UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MASTER THESIS

"Implementation of a Mobile Wireless Sensor Node for the Replacement of Failed Nodes"

Author

Ntelis Giorgos

Supervisor

Lalis Spyros, Associate Professor

Committee members

Antonopoulos Christos, Assistant Professor

Bellas Nikolaos, Associate Professor

Volos, March 2014

Πανεπιστήμιο Θεσσαλίας

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μεταπτυχιακή Διπλωματική Εργασία

“Ανάπτυξη Κινητού Κόμβου σε Ασύρματο Δίκτυο Αισθητήρων για την Αντικατάσταση μη Λειτουργικών Κόμβων”

Συγγραφέας

Ντελής Γιώργος

Επιβλέπων Καθηγητής

Λάλης Σπύρος, Αναπληρωτής Καθηγητής

Μέλη Επιτροπής

Αντωνόπουλος Χρήστος, Επίκουρος Καθηγητής

Μπέλλας Νικόλαος, Αναπληρωτής Καθηγητής

Βόλος, Μάρτιος 2014

University Of Thessaly

Department of Electrical and Computer Engineering

"Implementation of a Mobile Wireless Sensor Node for the Replacement of Failed Nodes"

A Thesis Presented

By

Ntelis Giorgos

APPROVED BY:

____

Lalis Spyros, Associate Professor

____

Antonopoulos Christos, Assistant Professor

____

Bellas Nikolaos, Associate Professor

Volos, March 2014

Dedicated to my family

Abstract

Usually wireless sensor networks consist of low-power sensor nodes with limited lifetime. This may impose a problem for an application relying on a wireless sensor network. A non-operating node may also lead to loss of data from other nodes as well. The purpose of this project is the detection and replacement of a non-operating stationary wireless node using an autonomous GPS-guided robot. An autonomous GPS robot was designed, constructed and tested. The robot was also integrated as part of an outdoor Wireless Sensor Network. Considering the sensor network a fixed topology was assumed and two applications were developed; one sensing application for the nodes and a Java application responsible for the detection of a failed node. Real world outdoor experiments were conducted using the robot and a number of Imote2 wireless sensor nodes, which confirmed the successful node replacement.

Περίληψη

Τα ασύρματα δίκτυα αισθητήρων αποτελούνται συνήθως από κόμβους χαρακτηριστικό των οποίων μπορεί να είναι η χαμηλή αυτονομία. Αυτό μπορεί να αποτελέσει πρόβλημα για μία εφαρμογή που στηρίζει τη λειτουργία της σε κάποιο ασύρματο δίκτυο αισθητήρων. Επίσης ένας μη λειτουργικός κόμβος μπορεί να έχει ως αποτέλεσμα τη μη λήψη δεδομένων και από άλλους κόμβους του δικτύου. Σκοπός αυτής της εργασίας είναι η ανακάλυψη και η αντικατάσταση ενός τέτοιου μη λειτουργικού κόμβου με τη χρήση ενός αυτόνομου ρομπότ το οποίο πλοηγείται με τη βοήθεια GPS. Κατά τη διάρκεια της διατριβής ένα τέτοιο αυτόνομο ρομπότ σχεδιάστηκε, κατασκευάστηκε και δοκιμάστηκε. Επίσης το ρομπότ υλοποιήθηκε ως μέρος του ασύρματου δικτύου αισθητήρων. Όσον αφορά το δίκτυο αισθητήρων υποθέτουμε ότι οι κόμβοι είναι τοποθετημένοι σε γνωστές, προκαθορισμένες θέσεις. Για την επίτευξη των στόχων μας δύο επιπλέον εφαρμογές αναπτύχθηκαν. Μία εφαρμογή που τρέχει στους κόμβους του δικτύου και μία εφαρμογή υλοποιημένη σε Java σκοπός της οποίας είναι η ανίχνευση ενός μη λειτουργικού κόμβου. Τέλος, πραγματοποιήθηκαν δοκιμές σε εξωτερικό χώρο χρησιμοποιώντας το ρομπότ και ένα ασύρματο δίκτυο αισθητήρων αποτελούμενο από τους ασύρματους κόμβους Imote2, οι οποίες επιβεβαίωσαν την επιτυχημένη αντικατάσταση του μη λειτουργικού κόμβου.

Acknowledgements

I would like to express my deepest gratitude to my advisors Lalis Spyros, Antonopoulos Christos, and Bellas Nikolaos for their help and guidance throughout this process; their ideas and feedback have been absolutely invaluable. I would also like to thank my family for the encouragement and love they shared with me throughout my entire life, for believing in me and for supporting my decisions. I dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) have become an important technology due to advances in low-cost low-power wireless communications combined with inexpensive processing power and sensing capabilities, all available after years of research. A WSN is basically a collection of (typically small sized, embedded) devices, referred to as *nodes*, organized into a cooperative network, providing an interface between the physical and online world. These devices are able to sense the environment and wirelessly forward data to a so-called *sink*, usually via multiple hops. The sink typically processes the gathered data locally and makes them available to external systems through an Internet-enabled *Gateway*. Nodes in a WSN can serve one or more functionalities in parallel. A sensor node can be a source node providing information to the network, a relay node which receives and retransmits information with or without processing, a sink node where the information is required, or an actuator, able to control some device.

WSNs are already used in several applications for surveillance and tracking, infrastructure protection, precision agriculture, in smart environments and health applications. Figure 1 illustrates a WSN that runs an event detection application, featuring a single sink node. When a wireless sensor node detects a possible danger or threat it forwards the information through the WSN to the sink node. From there the information can be processed locally or remotely through the Internet.

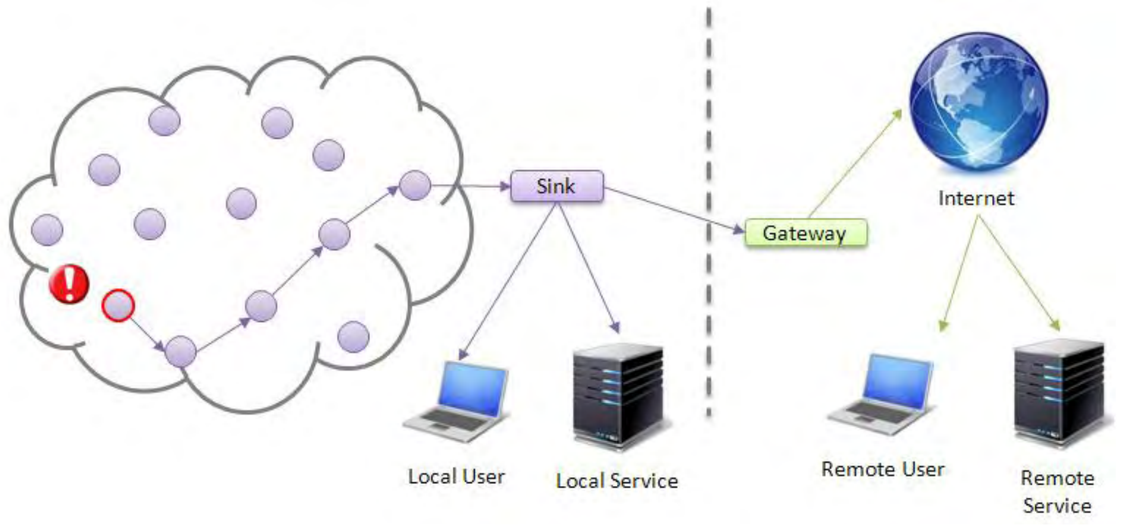


Figure 1 – Wireless Sensor Network example

Wireless sensor nodes are the building blocks in a WSN. A sensor node offers capabilities of on-board processing, sensing and networking. A high level block diagram describing a sensor node is presented in figure 2. More specifically a node usually consists of:

- A Microcontroller optimized for embedded systems with low power consumption (e.g. ATmega 8 bit controllers, TI MSP430)
- Memory (RAM and ROM)
- A transceiver which enables the communication with other nodes
- A Power unit responsible for providing the required energy to the sensor node.
- A set of sensors that can sense a physical phenomenon.

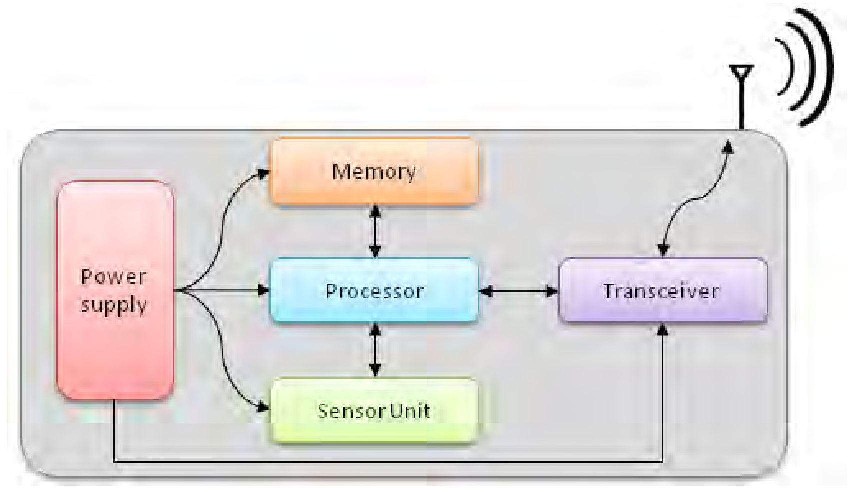


Figure 2 - Typical node architecture

Designing WSNs is challenging.

1) Generally a node is limited in energy, computation, storage, bandwidth, and transmission range. Especially in outdoor deployments, nodes are powered using a battery which calls for a trade-off between performance and lifetime. Microcontrollers with limited processing power are typically used to keep power consumption low.
2) Nodes operating in nature are prone to damage and may experience communication problems due to harsh environmental conditions and interference.
3) The deployment of nodes can vary depending on the application. It may be random (e.g. nodes dropped from an aircraft), planned in advance (e.g. in buildings), or dynamic (if the nodes move around).
4) In remote, inaccessible or large deployments manual node maintenance may not be an option. Hence it is important to be able to monitor and repair/reprogram the WSN remotely.

5) Last but not least, one may need to deal with security issues, e.g., the physical tampering of nodes and the interception of confidential information that is transmitted over the air.

The functionality of a WSN can be further enhanced by introducing mobility to some of the nodes, which is a key aspect of this thesis. Mobile nodes combine the properties of sensing, computation, communication, and actuation making collaboration between robots and WSN a research topic of growing interest. Autonomous robots have been usually used as high-performance sensor nodes in WSNs, enhancing the functionalities of the WSN. By manipulating the mobile sensor nodes, it is possible to create or dynamically change certain topologies according to the application's requirements. For example Hybrid WSNs consisting of both stationary and mobile nodes have helped to develop Testbeds for general experimentation. Authors in [1] established a common framework for evaluation and comparison of different methods and interoperations of robots and WSNs. Authors in [2] have been developing an information gathering system using mobile robots and WSNs in underground spaces in post-disaster environments. Authors in [3] used mobile robots that interact with the deployed nodes, gathering data which are then being reported to a gateway by the robots. Others used already deployed WSNs to access and control the robots [4].

Control and navigation of mobile nodes in a WSN imposes the challenge of finding the optimal path planning to efficiently achieve an objective. Two main approaches exist in WSN navigation. In classical methods positioning information of the navigation path exists, but in WSN-based methods the robot knows which node should reach next but it has no location information. Authors in [5] proposed an algorithm based on received signal strength indicator potential field (RSSI-PF) that enables the robot to reach a sensor node without any location and range information. Other approaches use a mobile node equipped with a GPS module [6]. This node moves around and periodically broadcasts its position to near static nodes.

1.2 Scope of this thesis

Continuous stream of data from the nodes is often essential for an application relying on a Wireless Sensor Network. Therefore, detecting nodes that have stopped operating may be crucial for the application. Also failure of a node may lead to loss of data from other nodes of the network due to possible broken communication links. The purpose of this project is the detection and replacement of a non-operating stationary wireless node using an autonomous GPS-guided robot called "E-vlampios".

We assume a WSN featuring a single sink node connected to a Gateway. Each sensor runs a sensing application measuring the environment’s light density, and forwards the data to the sink node. Moreover, the nodes are placed to known locations described by a set of coordinates. The Gateway runs a Java application which keeps track of the received measurements and it is able to detect a possible failure of a sensor node. A node is declared faulty when no further measurements are being received from it. Detection of a failed node initiates a recovery process. The Gateway informs the robot about the node that needs to be replaced and the robot navigates to the desired location using a GPS module and a combination of other sensors, and then it deploys a new node. Our approach differs from others in that the faulty node is physically replaced by a new node.

This thesis describes work in the following topics:

- Construction and navigation of the “E-vlampios” robot
- Implementation of the sensing application
- Detection of a faulty node
- Real world experiments

1.3 Organization of this thesis

The remainder of this thesis is organized as follows:

- **Chapter 2** describes the design of the mobile robot. This chapter discusses operational principles of the autonomous robot, and presents details for the robot implementation.
- **Chapter 3** presents details about the navigation algorithm that makes use of the GPS module. For example how distance and bearing between two points on the globe can be calculated. Also discusses the implementation of a GPS data logger.
- **Chapter 4** focuses on the WSN. It presents details about the sensing application, and then it goes through the algorithm developed for the failed node detection.
- **Chapter 5** focuses on outside experiments conducted in an open area and also presents the related results.

2 The Robot System

2.1 Overall System architecture

This chapter deals with the assembling process of the robot. Technical details about the platforms used and their integration with the sensors and actuators will be described. One of the main tasks of this thesis was to build an autonomous mobile robot that could be used in real world experiments. The mechanical design of the robot is based on the project described in [7]. For the purposes of this thesis many alterations had to be made involving new configuration of the chassis and the use of new hardware. The robot should be able to navigate without external help using a GPS receiver and a number of additional sensors. Multiple sensors and actuators have been used in this project, like: a GPS receiver, a digital compass, ultrasonic sensors for obstacle detection, an LCD display, two DC motors with their control board, and one servo motor combined with a robotic claw to transfer a node. A Wi-Fi module and a camera have been used to feed video back to the user through an Ad-Hoc network between the robot and the user's computer. Also special circuits have been constructed to supply the robot with the necessary power.

The robot's main processing unit is the BeagleBoard platform, which is a low power, single board computer based on an ARM CPU, running at 720MHz. The BeagleBoard has a limited number of I/O pins and it does not have analog pins. Furthermore connecting multiple sensors and other external devices may be a tricky procedure due to the lack of libraries and the limited interest of other developers. For these reasons it was decided to use the popular ATmega328 microcontroller complementary to the BeagleBoard. Most of the sensors and actuators were finally connected to the ATmega328. The communication between the ATmega328 and the BeagleBoard is possible through a Serial interface. The ATmega328 receives commands from BeagleBoard to control the motors and servo, and also collects data from the sensors feeding the BeagleBoard with useful information.

During the development of this thesis an Arduino UNO R3 board was used for testing all these peripherals. The Arduino UNO R3 is also based on the ATmega328 making perfect platform for testing. Instead of using the Arduino on the final product, a new board based on the ATmega328 was especially designed and developed. The BeagleBoard is responsible for navigating the robot, communicating with the Gateway and also broadcasting the video captured from the webcam. The robot was implemented as a mobile wireless sensor node by fitting on it an Imote2 Wireless Sensor Node. The Imote2 communicates through a serial port with the BeagleBoard and allows the robot to communicate with other wireless nodes of the WSN.

More information about the BeagleBoard and the Arduino UNO can be found in Appendix A and Appendix B respectively. A high level diagram of the robot's configuration containing all the basic components is represented in figure 3 below.

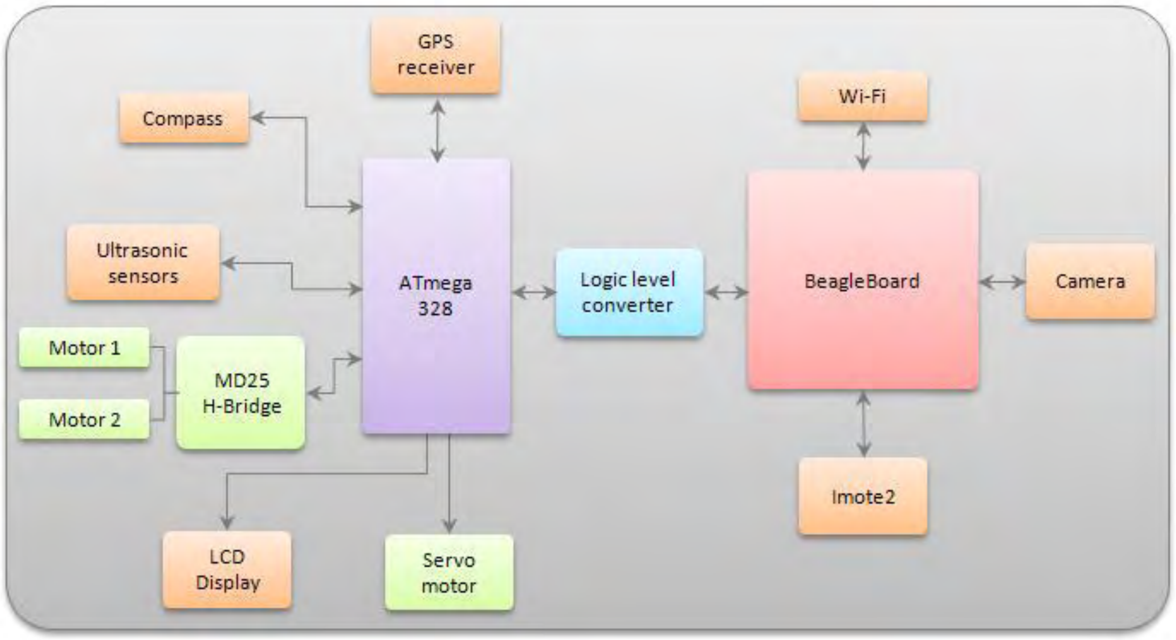


Figure 3 - Robot's high level block diagram

2.2 Robot Hardware

2.2.1 The RD02 Robot Drive System

The Devantech MD25 Dual 12V 2.8A H-Bridge DC Motor Driver is designed to work with the EMG30 gear motor. The MD25 Dual Motor Driver is able to drive two of these motors. Together with two mounting brackets, two 100mm wheels with hubs already fitted they make up the RD02 Robot Drive system, shown in figure 4. This motor driver features two modes of operation, direct individual control of the motors or the ability to send a speed and a turn commands. Only a single 12v battery capable

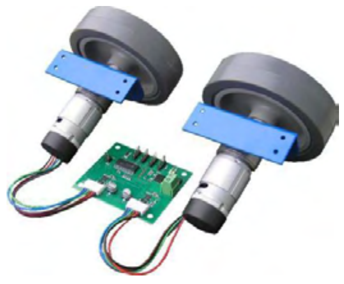


Figure 4 - The RD02 Robot Drive System

of supplying peak current of 6Amps is required to power the system. Power for the logic comes from an on-board 5v regulator which is also capable of supplying up to 300mA to external circuits. The EMG30 is a 12V motor fully equipped with encoders and a 30:1 reduction gearbox. By reading the motor encoders, the distance traveled and direction can be determined. The board can be controlled by Serial or the I2C interface. The two jumpers on the MD25 board shown in figure 5 (Serial/I2C Select) must be removed in order to activate the I2C mode. The clock frequency of the I2C Bus must not be greater than 100 KHz. Details about the I2C protocol are presented in Appendix C.

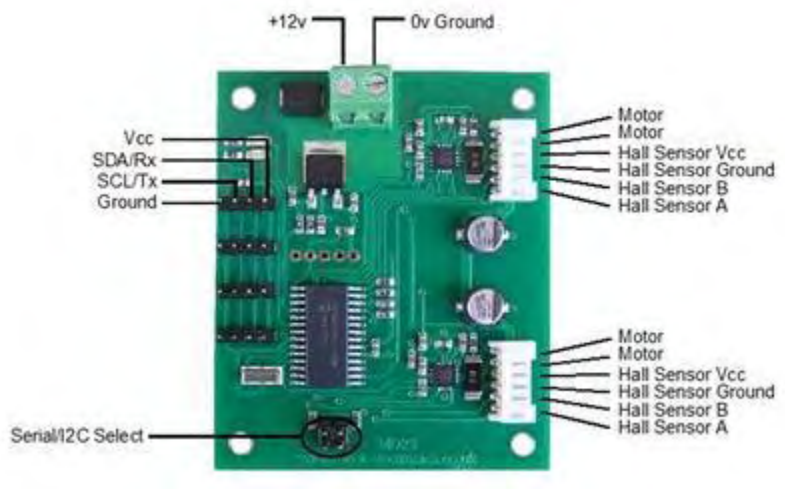


Figure 5 - The MD25 H-Bridge Motor Driver

2.2.2 The Adafruit Ultimate GPS module

The Adafruit Ultimate GPS module was the one chosen for this thesis. The breakout board is built around the MTK3339 chipset, a high-quality GPS module that can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver (-165 dB), and a built in antenna [8]. It can support an update frequency of up to 10Hz while consuming only 20mA. The onboard voltage regulator enables it to operate at voltages ranging from 3.3 to 5VDC. It also comes with an on-board LED which blinks once per second when no fix is available and then every 15 seconds to preserve energy.

Figure 6 - The Adafruit Ultimate GPS

Data acquisition is possible through a serial interface. The module's default baud rate is 9600 bps, but the module can be configured to use different rates. A text like the following appears when the module acquires a fix. Each row of the text below starting with the $GP prefix is known as a NMEA 0183 sentence. More details about how the GPS works and the NMEA format are available in Appendix D6Appendix D and Appendix E respectively.

```
$GPGGA,172725.000,3921.8851,N,02257.0867,E,2,8,1.27,44.4,M,35.9,M,0000,0000*6A
$GPRMC,172725.000,A,3921.8851,N,02257.0867,E,1.77,121.21,050813,,,D*66
$GPGGA,172726.000,3921.8847,N,02257.0867,E,2,8,1.27,44.4,M,35.9,M,0000,0000*6E
$GPRMC,172726.000,A,3921.8847,N,02257.0867,E,1.42,124.03,050813,,,D*61
$GPGGA,172727.000,3921.8843,N,02257.0871,E,2,8,1.27,44.4,M,35.9,M,0000,0000*6C
$GPRMC,172727.000,A,3921.8843,N,02257.0871,E,1.59,125.29,050813,,,D*60
```

Figure 7 - GPS module output

As mentioned earlier the module's built-in antenna provides -165dBm of sensitivity. The results of the first experiments were promising, but it was decided to boost the GPS's performance using an external active antenna which provides an additional 28dBm of gain. For this purpose an embedded SMA antenna was purchased. This antenna comes with a standard SMA connector on the end. To connect it to the Ultimate GPS a uFL to SMA adaptor is necessary. The Ultimate GPS will automatically detect and use the attached antenna and no additional configuration is needed.

Figure 8 - External GPS antenna

To check whether the internal or external antenna is used, the GPS module must be configured to output the antenna status using the sentence: "$PGCMD,33,1*6C" . Sending this command to the module will force it to output a sentence that contains antenna information like this: "$PGTOP,11,x " where x is the status number explained in the table below.

Table 1- External GPS antenna status

x value	Status
1	Antenna short problem
2	Using internal antenna
3	Using external antenna

2.2.3 Compass

Now that the robot it is able to acquire its current position using the GPS module, it can use this information to calculate the direction of movement in order to reach a certain waypoint, described by its own latitude and longitude. To do that the robot must somehow find its current heading. This can be accomplished using a digital compass or the GPS module again. To acquire valid heading information from the GPS module, the robot must be moving at a sufficient speed. Getting valid heading values from GPS when the robot is stationary is not an option. As a consequence the robot at first must be travelling around randomly (possibly in the wrong direction) until it gets a valid heading value and then it must periodically make corrections to its movement, based on updated heading values.

To address the problems mentioned previously, a digital compass was finally used. Getting valid data from a digital compass may also be tricky. The mathematical formulas used must be carefully chosen as well as the positioning of the compass on the robot. A digital compass is very easily affected by anything that emits EM, so it must be kept away from motors, transformers, inductors, etc.

One problem that traditional compasses have and also exist to its digital counterpart is that they need to be held flat to function properly. Compasses use only the X and Y axis of the Earth’s magnetic field, so when the compass is not parallel to these axes the amount of magnetism felt by the needle will change leading to inaccurate readings. To implement a tilt compensated compass the magnetic field felt by the Z axis must be included in the calculations. A tilt compensated compass is implemented by incorporating an accelerometer into the system. How that is achieved will not be explained because it is out of this thesis’s scope. During the project, a tilt compensated compass was chosen at first, but soon after some tests in the real world, some problems emerged. The vibrations caused by the robot’s rigid frame greatly affected the accelerometer readings, and as a result the compass readings fluctuated violently. To cope with this problem a simple magnetometer was finally used.

The board used is based on the HMC5883L compass module which is a three axis magnetometer, capable of sensing in three directions although only two (X and Y) were used for the reasons explained earlier. The direction of each axis is printed on the board as we can see in figure 9. Also the measuring range varies from ±1.3 to ±8 Gauss. Before starting working with the HMC5883L five header pins must be soldered. The module communicates over I2C (like the MD25 board mentioned earlier in this chapter). Furthermore the board can operate in voltages ranging from 3.3VDC to 5VDC.

Figure 9 - The HMC5883L compass module

2.2.3.1 Making turns based on compass readings

One of the primary tasks for every outdoor robot is its ability to turn to a desired heading accurately when knowing its current heading. One of the most common ways to achieve that is by using a kind of sensors named encoders. These sensors are attached to a rotating object (the EMG30 motors for example). A typical encoder uses optical sensors, a moving mechanical component and a special reflector to provide a series of electrical pulses as output [9]. Each wheel rotation is measured in certain pulses (or counts), 360 in the case of the EMG30 motor. Knowing the wheel's circumference and the number of counts, useful values like velocity, acceleration, and distance traveled, etc. can be calculated.

However, the use of encoders imposes some problems. It is usual (especially for an outdoor robot) that the wheels may slip. This means that the wheels rotate but the robot is not moving at all, leading to an increased error. Furthermore, the majority of encoders have an accuracy of approximately ±1 degree. This number may sound good enough for most applications, but inaccuracy multiplies over distance. For these reasons the idea of using the encoders lying in the EMG30 motors as a control mechanism of the robot's rotation was abandoned.

To accomplish that, it was decided to make turns based on the compass readings. Figure 10 presents a basic concept, where the robot must turn from the direction it is facing (CH) to a desired angle (DH).

Let current heading CH be 290° and desired heading DH be 190°. The challenge at this step is to turn the robot to the desired heading the most efficient way possible. Knowing its current heading the robot must decide to turn or right based on which is the shortest distance. In our example, the robot should turn counterclockwise turning only 100° and not clockwise which requires a 260° turn. Please note that a tolerance value must be inserted to the system due to the uneven terrain at which the robot operates and also due to the robot's rigid frame which causes vibrations during movement. The next code segment describes the basic formula behind the turning mechanism.

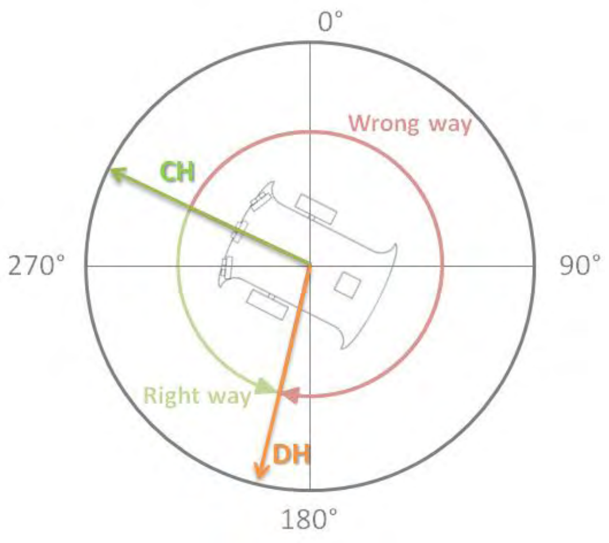


Figure 10 - Robot turn example

```
//desired_angle is a global variable
//TURN_TOLERANCE in degrees

void turn_robot(){

  int heading;

  //Get current heading from digital compass
  heading = read_compass();
  int diff = abs(heading - desired_angle);

  if(diff <= TURN_TOLERANCE){
    //stop motors then exit
    return;
  }

  if( diff < 180 && desired_angle > heading){
    //turn right
  }
  else if(diff < 180 && desired_angle < heading){
    //turn left
  }
  else if(diff > 180 && desired_angle > heading){
    //turn left
  }
  else if(diff > 180 && desired_angle < heading){
    //turn right
  }
  turn_robot();
}
```

Figure 11 - Turning formula

2.2.4 Ultrasonic sensors

The ability of object detection is essential for an autonomous robot that operates outdoors. To achieve that, a number of ultrasonic sensors were fitted on the robot. These sensors have a similar working as a sonar or radar. It uses high-frequency sound waves that are evaluated when the sensor receives them back. These sensors cannot be affected by surface, light, dust, etc. making them ideal for outside use.

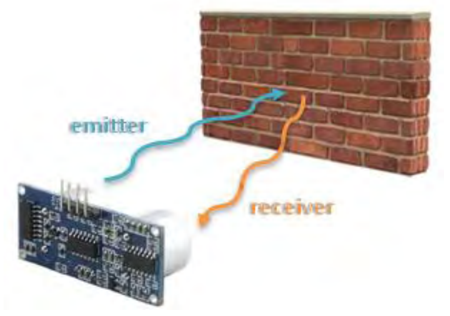


Figure 12 - Ultrasonic sensor

An ultrasonic sensor consists of two main parts; an emitter and a detector. It uses a piezoelectric transude to send and detect sound waves. The emitter creates high-frequency sound waves and the detector evaluates the echo of these signals after reflecting off the target. Sensors calculate the time between sending the signal and receiving the echo, in order to determine the distance from the object. It is known that distance equals the product of velocity and time. The same principal applies also here. Distance from an object can be calculated using the following equations.

$$Distance\ traveled\ by\ sound = Speed\ of\ sound * Time\ that\ sound\ travels$$

$$Distance\ from\ object = \frac{Distance\ traveled\ by\ sound}{2}$$

One of the most popular ultrasonic sensors used in robotics is the *HC-SR04*. This sensor provides measurement function between 2 and 400 centimeters at range accuracy of 3mm. The effective range of this sensor is approximately 30°, so three of these sensors were installed on the robot.

2.2.5 LCD Display

An LCD display was also fitted on the robot to display information about the robot's status. A basic 16 characters by 2 lines LCD was chosen as the primary display output (). Most LCDs have a parallel interface, meaning that a microcontroller has to manipulate several interface pins at once to control the display.

Figure 13 - Basic 16x2 LCD Display

The process of controlling the LCD involves putting data that we want to display into the data registers, and then putting instructions in the instruction register. Due to the limited number of the ATmega328p's I/O connections, it was essential to make the best use of the available pins. 6 pins are required for the LCD's parallel interface but this number can be brought down to only 3 pins using a Shift register like the *74HC595*. Details about controlling an LCD display using a shift register are presented in Appendix F.

2.2.6 Push buttons

Except the other components two push buttons were also added to the system. One for controlling the robotic gripper and one for implementing a simple menu system for the LCD display. Usually each button occupies one digital pin, but due to the limited number of I/O ports it was decided to use an analog port. Using an analog port the Arduino board is able to handle a series of buttons. This can be accomplished using a resistor network as voltage dividers, letting each button feed a different voltage to the analog pin. Thus by detecting the input voltage the button pressed can be identified.

An Analog to digital Converter is a very useful IC that converts an analog voltage to a digital representation [10]. These electronic circuits connect the analog world around us with the digital one. The ATmega328 has a built in ADC meaning that it can be used without any additional hardware. The ADC on the ATmega328 is a 10-bit ADC meaning it has the ability to detect 1,024 (2^{10}) discrete analog levels while offering 6 I/O pins (A0 – A5). Analog to digital conversions depend on the system voltage. ATmega328's operating voltage is 5V, meaning that 5V is equal to 1023, and anything less than 5V will be a value between 0 and 1023. The following formula can be used to calcite the digital representation of an input voltage.

$$\frac{ADC's\ resolution}{System\ voltage} = \frac{ADC\ reading}{Input\ voltage}$$

The circuit used is presented in figure 14figure 14 - two buttons one analog pin. When Button 0 is pressed input voltage is 5V or 1024 in the digital representation. On the other hand when Button 1 is pressed, the input voltage is ≈ 2.368V or 485.

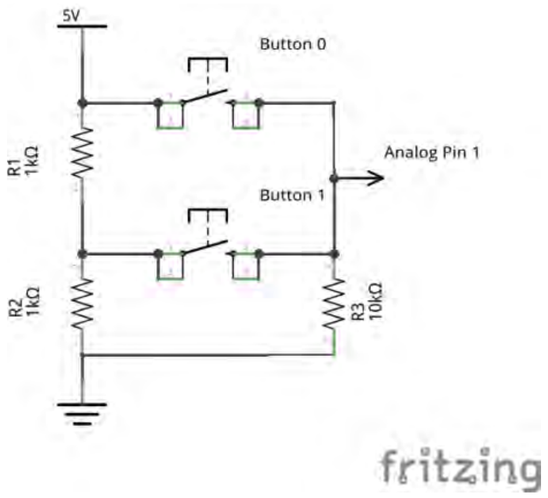


Figure 14 - Two buttons one analog pin

Buttons are imperfect like many other things. Often when a button is pressed only one time, two or more presses may be registered. It produces some oscillations called glitches or bounces. This phenomenon is related to the physical properties of buttons. When you press a button it may not make a perfect contact immediately [11]. This phenomenon lasts only some milliseconds but the microcontroller is fast enough to detect it. These bounces don't occur when the button is held down or not pressed. They only occur between during the press or release of a button. To encounter this problem a debounce technique must be applied. On way to accomplish that is by accepting a change in the button's state only if it lasts a pre-defined time period (for example 200ms).

A system that supports a simple LCD menu can be implemented using only one push button. Each time the button is being pressed the content of the LCD display changes. So the software must monitor the button 0 state. The next three figures represent the different display modes of the LCD screen. After the boot of the system, the screen displays information about the heading of the robot and the distances from objects which are acquired from the ultrasonic sensors. When button 0 is pressed the GPS coordinates appear, and if it is pressed again the number of satellites and the battery voltage levels appear.

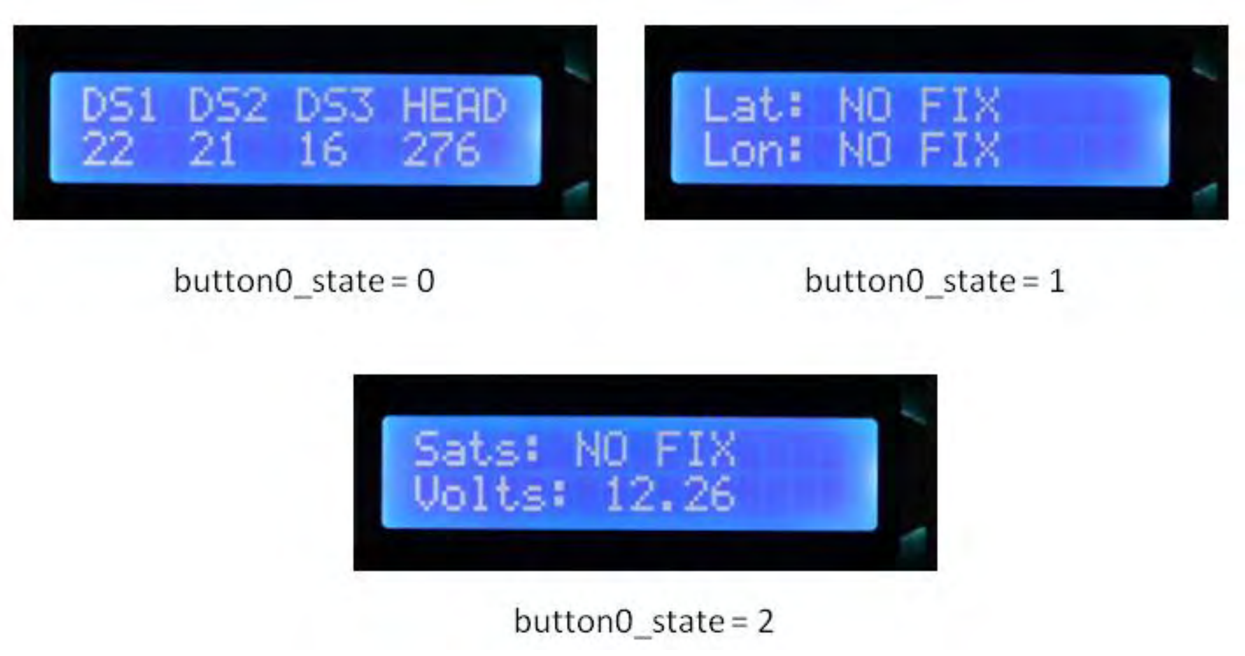


Figure 15 - LCD display modes

2.2.7 Logic Level converter

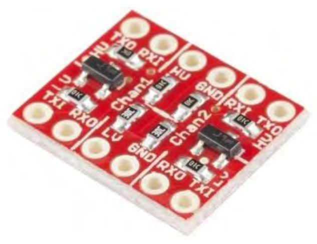


Figure 16 - Logic Level Converter

Since different logic families may use different voltages to represent 1 and 0, special techniques must be used for the interconnection between two different logic families. When interfacing a 3.3V device with a 5V device, a logic 1 from the 3.3V device will be interpreted as a logic 1 to the 5V device. However, when interfacing from a 5V device to a 3.3V a logic 1 may cause permanent damage to the 3.3V device. The BeagleBoard uses 1.8V logic levels while the ATmega328 uses 5V so converting the two signals of the serial interface is crucial.

To address this problem the Sparkfun Logic Level Converter (BOB-11978) was used, which can safely step down or step up signals. This board needs to be powered from the two voltage sources: high voltage 5V, and low voltage 1.8V. This board contains two Rx lines consisting of simple voltage dividers (not bidirectional) and two Tx lines which are bidirectional and they can also be used for an I2C bus. Each level shifter on the Tx lines is implemented using one N-channel MOS-FET with the following configuration. The gates are connected to the lowest supply voltage, the sources to the line of the lower voltage section and the drains to the line of the higher voltage section. The diode between the drain and substrate is present inside the MOS-FET.

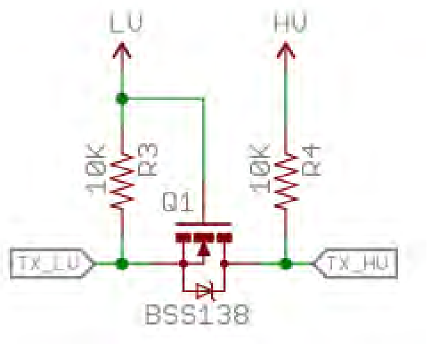


Figure 17 - MOSFET logic level converter

2.2.8 Servo motor and mechanical claw

The sensor node intended to replace the failed node was attached on a wooden base. A servo motor combined with a robotic claw was used to transfer and deploy the new sensor node to its desired location. The claw opens to about 2” and depending on the servo motor used it can pick some relatively heavy objects. A medium sized, high quality servo (S05NF STD) has been used and its main features are:

- **180 degree rotation**
- **Operating Voltage: 4.8V~6.0V**
- **Operating speed: 0.20sec/60 degree (4.8V)**
- **Operating speed: 0.18sec/60 degree (6V)**
- **Output torque: 2.8kg/cm (19.6oz/in) (4.8V)**
- **Output torque: 3.2kg/cm (19.6oz/in) (6V)**

A servo motor is an actuator with a built-in feedback mechanism that responds to a control signal by moving to or holding a position, or by moving at a continuous speed. Two types of servos exist: the standard servo which can only rotate 180 degrees and the continuous rotation servo which is able to rotate all the way around. A servo motor consists of a small dc motor, a gearbox reducing the revolutions and increase output torque and electronics that interpret the input signal and deliver power to the motor. Servo's interface consists of three wires: power, ground, and signal. Power is usually between 4V and 6V and should be separate from system power as servo are electrically noisy and even a small servo can draw up to 1A of current under heavy load.

2.2.9 NPN transistor

A general purpose super bright LED was added to the platform to indicate an interesting event that the user may want to be informed about. The Arduino pins cannot provide enough current (40mA) to drive a Super Flux bright LED, meaning that the LED cannot be controlled simply by setting the output pin HIGH or LOW. To turn on and off the LED on demand an NPN transistor (2N3904) was used acting as a switch. A simple transistor consists of a collector, a Base and an emitter. The current flows from the collector to the emitter and the Base acts like a switch that enables and disables this current flow. Base is separate from the Collector and Emitter meaning that they can be powered separately.

The transistor selection depends on the load's voltage and current draw. The 2N3904 can provide a maximum of 200mA of continuous current flow between Collector and Emitter which is more than enough for our LED. Figure 18 represents the necessary circuit to control an LED using a transistor. A resistor must be added between the Arduino pin and the Base to limit the Base current. The Blue LED operates at 3.0 – 3.2 V according to its datasheet. Also the value of the R2 resistor depends on the LED type.

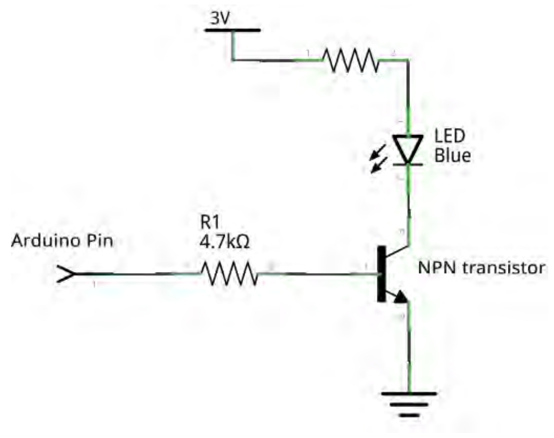


Figure 18 - Controlling an LED using an NPN transistor

The NPN transistor is controlled simply by setting the output pin to HIGH or LOW, causing the LED to turn on and off respectively.

2.2.10 Power supply

The battery chosen as the robot's main power supply is a Lead Acid batter 12V 7A. Lead Acid batteries are made of spongy lead grills with special electrolytes on each plate, and each cell has a 2V potential. They have been available since the 1950s and have become the widely used type of battery today. Lead Acid batteries are the old rusty trucks of batteries, but need almost no maintenance. They are incredibly cheap, rechargeable and easily available. However it must be taken under consideration that Lead Acid batteries are very large and heavy, they always need to be kept charged, and do not have the high discharge rates as the more modern batteries.

The charge and discharge current of a battery is measured in C-rate [12]. A discharge of 1C draws a current equal to the rated capacity. For example, a battery rated at 1A provides 1A for one hour if discharged at 1C. At 2C, the same battery delivers 2A for 30 minutes, and so on. If a battery is discharged more slowly, (let's assume C/10), then we might except that the battery would run longer before becoming discharged. The relationship between battery capacity and discharge current is not linear. Less energy is recovered at faster discharged rates. The battery used is rated at a 20hr rate, meaning that in order to get the fully estimated 7A the battery must be drained to dead over 20 hours. Pulling more amps causes the discharge efficiency to go down. The diagram in figure 19 presents some characteristic discharge curves for a typical Lead Acid Battery.

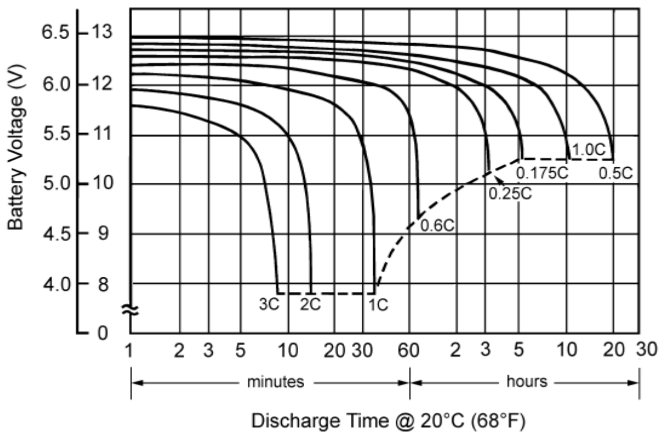


Figure 19 - Acid lead battery discharge curves

It is obvious that some of the robot's components do not operate at the same voltage. Using multiple different rated batteries obviously isn't efficient. Various parts of the robot will stop working at different times and multiple batteries will have to be charged. Two different voltage levels needed in our implementation; 12V for the MD25 board and 5V for the BeagleBoard, the ATmega based platform and all its peripherals. Producing a lower voltage from a higher one can be achieved using a linear voltage regulator or by using a switching regulator. A linear regulator uses a resistive voltage drop to regulate the voltage, losing power in the form of heat. With typical efficiencies of 40%, and reaching as low as 14%, linear voltage regulation generates a lot of waste heat which must be dissipated with bulky and expensive heat sinks.

Switching power supplies can step-up, step-down, and invert. A switching regulator works by taking small chunks of energy, bit by bit, from the input voltage source, and moving them to the output. Two main components are needed in order to accomplish that; an electrical switch and a controller which regulates the rate at which energy is transferred to the output. A switched regulator can typically have 85% efficiency. The energy losses are relatively small, especially when comparing to linear regulators. Also most commonly used linear regulators are not able to provide a big amount of current to the load.

Sparkfun BOB-09370 DC/DC Converter Breakout was chosen for this implementation. This is a breakout board for the Sparkfun 6A DC-to-DC Converter Module. This module is a step-down switch converter able to convert any DC voltage between 4.5-14VDC to a selectable voltage between 0.59 and 5.5VDC. Using a resistor between the Trim and GND pins, the output voltage can be set to the desired level.

Figure 20 - Sparkfun 6A DC/DC converter

The diagram shown in figure 21 presents the converter's efficiency for different values of the output current and input voltages, when the output voltage is 5.0VDC. Using this graph it can be easily observed that efficiency is over 90% leading to a longer battery life.

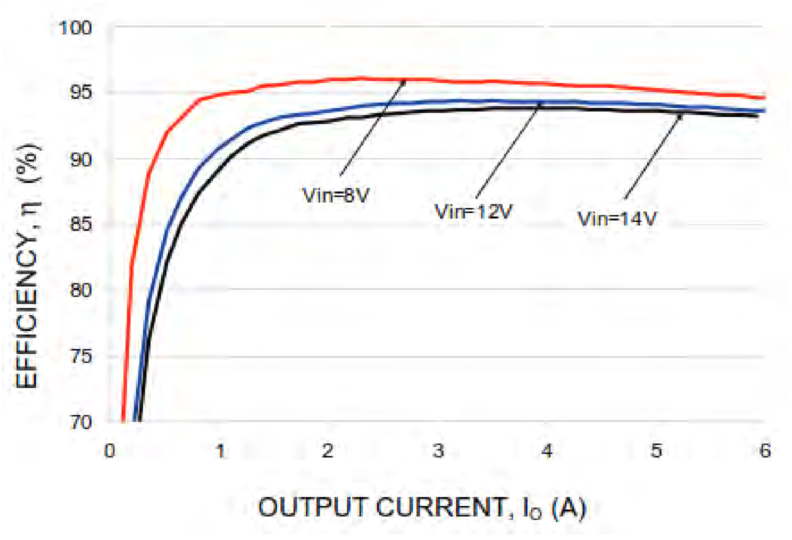


Figure 21 - DC/DC converter efficiency (5VDC output)

An abstract diagram of the system's power supply implementation is illustrated in the next figure.

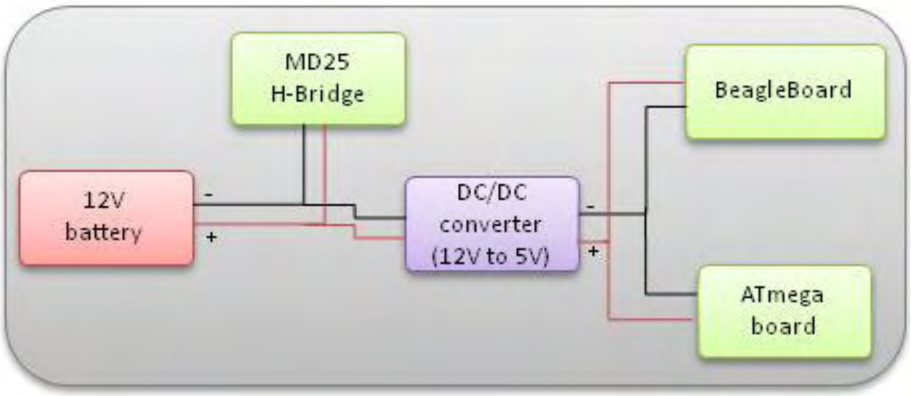


Figure 22 - Robot's power supply

2.3 The ATmega328-based platform

Arduino UNO board is really amazing for prototyping, but sometimes a custom made board may be a better solution. Using the Arduino combined with a solderless breadboard provides a solution for fast prototyping. Breadboards have solderless connections that allow components to be changed quickly without damaging from continuous soldering. They are suitable for prototyping circuits with a few components, and they cannot be used for high voltages and currents. Also a very important problem is that the connections are not as secure as with a prototyping board, especially for a project like this. Solderless connections can suffer from the robots vibrations during its movement.

A perfboard was chosen among other prototyping methods. Perfboards are perfect for quick long lasting prototypes. A perfboard is a thin, rigid sheet with holes pre-drilled at standard interval across a grid, usually a grid of 2.54mm spacing. Many types of these boards exist but the one used consists of an 8x10cm grid with each pad being isolated from the others. Also it is high quality providing pads on both sides (plate-through holes). An example of a perfboard can be seen in figure 23.

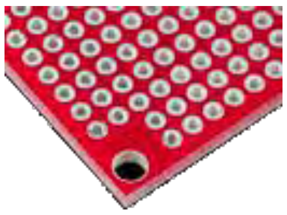


Figure 23 - Perfboard

A custom made board can provide the necessary functionality more reliably. An ATmega328 microcontroller loaded with the Arduino UNO (16MHz) Bootloader allows the developer to use the Arduino code and Arduino IDE in his embedded project, without having to use an actual Arduino. The ATmega328 that was used in this thesis was preloaded with the Arduino UNO Bootloader. In case that the microcontroller does not come pre-loaded with the Bootloader a special procedure must be performed using an Arduino Uno as an in-system program (ISP) combined with the Arduino IDE to burn the Bootloader.

A fully working standalone ATmega328 running at 16 MHz requires the following components:

- A 16 MHz crystal
- Two 16 to 22 pF ceramic capacitors
- A 10K resistor

The components described are not necessary if someone wants the minimal configuration. In such occasion the internal 8 MHz internal clock of the ATmega328 is used.

The ATmega based platform was designed to accommodate most of the components mentioned in the previous section like: the LCD display, the power unit containing the DC/DC step down converter, the GPS module, the Logic Level converter and the necessary I/O pins for connecting the ultrasonic sensors, the MD25 H-Bridge board, the compass, and the servo motor. The Fritzing software was first used to design a digital copy of the board. The next two figures represent the board designed with Fritzing and the final soldered board.

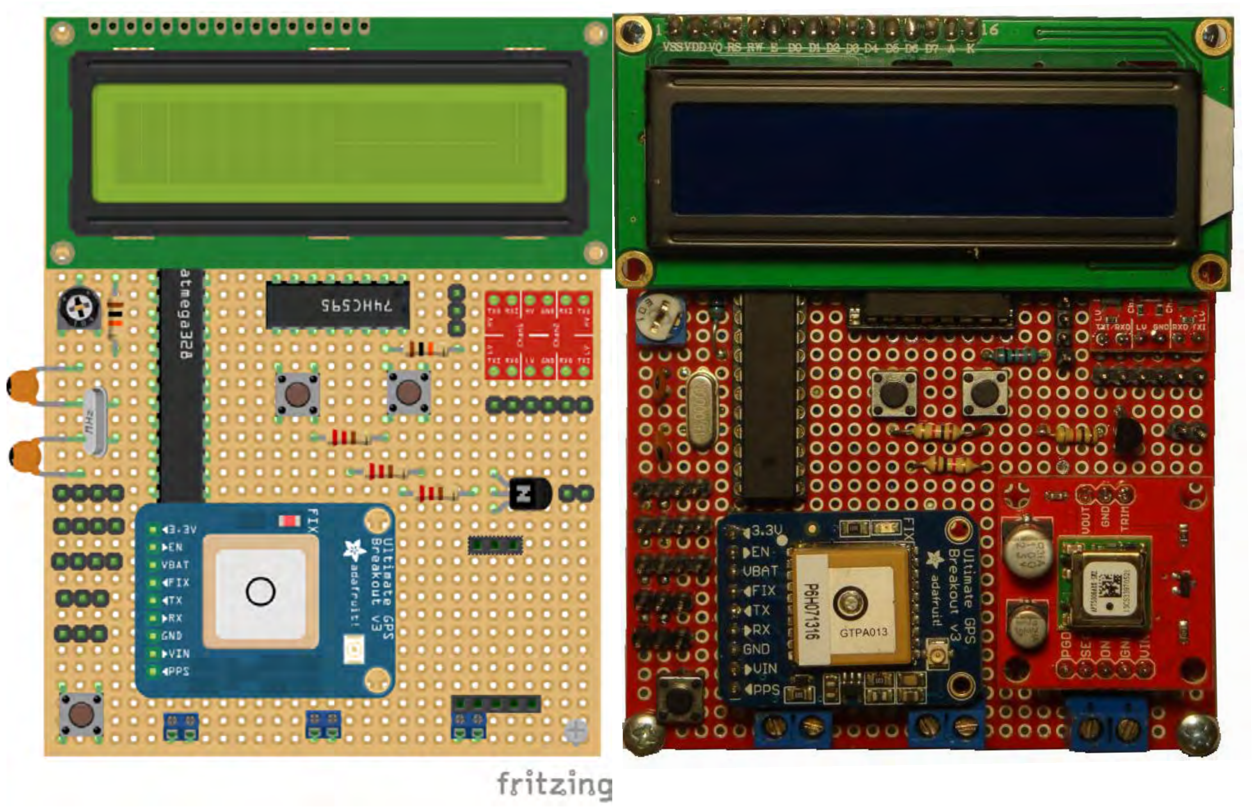


Figure 24 - The ATmega328-based board

The platform’s connections are described in the next figure. The programming port consists of four pins: GND, digital pin1 (Tx), Digital pin0 (Rx), and Reset. These four pins must be connected to the corresponding pins of an FTDI chip or an Arduino Board (without the microcontroller) in order to upload sketches.

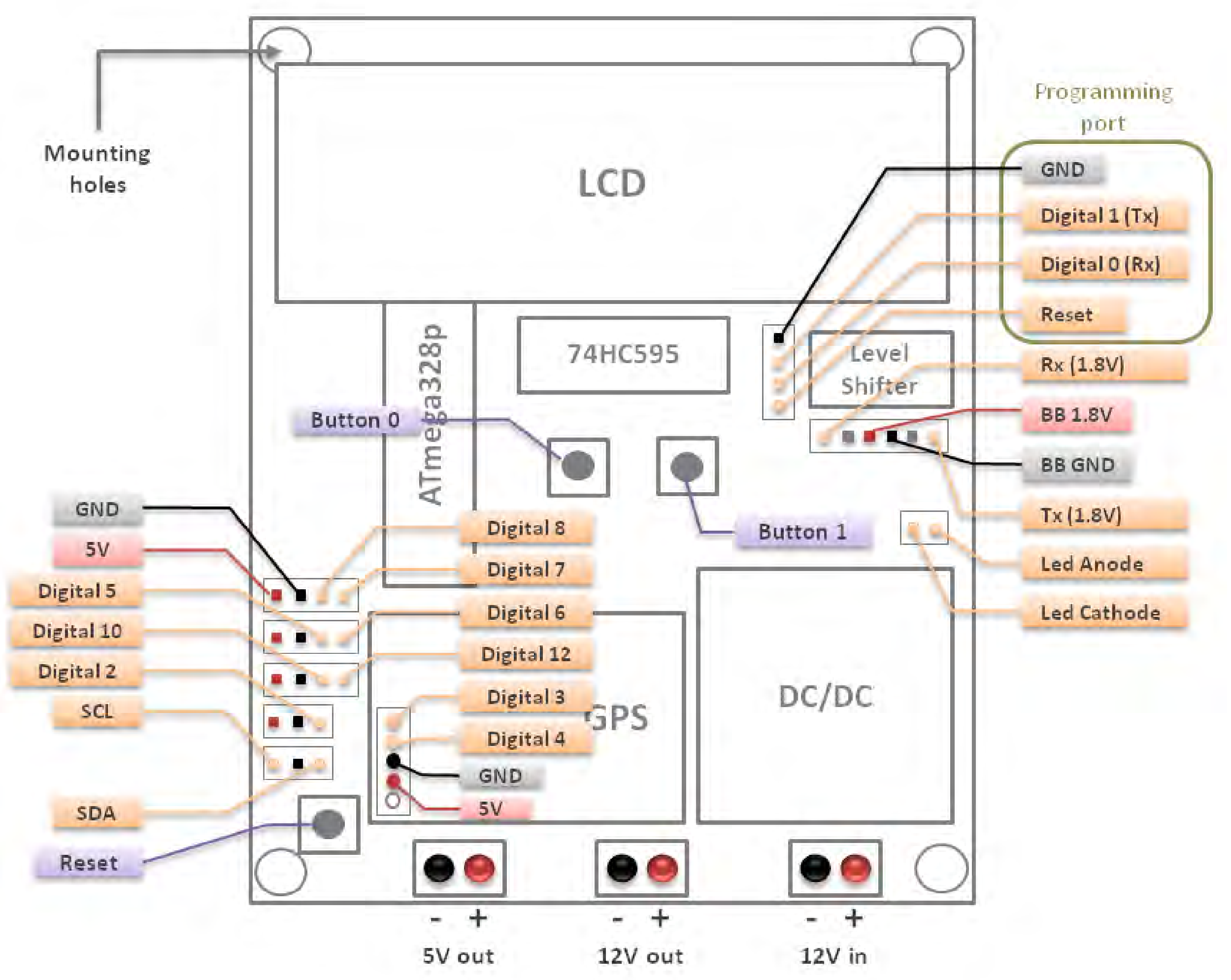


Figure 25 – Atmega328 board pinout

2.4 Streaming video over Wi-Fi

It was also desired to receive live video stream from the robot while operating in the range of a Wi-Fi network. A Logitech C270 web camera was used for this purpose, which is a UVC device. UVC stands for the USB video class that describes devices capable of streaming video. UVC devices do not require a custom driver, instead they use the driver provided by the operating system. Webcam support in Linux is mainly provided by the Linux UVC Project's UVC driver. A UVC device is also responsible for the compression, hence keeping the CPU load low.

The application used for the video streaming is called *MJPG Streamer* [13]. It is a command line application capable of streaming JPEG files over an IP-based network from the web camera to a viewer like Chrome, Firefox, and VLC. It is a light solution especially designed for embedded devices and regular servers. When available it uses the device’s hardware compression keeping the CPU from spending computing power for frame compressing. After successfully compiling the application the user can stream video using the following command:

```
./mjpg_streamer -i "./input_uvc.so -d /dev/video1 -n  -f 10 " -o "./output_http.so -w ./www"
```

The important files are:

- The mjpg_streamer – the application that copies JPGs from an input plugin to one or more output plugins.
- The input_uvc.so plugin which captures JPG frames from a given device (/dev/video0)
- The output_http.so plugin is a fully functional HTTP 1.0 webserver which streams the captured frames according to existing M-JPEG standards.

The user can also configure settings like the frame rate, resolution of the video device, and the HTTP TCP port (default port is 8080). The user is able to view the video stream by pointing any browser on the BeagleBoard IP address like this:

```
http://192.168.1.5:8080/?action=stream
```

2.4.1 Creating an Ad-Hoc network

An Ad-hoc network between the robot and the gateway PC enabled us to stream video from the robot. This is a type of network where stations communicate only peer to peer. This is accomplished using the Independent Basic Service Set (IBSS). The user must make sure that the wireless adapters used, support the IBSS/Ad-hoc mode. To join an IBSS network and assign stationary IPs the following commands should executed as administrator. The other computer only needs to alternate the IP address.

```
$ ifconfig wlan0 down
$ iw wlan0 set type ibss
$ ifconfig wlan0 192.168.0.1 up
$ iw wlan0 ibss join evlampios-adhoc 2412
$ iw wlan0 info
$ iw wlan0 link
```

On the BeagleBoard these commands are organized in a script which is executed during the system’s boot in order to automatically initialize the network. To run the script as administrator, a copy of it must be placed in /etc/init.d. Also it should be marked as executable using the “sudochmod +x /etc/init.d/*script*” command. After that a symbolic link must be created in the run level that the user wants to use (/etc/rc*.d). The symbolic link’s name must signify when the script will be executed by indicating the start with an “S” and then the execution order. It is better to mark the script as the last script to execute (for example S99myscript).

2.5 Robot construction

This section describes the construction of the “E-vlampios” robot platform. The robot’s chassis is based on the platform developed in [7]. This robot was intended for indoor use only, so modifications had to be applied for outdoor deployment. Plexiglas is a great material for building robot bodies. It is a transparent material but in contrast to glass it is shatter resistant to a certain extent. Cutting and drilling Plexiglas must be carefully done as it is easy to crack or melt it. Using almost the same platform used for the indoor robot immediately opposes some disadvantages.

- The robot’s rigid chassis and the lack of any type of suspension mean that there is no way to filter the vibrations when the robot is moving.
- The wheels that come with the RD02 drive system are made from hard plastic offering poor traction in real world situations outdoors.

Some alterations had to be made to increase the ground clearance of the robot, and to accommodate the new sensors and boards. Two platforms were tested: a four-wheeled and a two-wheeled chassis. After experimentation the latter was finally chosen. Four wheels offered greater stability but some other problems occurred; due to the rigid chassis often one or two wheels were not making contact with the ground, making accurate turns was not easy, and the ride was much bumpier. As a result the two-wheeled version was finally used combined with a free rotating wheel on the opposite side of the robot to maintain equilibrium.

The robot’s frame consists of two main Plexiglas sheets connected together using four 6mm screws. The upper Plexiglas sheet measures 3mm in height while the lower 5mm to support the weight for efficiently. The lower accommodates the battery, the motors, the MD25 board, and the robotic claw while the other components lay on the upper sheet. The next figure shows a high level diagram of the robot’s main parts. Lack of traction was an issue that had to be solved. Since no other set of wheels designed for outdoor use was not available, it was decided to cover the wheels with a soft rubber containing some kind of tread. This solution worked better than excpetced.

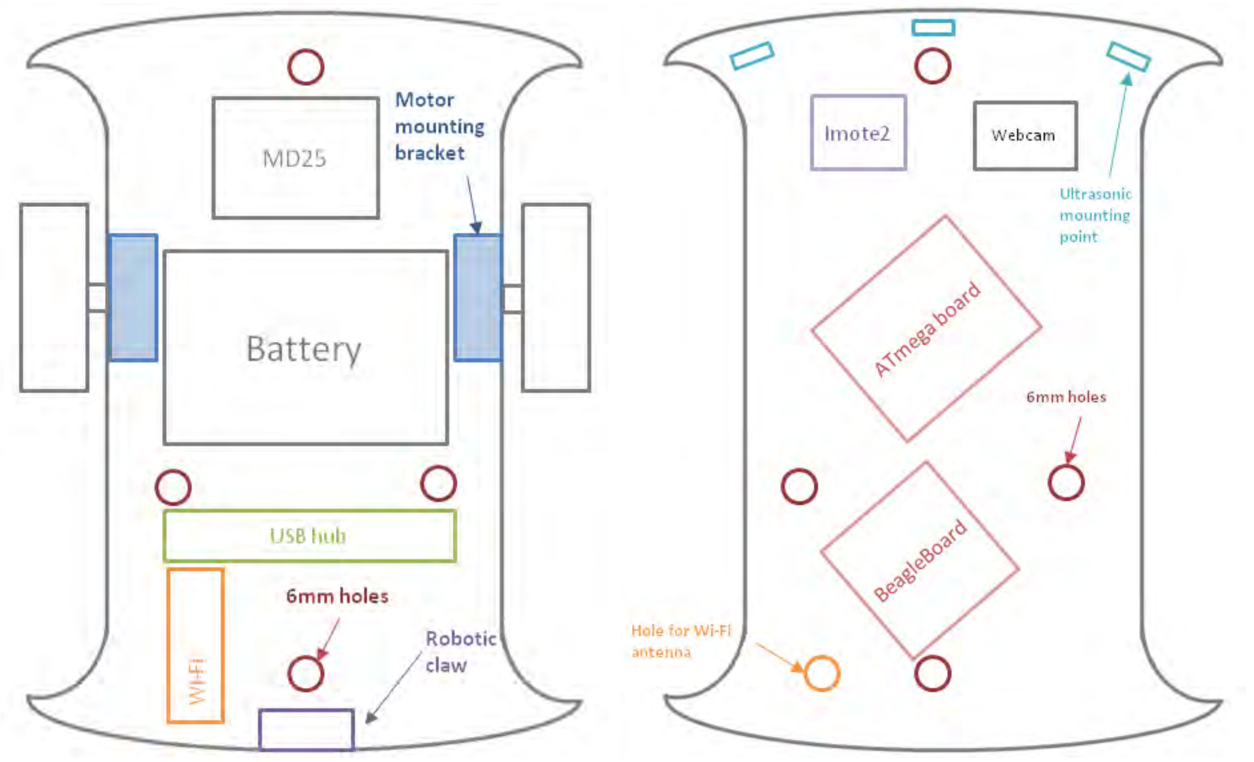


Figure 26 - Robot's lower and upper Plexiglas sheets

The fully assembled “E-vlampios” robot can be seen in the following figure.

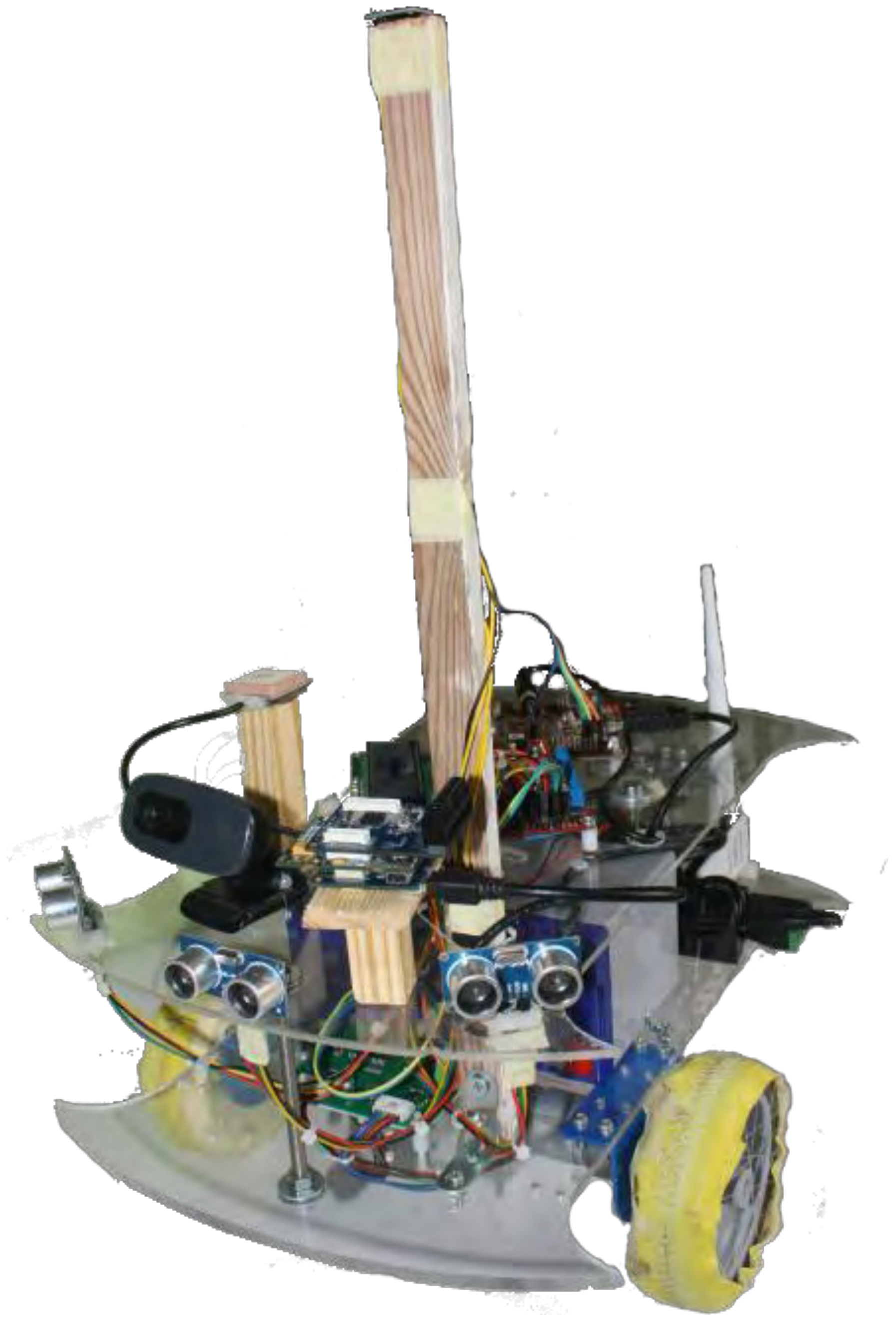


Figure 27 - The "E-vlampios" robot

2.6 Robot Kinematics

The two separately controlled wheels are placed on either side of the robot body making it a *differential wheeled robot*. It can thus change direction by varying the relative speed of its wheels and hence does not need an additional steering motion. This is the most common control mechanism. The following fundamental cases can happen in a differential wheeled robot:

- When the wheels are driven at the same velocity and the same direction the robot moves forwards or backwards according to the direction of rotation (clockwise or counterclockwise).
- When the wheels have the same angular velocities but rotate in opposite directions the robot tends to spin around its vertical axis making a zero radius turn.
- Finally the robot can make a curve motion if angular velocities are different.

These scenarios are represented in figure 28 bellow. Perhaps the most serious disadvantage of this method is that the robot does not always drive as expected, especially when DC motors are used which are not very precise. For example the robot may not always drive along a straight line nor turn exactly. For this reason the movement of the robot must be constantly monitored and adjusted.

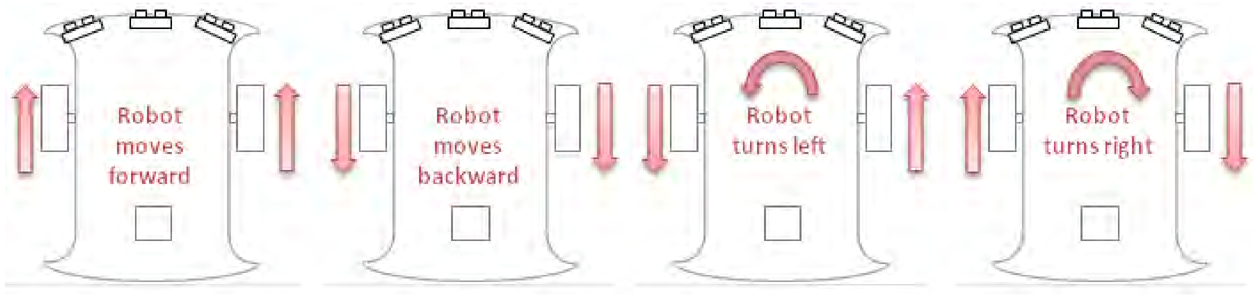


Figure 28 - Differential drive fundamental cases

3 Navigation

3.1 Distance and bearing between two points on the globe.

Calculating the distance and bearing between two points on the globe is essential for GPS navigation. The Haversine formula [14] is an equation very important in navigation, giving great circle distances between two points on a sphere from their longitudes and latitudes. The great-circle distance is the shortest distance between two points on the surface of a sphere. The distance between two points in Euclidean space is the length of a straight line between them, but on a sphere there are no straight lines. In non-Euclidean geometry, straight lines are replaced with Geodesics. Geodesics on the sphere are the great circles [15]. Through any two points on a sphere which are not directly opposite each other, there is a unique great circle the two points separate the great circle onto two arcs. The length of the sorter arc is the great-circle distance between the points.

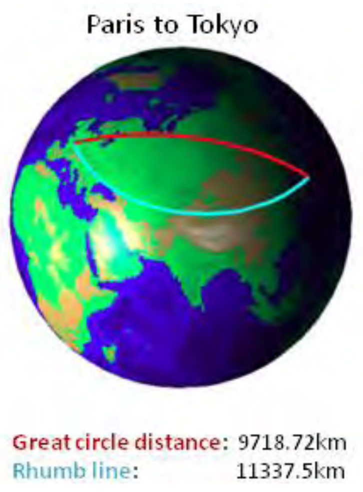


Figure 29 - Great circle vs. Rhumb line distances

If the distance between the two points is less than about 20km then the Pythagorean Theorem may be used, resulting to an error of 9 to 30 meters. On the other hand the Haversine formula will give mathematically and computationally exact results. The Earth is nearly spherical so great circle provides a distance correct to within 0.5% or so [16]. The Haversine formula remains particularly well-conditioned for numerical computation even at small distances. Following a great-circle path, causes current heading to vary. The distance between two points can also be computed using the Spherical Law of Cosines which is a simpler alternative. This method gives well-conditioned results down to distances as small as around 1 meter.

The function described in the next figure implements in C language the Haversine method used in this thesis, which computes the distance between two waypoints. Note that degrees must be converted to radians.

```
double distance_haversine(double lat1, double lon1, double lat2, double lon2)
{
    int earth_radius = 6371;

    double dlat = degrees_to_rad(lat2 - lat1);
    double dlon = degrees_to_rad(lon2 - lon1);
    lat1 = degrees_to_rad(lat1);
    lat2 = degrees_to_rad)lat2);

    double a = sin(dlat/2) * sin(dlat/2) + sin(dlon/2) * sin(dlon/2) *
               cos(lat1) * cos(lat2);

    double c = 2 * atan2(sqrt(a), sqrt(1-a));

    return c * earth_radius;
}
```

Figure 30 - Distance between two points using the Haversine method

Figure 31 describes a typical scenario when the robot is located at point A and its goal is to reach point B. In order to do this the robot must first calculate the bearing between its current location and point B (angle ϑ). By retrieving its current heading (CH in figure) value from the compass, the robot can calculate the necessary turning angle in order to face at point B.

$$turn = \varphi - \theta$$

The final heading will differ from the initial heading by varying degrees according to distance and latitude. The formula described below is for the initial bearing which if followed in a straight line along a great-circle arc will lead from the start point to the end point.

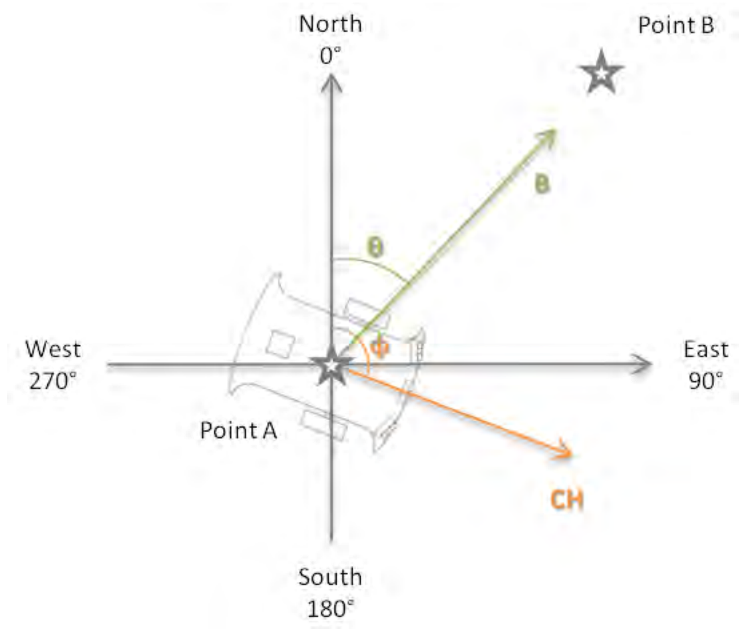


Figure 31 - Typical navigation scenario

```
double calculate_bearing (double lat1, double long1, double lat2, double long2) {

    double dlon = degrees_to_rad(long2-long1);
    lat1 = degrees_to_rad(lat1);
    lat2 = degrees_to_rad(lat2);
    double a1 = sin(dlon) * cos(lat2);
    double a2 = sin(lat1) * cos(lat2) * cos(dlon);
    a2 = cos(lat1) * sin(lat2) - a2;
    a2 = atan2(a1, a2);

    if (a2 < 0.0) {
        a2 += (2*M_PI);         }

    return (double)(a2 *180)/(double)M_PI;
}
```

Figure 32 - Bearing between two points using the Haversine method

The *navigate_to_waypoint* function is the heart of the navigation algorithm. The robot always:

- Updates the navigation data (current location, distance from objects, and heading).
- Periodically checks the bearing between the current location and the desired waypoint and makes corrections if needed.
- Periodically updates the logger with its current position.

This function follows the flowchart presented in the next page.

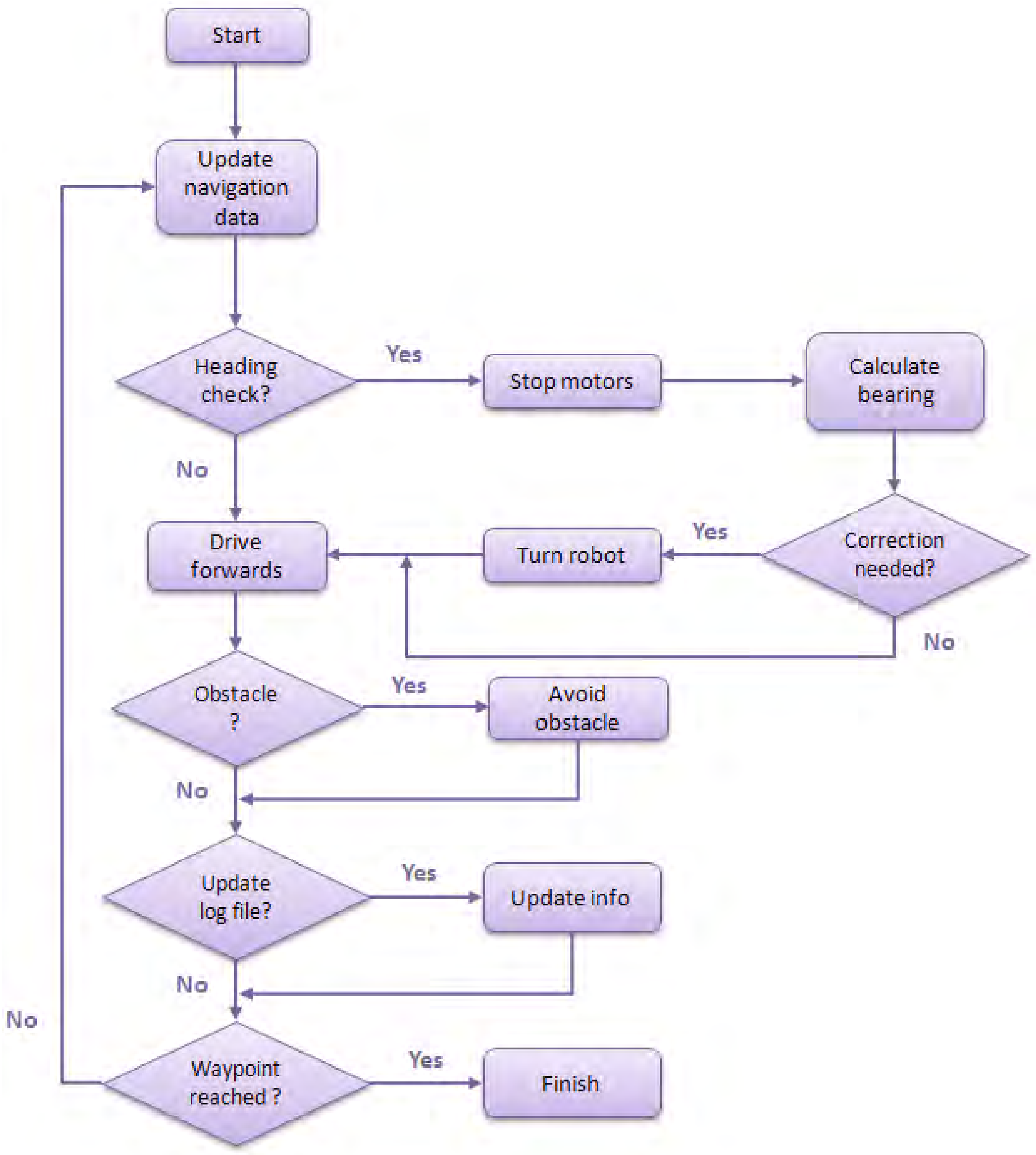


Figure 33 - *navigate_to_waypoint* function flowchart

3.2 Acquiring navigation information

In order to update the navigation data, the robot constantly monitors the serial communication between the BeagleBoard and the ATmega328 microcontroller. The ATmega328 is responsible for gathering information about the GPS coordinates, distances from objects using the ultrasonic sensors and heading from the digital compass. All this information is passed to the BeagleBoard as a sentence which has to be parsed by the BeagleBoard's software hence updating the navigation data. Character "*" has been used as a special character, indicating the start of a new sentence. An example of this sentence and its main fields can be seen below.

Special character	Latitude	Longitude	Satellites	Ultrasonic 1 (cm)	Ultrasonic 2 (cm)	Ultrasonic 3 (cm)	Heading (°)
*	39.364757	22.951415	8	50	0	0	275

3.3 GPS data logger

KML is an XML based file format for displaying data in an Earth browser like Google Earth or Google Maps [17]. A KML file can be generated within Google Earth itself, by handcrafting KML files or by programs that generate well formatted KML document. There are five basic KML documents that can be automatically created by Google Earth: Placemarks, ground overlays, paths and polygons. This can be altered through Google Earth's preferences.

The most useful KML document for the purposes of this thesis is the KML that describes a path. Together with the navigation software a GPS logger has been developed, monitoring the robots movement. It is capable of creating KML files that represent the path followed by the robot on the Google Earth. Some further details about the KML file created by the GPS logger are presented below. Each time the robot starts a new mission, a new KML file is generated with its own unique name based on the system's time.

4 Wireless Sensor Network

4.1 Mote-PC communication

The user needs to read data out of a TinyOS network. The most common approach is to attach a node to a PC or laptop with a wired connection. The wireless node generally talks to a serial port. Most wireless sensor nodes provide serial port or similar interfaces so that they can talk to the serial port in a computer directly. The basic abstraction for mote-PC communication is a packet source which is a communication medium over which an application receives and sends packets to the mote. Some examples of packet sources are serial ports, TCP sockets, and the SerialForwarder tool. The PC code used for this purpose is written in Java, using the Java tool chain distributed with TinyOS.

Although it is possible to write code for reading and writing the raw bytes in packets, the TinyOS uses necS's external types to specify packet layouts, and named constants to specify specific values [18] . An external type is the same as a normal type except that it has nx_ or nx_le preceding it. The mig tool (for "message interface generator") distributed with TinyOS, generates code to encode and decode a byte array whose layout is specified by a nesC external type. The TinyOS serial communication stack deals with the exchange of packets over a serial connection. Like radio packets these are "active message" packets. The layout of serial AM packets is defined using external types:

```
typedef nx_struct serial_header {
    nx_uint16_t dest;
    nx_uint16_t src;
    nx_uint8_t length;
    nx_uint8_t group;
    nx_uint8_t type;
} serial_header_t;

typedef nx_struct serial_packet {
    serial_header_t header;
    nx_uint8_t data[];
} serial_packet_t;
```

The application developed in Java for PC – node communication uses simple external types to specify its message layout:

```
typedef nx_struct serial_msg {
    nx_uint16_t msg_id;
    nx_uint16_t period;
    nx_uint16_t data;
    nx_uint8_t sender;
    nx_uint8_t predecessor;
    nx_uint8_t hop_level;
}serial_msg_t;
```

The resulted connection between serial_packet_t and serial_msg_t is shown in the following figure.

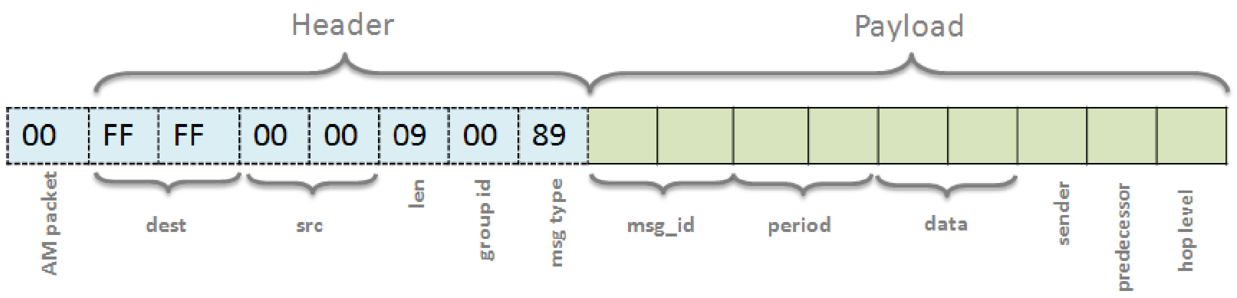


Figure 34 - Imote2 Serial message format

The mig tool generates a class that represents a value of a given nesC nx_struct (for example the serial_msg_t above). This class provides methods to read, write, and report properties of the fields of M. The net.tinyos.message package provides a class MoteIF (for “mote interface”) that makes it simple to transmit and receive messages specified by a mig-generated class. Before using MoteIF, you need to open a connection to your base station mote. This is done by calling the net.tinyos.packet.BuildSource.makePhoenix method, which takes as argument a packet source name. This packet source name specifies how you reach the base station mote (e.g., through a serial port) and all the relevant parameters (e.g., device name, baud rate, etc). In the rest of this section, we will use "serial@COM1:intelmote2" as our packet source, which denotes serial communication over serial port COM1: at the normal Imote2 mote baud rate (115200 baud).

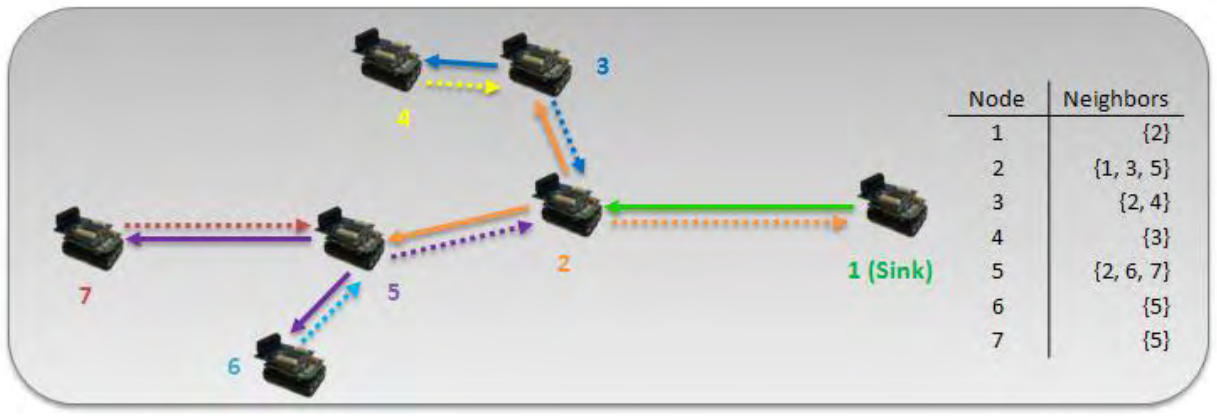
Using MoteIF and mig, the code to transmit serial_msg_t messages is quite simple. Transmission of consecutive packets is performed using MoteIF’s send method, which takes an object of a mig-generated class as argument. Receiving a message is done by declaring a messageReceived method, and registering it as the handler for test serial_msg_t.

4.2 Sensing application

Each node of the WSN runs a sensing application implemented in nesC, which measures the environment's light density. The sensing application on each node starts when the Gateway broadcasts a so-called *query* message to the network, containing among other information the sensing period and a unique ID. This message is forwarded through the sink to all nodes via flooding. Flooding is one of the simplest routing algorithms in which every node acts as both a receiver and a transmitter, retransmitting any received packet. Whenever a new query message is broadcasted to the WSN the communication links among the nodes are re-constructed and the sensing period is defined is set according to the corresponding field of the query message. The definition of the query message can be seen in figure 35.

The basic problem with the flooding algorithm is that the number of packets in circulation just for a single source packet quickly grows without bound. To address this problem, each node remembers the identity of the retransmitted packets. When a copy of the packet arrives back, it is simply discarded. Also measures have been taken to avoid simultaneous retransmissions (collisions) of packets from the nodes.

The next figure represents an illustration of a hypothetical WSN and the communication links between the seven nodes. The sink node broadcasts a new query message (green arrow). Only node 2 receives that message and re-broadcasts it. Node 1 receives again the same message and simply discards it (orange dotted arrow). The same pattern continues until every node receives the query message. Each node sets as its parent the node from which it received the new query for the first time.



Node	Neighbors
1	{2}
2	{1, 3, 5}
3	{2, 4}
4	{3}
5	{2, 6, 7}
6	{5}
7	{5}

Another type of message has been defined for this application. A result message is used for reporting the measurements back to the sink, containing also other useful information for the network. The definition of this message can also be seen in figure 35. The predecessor and hops fields in the result message are used by the Gateway to visualize the WSN. The group field is used allows distinct groups of nodes to share the same radio channel. Group 1 is used for the sensing application whereas group 2 is used for the gateway-robot communication.

```
typedef nx_struct queryMsg{

    nx_uint16_t sensing_period;
    nx_uint16_t msg_id;
    nx_uint16_t sender;
    nx_uint8_t group;

}query_msg;

typedef nx_struct resultMsg{

    nx_uint16_t data;
    nx_uint16_t from;
    nx_uint16_t to;
    nx_uint16_t predecessor;
    nx_uint8_t hops;
    nx_uint8_t group;

}result_msg;
```

Figure 35 – Definitions of query and result messages

A sensor can read the light density using TinyOS's components that provide one or more split-phase Read interfaces. For example:

```
Interface Read<val_t> {
        Command error_t read();
        Event void readDone(error_t result, val_t val);
}
```

When a node receives a new query message it toggles the Imote2's red LED, starts the Timer according to the value of the *sensing_period* field in the query_msg struct, and then forwards the message to the other nodes (Figure 36). Also it keeps the query's ID and assigns the sender as its parent. Receiving a query message with a different ID causes the re-construction of the communication tree, and the nodes update their data about their parents.

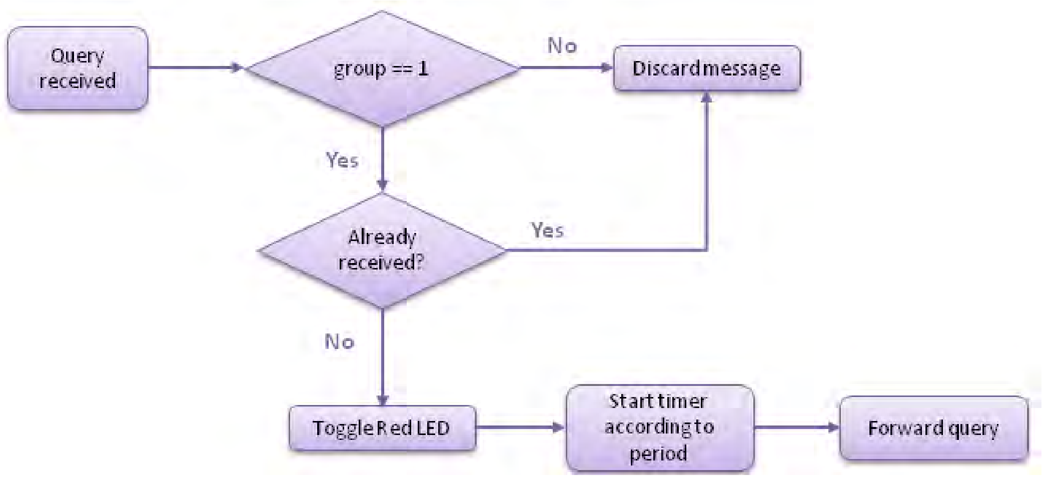
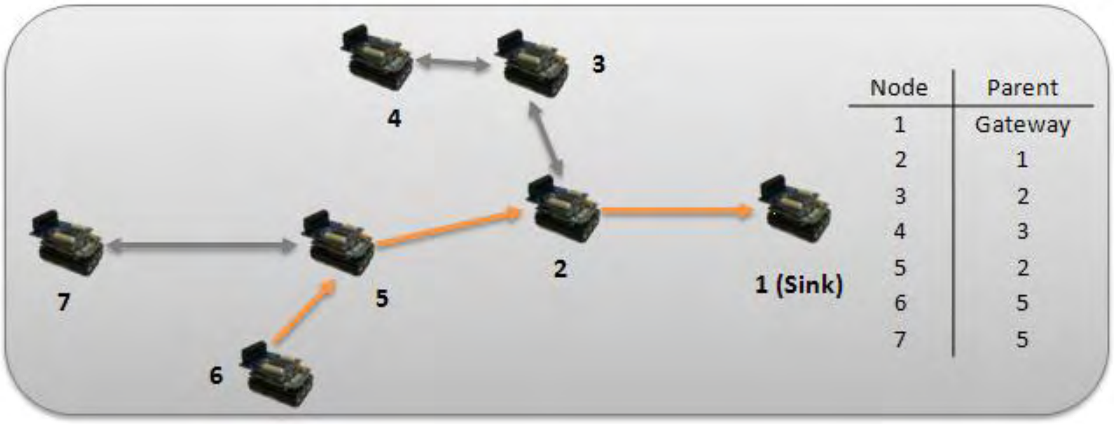


Figure 36 - Receiving a new query

When a result message is received, the node checks if it is the receiver of the message and then forwards it to its own parent (Figure 37). In our example, a light measurement from node 6 reaches the sink through nodes 5 and 2.



Node	Parent
1	Gateway
2	1
3	2
4	3
5	2
6	5
7	5

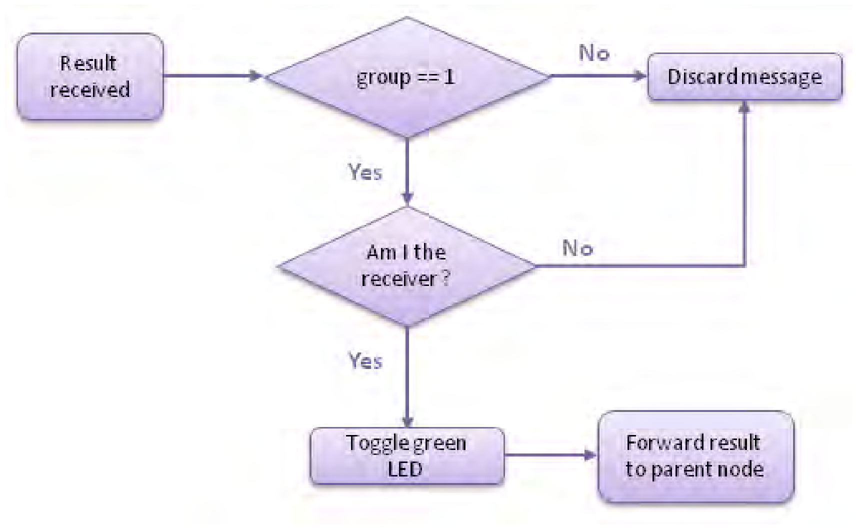


Figure 37 - Receiving a result message

4.3 Failed node discovery

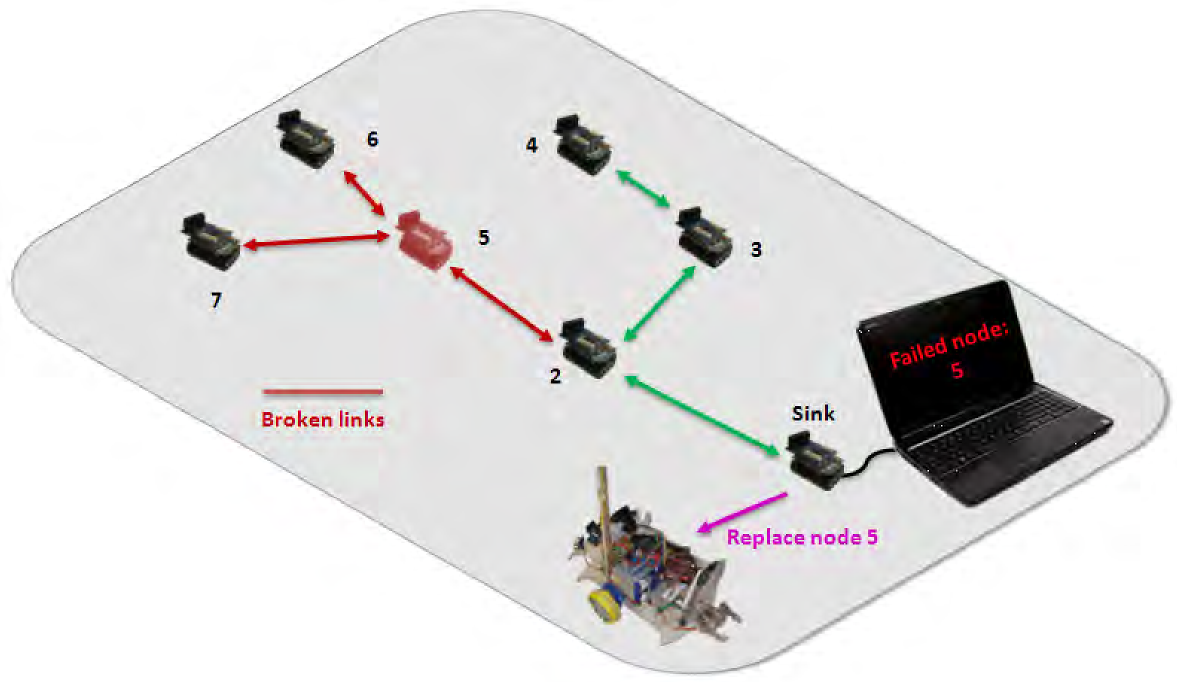
A Java application was implemented during this thesis for the discovery of a failed node that makes use of the Java Universal Network/Graph Framework (JUNG). JUNG is a free, open-source software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network [19]. It is written in Java, which allows JUNG-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries.

Usually a network consists of entities and known relationships among them. The JUNG library is designed to support these relationships in multiple forms: directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs. JUNG includes implementations of algorithms from graph theory, data mining, and social network analysis [20]. JUNG also provides a visualization framework which user can use to create custom layouts.

Graphs consist of a set of vertices V and a set of edges, E. Vertices represent entities (for example network's wireless nodes), and edges represent relationships among them (communication links). A graph is a container of vertices and edges, with several methods for accessing and modifying these sets, for defining vertex and edge constraints, and for specifying listeners. For our purposes a Directed sparse graph has been used. The edges represent the spreading of the query message from the sink to the other nodes of the WSN. The Gateway uses the information contained in the result messages to construct a graph representing the WSN.

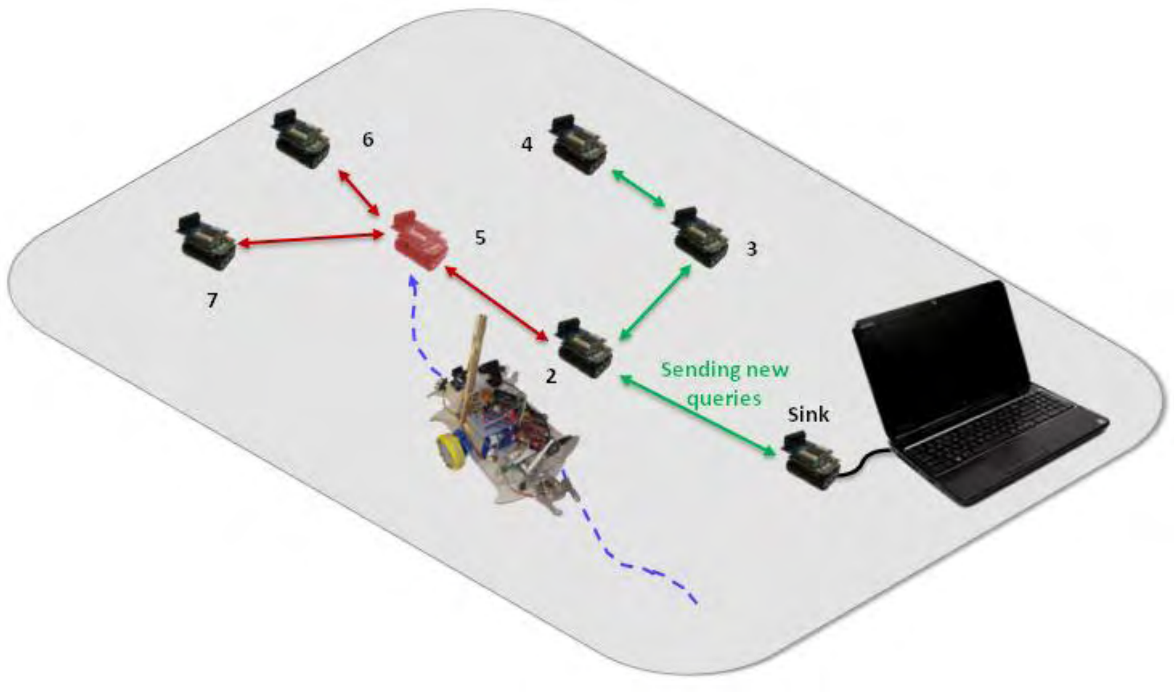
Discovery of a failed node is based on timestamps. For every node in the network the Gateway keeps among other information a timestamp of the last received measurement of the node. Sometimes a message from a node may be lost due to interference between the sensors. So declaring a node as faulty must happen after a predefined number of sensing periods. During the application's execution the Gateway always checks the status of the timestamps for all nodes of the network. The application conducts a Breadth-first search of the graph, examining every node on a level before going to a lower level. If a predefined number of sensing periods have been passed without receiving a measurement from a node, the gateway declares this node as faulty, and starts the process of node replacement. Complementary to the query and result messages a new message was defined for the robot – Gateway interaction.

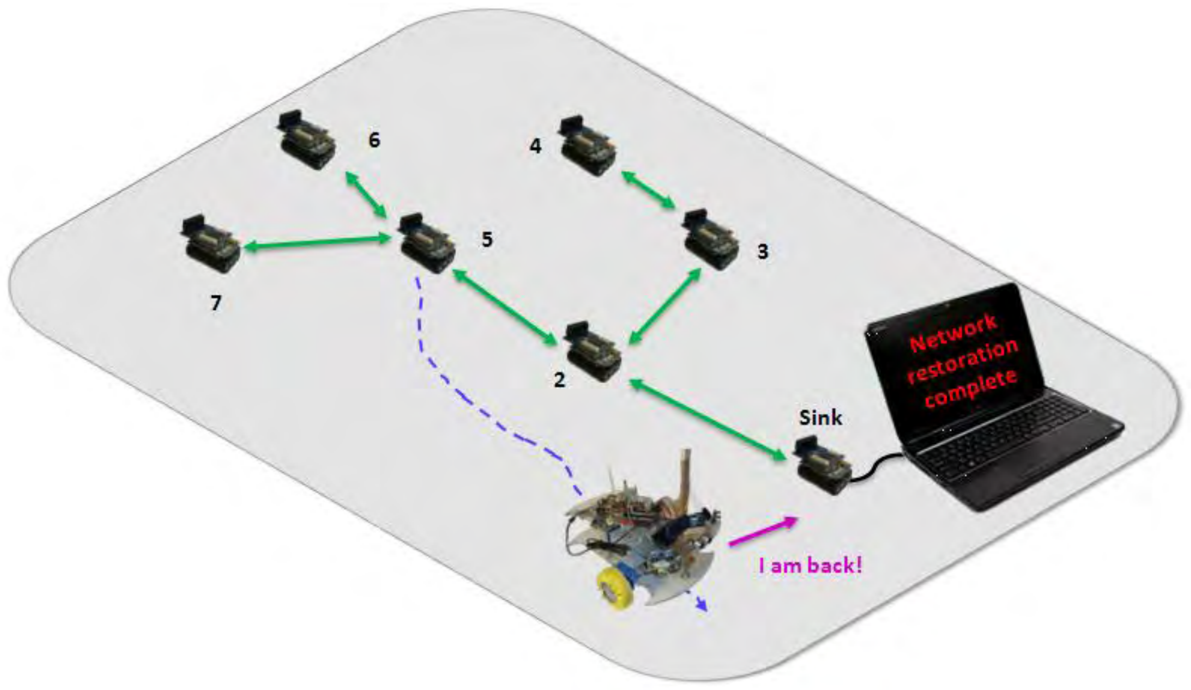
For the example WSN mentioned earlier, the Gateway constructs a graph representing the network. Let us assume that node 5 stops operating after some time. Nodes 6 and 7 will keep sending their light measurements but due to the missed communication links, these messages will never reach the gateway (node 5 is part of the critical path to the gateway). The timestamps of these nodes will also be old making these nodes suitable for declaring as faulty, but the Breadth-first search of the graph will first discover node 5. After the discovery of a failed node the Gateway informs the robot about the presence of a faulty node that needs to be replaced. The robot is informed though its Imote2 node.



After the Gateway informs the robot, it starts a round of broadcasting new query messages. It periodically creates new queries each one with a unique ID and broadcasts them to the network. As mentioned earlier, whenever a query message with a new ID is forwarded to the network the communication links are created again. This procedure will keep going on until the gateway receives data from all nodes. At the meantime the robot travels to the desired location, and once it deploys the new node the next query will complete the network restoration.

In our example while the robots travels to the desired location, the queries periodically sent from the Gateway will only reach nodes 2, 3, and 4. The Gateway will receive results only from these nodes meaning that the network restoration is not completed yet. The query from the Gateway that follows the deployment of the new node will result in a fully working WSN. When the new node has been deployed the robot travels back to the gateway location. When the robot reaches that location, it broadcasts a message whose receiver is the Sink, declaring that the robot is available for a future node replacement.





The Java application responsible for the faulty node detection was further extended in order to create a useful user interface. The main window of the application makes use of the JSplitPane and uses two JPanels; one for displaying the sensor light measurements and the other for representing the network. Visualization of the WSN was possible using the JUNG's great visualization capabilities. Some new features were also added, like:

- The ability to represent a node using an actual image
- Paint vertices, edges and arrows according to the status of the communication links.
- A mouse was also added to the visualization viewer allowing picking, translating, and zooming.

Examples of the final GUI can be seen on chapter 5.

5 Testing and Results

The performance of the "E-vlampios" robot was evaluated in different circumstances. In this section, results based on different tests will be described. Two main categories of tests have been conducted. During the first test we wanted to evaluate the robots movement precision based on the GPS receiver. The robot should be able to follow a predefined path and reach a certain number of waypoints. The latter was a complete test of our main purpose, involving the robot and a deployed Wireless Sensor Network. In both cases the robot runs continuously without external help or manual control. During some of the tests the BeagleBoard's serial port was used connected to a laptop. This allowed me to change configuration settings and also helped the debugging process.

The tests were conducted in a wide area near to the Panthessaliko stadium in the city of Volos. This place was chosen for various reasons. First of all it is large open area with a very good view of sky without tall buildings making it ideal for our GPS guided robot. Also the ground is almost flat, and few people and animals are around.

5.1 Mission planning

The sensor nodes are represented as waypoints in latitude, longitude decimal degrees format. The sensor nodes coordinates are stored in a constant array. These values can change according to the mission requirements. An example of five sensor nodes is shown in the next code sample.

```
typedef struct{
    double latitude;
    double longitude;
}coordinates;

coordinates nodes[]=
{
    {39.385067, 22.929438},
    {39.385216, 22.929326},
    {39.385381, 22.929196},
    {39.385465, 22.929404},
    {39.385288, 22.929528}
};
```

5.2 Following a pre-defined path

In this test, the robot starts from a random position and follows a desired path according to a pre-defined set of coordinates describing the waypoints. The coordinates of the waypoints were entered in the form described above, and the Robot was commanded to proceed towards those targets in sequence. Two paths used for the experimentation purposes. The first run consists of five waypoints named as wp_0 to wp_4. The distances and the bearing between the waypoints are given in the next table.

Table 2 – First route waypoints (Distances and Bearings)

Waypoints	Distance (m)	Bearing
wp_0, wp_1	18.95	322°
wp_1,wp_2	21.79	328°
wp_2,wp_3	19.73	61°
wp_3,wp_4	25	144°

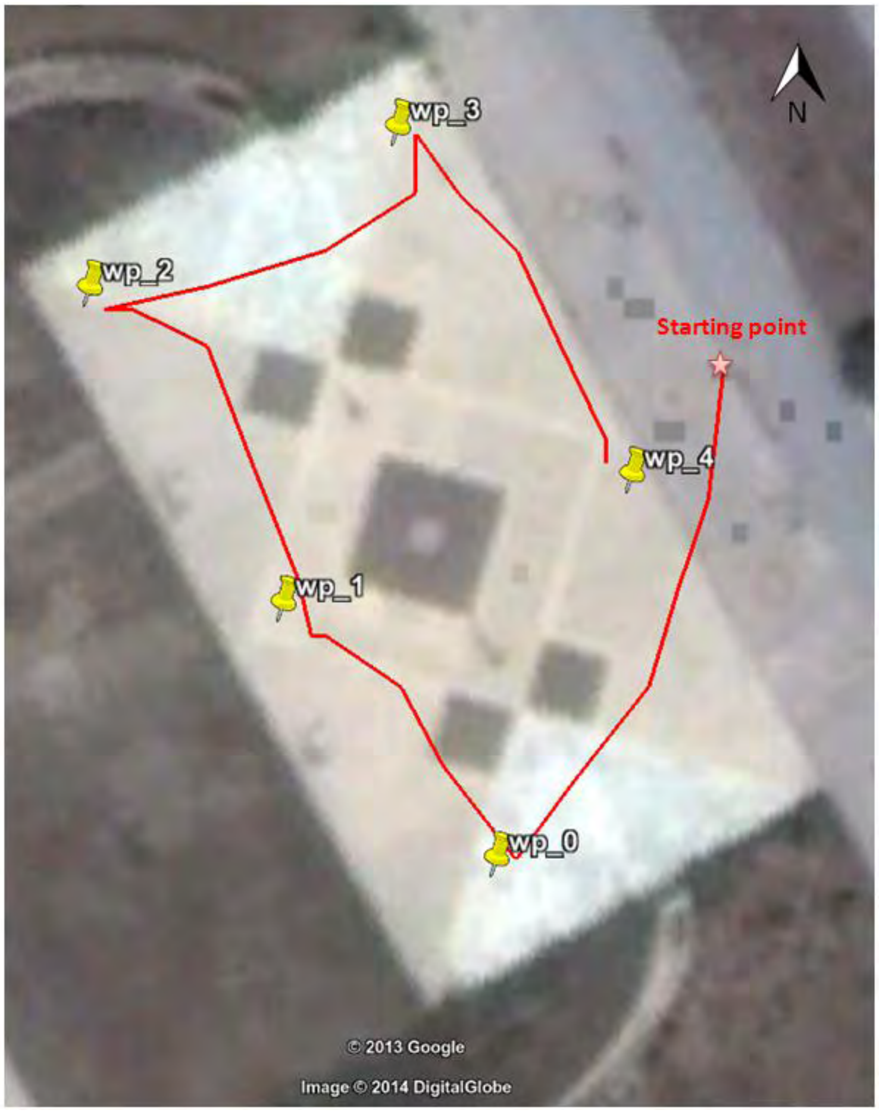


Figure 38 - First route result path

Distance travelled: 118 m

During the tests a radius in meters around the waypoints was defined. This constant is necessary in order to avoid the robot circling endlessly around a waypoint. When the robot is in a pre-defined perimeter of the actual point, a switch to the next waypoint is activated. During the path-following tests, this constant was set to 2 meters. Figure 38 shows the path followed by the robot trying to reach the five waypoints that form the first test trajectory. During this test the robot was able to reach all the nodes in the desired radius. All the distances from the waypoints were less than 2 meters. The total distance travelled by the robot was 118 meters.

The second route consisted of six waypoints. The distances and bearings among them are listed in Table 3 . In the next two figures we present the results from two different runs by the robot. Again the robot reached all the waypoints in the desired distances. The total distances travelled were 110.9 and 105 meters respectively.

Table 3 - Second Route waypoints (distances and bearings)

Waypoints	Distance (m)	Bearing
wp_0, wp_1	15.20	317°
wp_1,wp_2	14.90	51°
wp_2,wp_3	15.94	326°
wp_3,wp_4	14.89	240°
wp_4,wp_5	33	142°

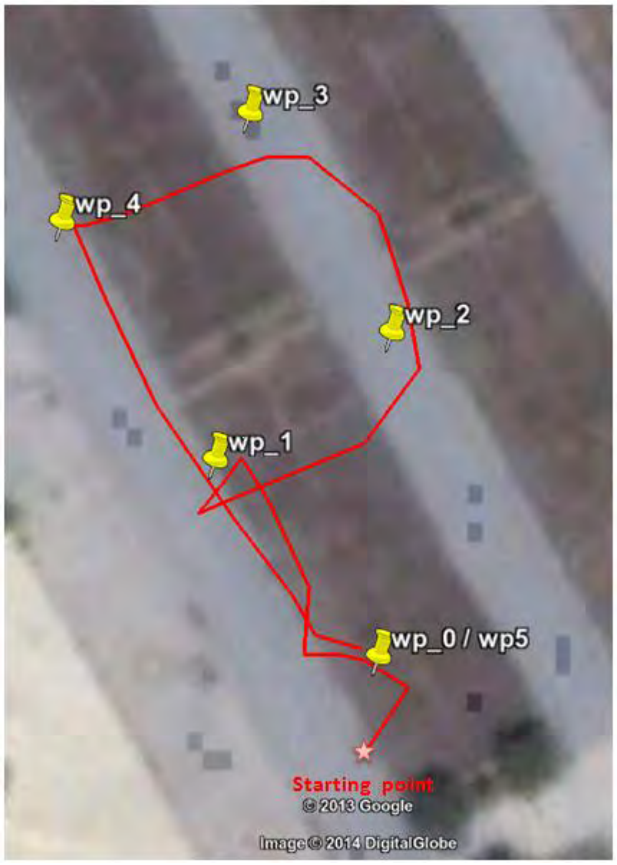


Figure 39 - Second Route, first run.

Distance travelled: 110.9 m.

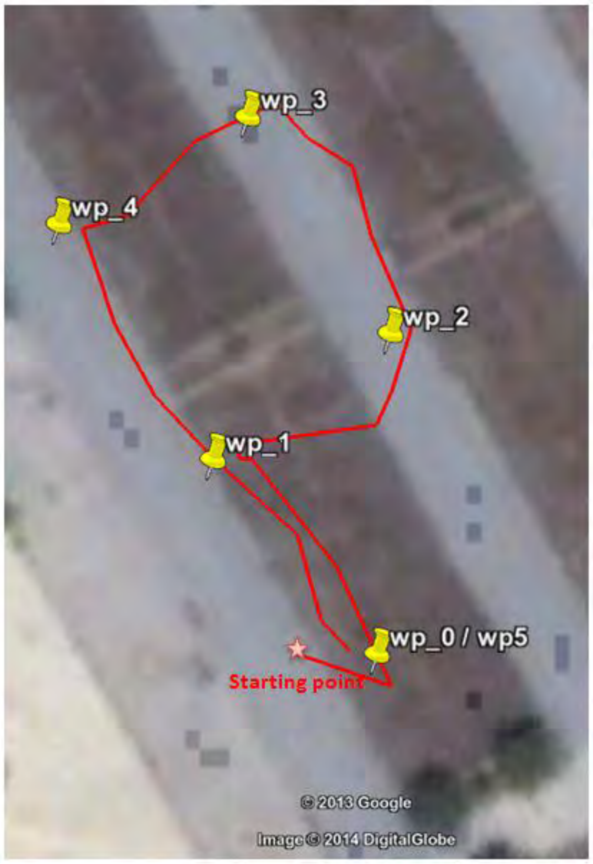


Figure 40 - Second Route, second run.

Distance travelled: 105 m.

In the second test the robot worked together with a deployed Wireless Sensor Network to perform a complete test. The gateway should identify any node failure and send the robot to the desired location for node replacement. Special wooden stands for the nodes were also constructed to install the sensor nodes. The robot and the nodes with their stands used for the test are shown in the following figure.

The sink node was attached to a laptop (the gateway) running the Java application described in the previous chapter, which identifies a node failure. In such an occasion the robot is informed from the gateway to initiate the node replacement process. The nodes were placed in the positions shown in figure 42.

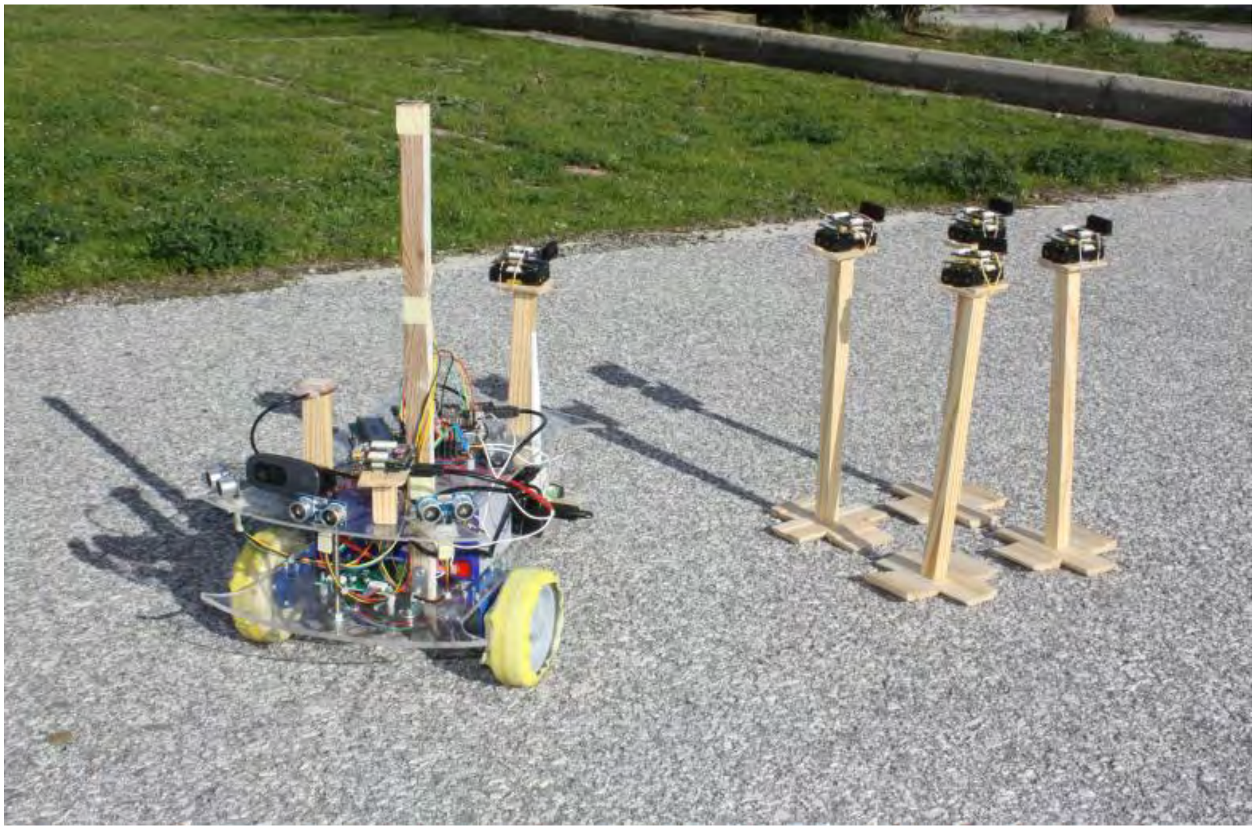


Figure 41 - "E-vlampios" and Imote2 sensor nodes

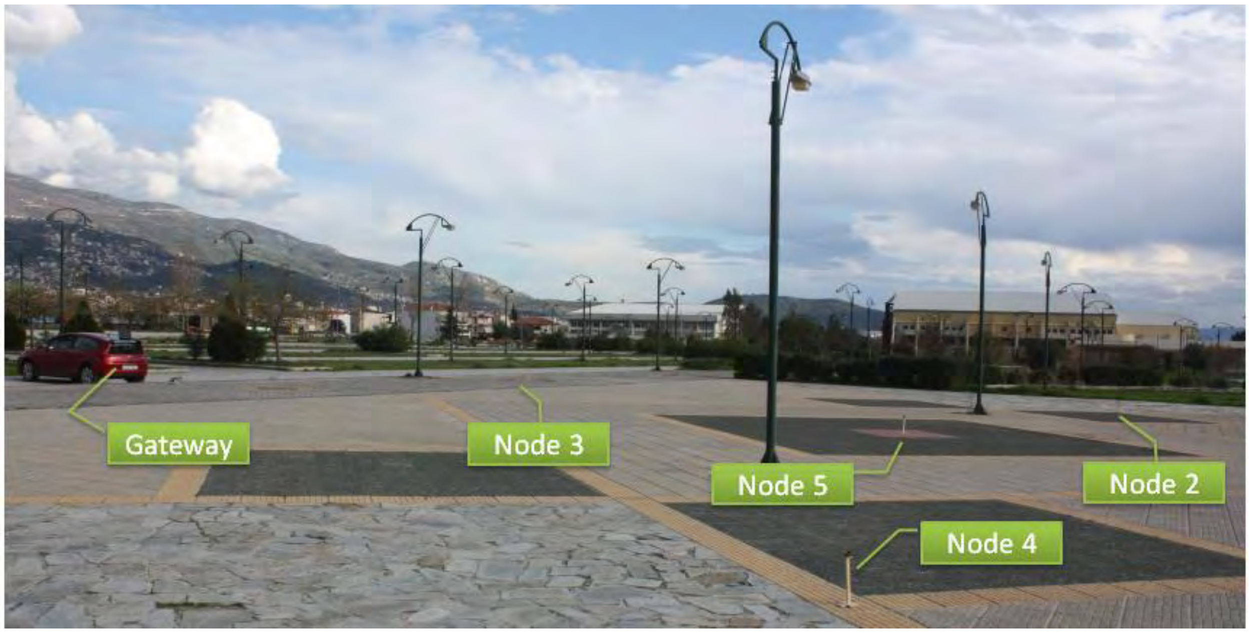
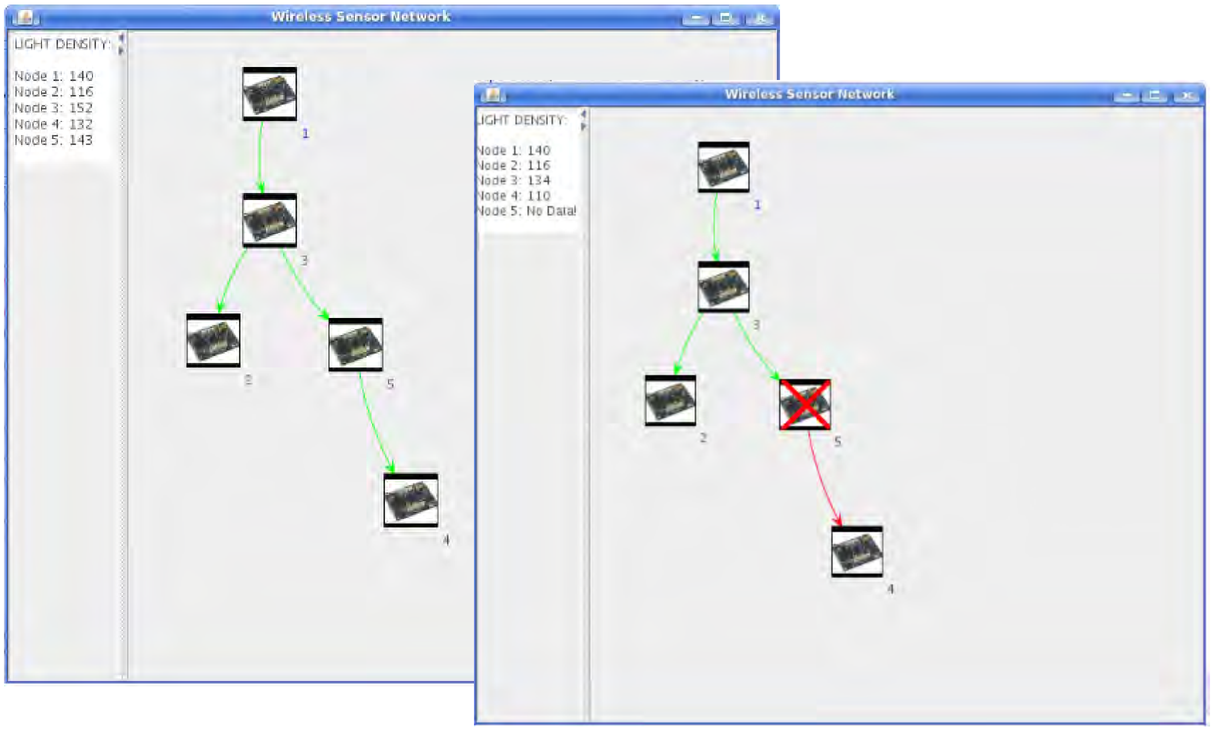


Figure 42 - Placement of sensor nodes

The communication links created during this test can be seen in the next two screenshots taken during the test. The figure on the right shows the output when node 5 is turned off.



```
LIGHT DENSITY:

Node 1: 140
Node 2: 116
Node 3: 152
Node 4: 132
Node 5: 143
```

```
LIGHT DENSITY:

Node 1: 140
Node 2: 116
Node 3: 134
Node 4: 110
Node 5: No Data!
```

Figure 43 - Gateway GUI during the test. Before and after node failure.

Nodes 5 and 2 were used as candidates for replacement. The next four figures illustrate the routes followed by the robot trying to reach the desired sensor node. During the third attempt of replacing node 5 the radius mentioned earlier was set to 1.0m causing the robot to circle around the node while trying to reach it.

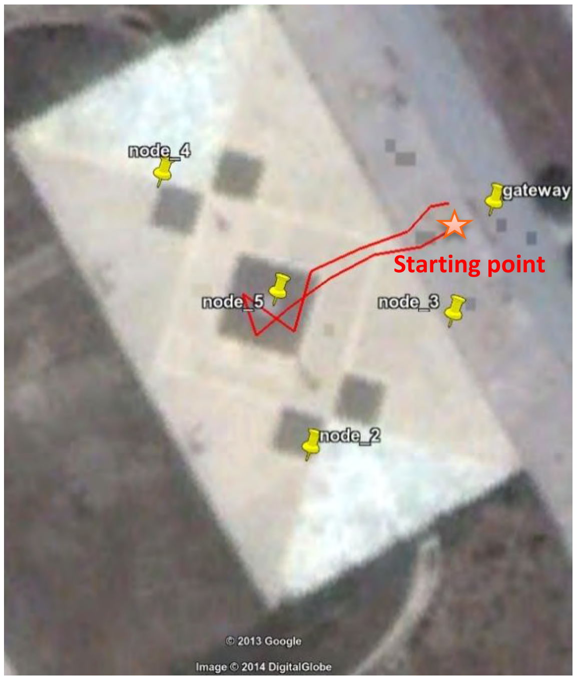


Figure 44 - Replacing node 5, first attempt.

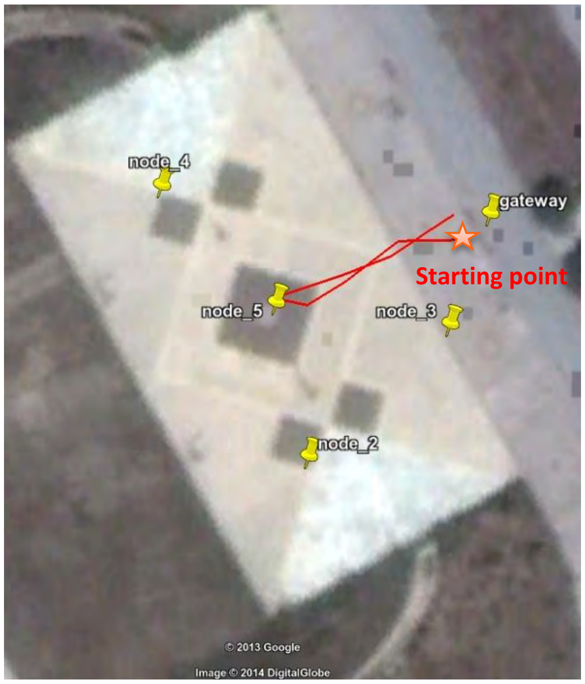


Figure 45 - Replacing node 5, second attempt.

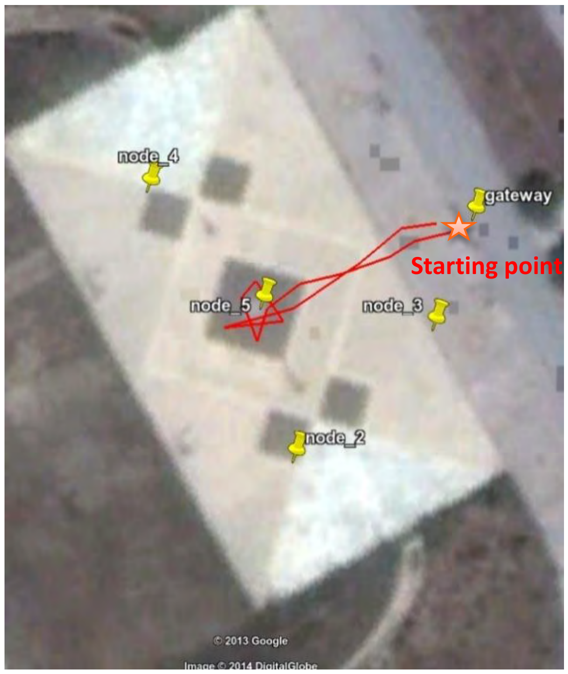


Figure 46 - Replacing node 5, third attempt.

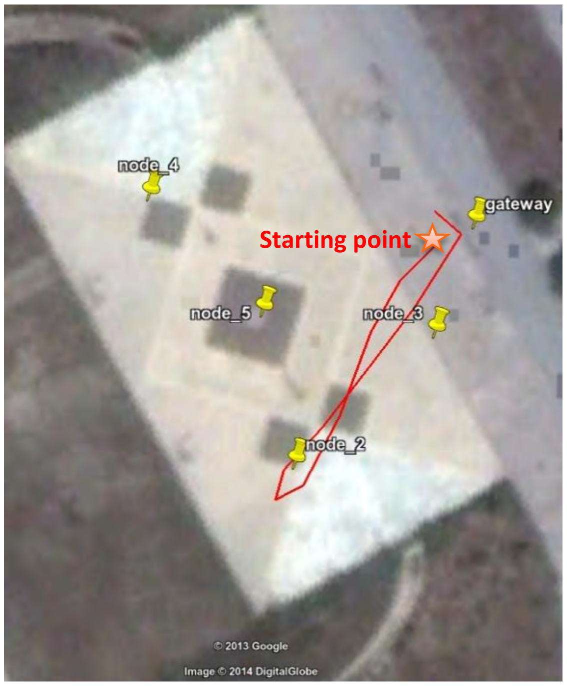
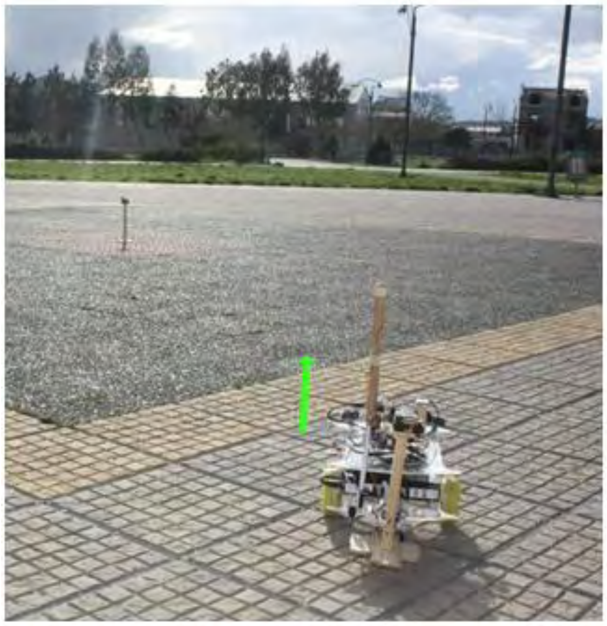


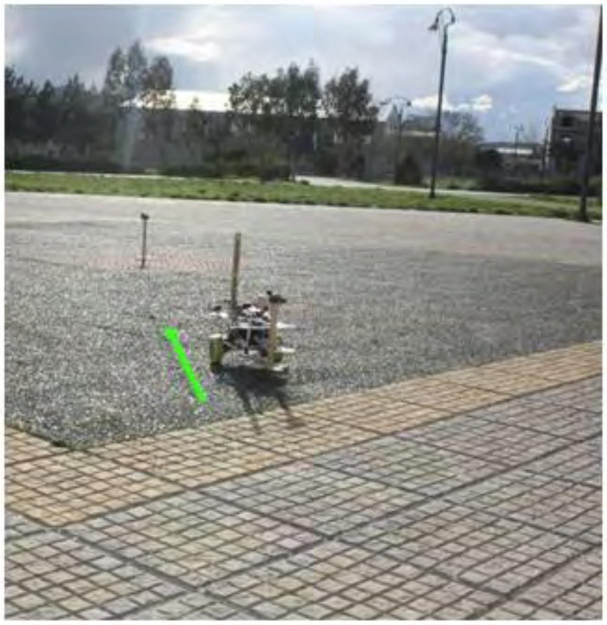
Figure 47 - Replacing node 2

Table 4- Second test total distances

Failed node	Distance Travelled (m)
Node (5) 1st attempt	37.56
Node(5) 2nd attempt	49.08
Node (5) 3rd attempt	59.96
Node (2)	65.12

The average proximity achieved during this test was around 2 meters from the actual node, which was sufficient for network restoration. In the next two figures we can see how the robot modified its current heading in order to reach node 5. The green arrow indicates the current heading of the robot.





During the test the robot was also able to transmit live video from the on-board web camera through an ad hoc network. Three different video resolutions were tested: 320x240, 640x480, and 1280x720 pixels. The maximum range of the Wi-Fi signal was not evaluated. The figure on the right shows an image captured from the web camera.

5.3 Conclusion and Future Improvements

In this thesis we tried to integrate a mobile robot as a part of a WSN. The main goal of this project was the detection and replacement of a non-operating stationary wireless node using an autonomous GPS-guided robot. The WSN that was used consisted of nodes placed in known locations, and a single sink node connected to a Gateway. Concerning the WSN, two applications were developed; one sensing application implemented in nesC that run on the sensor nodes, and another implemented in Java that run on the Gateway. The first was used to measure the environment’s light density and forward these measurements to the Gateway and the latter was used to detect a possible failed node. Furthermore a GPS guided autonomous robot was designed, constructed and tested. The robot was used to physically replace the failed node with a new one. The accuracy of the GPS was questioned at first, but the results were promising. The experiment results showed that the mobile robot could accurately navigate to the desired waypoints and the average distance accuracy in the three experiments was around 2 meters. The external antenna also helped to improve the GPS signal reception.

Within the scope of this thesis, some alternative approaches could not be tried out. Some future improvements may include:

- The construction of a more suitable chassis, aimed for outdoor use.
- The robot’s accuracy during its movement can be further enhanced by integrating additional sensors. These devices are not free of errors; the readings float or are disturbed by noise. The fusion and processing of sensor data can be done using a Kalman filter.
- If greater distance accuracy from the waypoints is necessary, computer vision algorithms can be used.

6 Bibliography

[1] **Jim, A.** *Towards an open testbed for the cooperation of robots and wireless sensor networks.*

[2] *Autonomous Deployment and Restoration of Sensor Network using Mobile Robots.* **Tsuyoshi Suzuki, Ryuji Sugizaki, Kuniaki Kawabata,Yasushi Hada, Yoshito Tobe.** 2, s.l. : International Journal of Advanced Robotic Systems, 2010, Vol. 7.

[3] *Using Mobile Robots to Harvest Data from Sensor Fields.* **Onur Tekdas, Jong Hyun Lim, Andreas Terzis, Volkan Isler.** 2008.

[4] *Control of Mobile Robots Through Wireless Sensor Networks.* **Ricardo S. Souza, Lucio Agostinho, Fabio Teixeira, Diego Rodrigues,Leonardo Olivi, Eliane G. Guimaraes, Eleri Cardozo.** s.l. : impósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

[5] *Mobile Robot Navigation Using RSSI Potential Field in Wireless Sensor Network .* **Xiao SUN, Xiaoguang ZHAO, En LI, Hongchun LI, Zize LIANG.** 14, s.l. : Journal of Computational Information Systems, 2010, Vol. 6, pp. 4751-4759.

[6] *Localization with mobile anchor points in wireless sensor networks.* **Kuo-Feng Ssu, Chia-Ho Ou, Hewijin Christine Jiaju.** 3, May 2005, Vehicular Technology, IEEE Transactions on, Vol. 54, pp. 1187-1197.

[7] **Ntelis, Giorgos.** Remote vehicle control using gesture recognition. [Online] http://www.inf.uth.gr/cced/wp-content/uploads/formidable/gidelis_726.pdf.

[8] Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3. *Adafruit.* [Online] http://www.adafruit.com/products/746.

[9] SENSORS - ROBOT ENCODER. *Society of robots.* [Online] http://www.societyofrobots.com/sensors_encoder.shtml.

[10] Analog to Digital Conversion . *Sparkfun.* [Online] https://learn.sparkfun.com/tutorials/analog-to-digital-conversion.

[11] Tutorial 19: Debouncing a Button with Arduino. *OpenSourceHardwareGroup.* [Online] http://opensourcehardwaregroup.com/tutorial-19-debouncing-a-button-with-arduino-old-version/.

[12] Lecture: Lead-acid batteries. [Online] http://ecee.colorado.edu/~ecen4517/materials/Battery.pdf.

[13] MJPG-streamer. *Sourceforge.* [Online] http://sourceforge.net/apps/mediawiki/mjpg-streamer/index.php?title=Main_Page.

[14] Haversine formula. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Haversine_formula.

[15] Great-circle distance. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Great-circle_distance.

[16] Calculate distance, bearing and more between Latitude/Longitude points. *Movable Type Scripts.* [Online] http://www.movable-type.co.uk/scripts/latlong.html.

[17] KML Tutorial. *Google Developers.* [Online] https://developers.google.com/kml/documentation/kml_tut.

[18] **Gay, Philip Levis and David.** *TinyOS Programming.* [Online] July 16, 2009. http://csl.stanford.edu/~pal/pubs/tos-programming-web.pdf.

[19] JUNG - Java Universal Network/Graph Framework. [Online] http://jung.sourceforge.net/.

[20] *Analysis and Visualization of Network Data using JUNG.* **Joshua O'Madadhain, Danyel Fisher, Padhraic Smyth, Scott White, Yan-Biao Boey.** II, s.l. : Journal of Statistical Software, Vol. VV.

[21] Introduction to BeagleBoard. *Testlabs.* [Online] http://www.teslabs.com/wp-content/uploads/2010/02/beagleboard.pdf.

[22] Minicom. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Minicom.

[23] Embedded Linux Wiki . *BeagleBoardUbuntu.* [Online] http://elinux.org/BeagleBoardUbuntu.

[24] Why the Arduino Won and Why It's Here to Stay. *MAKE.* [Online] http://makezine.com/2011/02/10/why-the-arduino-won-and-why-its-here-to-stay/.

[25] ArduinoBoardUno. *Arduino.* [Online] http://arduino.cc/en/Main/ArduinoBoardUno#.UxYKDkNGmdk.

[26] Arduino Development Environment. *Arduino .* [Online] http://arduino.cc/en/Guide/Environment#.UxYLyUNGmdk.

[27] Introduction to I2C. *Embedded.* [Online] http://www.embedded.com/electronics-blogs/beginner-s-corner/4023816/Introduction-to-I2C.

[28] I2C Tutorial. *Robot-electronics.* [Online] http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html.

[29] **Zogg, Jean-Marie.** GPS - Essentials of Satellite Navigation. [Online] http://zogg-jm.ch/Dateien/GPS_Compendium%28GPS-X-02007%29.pdf.

[30] GPS.gov. *HOW GPS WORKS.* [Online] http://www.gps.gov/multimedia/poster/poster-web.pdf.

[31] *Navigating with GPS.* **Stefan, Jeff.** 123, 2000, Circuit Cellar.

[32] 8-BIT SHIFT REGISTERS WITH 3-STATE OUTPUT REGISTERS. [Online] http://www.ti.com/lit/ds/symlink/sn74hc595.pdf.

[33] Introduction to 74HC595 shift register – Controlling 16 LEDs. *Protostack.* [Online] http://www.protostack.com/blog/2010/05/introduction-to-74hc595-shift-register-controlling-16-leds/.

[34] **Hernandez, Juan.** Arduino's LiquidCrystal Library with SPI. [Online] Juan Hernandez’s .

Appendices

Appendix A The BeagleBoard platform

The BeagleBoard is a low power, low cost, fan-less, single board computer produced by Texas Instruments in association with Digi-key and Newark element14. The board was designed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and open source software capabilities. As a result, development costs decrease as there is no need to buy any compiler or programming tools [21]. The *BeagleBoard C4* used in this thesis is based on Texas Instrument's OMAP3530 system on a chip; it measures approximately 3x3" and has all the functionality of a basic computer (Figure 48). The OMAP3530 includes an ARM Cortex-A8 CPU running at 720 MHZ. The BeagleBoard has been equipped with a minimum set of features to allow the user to experience the power of the OMAP3530 and is not intended as a full development platform as many of the features and interfaces supplied by the OMAP3530 are not accessible from the BeagleBoard.

The main features of BeagleBoard are listed below:

- OMAP3530 processor based on ARM Cortrex-A8 running at 720Mhz
- 256MB of NAND Flash and 256MB of SDRAM
- SD/MMC Card slot
- USB Host port which gives the option to connect a full set of peripherals using a self-powered hub
- HDMI digital output and S-Video (NTSC,PAL) output
- Stereo audio output and microphone input
- Expansion header supporting: I2C, SPI, UART, GPIO

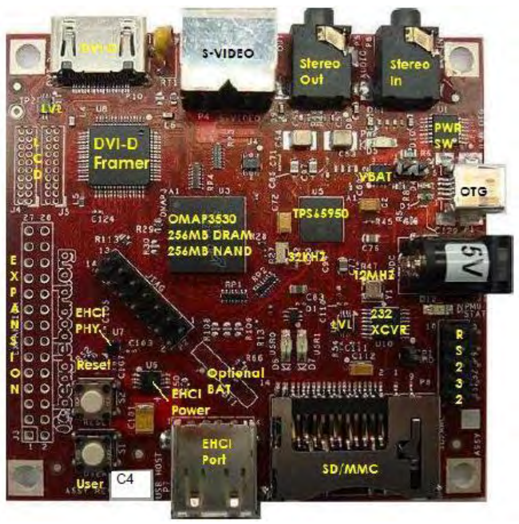


Figure 48 - BeagleBoard C4

BeagleBoard is able to support multiple Operating Systems like Linux, Risc OS, Symbian and Android. The best choice for our purposes was a Linux distribution due to the sufficient number of developers around the world providing the necessary drivers to support BeagleBoard's hardware and other peripherals. Figure 49 shows a high level block diagram of the BeagleBoard, configured to match the component placement of the BeagleBoard.

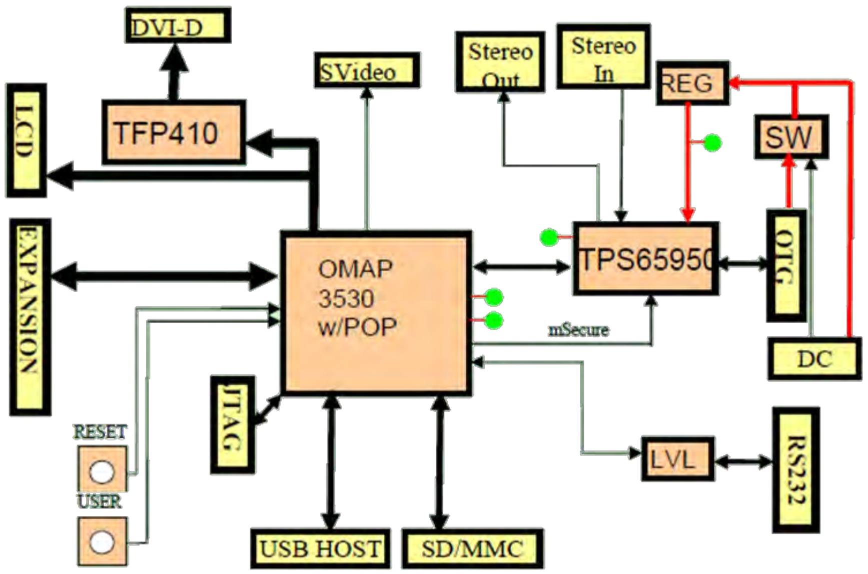


Figure 49 – BeagleBoard high level block diagram

In order to start working with the BeagleBoard the following items are necessary:

- A self-regulated HUB
- A keyboard and a mouse
- An SD/MMC card with capacity greater than 4GB
- A Serial adaptor and a serial RS232 cable.
- A USB to 5.5mm cable for BeagleBoard's power supply

Also for the purposes of this project the following peripherals have been also used:

- A TP-LINK TL-WN722N USB Wi-Fi stick
- A Logitech C270 web camera.

We are forced to use a self-powered HUB instead of a simple HUB, because the BeagleBoard's USB Host cannot provide more than 100mA of current, which is insufficient if multiple peripherals will be connected to the system.

Serial communication between BeagleBoard and Ubuntu pc

It is possible to communicate with BeagleBoard without the need of an external monitor and a graphical interface. In order to do this a terminal emulator must be installed, which emulates a video terminal within some other display architecture. Minicom [22], which was used in this project, is a text-based modem control and terminal emulation program for Unix-like operating systems. A common use for Minicom is when setting up a remote serial console as a way to access a computer. If a Serial port is not available, a USB-to-Serial cable can be used for this purpose. The serial communication with BeagleBoard can be initialized as follows:

--First check if Minicom is already installed. If not, install the program with:

```
$ sudo apt-get install minicom
```

--Run Minicom as root:

```
$ sudo minicom -s
```

```
+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

--Minicom must be configured as listed below. The USB-to-Serial appeared in /dev/ttyUSB0 which may differ in other systems. The serial device must be set to /dev/ttyUSB0 at 115200Bps 8N1 and also with flow control disabled.

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
        | Screen and keyboard      |
        | Save setup as dfl        |
        | Save setup as..          |
        | Exit                     |
        | Exit from Minicom        |
        +--------------------------+
```

--Output when powering up the BeagleBoard:

```
Welcome to minicom 2.2

OPTIONS:
Compiled on Sep  8 2008, 17:03:34.
Port /dev/ttyUSB0

              Press CTRL-A Z for help on special keys



Texas Instruments X-Loader 1.41
Starting OS Bootloader...

U-Boot 1.3.3 (Jul 10 2008 - 16:33:09)

OMAP3530-GP rev 2, CPU-OPP2 L3-165MHz
OMAP3 Beagle Board + LPDDR/NAND
DRAM:  128 MB
NAND:  256 MiB
In:    serial
Out:   serial
Err:   serial
Audio Tone on Speakers  ... complete
OMAP3 beagleboard.org #
```

The BeagleBoard Boot process

The BeagleBoard can boot from several devices. The most common way is to boot from the SD card. When booting from the SD card the board looks for an active (boot flag set) primary partition on the card, formatted in FAT12/16/32. Once this partition has been found, the BeagleBoard attempts to load the boot image X-Loader (MLO), and executes it.

The boot process described also in figure 50 is a three step process:

- Execute X-Loader (MLO)
- MLO finds and executes a boot loader such as U-boot
- The boot loader finds and executes the Linux kernel

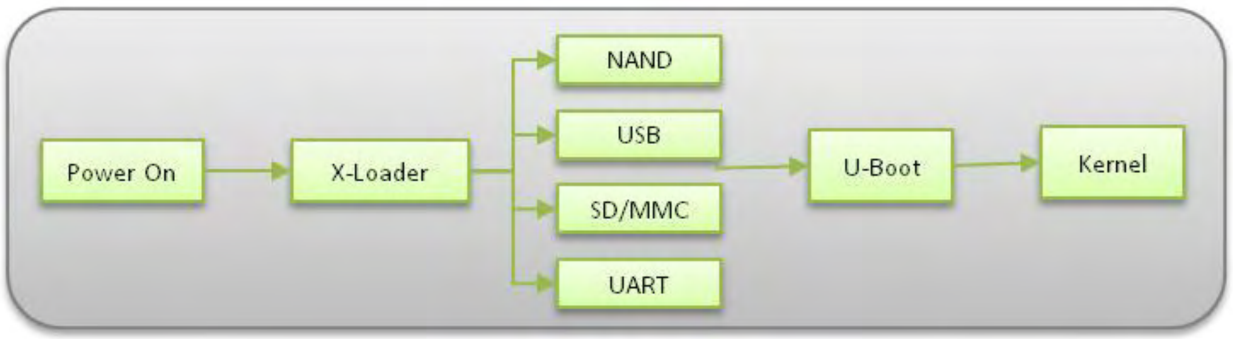


Figure 50 – BeagleBoard Boot process

OS installation

In this project a Linux distribution was selected as the operating system. The two most commonly used distributions for the BeagleBoard are Ångström Linux and Ubuntu. The latter distribution was finally chosen due to better support of multiple peripherals. This section is about running Ubuntu on the BeagleBoard from the SD card. As mentioned in the introduction, an SD card greater than 4GB is required, also a good self-powered hub, a keyboard, a mouse and an LCD monitor attached to the BeagleBoard's HDMI port. The method that follows describes the Ubuntu installation using a complete pre-configured Image of the OS [23]. If a customized kernel is needed, someone can built his own kernel by source, using a cross compiler for ARM architectures. Ubuntu (version) can be installed on BeagleBoard by following the next steps:

--Download the prebuilt image:

```
$ wget https://rcn-ee.net/deb/rootfs/saucy/ubuntu-13.10-console-armhf-2013-12-17.tar.xz
```

--Verify image with:

```
$ md5sum ubuntu-13.10-console-armhf-2013-12-17.tar.xz

bdbb19ae79eb60c3ba6905566c9f0ac6  ubuntu-13.10-console-armhf-2013-12-17.tar.xz
```

--Unpack image:

```
$ tar xJf ubuntu-13.10-console-armhf-2013-12-17.tar.xz

$ cd ubuntu-13.10-console-armhf-2013-12-17
```

--Find the SD card location.

```
$ lsblk
NAME     MAJ:MIN RM    SIZE   RO TYPE   MOUNTPOINT
sda           8:0       0  465.8G    0 disk
├sda1         8:1       0    100M    0 part
├sda2         8:2       0    14.7G   0 part
├sda3         8:3       0 380.7G     0 part   /media/OS
├sda4         8:4       0     1K       0 part
├sda5         8:5       0   66.6G    0 part   /
└sda6         8:6       0     3.7G    0 part   [SWAP]
sdb           8:16      1   14.9G    1 disk
├sdb1         8:17      1   400M     1 part
└sdb2         8:18      1   14.5G    1 part
```

In this example the /dev/sda contains two operating systems; therefore the /dev/sdb is the other drive in the system which is the SD/MMC card.

--Install the image by using the following script configured for BeagleBoard

```
$ sudo ./setup_sdcard.sh --mmc /dev/sdb --uboot beagle
```

Additional options:

- --rootfs (ext4 default)
- --swap_file (swap file size in megabytes)

--The SD card now is ready with the operating system loaded and it can be inserted into the BeagleBoard. At this point there is no graphical interface, so Ubuntu will boot up in console mode. We didn’t want a GUI in order to have the lightest version possible. Default user is *ubuntu* and the password is *temppwd.*

Appendix B The Arduino platform

The very popular microcontroller-based board *Arduino* was used complementary to the BeagleBoard. Arduino is an open-source platform used for building electronic projects that can sense and control more of the physical world than a desktop computer. It is a computing platform that consists of a simple microcontroller (AVR 8-bir architectures) and a development environment (IDE or Integrated Development Environment) for writing software for the board. There are many reasons why Arduino has become very popular [24].

- Cross platform support – The Arduino software runs on Macintosh OSX, Windows and Linux operating system.
- The Arduino programming environment is easy to use for beginners, yet flexible enough for advanced users to take advantage of as well. Also the IDE is open source and available for extensions from experienced programmers. The language can be expanded through C++ libraries. Anyone who wants to understand the technical details can make the leap to the AVR C programming language on which it is based.

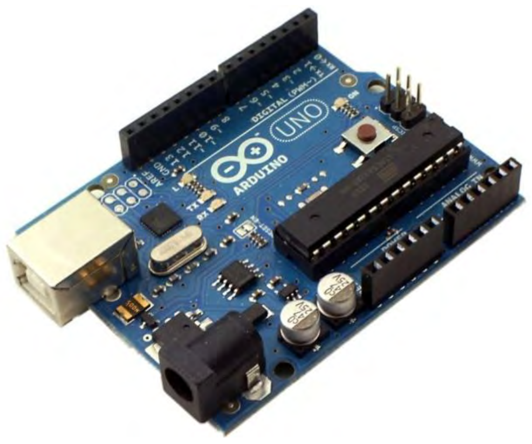


Figure 51 - Arduino UNO R3

- Arduino boards are relatively cheap compared to other microcontrollers and someone can build a much less expensive hand assembled module.
- There is a thriving community of developers using the Arduino board and sufficient documentation can be found easily.

There are a number of different Arduino versions to choose from. The most commonly used version of the Arduino is the Arduino UNO R3 on its latest version [25]. This board is what most people are talking about when they refer to an Arduino. The Arduino UNO is based on the 8-bit ATmega328 microcontroller which comes from a company named Atmel. This microcontroller runs at only 16 MHz being slow in modern terms, and has a very limited amount of memory with 32KB of storage, and 2 KB of RAM memory. The Arduino has 14 digital input/output pins (6 of which can be used as PWM outputs), 6 analog inputs, a USB connection, a power jack, an ICSP connector and a reset button. Powering up the Arduino is fairly simple; just connect it to a computer with a USB cable or use an AC-to-DC adapter or a battery.

When a sketch is uploaded to the board, the boot-loader is used. This is a small program that has been loaded on to the microcontroller and allows the users to upload code without using any additional hardware. The boot-loader occupies 0.5KB of Flash memory and is active for a few seconds when the board resets. Arduino UNO is programmed via USB using a USB to serial adapter. The main features of Arduino are presented in Table 5 below.

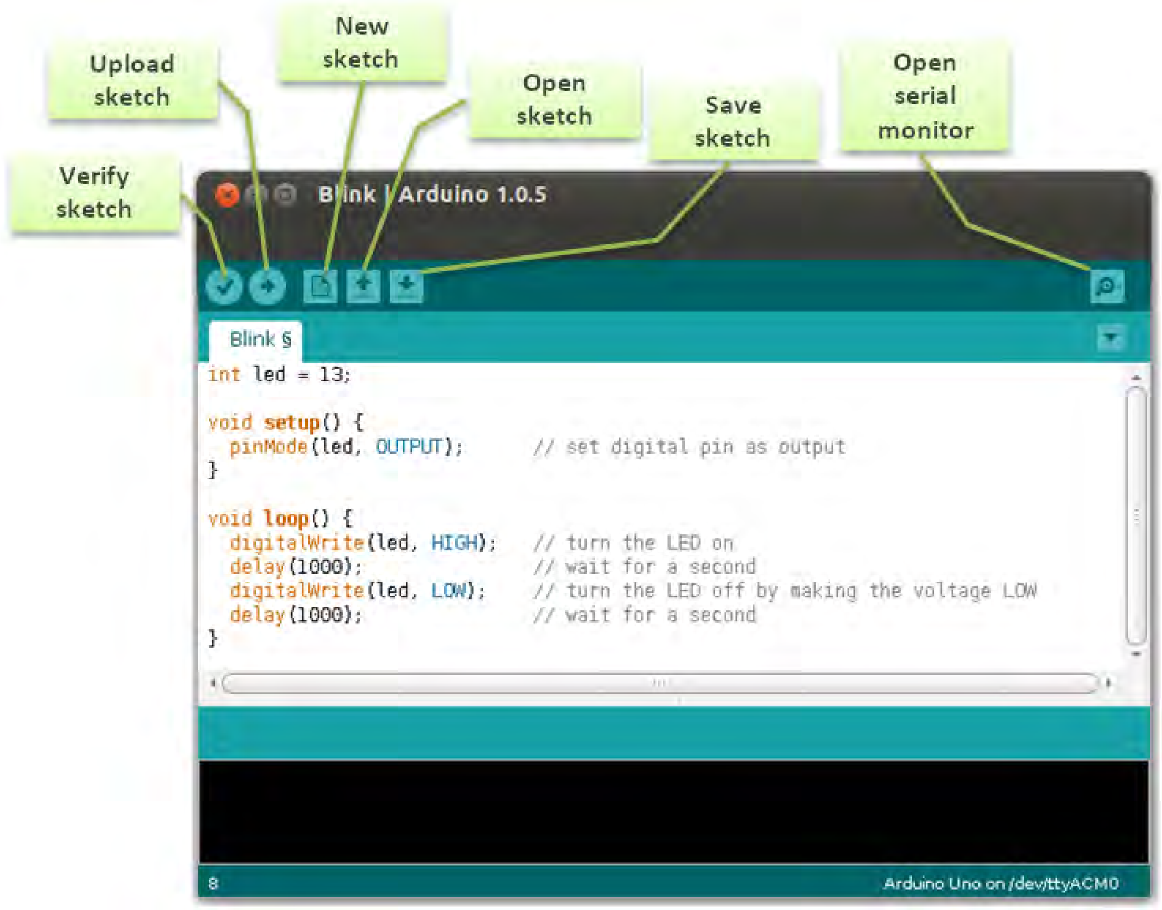
Table 5 - Arduino features

Summary (Arduino UNO)	
Microcontroller	ATmega328p
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Digital I/O	14 (6 provide PWM output)
Analog I/O pins	6
DC current I/O pin	40mA
Flash Memory	32KB (0.5KB used by bootloader)
SRAM	2KB
EEPROM	1KB
Clock Speed	16Mhz

Arduino environment

The Arduino software consists of a development environment (IDE) and the core libraries [26]. The IDE is written in Java and based on the Processing development environment. The core libraries are written in C and C++ and compiled using avr-gcc and AVR Libc. Programs are uploaded to Arduino using the avrdude. AVRDUDE (or AVR Downloader Uploader) is a program used for downloading and uploading the on-chip memories of Atmel's AVR microcontrollers. It can program the Flash and EEPROM, and where supported by the serial programming protocol, it can program fuse and lock bits. This procedure is done automatically by the IDE. This level of abstraction provides a very convenient way of writing programs. The Arduino development environment contains a text editor for writing code, a message area and a text console. It connects to the Arduino hardware to upload programs and communicate with them.

Programs written using Arduino is called *sketches*. These sketches are written in the text editor and are saved with the extension "*.ino"*. Each Arduino program has three main parts: a section where variables are declared, a "setup" section and a "loop" section. When a program executes, it will first define the variables, will then execute the setup section where settings are initialized and then executes the loop section over and over. Bellow an example program is shown. This program turns on an LED on for one second and then off for one second, repeatedly. It uses the onboard LED which is connected to digital pin 13. By clicking the Upload button the IDE compiles and uploads the sketch to the Arduino board.



```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);      // set digital pin as output
}

void loop() {
  digitalWrite(led, HIGH);   // turn the LED on
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Figure 52- The Arduino IDE

The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches. They are a collection of code that makes it easy for the user to connect a sensor, display, module, etc. In order to use a library the #include statement must be inserted at the top of the sketch. Libraries are uploaded to the board with the sketch so the space required by the program increases. Some libraries are included with the Arduino software but it is possible to import a third-party library to support for example a peripheral or sensor. Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be a .cpp file, a .h file and often a keywords.txt file, examples folder, and other files required by the library. The folder containing all the files above must be copied into the libraries folder. If the files are nested in an extra folder the library won't work. After restarting the Arduino application the new library should appear in the Sketch->import Library menu.

Appendix C The I2C Protocol

The Inter-Integrated Circuit (I2C) Protocol is a protocol developed by Phillips intended to allow multiple slaves, digital integrated circuits, to communicate with one or more master ICs. It is a simple, low-bandwidth protocol and it is only intended for short distance communications within a single device or small projects. The i2C has a built-in addressing scheme which makes it easy to link multiple devices. Many examples of I2C compatible devices can be found in embedded systems, such as EEPROMs, real-time clocks and different kind of sensors.

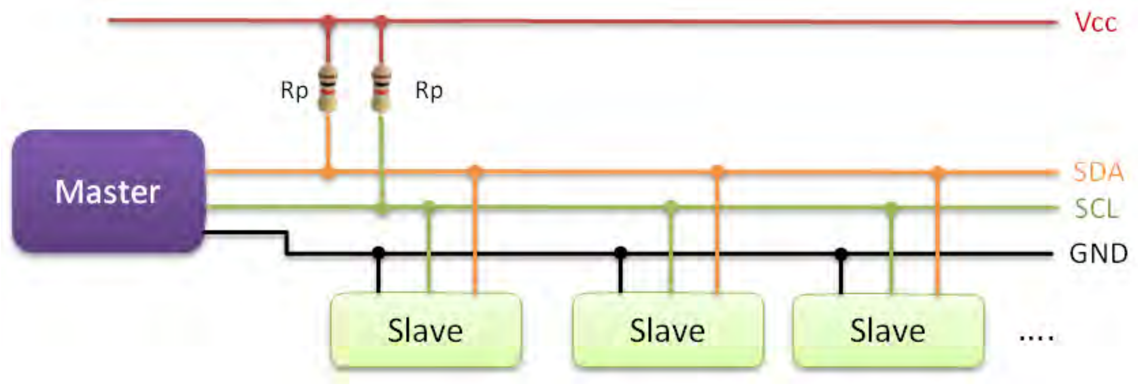


Figure 53 – I2C configuration example

I2C is appropriate for interfacing to devices on a single board or across multiple boards inside a closed system. A typical I2C configuration is illustrated in figure 53 above. I2C is a two wire serial bus using two bidirectional open-drain lines: Serial Data (SDA) and Serial Clock (SCL) pulled up with resistors, supporting the serial transmission of 8-bit bytes. The device that controls the clock signal and initiates the communication is called the master. A device being addressed by the master is termed a slave [27]. The I2C protocol supports multiple masters but most systems include only one. Both masters and slaves can send and receive data. The I2C reference design has a 7-bit address space with 16 reserved addresses, so a maximum of 112 slaves can communicate on the same bus. When data is being transferred, SDA must remain stable and not change whilst SCL is high.

When the master wishes to talk to a slave (the MD25 for example) it begins by issuing a start sequence on the I2C bus. There are two special sequences defined for the I2C protocol. Except the start sequence there is also the stop sequence [28]. Special means that these are the only places where the SDA line is allowed to change while the SCL line is high. These two special sequences mark the beginning and end of a transaction with a slave device.

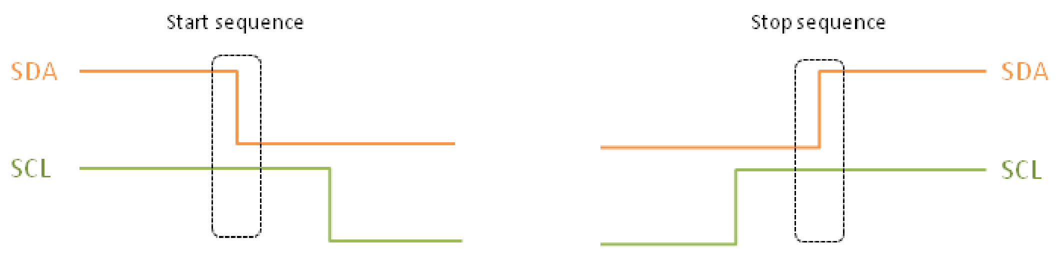


Figure 54 - Start and Stop sequences

After the start sequence the master sends a unique slave address (let's assume 7-bits). Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the most significant bit (MSB). The SCL line is then pulsed high, then low. The eighth bit specifies whether the master wants to send (0) or read data (1) from the slave. This is followed by an ACK bit issued by the receiver, acknowledging receipt of the previous byte. So in total there are 9 clock pulses to transfer 8 bits of data. Then the transmitter (slave or master according to the bit) transmits a byte of data and the receiver issues a new ACK bit. When more bytes are to be sent the same 9-bit pattern is repeated.

In a write transaction (slave receiving), when the master is done transmitting all of the data bytes it wants to send, it monitors the last ACK and then issues the stop condition (P). In a read transaction (slave transmitting), the master does not acknowledge the final byte it receives. This tells the slave that its transmission is done. The master then issues the stop condition. An example of the I2C communication described above can be seen in figure 55.

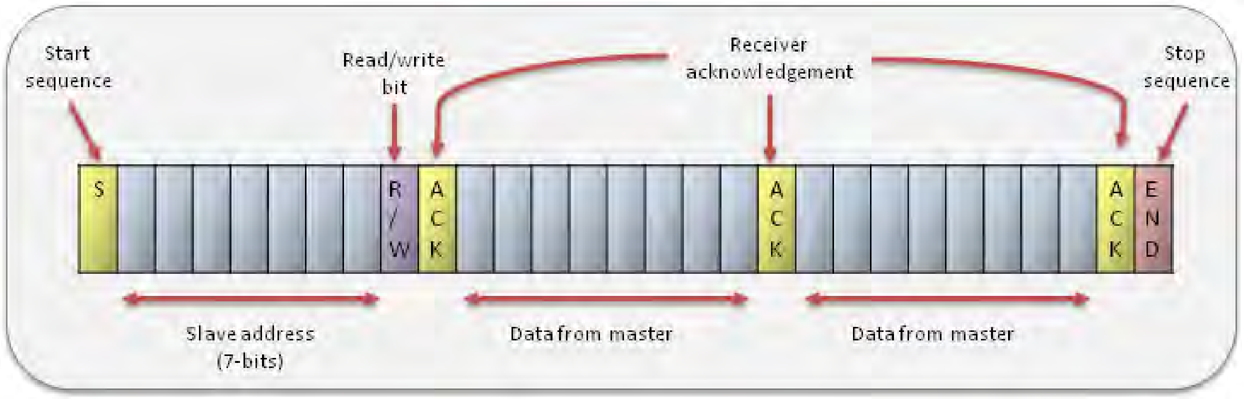


Figure 55 - I2C write transaction example

Typical voltages used are +5V or 3.3V. Systems with higher or lower voltages are also permitted. When two devices running at different voltages communicate via the I2C protocol, a logic level converter should be used in order to avoid any damage to the boards. The most common I2C bus modes are the 100 Kbit/S Standard mode and the 10 Kbit/S low-speed mode. Recent revisions of I2C can host a greater number of nodes by supporting a 10-bit addressing scheme and run faster (400 Kbit/S Fast mode, 1 Mbit/S Fast mode, and 3.4 Mbit/S High Speed mode). All things considered, the I2C protocol offers good support for communication with devices that are accessed on occasional basis.

Appendix D Global Positioning System (GPS)

Among the most stunning technological developments in recent years have been the immense advances in the realm of satellite navigation or Global Navigation Satellite Systems (GNSS) technologies [29]. In a matter of a few years, satellite navigation has evolved from the level of science fiction to science fact Satellite navigation is a method employing a Global Navigation Satellite System to accurately determine position and time anywhere on earth. Using a GNSS system the following values can accurately be determined anywhere on the globe:

1. Exact position (longitude, latitude and altitude) accurate to within 20m to approx. 1mm.
2. Exact time (Universal Time Coordinated, UTC) accurate to within 60ns to approx. 5ns

Speed and direction of travel (course or bearing) can also be derived from these values, which are obtained from satellites orbiting the earth. Speed of travel may also be determined by means of Doppler shift measurements.

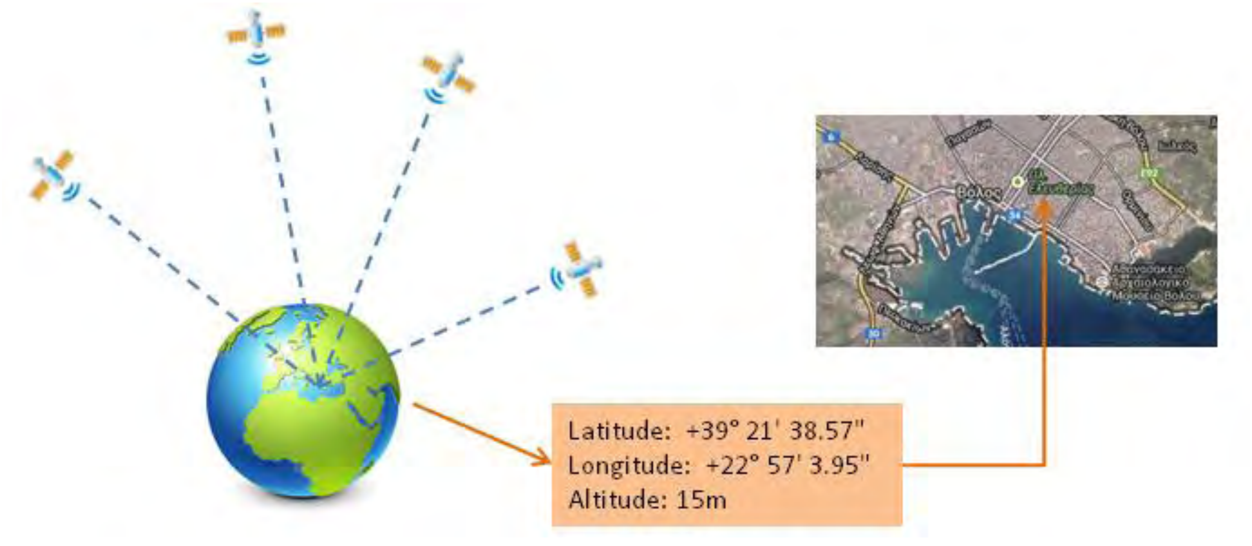


Figure 56 - Illustration of satellite navigation

NAVigation System with Timing And Ranging Global Positioning System, NAVSTAR-GPS is intended for both civilian and military use. The civilian signal SPS (Standard Positioning Service) can be used by the general public, while the military signal PPS (Precise Positioning service) is available only to authorized government agencies. The first satellite was placed in orbit in February 22, 1978 and it is planned to have up to 32 operational satellites orbiting the Earth at an altitude of 20,180km on 6 different orbital planes. The orbits are inclined at 55° to the equator, ensuring that at least 4 satellites are in radio communication with any point on the plane. Each satellite orbits the Earth in approximately 12 hours and has four atomic clocks onboard.

Within the next five or six years there will be 3 fully independent GNSS systems available. The United States will continue to provide GPS and Russia and the European Union should respectively bring their GLONASS and GALILEO systems into full operation.

Each satellite is aware of its absolute location and is constantly streaming that information at a specific channel of the radio band 1500MHz. In order for a GPS receiver to determine its position, it must receive time signals from at least three separate satellites, although fore better accuracy more than four satellites are required. The GPS signals travel through space at the speed of light. When a device receives the radio signals, based on the exact time of arrival of every message, it is able to calculate its distance from each satellite [30]. Let {t1} be the precise time a satellite broadcasts a message, {c} the speed of light and {t2} the exact time of arrival. Then the distance the message travelled is equal to:

$$distance = c * \Delta t$$

$$\Delta t = \{t2\} - \{t1\}$$

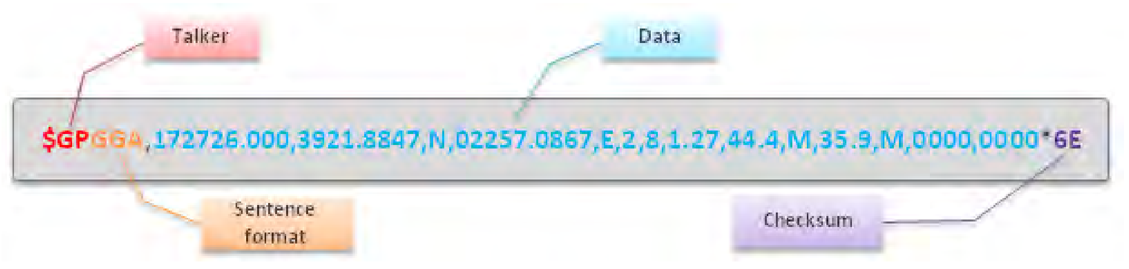
Once the GPS receiver knows its distance from these satellites, it can use geometry (a method call trilateration) to determine its location on Earth in three dimensions.

The GPS message is a continuous stream of data transmitted at 50 bits per second and contains the following information: a pseudorandom code, ephemeris data and almanac data. The first one is simply an I.D. code that identifies which satellite is transmitting information. Ephemeris data, which is constantly transmitted by each satellite, contains important information about the status of the satellite (healthy or unhealthy), precise orbit information, current date and time. This part of the signal is necessary for precise positioning. Each satellite broadcasts only its own ephemeris data. The almanac data tells the GPS receiver where each GPS satellite should be at any time throughout the day. Each satellite transmits almanac data showing the orbital information for all satellites.

Appendix E Understanding the NMEA sentences

Most marine electronic devices like sonars, anemometers, autopilot, and GPS use the NMEA 0183 format. It is a combined electrical and data specification for communication defined by the National Marine Electronics Association [31]. The Adafruit Ultimate GPS is no exception. The NMEA messages are one way messages: form talker to a listener. A NMEA message can be identified by a two-character ID string with a “GP” prefix. A NMEA sentence contains an address field, data field, and checksum. In order to acquire useful information software must be used to parse a NMEA sentence. This procedure is easy for a program because these sentences are consistent and well defined. The general format of a NMEA sentence is:

$<Address>, <Data>*<Checksum><CR><LF>


Talker
Data
$GPGGA,172726.000,3921.8847,N,02257.0867,E,2,8,1.27,44.4,M,35.9,M,0000,0000*6E
Sentence format
Checksum

The Address field is broken down as <talker><sentence format). All fields are separated by commas except checksum which is delimited by a star (*). There are many sentences in the NMEA standard for all kinds of devices that may be used in a Marine environment. Some of the most frequently used sentences are listed in the table below:

Table 6 - Examples of NMEA sentences

Sentence	Interpretation
GPALM	Almanac data
GPGLL	Latitude, longitude information
GPZDA	Date and time
GPGGA	Global Positioning System Fix Data
GPRMC	Recommended minimum data for GPS
GPVTG	Vector track and speed over the ground

The two sentences usually used for GPS navigation are the GPRMC and GPGGA, which contain all the basic information someone may need. For the purposes of this thesis only the GPGGA sentence was used. Details about the GPGGA sentence are described in the next table.

Table 7 - The GPGGA NMEA sentence

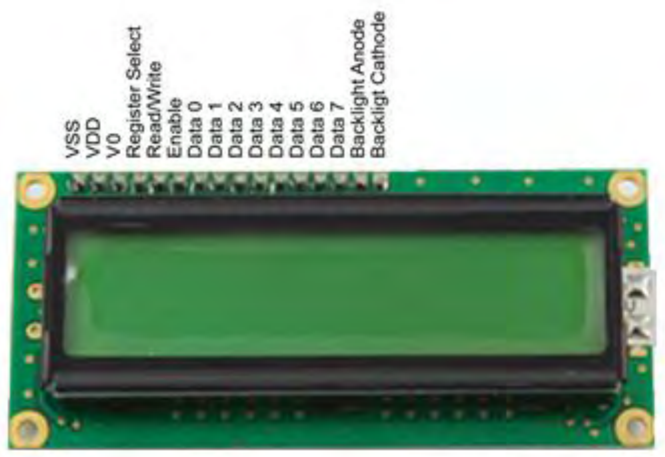
Name	Example Data	Description
Sentence Identifier	$GPGGA	Global Positioning System Fix Data
Time	172726.000	17:27:26
Latitude	3921.8847, N	Latitude information
Longitude	02257.0867, E	Longitude information
Fix Quality: - 0 = Invalid - 1 = GPS fix - 2 = DGPS fix	2	Data is from a DGPS fix
Number of Satellites	08	8 Satellites are in view
Horizontal Dilution of Precision (HDOP)	1.27	Relative accuracy of horizontal position
Altitude	44.4, M	44.4 meters above sea level
Height of geoid above WGS84 ellipsoid	35.9, M	35.9 meters
Time since last DGPS update	0000	No last update
DGPS reference station id	0000	Station id information
Checksum	*6E	Used to check for transmission errors

All NMEA sentences are emitted as ASCII data. The latitude and longitude in NMEA 0183 format are presented in degrees, minutes and decimal minutes (or ddmm.mmmm, where dd equals degrees, mm equals minutes, and .mmmm is decimal minutes). In order to use latitude and longitude values for heading and distance calculations they must be converted to a more appropriate form. The form usually used is decimal degrees (or dd.dddddd). The following steps describe the conversion from degrees- minutes-decimal minutes to decimal degrees.

1. Separate and keep the degrees (dd)
   (*Example*: latitude = 3921.8829, dd = 39)
2. Divide mm.mmmm by 60, resulting in an exponent of 0 and a new mantissa, 0.mmmm
   (21.8839 / 60 = 0.364761)
3. Finally add the saved dd to the previous result
   (Latitude = 39.364761)

Appendix F Controlling an LCD Display using a Shift register

Most LCDs have a parallel interface, meaning that a microcontroller has to manipulate several interface pins at once to control the display.



The LCD's interface consists of the following pins:

- A register select (RS) pin which controls where in the LCD's memory the data will be written to is. You select either the data register or an instruction register. The first holds what goes on the screen and the latter is where the LCD's controller looks for instructions on what to do next.
- A Read/Write(R/W) pin that selects reading mode or writing mode.
- An Enable Pin (E) that enables writing to the registers.
- Eight data pins (D0 – D7)
- A display contrast pin (VO), which controls the display contrast.
- Power supply pins to power up the LCD.
- And finally the LED backlight pins.

The process of controlling the LCD involves putting data that we want to display into the data registers, and then putting instructions in the instruction register. Due to the limited number of the ATmega328p's I/O connections, it is essential to make the best use of the available pins. As described earlier 6 pins are required for the parallel interface but this number can be brought down to only 3 pins. This can be achieved using a Shift register like the *74HC595* [32] .

What is a Shift register?

A Shift register is an easy way to increase the number of digital out pins on a microcontroller. Shift registers are chips which use logic gates to control many inputs or outputs at once. They are digital, like the digital pins on a microcontroller and they should not be used to read analog data e.g. from sensors. The 74HC595 used in this project has an 8-bit storage register and an 8 bit shift register [33]. Data is written to the shift register serially, and then latched onto the storage register, which then controls 8 output lines labeled Q0-Q7. The 74HC595 cannot read data from these pins; instead they can only be used as outputs.

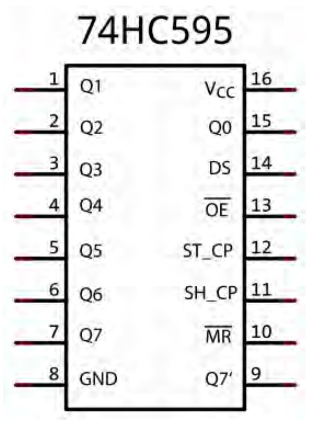


Figure 57 - Shift Register pinout

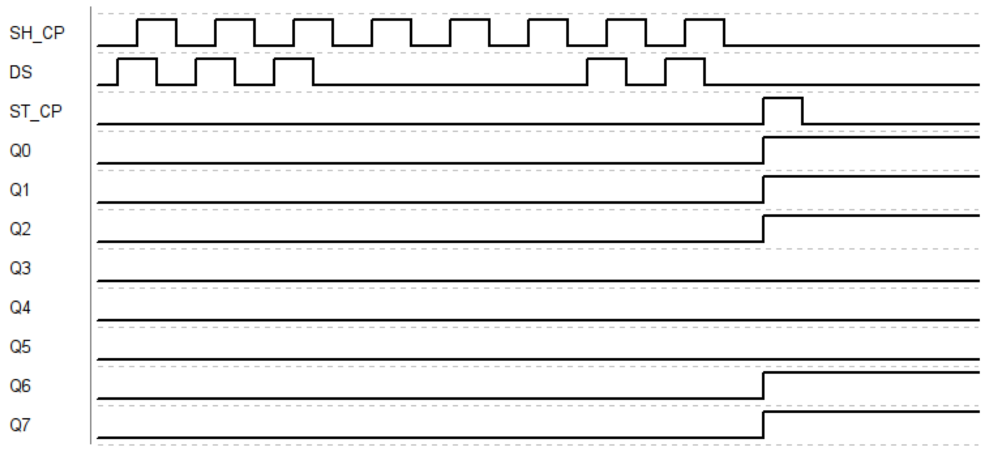
The 74HC595 can be controlled using only 3 pins. The data pin (DS), the clock pin (ST_CP) and the latch pin (SH_CP). The process of sending data to the 74HC595's outputs can be easily described in the next few steps:

- First the ST_CP is set to Low in order to disable the outputs and it is held low while data is being written to the shift register.

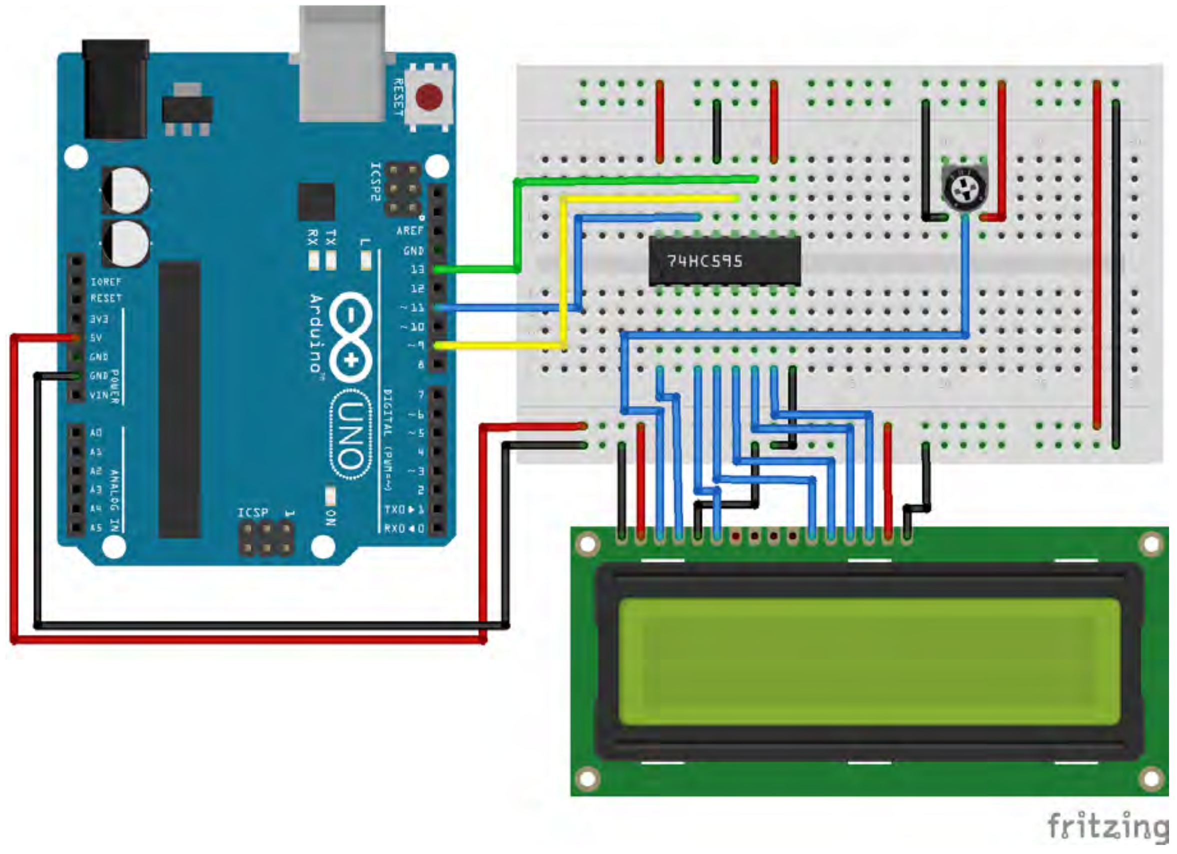
- Next new data is sent to the 74HC595 serially from the microcontroller by pulsing the clock pin and sending each byte of new data out the data pin bit by bit.

- Then the SH_CP goes High and the values of the shift register are outputted to pins Q0-Q7.

The timing diagram below demonstrates how to set the output pins to 11100011, assuming starting values of 00000000.



For this purpose a library that uses the SPI functionality should be used. Juan Hernandez's implementation of the LiquidCrystal Library enables us to control the LCD display using the shift register. The essential connections among the components (Arduino, shift register, and LCD) are illustrated in following figure.


74HC595
Arduino
UNO
fritzing