İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

REAL-TIME SIMULTANEOUS LOCALIZATION AND MAPPING OF MOBILE ROBOTS

M. Sc. Thesis by

Başar DENİZER, B.Sc.

Department : **Mechatronics Engineering**

Programme: **Mechatronics Engineering**

JUNE 2008

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

REAL-TIME SIMULTANEOUS LOCALIZATION AND MAPPING OF MOBILE ROBOTS

M. Sc. Thesis by
Başar DENİZER, B.Sc.
518051022

Date of submission : 2 May 2008

Date of defence examination: 11 June 2008

Supervisor (Chairman): Assist. Prof. Dr. Erdinç ALTUĞ

Members of the Examining Committee: Assist. Prof. Dr. Ümit SÖNMEZ

Assist. Prof. Dr. Levent OVACIK

JUNE 2008

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MOBİL ROBOTLARDA GERÇEK ZAMANLI EŞZAMANLI LOKALİZASYON VE HARİTALAMA

YÜKSEK LİSANS TEZİ
Müh. Başar DENİZER
518051022

Tezin Enstitüye Verildiği Tarih : 2 Mayıs 2008

Tezin Savunulduğu Tarih : 11 Haziran 2008

Tez Danışmanı : Yrd. Doç. Dr. Erdinç ALTUĞ

Diğer Jüri Üyeleri: Yrd. Doç. Dr. Ümit SÖNMEZ

Yrd. Doç. Dr. Levent OVACIK

HAZİRAN 2008

PREFACE

In the scope of this thesis, a robotic system that is capable of mapping an indoor environment and locating itself within the environment is developed. The system is mainly comprised of two parts: a mobile robot and a remote computer. The mobile robot is connected to a remote computer through the wireless RF link which provides transferring most of the processing load to the faster computer and also allows the user to control the robot from this remote computer. The output of the system is the map which the user is able to see on the remote computer.

This study has been performed under supervision of Assist. Prof. Dr. Erdinç Altuğ in Istanbul Technical University. I would like to thank to him for his very valuable guidance suggestions, advice and encouragement at all stages of this work It was a great honor working with him.

I would like to express my sincere feelings to my family members for their encouragements during my study and also I would like to thank my family for their patience and support.

June 2008 **Başar DENİZER**

CONTENTS

ABBREVIATIONS

CI: Covariance Intersection
CTS: Clear To Send
DR: Dead Reckoning
EEPROM: Electronically Erasable Programmable Read-Only Memory
I²C: Inter Integrated Circuit
I/O: Input/Output
MCL: Monte Carlo Localization
MCU: Micro Controller Unit
PWM: Pulse Width Modulation
RF: Radio Frequency
RS-232: Recommended Standard 232
RTS: Request To Send
SCL: Serial Clock
SDA: Serial Data
SLAM: Simultaneous Localization and Mapping
SPI: Serial Peripheral Interface
USART: Universal Synchronous Asynchronous Receiver Transmitter
USB: Universal Serial Bus

TABLE LIST

Page No

FIGURE LIST

Page No

MOBİL ROBOTLARDA GERÇEK ZAMANLI EŞZAMANLI LOKALİZASYON VE HARİTALAMA

ÖZET

Bu çalışmanın amacı çeşitli algılayıcılara sahip mobil robot ile kapalı, bilinmeyen ortamların haritasını çıkarmak ve aynı zamanda robotun kendini bulunduğu ortam içinde konumlandırmasıdır.
Yapılan çalışmada robotun çevre ile olan etkileşimi kızılötesi ve ultrasonik algılayıcılar ile sağlanmaktadır. Ultrasonik algılayıcılar ucuz ve başarılı bir algılayıcı tipi olmasının yanında, yapısından kaynaklanan problemlerden dolayı çalışması zor olan algılayıcı tiplerinden biridir.
Yapılan çalışmalar sırasında bu problemlerin en az seviyeye indirilmesi sağlanmıştır. Kızılötesi algılayıcılar ise yakın mesafeden yaptıkları doğru ölçümlerden dolayı çarpışma önleyici güvenlik sistemi amaçlı kullanılmıştır.
Ortam haritasının çıkarılmasında ultrasonik mesafe ölçerler ve dijital pusula kullanılmıştır. Bununla birlikte robotun konumunun takip edilebilmesi için robotun üzerinde enkoderli motorlar kullanılmıştır.
Robotun konumlandırılması ve harita çıkarma doğruluğu büyük ölçüde tasarımda kullanılan algılayıcı ve eyleyicilere bağlıdır. Algılayıcı ve eyleyicilerin seçiminde boyutları, doğrulukları ve mikroişlemci ile olan arayüzleri dikkate alınmıştır. Algılayıcılar tarafından ölçülen veriler mikroişlemci tarafından alınıp işlenmekte ve daha karmaşık hesaplama, bilgi depolama, konumlandırma, harita çıkarma işlemi yapacak olan bilgisayara kablosuz RF iletişimi ile aktarılmaktadır.
Robotun haritalama algoritmasını test etmek amacıyla çeşitli şekillerden oluşan deneysel ortamlar oluşturulmuştur. Bu ortamlarda yapılan deneyler sonucunda, robotun ortamı keşfi sırasında yaptığı dönüşlerde paletlerinin az miktarda kaydığı görülmüştür. Bu kayma sonucunda az miktarda odometry hatası oluşmaktadır. Bu odometry hatalarının birikerek artması da oluşturulan haritada ve robotun harita üzerindeki konumunda hatalar oluşturmaktadır. Bu hataları en aza indirmek veya yoketmek için robotun paletlerinin yüzeyi pürüzlendirebilir, robotun kaymasını önleyici frenleme algoritmaları geliştirilebilir veya varolan sensörlere ek olarak daha hassas başka sensörler kullanılabilir.

REAL-TIME SIMULTANEOUS LOCALIZATION AND MAPPING OF MOBILE ROBOTS

SUMMARY

The aim of this study is localization and mapping of the unknown indoor environments using mobile robot that have various sensors. The mobile robot provides interaction with the surroundings by using infrared and ultrasonic sensors. The ultrasonic sensors are cheap and successful but also they have some problem arise from the structure of them.

These problems are reduced to the lower level during the study. Infrared sensors perform accurate measurements from the closer range therefore they are used for collision avoidance security purposes.

Environment mapping is generated by using ultrasonic range finders and digital compass. In addition to this, to observe the localization of the robot, motors with encoders are used.

Localization of the robot and accuracy of mapping are mostly related to used sensors and actuators of the robot design. The selection of the sensors and the actuators are considered according to their sizes, accuracies, interfaces to the microprocessor. Data measured by the sensors that is received and processed at the microprocessor. Then, data processed by the microprocessor is sent to the remote computer via RF communication for the complicated computation, data storage, localization and generating map.

Experimental environments that have different shapes are set up so as to test the map generation algorithm of the robot. According to the experiment results, tracks of the robot slips a little at the time of the robot turnings. Due to this slippage, odometry error occures. The accumulation of odometry errors causes errors on the generated map and the robot localization in the map. To minimize or destroy these errors on the map, the track surfaces of the robot can be roughened, slippage preventive brake algorithms can be developed or in addition to the sensors on the board, more sensitive sensors can be used to improve the odometry.

1. INTRODUCTION

The automated movement of an autonomous mobile robot is the main research topic for many years, that's why many experiments are performed on this issue.

Robotics is the science of sensing and manipulating the physical world through computer controlled devices. Example of successful robotics systems includes mobile platforms for planetary exploration. Mobile robots are originally designed as reprogrammable, multifunctional devices and to move materials in industrial environments. Mobility has been greatly improved. It was expected at the beginning of the nineties that in 2000 there will be 50000 independently operating autonomous robots in the production areas [1].

A robot obviously is made from hardware and the functionality of a robot's sensors and actuators affects its behavior greatly. Mobility is almost pointless without the ability of goal-directed motion such as navigation.

The mobile robot navigation is taking some inspiration from the most successful navigators on earth: living beings highlight the mechanisms used in successful robot navigation systems: self organization, emergent functionality and autonomous mapping of the environment that the robot perceives it.

The proof of a robot control program is still in physical experiments. To know the robot behavior will result from a specific robot control program. Numerical models of the complex interaction between robot and environment interaction are still imprecise approximations, due to the sensitivity of robot sensors to variations in environment conditions. Therefore, the problem of robotic mapping has received considerable attention over the recent years. Mapping is the problem of the generating models of robot environments from sensor data. The importance of problem is due to the fact that many successful mobile robot systems require maps for their operations.

In this study, a robotic system that is capable of mapping an area of its environment and locating itself within the environment is developed. The system is mainly

comprised of a mobile robot and a remote computer as shown in Figure 1.1. The mobile robot connects to a remote computer through the wireless RF link which enables transferring most of the processing load to the faster computer and also allows the user to control the robot from this remote workstation. The output of the system is the map which the user is able to see on the computer screen. Also, various sensor data can be seen on the computer.

The exploration and map-building capabilities of a robot are strongly dependent of its sensors and actuators. Cheap, small but reliable sensors and actuators are selected for a number of purposes. An ultrasonic range finder and a digital compass are used to map the area and orient the robot within it. Furthermore, each wheel on the robot has an incremental encoder which enables the robot to keep track of the robot location. When choosing sensors and actuators the size, accuracy and electronics interface of them are especially taken into account. All the data acquired from the sensors are taken and processed by the ATMEL's ATmega128 microcontroller on the lowest level and sent to the remote computer through wireless RF link for running complex calculation, storing data and interfacing with a user.

1.1 Robot Definitions

The word "robot" stems from a play from 1921 called "R.U.R." ("Rossum's Universal Robots") by the Czech playwright Karel Capek. Capek derived the word "robot" from the Czech "robota", meaning "forced labour" In 1942; the word "robotics" appeared for the first time in a novel called "Runaround" by the American scientist and writer Isaac Asimov [2].

The Robotics Institute of America (RIA) considers only the machines which are at least in class 3 as robots:

A robot is re-programmable, multi-functional device designed to move material, parts, tools, or specialized device through variable programmed motions for the performance of a variety of tasks.

Mobile Robot: Most robots in industrial use today are assembly robots that operate within a bounded workspace, and can not move.

Mobile robots are entirely different: they can change their location through locomotion. The most common type of mobile robot is the Automated Guided Vehicle.

AGV's operate in specially engineered modified environments, they are inflexible and fragile. Altering the route is costly and any changes can lead to failure in a mission. Therefore, autonomous mobile robots are built for obtaining fixed-program mobile robots.

1.2 History of Mobile Robots

Mobile robotics research has played a key role in the application of robots in our world. At Stanford University Nils Nilsson [3] developed the mobile robot SHAKEY in 1969. This robot possessed a visual range finder, a camera and binary tactile sensors. It was the first mobile robot to use artificial intelligence to control its actions. Its main objective was to navigate through highly structured environments such as office buildings. The JPL Lunar rover [4], developed in the 1970s at the Jet Propulsion Laboratory, was designed for planetary exploration. Using a TV camera, laser range finder and tactile sensors, the robot categorized its environment as traversable, not traversable and unknown. In the late 1970s Hans Moravec [5] developed CART in the Artificial Intelligence laboratory at Stanford. The robot was capable of following a white line on a road. A television camera mounted on a rail on the top of CART took pictures from several different angles and relayed them to a computer, which performed obstacle avoidance by gauging the distance between CART and obstacles in its path. In 1994, the CMU Robotics Institute's Dante II [6], a six-legged walking robot, explored the Mt. Spurr volcano in Alaska to sample volcanic gases. In 1997 NASA's Mars Pathfinder delivered the Sojourner rover [7] to Mars. Sojourner sent back images of its travels on the distant planet. Also in this same year, Honda showcased the P3 [8], an extraordinary prototype in humanoid robotic design. Currently NASA's rovers SPIRIT and OPPORTUNITY [9] are exploring the Martian soil for signs of water.

1.3 Types of Mobile Robots

The simplest case of mobile robots are wheeled robots, as shown in Figure 1.1. Wheeled robots comprise one or more driven wheels (drawn solid in the figure) and have optional passive or caster Wheels (draw hallow) and maybe even steered wheels (drawn inside a circle). Most designs require two motors for driving (including steering) a mobile robot.

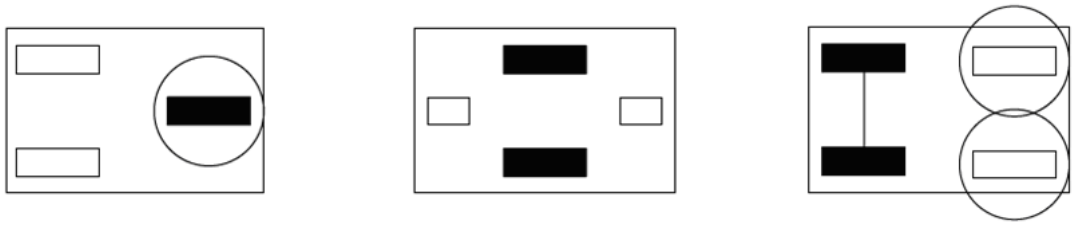

Figure 1.1 : Wheeled robots

The design on the left-hand side of Figure 1.1 has a single driven wheel that is also steered. It requires two motors, one for driving the wheel and one for turning. The advantage of this design is that the driving and turning actions have been completely separated by using two different motors. Therefore, the control software for driving curves will be very simple. A disadvantage of this design is that the robot cannot turn on the spot, since the driven wheel is not located at its center.

The robot design in the middle of Figure 1.1 is called “the differential drive” and is one of the most commonly used mobile robot designs. The combination of two driven wheels allows the robot to be driven straight, in a curve, or to turn on the spot. The translation between driving commands, for example a curve of a given radius, and the corresponding wheel speeds has to be done using software. Another advantage of this design is that motors and wheels are in fixed positions and do not need to be turned as in the previous design. This simplifies the robot mechanics design considerably.

Finally, on the right-hand side of Figure 1.1 is the so-called “Ackermann Steering” , which is the standard drive and steering system of a rear-driven passenger car. We have one motor for driving both rear wheels via a differential box and one motor for combined steering of both front wheels.

It is interesting to note that all of the these different mobile robot designs require two motors in total for driving and steering.

A special case of a wheel robot is the omni-directional “Mecanum drive” robot in Figure 1.2, left. It uses four driven wheels with a special wheel design.

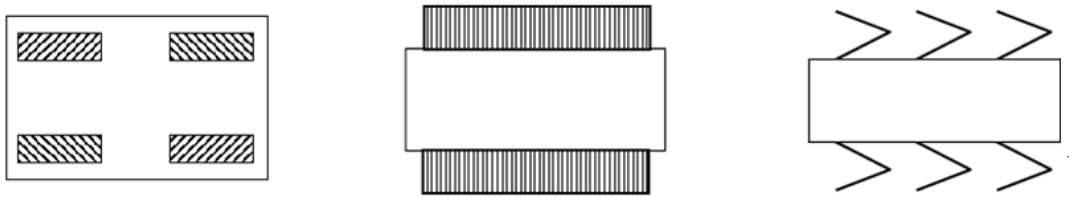

Figure 1.2 : Omni-directional, tracked, and walking robots

An advantage of the omni-directional robot is that there is no constraint about movement of the robot. It can go in everywhere that it fit into, on the contrary other wheeled robots must use complex parking methods. Therefore the control of the omni-directional robot is easy.

One disadvantage of all wheeled robots is that they require a street or some sort of flat surface for driving. Tracked robots (see Figure 1.2, middle) are more flexible and can navigate over rough terrain. However, they cannot navigate as accurately as a wheeled robot. Tracked robots also need two motors, one for each track.

1.4 History of SLAM

Learning maps is a fundamental task of mobile robots and a lot of researchers focused on this problem. In the literature, the mobile robot mapping problem is often referred to as the simultaneous localization and mapping (SLAM) problem [36–43]. In general, SLAM is a complex problem because for learning a map the robot requires a good pose estimate while at the same time a consistent map is needed to localize the robot. This dependency between the pose and the map estimate makes the SLAM problem hard and requires searching for a solution in a high-dimensional space.

Murphy, Doucet, and colleagues [42, 44] introduced Rao–Blackwellized particle filters (RBPFs) as an effective means to solve the SLAM problem. The main problem of Rao–Blackwellized particle filters lies in their complexity, measured in terms of the number of particles required to learn an accurate map. Either reducing this quantity or improving the algorithm so that it is able to handle larger sample sets is one of the major challenges for this family of algorithms.

One of the greatest difficulties in deploying SLAM is the computational cost. The state space is proportional to the number of beacons and the computational and storage costs scale nonlinearly with the size of the state space. One family of algorithms which addresses this issue is based on Covariance Intersection (CI). In the paper [45], algorithms, which exploit partial cross correlation information, are presented. CI has the advantage that it requires no special structure or management, which is a requirement of many other suboptimal SLAM algorithms. However, in its raw form CI leads to extremely conservative estimates and is of limited use. In this article [45], an overview of the CI algorithm is given. This begins with the algorithm's basic form, and shows how a number of refinements can be made to improve its performance significantly.

SLAM and navigation technique are crucial problems for autonomous mobile robots. Until now, many researches on SLAM and navigation algorithms are developed [40, 46, 47]. If we think of real applications, like home cleaning robots and service robots require a successful integration of SLAM solution and navigation technique. Moreover the algorithms should guarantee their performance even using cheap sensors. Several attempts to integrate the SLAM and navigation algorithms are suggested in a sense of integrated exploration [48, 49]. These algorithms focus on the development of navigation solution which reduces the uncertainty of SLAM estimate or investigates unexplored regions.

1.5 Problem Statement and Thesis Organization

The problem of robotic mapping is that of acquiring a spatial model of a robot's environment. Maps are commonly used for robot navigation [30]. To acquire a map, robots must have sensors that enable it to perceive the outside world. Sensors commonly brought to bear for this task include cameras, range finders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, and GPS. However,

all these sensors are subject to errors, often referred to as measurement noise. More importantly, most robot sensors are subjected to strict range limitations. For example, light and sound cannot penetrate walls. These range limitations makes it necessary for a robot to navigate through its environment when building a map. The motion commands issued during environment exploration carry important information for building maps, since they convey information about the locations at which different sensor measurements were taken. Robot motion is also subjected to errors, and the controls alone are therefore insufficient to determine a robot's pose (location and orientation) relative to its environment.

The other challenge arises from the fact that robots must choose their way during mapping. The task of generating robot motion in the pursuit of building a map is commonly referred to as robotic exploration. While optimal robot motion is relatively well-understood in fully modeled environments, exploring robots have to cope with partial and incomplete models. Hence, any effective exploration strategy has to be able to accommodate surprises that might arise during map acquisition. For this reason, exploration is a challenging planning problem, which is often solved suboptimally via simple learning methods. When choosing where to move, various quantities have to be traded off like the expected gain in map information, the time and energy it takes to gain this information, the possible loss of pose information along the way, and so on. Furthermore, the underlying map estimation technique must be able to generate maps in real-time, which is an important restriction that rules out many existing approaches.

As noted above, the literature refers to the mapping problem often in conjunction with the localization problem, which is the problem of determining a robot's pose. The reason for suggesting that both problems (the problem of estimating where things are in the environment and the problem of determining where a robot is) have to be solved in conjunction will become a bit more obvious below, when we state the basic statistical estimators that underlie all state-of-the-art techniques. In essence, both the robot localization and the map are uncertain, and by focusing just on one the other introduces systematic noise. Thus, estimating both at the same time has the pleasing property that both the measurement and the control noise are independent with regards to the properties that are being estimated the state. The robot mapping problem is like a chicken and egg problem: If the robot's pose was known all along,

building a map would be quite simple. Conversely, if a map of the environment is known, there exist computationally efficient algorithms for determining the robot's pose at any point in time [30]. In combination, however, the problem is much harder.

The main purpose of this work is localization and mapping of the unknown indoor environments by using the designed tracked mobile robot that has many sensors.

In the second chapter of this work, design and assembly of the mobile robot are briefly described. First mechanical construction of the mobile robot is explained. Then, sensors and actuators mounted on the mobile robot for exploration and map building purposes are investigated. In the remaining sections of this chapter, the mobile robot's control system and programming are described respectively. Third chapter is dedicated to the localization and mapping theories. A variety of methods used in robotic mapping application are explained.

2. DESIGN OF THE TRACKED ROBOT

The tracked robot's overall architecture is shown in Figure 2.1. The system can be broken down into four hierarchical levels. First, several sensors provide information about the robot and its location in its environment. This information is sent to the second level, a high level processor that acts as the "brain" of the robot by interpretting the information and deciding how to move the robot. This main processor generates drive signals and sends to the third level, motor driver, that translates the drive signals into power signals. The power signals are sent to the last level, the motor, that drive the car.

The parts of the tracked robot that is used for localization and mapping, is mentioned in the following sections.

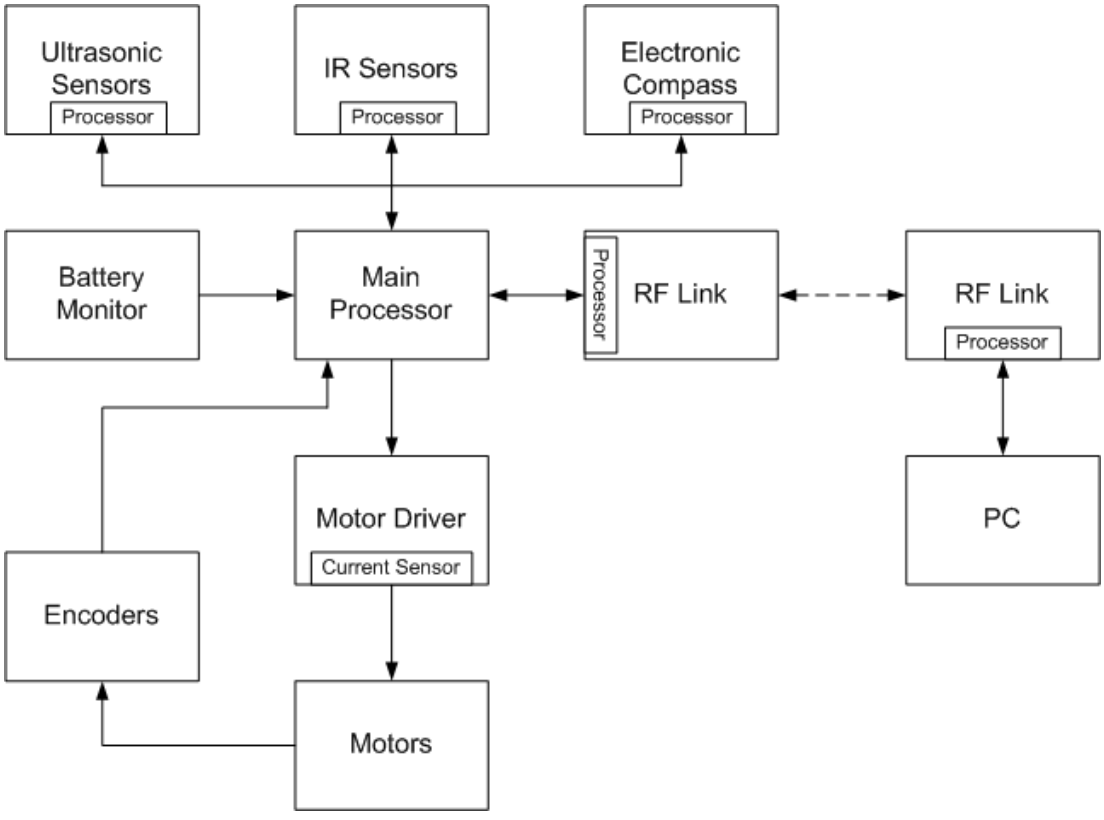

Figure 2.1: The overall architecture of the system

2.1 Mechanical Construction

A tracked mobile robot can be seen as a special case of a wheeled robot with differential drive. In fact, the only difference is the robot's better maneuverability in rough terrain and its higher friction in turns, due to its tracks and multiple points of contact with the surface.

Figure 2.2 shows BSRTrack, a model tank that was modified into a mobile robot. A tracked vehicle has two driving motors, one for each track.

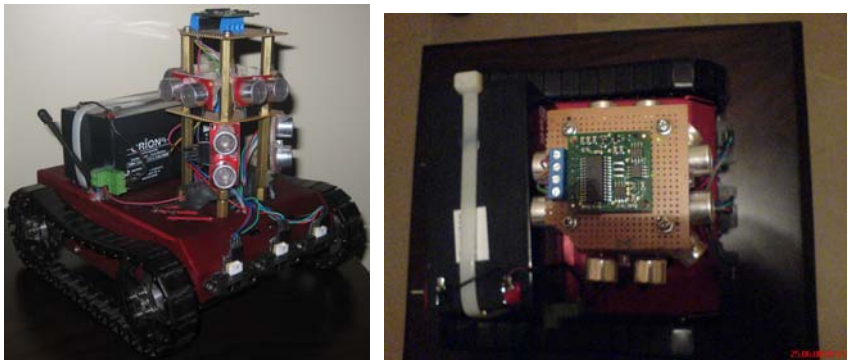

Figure 2.2: BSRTrack robot and top view with sensors attached

BSRTrack is equipped with a number of sensors required for navigating rough terrain. Most of the sensors are mounted on the top of the robot. In Figure 2.2, the following are visible, top: ultrasonic sensors and digital compass, front: infrared sensors. Electronic controller is placed in the BSRTrack. The sensors used on this robot are explained in detail later.

2.1.1 Differential Drive

The differential drive design has two motors mounted in fixed positions on the left and right side of the robot, independently driving one wheel each. Since three ground contact points are necessary, this design requires one or two additional passive caster wheels or sliders, depending on the location of the driven wheels. Differential drive is mechanically simpler than the single wheel drive, because it does not require rotation of a driven axis. However, driving control for differential drive is more complex than for single wheel drive, because it requires the coordination of two driven wheels.

The minimal differential drive design with only a single passive wheel cannot have the driving wheels in the middle of the robot, for stability reasons. So when turning on the spot, the robot will rotate about the off-center midpoint between the two driven wheels. The design with two passive wheels or sliders, one each in the front and at the back of the robot, allows rotation about the center of the robot. However, this design can introduce surface contact problems, because it is using four contact points.

Figure 2.3 demonstrates the driving actions of a differential drive robot. If both motors run at the same speed, the robot drives straight forward or backward, if one motor is running faster than the other, the robot drives in a curve along the arc of a circle, and if both motors are run at the same speed in opposite directions, the robot turns on the spot.

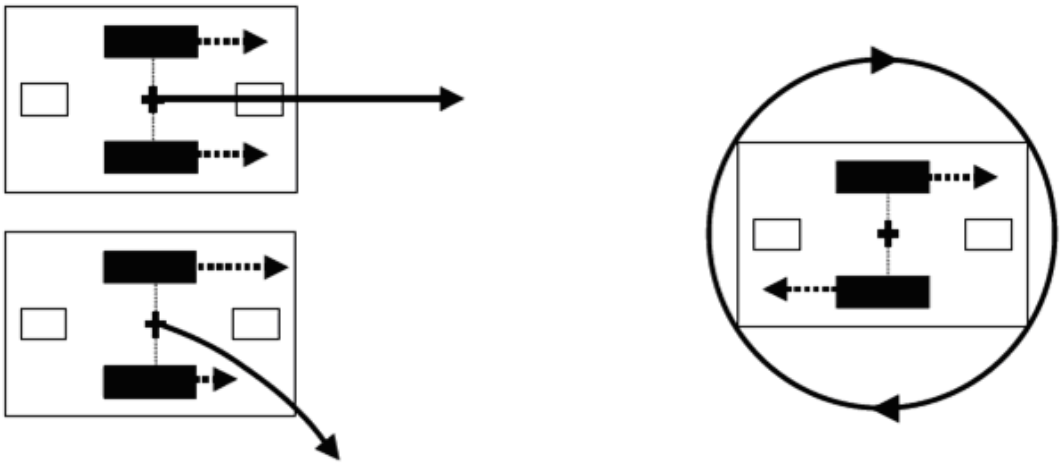

Figure 2.3: Driving and rotation of differential drive

- Driving straight, forward: vL = vR, vL > 0
- Driving in a right curve: vL > vR, e.g. vL = 2·vR
- Turning on the spot, counter-clockwise: vL = –vR, vL > 0

vL: velocity of the left wheel

vR: velocityof the right wheel

A mobile robot is built using a differential drive. It carried a PWM controlled dual full bridge driver circuit for driving motors. The robot has a differential drive

actuator design, using two Hennkwell motors with encapsulated gearboxes and encapsulated encoders. The robot is equipped with a number of sensors:

- Shaft encoders (2 units)
- Infrared proximity sensors (3 units)
- Ultrasonic range sensors (6 units)
- Digital compass (1 unit)
- Current Sensors (2 units)

2.2 Actuators and Sensors of the Tracked Robot

Sensors are devices that can sense and measure physical properties of the environment, such as temperature, luminance, distance, resistance to touch, weight, size. Sensors deliver low level information about the environment the robot is working in. This information is noisy, often contradictory and ambiguous.

Table 2.1: Actuators and sensors used on the tracked robot

Component	Model	Manufacturer
Ultrasonic range finder	SRF08	Devantech
Infrared range finder	GP2D12	Sharp
Electronic Compass	CMPS03	Devantech
Motor with quadrature encoder	HG37D67WE12-052	Hennkwell

This part describes the sensors and actuators used in the tracked robot, their strengths, limitations and applications.

2.2.1 DC Gear motor with encoder

The first step when building robot hardware is to select the appropriate motor system. The best choice is a motor combination containing,

- DC motor
- Gearbox
- Optical or magnetic encoder

Using encapsulated motor systems has the advantage that the solution is much smaller than those using separate modules, plus the system is dust-proof and shielded against stray light required for optical encoders. The disadvantage of using a fixed assembly like this is that the gear ratio may only be changed with difficulty, or not at all. In the worst case, a new motor/gearbox/ encoder combination has to be used.

A magnetic encoder comprises a disk equipped with a number of magnets and one or more Hall-effect sensors. An optical encoder has a disk with black and white sectors, an LED, and a reflective or transmissive light sensor. If two sensors are positioned with a phase shift, it is possible to detect which one is triggered first (using a magnet for magnetic encoders or a bright sector for optical encoders). This information can be used to determine whether the motor shaft is being turned clockwise or counterclockwise.

For the reasons mentioned above, Hennkwell HG37D67WE12-052 is selected for the actuator of the robot. This 7.2V gearhead motor features 7.2 Kg-cm of torque, 160 rpm no load speed, a 6mm drive shaft, and comes with a built in quadrature encoder. It has a ratio of 1:52 gear reduction. It's no load current draw is 300mA and max current draw is 2.0A.

Standard DC motors revolve freely, unlike for example stepper motors. Motor control therefore requires a feedback mechanism using shaft encoders as shown Figure 2.4.

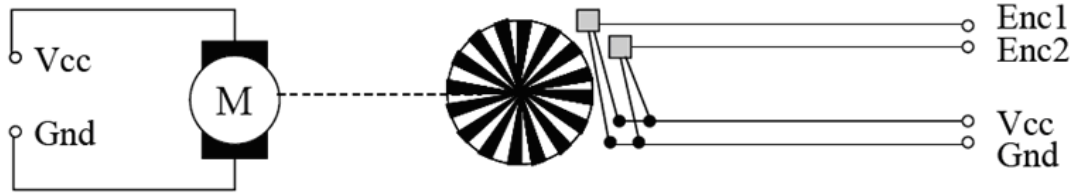

Figure 2.4: The Motor-encoder combination

The DC motor drives the wheels of the vehicle. A DC motor converts DC current into torque. The more current that is sent through the windings in the motor, the more torque it produces and the faster the shaft turns. Since current is proportional to voltage, the shaft speed is proportional to the voltage across the motor. However, motors are typically controlled by digital electronics that only output 0V and 5V.

However, if you apply 0V or 5V to the motor, it will only run at two speeds. To control the motor using a digital signal, a PWM signal is used. PWM signals take advantage of the fact that DC motors can not respond as quickly as digital electronics can operate. So, if the motor is given a signal, as in Figure 2.5, which is changing between 0V and 5V at a speed of 2 kHz with a 50 percent duty cycle (defined as the ratio of the high time to the period of the signal), the motor can not possibly turn on and off so quickly. The result is that the motor acts as if it is receiving 2.5V. In general, the voltage seen by the motor is the average over one period of the PWM signal.

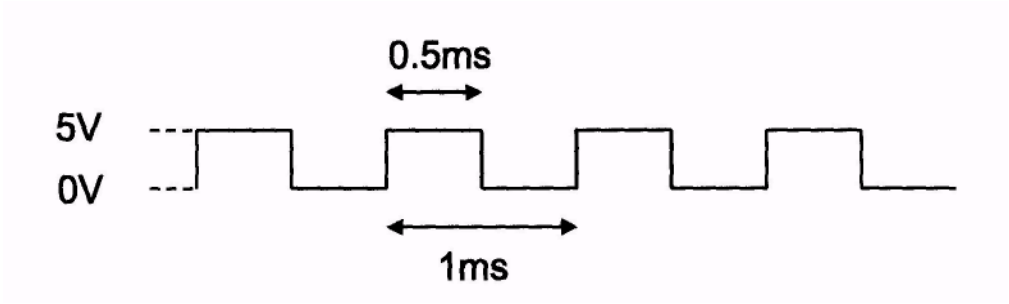

Figure 2.5: The PWM control signal with a 50-percent duty cycle

Another issue with motor control through digital electronics is that such circuitry can not deliver the current a motor needs. The impedance of a DC motor is only a few ohms. If a digital processing chip is connected directly to a motor, the chip will surely be damaged. Special interface circuitry known as a H-bridge is usually used to convert the low-power digital signal to a higher power one and also isolate the motor from the more sensitive digital electronics. A typical H-bridge is shown in Figure 2.6. This circuit works by turning on opposite transistors simultaneously; for example, transistors A and D are turned on while B and C are turned off. In such a configuration, the current flows through the motor in the direction of the arrow. If the situation is reversed (B and C are on while A and D are off), the current flows in the other direction, causing the motor to turn opposite of the previous direction. The transistors are turned on and off by the digital electronics. In designing an H-bridge controller, it is important that transistors A and C are not turned on simultaneously, as that would cause a short between the power and the ground. Likewise, transistors B and D should not be turned on at the same time.

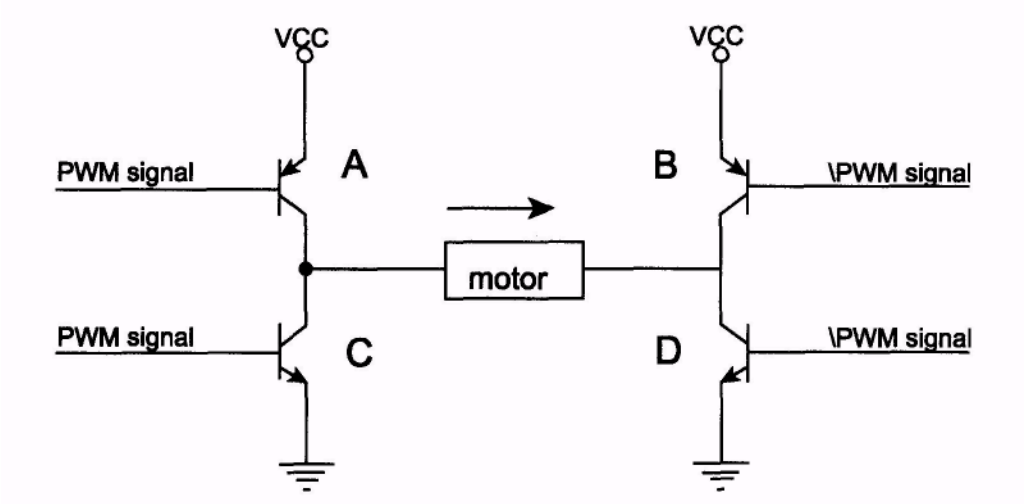

Figure 2.6: A H-bridge circuit used to drive a motor

2.2.2 Infrared sensors

In this Project, Infrared sensors perform accurate measurements from the closer range therefore they are used for collision avoidance security purposes.

During navigation process, mobile robot is coming closer to an obstacle; it is detected by infrared sensors.

In this study three pieces of infrared sensors are used and mounted on the normal line to the trucked robot's front side. By using the infrared sensors in front of the tracked robot, objects can be detected in a range of 10 to 80 cm. Sharp GP2D12 is used for each infrared sensor on the robot shown in Figure 2.7.

Figure 2.7: The Sharp GP2D12 infrared range finder

The Sharp GP2D12 Infrared Range Finder Technical Characteristics:

- Voltage : 4.5- 5.5V
- Low Current : 35mA (typ.) - 55 mA (max)
- Max Range : 80 cm
- Min Range : 10 cm
- Accuracy : 3 cm

2.2.2.1 Operation principles of the infrared sensors

Transmitted infrared light collides to the object and reflects back to the sensor. The differences of the infrared light energy of the transmitted and received infrared lights are processed by the sensor signal processing circuit and then an analog voltage generated. So the distance measurement of an object is obtained as an analog voltage.

The distance measurement process consists of two parts. First measurement process perform without transmitted infrared signal, thus the system (sensor) determines the amount of energy on the environment. Second measurement process performs with transmitted infrared signal.

To aim by using these steps, performed measurements are less affected from the amount of energy on the environment.

2.2.2.2 Distance measurement problems with the infrared sensors

From time to time the measurement performed is affected negatively from the amount of energy on the environment.

Furthermore, measurements error can be occurred because of the illumination of the environment, the shape of the object, the surface color. The infrared sensors give the different measurement results according to the illumination of the environment. Transmitted infrared signal reflects from the object according to the geometrical structure of it and how much infrared signal return back to the sensor is an important parameter. The infrared signals reflect to the different directions when they are received without perpendicular normal line of the object. The surface color of the

object and structure of the object are important for the measurement accuracy. The black surfaces absorb the infrared signals more than white surfaces. Therefore less infrared signal returns back to the sensors as a result of this, the black object is perceived far away from the actual distance. And also same case is valid for the shiny and non-shiny surfaces

2.2.3 Ultrasonic sensors

In this study, ultrasonic sensors are used for gathering the information about the environment so as to generate environment map and navigation process. Environment map illustrates the boundaries of the processing area, and also indicates the obstacles inside the area.

The Devantech SRF08 Ultrasonic Range Finder shown in Figure 2.8 is used in the robot.

Figure 2.8: The Devantech SRF08 ultrasonic range finder

The Devantech SRF08 Ultrasonic Range Finder is unique in that it has a separate transmitter and receiver. This allows for the smallest minimum detectable distance. The frequency of the ping is 40 kHz. The reason for a 40 kHz frequency sound wave is to reduce the chances of false echoes. For example, it is unlikely that a 40 kHz sound wave will come from any other source other than the actual ultrasonic sensor itself. The external timing circuit looks for a 40 kHz return signal to identify it as an echo. The SRF08 offers precise ranging information from roughly 3cm to 6 meters. This range and minimal power requirements, 5 volts, make this an ideal ranger for robotics applications.

SRF08 Ultrasonic Range Finder Technical Characteristics:

- Voltage - 5V

- Low Current - 15mA Typ. 3mA Standby
- Frequency: 40KHz
- Max Range: 6 m
- Min Range: 3 cm
- Accuracy: 1 cm
- Small Size - 43mm w x 20mm d x 17mm h

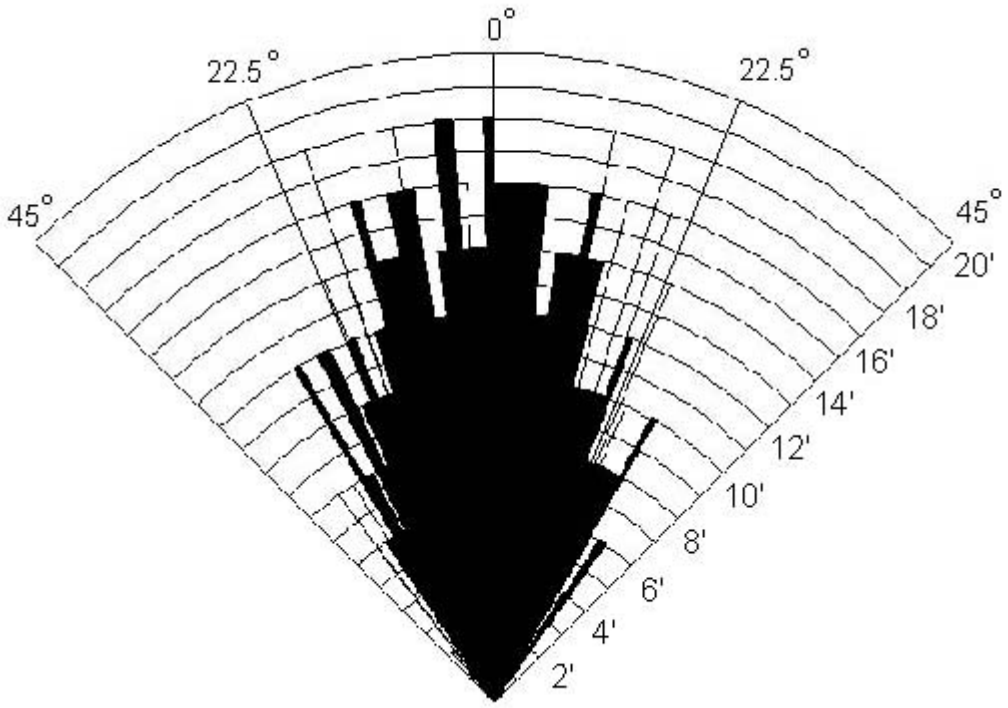

Figure 2.9: The Devantech SRF08 ultrasonic sensor beam pattern [22]

2.2.3.1 Operation principles of the ultrasonic sensors

Distance measurement by ultrasonic sensors, is calculated from the flight duration of the ultrasonic sound wave on the air. Calculation of the distance measurement is like as equation 2.1

$$D = C_{sound} * t / 2 \quad \textbf{(2.1)}$$

"D" points the measurement distance, "C_{sound}" points the speed of the ultrasonic sound wave on the air, and "t" points the traveling time duration of the ultrasonic sound wave on the air.

Ultrasonic sensors are preferred mostly to generate map and navigate in the indoor environment for following reasons [10, 11].

Ultrasonic sensors are low cost components. Cost is very important criterion for many robots in daily use. Ultrasonic sensors are cheaper and able to find easily than lasers and camera systems. By using ultrasonic sensors, the total cost can be decreased.

The Ultrasonic sensors are rugged and are not needed to maintenance. Ultrasonic sensors take small places and break down rarely because of their structures from the other kinds of sensor. TTL voltage levels are used to operate the ultrasonic sensors as result of this, it is easy to interface the ultrasonic sensors with the other processor units. Ultrasonic sensors can be connected to the inputs and outputs of the microprocessors directly.

The long distances are measured accurately by the ultrasonic sensors. The accuracy of the ultrasonic sensors is maximum 1 % for the long distance measurements [12, 13, 14].

2.2.3.2 Physical definitions about the ultrasonic sensors

The only difference between the ultrasonic sound waves and the sound waves is the frequency level of the wave. The ultrasonic sound waves provide the equation 2.2

$$C_{sound} = \lambda / T = \lambda f \qquad \textbf{(2.2)}$$

"λ" points the ultrasonic wavelength, "T" points the period of the ultrasonic sound wave, and "f" points the frequency of the ultrasonic sound wave.

Ultrasonic sound waves have reflection and absorption properties like other sound waves. The speed of the ultrasonic sound wave on the air is very important because the distance measurement by ultrasonic sensors is calculated from the flight duration of the ultrasonic sound wave on the air. C_{sound} is the speed of sound wave on the air. The speed of sound wave varies depending on the media and the temperature. The relationship of the motion of the ultrasonic sound wave on the air and the temperature of the media (T_{media}) is like as equation 2.3 and the unit of T_{media} is Kelvin.

$$Csound = 331.4 \sqrt{(Tmedia / 273)} \qquad \textbf{(2.3)}$$

2.2.3.3 Distance measurement problems with the ultrasonic sensors

Improper distance measurement can be performed by the ultrasonic sensors in the complex and bounded areas. By means of the bounded area that is less than 6 meters and consists of many objects in it. There are two main problems about the distance measurements. These are angular uncertainty and reflection in measurements by using ultrasonic sensors.

Angular uncertainty problem

When data with r value is obtained from the ultrasonic sensors, the angular position of the object that is on the r distance from the ultrasonic sensor is not known. The ultrasonic sensors only give information about the distance of the object. Thus, the object with r distance is not known where it's angular position. The angular uncertainty situation is shown in Figure 2.10. The same distance measurement (R) is obtained for each ball in Figure 2.10. The measurement is not known for which one of the balls.

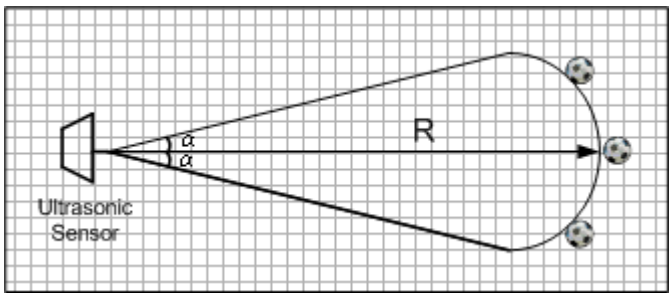

Figure 2.10: Angular uncertainty problem for the ultrasonic sensors

Reflection Problem of Sound Waves

Reflection problem of sound waves is as follows. Transmitted sound waves from sensor collide to the object but they can not return back directly to the sensor. They also collide and reflect from the other object in the environment and then return back to the sensor. As a result, the flight duration of the ultrasonic sound wave on the air increases and the distance measured is longer than as it shall be.

Sensor axis and surface of the object is perpendicular to each other in Figure 2.11. In this situation, most of the transmitted sound waves return back to the ultrasonic

sensor. Sensor axis and surface of the object is not perpendicular to each other in figure 2.12. Therefore, most of the transmitted sound waves reflect from the surface of the object and they are scattered to the other areas and then return back to the ultrasonic sensor. As a consequence the distance measurement is longer than as it shall be or can not be measured by the ultrasonic sensor. The amount of the sound waves that return back to the ultrasonic sensor changes according to the position of the object surface and the shape of the object.

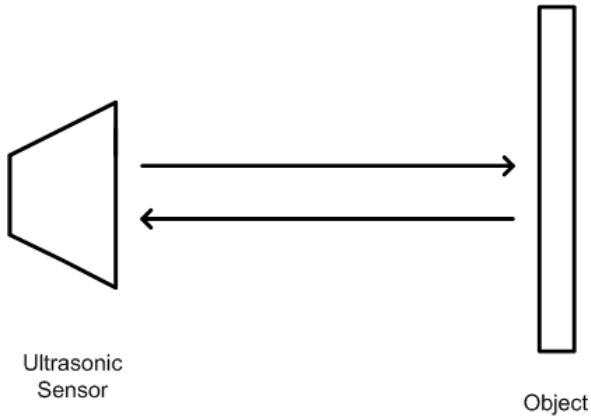

Figure 2.11: A collision perpendicular to the object

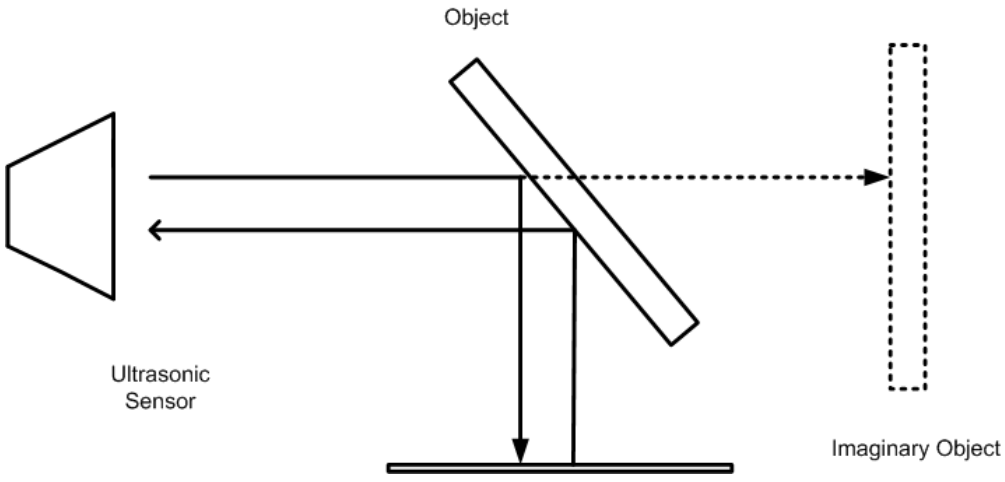

Figure 2.12: A collision non-perpendicular to the object

Reflection problem of sound waves becomes fact that the angle value between normal line of the transmitted sound wave and surface of the measuring object is greater than a critical value.

The critical value is between 7-8 to 90 degrees in rigid and rough surfaces and almost 25 degrees in the smooth surfaces [15, 16].

2.2.4 Compass Module

A compass provides a way for the robot to acquire absolute information about its orientation. This information is used for navigation and mapping processes. In open areas, compasses are very reliable, and once calibrated to local magnetic north, they are also accurate. On the other hand, when it is used indoor, because of the magnetic fields from electrical wiring, structural steel in buildings, and even the metal components of the robot itself can produce large errors in the compass reading.

The Devantech CMPS03 Compass Module shown in Figure 2.13 is used in the robot as a heading sensor. The compass uses two pieces of the Philips KMZ51 magnetic field sensor, which is sensitive enough to detect the Earths magnetic field. The output from two of them mounted at right angles to each other is used to compute the direction of the horizontal component of the Earths magnetic field [17].

Figure 2.13: The Devantech CMPS03 compass module

The Devantech CMPS03 Compass Module Technical Characteristics:

- Voltage: 5VDC only required
- Current: 20mA
- Resolution: 0.1°
- Accuracy: 3-4° approx. after calibration
- Output 1: Timing Pulse 1ms to 37ms in 0.1 ms increments

- Output 2:12C Interface, 0-255 and 0-3599, SCL speed up to 1MHz
- Small Size: 32mm x 35mm

2.2.5 Radio Frequency Communication Module

The communication between the computer and the Robot is realized by radio frequency link. For this purpose, UDEA UTR-C12U Transceiver is used by the Robot and the computer. The UDEA UTR-C12U transceiver provides high performance, simple to use radio devices that can transfer data over a range of up to 800 meters Line Of Sight (LOS) and operates at 434MHz.

Figure 2.14: The UDEA UTR-C12U Transceiver

The UDEA Transceiver is a complete sub-system that combines a high performance very low power RF transceiver, a microcontroller and a voltage regulator as depicted in Figure 2.15. The Serial Data Input and Output operate at the standard 2400 bps and it can operate at the 38400 bps maximum optionally[18].

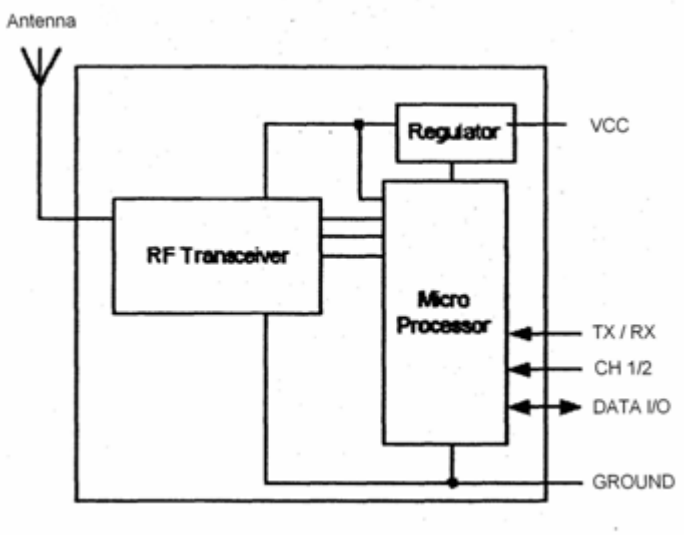

Figure 2.15: The RF transceiver block diagram

Any other UDEA Transceiver within range that 'hears' the transmission will decode the message and place the recovered data within a receive buffer that can then be unloaded to the receiving host for processing and interpretation. Transmission and reception are bi-directional half duplex i.e. transmit or receive but not simultaneously.

To be able to establish communication between the desktop computer and the robot, the computer also needs to be attached a RF communication module as shown in Figure 2.16. For this purpose, Telemetry module which is the custom designed for this project is used. The module is powered from an AC/DC adaptor.

Figure 2.16: Serial telemetry module

The main part of this module is the UDEA UTR-C12U Transceiver. To use the serial telemetry module, the computer has to have a serial port or a USB to serial cable have to be used.

Application software accesses the Windows Com Port using the Microsoft Visual Studio.NET Library. The COM port of the computer should be set up for 2400 baud, 8 data bits, no parity and one stop bit.

2.3 Designing the Control System of the Tracked Robot

Today most of the robots are controlled by microcontrollers (MCUs). Microcontrollers are like the microprocessor (Central processing unit, or CPU) inside large home computers. MCUs are slow and can address less memory than CPUs, but they are designed for real world control problems and are both

inexpensive and easy to use. One of the biggest differences between CPUs and MCUs is the number of external components needed to operate them. MCU can often run with zero external parts. ATMEL's high performance flash ATmega128 microcontroller illustrated in Figure 2.17 is used as the brain of the robot in this project because of the advantages it offered. Before discussing how control circuit is designed for the tracked robot and how software is developed for it, a basic understanding of what features the ATMEL ATmega128 provides will be discussed. Later, how robot microcontroller is integrated into the mobile robot will be introduced. An important part of this is showing how different sensors, actuators, and peripherals are wired to the microcontroller as well as interfaced to.

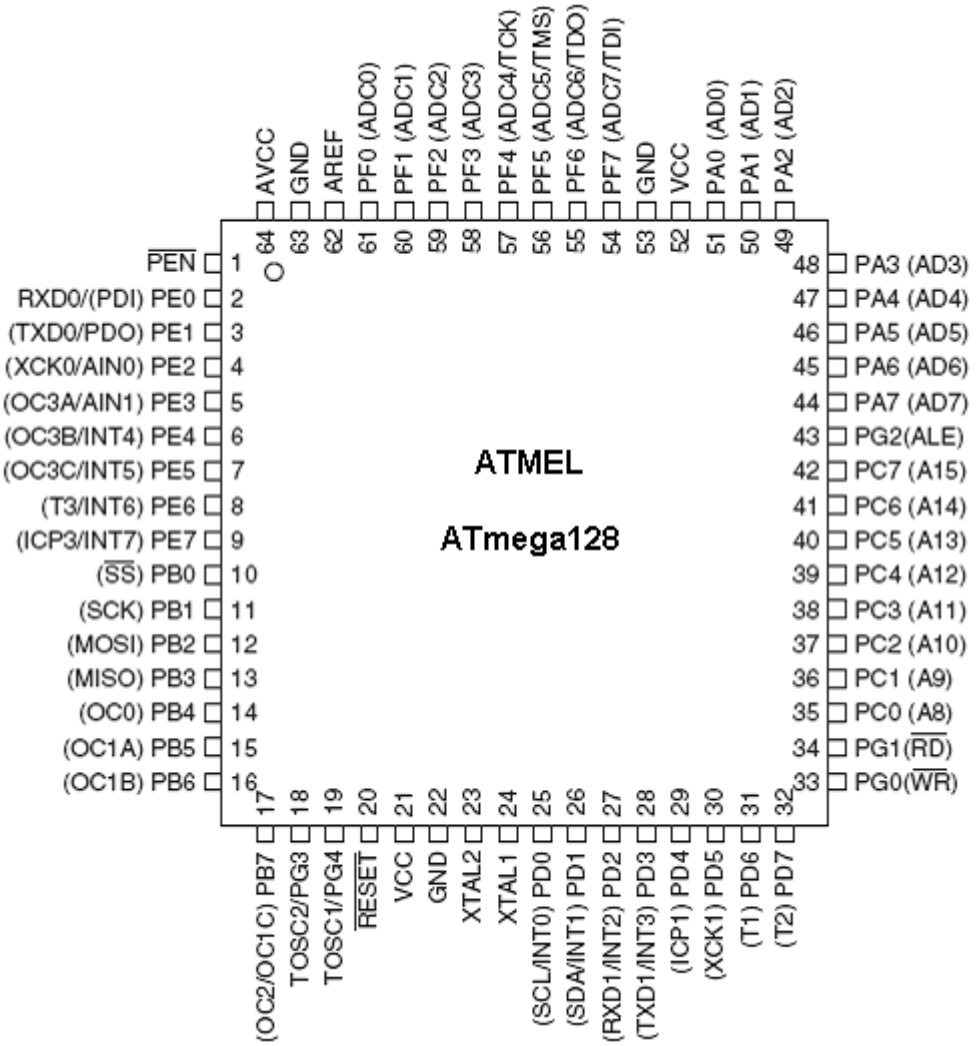

Figure 2.17: ATMEL high performance ATmega128 microcontroller

2.3.1 Main processing unit and peripheral units

The most important thing is to define requirements that a microcontroller should have before starting to choose what types of microcontroller will be used as a brain of the robot. First of all, since the robot created in this project has sensors and actuators like ultrasonic range finder and DC motor, it should have enough amount of Digital I/O port for interfacing these components to the microcontroller. Internal timers are an important resource for microcontroller and robot applications also. Since the DC motors used on the robot are controlled by the pulse width modulation, the controller should be chosen has to have adequate quantity of timers and interrupt capabilities. Furthermore, the robot has encoders mounted on its each wheels to keep track of the location. Therefore, at least two counters should be built into the microcontroller to determine exactly how many numbers of revolutions the wheels has turned. Moreover, the fact that I²C interface is used for getting bearing from the compass module, ultrasonic and infrared sensors require the controller should support I²C serial communication protocol. Furthermore the data having collected from the sensors of the robot will be sent to the Desktop Computer. This requires Universal Asynchronous Receiver and Transmitter (UART) feature to be provided by the controller to communicate with the desktop computer.

By taking into account all above features that the robot's microcontroller should have, ATMEL High performance ATmega128 microcontroller is chosen illustrated in Figure 2.18.

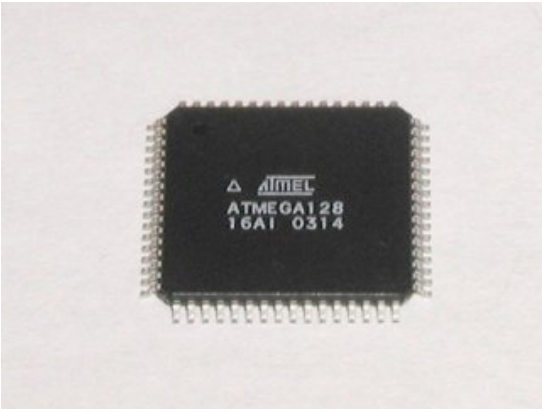

Figure 2.18: ATMEL ATmega128

The ATmega128 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock

cycle, the ATmega128 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega128 provides the following features: 128K bytes of In-System Programmable Flash with Read-While-Write capabilities, 4K bytes EEPROM, 4K bytes SRAM, 53 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), four flexible Timer/Counters with compare modes and PWM, 2 USARTs, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain, programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the Crystal/Resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel’s high-density nonvolatile memory technology. The Onchip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the

application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications [19].

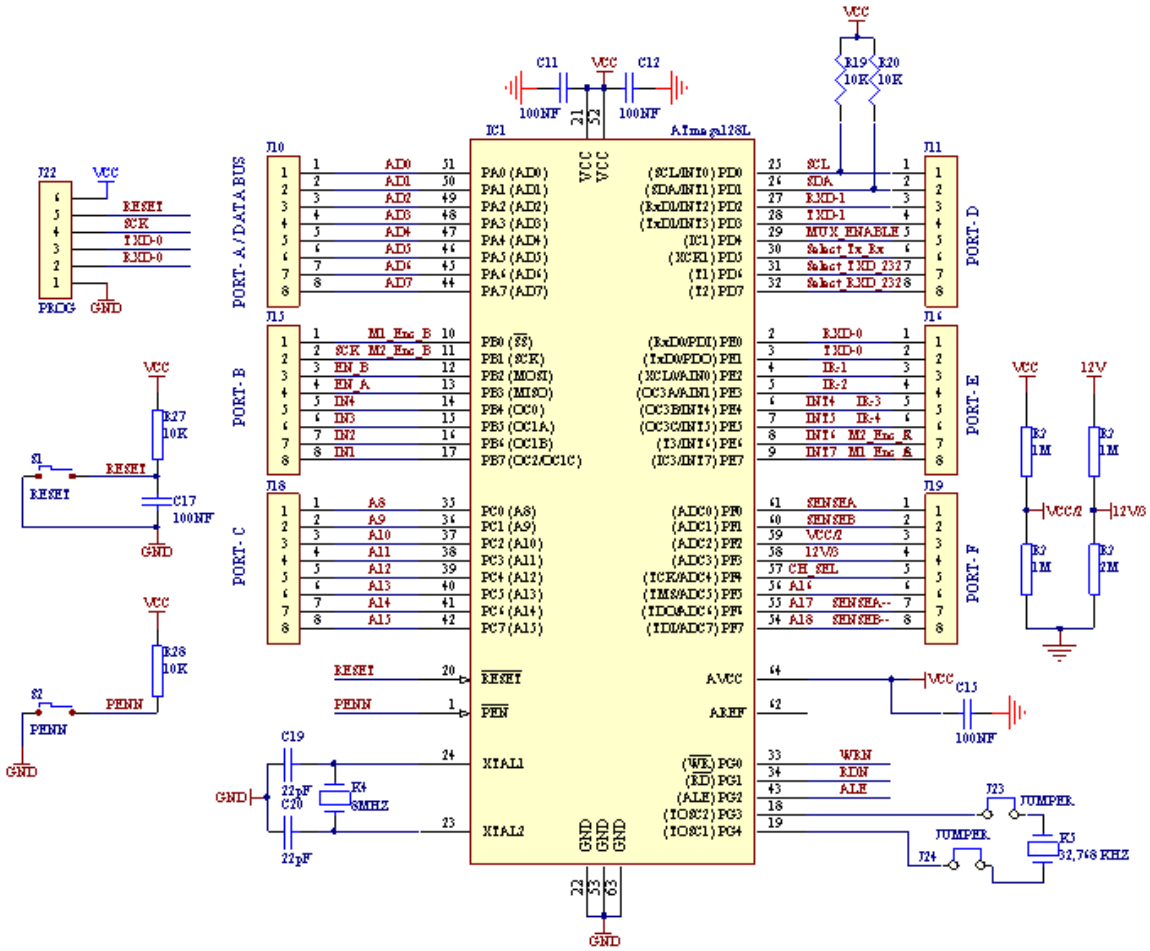

Figure 2.19: Robot main proccessing unit schemeatic diagram

2.3.2 Sensor communication bus

Sensor communication bus is based on I²C Bus which is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cell phone. The name stands for Inter-Integrated Circuit and is pronounced I-squared-C or I-two-C [20].

I²C uses only two bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other, higher or lower, voltages are permitted.

The I²C reference design has a 7-bit address space with 16 reserved addresses, so a maximum of 112 nodes can communicate on the same bus. The most common I²C bus modes are the 100 kbit/s standard mode and the 10 kbit/s low-speed mode, but clock frequencies down to DC are also allowed. Recent revisions of I²C can host more nodes and run faster (400 kbit/s Fast mode, 1 Mbit/s Fast mode plus or Fm+, and 3.4 Mbit/s High Speed mode), and also support other extended features, such as 10-bit addressing.

The maximum number of nodes is obviously limited by the address space, and also by the total bus capacitance of 400 pF.

The reference design, as mentioned above, is a bus with a clock (SCL) and data (SDA) lines with 7-bit addressing. The bus has two roles for nodes: master and slave:

- Master node — node that issues the clock and addresses slaves
- Slave node — node that receives the clock line and address.

The bus is a multi-master bus which means any number of master nodes can be present. Additionally, master and slave roles may be changed between messages after a STOP is sent.

There are four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:

- master transmit — master node is sending data to a slave
- master receive — master node is receiving data from a slave
- slave transmit — slave node is sending data to a master
- slave receive — slave node is receiving data from the master

The master is initially in master transmit mode by sending a start bit followed by the 7-bit address of the slave it wishes to communicate with, which is finally followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave.

If the slave exists on the bus then it will respond with an ACK bit (acknowledge) for that address. The master then continues in either transmit or receive mode according to the read/write bit it sent, and the slave continues in its complementary mode receive or transmit, respectively.

The address and the data bytes are sent most significant bit first. The start bit is indicated by a high to low transition of SDA with SCL high; the stop bit is indicated by a low to high transition of SDA with SCL high.

If the master wishes to write to the slave then it repeatedly sends a byte with the slave sending an ACK bit. In this situation, the master is in master transmit mode and the slave is in slave receive mode.

If the master wishes to read from the slave then it repeatedly receives a byte from the slave, the master sending an ACK bit after every byte but the last one. In this situation, the master is in master receive mode and the slave is in slave transmit mode.

The master then ends transmission with a stop bit, or it may send another START bit if it wishes to retain control of the bus for another transfer (a "combined message").

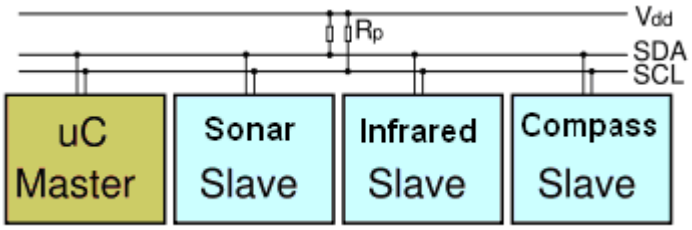

Figure 2.20: A sample schematic with one master (a microcontroller) and three slave nodes (a sonar sensor, an infrared sensor, and an electronic compass module) with pull-up resistors R_p

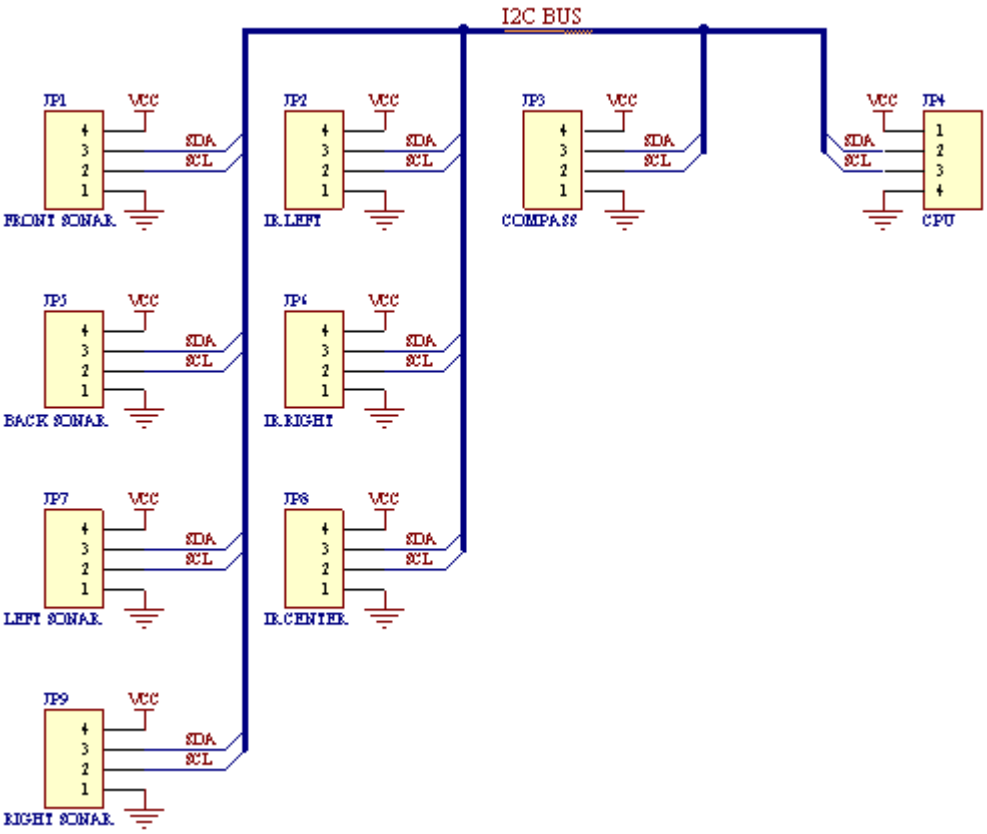

Figure 2.21: Sensor communication (I²C) bus schematic diagram

2.3.3 Power distribution unit

Power for the robot is provided by on-board battery. The robot has common battery for motors and electronics. Battery is a lead acid type rechargeable 12V 2200 mAh.

Figure 2.22: Battery pack for DC motors

Most of the electronics units and components require a particular voltage 5V such as ultrasonic sensors, infrared sensors, electronic compass, encoders of the motors, microcontroller and its peripherals. On the other hand, the RF communication

module requires 3V power input. Therefore, two separate voltage regulator have to be used to convert the 12V battery voltage to 5V and 3V voltages. For this reason the LM7805 and LM317 voltage regulators are used to obtain 5V and 3V voltages respectively shown figure below.

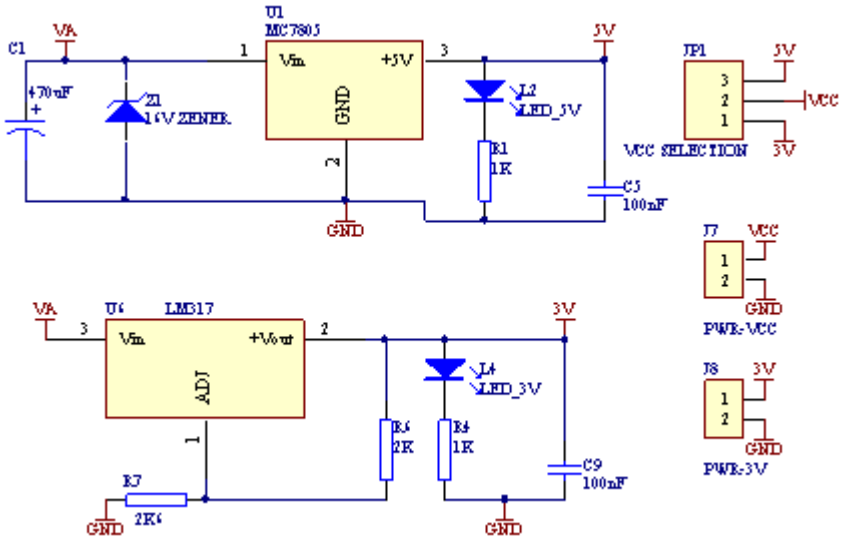

Figure 2.23: Power supply for the electronics

2.3.4 Motor control unit

As mentioned before, mobile robot has two separate motors and their encoders. To achieve differential drive mechanism, motors have to be controlled separately. If the microcontroller chip is connected directly to a motor, the chip will surely be damaged, because the impedance of a DC motor is only a few ohms. Therefore a high voltage, high current dual full-bridge driver integrated circuit (L298N) is used to drive the motors.

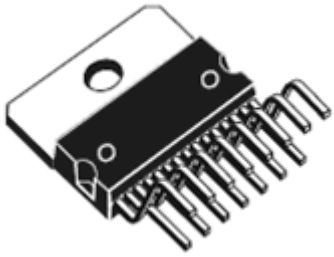

Figure 2.24: L298N Dual Full-Bridge Driver IC

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor [21]. By using this feature, the current flows of the motors are measured and main processor can detect the jam or being forced on the motors. The L298N, its block diagram and robot motor controller and enconder schematic diagram are shown figures below.

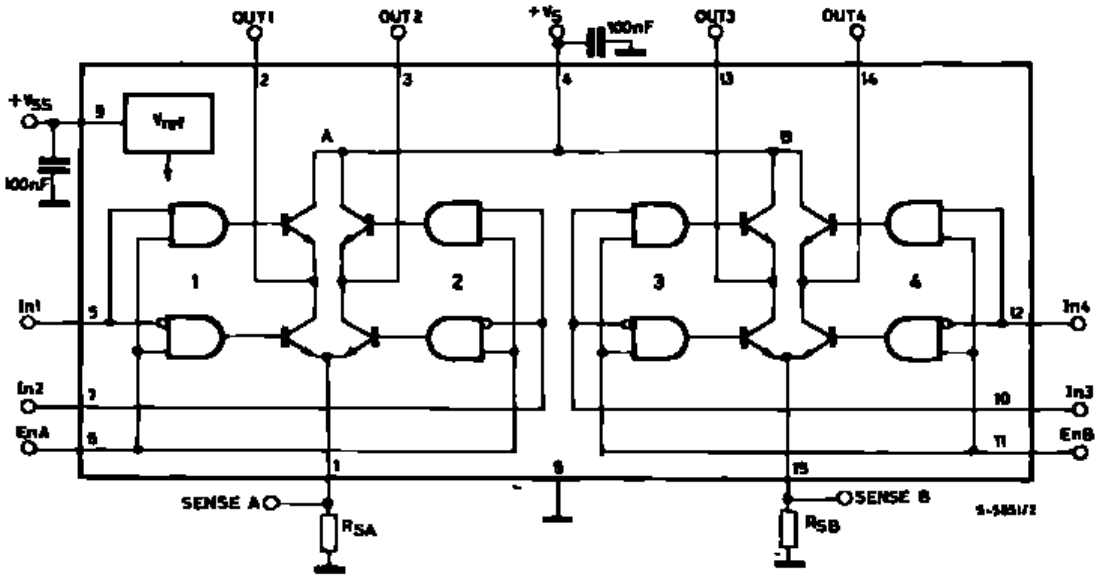

Figure 2.25: L298N Block Diagram [21]

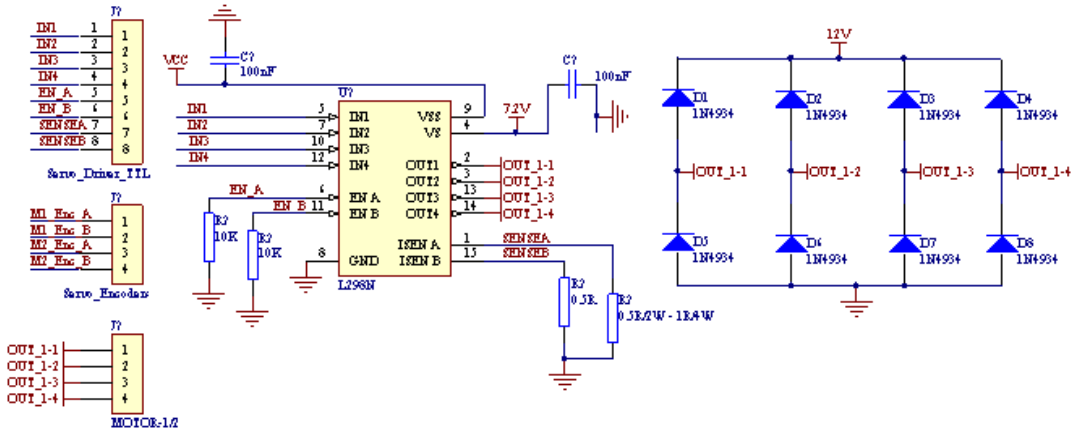

Figure 2.26: Robot Motor Controller and Enconder Schematic Diagram

2.3.5 RF communication unit

The communication between the computer and the Robot is realized by radio frequency link. For this purpose, UDEA UTR-C12U Transceiver is used at the both side of the Robot and the computer. Because of the following two reasons, UDEA UTR-C12U Transceiver can not be connected directly to the main processor ATMEL ATmega128 of the robot which sends RF communication packets from the robot to the PC application software. First, UDEA UTR-C12U Transceiver works with 3V low voltage TTL signals; on the other hand the main processor works with 5V voltage TTL signals. If the units are connected each other without using a voltage level converter, the transceiver will be damaged. Second, the transceiver uses shared one pin for data input and data output. It has an activation input pin for the selecting of the transmitting and receiving mode at a time. On the other hand, the processor has two separate data lines for transmitting and receiving. Thus, the 74HC4053 integrated circuit which is a three terminal analog switch is used for the changing the data lines for transmitting and receiving processes. As a voltage level converter, MAX232 and MAX3232 RS-232 line drivers are used. MAX3232 converts the 3V signals coming from the RF transceiver to the +/- 12V RS-232 signals, then MAX232 takes this RS-232 signals and converts them to the 5V TTL signals. These signals are sent to the processor via the 74HC4053 analog switch. The RF communication circuit schematic diagram is shown figure below.

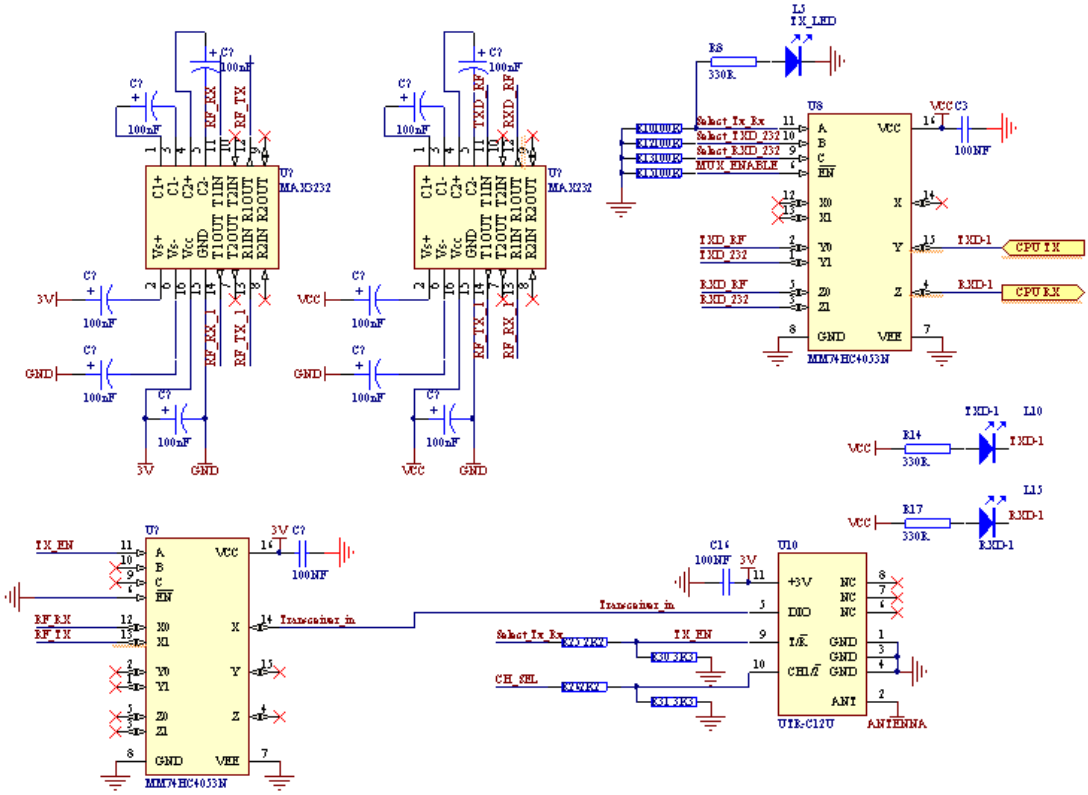

Figure 2.27: RF communication circuit schematic diagram

2.3.6 Serial debugger unit

The serial debugger unit is used for accessing the real-time operating system which runs on the main processor of the robot. By using this unit, you can see the debug messages of the embedded system and running programs on it. Furthermore, memories of the system like RAM and EEPROM can be read and edited. Moreover, processes can be started or terminated by using serial debugger unit.

Serial debugger unit uses the UART-0 peripheral of the main processor. This peripheral uses 5V TTL signals for communication. Therefore, the MAX232 integrated circuit is used to convert the 5V TTL signals to the +/- 12V RS-232 signals, because of the fact that PC serial ports uses RS-232 serial communication protocol.

The MAX232 device consists of two line drivers, two line receivers, and a dual charge-pump circuit with ±15-kV ESD protection pin to pin (serial-port connection pins, including GND). The device meets the requirements of TIA/EIA-232-F and provides the electrical interface between an asynchronous communication controller

and the serial-port connector. The charge pump and four small external capacitors allow operation from a single 5V supply. The RS-232 communication circuit schematic diagram is shown Figure 2.28.

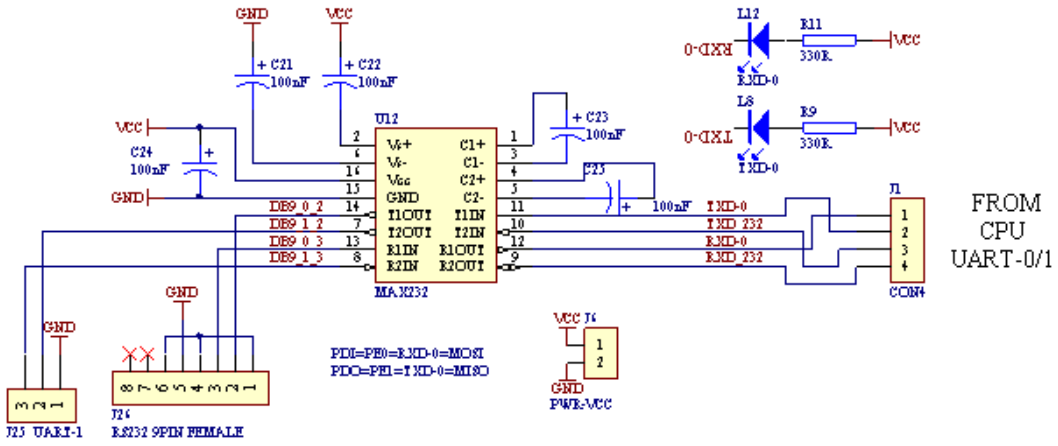

Figure 2.28: RS-232 communication circuit schematic diagram

2.4 Programming the Tracked Robot

In the previous chapter, the electronics hardware of the actuators and sensors of the robot for performing the navigation and object distance measurement have been introduced. In this section, the programming used in robot operating system, device drivers of the sensors and the actuators, and communication systems for the robot will be explained in detail.

2.4.1 Interrupts and timer-activated tasks

A different way of designing a concurrent application is to use interrupts, which can be triggered by external devices or by a built-in timer. Both are very important techniques; external interrupts can be used for reacting to external sensors, such as counting ticks from a shaft encoder, while timer interrupts can be used for implementing periodically repeating tasks with fixed time frame, such as motor control routines.

The event of an external interrupt signal will stop the currently executing task and instead execute a called “interrupt service routine” (ISR). As a general rule, ISRs should have a short duration and are required to clean up any stack changes they performed, in order not to interfere with the foreground task. Initialization of ISRs

often requires assembly commands, since interrupt lines are directly linked to the CPU and are therefore machine-dependent (Figure 2.29).

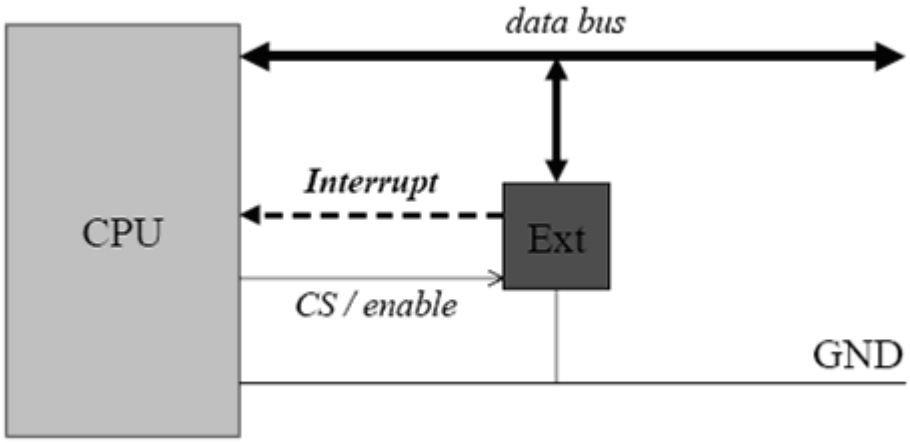

Figure 2.29: Interrupt generation from external device

Somewhat more general are interrupts activated by software instead of external hardware. Most important among software interrupts are timer interrupts, which occur at regular time intervals.

In the robot operating system, a general purpose 100Hz timing interrupt has been implemented. User programs can attach or detach up to 8 timer interrupt ISRs at a rate starting from 100Hz (0.01s). This is achieved with the following operations:

```
void timerAttach(u08 interruptNum, void (*userFunc)(void) );
void timerDetach(u08 interruptNum);
```

The timing scale parameter (range 1..100) is used to divide the 100Hz timer and thereby specifies the number of timer calls per second (1 for 100Hz, 100 for 1Hz). Parameter userFunc is simply the name of a function without parameters or return value (void).

An application program can implement a background task, for example keeping track of the encoder's movements, which will be executed several times per second. Although activation of an ISR is handled in the same way as preemptive

multitasking, an ISR itself will not be preempted, but will keep processor control until it terminates. This is one of the reasons why an ISR should be rather short in time. It is also obvious that the execution time of an ISR (or the sum of all ISR execution times in the case of multiple ISRs) must not exceed the time interval between two timer interrupts, or regular activations will not be possible.

The example in Figure 2.30 shows the timer routine and the corresponding main program. The main program initializes the timer interrupt to once every second. While the foreground task prints consecutive numbers to the screen, the background task generates an acoustic signal once every second.

```
void interrupt_1sec (void)      /* background task */
{ rprintfProgStrM("\r\nBearing: ");
rprintfFloat(4,compass_driver());
}
int main(void)
{ uartInit();
rprintfInit(uart0SendByte);
i2cInit();
timerInit();
timerAttach(TIMER0OUTCOMPARE_INT, interrupt_1sec);
/* foreground task: loop until key press */
while(uart0GetByte())
{ rprintfProgStrM("\r\nCompass Test Terminated by user\r\n");
timerDetach (TIMER0OUTCOMPARE_INT);
return 0;
}
```

Figure 2.30: Timer-activated code example

2.4.2 Command-Line interface

This Command-Line interface library is meant to provide a reusable terminal-like user interface much like a DOS command line or UNIX terminal. A terminal with VT100 support is assumed on the user's end. Common line-editing is supported, including backspace, arrow keys, and even a small command-history buffer.

The cmdline library does the following things:

- Prints command prompts

- Gathers a command string from the user with editing features

- Parses the command string when the user presses [ENTER]

- Compares the entered command to the command database

    -- Executes the corresponding function if a match is found

    -- Reports an error if no match is found

-Provides functions to retrieve the command arguments:

    --as strings

    --as decimal integers

    --as hex integers

Supported editing features include:

- Backspace support

- Mid-line editing, inserting and deleting (left/right-arrows)

- Command History (up-arrow)

To use the cmdline system, programmer needs to associate command strings (commands the user will be typing) with the function that you wish to have called when the user enters that command. This is done by using the cmdlineAddCommand() function.

To setup the cmdline system, you must do these things:

- Initialize it: cmdlineInit()

- Add one or more commands to the database: cmdlineAddCommand()

- Set an output function for your terminal: cmdlineSetOutputFunc()

To operate the cmdline system, you must do these things repeatedly:

- Pass user input from the terminal to: cmdlineSetOutputFunc()

- Call cmdlineMainLoop() from your program's main loop

The cmdline library does not assume an input or output device, but can be configured to use any user function for output using cmdlineSetOutputFunc() and accepts input by calling cmdlineInputFunc(). This means the cmdline library can operate over any interface including UART (serial port), I²C, ethernet, etc.

2.4.3 Programming the navigation of the robot

The robot can be programmed to perform a variety of maneuvers: forward, backward, rotate left, rotate right, and pivoting turns. Figure 2.49 shows the Robot's front, back, left, and right sides.

Programming the robot to travel a distance is accomplished by getting feedback from the incremental encoder and converting this data to the distance it moved and control the movement depending on the target distance.

The robot also can be started or stopped with ramping. Ramping is a way to gradually increase or decrease the speed of the motors instead of abruptly starting or stopping. This technique can increase the life expectancy of robot's batteries.

The motors of the robot are controlled by H-bridge driver and PWM control signals line. PWM signal controls speed of the motor and H-bridge driver controls turning direction of the motor. Controlling a motor's speed and direction involves a program that makes the main processor generating PWM signals continuously. To change the speed of the motor, the PWM signals' frequency is changed.

Moving forward: The robot's left motor has to turn counterclockwise and its right motor has to turn clockwise to make the robot go forward.

Moving backward: The robot's left motor has to turn clockwise and its right motor has to turn counterclockwise to make the robot go backward.

Turning left: The robot's both motor have to rotate clockwise to make the robot turn left.

Turning right: The robot's both motor have to rotate counterclockwise to make the robot turn left.

2.4.4 Measuring how far the robot moves

As mentioned before, each motor on the robot has an encoder that maintains a counter indicating the angular position of the wheel. As the wheel rotates forward, the counter on the main processor counts up. The distance the wheel travels across the floor with each count can be calculated by dividing the circumference of the wheel by the number of counts the encoder increments with one full revolution of the

wheel. At the end, the total distance traveled by the Robot is calculated by multiplying the total number of counts and the distance per count when the robot stops.

The main processor operating system communicates with the quadrature encoders using the quadrature encoder device driver. It allows easy interfacing of standard quadrature encoders used for sensing shaft rotational position and speed. The driver uses external interrupts to sense and keep track of the encoder's movements. The driver is extendable with the maximum number of encoders equal to the total number of external interrupts available on the target processor.

Quadrature encoders have two digital outputs usually called PhaseA and PhaseB. When the encoder rotates, PhaseA and PhaseB produce square wave pulses where each pulse represents a fraction of a turn of the encoder shaft. Encoders are rated for a certain number of pulses (or counts) per complete revolution of the shaft. By counting the number of pulses output on one of the phases starting from an initial time, you can calculate the total rotational distance the encoder has traveled.

However, the current position is needed not just total distance traveled. For this it is necessary to know not only how far the encoder has traveled, but also which direction it was going at each step of the way. To do this both outputs (or phases) of the quadrature encoder are needed to use.

The pulses from PhaseA and PhaseB on quadrature encoders are always aligned 90 degrees out-of-phase. This special phase relationship lets us extract both the distance and direction the encoder has rotated from the outputs.

To do this, Phase A is considered to be the distance counter. On each rising edge of PhaseA, processor will count 1 "tic" of distance, but it is needed to know the direction. As seen the waveform plot Figure 2.31, when the motor turns forward in time (left->right), PhaseB is always low (logic 0) at the rising edge of PhaseA. When the motor turns backwards in time (right->left), PhaseB is always high (logic 1) at the rising edge of PhaseA. Note that traveling forward or backwards in time is the same thing as rotating forwards or backwards. Thus, if PhaseA is the counter, PhaseB indicates direction.

Here is an example waveform from a quadrature encoder:

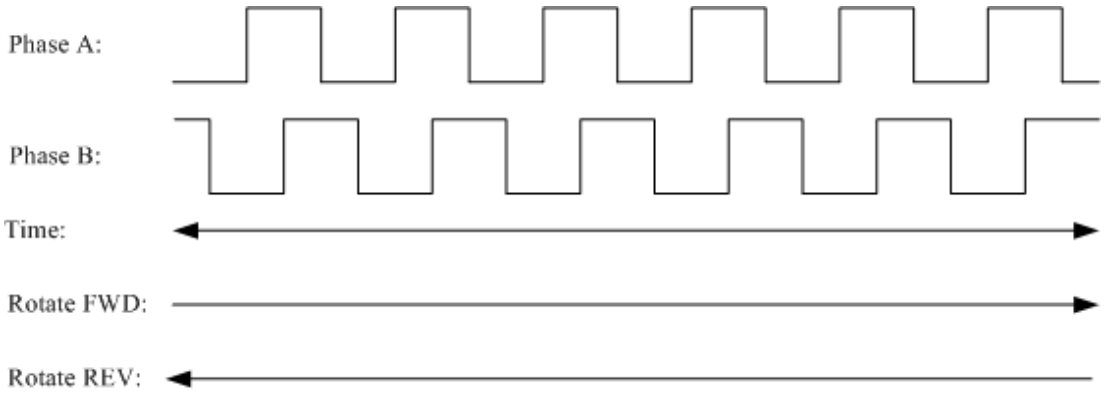

Figure 2.31: A sample waveform from a quadrature encoder

To keep track of the encoder position in software, PhaseA is connected to an external processor interrupt line and PhaseB to any I/O pin. The external interrupt is set up to trigger whenever PhaseA produces a rising edge. When a rising edge is detected, processor interrupt handler function is executed. Inside the handler function, the PhaseB line is quickly checked to see if it is high or low. If it is high, the encoder's position counter is incremented, otherwise it is decremented. So, the encoder position counter can be read at any time to find out the current position.

2.4.5 Object distance measurement by using ultrasonic sensor

Communication with the Devantech SRF08 ultrasonic range finder is via the I²C bus. I²C bus communication protocol is mentioned before at the sensor communication bus section. In this communication, main processor acts as a master role and sensors act as slave units. The SRF08 behaves in the same way as the ubiquitous 24xx series EEPROM's, except that the I²C address is different. The default shipped address of the SRF08 is 0xE0. It can be changed by the user to any of 16 addresses E0, E2, E4, E6, E8, EA, EC, EE, F0, F2, F4, F6, F8, FA, FC or FE, therefore up to 16 sonar's can be used. In addition to the above addresses, all sonar's on the I²C bus will respond to address 0 - the General Broadcast address. This means that writing a ranging command to I²C address 0 (0x00) will start all sonar's ranging at the same time. The results must be read individually from each sonar's real address [22].

Registers

The SRF08 appears as a set of 36 registers.

Table 2.2: The SRF08 registers

Location	Read	Write
0	Software Revision	Command Register
1	Light Sensor	Max Gain Register (default 31)
2	1st Echo High Byte	Range Register (default 255)
3	1st Echo Low Byte	N/A
~~	~~~~	~~~~
34	17th Echo High Byte	N/A
35	17th Echo Low Byte	N/A

Only locations 0, 1 and 2 can be written to. Location 0 is the command register and is used to start a ranging session. It cannot be read. Reading from location 0 returns the SRF08 software revision. By default, the ranging lasts for 65ms, but can be changed by writing to the range register at location 2.

Location 1 is the onboard light sensor. This data is updated every time a new ranging command has completed and can be read when range data is read. The next two locations, 2 and 3, are the 16bit unsigned result from the latest ranging - high byte first. The meaning of this value depends on the command used, and is either the range in inches, or the range in cm or the flight time in µs. A value of zero indicates that no objects were detected. There are up to a further 16 results indicating echo's from more distant objects.

Commands

There are three commands to initiate a ranging (80 to 82), to return the result in inches, centimeters or microseconds. And there are three commands to change the device slave address.

Table 2.3: The SRF08 commands

Command		Action
Decimal	Hex	
80	0x50	Ranging Mode - Result in inches
81	0x51	Ranging Mode - Result in centimeters
82	0x52	Ranging Mode - Result in micro-seconds
160	0xA0	1st in sequence to change I²C address
165	0xA5	3rd in sequence to change I²C address
170	0xAA	2nd in sequence to change I²C address

Ranging Mode

To initiate a ranging, it is needed to write one of the above ranging commands to the command register and wait the required amount of time for completion and read as many results as you wish. The echo buffer is cleared at the start of each ranging. The first echo range is placed in locations 2, 3. The second is placed in 4, 5, etc. If a location (high and low bytes) is 0, then there will be no further reading in the rest of the registers. The default and recommended time for completion of ranging is 65 ms, however it can be shorten by writing to the range register before issuing a ranging command. Light sensor data at location 1 will also have been updated after a ranging command.

Checking for Completion of Ranging

It is not needed to use a timer on the main processor to wait for ranging to finish. The SRF08 do not respond to any I²C activity while ranging. Therefore, if the SRF08 is tried to read then it will respond 255 (0xFF) while ranging. This is because the I²C data line (SDA) is pulled high if nothing is driving it. As soon as the ranging is complete the SRF08 will again respond to the I²C bus, so the register is kept reading until its not 255 (0xFF). Then, the sonar data can be read. By using this method, the main processor can perform other tasks while the SRF08 is ranging.

Changing the I²C Bus Address

In this study, six pieces of SRF08 are used on the robot. They must have unique and correct static addresses for the communication. To change the I²C address of a SRF08, only one sensor has to be mounted on the bus. To initiate an address change, the 3 sequence commands have to be written in the correct order followed by the

address. For example; to change the address of a sonar currently at 0xE0 (the default shipped address) to 0xF2, it is needed to write the following to address 0xE0; (0xA0, 0xAA, 0xA5, 0xF2). These commands have to be sent in the correct sequence to change the I²C address. Additionally, no other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 0, which means 4 separate write transactions on the I²C bus. When done, the sonar with its new address is ready to use.

2.4.6 Object distance measurement by using infrared sensor

Communication with the Sharp GP2D12 infrared range finder is via the I²C bus. I²C bus communication protocol is mentioned before at the sensor communication bus section. In this communication, main processor acts as a master role and sensors act as slave units. The GP2D12 offers true ranging information in the form of an analog voltage output then, the output is converted to digital data and I²C interface is provided by Solarbotics I2C-It module. So, the GP2D12 behaves in the same way as the ubiquitous 24xx series EEPROM's, except that the I²C address is different. The default shipped address of the 0x20. It can be changed with simple solder-jumpers by the user to any of 8 addresses given at the table below; therefore up to 8 infrared sensors can be used [23, 24].

Table 2.4: The infrared sensor jumper settings for the device addresses

Device Address	JP1	JP2	JP3
0x20	0	0	0
0x22	0	0	1
0x24	0	1	0
0x26	0	1	1
0x28	1	0	0
0x2A	1	0	1
0x2C	1	1	0
0x2E	1	1	1

Commands

There are three commands to initiate a ranging (1 to 3), to return the result in inches, centimeters or microseconds.

Table 2.5: The infrared sensor commands

Command		Action
Decimal	Hex	
1	0x01	Ranging Mode - Result in inches
2	0x02	Ranging Mode - Result in centimeters
3	0x03	Ranging Mode - Result in micro-seconds

Ranging Mode

To initiate a ranging, it is needed to write one of the above ranging commands to the command register and wait the required amount of time for completion and read one result at a time.

2.4.7 Measuring the orientation by using CMPS03

Communication with the Devantech CMPS03 compass module is via the I²C bus. I²C bus communication protocol is mentioned before at the sensor communication bus section. In this communication, main processor acts as a master role and compass module acts as a slave unit. The CMPS03 behaves in the same way as the ubiquitous 24xx series EEPROM's, except that the I²C address is different. The default shipped address of the CMPS03 is 0xC0. It can be changed by the user to any of 8 addresses 0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC or 0xCE; therefore up to 8 modules can be used.

Registers

The CMPS03 appears as a set of 16 registers.

Table 2.6: The CMPS03 registers

Register	Function
0	Software Revision Number
1	Compass Bearing as a byte, i.e. 0-255 for a full circle
2,3	Compass Bearing as a word, i.e. 0-3599 for a full circle, representing 0-359.9 degrees.
4,5	Internal Test - Sensor1 processed difference signal – 16 bit signed word
6,7	Internal Test - Sensor2 processed difference signal – 16 bit signed word
8,9	Internal Test - Sensor1 raw data - 16 bit signed word
10,11	Internal Test - Sensor2 raw data - 16 bit signed word
12	Unlock code1
13	Unlock code2
14	Unlock code3
15	Command Register

Register 0 is the Software revision number. Register 1 is the bearing converted to a 0-255 value. This may be easier for some applications than 0-360 which requires two bytes. In this study, registers 2 and 3 (high byte first) are used for better resolution which are a 16 bit unsigned integer in the range 0-3599. This represents 0-359.9°. Registers 4 to 11 are internal test registers. Registers 8, 9 and 10, 11 contain the raw sensor data. These are the signals coming directly from the sensors, and are the starting point for all the internal calculations which produce the compass bearing. Registers 12, 13 and 14 are for writing the unlock codes for I²C address change or restoring factory calibration. Register 15 is the command register.

Commands

Register 15 is the command Register. There are very few commands - 0xC0-CE for I²C address change and 0xF2 for restoring factory calibration - these require unlock codes, see below. Also 255 (0xFF) is the calibrate command. There are no unlock codes required for this.

Operating

To initiate the module for reading, first it is needed to send a start bit, the module address (0xC0) with the read/write bit low, then the register number to read. This is followed by a repeated start and the module address again with the read/write bit high (0XC1). Then read one or two bytes for 8bit or 16bit registers respectively. 16bit registers are read high byte first. The compass has a 16 byte array of registers, some of which double up as 16 bit registers as shown Table 2.6.

Changing the I²C Bus Address

It is possible to change the I²C address to any of 8 addresses 0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC or 0xCE. It is done by writing unlock codes to registers 12,13 and 14 and the new address to register 15. Note that the unlock codes are different to the ones which restore factory calibration.

Table 2.7: Unlock codes for the CMPS03 I²C bus slave address change

Register	Unlock Code
12	0xA0
13	0xAA
14	0xA5

Restoring Factory Calibration

It is possible to restore the factory calibration settings. It is done by writing unlock codes to registers 12, 13 and 14 and the restore command (0xF2) to register 15. Note that the unlock codes are different to the ones which used for changing the I²C address.

Table 2.8: Unlock codes for the CMPS03 restoring factory calibration

Register	Unlock Code
12	0x55
13	0x5A
14	0xA5

Calibration

The module has already been calibrated in factory workshop for the inclination, which is 67 degrees. Calibration only needs to be done once - the calibration data is

stored in EEPROM on the module. It is not needed to re-calibrate every time the module is powered up.

Before calibrating the compass, it is needed to know exactly which direction is North, East, South and West. When calibrating, the compass is horizontal at all times with components upwards. All magnetic and ferrous materials must be kept away from the compass during calibration.

To calibrate using the I²C bus, it is needed to write 255 (0xff) to register 15, once for each of the four major compass points North, East, South and West. The 255 is cleared internally automatically after each point is calibrated. The compass points can be set in any order, but all four points must be calibrated.

2.4.8 Programming the wireless communication

There are a number of tasks where a wireless communication is helpful for a robot and a host computer. These are;

1. To remote-control the robot

For example, giving low-level driving commands or specifying high level goals to be achieved.

2. To monitor robot sensor data

For example, displaying sensor data from the robot or recording a robot's distance sensor data over time.

3. To create a monitoring console for the robot

For example, monitoring the robot’s position, orientation, and status in a common environment.

A special error protocol is required because of the high error rate of mobile wireless data exchange. Messages are transmitted and received between the PC and the robot by using STX/ETX packet protocol. Both of the units that communicates each other have STX/ETX communication protocol library for processing the incoming and outgoing packets. This library provides functions needed to transmit and receive STX/ETX packets over any interface which can send and receive bytes. STX/ETX is a simple packet protocol for serial data streams and offers packetization, type tagging, and checksum protection for user data. Common uses of STX/ETX might

include wireless communications were it can improve data reliability over lossy channels. STX/ETX may also be used effectively anywhere multiple access to the communication medium is required. The packets can be made to contain destination addresses or routing information as well as data.

STX/ETX is a simple packet protocol that can be wrapped around user data for one or more of the following reasons:

First, packetization is needed. Using packets can be helpful if the data naturally forms little "bunches" or if different types of data must be sent over the same channel. If the data forms "bunches", user data can be sent inside STX/ETX packets with a predetermined structure, like an array of A/D conversion results or sensor distance measurement results. The TYPE field in the STX/ETX packet is used to tell the receiver what kind of data is being sent.

Second, error checking is needed. STX/ETX packets will add a checksum to the data. This allows the receiver to verify that data was received correctly and is error-free. Packets which are corrupted in transmission and fail the checksum test are automatically discarded. Error checking is especially useful when the data transmission channel is unreliable or noisy.

STX/ETX packets have the following structure:

[STX][status][type][length][user data...][checksum][ETX]

All fields are 1 byte except for user data which may be 0-255 bytes. Uppercase fields are constant (STX=0x02, ETX=0x03), lowercase fields vary. The length field is the number of bytes in the user data area. The checksum is the 8-bit sum of all bytes between but not including STX/ETX.

2.5 Monitoring and Controlling the Tracked Robot from the Computer

As has been mentioned before, the wireless robot communication feature serves two purposes:

- Message exchange between the robot for application specific purposes under user program control and
- Monitoring and remote controlling the robot from a host computer.

Both communication protocols are implemented as different message types using the same message structure, transparent to both the user program and the higher levels of the robot operating system itself.

In the following, the design and implementation of the robot console that allows the monitoring of robot behavior, the robot's sensor readings, internal states, as well as current position and orientation in an environment is discussed. Remote control of the robot is also possible by transmitting input button data, which is interpreted by the robot's application program.

The host system is a computer running MS Windows. The application program running on the host system accesses a serial port linked to a wireless module, identical to the one on the robot.

2.5.1 The application program

In this project, Visual Basic 8.0 is used as the development software to be able to control the robot from the remote computer and perform mapping and localization. The development environment is Visual Studio 2005. Figure 2.32 shows the main screen of the application program.

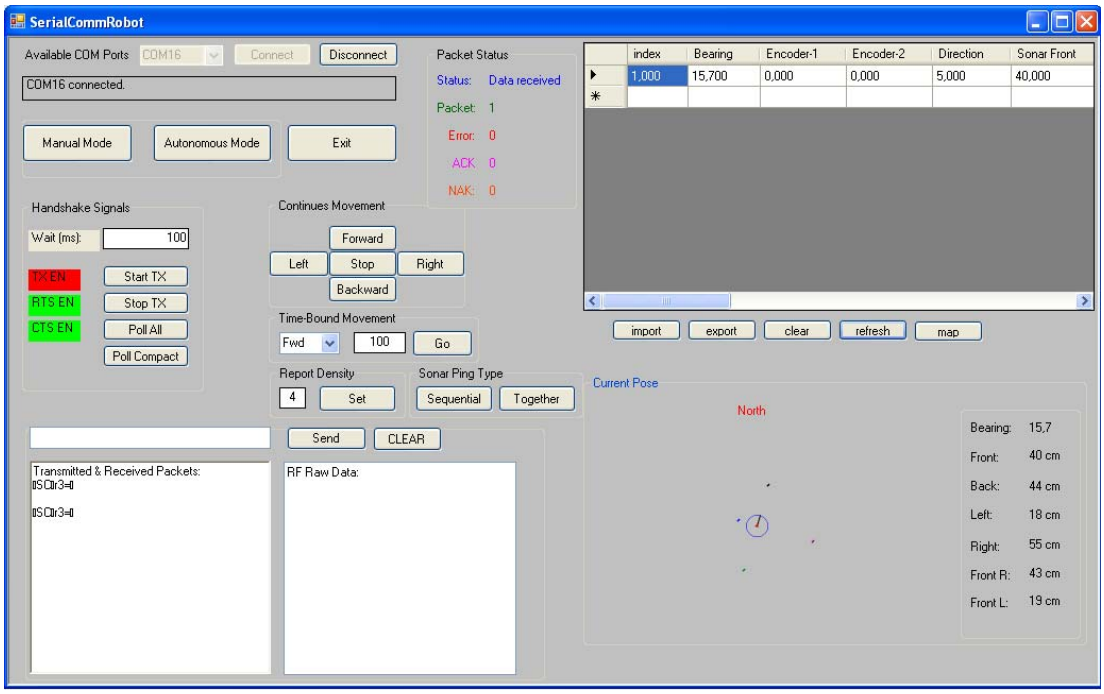

Figure 2.32: Main control page of the application software

As explained before, the communication between the robot and computer is realized through the wireless radio frequency link by using RS232 serial communication protocol. The application program always listens the specified serial port and as soon as new data arrives it calls a dedicated procedure for acquiring the new data having sent from the robot.

Operation Modes of the Robot

The robot has two modes of operation. Once the com port settings are done and communication is established, the robot is ready to be controlled from the remote computer.

In "Manual" control mode, the robot waits for user commands. In this mode, the control is in the user and the user can control the navigation of the robot such as forward, backward, turn left and right by using dedicated buttons. Moreover the user is able to measure the range and orientation of the robot by using related buttons on the screen as shown in Figure 2.33.

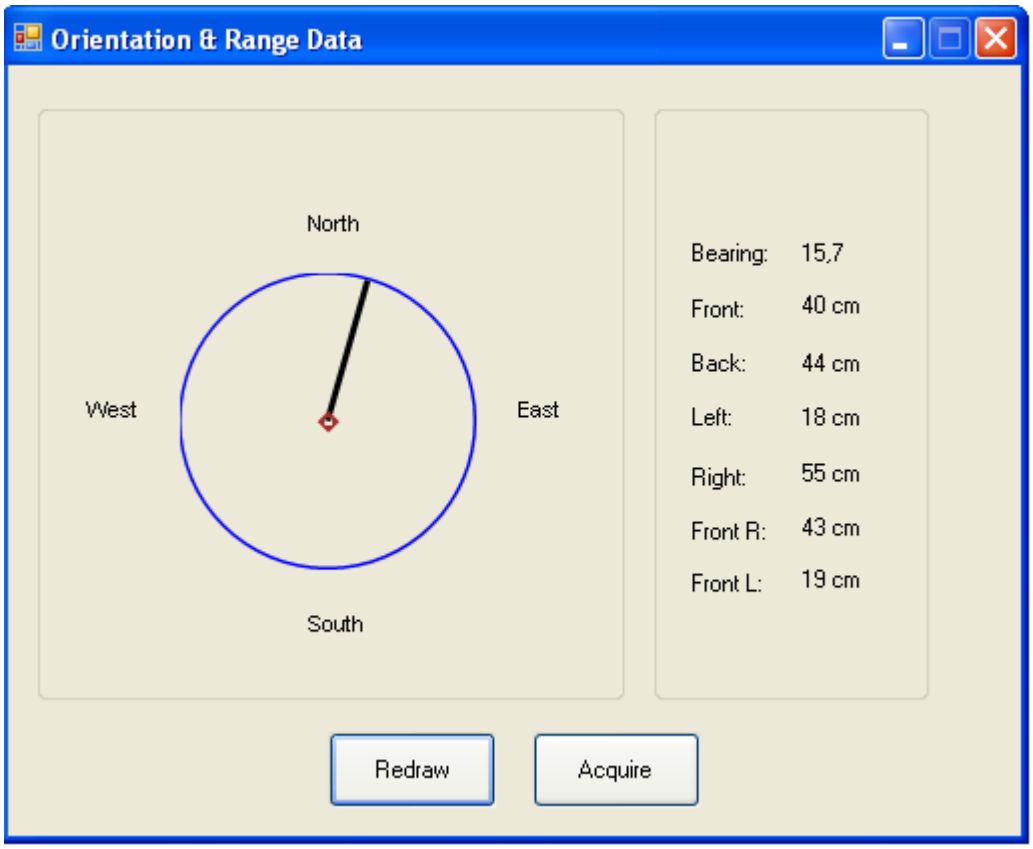

Figure 2.33: Measuring the orientation and range manually

In the "Mapping" mode, the robot performs mapping and localization. Mapping and localization screen is shown in Figure 2.34.

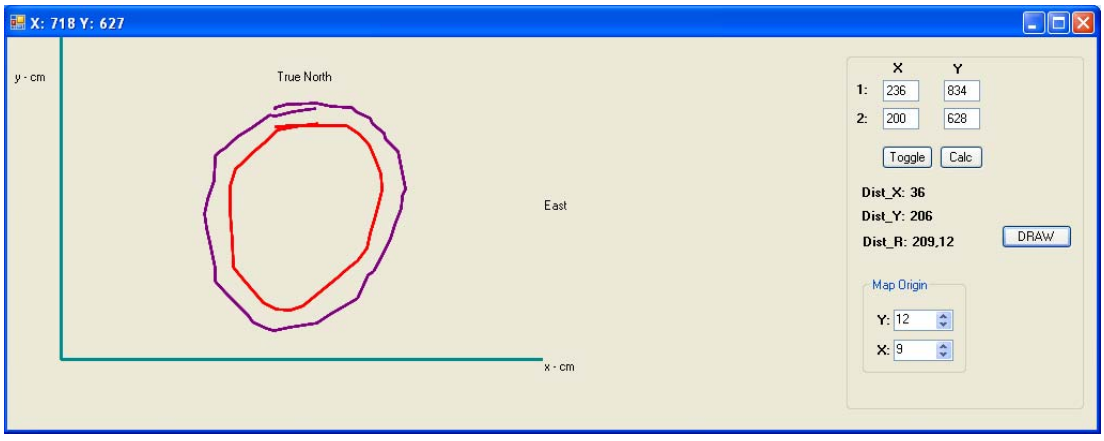

Figure 2.34: Mapping and localization screen

2.5.2 PC Serial telemetry module

As mentioned before, the communication between the computer and the Robot is realized by radio frequency link. For this purpose, UDEA UTR-C12U Transceiver is used at the both side of the Robot and the computer.

The transceiver uses shared one pin for data input and data output. It has an activation input pin for the selecting of the transmitting and receiving mode at a time. On the other hand, the RS-232 communication protocol works full duplex which has two separate data lines for transmitting and receiving. Thus, the 74HC4053 integrated circuit which is a three terminal analog switch is used for the changing the data lines for transmitting and receiving processes. Two RS-232 signals (RTS/CTS) are used for controlling of the switching the data directions. To enable the transmitter of the transceiver, RTS signal is pulled low and CTS signal is checked for the completion of the switching. To enable the receiver of the transceiver, RTS signal is pulled high and CTS signal is checked for the completion of the switching.

As a voltage level converter, MAX3232 RS-232 line driver is used. MAX3232 converts the 3V signals coming from the RF transceiver to the +/- 12V RS-232 signals. These signals are sent to the PC. The PC Serial telemetry module schematic diagram is shown Figure 2.35.

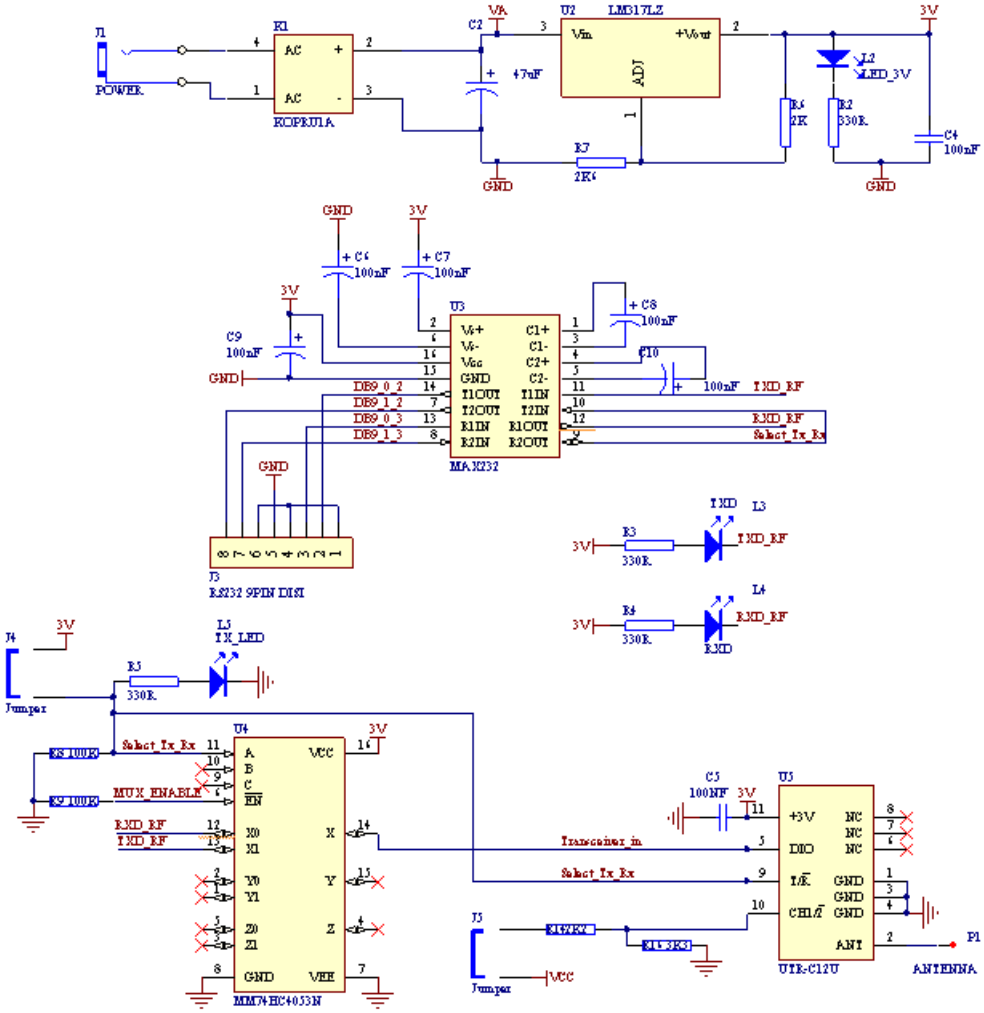

Figure 2.35: PC Serial telemetry module circuit schematic diagram

3. LOCALIZATION, NAVIGATION AND MAPPING

Localization and navigation are the two most important tasks for mobile robots. We want to know where we are, and we need to be able to make a plan for how to reach a goal destination. These two problems are not isolated from each other, but rather closely linked. If a robot does not know its exact position at the start of a planned trajectory, it will encounter problems in reaching the destination.

3.1 Localization

One of the central problems for driving robots is localization. For many application scenarios, we need to know a robot's position and orientation at all times. For example, a cleaning robot needs to make sure it covers the whole floor area without repeating lanes or getting lost, or an office delivery robot needs to be able to navigate a building floor and needs to know its position and orientation relative to its starting point. This is a non-trivial problem in the absence of global sensors.

The localization problem can be solved by using a global positioning system. In an outdoor setting this could be the satellite-based GPS. In an indoor setting, a global sensor network with infrared, sonar, laser, or radio beacons could be employed. These will give us directly the desired robot coordinates. But, this is not a realistic method for exploring an unknown environment that's why a global sensor network is not used for localization and mapping in this study.

3.1.1 Localization methods

3.1.1.1 Odometry

Odometry is the study of position estimation during wheeled vehicle navigation. The term is also sometimes used to describe the distance traveled by a wheeled vehicle. Odometry is used in this study to estimate (not determine) their position relative to a starting location. Odometry is the use of data from the rotation of wheels or tracks to estimate change in position over time. This method is often very sensitive to error.

Rapid and accurate data collection, equipment calibration, and processing are required in most cases for odometry to be used effectively.

The odometry implies vehicle displacement along the path of travel is directly derived from some onboard “odometer.” A common means of odometry instrumentation involves encoders directly coupled to the motor armatures or wheel axles.

The word odometry is composed from the Greek words hodos (meaning "travel", "journey") and metron (meaning "measure") [25].

3.1.1.2 Dead reckoning

Dead reckoning (DR) is the process of estimating one's current position based upon a previously determined position, or fix, and advancing that position based upon known speed, elapsed time, and course. While traditional methods of dead reckoning are no longer considered primary in most applications, modern inertial navigation systems, which also depend upon dead reckoning, are very widely used.

Dead reckoning derived from “deduced reckoning” of sailing days is a simple mathematical procedure for determining the present location of a ship by advancing some previous position through known course and velocity information over a given length of time. The vast majority of land-based mobile robotic systems in use today rely on dead reckoning to form the very backbone of their navigation strategy, and like their nautical counterparts, periodically null out accumulated errors with recurring “fixes” from assorted navigation aids [26].

3.1.1.3 Bayes filter

Bayes filter is an algorithm used in Computer Science for calculating the probabilities of multiple beliefs to allow a robot to infer its position and orientation. Essentially, Bayes Filters allow robots to continuously update their most likely position within a coordinate system, based on the most recently acquired sensor data. This is a recursive algorithm [27].

In a simple example, a robot moving throughout a grid may have several different sensors which provide it with information about its surroundings. The robot may start out with certainty that it is at position 0,0. However, as it moves further and further from its original position, the robot has continuously less certainty about it's position;

Using Bayes filter, a probability can be assigned to the robots belief about it's current position, and that probability can be continuously updated from additional sensor information.

3.1.1.4 Monte Carlo localization

In robotics and sensors, Monte Carlo localization, or MCL, is a Monte Carlo method to determine the position of a robot given a map of its environment based on Markov localization. It is basically an implementation of the Particle filter applied to robot localization, and has become very popular in the Robotics literature. In this method a large number of hypothetical current configurations are initially randomly scattered in configuration space. With each sensor update, the probability that each hypothetical configuration is correct is updated based on a statistical model of the sensors and Bayes' theorem. Similarly, every motion the robot undergoes is applied in a statistical sense to the hypothetical configurations based on a statistical motion model. When the probability of a hypothetical configuration becomes very low, it is replaced with a new random configuration.

3.2 Navigation

Navigation, however, is much more than just driving to a certain specified location. It all depends on the particular task to be solved.

There are a number of well-known navigation algorithms, which will be briefly explained in the following. However, some of them are of a more theoretical nature and do not closely match the real problems encountered in practical navigation scenarios. For example, some of the shortest path algorithms require a set of node positions and full information about their distances. But in many practical applications there are no natural nodes or their location or existence is unknown, as for partially or completely unexplored environments.

3.2.1 Navigation Algorithms

3.2.1.1 Wall-Following Algorithm

The most widely used approach for the exploration part of the problem is to always follow the right wall. For example, if the robot comes to an intersection with several

open sides, it follows the rightmost path. The start square is assumed to be at position [0, 0], the four directions north, south, east and west are encoded as integers [29].

The procedure is very simple and comprises only a few lines of code. It contains a single while-loop that terminates when the goal square is reached x and y coordinates match. In each iteration, it is determined by reading the robot's sensors whether a wall exists on the front, left, or right side. The robot then selects the "rightmost" direction for its further journey. That is, if possible it will always drive right, if not it will try driving straight, and only if the other two directions are blocked, will it try to drive left. If none of the three directions are free, the robot will turn on the spot and go back one square, since it has obviously arrived at a dead-end.

3.2.1.2 Wandering Standpoint Algorithm

The other approach for the exploration part of the problem is trying to reach goal from start in direct line. When encountering an obstacle, measure avoidance angle for turning left and for turning right, turn to smaller angle. Continue with boundary following around the object, until goal direction is clear again [29].

Figure 3.1 shows the subsequent robot positions from Start through 1..6 to Goal. The goal is not directly reachable from the start point. Therefore, the robot switches to boundary-following mode until, at point 1, it can drive again unobstructed toward the goal. At point 2, another obstacle has been reached, so the robot once again switches to boundary-following mode. Finally at point 6, the goal is directly reachable in a straight line without further obstacles. Realistically, the actual robot path will only approximate the waypoints but not exactly drive through them.

The algorithm can lead to an endless loop for extreme obstacle placements. In this case the robot keeps driving, but never reaches the goal.

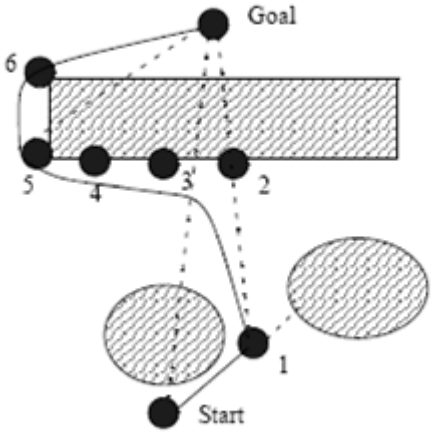

Figure 3.1: Wandering standpoint example

3.2.1.3 DistBug Algorithm

The selected approach in this study for the exploration part of the problem is driving straight towards the goal when possible, otherwise do boundary following around an obstacle. If this brings the robot back to the same previous collision point with the obstacle, then the goal is unreachable [29, 35].

Although this algorithm has nice theoretical properties, it is not very usable in practice, as the positioning accuracy and sensor distance required for the success of the algorithm are usually not achievable. Most variations of the DistBug algorithm suffer from a lack of robustness against noise in sensor readings and robot driving/positioning.

Figure 3.2 shows two standard DistBug examples. In the example on the left hand side, the robot starts driving forward towards the goal, until it hits the U-shaped obstacle. A hit point is recorded and boundary following subroutine is called. After a left turn, the robot follows the boundary around the left leg, at first getting further away from the goal, then getting closer and closer. Eventually, the free space in goal direction will be greater or equal to the remaining distance to the goal. This happens at the leave point. Then the boundary follow subroutine returns to the main program, and the robot will for the second time drive directly towards the goal. This time the goal is reached and the algorithm terminates.

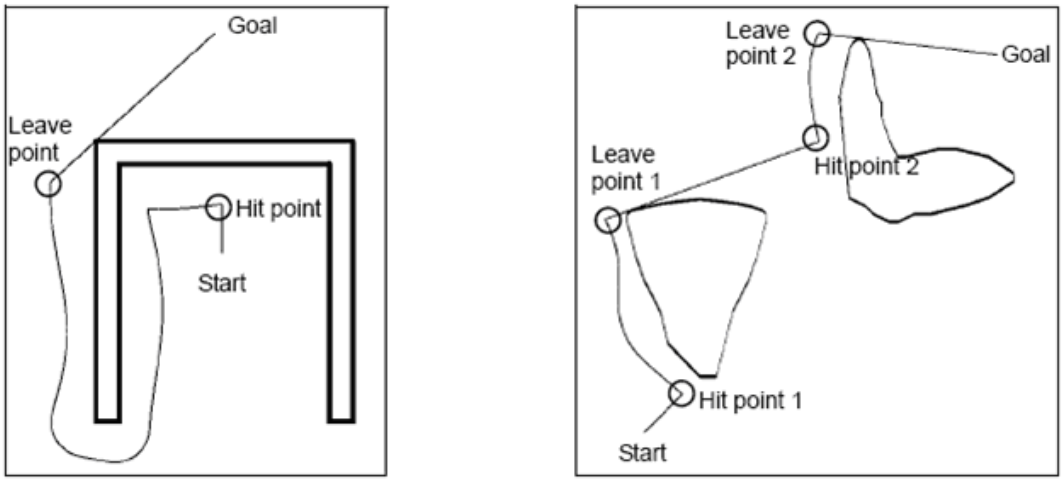

Figure 3.2: Distbug examples

Figure 3.2, right, shows another example. The robot will stop boundary following at the first leave point, because its sensors have detected that it can reach a point closer to the goal than before. After reaching the second hit point, boundary following is called a second time, until at the second leave point the robot can drive directly to the goal.

Figure 3.3 shows two more examples that further demonstrate the DistBug algorithm. In Figure 3.3, left, the goal is inside the E-shaped obstacle and cannot be reached. The robot first drives straight towards the goal, hits the obstacle and records the hit point, then starts boundary following. After completion of a full circle around the obstacle, the robot returns to the hit point, which is its termination condition for an unreachable goal.

Figure 3.3, right, shows a more complex example. After the hit point has been reached, the robot surrounds almost the whole obstacle until it finds the entry to the maze-like structure. It continues boundary following until the goal is directly reachable from the leave point.

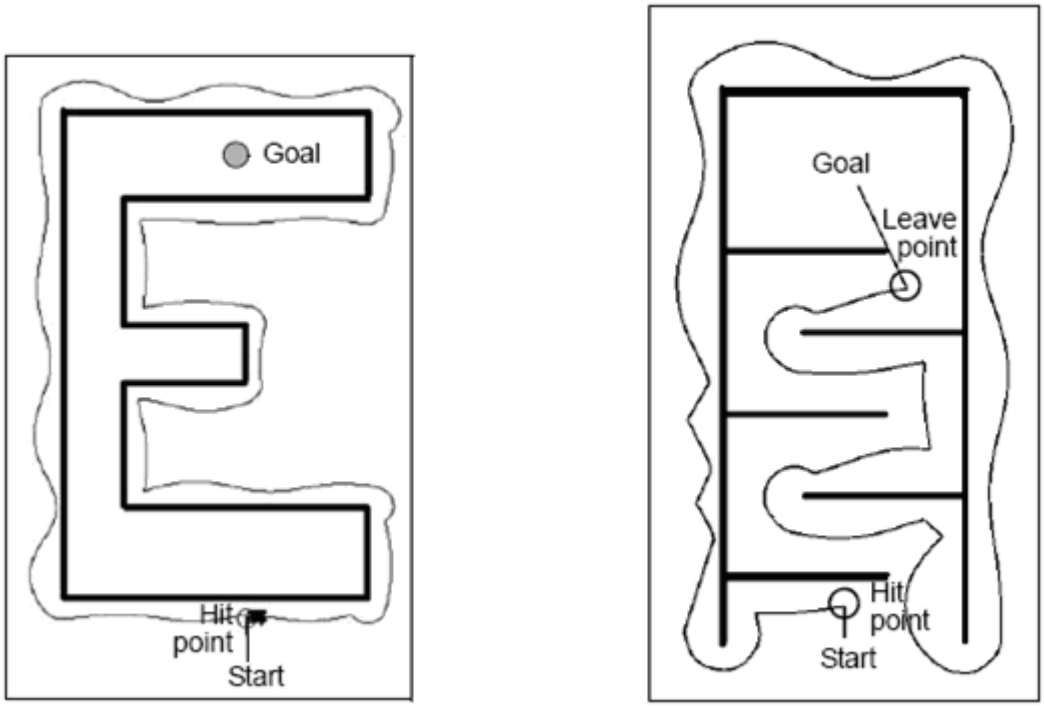

Figure 3.3: Complex Distbug examples

3.3 Map Generation

A problem related to map-building is autonomous exploration which is mentioned before in navigation section. In order to build a map, the robot must explore its environment to map uncharted areas. Typically it is assumed that the robot begins its exploration without having any knowledge of the environment. Then, a certain motion strategy is followed which aims at maximizing the amount of charted area in the least amount of time. Such a motion strategy is called exploration strategy, and it depends strongly on the kind of sensors used.

The problem of robotic mapping is that of acquiring a spatial model of a robot's environment. Maps are commonly used for robot navigation [30, 31]. To acquire a map, robots must possess sensors that enable it to perceive the outside world. Sensors commonly brought to bear for this task include cameras, range finders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, and GPS. Most robot sensors are subject to strict range limitations. For example, light and sound cannot penetrate walls. These range limitations makes it necessary for a robot to navigate through its environment when building a map. The motion

commands (controls) issued during environment exploration carry important information for building maps, since they convey information about the locations at which different sensor measurements were taken [32].

The type of spatial representation system used by a robot should provide a way to incorporate consistently the newly sensed information into the existing world model. It should also provide the necessary information and procedures for estimating the position and pose of the robot in the environment. Information to do path planning, obstacle avoidance, and other navigation tasks must also be easily extractable from the built world model [33].

3.4 Mapping algorithm

For implementation, the tracked robot is used. It uses four ultrasonic sensors which return a distance value to the nearest obstacle. In principle, sensors are positioned and oriented around the robot with 90 degrees each other.

Accuracy and speed are the two major criteria for map generation. Although the quad-tree representation seems to fit both criteria, it was unsuitable for our setup with limited accuracy sensors. Instead, the approach of visibility graphs [34] with configuration space representation is used. Since the typical environments with only few obstacles and lots of free space, the configuration space representation is more efficient than the free space representation.

The basic tasks for map generation are to explore the environment and list all obstacles encountered. In this implementation, the robot starts exploring its environment. If it locates any obstacles, it drives toward the nearest one, performs a boundary-following algorithm around it, and defines a map entry for this obstacle. This process continues until all obstacles in the local environment are explored. The robot then proceeds to an unexplored region of the physical environment and repeats this process. Exploration is finished when the internal map of the physical environment is fully defined [29].

The task planning structure is specified by the mapping algorithm in Figure 3.4.

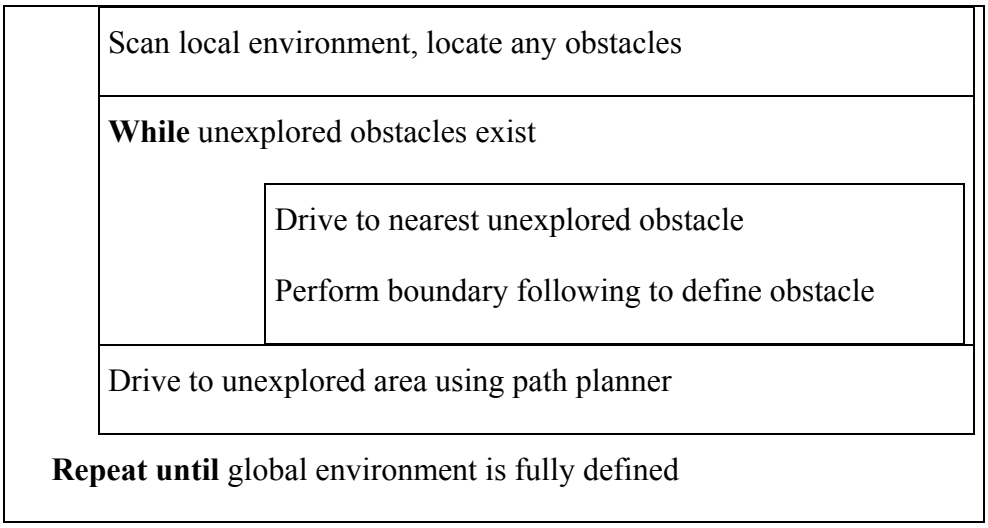

Figure 3.4: Mapping algorithm

3.4.1 Boundary-Following Algorithm

When a robot encounters an obstacle, a boundary-following algorithm is activated, to ensure that all paths around the obstacle are checked. This will locate all possible paths to navigate around the obstacle [29].

In this approach, the robot follows the edge of an obstacle (or a wall) until it returns close to its starting position, thereby fully enclosing the obstacle. Keeping track of the robot's exact position and orientation is crucial, because this may not only result in an imprecise map, but also lead to mapping the same obstacle twice or failing to return to an initial position. This task is non-trivial, since the robot works without a global positioning system and uses imperfect sensors.

Care is also taken to keep a minimum distance between robot and obstacle or wall. This should ensure the highest possible sensor accuracy, while avoiding a collision with the obstacle. If the robot were to collide, it would lose its position and orientation accuracy and may also end up in a location where its maneuverability is restricted.

Obstacles have to be stationary, but no assumptions are made about their size and shape. For example, it is not assumed that obstacle edges are straight lines or that edges meet in rectangular corners. Due to this fact, the boundary following algorithm must take into account any variations in the shape and angle of corners and the curvature of obstacle surfaces.

Path planning is required to allow the robot to reach a destination such as an unexplored area in the physical environment. Many approaches are available to calculate paths from existing maps. However, as in this case the robot is just in the process of generating a map, this limits the paths considered to areas that it has explored earlier.

This project's approach relies on a path planning implementation that uses minimal map data like the DistBug algorithm [35] to perform path planning. The DistBug algorithm uses the direction toward a target to chart its course, rather than requiring a complete map. This allows the path planning algorithm to traverse through unexplored areas of the physical environment to reach its target.

4. ROBOT EXPERIMENTS AND RESULTS

Using removable walls, we set up several environments to test the performance of our map generation algorithm. The following figures show a photograph of the environment together with a measured map and the generated map after exploration through the robot. There are two different color lines on the generated map. The red line shows localization of the robot on the map and the purple line represents the boundary of the map. We will analyse that the generated map how to be affected by the localization errors due to the odometry error and sensor measurement errors with the help of following mapping experiments.

For this experiment, we used the environment shown in Figure 4.1. It has all the required characteristic features to test our algorithm. The map boundary contains two corridors and corners require sharp 90° turns during the boundary following routine.

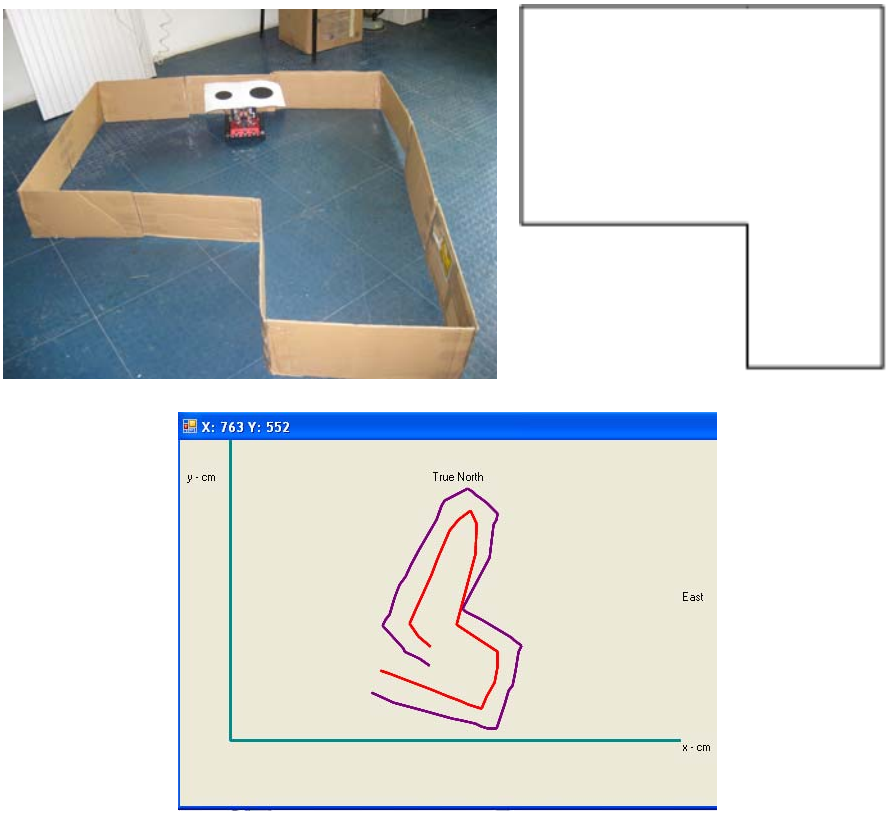

Figure 4.1: Experiment 1 photograph, measured map and generated map.

In Figure 4.1, the comparison between the measured and generated map shows a good resemblance between the two maps. As a result of the odometry error, the starting and ending points of the map couldn't intersect. The corners of the map are found successfully because the corner angles are greater than or equal to right angle.

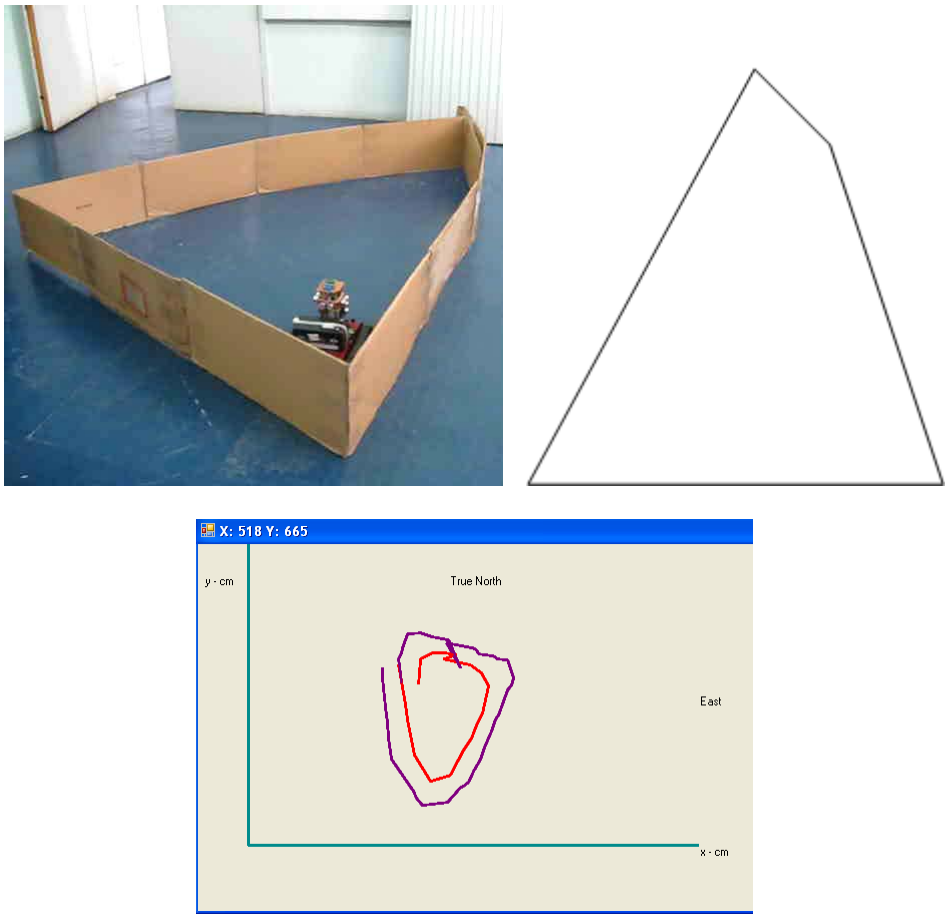

Figure 4.2: Experiment 2 photograph, measured map and generated map.

Figure 4.2 displays a triangular map that requires less frequent turns by the robot. Thus, the odometry error is less than it is in the previous experiment. As a result of the odometry error, the starting and ending points of the map couldn't intersect. Again, both maps show the same shape, although the corner angles are not as closely mapped as in the previous example because of the reflection problem of the ultrasonic sensors as described in section 2.2.3.3.

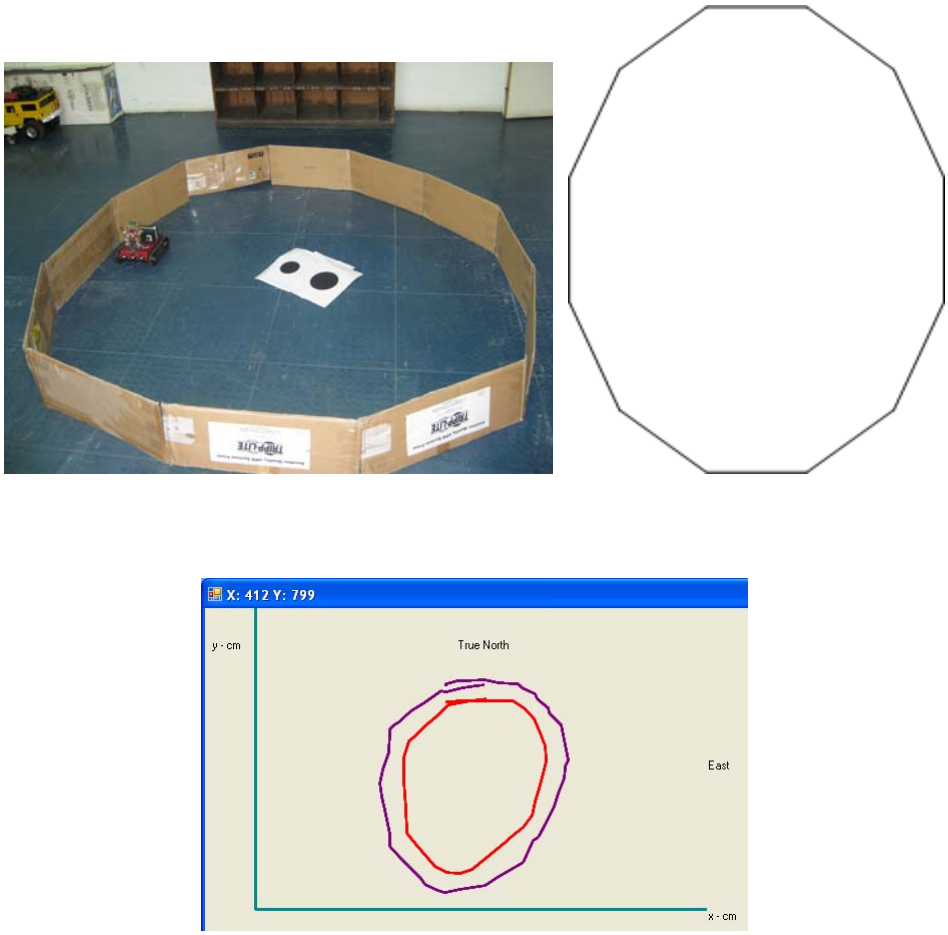

Figure 4.3: Experiment 3 photograph, measured map and generated map.

The final environment in Figure 4.3 contains non-rectangular walls. This tests the algorithm's ability to generate a map in an environment without the assumption of right angles. The turnings with less than right angle at the corners causes almost zero odometry error because of the less slippage of the robot, so the starting and ending points on the map could intersect. So, the generated map shows the same shape with the measured map. But, there is a little error on the map due to the ultrasonic sensor measurement error. It can be seen on the generated map, the purple starting and ending points of the boundary lines couldn't intersect.

For comparing the measured map with the generated map, a relation between the two maps must be found. One of the maps must be translated and rotated so that both maps share the same reference point and orientation. Next, an objective measure for map similarity has to be found. Although it might be possible to compare every single point of every single line of the maps to determine a similarity measure, we believe that such a method would be rather inefficient and not give satisfying results. Instead, we identify key points in the measured map

and compare them with points in the generated map. These key points are naturally corners and vertices in each map and can be identified as the end points of line segments. Care has to be taken to eliminate points from the correct map which lie in regions which the robot can not sense or reach, since processing them would result in an incorrectly low matching level of the generated map.

The main source of error is the robot positioning using odometry. With every move of the robot, small inaccuracies due to friction and wheel slippage leads to errors in the perceived robot position and orientation. Although these errors are initially small, the accumulation of these inaccuracies can lead to a major problem in robot localization, which will directly affect the accuracy of the generated map.

The second source of error in our map generation algorithm is related to the ultrasonic sensors. These sensors give reliable measurements within the range of 3 to 100 centimeters. However, individual readings can deviate by as much as 3 centimeters. Consequently, these errors must be considered and included in our test environment. Furthermore, another source of the ultrasonic sensor error is the reflection problem of the ultrasonic sensors. Due to this problem, the corners of the map that have less than right angle between the walls can not be shown accurately. As a result of the collision avoidance algorithm of the robot, it can not move to the inside of the corner and it can not sense the acute angled corners accurately because of the reflection problem of the ultrasonic sensors.

Varying error magnitudes were introduced to test our implementation of the map generation routine.

Especially corners in the environment become less accurately defined when they are acute angled. Nevertheless, the mapping system maintained the correct shape and structure even though errors were introduced.

We achieve a high level of fault tolerance by employing the following techniques:

• Controlling the robot to drive as close to an obstacle as possible.

• Limiting the error deviation of the ultrasonic sensors.

• Selectively specifying which points are included in the map.

By limiting the points entered into the map representation to critical points such as corners, we do not record large deviations along a straight edge of an obstacle.

This makes the mapping algorithm insensitive to sensor measurement errors along straight lines. Furthermore, by maintaining a close distance to the obstacles while performing boundary following, any sudden changes in sensor readings from the electronic compass or ultrasonic sensors can be detected as errors and eliminated.

Collision detection and avoidance routines are of greater importance in the physical environment, since the robot relies on odometry using wheel encoders for determining its exact position. A collision may cause wheel slippage, and therefore invalidate the robot's position data.

5. CONCLUSIONS AND FUTURE WORK

The aim of this project is to create a mobile robot that can map its environment and locate itself within the environment. This objective is mostly accomplished. The robot was designed and built in such a way that it can be used for localization and mapping applications. The robot was equipped with many sensors and actuators so as to sense the environment and navigate through the environment without colliding the obstacles.

According to the experiments made with the robot, map generation is affected by robot positioning using odometry. With every move of the robot, small inaccuracies due to friction and wheel slippage leads to errors in the perceived robot position. The accumulation of these inaccuracies can lead to a major problem in robot localization, which will directly affect the accuracy of the generated map. The other error source in map generation is related with the reflection problem of the ultrasonic sensors. Due to this problem, the corners of the map that have less than right angle between the walls can not be shown accurately.

The accuracy of the localization is a very important part in the map construction problem. Odometry is used as a localization method in this project. Although fast and relatively easy to implement, odometry suffers from the accumulation of errors. If an update from one or more bad locations is included in the map, the map will be destroyed. Therefore, major improvements can be made to the localization algorithm used in this project. Probabilistic techniques such as Bayes Filter or Monte Carlo Localization as explained in previous chapter can be used to improve the localization accuracy. This enables the robot to determine its location from previous measurements. Combined with probabilistic methods, odometry can provide a more accurate estimation of real location by eliminating any accumulated errors. And also sensors used for odometry can be improved or more accurate inertial measurement unit can be used for localization.

Probabilistic techniques can also be used to for mapping due to the inaccuracy of sonar range readings. The inaccuracies inherent in a sonar measurement require more

than marking occupied or unoccupied cells. Therefore, it will be more useful to specify an occupancy value for each cell in the grid map that indicate whether the cell is occupied, empty or unexplored. Then, the probability that a cell is occupied is calculated and thresholds can be used to determine if the cell is occupied or unoccupied. On the other hand, ultrasonic range finders can be replaced with laser range finder for more accurate measurements.

REFERENCES

[1] **Schraft R. D. and Volz H.,** 1996. Serviceroboter, Innovative Technik in Dienstleistung und Versorgung, Springer-Verlag, Berlin, Heidelberg.

[2] **Nehmzow, Ulrich,** 2000. Mobile robotics: a practical introduction. Springer, London.

[3] **N.J. Nilson,** 1969. A Mobile Automation: An application of Artificial Intelligence Techniques, Proc. IJCAI, Washington DC.

[4] **A.M. Thompson,** 1977. The Navigation System of the JPL Robot, Proceedings Fifth IJCAI, Cambridge MA.

[5] **Hans Moravec,** 1979. Visual Mapping by a Robot Rover, Proc. 6th IJCAI, Tokyo.

[6] **iiRbotics Web Site,** http://iirobotics.com

[7] **Muirhead B.K.,** 1997. Mars Pathfinder flight system integration and test, Aerospace Conference, 1997. Proceedings., IEEE , Volume: 4 , 1-8 Feb.

[8] **Tanie, K,** 2003. Humanoid robot and its application possibility, Multisensor Fusion and Integration for Intelligent Systems, MFI2003. Proceedings of IEEE International Conference on , 30 July-1 Aug.

[9] **Aronson, Z.H., Lechler, T., Reilly, R.R., Shenhar, A.J.,** 2001. Project spirit-a strategic concept, Management of Engineering and Technology, 2001. PICMET '01. Portland International Conference on , Volume: 1 , 29 July-2 Aug.

[10] **John J. Leonard and Hugh F. Durrant Whyte,** 1992. Directed Sonar Sensing for Mobile Robot Navigation, Kluwer Academic Publishers.

[11] **Raschke U. and Johann Borenstein,** 1990. A comparison of Grid-type Map building Techniques by Index of Performance, Proceedings of IEEE International Conference on Robotics and Automation, VOL. 3, pp. 1828-1832.

[12] **Polaroid,** 1987. Ultrasonic Ranging System.

[13] **Polaroid,** 1998. 6500 Series Ranging Module.

[14] **C. Biber, S. Ellin, E. Shenk and J. Stempeck,** 1980. The Polaroid Ultrasonic Ranging System, Presented at the 67th AES Convention.

[15] **Martin Beckerman and E. M. Oblow,** 1990. Treatment of Systematic Errors in the Processing of Wide-Angle Sonar Sensor Data for Navigation,

IEEE Transactions on Robotics and Automation, VOL. 6, No2, pp. 137-145.

[16] **Roman Kuc and Billur Barshan,** 1989. Navigation Vehicles Through An Unstructured Environment With Sonar, IEEE Journal of Robotics and Automation, pp. 1422-1426.

[17] **Devantech CMPS03 Technical Specification,** http://www.robot-electronics.co.uk/htm/cmps3doc.shtml

[18] **UDEA UTR-C12 FSK Transceiver Module Datasheet,** http://www.udea.com.tr/eng/1/UTR-C12%20data%20sheet.pdf

[19] **ATMEL Atmega 128 Microcontroller Datasheet,** http://www.atmel.com/dyn/products/product_card.asp?part_id=2018

[20] **I2C-Bus Specification and User Manual,** http://www.nxp.com/acrobat_download/usermanuals/UM10204_3.pdf

[21] **L298 Dual Full-Bridge Driver Datasheet,** http://www.st.com/stonline/books/pdf/docs/1773.pdf

[22] **Devantech SRF08 Ultrasonic Range Finder Technical Specification,** http://www.robot-electronics.co.uk/htm/srf08tech.shtml

[23] **Sharp GP2D12 Infrared Range Finder Datasheet,** http://document.sharpsma.com/files/GP2D12-DATA-SHEET.PDF

[24] **Hvwtech I2C-It IR Range Finder Technical Specification,** http://www.hvwtech.com/products_view.asp?ProductID=665

[25] **Wikimedia Foundation, Inc.,** 2006. Wikipedia, The Free Encyclopedia http://en.wikipedia.org/wiki/Odometry

[26] **Wikimedia Foundation, Inc.,** 2006. Wikipedia, The Free Encyclopedia http://en.wikipedia.org/wiki/Dead_reckoning

[27] **Wikimedia Foundation, Inc.,** 2006. Wikipedia, The Free Encyclopedia http://en.wikipedia.org/wiki/Bayes_filter

[28] **Wikimedia Foundation, Inc.,** 2006. Wikipedia, The Free Encyclopedia http://en.wikipedia.org/wiki/Monte_Carlo_localization

[29] **Thomas Bräunl,** 2003. Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems. Springer-Verlag Berlin Heidelberg, Germany.

[30] **J. Borenstein, B. Everett, and L. Feng,** 1996. Navigating Mobile Robots: Systems and Techniques. A. K. Peters, Ltd., Wellesley.

[31] **D. Kortenkamp, R.P. Bonasso,** 1998. Al-based Mobile Robots, Case studies of Successful Robot Systems, Cambridge, MIT Press.

[32] **Sebastian Thrun,** 2002. Robotic Mapping: A Survey, Carnegie Mellon University, Pittsburgh, CMU-CS-02-111.

[33] **J. Borenstein, H. R. Everett, L. Feng,** 1996. Sensors and Methods for Mobile Robot Positioning, The University of Michigan, Pittsburgh.

[34] **Phillip C.-Y Sheu and Q. Xue,** 1993. Intelligent Robotic Planning Systems, World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[35] **Ishay Kamon and Ehud Rivlin,** 1997. Sensory-Based Motion Planning with Global Proofs, IEEE Transactions on Robotics and Automation, VOL. 13, No. 6.

[36] **A. Eliazar, R. Parr,** 2003. DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks, in: Proc. of the Int. Conf. on Artificial Intelligence, IJCAI, Acapulco, Mexico, pp. 1135–1142.

[37] **G. Grisetti, C. Stachniss, W. Burgard,** 2005. Improving Grid-Based SLAM With Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling, in: Proc. of the IEEE Int. Conf. on Robotics & Automation, ICRA, Barcelona, Spain, pp. 2443–2448.

[38] **J.-S. Gutmann, K. Konolige,** 1999. Incremental Mapping of Large Cyclic Environments, in: Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation, CIRA, Monterrey, CA, USA, pp. 318–325.

[39] **D. H¨ahnel, W. Burgard, D. Fox, S. Thrun,** 2003. An efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments From Raw Laser Range Measurements, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS, Las Vegas, NV, USA, pp. 206–211.

[40] **M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit,** FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges, in Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence, pp. 1151-1156, 2003.

[41] **M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit,** 2002. FastSLAM: A Factored Solution to Simultaneous Localization and Mapping, in: Proc. of the National Conference on Artificial Intelligence, AAAI, Edmonton, Canada, pp. 593–598.

[42] **K. Murphy,** 1999. Bayesian Map Learning in Dynamic Environments, in: Proc. of the Conf. on Neural Information Processing Systems, NIPS, Denver, CO, USA, pp. 1015–1021.

[43] **S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, H. Durrant-Whyte,** 2004. Simultaneous Localization and Mapping With Sparse Extended Information Filters, International Journal of Robotics Research 23 (7–8).

[44] **A. Doucet, J. de Freitas, K. Murphy, S. Russel,** 2000. Rao–Blackwellized Particle Filtering for Dynamic Bayesian Networks, in: Proc. of the Conf. on Uncertainty in Artificial Intelligence, UAI, Stanford, CA, USA, pp. 176–183.

[45] **Simon J. Juliera,_, Jeffrey K. Uhlmannb,** 2006. Using Covariance Intersection for SLAM, ScienceDirect Robotics and Autonomous Systems 55 (2007) 3–20.

[46] **J. E. Guivant and E. M. Nebot,** 2001. Optimization of the Simultaneous Localization and Map-Building Algorithms for Real-Time Implementation, in IEEE Trans. on Robotics and Automation, vol. 17, no. 3, pp. 242-257.

[47] **H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun,** Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, 2005.

[48] **A. A. Makarenko, S. B. Williams, F. Bourgault and H. F. Durrant-Whyte,** An Experiment in Integrated Exploration, in Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 534-539, 2002.

[49] **R. Sim and N. Roy,** Global A-Optimal Robot Exploration in SLAM, in Proc. of IEEE Int. Conf on Robotics and Automation, pp. 673-678, 2005.

CURRICULUM VITAE

Başar Denizer was born in İstanbul, Turkey, in1981. He received B.Sc. degree in electrical and electronics engineering in 2004 from the Yeditepe University. He started his M.Sc. program in mechatronics engineering in 2005.