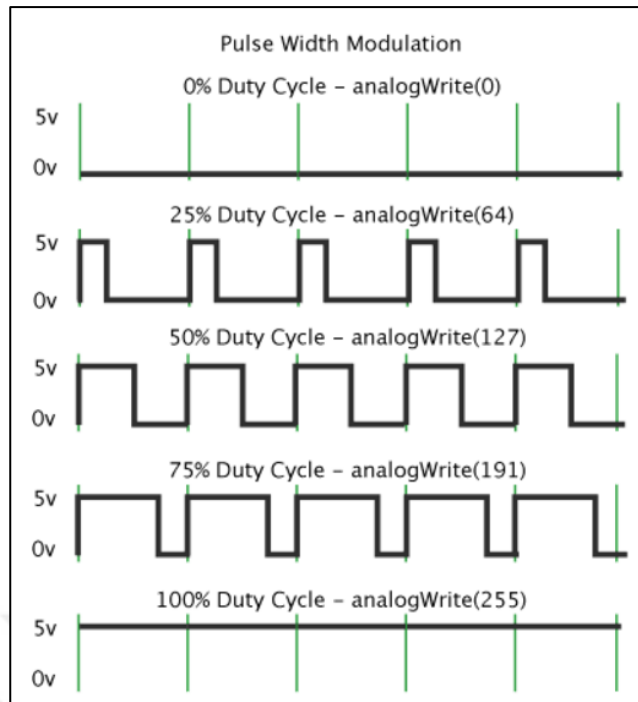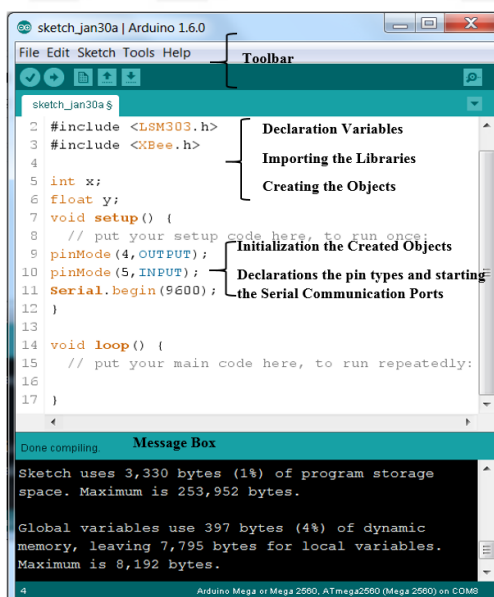
Figure 4.2 : Pulse Width Modulation [51].

5. PROGRAMMING THE MOBILE ROBOTS

5.1 The Used Programming Software

Arduino Software (IDE-integrated development environment) was used to write the control code of robots. The Arduino software is written in Java. It contains a text editor in order to write codes, a message box which can show the errors while compiling or uploading the codes, a toolbar with buttons for common functions and a series of menus. The software can connect to Arduino boards and upload the written codes to them.

The software consist of two main parts, void setup and void loop. The commands like declaration of the pin types, starting the serial communication ports or initialization of the created objects which runs only one time are written in the void setup. The commands which run repeatedly are written in the void loop. Declaration of the variables, importing the libraries and creating the objects are written before the void setup (Figure 5.1).

Figure 5.1 : Structure of Arduino software.

The software allows to upload the written code to Arduino boards. Before the code is uploaded to the boards, the appropriate ones are needed to be selected in the Board section, which is in the Tools menu on the toolbar. At the same time, the connection

port between Arduino board and computer is required to be selected in order to upload the code, otherwise software gives error during uploading. The connection port also can be selected from the Tools menu on the Port section.

5.2 Converting the Control Algorithm to Programming Code

The control algorithm was converted to the code by using the Arduino Software. In the code, four different libraries were used in order to make the programming easier. These libraries are XBee.h, LSM303.h, Wire.h, and LiquidCrystal.h. The first two libraries were downloaded from the web page of the used XBee and digital compass. The other ones were already inside the Arduino software. The XBee library was used to prepare the data package which is send to other XBee-s. Also it was used to convert the received data to meaningful data.

The digital compass uses I^2C (Inter-Integrated Circuit) communication protocol, which uses the SCL (Serial Clock Line) and SDA (Serial Data Line) pins of Arduino Mega. In order to use this communication protocol, the Wire.h library is required. LSM303.h library was used to directly obtain the orientation angle (θ) of the robot (Figure 5.2). The LiquidCrystal.h library was used to write data to the LCD screen, such as orientation angle of the robot, received data or obtained distance value from the analog distance sensors.

After all the libraries were imported to the control code, ten different sub-function were written. These sub-functions were called in the void loop when the functions were needed. The sub-functions are going to be explained in section 5.3.

The control parameters of the robots are shown in the Figure 5.2. These parameters are the sensed North direction (n_s), forward velocity (u) and velocity of the left and right motors (V_L and V_R).
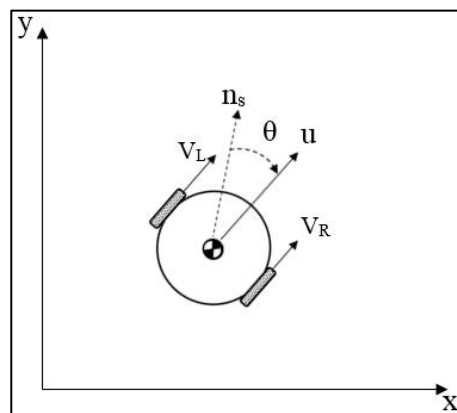
Figure 5.2 : Control parameters of the robots.

5.3 Sub-functions in the Control Code

After the libraries were imported, three different objects were created for XBee, digital compass and LCD screen respectively. The creation of these objects was needed due to the structure of the libraries. These created objects are *xbee*, *compass* and *lcd* by using the XBee.h, LSM303.h and LiquidCrystal.h libraries respectively.

The required variables were declared after the creation of the *xbee, compass* and *lcd* objects. The created objects, the used Serial Port which is Serial3 for the XBee and the used output pins which are from 9 to 13 of PWM pins were initialized and declared in the void setup. Also one integer variable which is '*k*' was added for each robot. Since all robots have the same MY address, when the data taker receives the data from the robots, it is not able to distinguish the source of the received data. Therefore, the *k* variable was used as an identifier for each robot. Afterwards the obtained calibration data of each digital compass, which is the maximum and minimum compass values, were attached to each robot by using the *k* variable with **if** command. On the other hand the *k* variable was used to code the sending orientation angle and velocity data from the robots to data taker on PC. Therefore the data taker can distinguish which robot is transmitting the received data.

After assigning the address to each robot, the written sub-functions were used in the void loop. The written sub-functions are ordered as follows:

- *double f (int x),*
- *void ReadSensors( ),*
- *void RobotmotorsWrite(int x, int y),*
- *void senddata(int x),*
- *void getdata( ),*
- *void orientation(int degree, int mxx),*
- *void parallelorientation(int mxx),*
- *void ra(int x),*
- *void rr(int x),*
- *void LCDprint( ).*

Some of these sub-functions were used inside other sub-functions.

5.3.1 double f(int x)

This sub-function takes one integer value. The sub-function puts the value inside the fifth degree polynomial function (Figure 3.2) obtained from the calibration of the analog distance sensors. The output of the function is the real distance, measured by the sensors, in mm.

5.3.2 void ReadSensors()

This sub-function does not take any argument. In this sub-function one *for* loop was used in order to read quickly the analog outputs of the sensors. As soon as the analog output was read, it was converted to millimeters by using the function *f(int x)*. Then, the obtained real distance value was attached to one element of the s[ ] array (Figure 5.3), which was declared before the void setup. The elements of the s[ ] array were used to check the distances in the other parts of the control code.
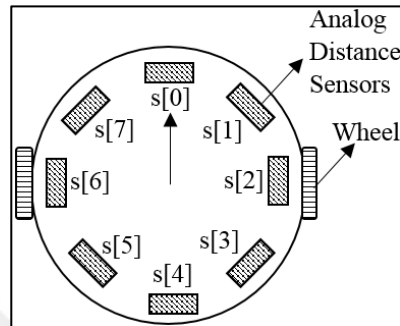
Figure 5.3 : Representation of the sensors in the code.

5.3.3 void RobotmotorsWrite(int x, int y)

The sub-function takes two arguments. These arguments are the sending PWM signals to the left and to the right motor respectively. Two PWM output signals are needed to run forward and backward each motor. Therefore totally 4 PWM outputs were used for two motors. While the PWM 9 and 10 were used to run the left motor, the PWM 11 and 12 were used to run the right motor. When one PWM signal was sent to one pole of the motor, zero PWM signal has to be sent to the other pole of the motor.

5.3.4 void senddata(int x) and void getdata( )

The XBee can send only 255 bytes of data. In order to send a variable or data of size greater than 255, the variable or data is divided into two parts and attached to an array that contains two elements. This array is called *payload[]*. The first element of the array (payload[0]) is equal to the integer part of the division of the sent data to 256. The other element is the mode of the sent data with respect to 256. For example the sending integer data is 350. The elements of the payload[] are going to be 1 and 94 respectively. Then, the data package is prepared. The prepared data package contains the MY address of the data transmitting XBee, payload[] array and the size of payload array. After the preparation, the data package is sent. All these operations were performed within the function void senddata(int x).

In the void getdata() sub-function, data is received. The MY address of the data transmitting XBee and RSSI value of the data are received with the data package itself. The pure data contains two elements because the sender XBee divides the data into two parts. In getdata sub-function, these two parts of data were merged again to obtain the original data. The first element of data was multiplied with 256 and summed with the second element of data. In this way, the original data was obtained.

5.3.5 void orientation(int degree, int mxx)

This sub-function takes two variables, which are desired direction angle and velocity of the motor. The first variable is the receiving data from the remote controller. The second one was defined in the code. The desired direction angle and orientation angle at that moment were shown in the Figure 5.4 as α and θ respectively. The void orientation sub-function changes orientations of the robots from the current orientation to desired direction.
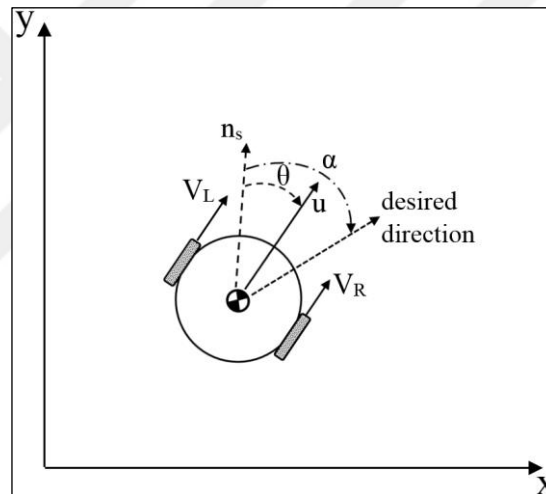
Figure 5.4 : Control parameters of the robots.

The working principle of the void orientation sub-function is explained as follows;

- The function calculates the angle difference between θ and α.
- The function checks shortest route to reach the desired direction.

The function determines the velocity of the motors according to the shortest route. For example if the desired direction is near the right side of the robot, the function increases the velocity of the right motor, while the velocity of the left motor is decreased. The change rate of velocity is dependent on the angle difference between θ and α. The effect of angle difference (ϕ) on the movement of the robot is shown in the Figure 5.5. If $\phi \approx 0°$, the movement of the robot is governed by translation. If $\phi \approx 90°$, the

33

movement of the robot is governed by rotation. If $\phi > 90°$, the movement of the robot is governed by rotation.
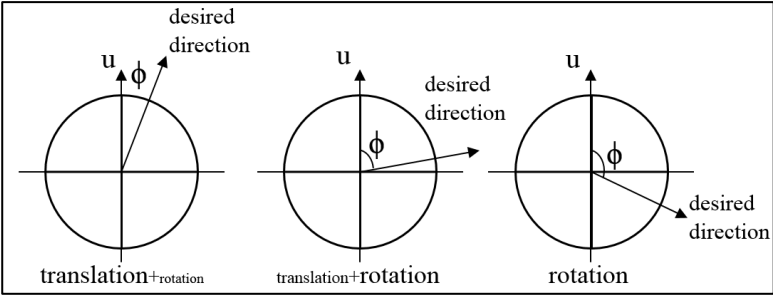
Figure 5.5 : The working principle of the void orientation.

5.3.6 void parallelorientation(int mxx)

This sub-function takes only one variable, the velocity of the motors. In this sub-function four different borders were defined, namely mdmin, border2, border1 and mdmax. These borders define the radius of the attraction field, parallel orientation field and repulsion field respectively (Figure 5.6).
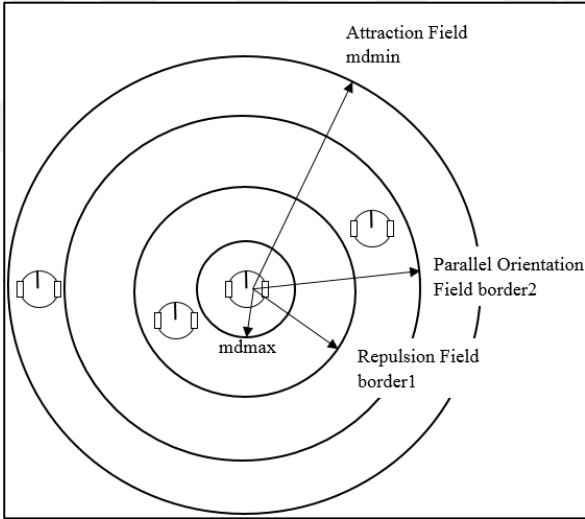
Figure 5.6 : The borders of the fields.

This sub-function starts with the void orientation function by using the received angle data from the remote controller. Then the distance data were taken from the analog distance sensors in order to sense the robots. The robots behave according to the measurements of the sensors. If one of the measurements of the s[1], s[2], s[6] or s[7] is between mdmin value and border2 value, the robots try to move towards each other until the measurements of the sensors are between border2 and border1. The movement of the robots towards each other was performed by increasing the velocity of one the motors. For example if s[1] or s[2] is in the range of attraction field, the

velocity of the left motor is increased. If the frontal sensor measurement (s[0]) is between the mdmin and border2, the robot get closer to the detected robot increasing the velocities of both motors, until the s[0] reaches a value between border2 and border1.

If one of the measurements of s[1], s[2], s[6] or s[7] is between mdmax value and border1 value, the robots try to move away from each other until the measurements of the sensors are between border2 and border1 values. The drifting movement of the robots was performed by increasing the velocity of one of the motors. For example if s[1] or s[2] is in the range of repulsion field, the velocity of the right motor is increased. If s[4] is in the range of repulsion field, the robot move away from the other robots behind by increasing the velocities of both motors, until s[4] reaches a value between border2 and border1.

If s[3] or s[5] is in the range of the repulsion field, the robot moves away from the other robots behind by increasing the velocities of both motors, until s[3] or s[5] reaches a value between border2 and border1.

5.3.7 void ra(int x) and void rr(int x)

In the void parallelorientation sub-function the velocity of the robots were increased according to the cases. These situations were explained in the section 5.3.6. The change rate of the velocities of the motors were calculated in the *void ra* and *void rr* sub-functions. When the robots were in the attraction field, void ra function was used in order to calculate the velocity of the motors. The velocity of the motors can increase up to two times the normal velocity that is mxx value. The change rate is the same for void rr function as well. The void rr function was used in order to calculate the velocity of the motors, when the robots were in the repulsion field.

In the code mxx was declared as 60, that is the sending PWM signal to the motors. The borders mdmin, border2, border1 and mdmax were declared as 290 mm, 160 mm, 110 mm and 60 mm respectively. In the void ra function the sending PWM signals were calculated using a linear function that gives maximum and minimum PWM signals when the distances are equal to 290mm (mdmin) and 160mm (border2) respectively.
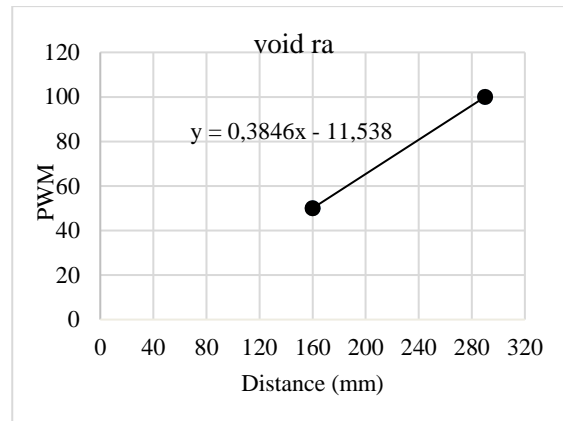
Figure 5.7 : Calculation of the PWM signals in the void ra.
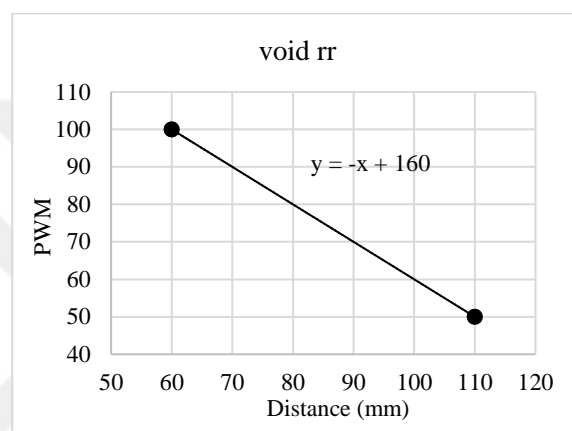
Figure 5.8 : Calculation of the PWM signals in the void rr.

5.3.8 void LCDprint()

This sub-function was used to show the orientation angle of the robot, received angle data or measured distances on the LCD screen. By using this sub-function a lot of data can be written on the LCD screen. The main reason of the usage of the LCD screen and sub-function was to crosscheck the measurements and the other data.

5.4 Sending Data to the Data Taker

The robots send their orientation and velocity to the data taker on PC. Since two data needed to send either two different data packet is required to prepare and send with a certain time delay or one data packet which contains two data can be prepared and send. In the first option sending two data packet caused the overloading to the data taker. Therefore the second option was used to send data to data taker. In order to prepare one data packet the two data was converted to one integer data. Since the addresses of the XBees, one identification parameter is needed to be added in the data packet. During the procedure one **k** variable was also used to identification of the data

36

packet for each robot. The prepared data is a five-digit number. The first digit shows the *k* variable. The three digits after the first digit show the orientation data. The last digit shows the velocity of the robots.
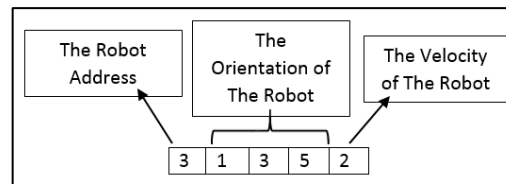
Figure 5.9 : The prepared data packet.

The velocity of the motors is 320 Rpm @ 6V. Since maximum voltage can be 5V for the motors, the velocity of the motors decreases to 267 Rpm @ 5V. This velocity was converted to the rad/s by multiplying $2\pi/60$. Then obtained velocity was converted to the mm/s by multiplying the radius of the wheel (16 mm). The 447.4 mm/s was obtained as the velocity of the robots when the 255 PWM signal was sent to the each motor. In this study, used minimum and maximum PWM signals were 50 and 100. The velocity of the robots was calculated as average of the velocities of the motors.

The last digit was calculated as the ratio of the current velocity of the robot to minimum velocity of the robot. The ratio is going to be a float number between 1 and 2. This ratio was multiplied by 10 and two digits integer number was obtained. These ten two-digit numbers were converted to one-digit number from 1 to 9 by using map function in the Arduino. Lastly the last digit is going to be between 1 and 9. This mapping procedure was performed in order to send an *int* data to the data taker.

For example, orientation of the second robot 325° and sending PWM signals to the motors are 60 and 80 respectively. The average velocity is 70 and the ratio is 1.16. After the ratio is multiplied by ten, 11 is obtained. After the using *map* function of the Arduino Software, the last digit of the data packet is obtained as 2. The total data is going to be '23252'. Lastly this prepared data is sent to the data taker on PC.

5.5 Programming the Remote Controller

The Arduino Shield circuit was used for the remote controller as well. The remote controller has XBee and digital compass. The all libraries were used again in the remote controller code. The remote controller reads the current orientation data from the digital compass and sends it to the robots. Therefor the robots try to move with the same orientation angle of the remote controller.

6. PROGRAMMING THE DATA TAKER AND SIMULATION

In this study four different codes were written, the control code of robots, code of the remote controller, code of the data taker and simulation code. The first and second code were explained in section 5. In this section the code of the data taker and simulation code are going to be explained.

6.1 Data Taker and Programming the Data Taker

One Arduino Mega 2560 board and one XBee S1 module were used as data taker. The data taker was connected to the computer for serial communication. After the data taker gathered the information from the robots, it sent this information to the computer by using serial communication. The received data by the computer were used in the simulation as inputs.

Table 6.1 : Output of the Map function.

The last digit of the Data	1	2	3	4	5	6	7	8	9
The Converted Two Digit Number	10	11	12	13	14	15	16	17	18

The robots send five-digit number to the data taker as explained in section 5.4. While the first digit shows the number of the robot, the three digits in the middle show its orientation. The last digit of the data represents the velocity of this robot. As a first step, this last digit was converted to a two digit number between 10 and 18, since the original velocity data which is the ratio of the motor velocities, is in this range. For example the received data is '30533'. It shows that the orientation of the third robot is 53°. The last digit is converted to 12 by using *map* function (Table 6.1). Then the real velocity of the robot is obtained by multiplying 5*447/255 with this two digit number. The data '30533' means the velocity of the third robot is 105 (12*5*447/255) mm/s.

After the data taker receives the real velocity and orientation data for each robot, it produces a seven digit number which contains velocity and orientation data of one robot to send to the simulation (Figure 6.1). The first digit is the address of the robot. While the first three digits of the remaining digits show the orientation data, the last

39

three digits show the velocity of the robot in mm/s. Since the data that is greater than 255 byte cannot be send over the serial connection, this seven digit number was sent as a string to the simulation. These seven digit numbers were sent to simulation separately for each robot.
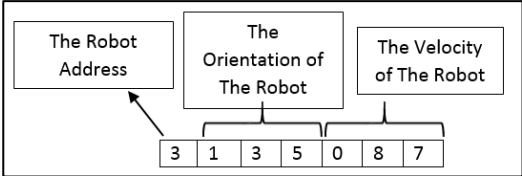
Figure 6.1 : The sending data to the simulation by data taker.

6.2 Programming the Simulation

The first aim of the written simulation was to observe the collective motion of the robots by using the received robot data from the data taker. The second one was to save the received data to one Excel file in order to observe the behavior of the swarm in terms of polarization and expanse values. These two terms are going to be explained in section 7.

The simulation code was written in C#. One circle and one line were used to show each robot. In the simulation interface two textbox were used to see received orientation and velocity data for each robot (Figure 6.2). There are two buttons on the simulation interface, named *Stop Communication* and *Export to Excel*. When the simulation is stopped or closed, it gives error since the serial communication is not closed properly. For this reason, firstly the serial communication between simulation and data taker is required to be stopped. The *Stop Communication* button was used to stop serial communication between data taker and simulation. After closing the serial communication, orientation, velocity and position data of the robots can be saved in Excel file by using *Export to Excel* button.
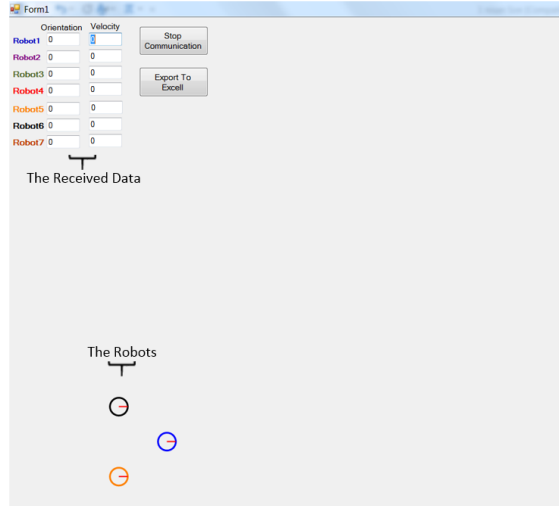
Figure 6.2 : The simulation interface.

In the simulation, each robot starts to move at a certain position that is defined in the code. There is a determined distance of 100 twips between every two robots at the beginning of the simulation. When the data taker sends the data to simulation, the robots start to move by using the received orientation and velocity data. The simulation firstly separates into three parts the received seven digit number. After the separation, the obtained velocity and orientation data for each robot are used to calculate the positions of the robots in the simulation. A very simple mathematical model was used in the simulation in order to calculate the positions of the robots [20, 50].

Time was increased at time steps of 50 milliseconds. The new time is calculated by adding the time step to the previous one as,

$$t = t + \Delta t \qquad \text{(6.1)}$$

Where **t** is time and **Δt** is time step.

$$x_j(t) = x_{j_{@(t-\Delta t)}} + \Delta t \, v_j(t) \, \cos \theta_j(t) \qquad \text{(6.2)}$$

$$y_j(t) = y_{j_{@(t-\Delta t)}} + \Delta t \, v_j(t) \, \sin \theta_j(t) \qquad \text{(6.3)}$$

Where $x_j(t)$ and $y_j(t)$, $v_j(t)$ and $\theta_j(t)$ describe position, velocity and orientation of the robot **j** at time **t**, respectively.

During the simulation all the orientation, velocity and position data of the robots were stored in an array for each robot. These data were saved in *output* Excel file before the simulation was closed. These data are going to be used for calculation of the polarization and expanse values of the swarm.

7. CHARACTERIZATION OF SWARM

Two parameters were calculated in order to observe the characterization of the swarm robots. These parameters are polarization and expanse. Polarization p, is an average of the angle differences between the orientation of the each robot and movement direction of the swarm [50]. The polarization was calculated by using the saved data in Excel output file using Equations 7.1 and 7.2;

$$\text{Pol}_{av}(t) = \left[\sum_{j=1}^{n} \theta_j(t)\right]/n \tag{7.1}$$

$$\text{Pol}(t) = \left[\sum_{j=1}^{n} \text{Pol}_{av}(t) - \theta_j(t)\right]/n \tag{7.2}$$

Where $\text{Pol}_{av}(t)$, $\text{Pol}(t)$ and $\theta_j(t)$ show the movement direction of the swarm, polarization of the swarm and orientation of the jth robot respectively.

Expanse a, is defined as the arithmetic distance average between each robot and center of the swarm [50]. The expanse was calculated using Equations 7.3, 7.4 and 7.5:

$$X_{av}(t) = \left[\sum_{j=1}^{n} x_j(t)\right]/n \tag{7.3}$$

$$Y_{av}(t) = \left[\sum_{j=1}^{n} x_j(t)\right]/n \tag{7.4}$$

$$a(t) = \left[\sum_{j=1}^{n} \sqrt{\left(X_{av}(t) - x_j(t)\right)^2 + \left(Y_{av}(t) - y_j(t)\right)^2}\right]/n \tag{7.5}$$

Where $X_{av}(t)$ is the horizontal position of the center of the swarm, $Y_{av}(t)$ is the vertical position of the center of the swarm and $a(t)$ is the expanse value of the swarm. $x_j(t)$ and $y_j(t)$ are horizontal and vertical position of jth robot at time **t**.

The movement of the robots can be shown by using grayscale representation (Figure 7.1). The motion of the robot can be determined by contrast of the arrows. The contrast

of arrows gradually increases from the beginning to the end of the motion. This illustration is going to be used to observe polarization and expanse values of swarm for a certain time interval.
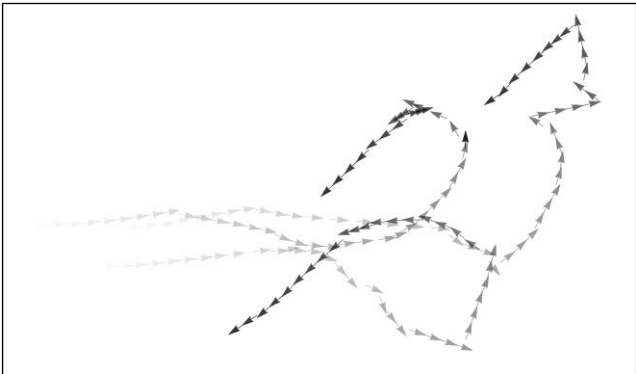
Figure 7.1 : Grayscale representation of the motion.

8. PERFORMED TEST RESULTS

The collective motion of the robots was tested for a group of two, three, four and five robots. The expanse and polarization values are presented for each test in this section.

(a) Polarization

(b) Expanse

(c) Grayscale representation
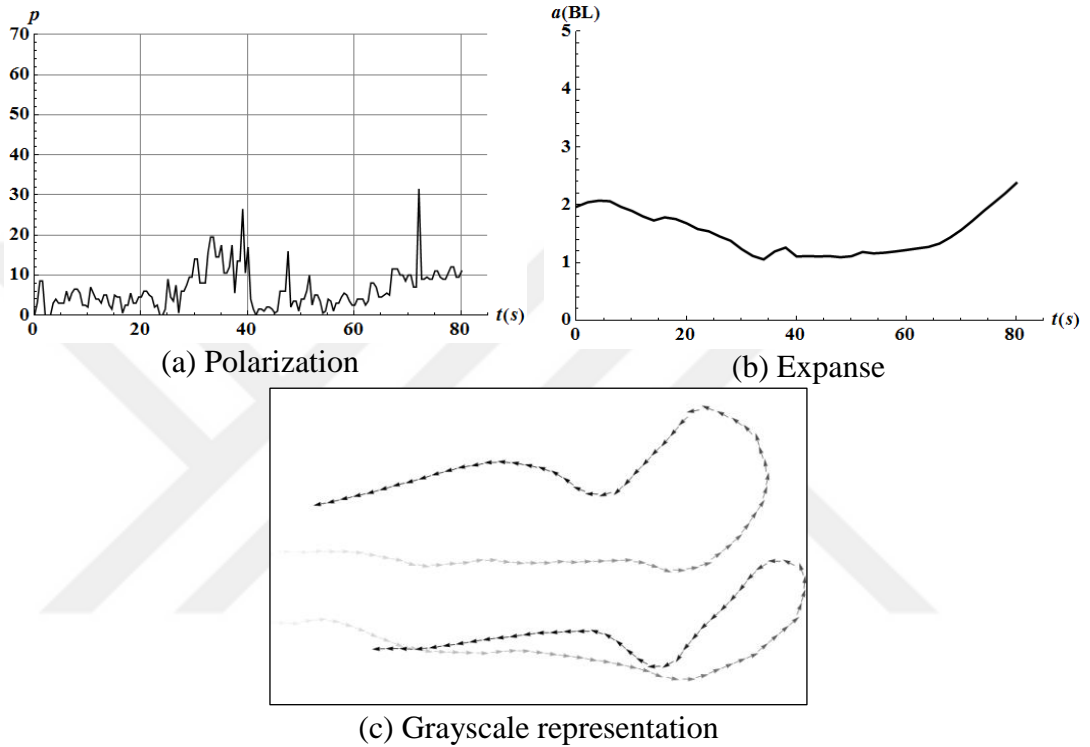
Figure 8.1 : The test results of swarm with two robots.

The motion of the swarm with two robots was recorded for 85 seconds and the polarization and expanse values were presented in Figure 8.1. The polarization values can change between 0° and 90° during the motion [50]. If p is equal to 90°, the swarm is maximally confused. If p is equal to 0°, the swarm is optimally paralyzed (polarized). It can be seen that the polarization reaches a maximum value of 31.5°, while the average polarization during the motion is 6.5°. This shows the swarm is polarized and the robots are in the parallel orientation field during the motion which is the desired behavior. If the robots stay together during the motion the expanse has a value of 1 BL [50]. The average expanse during the motion has a value of 1.5 BL. This shows the robots stay together during the motion. Figure 8.2 shows the positions of the robots during the motion for every 10 seconds.
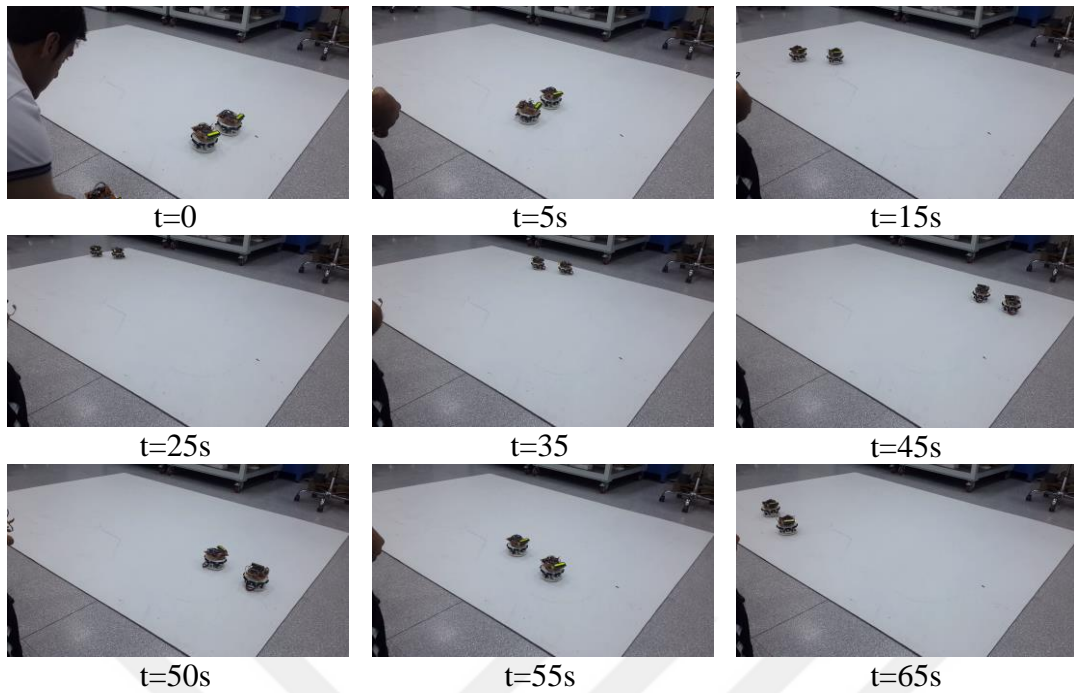
Figure 8.2 : The positions of two robots every 10 second during the motion.

The motion of the swarm with three robots was recorded for 100 seconds and the polarization and expanse values are shown in Figure 8.3. While the maximum polarization value is 72° for one second, the average polarization has a value of 15.5° during the motion. The average expanse has a value of 1.48 BL while the maximum expanse value is 2.45 BL. These results show the robots stay together but not as much as the swarm with two robots. The grayscale representation of motion is shown for two different time periods. These time periods are 0s − 40s (Figure 8.3(c)) and 40s − 100s (Figure 8.3(d)).

The test results of the swarm with four robots were shown in Figure 8.5. The average polarization has a value of 13.9° for this test. The maximum polarization value is 46.5° for this motion. This maximum polarization value occurs during the 90° turn of the swarm. The average expanse has a value of 2.67 BL while the maximum expanse value is 3.3 BL. When these values are compared with the results of the swarm with three robots, it can be seen that the number of the robots in the swarm effects the expanse. When the number of the robots is increased, the robots cannot stay together as much as a swarm with three robots. The grayscale representation of motion is shown for two different time periods. These time periods are 0s − 45s (Figure 8.5 (c)) and 45s − 80s (Figure 8.5 (d)).

(a) Polarization

(b) Expanse

(c) Grayscale representation (0s – 40s)

(d) Grayscale representation (40s – 100s)
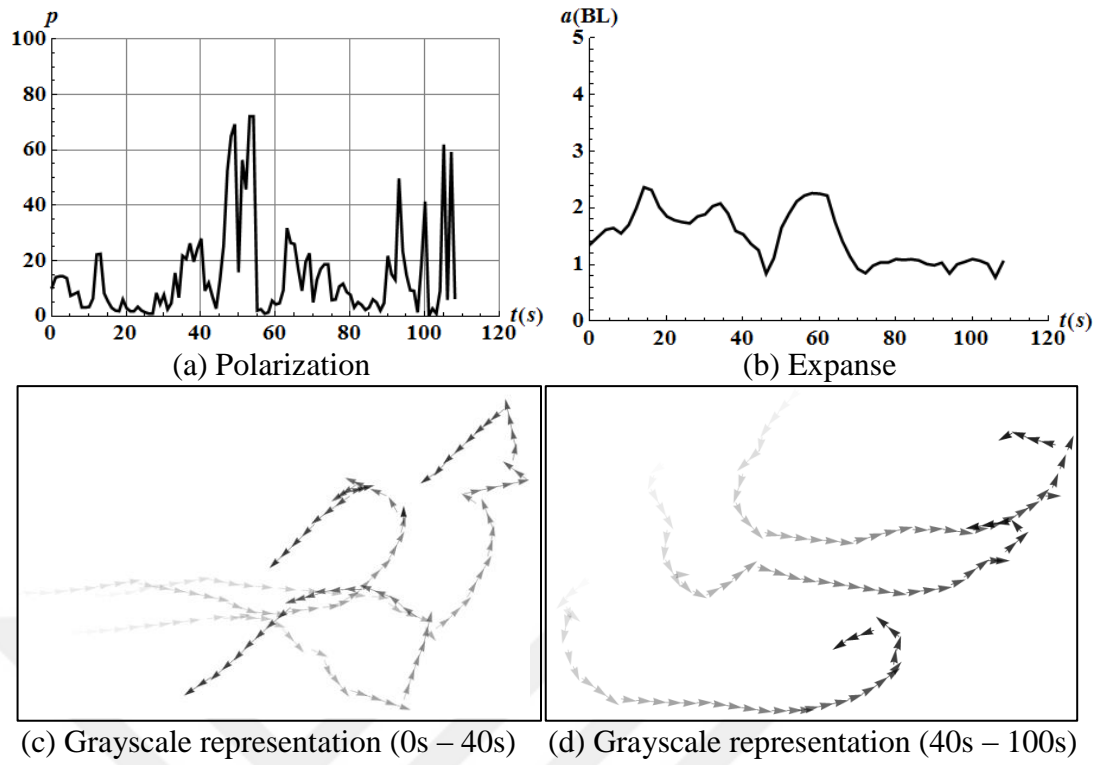
Figure 8.3 : The test results of swarm with three robots.

t=0

t=10s

t=20s

t=30s

t=35

t=45s
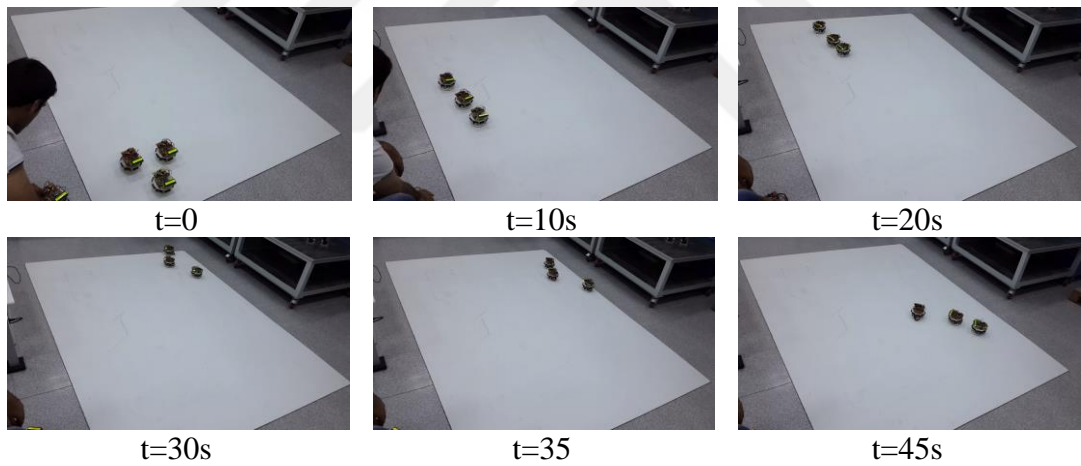
Figure 8.4 : The position of three robots during the motion.

(a) Polarization

(b) Expanse

(c) Grayscale representation (0s − 45s)

(d) Grayscale representation (45s − 80s)
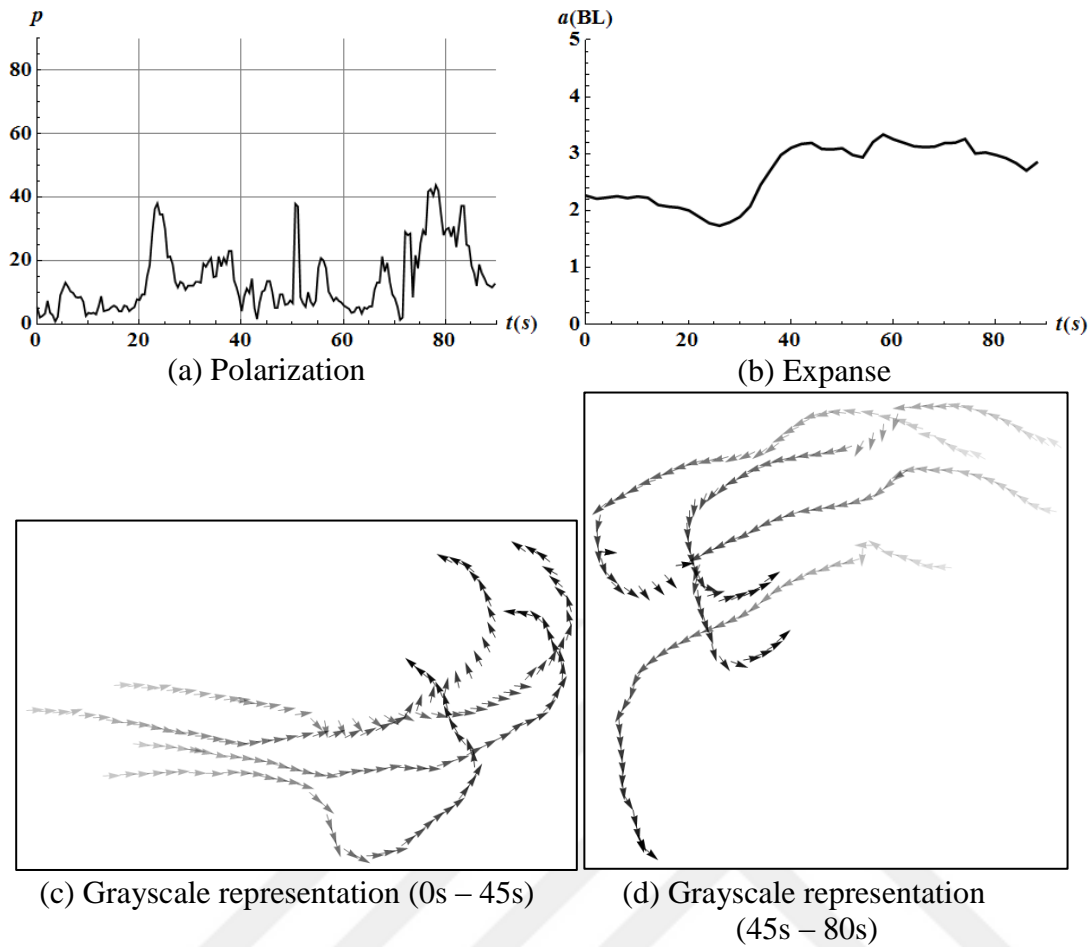
Figure 8.5 : The test result of swarm with four robots.

t=0

t=10s

t=20s

t=30s

t=40

t=50s
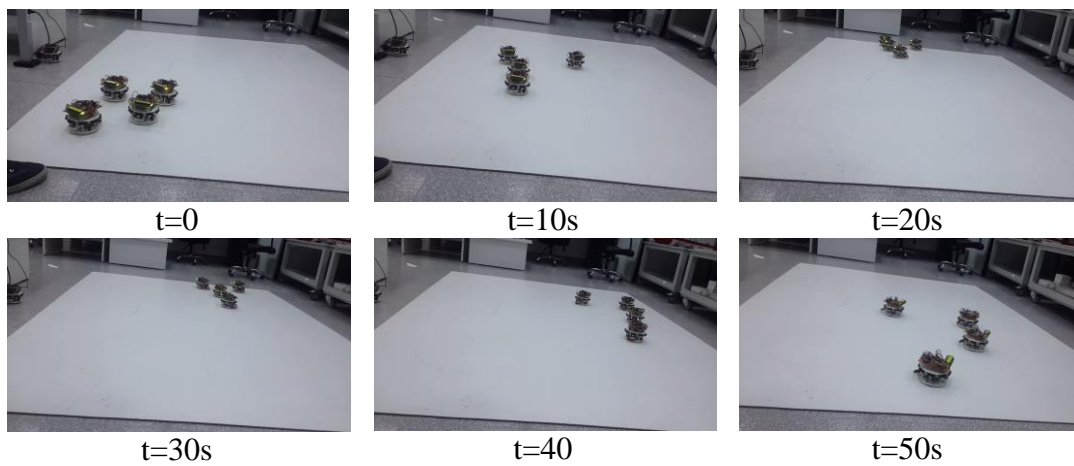
Figure 8.6 : The positions of four robots in every 10 second during the motion.

The test results of the swarm with five robots are shown in Figure 8.7. The average polarization has a value of 20.6° for this test. The maximum polarization value is 46.5° for this motion. The average expanse has a value of 5.96 BL while the maximum

expanse value is 10.96 BL. The grayscale representation of motion is shown for two different time periods. These time periods are 0s − 55s (Figure 8.7 (c)) and 55s − 90s (Figure 8.7 (d)).

(a) Polarization

(b) Expanse

(c) Grayscale representation (0s − 55s)

(d) Grayscale representation
(55s − 90s)
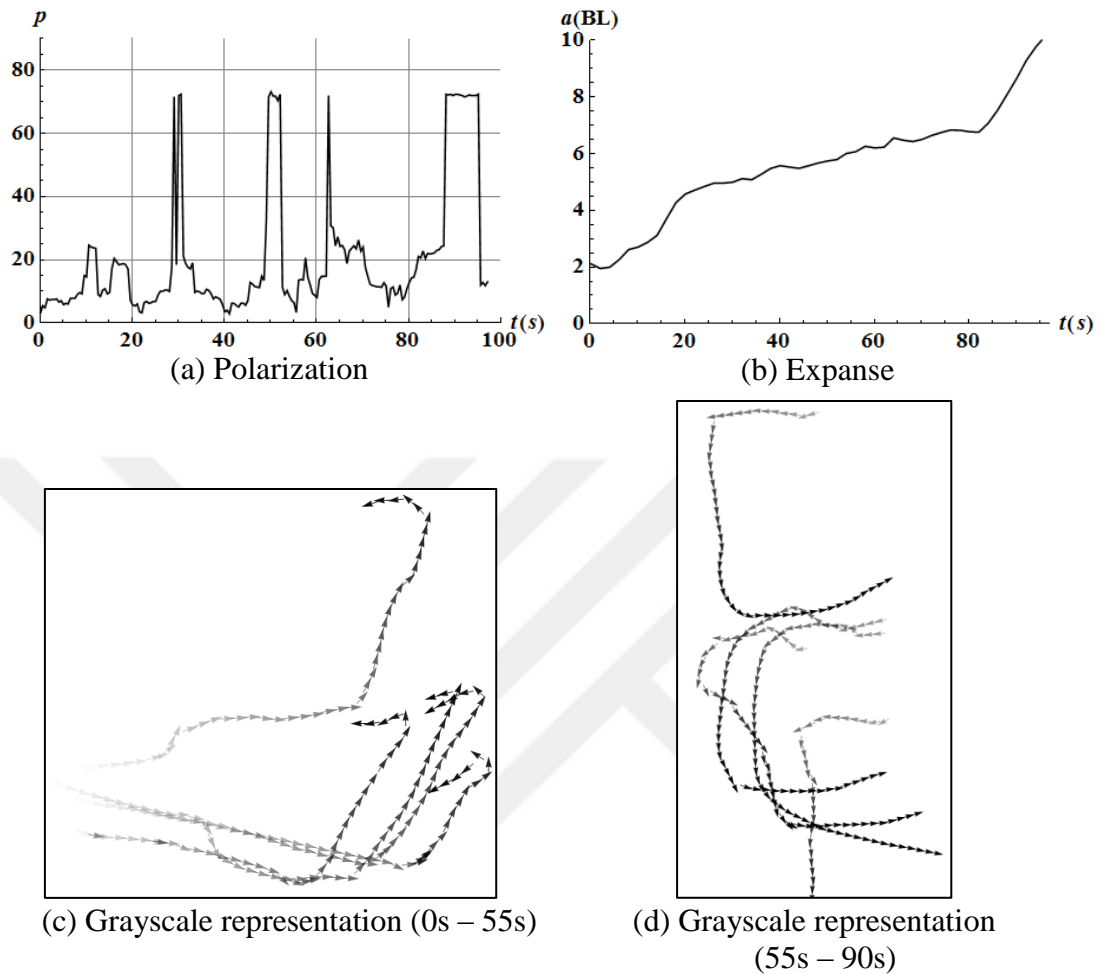
Figure 8.7 : The results of swarm with five robots.

According to the results, when the number of the robots increases, polarization of the motion is in the same range. However, expanse values increase considerably. The first reason of the expanse difference between the motions is the increment of the occurring time delay during the sending of the orientation data from the remote controller to the robots. The occurring time delay during the two robot motion is less than the time delay that occurs during the motion of the swarms which contain more than two robots. The time delay also occurs because the robots send data packages to the data taker.

Table 8.1 : Test results**.**

Number of Robots	The Average Polarization	The Average Expanse
2	6.5°	1.5 BL
3	15.5°	2.45 BL
4	13.9°	2.7 BL
5	20.6°	5.96 BL

The second reason the expanse value increases considerably is the output difference between the digital compasses. The digital compasses are affected by any metallic object in the environment. Because of this, even when the robots are parallel and very close to each other, the digital compasses gives measurements with 3 – 5° error.

Polarization alone is not sufficient to give a concluding remark whether the movement of the swarm is well organized or not. Expanse is equally important for this statement. In the observed swarms, expanse varied between 1.5BL to 5.96 BL (Table 8.1). A small expanse value indicates that the robots are moving in close vicinity with each other, while a greater expanse indicates increased distance among them. Again, the increase in number of robots in the swarm increased the value of expanse.

A combination of both these parameters indicates that a swarm composed of a smaller number of robots has a better organized movement, while the swarm containing more robots (i.e 5 robots) has a more confused movement. The reason for this behavior is the summation of errors on the compass measurements for all the robots of any given swarm.

9. CONCLUSION

In this thesis the motion of the swarm robots as a flocking problem was investigated. One simulation code was also written to observe the motion of the robots and to record their orientation and velocity. The orientation of the robots was controlled remotely by a user during the motion of the robots. The transmitting of orientation data from the remote controller to the robots was carried out by using XBee modules.

The robots were designed and produced by the author according to determined design criteria and the used distance sensors, motors, and battery. In order to carry out the experiments in small areas, the robots were produced in smallest possible size conforming the design criteria conditions.

The Arduino Mega 2560 programmable card was used as a control unit of the robots. However, an external electronic circuit, named Arduino Shield Circuit was designed to connect electronic components such as sensors, digital compass, XBee module and lcd screen to Arduino Mega properly. This external circuit was designed in Proteus 8 Professional. The designed external card was also manufactured by the author.

In this thesis four different codes were written. These are the control code of the robots, the code of the remote controller, the code of the data taker and the simulation code.

Collective motion was performed by robots that are moving with respect to some pair-wise interactions. The pair-wise interactions between the robots were performed based on three rules, namely attraction, parallel orientation and repulsion field rules. While the mobile robots try to move toward their neighbors in attraction field, in parallel orientation field they try to remain close to their neighbors. The repulsion field rule avoids the collision with each other during the collective motion. The control code was written according to these field rules. The robots sense each other by using the distance sensors, then according to the output of the sensors, the robots decide the movement type.

The code of remote controller was written to send the orientation data to the robots for every 100ms. The Arduino Shield Circuit was also used as remote controller. The data taker code was written to send the received data from the robots to the simulation. The simulation works as a real-time simulation. The simulation uses the data received from

the robots to simulate their motion. The simulation also saves all the received data from the robots in one Excel file.

The motion of swarms having two, three, four and five robots were recorded and the data of the robots were saved by using the created simulation. Two different parameters, expanse and polarization, were calculated to characterize the motion. These parameters were plotted as functions of time, in graphs, for better visualization. One graphical representation method, named grayscale representation, was also used and the motion of the robots was shown by using this representation.

Polarization for the observed swarms varied from 6.5° to 20.6°. A small polarization value, close to 0° indicates a well-polarized swarm, while high polarization values, close to 90°, indicate a highly confused swarm. The observed swarms had relatively low polarization, thus the motion of these swarms was polarized. Yet, it was observed that the increase in number of robots in the swarm increased the polarization as well.

Polarization alone is not sufficient to give a concluding remark whether the movement of the swarm is well organized or not. Expanse is equally important for this statement. In the observed swarms, expanse varied between 1.5BL to 5.98 BL. A small expanse value indicates that the robots are moving in close vicinity with each other, while a greater expanse indicates increased distance among them. Again, the increase in number of robots in the swarm increased the value of expanse.

A combination of both these parameters indicates that a swarm composed of a smaller number of robots has a better organized movement, while the swarm containing more robots (i.e 5 robots) has a more confused movement. The reason for this behavior is the summation of errors on the compass measurements for all the robots of any given swarm.

REFERENCES

1. Bayindir, L. and E. Şahin, *A review of studies in swarm robotics.* Turkish Journal of Electrical Engineering & Computer Sciences, 2007. **15**(2): p. 115-147.

2. Şahin, E. and A. Winfield, *Special issue on swarm robotics.* Swarm Intelligence, 2008. **2**(2): p. 69-72.

3. Turgut, A.E., *Self-Organized Flocking with A Mobile Robot Swarm*, in *Mechanical Engineering*. 2008, Middle East Technical University. p. 133.

4. Navarro, I. and F. Matía, *An Introduction to Swarm Robotics.* ISRN Robotics, 2013. **2013**: p. 1-10.

5. Tan, Y. and Z.-y. Zheng, *Research advance in swarm robotics.* Defence Technology, 2013. **9**(1): p. 18-39.

6. Brambilla, M., et al., *Swarm robotics: a review from the swarm engineering perspective.* Swarm Intelligence, 2013. **7**(1): p. 1-41.

7. Bahgeçi, E. *Evolving aggregation behaviors for swarm robotic systems: A systematic case study*. in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. 2005. IEEE.

8. Fredslund, J. and M.J. Mataric, *A general algorithm for robot formations using local sensing and minimal communication.* Robotics and Automation, IEEE Transactions on, 2002. **18**(5): p. 837-846.

9. Fredslund, J. and M.J. Mataric. *Robot formations using only local sensing and control*. in *Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on*. 2001. IEEE.

10. Nouyan, S. and M. Dorigo, *Chain formation in a swarm of robots.* IRIDIA, Université Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2004-18, 2004.

11. Trianni, V., et al., *Evolving aggregation behaviors in a swarm of robots*, in *Advances in artificial life*. 2003, Springer. p. 865-874.

12. Hayes, A.T. and P. Dormiani-Tabatabaei. *Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots*. in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. 2002. IEEE.

13. Stormont, D.P. *Autonomous rescue robot swarms for first responders*. in *Computational Intelligence for Homeland Security and Personal Safety, 2005. CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on*. 2005. IEEE.

14. Hamann, H. and H. Wörn, *An analytical and spatial model of foraging in a swarm of robots*, in *Swarm Robotics*. 2006, Springer. p. 43-55.

15. Labella, T.H., M. Dorigo, and J.-L. Deneubourg, *Efficiency and task allocation in prey retrieval*, in *Biologically Inspired Approaches to Advanced Information Technology*. 2004, Springer. p. 274-289.

16. Kantor, G., et al. *Distributed search and rescue with robot and sensor teams*. in *Field and Service Robotics*. 2003. Springer.

17. Zafar, K., S.B. Qazi, and A.R. Baig. *Mine detection and route planning in military warfare using multi agent system*. in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*. 2006. IEEE.

18. Landis, G.A., *Robots and humans: synergy in planetary exploration.* Acta astronautica, 2004. **55**(12): p. 985-990.

19. Reynolds, C.W. *Flocks, herds and schools: A distributed behavioral model*. in *ACM SIGGRAPH computer graphics*. 1987. ACM.

20. Inada, Y., *Steering mechanism of fish schools.* Complexity international, 2001. **8**: p. 1-9.

21. Strömbom, D., *Collective motion from local attraction.* Journal of theoretical biology, 2011. **283**(1): p. 145-151.

22. Triandaf, I. and I.B. Schwartz, *A collective motion algorithm for tracking time-dependent boundaries.* Mathematics and Computers in Simulation, 2005. **70**(4): p. 187-202.

23. Varghese, B. and G. McKee, *A mathematical model, implementation and study of a swarm system.* Robotics and Autonomous Systems, 2010. **58**(3): p. 287-294.

24. Oboshi, T., et al., *A simulation study on the form of fish schooling for escape from predator.* FORMA-TOKYO-, 2003. **18**(2): p. 119-131.

25. Venayagamoorthy, G.K., L.L. Grant, and S. Doctor, *Collective robotic search using hybrid techniques: Fuzzy logic and swarm intelligence inspired by nature.* Engineering Applications of Artificial Intelligence, 2009. **22**(3): p. 431-441.

26. Castro, E. and M.d.S.G. Tsuzuki, *Swarm Intelligence applied in synthesis of hunting strategies in a three-dimensional environment.* Expert Systems with Applications, 2008. **34**(3): p. 1995-2003.

27. Fukuda, T., et al. *Structure decision method for self organising robots based on cell structures-CEBOT*. in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. 1989. IEEE.

28. Atyabi, A., S. Phon-Amnuaisuk, and C.K. Ho, *Navigating a robotic swarm in an uncharted 2D landscape.* Applied soft computing, 2010. **10**(1): p. 149-169.

29. Li, W. and W. Shen, *Swarm behavior control of mobile multi-robots with wireless sensor networks.* Journal of Network and Computer Applications, 2011. **34**(4): p. 1398-1407.

30.   Ijspeert, A.J., et al., *Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment.* Autonomous Robots, 2001. **11**(2): p. 149-171.

31.   Turgut, A.E., et al., *Self-organized flocking in mobile robot swarms.* Swarm Intelligence, 2008. **2**(2-4): p. 97-120.

32.   Trianni, V., S. Nolfi, and M. Dorigo, *Cooperative hole avoidance in a swarm-bot.* Robotics and Autonomous Systems, 2006. **54**(2): p. 97-103.

33.   Trianni, V. and M. Dorigo. *Emergent collective decisions in a swarm of robots*. in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. 2005. IEEE.

34.   Soysal, O. *Probabilistic aggregation strategies in swarm robotic systems*. in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. 2005. IEEE.

35.   Martinoli, A. and K. Easton, *Modeling swarm robotic systems*, in *Experimental Robotics VIII*. 2003, Springer. p. 297-306.

36.   Martinoli, A., K. Easton, and W. Agassounon, *Modeling swarm robotic systems: A case study in collaborative distributed manipulation.* The International Journal of Robotics Research, 2004. **23**(4-5): p. 415-436.

37.   Lerman, K., et al., *A macroscopic analytical model of collaboration in distributed robotic systems.* Artificial Life, 2001. **7**(4): p. 375-393.

38.   Lerman, K., A. Martinoli, and A. Galstyan, *A review of probabilistic macroscopic models for swarm robotic systems*, in *Swarm robotics*. 2004, Springer. p. 143-152.

39.   Berman, S., et al., *Algorithms for the analysis and synthesis of a bio-inspired swarm robotic system*, in *Swarm Robotics*. 2006, Springer. p. 56-70.

40.   Shen, W.-M., C.-M. Chuong, and P. Will. *Simulating self-organization for multi-robot systems*. in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002. IEEE.

41.   Mondada, F., E. Franzi, and A. Guignard. *The development of khepera*. in *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*. 1999.

42.   Pugh, J., et al., *A fast onboard relative positioning module for multirobot systems.* Mechatronics, IEEE/ASME Transactions on, 2009. **14**(2): p. 151-162.

43.   Mondada, F., et al. *The e-puck, a robot designed for education in engineering*. in *Proceedings of the 9th conference on autonomous robot systems and competitions*. 2009. IPCB: Instituto Politécnico de Castelo Branco.

44.   Caprari, G. and R. Siegwart. *Mobile micro-robots ready to use: Alice*. in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. 2005. IEEE.

45.   Kornienko, S., O. Kornienko, and P. Levi. *Minimalistic approach towards communication and perception in microrobotic swarms*. in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. 2005. IEEE.

46. Mondada, F., et al., *The cooperation of swarm-bots: Physical interactions in collective robotics.* Robotics & Automation Magazine, IEEE, 2005. **12**(2): p. 21-28.

47. Turgut, A.E., et al., *Kobot: A mobile robot designed specifically for swarm robotics research.* Middle East Technical University, Ankara, Turkey, METUCENG-TR Tech. Rep, 2007. **5**: p. 2007.

48. McLurkin, J.D., *Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots*. 2004, Massachusetts Institute of Technology.

49. Ata Elahi, A.G., *ZigBee Wireless Sensor and Control Network*. 1 ed. 2009: Prentice Hall.

50. Huth, A. and C. Wissel, *The simulation of the movement of fish schools.* Journal of theoretical biology, 1992. **156**(3): p. 365-385.

51. *https://www.arduino.cc/en/Tutorial/PWM*.

APPENDICES

APPENDIX A: The control code of the robots

APPENDIX B: The code of the remote controller

APPENDIX C: The code of the data taker

APPENDIX D: The code of the simulation

APPENDIX A

```
#include <XBee.h>

#include <Wire.h>

#include <LSM303.h>

#include <LiquidCrystal.h>

LSM303 compass;

XBee xbee = XBee();

XBeeResponse response = XBeeResponse();

uint8_t payload[]={0,0};

uint8_t data = 0;

uint8_t rssi = 0;

Tx16Request tx = Tx16Request(0x8, payload, sizeof(payload));//0x8 address of the
other Xbee

TxStatusResponse txStatus = TxStatusResponse();

Rx16Response rx16 = Rx16Response();

//used variables

long last;

int time=0;

int dfx=2,xr,xl,cxra,cxrr,adr,x1, direc,v=0,diff,degree,aa,bb, speedLeft, speedRight;

int i, a = 0, data1, data2,  cntr2 = 0, adress, k, degav = 0;

int mdx, mdmin = 290.00, border1 = 110.00, border2 = 160.00, mdmax=60.00;

double mxx,a1,a2,b1,b2,vratio;

int sensorpins[] = {A0, A1, A2, A3, A11, A10, A9, A8};

int s[8] = {}; //reading distances from the Sharps

int robots[8][2] = {};// assign to received data to robots matrice.

float heading;

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(50, 51, 34, 32, 30, 28); //LCD Digital Pins Declaration
```

58

```cpp
void setup()
{
 k=1;
 lcd.begin(16, 2);
 Serial.begin(9600);
 Serial3.begin(9600);
 xbee.setSerial(Serial3);
 Wire.begin();
 compass.init();
 compass.enableDefault();
if(k==1)
{mxx=50.00;
compass.m_min = (LSM303::vector<int16_t>) { -2512, -2874, -2748};
compass.m_max = (LSM303::vector<int16_t>) { +2147, +2236, +1988};
}
if(k==2)
{mxx=50.00;
compass.m_min = (LSM303::vector<int16_t>) { -2612, -2516, -3061};
compass.m_max = (LSM303::vector<int16_t>) { +2316, +2482, +1689};
}
if(k==3)
{mxx=53.00;
compass.m_min = (LSM303::vector<int16_t>) { -2848, -2585, -2153} ;
compass.m_max = (LSM303::vector<int16_t>) { +2003, +2348, +2676};
}
if(k==4)
{ mxx=54.00;
compass.m_min = (LSM303::vector<int16_t>) { -2245, -2400, -955};
compass.m_max = (LSM303::vector<int16_t>) { +2650, +3052, +3750};
```

```
 }
   if(k==5)
{mxx=55.00;
compass.m_min = (LSM303::vector<int16_t>) { -2331, -2634, -2032};
compass.m_max = (LSM303::vector<int16_t>) { +2589, +2407, +2658};
}
   if(k==6)
{
mxx=44.00;
compass.m_min = (LSM303::vector<int16_t>) { -2158, -2547, -2944};
compass.m_max = (LSM303::vector<int16_t>) { +2572, +2381, +1801};
}
 pinMode(9, OUTPUT);
 pinMode(10, OUTPUT);
 pinMode(11, OUTPUT);
 pinMode(12, OUTPUT);
 pinMode(13, OUTPUT);
 randomSeed(analogRead(A4));//for randomly starting
}//end void setup
void loop()
{
 compass.read();
 ReadSensors();
 heading = compass.heading();
 direc=heading;
 last=k*10000+direc*10+vratio;
 getdata();
 delay(60);
 senddata(last);
```

```
delay(50);

switch (k)

{

case 1: mxx=52;xr=52;xl=52;

break;

case 2: mxx=51;xr=51;xl=51;

break;

case 3: mxx=54;xr=54;xl=54;

break;

case 4: mxx=51;xr=51;xl=51;

break;

case 5: mxx=55;xr=55;xl=55;

break;

case 6: mxx=44;xr=50;xl=50;

break;

default:

break;

}

if(data2>300&&data2<350){cntr2=1;}

if(cntr2==1){parallelorientation(mxx);}

LCDprint();

}

void senddata(int x)//

{

 payload[0] = x >> 8 & 0xff;

 payload[1] = x & 0xff;

 xbee.send(tx);

}

//
```

```
void getdata()

{

 xbee.readPacket();

 if (xbee.getResponse().getApiId() == RX_16_RESPONSE)

 {

  xbee.getResponse().getRx16Response(rx16);

  rssi = rx16.getRssi();

  adr=rx16.getRemoteAddress16();//adress of the other Xbee that sent data packet

  //ÖnEMLİ KISIM&//

  data = rx16.getData(0);

  data1 = rx16.getData(1);

  data2 = data * 256 + data1;

 }

}

double f(int x)//converting output voltage of sensors to millimeter

{

double a;

a = 595.121 - 6.39166 * x + 0.0360895 * x * x - 0.000109734 * x * x * x +1.67994e-
7 * x * x * x * x - 1.01098e-10 * x * x * x * x * x;

return a;

}

void RobotmotorsWrite(int x, int y)

{

 double vx,vy;

 vx=x*447.0/255;

 vy=y*447.0/255;

 v=(vx+vy)/2;

 vratio=(x+y)/2;

 vratio=(vratio/mxx)*10;
```

```
if(vratio==0){vratio=0;}
else{vratio=map(vratio,10,19,1,9);}
if (x >= 0 && y >= 0)
 {
switch (k)
{
case 1: x=x; y=y;
break;
case 2: x=x; y=y;
break;
case 3: x=x; y=y;
break;
case 4: x=x+5;y=y-1;
break;
case 5: x=x+5;
break;
case 6: x=x+5; y=y-1;
break;
default:
break;
}
}
if (x >= 0 && y >= 0)
 {
  analogWrite(10, x);
  digitalWrite(9, LOW);
  analogWrite(11, y);
  digitalWrite(12, LOW);
  delay(35);
```

```
  }
}
//Function of orientation
void parallelorientation(int mxx)
{
  LCDprint();
  orientation(data2,mxx);
  getdata();delay(5);
  ReadSensors();
  while (s[0]<=mdmin+5 && s[0]>border2)
  {
ReadSensors();
if(s[1]<=border2 || s[2]<=border2 || s[7]<=border2 || s[6]<=border2 ||
s[0]<=border2)break;
//ra(s[0]);if(cxra>=mxx && cxra<= 2*mxx){RobotmotorsWrite(cxra,cxra);}
 mxx=mxx+dfx;if(mxx>2*mxx){mxx=2*mxx;}RobotmotorsWrite(mxx,mxx);
  }
  while (s[0]<=border1-10 ||s[1]<=border1-10 ||s[7]<=border1-10 )
  {ReadSensors(); mxx=mxx-dfx;
  if(mxx<0){mxx=0;}
  RobotmotorsWrite(mxx,mxx);
  }
while ((((s[1]>border1 && s[1]<border2) ||(s[2]>border1 && s[2]<border2) ||
(s[3]>border1 && s[3]<border2))&&((s[5]>border1 && s[5]<border2) ||
(s[6]>border1 && s[6]<border2) || (s[7]>border1 && s[7]<border2)))
{    ReadSensors();
 if(s[0]<border1 || s[1]<border1 ||s[2]<border1 ||s[3]<border1 ||s[6]<border1
||s[7]<border1)break;
    getdata(); delay(20); orientation(data2,mxx);
}
```

64

```
while(s[1]<=border1||s[2]<=border1)

{LCDprint();ReadSensors();if((s[1]>=border1 && s[2]>=border1) || (s[6]<=border1
|| s[7]<=border1))break;mdx=min(s[1],s[2]);rr(mdx);

RobotmotorsWrite(mxx,cxrr);RobotmotorsWrite(mxx,mxx);

}

while((s[1]>border2 && s[1]<=mdmin)||(s[2]>border2&&s[2]<=mdmin))

{ReadSensors();

 if(s[6]>border2 && s[7]>border2)

{

 LCDprint(); ReadSensors();if(s[1]<=(border2)||s[2]<=(border2)||s[0]<=120)break;

 if(s[1]<=mdmin &&
s[2]<=mdmin){mdx=max(s[1],s[2]);}else{mdx=min(s[1],s[2]);}

 ra(mdx);if(cxra>=mxx && cxra<=2*mxx)

{RobotmotorsWrite(cxra,mxx);}delay(time);RobotmotorsWrite(mxx,mxx);}

if(s[6]<=mdmin && s[7]<=mdmin){orientation(data2,mxx);}

}

while(s[6]<=border1||s[7]<=border1){LCDprint();ReadSensors();

if((s[6]>=border1 && s[7]>=border1) || (s[1]<=border1 || s[2]<=border1))break;

mdx=min(s[6],s[7]);rr(mdx);

RobotmotorsWrite(cxrr,mxx);RobotmotorsWrite(mxx,mxx);}

while((s[6]>border2 && s[6]<=mdmin)||(s[7]>border2 && s[7]<=mdmin))

{ReadSensors();

  if(s[1]>border2 && s[2]>border2)

 {LCDprint();ReadSensors();

if(s[6]<=(border2) || s[7]<=(border2) || s[0]<=120)break;

  if(s[6]<mdmin && s[7]<mdmin){mdx=max(s[6],s[7]);}else{mdx=min(s[6],s[7]);}

  ra(mdx);if(cxra>=mxx && cxra<= 2*mxx){RobotmotorsWrite(mxx,cxra);
delay(time);RobotmotorsWrite(mxx,mxx);}

 }
```

```
if(s[1]<=mdmin && s[2]<=mdmin){orientation(data2,mxx);}

}

while ((s[4]<mdmin && s[4]>border2)||(s[3]<mdmin &&
s[3]>border2)||(s[5]<mdmin && s[5]>border2))

{ReadSensors();LCDprint();if(s[3]<border2|| s[4]<border2 || s[5]<border2)break;
mxx=mxx-2;if(mxx<xl-10){RobotmotorsWrite(mxx-10,mxx-
10);}else{RobotmotorsWrite(mxx,mxx);}}

while(s[3]<=mdmax+20)

{ReadSensors();if(s[3]>=border1||s[0]<=150 || s[6]<=border1+5 ||
s[7]<=border1+5)break;

mxx=mxx+dfx; if(mxx>2*xl){mxx=2*xl;RobotmotorsWrite(mxx,mxx);}

else{RobotmotorsWrite(mxx,mxx);}}

while(s[4]<=(mdmax+20))

{ReadSensors();if(s[4]>=border1||s[0]<=150 || s[1]<=border1+5 ||
s[7]<=border1+5)break;

mxx=mxx+dfx; if(mxx>2*xl){mxx=2*xl;RobotmotorsWrite(mxx,mxx);}

else{RobotmotorsWrite(mxx,mxx);}}


while(s[5]<=(mdmax+20))

{ReadSensors();if(s[5]>=border1||s[0]<=150|| s[1]<=border1+5 ||
s[2]<=border1+5)break;

mxx=mxx+dfx; if(mxx>2*xl){mxx=2*xl;RobotmotorsWrite(mxx,mxx);}

else{RobotmotorsWrite(mxx,mxx);}}

if(data2>360){data2=data2%360;}

}//END Parallel Orientation

void ra(int x)

{

    a1=(mxx/(double)(mdmin-border2));

    b1=mxx*(1-border2/(double)(mdmin-border2));

    cxra=a1*x+b1;
```

```
}
void rr(int x)
{
  a2=mxx/(double)(mdmax-border1);
  b2=mxx*(1-(border1/(double)(mdmax-border1)));
  cxrr=a2*x+b2;
}


void orientation(int degree, int mxx)
{ diff = heading - degree;
 if (diff > 180)
  {
   diff = -360 + diff;
  }
  else if (diff < -180)
  {
   diff = 360 + diff;
  }
  diff = map(diff, -180, 180, -mxx, mxx);
  if (diff > 0) {
   // keep the right wheel spinning,
   // change the speed of the left wheel
   speedLeft = mxx - diff;
   speedRight = mxx +2*diff;
  } else {
   // keep the right left spinning,
   // change the speed of the left wheel
   speedLeft = mxx -2*diff;
   speedRight = mxx + diff;
```

```
  }
  RobotmotorsWrite(speedLeft, speedRight);
}
//Function of Reading distances
void ReadSensors()
{  x1=0;
  for (i = 0; i <= 7; i++)
  {
    for (int y = 0; y < 25; y++) {x1 = x1+analogRead(sensorpins[i]);}
    x1=x1/25;
    s[i] = f(x1);
  }
}
//function to write something to LCD
void LCDprint()
{
  lcd.print("d:");
  lcd.print(heading); lcd.print("/");lcd.print("/");lcd.print(s[1]);
  lcd.setCursor(0, 1);
  lcd.print(data2); lcd.print("/"); lcd.print(cxrr);
lcd.print("/");lcd.print(s[3]);lcd.print("/");
  delay(50);
  lcd.clear();
}
```

APPENDIX B

```cpp
#include <XBee.h>

#include <Wire.h>

#include <LSM303.h>

#include <LiquidCrystal.h>

LSM303 compass;

XBee xbee = XBee();

XBeeResponse response = XBeeResponse();

uint8_t payload[2];

uint8_t data = 0;

uint8_t rssi = 0;

Tx16Request tx = Tx16Request(0x1, payload, sizeof(payload));//0x1 address of the
other Xbee

TxStatusResponse txStatus = TxStatusResponse();

Rx16Response rx16 = Rx16Response();

//used variables

int data1, data2;

float heading;

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(50, 51, 34, 32, 30, 28); //LCD Digital Pins Declaration

void setup()

{

 lcd.begin(16, 2);

 Serial.begin(9600);

 Serial3.begin(9600);

 xbee.setSerial(Serial3);

 Wire.begin();

 compass.init();
```

```
compass.enableDefault();

compass.m_min = (LSM303::vector<int16_t>) { -2512, -2874, -2748} ;

compass.m_max = (LSM303::vector<int16_t>) { +2147, +2236, +1988};

}//end void setup

void loop()

{

  compass.read();

  heading = compass.heading();

  senddata(int(heading));

  delay(50);

  LCDprint();

}

void senddata(int x)//

{

  payload[0] = x >> 8 & 0xff;

  payload[1] = x & 0xff;

  xbee.send(tx);

}

//

void getdata()

{

  xbee.readPacket();

  if (xbee.getResponse().getApiId() == RX_16_RESPONSE)

  {

    xbee.getResponse().getRx16Response(rx16);

    rssi = rx16.getRssi();

    data = rx16.getData(0);

    data1 = rx16.getData(1);

    data2 = data * 256 + data1;
```

```
}

}

//function to write something to LCD

void LCDprint()

{

 lcd.print("d:");

 lcd.print(heading); lcd.print("/");

 lcd.setCursor(0, 1);

 //lcd.print(data2); lcd.print("/"); lcd.print(cxrr);
lcd.print("/");lcd.print(cxra);lcd.print("/");lcd.print(a1);

 delay(50);

 lcd.clear();

}
```

APPENDIX C

```
#include <XBee.h>

#include <Wire.h>

#include <LSM303.h>

LSM303 compass;

XBee xbee = XBee();

XBeeResponse response = XBeeResponse();

uint8_t payload[] = { 0, 0 };

uint8_t data = 0;

uint8_t rssi = 0;

Tx16Request tx = Tx16Request(0x1, payload, sizeof(payload));//0x1 address of the
other Xbee

TxStatusResponse txStatus = TxStatusResponse();

Rx16Response rx16 = Rx16Response();

long data2;

int c = 1;

int heading, x, i = 1, a, aa, bb, adress, data1, data3, data4, vel, adr, mdx, k, mdmin =
250.00, border1 = 120.00, border2 = 180.00, mdmax = 60.00, x1, cxra, cxrr, degree,
deg[8], v[8];

double a1, a2, b1, b2, mxx;

void setup()

{

 Wire.begin();

 Serial.begin(9600);

 Serial3.begin(9600);

 xbee.setSerial(Serial3);

 compass.init();

 compass.enableDefault();
```

```cpp
  compass.m_min = (LSM303::vector<int16_t>){ -32767, -32767, 32767};

  compass.m_max = (LSM303::vector<int16_t>) {+32767, +32767, +32767};

  if (c == 1)
  { for (int j = 1; j < 8; j++)
    { deg[j] = 0;
      v[j] = 0;
    }
    c = 2;
  }
}

void loop()
{
  getdata();
  if (i == 6)
{ i = 1; }
  i = i + 1;
  if (deg[i] >= 0 && deg[i] < 360 && v[i] >= 0 && v[i] < 200)
  {
    if (deg[i] < 10 && deg[i] >= 0 && v[i] >= 0 && v[i] < 10) {
      Serial.print(i, DEC);
      Serial.print("00");
      Serial.print(deg[i], DEC);
      Serial.print("00");
      Serial.print(v[i], DEC);
      Serial.print("\n");
    }
    if (deg[i] < 10 && deg[i] >= 0 && v[i] >= 10 && v[i] < 100) {
      Serial.print(i, DEC);
      Serial.print("00");
```

```
    Serial.print(deg[i], DEC);

    Serial.print("0");

    Serial.print(v[i], DEC);

    Serial.print("\n");

  }

  if (deg[i] < 10 && deg[i] >= 0 && v[i] >= 100) {

    Serial.print(i);

    Serial.print("00");

    Serial.print(deg[i]);

    Serial.print(v[i]);

    Serial.print("\n");

  }

  if (deg[i] >= 10 && deg[i] < 100 && v[i] >= 10 && v[i] < 100) {

    Serial.print(i, DEC);

    Serial.print("0");

    Serial.print(deg[i], DEC);

    Serial.print("0");

    Serial.print(v[i], DEC);

    Serial.print("\n");

  }

  if (deg[i] >= 10 && deg[i] < 100 && v[i] >= 100) {

    Serial.print(i, DEC);

    Serial.print("0");

    Serial.print(deg[i], DEC);

    Serial.print(v[i], DEC);

    Serial.print("\n");

  }

  if (deg[i] >= 10 && deg[i] < 100 &&  v[i] >= 0 && v[i] < 10) {

    Serial.print(i, DEC);
```

```
      Serial.print("0");

      Serial.print(deg[i], DEC);

      Serial.print("00");

      Serial.print(v[i], DEC);

      Serial.print("\n");

    }

    if (deg[i] >= 100 && v[i] >= 10 && v[i] < 100) {

      Serial.print(i, DEC);

      Serial.print(deg[i], DEC);

      Serial.print("0");

      Serial.print(v[i], DEC);

      Serial.print("\n");

    }

    if (deg[i] >= 100 && v[i] >= 100) {

      Serial.print(i, DEC);

      Serial.print(deg[i], DEC);

      Serial.print(v[i], DEC);

      Serial.print("\n");

    }

    if (deg[i] >= 100 && v[i] >= 0 && v[i] < 10) {

      Serial.print(i, DEC);

      Serial.print(deg[i], DEC);

      Serial.print("00");

      Serial.print(v[i], DEC);

      Serial.print("\n");

    }

  }

  delay(100);

}
```

```
void senddata(int x)//

{

  payload[0] = x >> 8 & 0xff;

  payload[1] = x & 0xff;

  xbee.send(tx);

  delay(40);

}

void getdata()

{

  xbee.readPacket();

  if (xbee.getResponse().getApiId() == RX_16_RESPONSE)

  {

    xbee.getResponse().getRx16Response(rx16);

    rssi = rx16.getRssi();

    adr = rx16.getRemoteAddress16(); //adress of the other Xbee that sent data packet

    data = rx16.getData(0);

    data1 = rx16.getData(1);

    data2 = data * 256 + data1;

    vel = data2;

  }

  if (data2 < 0 || (data2 > 0 && data2 < 4500))

  {

    data2 = data2 + 65536;

  }

  a = data2 / 10000;

  deg[a] = (data2 % 10000) / 10;

  v[a] = (data2 % 10000) - deg[a] * 10;

  if (v[a] == 0) {

    v[a] = 0;
```

76

```
  }
 else {
  v[a] = map(v[a], 1, 9, 10, 19);
  v[a] = v[a] * 447 * 5 / 255;
 }
}
```

```
 else {
  v[a] = map(v[a], 1, 9, 10, 19);
```

APPENDIX D

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using Excel = Microsoft.Office.Interop.Excel;
namespace simulation
{
    public partial class Form1 : Form
    {
        string[] a;
        string[] vv;
        string b,x;
        int i,j,k=0;
        int[] d;
        int[] v;
        double xi = 250, yi = 700, L,dx=80,dxx,dxy, dt = 0.03, t;
        double dt1=0.0379,dt2=0.018895;
        double[] xx;
        double[] yy;
        double[,] xrr= new double [8,400000];
        double[,] yrr = new double[8,400000];
        double[,] tetar = new double[8,400000];
        double[,] vrr = new double[8,400000];

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            dxx = dx *  Math.Cos((72) * Math.PI / 180);
            dxy = dx * Math.Sin((72) * Math.PI / 180);
            a = new string[8];
            d = new int[8];
            v = new int[8];
            vv = new string[8];
xx = new double[8] { 0, xi, xi + dxx, xi + dx, xi + dxx, xi - dx *Math.Cos((36)*
Math.PI / 180), xi - dx *Math.Cos((36) * Math.PI / 180), xi - dx * 0.5 };
```

78

```
yy = new double[8] { 0, yi, yi - dxy, yi, yi + dxy, yi + dx * Math.Sin((36) * Math.PI /
180), yi - dx * Math.Sin((36) * Math.PI / 180), yi - dx * 0.5 * Math.Sqrt(3) };

        // ovalShape1.Visible = false;
        //ovalShape2.Visible = false;
        //ovalShape3.Visible = false;
        //ovalShape4.Visible = false;
        // ovalShape5.Visible = false;
        ovalShape6.Visible = false;
        ovalShape7.Visible = false;
        //lineShape1.Visible = false;
        //lineShape2.Visible = false;
        //lineShape3.Visible = false;
        //lineShape4.Visible = false;
        ////lineShape5.Visible = false;
        lineShape6.Visible = false;
        lineShape7.Visible = false;

        this.WindowState = FormWindowState.Maximized;
        serialPort1.PortName = "COM32";
        serialPort1.BaudRate = 9600;
        serialPort1.Open();

    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        L = 10;
        textBox1.Text = Convert.ToString(d[1]);
        textBox2.Text = Convert.ToString(d[2]);
        textBox3.Text = Convert.ToString(d[3]);
        textBox4.Text = Convert.ToString(d[4]);
        textBox5.Text = Convert.ToString(d[5]);
        textBox6.Text = Convert.ToString(d[6]);
        textBox7.Text = Convert.ToString(d[7]);
        textBox8.Text = Convert.ToString(v[1]);
        textBox9.Text = Convert.ToString(v[2]);
        textBox10.Text = Convert.ToString(v[3]);
        textBox11.Text = Convert.ToString(v[4]);
        textBox12.Text = Convert.ToString(v[5]);
        textBox13.Text = Convert.ToString(v[6]);
        textBox14.Text = Convert.ToString(v[7]);
        k = k + 1;
        for (j = 1; j < 8; j++)
        {
            xx[j] = xx[j] + v[j] * dt * Math.Cos(-d[j] * Math.PI / 180);
            yy[j] = yy[j] + v[j] * dt * Math.Sin(-d[j] * Math.PI / 180);
            xrr[j,k] = xx[j];
            yrr[j,k] = yy[j];
            tetar[j,k] = d[j];
```

```csharp
            vrr[j,k] = v[j];
        }

        ovalShape1.Left = Convert.ToInt32(xx[1]);
        ovalShape1.Top = Convert.ToInt32(yy[1]);
        lineShape1.X1 = ovalShape1.Left + ovalShape1.Width / 2;
        lineShape1.Y1 = ovalShape1.Top + ovalShape1.Width / 2;
        lineShape1.X2 = lineShape1.X1 + Convert.ToInt32(L * Math.Cos((-d[1]) *
Math.PI / 180));
        lineShape1.Y2 = lineShape1.Y1 + Convert.ToInt32(L * Math.Sin((-d[1]) *
Math.PI / 180));

        //if (textBox1.Text != "" && (textBox1.Text is string))

        ovalShape2.Left = Convert.ToInt32(xx[2]);
        ovalShape2.Top = Convert.ToInt32(yy[2]);
        lineShape2.X1 = ovalShape2.Left + ovalShape2.Width / 2;
        lineShape2.Y1 = ovalShape2.Top + ovalShape2.Width / 2;
        lineShape2.X2 = lineShape2.X1 + Convert.ToInt32(L * Math.Cos((-d[2]) *
Math.PI / 180));
        lineShape2.Y2 = lineShape2.Y1 + Convert.ToInt32(L * Math.Sin((-d[2]) *
Math.PI / 180));
        //t = t + dt;
        ovalShape3.Left = Convert.ToInt32(xx[3]);
        ovalShape3.Top = Convert.ToInt32(yy[3]);
        lineShape3.X1 = ovalShape3.Left + ovalShape3.Width / 2;
        lineShape3.Y1 = ovalShape3.Top + ovalShape3.Width / 2;
        lineShape3.X2 = lineShape3.X1 + Convert.ToInt32(L * Math.Cos((-d[3]) *
Math.PI / 180));
        lineShape3.Y2 = lineShape3.Y1 + Convert.ToInt32(L * Math.Sin((-d[3]) *
Math.PI / 180));
        //t = t + dt;
        ovalShape4.Left = Convert.ToInt32(xx[4]);
        ovalShape4.Top = Convert.ToInt32(yy[4]);
        lineShape4.X1 = ovalShape4.Left + ovalShape5.Width / 2;
        lineShape4.Y1 = ovalShape4.Top + ovalShape5.Width / 2;
        lineShape4.X2 = lineShape4.X1 + Convert.ToInt32(L * Math.Cos((-d[4]) *
Math.PI / 180));
        lineShape4.Y2 = lineShape4.Y1 + Convert.ToInt32(L * Math.Sin((-d[4]) *
Math.PI / 180));
        //t = t + dt;
        ovalShape5.Left = Convert.ToInt32(xx[5]);
        ovalShape5.Top = Convert.ToInt32(yy[5]);
        lineShape5.X1 = ovalShape5.Left + ovalShape5.Width / 2;
        lineShape5.Y1 = ovalShape5.Top + ovalShape5.Width / 2;
        lineShape5.X2 = lineShape5.X1 + Convert.ToInt32(L * Math.Cos((-d[5]) *
Math.PI / 180));
        lineShape5.Y2 = lineShape5.Y1 + Convert.ToInt32(L * Math.Sin((-d[5]) *
Math.PI / 180));
        //t = t + dt;
```

```csharp
        ovalShape6.Left = Convert.ToInt32(xx[6]);
        ovalShape6.Top = Convert.ToInt32(yy[6]);
        lineShape6.X1 = ovalShape6.Left + ovalShape6.Width / 2;
        lineShape6.Y1 = ovalShape6.Top + ovalShape6.Width / 2;
        lineShape6.X2 = lineShape6.X1 + Convert.ToInt32(L * Math.Cos((-d[6]) *
Math.PI / 180));
        lineShape6.Y2 = lineShape6.Y1 + Convert.ToInt32(L * Math.Sin((-d[6]) *
Math.PI / 180));
        //t = t + dt;
        ovalShape7.Left = Convert.ToInt32(xx[7]);
        ovalShape7.Top = Convert.ToInt32(yy[7]);
        lineShape7.X1 = ovalShape7.Left + ovalShape7.Width / 2;
        lineShape7.Y1 = ovalShape7.Top + ovalShape7.Width / 2;
        lineShape7.X2 = lineShape7.X1 + Convert.ToInt32(L * Math.Cos((-d[7]) *
Math.PI / 180));
        lineShape7.Y2 = lineShape7.Y1 + Convert.ToInt32(L * Math.Sin((-d[7]) *
Math.PI / 180));

    }

    private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
    {
        if (serialPort1.IsOpen)
        {
            b = serialPort1.ReadLine();

            x=b.Substring(0,1);
            if (x!="" && x is string)
            { i = Convert.ToInt32(x); }

            a[i] = b.Substring(1,3);
            vv[i] = b.Substring(4,3);

            if (a[i] != "" && (a[i] is string) && vv[i] != "" && (vv[i] is string))
            {
                d[i] = Convert.ToInt32(a[i]);
                v[i] = Convert.ToInt32(vv[i]);

            }

        }

    }

    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {

    }
```

```csharp
        private void button1_Click(object sender, EventArgs e)
        {
            serialPort1.Close();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void button2_Click(object sender, EventArgs e)
        {
            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            object misValue = System.Reflection.Missing.Value;
            xlApp = new Excel.Application();
            xlWorkBook = xlApp.Workbooks.Add(misValue);
            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
            xlWorkSheet.Cells[2, 1] = "Time";
            for (int i = 1; i < 8; i++)
            {
                xlWorkSheet.Cells[1, 1+i+3*(i-1)] ="Robot"+i;
                xlWorkSheet.Cells[2, 1+i + 3 * (i - 1)] = "X Pos.";
                xlWorkSheet.Cells[2, i+1+1 + 3 * (i - 1)] = "Y Pos." ;
                xlWorkSheet.Cells[2, i+2+1 + 3 * (i - 1)] = "Orientation angle";
                xlWorkSheet.Cells[2, i+3 +1+ 3 * (i - 1)] = "Velocity";

            }

            for (int jj = 1; jj <= k; jj++)
            {
                for (int ii = 1; ii < 8; ii++)
                {
                    xlWorkSheet.Cells[jj+2, 1] = jj;

                    xlWorkSheet.Cells[jj+2, 1 + ii + 3 * (ii - 1)] = xrr[ii, jj];
                    xlWorkSheet.Cells[jj+2, 1 + ii + 1 + 3 * (ii - 1)] = yrr[ii, jj];
                    xlWorkSheet.Cells[jj+2, 1 + ii + 2 + 3 * (ii - 1)] = tetar[ii, jj];
                    xlWorkSheet.Cells[jj+2, 1 + ii + 3 + 3 * (ii - 1)] = vrr[ii, jj];

                }//end for

            }//end for


            xlWorkBook.SaveAs("C:\\Users\\HAYRETTiN\\Desktop\\KODLAR\\simulation\\ou
tput.xls", Excel.XlFileFormat.xlWorkbookNormal, misValue, misValue, misValue,
```

```
misValue, Excel.XlSaveAsAccessMode.xlExclusive, misValue, misValue,
misValue, misValue, misValue);
        xlWorkBook.Close(true, misValue, misValue);
        xlApp.Quit();


    }
```

CURRICULUM VITAE

Name Surname:	**Hayrettin ŞEN**
Place and Date of Birth:	**ÖDEMİŞ 23.05.1990**
Address:	**İzmir Katip Çelebi Üniversitesi Mühendislik ve Mimarlık Fakültesi, Mekatronik Mühendisliği Bölümü, Balatçık Kampüsü, Çiğli/İzmir, Türkiye**
E-Mail:	**hayrettinsenn@gmail.com**
B.Sc.:	**Mechanical Engineering**

List of Publications:

Can F.C. and Şen H., Sürü Simülasyon Programi Geliştirilmesi ve Performansının İncelenmesi, Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu (ASYU-2014), 163-166.

Can F.C. and Şen H., A Simulation Study on Collective Motion of Fish Schools. The Seventh International Conference on Swarm Intelligence: ICSI 2016, Advances in Swarm Intelligence, 9712(1), 131-141.