MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
DEPARTMENT OF AUTOMATION

DISTRIBUTED SYSTEM FOR THE MONITORING AND CONTROL OF A MOBILE ROBOT

DIPLOMA THESIS

Student: **Nándor VERBA**

Project Supervisor: **Prof. eng. Honoriu Vălean, PhD**

2015

str. Memorandumului nr.28, 400114 Cluj-Napoca, România
tel. +40-264-401200, fax +40-264-592055, secretariat tel. +40+264-202209, fax +40-264-202280
http://www.utcluj.ro/

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
DEPARTMENT OF AUTOMATION

VISED,

DEAN,

Prof. Eng. Liviu MICLEA, PhD

HEAD OF DEPARTMENT,

Prof. Eng. Honoriu Vălean, PhD

Student: **Nándor VERBA**

DISTRIBUTED SYSTEM FOR THE MONITORING AND CONTROL OF A MOBILE ROBOT

1. **The Problem:** *The purpose of this project is to design a distributed system that can control and monitor a mobile robot through a wireless network using a mobile device based application as a client, a mini computer that runs a web service and a video stream that communicates with a PCB which is tasked with controlling the motors and reading the sensor values.*

2. **Project Content:** *Presentation page, coordinator appreciation, synthesis, Introduction, Development of the Printed Circuit Board, Design of the Structure and Electric Supply, Programming the Raspberry Pi, The Client and User Interface, Testing and Validation, Conclusion and Future Work, References, Annexes*

3. **Place of documentation:** Technical University of Cluj-Napoca

4. **Consultants:**

5. **Thesis emission date:** 30 October 2014

6. **Thesis delivery date:** 12 June 2015

Student signature _____

Project's supervisor signature _____

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
DEPARTMENT OF AUTOMATION

Author declaration,

I *Nándor Verba*, graduate student of Faculty of Automatics and Computer Science from Technical University of Cluj-Napoca, declare that the ideas, analysis, design, development, results and conclusions within this Diploma Thesis represent my own effort except for those elements which do not and will be highlighted accordingly in the document.

I also declare that from my knowledge this thesis is in an original form and was never presented in other places or institutions but those explicitly indicated by me.

Date: 11 June 2015

Student: **Nándor VERBA**

Matriculate number: **21020918**

Signature: _____

Table of Content

List of Figures

List of Tables

Chapter 1.Introduction

1.1 Theoretical Background

1.1.1 Distributed Systems

In the past few years, there has been an increase of interest and growth in the area of distributed systems and networks. We can now see that an increasing number of processes and tasks of today's computer and communication world are done by distributed computing and distributed control. Distributed computing and control appears in a vast area of applications: Typical examples are parallel computers, the internet ,the European Train Monitoring System, Dam monitoring systems and others, examples of which can be found in [1],where a more detailed description is given of the last two mentioned systems. More recent examples of applications are peer-to-peer systems, sensor networks and multi-core architectures and maybe the most interesting being Internet of Things (IOT).

These applications are similar in the way that separate processors or entities (used as nodes in the literature) are active or available at the same time. These nodes have a degree of freedom, having their own hardware, code and independent tasks. These nodes may share information, resources and tasks in order to solve a complex problem. Coordination and a communication channel are necessary. An example of such a Communication network and setup can be seen in figure 1.1:

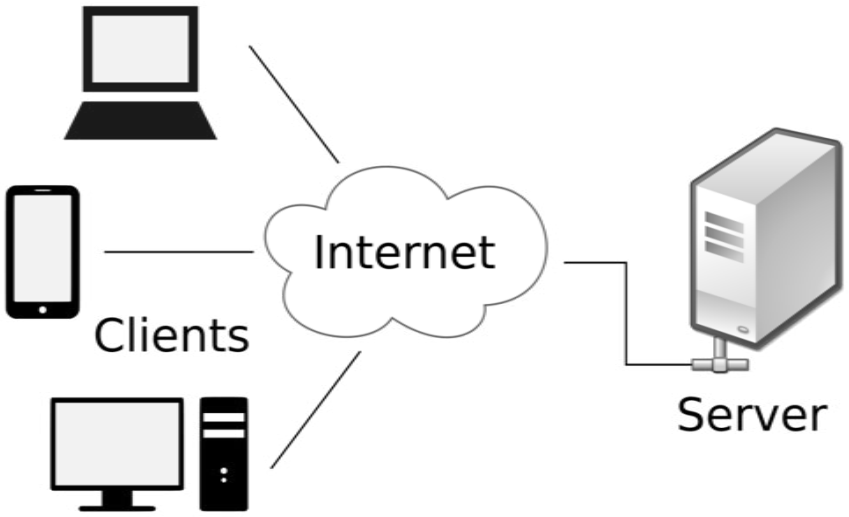


Figure 1.1 Example of a Client-Server Distributed System

Even though there are common features among distributed systems, systems like multi-core architectures and peer-to-peer system are quite different. In certain cases the nodes work synchronously in others they operate asynchronously. A more detailed description of the types of Distributed systems can be found in [2] where they are categorized using different aspects of each system.

These systems require a communication system in order to maintain component information updates, while the interchange of data through the system introduces significant variable delays relative to the processing speed and furthermore a certain level of synchronization is needed between software components in order to perform certain operations or in order to use some resources. Furthermore the use of a communication system reduces the reliability o the whole system. These systems traditionally have a large amount of data that is distributed throughout the system that might be changed locally, giving rise to

difficulty in gathering all the data and decision making and the difficulty to send control signals on time.

The basic principles of Distributed computing introduce fundamental issues that are crucial to the design of distributed systems:

- Communication: The cost of having distributed systems is communication which require processing, storage and power, sometimes being the main surce of consumption of these resources
- Coordination: There are different solutions for the system coordination, each suited for a certain type of system, these are chosen bases on the requirements of the system
- Fault-tolerance: Refers to the property of the system to recover from the partial or complete failure of certain parts of the system.
- Synchronization: There are many synchronization methods, central coordination, logical clock, token rings, virtual synchronization, etc. each of these methods has advantages and disadvantages. A model needs to be chosen that best suits the system.
- Uncertainty: This refers to the property of distributed systems that we cannot know what each node is doing at each exact moment and that nodes are required to solve tasks despite the lack of global knowledge.

1.1.2 Mobile Robots

A robot is a programmable multifunctional manipulator, designed to move objects, materials, tools or specialized sensors and devices after a programmed movement in order to accomplish tasks. The main characteristics of a robot include can be defined as follows: Automated machine-tool, repeated continuous tasks, reprogrammable system, intelligent system, adaptable to the surrounding environment, interact with the environment.

From the point of view of the cinematic structure robots can be classified as follows: Serial robots, parallel robots, hybrid robots, mobile robots, humanoid robots and special application robots (medical, military etc.).

For a device to classify as a robot it needs to have in its structure the following elements: Mechanical couplings, actuator systems, sensors, a controller, user interface, energy conversion systems (transducers, transformers).

Figure 1.2 Pioneer P3 mobile robot

Mobile robots are not fixed to a certain physical location but rather have to capability of moving around in their environment, they are able to do this using their locomotion system. In the case of autonomous mobile robots these are able to guide themselves within an environment using their sensors and the rationing behind the control. In the case of autonomous guided vehicles, these require communication from the outside, leaving the logic behind the movement to an external source having the task of control and relaying of sensor data. Some mobile robots use a combination of these two methods while receiving task from the outside they are able to autonomously move and react to actions given a certain set of

instructions. An example of a robot that can perform these actions is the Pioneer P3 seen in figure 1.2.

In order to produce movement, there has to be a set of forces acting on the vehicle. The cinematic study of mobile robots focuses on the geometric relations that define the system and the relation between the control parameters of the robot without taking into account the forces that act upon the robot.

The exact position of a mobile robot in a system can be represented in two different ways: Either using the posture of the robot or the orientation matrix. The equations and designs presented in the upcoming paragraphs is based upon the course material from [3] modified to suit the details of the application with regard to the movement of robots with two fixed wheels.

In the case of differential drive there are two driven wheels with independent actuators for each wheel. The name is derived from the motion vector of the robot which is the sum of the motion of each independent wheel. This design is used for the sake of simplicity, having the wheels usually placed on each side of the robot. The drawbacks of this type of wheel design are that it's difficult to have the robot move in a straight line, due to the independent actuation of the wheels.

The posture, p of the robot can be shown using the following equation where x and y represent the position of the robot while θ is the orientation of the robot. We also denote U to be the control parameters having v as the linear speed of the robot and ω as the angular speed of the robot.

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \qquad U = \begin{bmatrix} v \\ \omega \end{bmatrix} \qquad (1.1)[3]$$

` The equations in 1.1 are represented on the x, y axis having the direction of the wheels and the direction of the robot represented: Having the speed of the left and right wheels represented by V_L and V_R and the speed of the robot by v the as shown in figure 1.3:

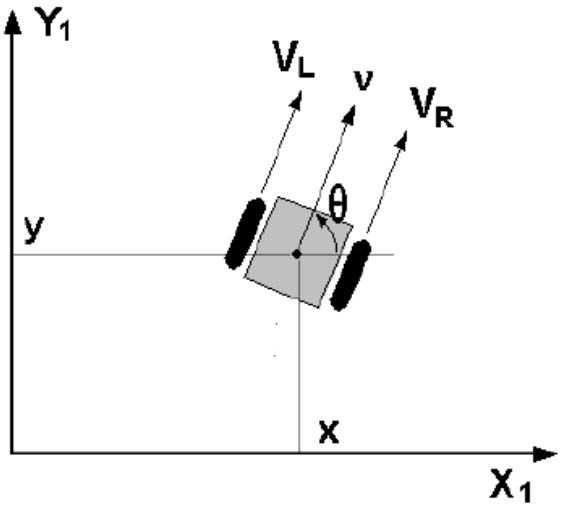


Figure 1.3 Representation of the posture values and wheel speed[3]

Furthermore the Cinematic equation between the posture values of the robot and the control parameters are in a direct relation if we take the simplified version of the orientation matrix presented in[3] and apply it to the know formulas. The resulting formula is shown in equation 1.2:

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix}=\begin{pmatrix}cos\,\theta & 0\\ sin\,\theta & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix} \quad (1.2)[3]$$

The speed of the wheals can be written as a product between the angular speed of the wheel and the radius of the wheel. Finally the control variables can be calculated knowing the speed of the wheels and the Length between the two wheels of the robot. The radius is denoted by *r* while the angular velocities of the left and right wheels are denoted by ω_L and $\omega_{R,}$ the length between the wheels is denoted by L and the notations for the angular velocity and speed are the same. These equations can be transformed in order to obtain the value of the wheel speed from the control parameters. The resulting equation is shown in 1.3:

$$\omega=\frac{r*\omega_R-r*\omega_L}{L} \qquad v=\frac{r*\omega_R+r*\omega_L}{2} \quad (1.3)[3]$$

1.1.3 Internet of Things

Google says that *"There is the potential for 50 billion connected devices by 2020"*. It's a concept that has been named merely in 1999 but the Internet of Things has been available since the early 1980's when at Carnegie Melon University the programmers connected a coke dispensing machine to the internet and checked that it needed to be refilled or not.

Internet of Things is a concept that envisions the connection of billions of *"Things"* like street lamps, wireless sensor arrays, and a vast range of other embedded devices. It offers solutions based on the integration of information technology into most of the everyday devices that will store, retrieve, and process data, as well as communication technologies that are used to communicate between entities. There is a rapid convergence of information and communication technology which is occurring on many layers of technology like the cloud, data and communication pipes/networks and devices.

In order to implement any internet of things technology that is in line with the concept at hand each and every device need to have its own unique identification, which means that the 32 bits of allocated unique identifiers in IPv4 will not suffice and a new packet form will need to be used in the form of IPv6, which will allow us to allocate 128 bites worth of addresses or approximately $3.4*10^{38}$ unique identities to things, which gives us a practically limitless number of unique identities.

In article [4] Janos Simon discusses a possible IOT Layered Architecture layer that would allow the integration of internet of things, this is visible in figure 1.3:

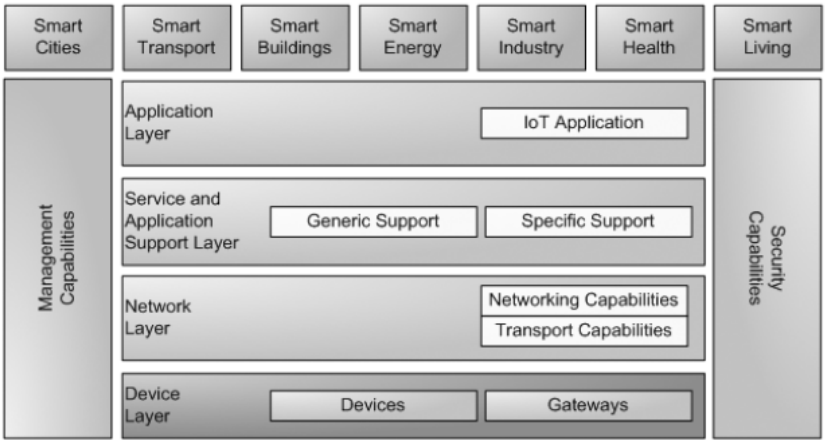


Figure 1.4 IOT Layered Architecture[4]

For the IOT to be a usable the devices will not only need to have their unique identifiers but will have to be able to communicate with other devices. This poses two problems, the first is the means of communication. There are a vast number of wireless

communication possibilities starting from the Radio Transmitter, GPRS, Zigbee, Wireless, Bluetooth and others. The second problem this poses is the communication protocol, these *"Things"* have to have the possibility to communicate with each other through Machine to Machine (M2M) communication, and a possible solution for this is the Message Queue Telemetry Transport (MQTT) protocol that is developed by the IBM Corporation. Finally for the user to be able to connect to the device each device need to be accessible much like a web application, this can be partially done using the Constrained Application Protocol(CoAP) protocol which is best suited for Wireless sensor arrays and in many ways is similar to the Hypertext Transfer Protocol(HTTP).

These devices also require having sensors and actuators in order to be able to sense and control the environment around them. These devices will have specific and application based sensors and actuators.

1.1.4 IOT and autonomous mobile robots

Combining the possibilities of distributed systems, internet of things and mobile robots opens the window for a whole new range of possibilities and applications which apply to the individual or even to the common good, making transport, telemetry sear and rescue and many other applications easier and safer.

Robots in the perspective of internet of things can be connected in two ways, In the first these robots would only send data to the cloud where soft real-time data charts and graphs would be available that would relate the status of the robot at each moment, this would give the user passive control, allowing only to view the status of each robot but not being able to control it, this would allow for a more simple and safe system. Another way of connecting the robots to the cloud would be something similar to Thingspeak where the robots could store information data on the cloud and use it's computing power to resolve complex calculations and problems, as well as receive information from relevant devices or even commands from the cloud.

In this concept Janos Simon in [4] talks about the concept of Internet of Vehicles (IoV) which is tightly tied with the concept of Internet of Energy (IoE) which is a possibility of evolution for the technologies. He describes a communication scheme that would allow M2M communication as well as outside communication with the grid infrastructure and telemetry, a possible communication scheme is shown in the following figure 1.5:

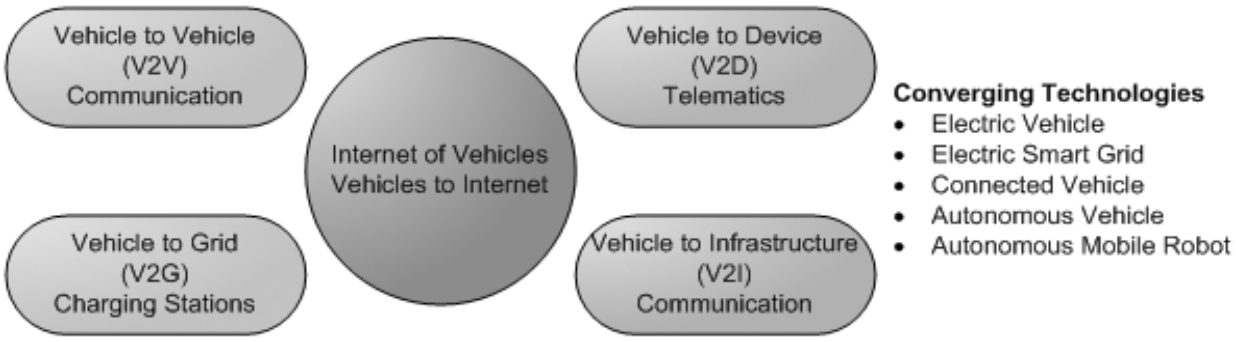


Figure 1.5 Converging Technologies [4]

Another way of looking at these devices is that of semi-independent vehicles that are capable of taking decisions without the need of external involvement, as well as being capable of talking to devices that are nearby and require crucial data and even task information. These mobile robots would be are able to send status updates on request and are able to receive commands from the right entities and execute them between certain parameters. One of the problems of this type of architecture is the security, resolving the correct source and format of the received commands.

1.2 State of the art

On the 14-th of June 2014, Ionut Muntean, Sergiu-Dan Stan, Mihail Abrudan and Kuo-Ming Chao published an article entitled: " An Architecture for Integrated Remote Management and Scheduling",[5] in which they proposed a cloud based monitoring and scheduling scheme where users are able control and monitor a set mechatronics system and the data related to the system based on their interests and needs. They will be able to read important performance indicators and receive statistics in real-time. They will also be able to receive planning rules on whether a certain system is has any changes in operation.

The publication included a safety procedure that had the task of making sure the system would run between certain constraints and that the system would be reset if foresaid constraints after an authorization signal is given.

The presented extended architecture from the paper is another step towards the development of scheduling and remote management system for a mechatronics system based on cloud architecture.

A similar architecture developed for a Mechatronics system and a sensor array is presented in [6] which is the work of the author of this thesis and Nagy Zoltan and was presented in May 2013 at the Conference in Mechatronics held by the Technical University of Cluj-Napoca. This article firstly realized a system that holds one system and can be controlled by scheduling from different users. Further developments in [7] made the system into an adaptable version that is implementable on the architecture presented in [5].

A further development of a sensor system using the same communication method with development in the RF communication part using Microchip processors was done in [8] where the possibility of a remote sensor array that would fit into the system in [5] was designed.

1.2.1 Architecture for the remote management

A proposed architecture is presented in figure 1.6 where the management and scheduling of the system is done via a cloud system. In this example we give a general view of how an example of a cloud based mechatronics system should function.

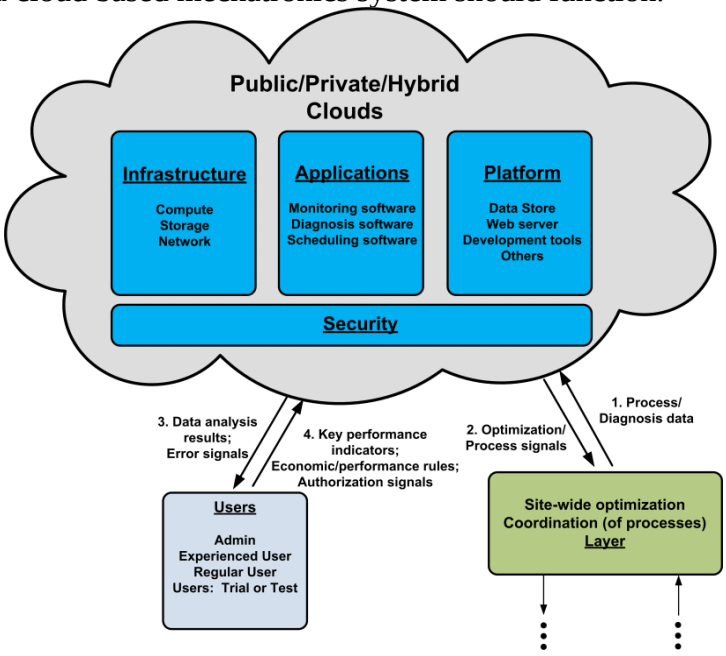


Figure 1.6 Proposed cloud architecture for the remote management and scheduling [5]

This architecture proposes the integration of implemented control and supervision system with the possibility to transfer data to the user and the site-wide optimization of the mechatronics system that will be realized through the cloud. As mentioned in [5] this system is based on 3 pillars, Infrastructure, Platform and Application. The first consists in the medium of communication which is the network as well as computing power and data storage. The infrastructure layer will include the operating system, web servers, database and development tools necessary for the workings of our system.

The authors of the article considered that in direct contact with the mechatronics system there would be an optimization layer that receives control parameters from the clous as well as sends details about the parameters of the system. The communication bettwene the cloud and the optimization layer will be done at regular intervals. Diagnosis and data is sent to the cloud and analysis will be performed on the set of data.

1.2.2 Cloud Laboratory

The partial implementation of the architecture presented in[5]is realized in [6] and [7] with advancements on both the design and the adaptability of the system.

The initial architecture is presented in [6] where there were setup two systems and a main server to run the mentioned systems. These systems were designed in a manner that would illustrate the possibilities of the use of such a system. The first system was an electronic system that had several sensors, leds, a dc motor and a servo attached to it and with several lessons that could be opened for the user to be able to learn from examples. The second system was a computerized numerical control machine or (CNC). This device allowed us to exemplify the control of a more complex system using the same technology. Further development in [7] we also attached a safety mechanism to the machine that allowed the system to overwrite user command in order to preserve its integrity.

The next stage of development was done in [7] where the architecture of the system was modified to allow multiple systems to be attached to a any number of instances of the main server and allowing them to be controlled in this manner. This new architecture is visible in figure 1.7 [7].

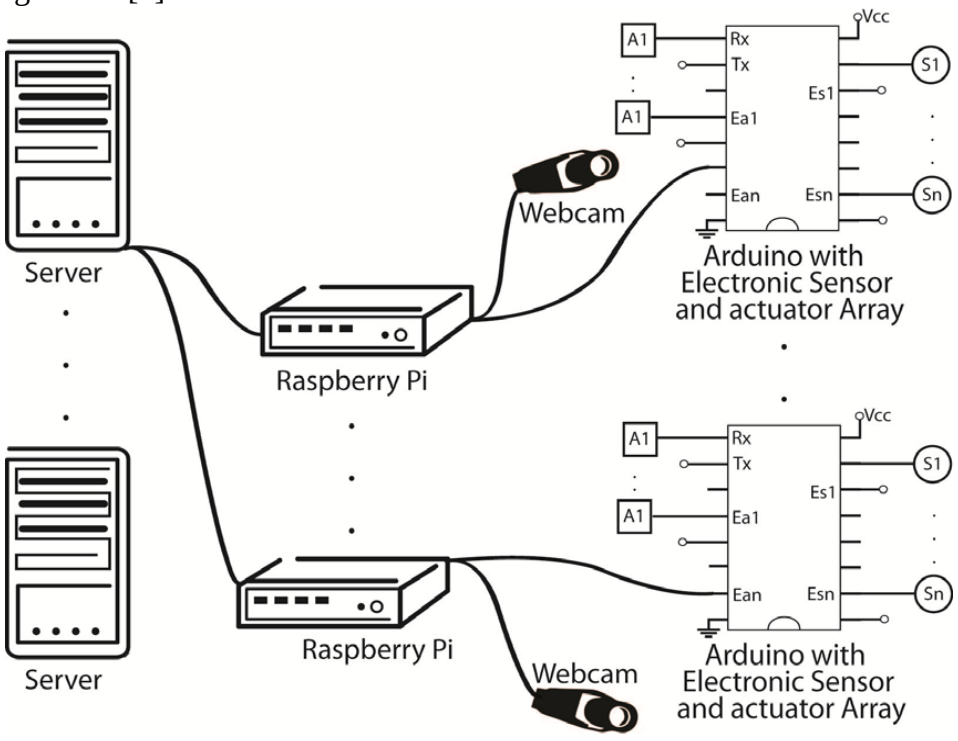


Figure 1.7 Architecture for the cloud laboratory [7]

This system has four major components: The microcontroller and the electronic circuits or other mechatronics devices attached to it; The attached Controller which communicates with the main server(s); The main server; The user interface.

The microcontroller has the role of gathering data from the digital and analog sensors depending on the current configuration of the system, as well as controlling the drivers for the motors, these range from dc, stepper and servo drivers. There are some sensors that send values through protocols like spi and i2c, this need to be handled by the microcontroller as well. The microcontroller communicates with the Controller or Raspberry pi using UART, which is responsible for the communication of the sensor data an control data as well as for programming the controller which can be done remotely from the main server.

The role of the Controller or Raspberry pi is to receive data from the microcontroller and transmit it to the Server as well as to host the video feed through the motion service. Furthermore, it can receive code from the Server, compile it and upload it to the microcontroller giving back successful compilation response or giving back the errors that were found in the code. An example of this can be found at the results section in the figure 1.8 We can see that the AVR compiler is used and the size of the hex file and other attributes are displayed when the compilation is successful.

The main server's main task is scheduling the use of the boards. This is done using a database which allows the user to check whether a certain board is in use or not at a certain point and if it's not to schedule himself for that use. The server also holds the user data and the authentication information as well as the saved codes and a history of all the interactions with the Controllers and their result.

The user interface is a vital part of the system. It allows the user to use the functionalities of the server and see the capabilities of each subsystem attached to the main server. In the use of the interface we define two types of users: the administrator user is allowed to modify the schedule and can view to code of users in order to help with debugging; the regular user can schedule appointments with Controllers if they are free or cancel his own appointment, as well as save his own code and view the examples. The user interface is Apache and MySQL based

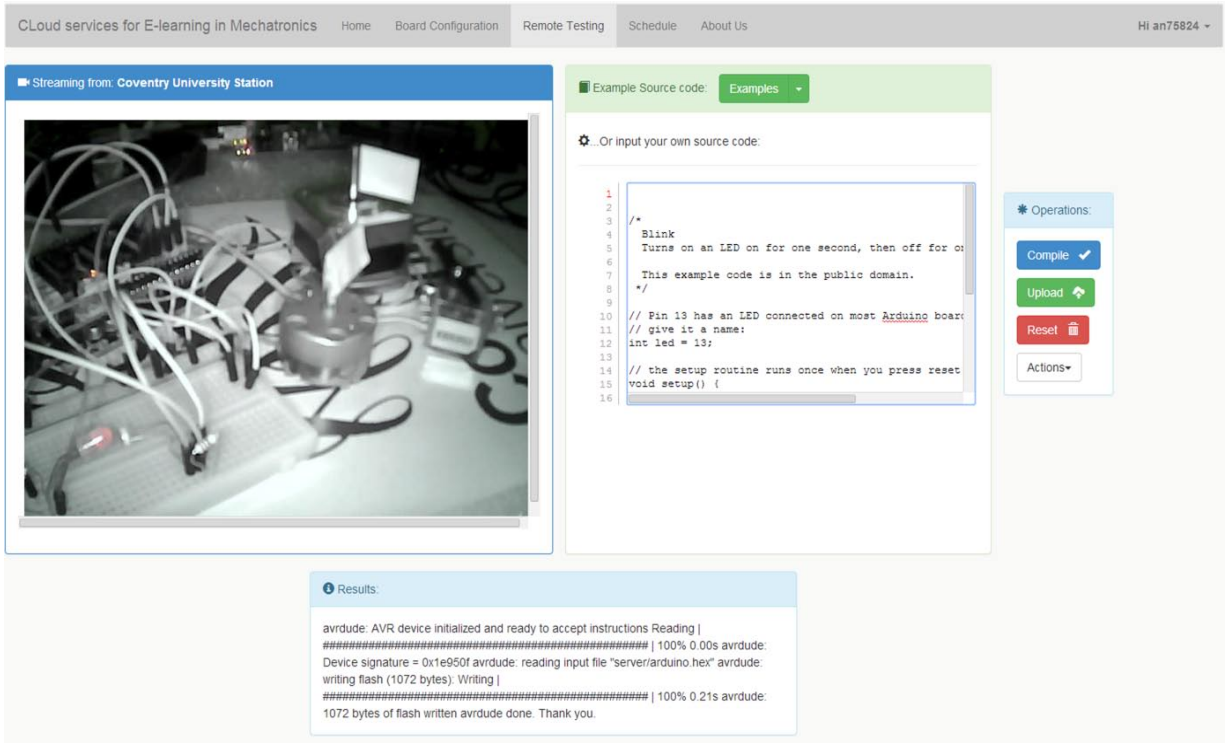


Figure 1.8 The web interface for communicating with the mechatronics system [5]

1.2.3 Microcontroller Advancements

Like in the previous applications in [8] the project is built with the purpose of monitoring and controlling different locations. For monitoring we used a set of several sensors, which include light, humidity, proximity, temperature and current sensors, with a range of 20 to 30 meters. For control we use different actuators that are triggered using the data received from the sensors. The user has the possibility to connect to the server and fully control the location.

The innovation in this part is the communication with the sensor's microcontroller works with a wireless radio connection on 434MHz running a TCP/IP three way handshake based protocol to synchronize the multiple microcontrollers and to convey data. This solution allows for the connection of multiple microcontrollers to the same hub, without having to use cables and it doesn't have the number of devices limited by the number of ports on the hub.

The issues that arise from using this method are mostly caused by interferences and overlapping of sent data. This issue can be countered by creating a synchronization method that can counter these effects and manage all the microcontrollers. This method needs to be run on the hub. It needs a powerful processor and a stable power supply.

The method we created to handle these problems is based on the TCP-Three way handshake protocol. The design is shown in Fig 1.9. It is separated into four cases. The first one depicts the case of data collision. The second shows how a successful three-way-handshake would look, while the last two show how the sensor data is sent, and how the actuators are controlled.

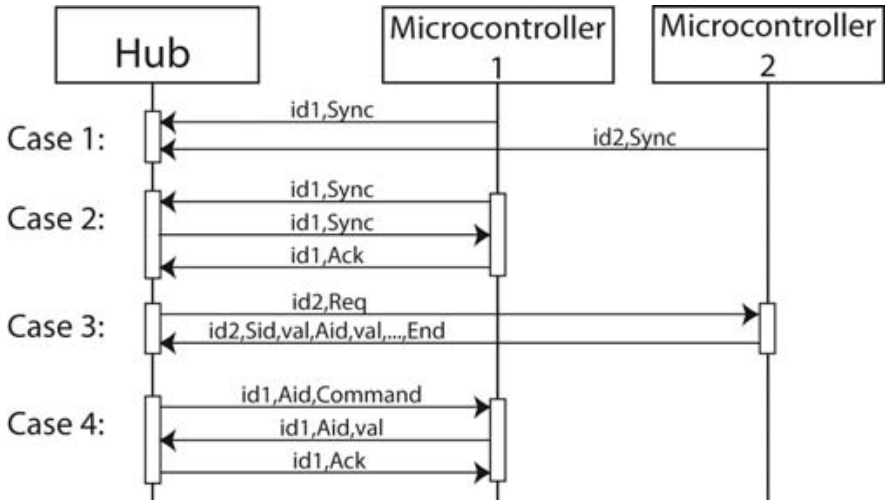


Figure 1.9 Sequence Diagram for the Microcontroller Hub Communication [8]

The second phase was testing out the radio communication protocol using three Arduino boards. The boards are good for testing because of the simple language required to program them. We had one board taking the place of the hub and the other two being the sensor platforms. We used a simplified version of the protocol that was written to work with only two boards. The protocol can easily be modified to recognize a higher number of microcontrollers.

The main interest in this article is attributed to the mentioned communication protocol which can handle a wide range of cases and devices. This protocol can be modified to suit our need for an IOT system, especially for the Machine to Machine communication, giving a basis to a reliable and simple communication protocol. The other interesting thing is the way it sends data to the hub and receives it. The use of unique id-s is another step towards IOT, if we consider allocating id's using a unique identifier we can implement the system for a wide range of sensors and actuator arrays.

1.3 Practical Applications of the System

The connection of vehicles, mobile robots and other robots to the Internet, giving arise to the Internet of Things brings about a wealth of new possibilities and applications which bring new functionalities, both for leisure, safety and work purposes which could make transport easier, healthcare more safe and cheaper as well as menial tasks around the house a thing of the past. In this context the internet of things can be separated into the Internet of Vehicles and the Internet of Energy, these two representing the main interests at the moment in the field.

1.3.1 Self-driving Vehicles

The concept of Autonomous cars has been around for some time now, the first attempt reaching back as far as the 1920s with no significant results. In the 1950's promising tests took place, but the first self-reliant and autonomous vehicle appeared in 1980's with two innovations: The Carnegie Mellon University's Navlab and ALV project in 1984 and with further advancements with the Mercedes-Benz and Bundeswehr University Munich's EUREKA Prometheus Project in 1987.

These vehicles are equipped with advanced sensory systems as well as means to acquire data from a wireless network array regarding stop signs, the level of traffic or any other information useful to the device. The sensors include Radar, lidar, Gps and Computer vision which would give the car a comprehensive view of the environment it's in and allow it to make decisions based on this. An example of how the Google car views the environment can be seen in figure 1.10:

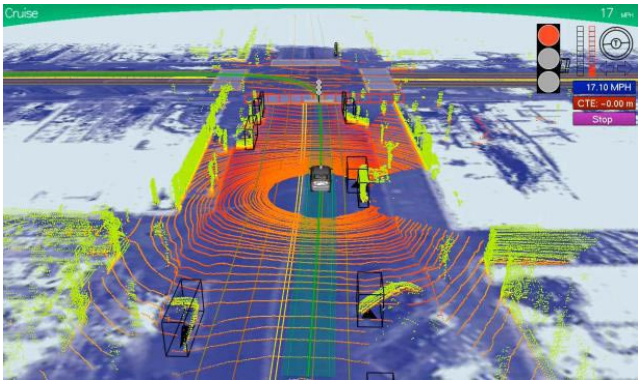


Figure 1.10 How the Google Car sees the World

Currently we can know that some states in the Us as well as some European countries like Belgium, France, Italy, UK, Germany, Netherlands and Spain have allow the testing of Driverless cars on the roads. Furthermore, as of 2014 a self-driving vehicle is available commercially, that are open-aired shuttles for pedestrian zones.

In their current state these cars are equipped with very expensive sensory equipment as well as a very expensive lidar system that allows the vehicles to map the surrounding areas inch by inch. This system generates a 3D image of the surrounding area that the car compares to high quality maps that allows itself to drive around. The image includes information about the lights and other road hazards.

In their current states these cars have their limitations. The prototypes have not yet been tested in conditions of bad weather such as heavy rain or snow due to safety concerns. Because the roots are based on pre programmed route data these cars might not take into account road works or temporary lights that might be put up. Furthermore in certain situations they might revert to an extra cautious mode which is slow and is activated in unmapped intersections and situations. They have problems when objects like trash and papers or light

debris are identified as harmful and they veer to avoid them. Furthermore the lidar can't spot certain potholes or even police officers that might signal directions to the car. Google says that these issues are to be fixed by 2020.

An example of what the Google self driving that was released in June 2014 car looks like today can be seen in figure 1.10:

Figure 1.11 Google Driverless Car

1.3.2 Cloud Medical Robots

The idea of cloud medical robots is based on the existence of a medial cloud (healthcare cluster) that contains up to date information like disease archive, electronic medical records of patients, a health management system, practice service, analytical service, clinical solutions and expert systems. The functionality of the robot would be to connect to these services and help the work of doctors or even deliver assistance to the patients, as well as assists as a co-surgery robot.

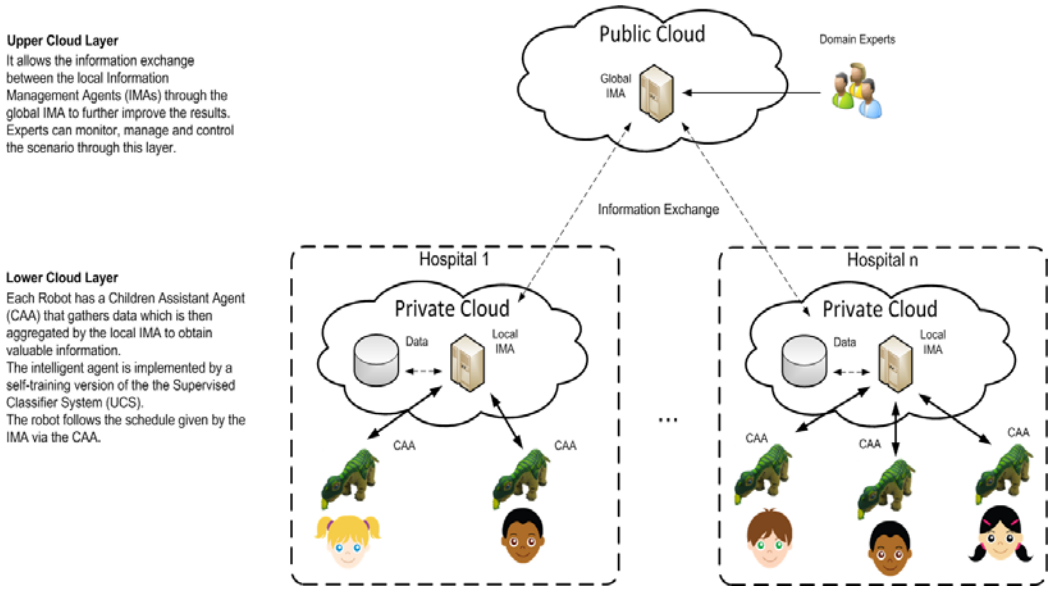


Figure 1.12 Scheme for the cloud architecture

A possible implementation of cloud features into a medical facility is presented in[9] where they discuss a cloud based robotics architecture to effectively develop complex tasks to assist hospitalizes children. The figure 1.12 shows the example architecture for such a human-robot interaction. This system would allow for the system to monitor the children using low-cost robots and due to the human-readable set of rules related to the child-robot relationship this would improve their experience. The article also shows a preliminary set of tests that are promising for the field.

The system involves using an intelligent system that would have personalized answers for every child and this would allow the child to make a correlation with its robot. This system is deployed using the Plea robots that are already in hospitals.

1.3.3 Industrial Robots and Smart Factory

The effects of the internet and indeed its use in the industry are best highlighted by [10] where the German Industry 4.0 Plan specifies that *"Industry is on the threshold of the fourth industrial revolution. Driven by the Internet, the real and virtual worlds are growing closer and closer together to form the Internet of Things. Industrial production of the future will be characterized by the strong individualization of products under the conditions of highly flexible (large series) production, the extensive integration of customers and business partners in business and value-added processes, and the linking of production and high-quality services leading to so-called hybrid products."*

This solution or proposition comes from Germany party because Germany is one of the most competitive industrialist and also a leading supplier worldwide. This is partly due to the specialization on the research and development in manufacturing and innovative production technologies and the ability to manage complex industrial processes.

After the Mechanization, Electrification and Computerization of Industry we are at the steps of a fourth industrial revolution using the Internet of Things, this is the base of action Industry 4.0.En example of how the systems will interact can be seen in figure 1.13 where the factory is put in the center having the main subsets: Smart mobility, which consists of the tracking if the company vehicles; smart logistics refers to the automatic ordering and resupplying of the warehouse and the sending of orders; smart grid, which refers to the smart use of electricity making production cheaper and the impact of factories on the grid less; smart buildings refer to the integration of sensors into the buildings for security heating, wage and other uses; smart products refers to making of products that can be tracked and thus increasing the reliability and diagnosis capabilities of the plant.

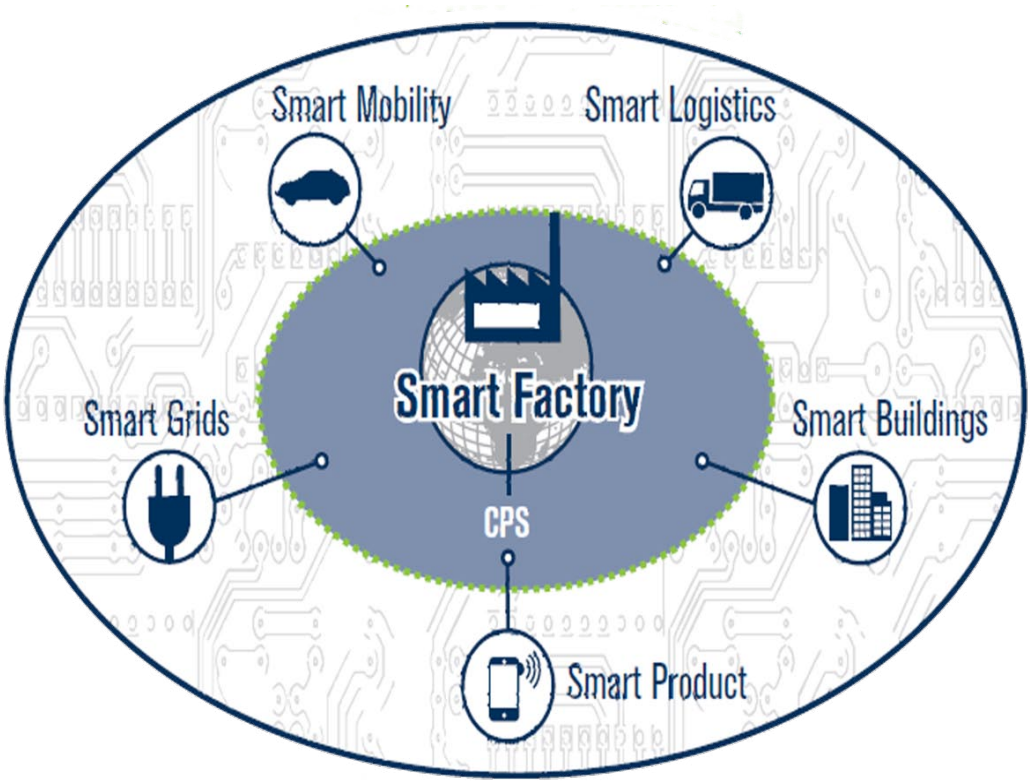


Figure 1.13 Smart Factory

The potential of industry 4.0 is immense: The Smart Factory can consider individual customer requirements and even profitably produce unique pieces. In industry 4.0 business and engineering Processes are dynamically designed, the production can be changed quickly and flexibly to disorders and injuries, based on, for example suppliers' reactions. The production is completely transparent and enables optimal decisions.

By Industry 4.0 a new form of values as well as new business models will be able to get implemented. This is especially good for start-ups and small businesses, this offers a Chance of developing downstream services and offers.

Industry 4.0 also makes a contribution to address the challenges posed by Resource and energy efficiency, urban production and demographic change. Resource productivity and efficiency can be continuously monitored and improved in Industry 4.0.

Finally, partially thanks to intelligent assistance systems, the focus on creative, value-adding activities can relieve the workers from routine mundane tasks. In light of an impending shortage of skilled workers, this solution can allow the system to rely on the productivity of older workers that will get a longer working life. The flexible Work organization allows employees to have a work-life balance and better training combined with each. All of these factors will add up to increased production and increased work satisfaction from the part of the factory worker.

1.4 Objectives and Challenges

The main objective of this thesis is to develop a distributed system that can control and monitor a mobile robot through a wireless network using a mobile device based application as a client, a mini computer that runs a web service and a video stream that communicates with a PCB which is tasked with controlling the motors and reading the sensor values of the designed mobile robot.

1.4.1 Electrical and Structural requirements

The Mobile robot base of the device has a few requirements that it needs to meet in order to be suitable for the system. The base needs to be a cheap solution, in order to allow the making of multiple platforms, with the addition that it has to be sturdy and resilient to the effects of testing. It needs to have spaces for both the wheels, the motors and to be able to comfortably fit the pcb leaving room for the sensors, proximity, line following and the digital hall sensors. The structure also has to accommodate the raspberry pi board and the adjacent raspi camera with and optional addition of a usb camera to replace the raspicam.

In addition to the Structure the system requires a battery that is at least 7.4 volts and can give a current that can go as high as 2.4 amps, this is order to power the motors and also to be able to power the pcb and the raspberry pi as well. Furthermore the system requires motors with a gearing that will allow the microcontroller to control the robot. In addition to this, the structure will require the design of a step-down voltage controller to assure that the raspberry pi doesn't get burned out due to high voltage.

1.4.2 PCB Requirements

The printed circuit board is required to be able to read the values of all the proximity and line following sensors, as well as to determine the time delay between two events generated by the hall sensor coming close to the magnets, as well as counting the events. The microcontroller also needs to have debugging features and a communication channel dedicated to possible Bluetooth, RF 434 or 2.4 GHz connections for M2M Communication.

The pcb also needs to implement a communication with the controller in such a manner that the data received and sent can be checked and the system can recover very easily

in case of errors or other problems or even the resetting of one or another system without cases where the system is no longer controllable.

1.4.3 Controller, Client, User Interface Requirements

The Controller needs to be able to communicate with the pcb, as well to host a REST server that can receive commands and transmit them to the microcontroller. Furthermore the Server needs to be able to stream a video feed from the raspberry pi or other usb device with as little a delay as possible. In addition to these features the controller will have to account for some safety features and the unique identification of the system.

The Client will host the user Interface. This client will have to be able to connect to the Controller and send POST and GET requests to the server and interpret the data as well as to be able to host the video feed. Furthermore a set of mini applications will have to be implemented in order to demonstrate the features of the system. The application needs to account for any errors that might occur during the communication or any erroneous sensor values.

Chapter 2.Development of the Printed Circuit Board

The printed circuit board or PCB is a mechanical support with electronic connections that connects the electronic components using electrically conductive tracks, pads, through holes and other features that are cut from copper sheets laminated with a non conductive substrate. The design of the pcb can vary a lot depending on the conductive material and the lamination material, usually they are categorized depending on the number of levels they have: single, double or multi-layer; Each type a pcb has a different set of written and unwritten rules that apply when you route the components. Some advanced multi layer pcb-s may have capacitors resistors or active devices embedded into the substrate.

The manufacturing of one sided pcb-s can be done at home as well and is usually associated with diy applications or simple remote controls or other appliances. The benefit of one sided pcb-s is the reduced cost and the simplicity, the payoff is that these require connection between wires that physically can't connect.

For this application a two layered pcb is used which doesn't require extra connections but introduces the problem of vias and the need for the two layers to be perfectly aligned, due to these issues and others, this type of pcb is usually done by factories and is difficult to do at home.

2.1 Electrical Components

Electrical components are any discrete devices or physical entities in an electronic system. These are mostly Industrial products available in a singular form. Electrical components have two or more electrical terminals, these are used to an electronic circuit in order to perform a function.

2.1.1 5V and 3.3V Voltage Regulator

For the circuit board a power supply of both 5V and 3.3V was needed due to the different set of sensors and communication devices attached as well as a direct connection for the H bridge to drive the motors using the min. 7.3 volt input.

For the lower voltages that were required different version of the Voltage regulators were used depending on the availability from the local suppliers. For the 5V voltage regulator the LM2594M-5V is used. The schematic of the connections and the adjacent capacitances and shottky diodes can be seen in figure 2.1.

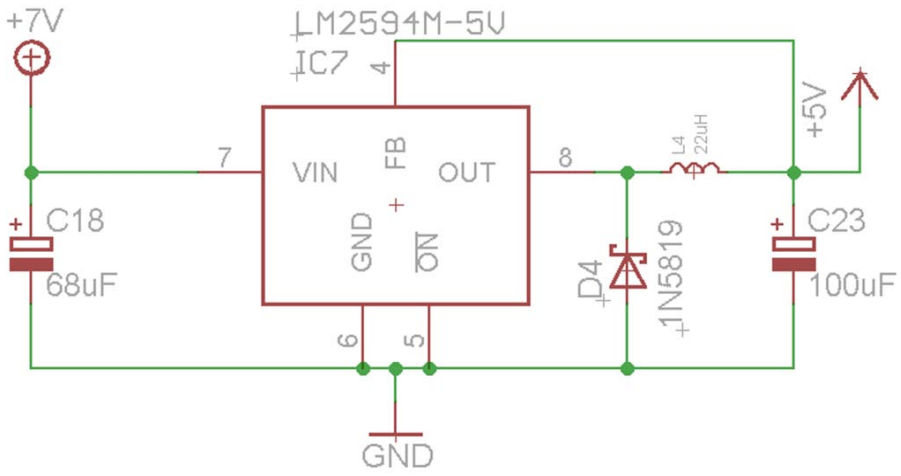


Figure 2.1 Connection Scheme of LM2594M-5V

The Integrated Circuit or IC is a Voltage Stabilizer with monolithic integrated circuits that provide all the functions of a step-down (buck) switch regulator, capable of driving 0.5A load with excellent line and load regulation. The input voltage can range from 45V to 5 Volts. The package it comes in is a Surface mount Device or SMD package with 8 pins with the following connections: pin1, pin2, pin3 are not connected or NC, but for safety reason are connected to the ground; pin4 is a feedback loop connection to the 5V output; pin5 and pin6 are connected to the ground or GND; pin 7 is connected to the input; pin8 is the output voltage;

For the correct functioning of the board and to reduce oscillations and similar effects the IC needs to be connected to a number of components. The first and last component is a 68uF and 100uF capacitance connected parallel between the input and the ground. For our application this is a Panasonic Surface Mount or SMD device that can handle up to 25 Volts, has a diameter of 6.3mm and the case type is D. This device is denoted by C18 and C23 on the figure 2.1.

The second component connected to the IC is a Shottky Diode connected parallel on the output of the IC with the ground, this is a SMD device that can withstand 60 Volts and 1 Amps of current that has the casing SOD123 and is produced by NXP. This device is denoted by D4 on the figure 2.1.

The third component is a wire based inductance that has the value of 22uH and can withstand 320mA load and has an internal resistance of 0.94Ohms, is and SMD device with the resonance frequency of 15Mhz and the dimensions 4.5x3.2x2.3mm produced by Murata. This device is denoted by L4 on the figure 2.1.

For the 3.3V voltage regulator the LM2574N-3.3V is used, the device differs from the 5V regulator due to the lack of stock from the distributor. The schematic of the connections and the adjacent capacitances and shottky diodes can be seen in figure 2.2.

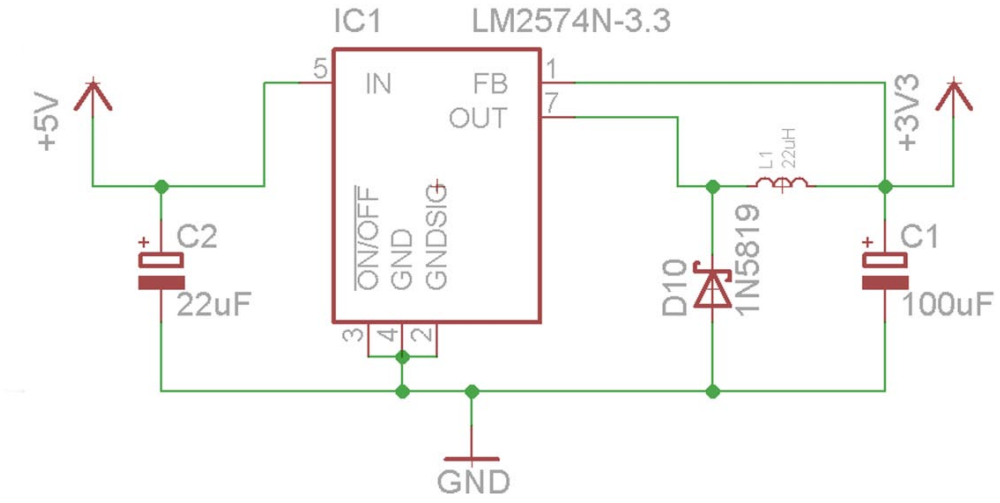


Figure 2.2 Connection Scheme of LM2574N-3.3V

The IC that is used is a product of Texas Instruments is a series regulator with monolithic integrated circuits that function like a step-down(buck) switch regulator, capable of driving 0.5A load with excellent line and load regulation. The input voltage can range from 40V to 3.3 Volts. The package it comes in is a through hole technology or THT package with 8 pins with the following connections:pin6 and pin8 are Not Connected; pin1 is the feedback loops input terminal; pin2, pin3,pin4 are connected to the ground activating it through the signal, turning it on and 4 being the ground; pin 5 is the input connection; pin 7 is the output connection.

This device as well has only 4 components connected to it in order to assure stability. The first and last component is a 22uF and 100uF capacitance connected parallel between the input and the ground. For our application this is a Panasonic Surface Mount or SMD device that can handle up to 25 Volts, has a diameter of 6.3mm and the case type is D. This device is denoted by C2 and C1 on the figure 2.2.

The second and third devices denoted by D10 and L1 are the same devices specified for the figure 2.1 by D4 and L4 and have the same specifications.

2.1.2 Status Light Emitting Diodes (LEDs)

The status leds are used when the PCB is not connected to the pc through the debugger and we still want to see whether connections are stable or if the board is held in reset or any faults or errors are detected.

The connection schemes of the two leds as well as the adjacent components are visible in figure 2.3.

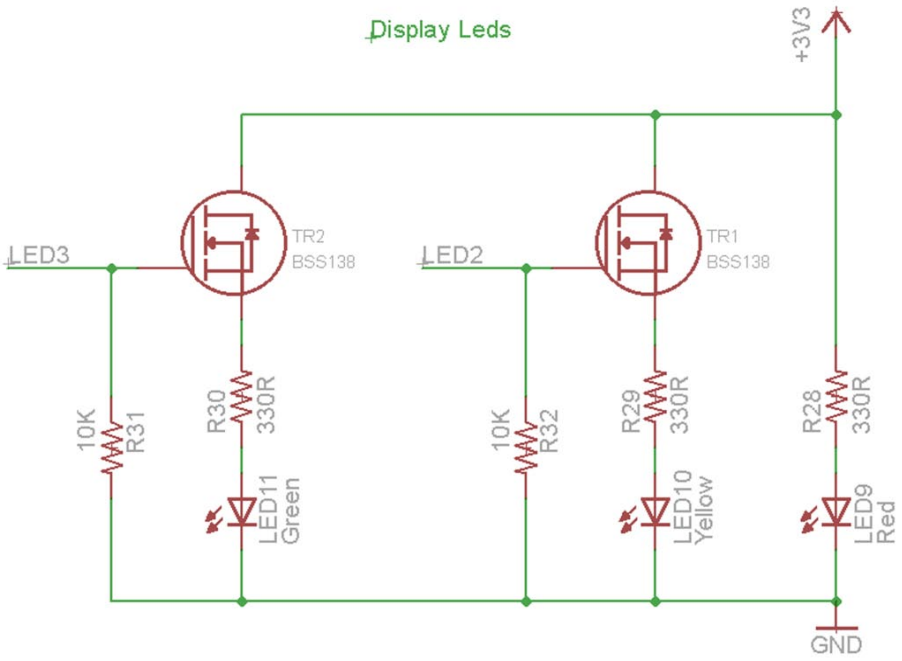


Figure 2.3 Connection Scheme for the Red, Green and Yellow Leds

There are slight differences between the three leads, the most obvious being their emission color but their general specifications are as follows: Surface Mount Devices; Casing 1206; The angle of the light is at 120º; Dimensions 3.2x1.6mm; Current for LED diode 20mA; Working Tension 2.2-2.5V;

The Red Led is used to signal if the board in under power and is directly connected to the output of the 3.3V voltage regulator and a series SMD resistance that is 330 Ohm in order to limit the current that crosses the Led. Using Ohms law we can simply calculate that the current through each led is a maximum of 10mA which is enough for the leds to glow but not enough to burn them out.

The operation of the Green and Yellow leds is more complicated then the operation of the Red one which is always on while under power. For simplicity purposes only the first led, the Green one will be explained the yellow led being identical in almost all features.

The Leds are controlled by the microcontroller B type outputs which might not be able to supply the leds with enough current to actually make them glow. In order to make sure enough current reaches the leds an N Mosfet semiconductor, labeled with TR1 and TR2, is inserted between the Led and control from the board. The Drain of the semiconductor is

connected to the 3.3 Volt line while the Gate is connected to the input from the board and the Sink is connected to a 330 Ohm SMD Resistance, labeled R29 and R30, which is connected in series with the LED itself. Another high value 10k Ohm resistance, labeled R31 and R28, is connected in parallel with the input and the ground in order to assure close to 0 Volts on the Gate when no command is given and is called the Pull Down. The Red, Green and Yellow leds are labeled in order as LED9, LED10 and LED11 on the figure 2.3.

2.1.3 Decoupling Capacitances

The role of decoupling capacitances is that of decoupling a part of the electrical circuit from another, these are mostly used for safety reasons and noise cancelation. These are also called bypass capacitors as they are sometimes directly connected to the power supply or other high impedance components on the circuit.

The connection scheme is visible in figure 2.4, when the actual connection are made on the board these capacitances are connected as closely as possible to the device they are intended to shield in order to improve their effect. Most of the other circuits have decoupling capacitances as well, either incorporated or added afterwards in concordance to the datasheet requirements.

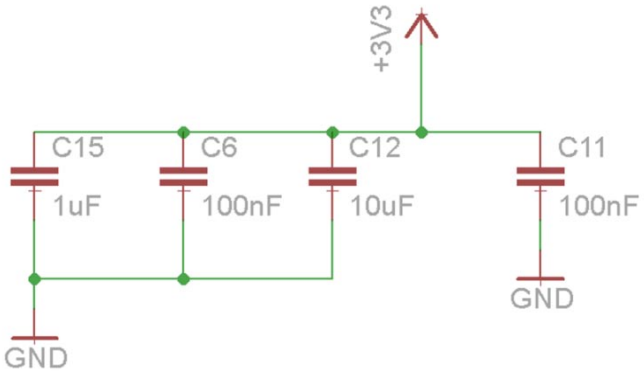


Figure 2.4 Connection Scheme for the Decoupling Capacitances

We have two 100nF capacitances labeled with C6 and C11 connected to different parts of the microcontroller to reduce noise at those parts, as well as a 1uF and 10uF capacitances, labeled C15 and C12 that are used to protect against noise at different frequencies.

2.1.4 Oscillator

The microcontrollers has an internal clock that has a maximum speed of 16 MHz, but in order to use the full capacity of the microcontroller a 20 MHz Oscillator is connected. This is the maximum value allowed in the datasheet. The connection and routing is in figure 2.5:

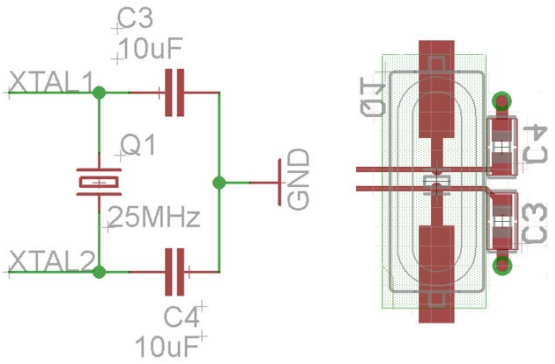


Figure 2.5 Oscillator Configuration Scheme and Routing

The oscillator for the microcontroller is a 20 MHz SMD Quartz resonator, one of the most important parts is how the connection between the microcontroller and the quartz is made and where are the capacitances located. As we can see from routing in figure 2.5 the adjacent decoupling capacitances for noise reduction are connected as closely as possible and the ground wires follow a path suggested in the data sheet.

The oscillator is denoted by Q1 and is connected in parallel between the pins of the microcontroller XTAL1 and XTAL2, the capacitances are SMD capacitances with the values 10uF and the labels C3 and C4.

2.1.5 Headers and Connectors

The PCB needs a range of Headers and input, output terminals to be able to control the motors, to receive current from the battery as well to communicate with the Raspberry pi, with the Bluetooth module, debugging header and programmer. All of these are crucial to the workings of the board.

The three most basic headers of the PCB are the power, Bluetooth, and debugging, these are presented in figure 2.6.

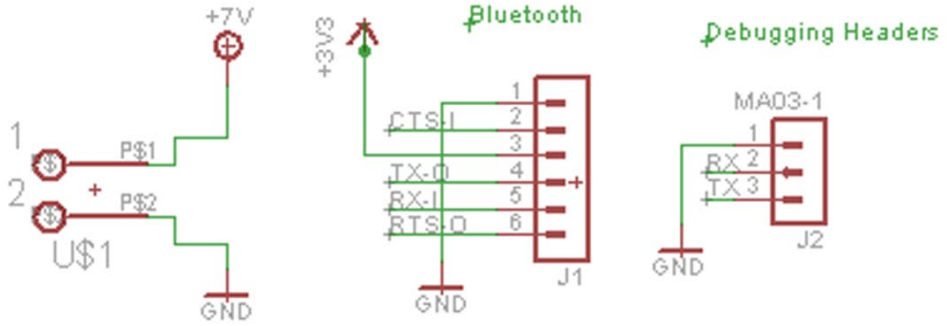


Figure 2.6 Power, Bluetooth and Debugging Headers

The power header is connected to the ground on the terminal 2 and to the 7Volt input on the 1-st terminal. The physical connection is done through an angled Terminal block which has 2.5mm spacing and is 5mm wide, with a height of 10.9 mm, and two connections, having the connection method to the board using THT and screw terminals.

The connection for the debugging headers is done with a pin strips male header that is gold plated and is soldered to the board using THT technology. The Bluetooth board requires 3.3 Volt power and an UART connection with separate pins for Reset and Clear selectors that can be used to access multiple entities on the same UART channel. This device is connected to the first UART Channel output of the board.

The connection to the debugging header is done with the same pin strips at the previous one, with the exception that this one has 3 pins. The connection is a UART connection with no extra commands. The debugger is connected to the second UART Channel.

A more complex Header is the Programming Header which is used to program the board using the Microchip PICkit3 that allows the rewriting o the flash memory and debugging features as well.

The connection of the header as well as the extra components required for such the programmer to function properly is mentioned in the Datasheet as well. The exact connection can be seen in figure 2.7 where we can see that the programmer is connected to the 3.3V grid, this is done so that the board may be powered by the Programmer, this would only allow for a limited number of sensors to work and is mostly used only to be able to program the board and not for tests.

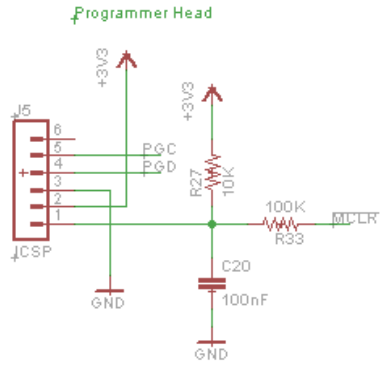


Figure 2.7 Schematics for the Programmer Headers

The PG-C and PG-D are allocated pins for the programming and cannot be used for other operations. The MCCR pin is used to reset the microcontroller, in certain cases reset a reset button may be attached to the device. For the Reset pin a number of required components are mentioned in the datasheet. We need to have a 10K Ohm pull up resistance connected parallel to the pin which is labeled with R27 as well as a 100K Ohm resistance connected in series between the pin and the MCCR pin of the microcontroller, which is labeled with R33. Furthermore a 100nF Capacitance, labeled C20, is required connected parallel between the gnd and the MCCR to reduce any noise on the channel.

The GPIO pins are used to connect the PCB to the Raspberry pi, in other applications where the connection is not required and communication is done through the Bluetooth Header these might be used to read the values from other advanced sensors, that can communicate through the spi. The schematics of the GPIO pins can be seen in figure 2.8.

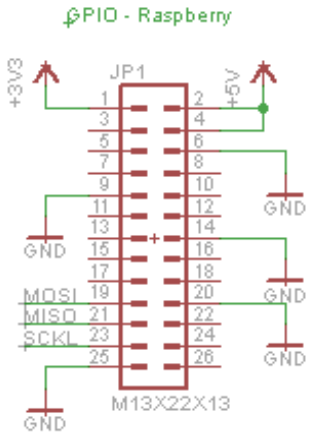


Figure 2.8 Schematics for the GPIO connection

The GPIO pins are connected both to the 3.3V grid and the 5V grid, this is done with the idea that other devices may be connected then the Raspberry pi. When working with the Raspberry pi the board is connected through a butterfly cable for which the power couplings are cut in order to protect both the board and the Raspberry pi. This is done because the raspberry pi doesn't have any protection when receiving power from the GPIO pins like it has when it's powered through the micro Usb where a polyfuse is put in place that protects it. Furthermore if there are any differences between the two voltages inverse currents might start to flow in one or the other direction.

The pins MOIS MISO and SCKL are used for the SPI communication and are connected to the first SPI channel of the microcontroller. These pins do not require any other safety protection.

2.1.6 *Digital Hall Effect Sensors*

The Hall Effect sensors operate sensing the magnetic field around them. The output of these devices switches from low to high when a south polarity magnetic field perpendicular to the Hall element exceeds the operate point threshold. After turn-on, the output is capable of sinking 25 mA and the output voltage remains the same. When the magnetic field is reduced below the release point, the device switches to low again. The difference in the magnetic field is the hysteresis of the system. This built-in hysteresis allows clean switching of the output, even in the presence of external mechanical vibration and electrical noise.

The connection of the Digital Hall sensors can be seen in figure 2.9.

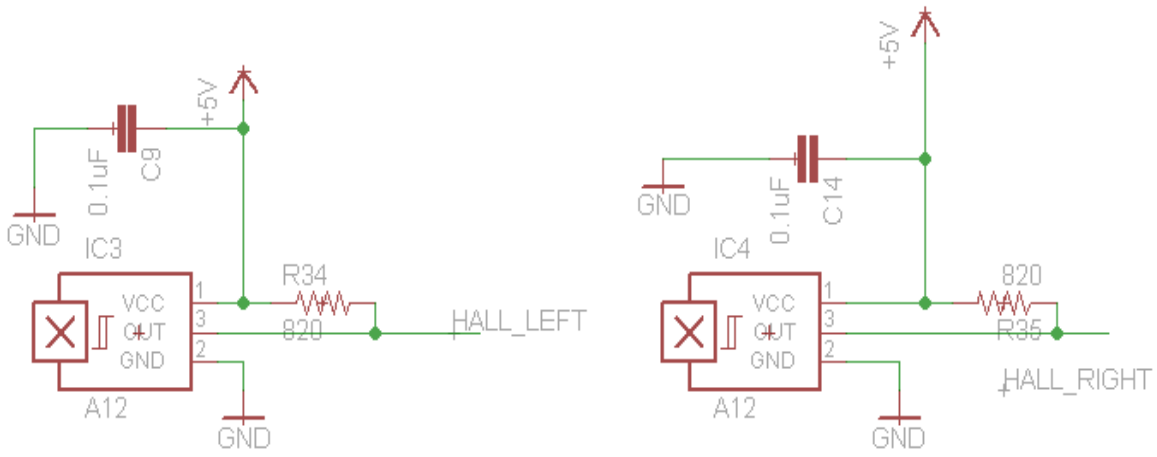


Figure 2.9 Connection Scheme for the Hall Effect Sensors

The outputs of the sensors should be connected to the external interrupts of the microcontroller in order to be able to measure the time between two commutations of the sensor. These sensors can also be used to measure distance, counting the pulses, and calculating the distance from that set of data. If both the sensors are connected correctly and we consider that there is no slip of the wheels it is also possible to calculate the rotation of the wheels and get the state of the robot from equation (1.3).

The IC device selected is the Allegro A1101LUA-T which comes in a THT mode with a SIP3 Casing having the supply voltage range between 5V-30V and giving a 25mA current when a digital high signal is given.

In order for the sensors to function optimally the datasheet the sensors require a noise cancelling 0.1uF Capacitance, labeled C9 and C14, added to the system. This capacitance is parallel with the 5V input and the GND. Furthermore, the sensor requires a parallel pull up resistance labeled R34 and R35, that is used to make sure the output has the value of the VCC is a digital high signal is given.

2.1.7 *Digital Infra Red Proximity and Line following Sensors*

The role of proximity sensors is to detect the presence of objects in the close vicinity of the sensor without needing physical contact. The proximity and the line following sensor work on the same principle, with the difference that the line following sensor will be at the same distance from the objects, just the color of the object will change, the color of the object affects the returning infrared light and thus changing its value.

The idea of a digital Proximity or line following sensor is that we are only interested if the returning beam's intensity is above or below certain parameters, in our case this translates to a certain distance and a certain color for the line following sensors.

The main concept of the IR and Line following sensors is based on using the receiver that has a varying resistance depending on the light received as a voltage divider and comparing the resulting voltage with a reference voltage with the help of an operational amplifier. Due to the construction of the Op-Amps in smd casing two sensors are grouped together into one case and have the same reference. To exemplify the working of these sensors the Front proximity sensors schematic can be viewed in figure 2.10:

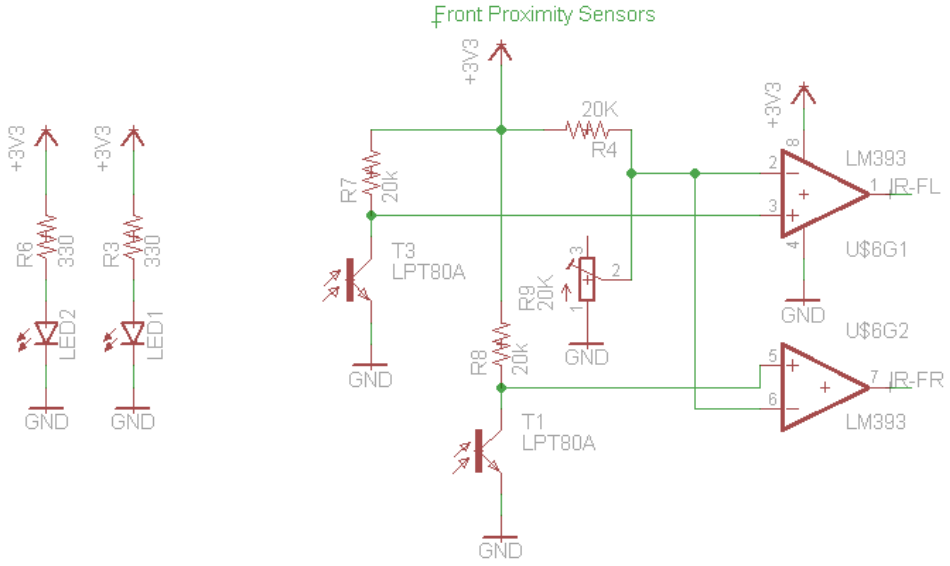


Figure 2.10 Schematic for the Front Proximity Sensors

Operational amplifiers or op-amps are DC-coupled high-gain electronic voltage amplifiers with a differential input and usually a single ended output. This configuration allows it to have an output potential that is thousands of times larger than the potential difference between the input terminals. In our case the op-amp is basically a comparator, which compares the non-inverting input to the inverting input and if the non-inverting is higher than the positive reference voltage is given on the output, if the inverting input is higher than the negative reference is given on the output.

Both the inverting and the non-inverting inputs of the op-ams are given as a result of two voltage dividers. The basic algorithm for calculating the output of a voltage divider can be seen in the equation (2.1).

$$V_{out} = \frac{R_{sensor}}{R_{sensor} + R_{static}} \cdot V_{reference} \tag{2.1}$$

The inverting input for both the op-amps is a reference generated by giving the voltage divider the $R_{Reference}$ as the 3.3V generated by the voltage stabilizer, the value for the $R_{sesnors}$ is a THT trim resistance, having it's resistance variable from 0 to 20kOhm and the value for R_{static} is a 20k Ohm SMD resistance. If we take into account the formulas presented in 2.1 we can see that our inverting input can vary from 1.65V to 3.3 V depending on the state of the trim resistance.

The non-inverting input for the op-amps is generated using the varying resistance property of the infrared sensor. This device changes its interior resistance based on the light it receives. In our case the original schematic had to be modified to use a 80kOhm resistance rather then a 20k Ohm resistance as it's shown in 2.10. The values of the Infrared sensor in normal conditions goes from a value of 150k Ohm to as low as 1K Ohm when in direct line of the infrared light coming from the infrared emitting Led. Taking these values into account we can see that the values for the non inverting input can vary from 2.2Volts to 0.1v or lower, which are considered as 0V.

The other part of this system is the Infrared LED or IRLED which is put in close proximity to the receiver and is connected in series with a 330 Ohm resistance and powered from the 3.3V terminal.

If we take the parts into consideration the labeling is as follows: U$6G1 and G2 are part of the same IC device and represent the operational amplifier having IR-FL and IR-FR being the digital signal sent to the microcontroller from the front left and front right sensors; R9 is the trim resistance; R4 is the static resistance in the inverting inputs voltage divider having the value 20kOhm; R7 and R8 are the two fixed resistance connected to the non-inverting input of the voltage dividers for the actual sensors; T3 and T1 are THT base infrared sensors;

Finally the working of the digital proximity sensor is as follows: The output is logical 1 if there is nothing in the proximity of the sensor; The output is logical 0 if the light threshold is breached or there is something in the proximity of the device. The Line following sensors work in much the same way: The output is logical 1 if the surface is sees is black; The output is logical 0 if the surface is white. The problem with the line following sensors is that they need precise calibration, the difference between black and white only being about 0.4Volts.

2.1.8 H bridge Connections

The H bridge is an electronic circuit that is mostly used in a wide range of applications from DC to AC converters or push-pull converters, stepper motor drivers and in our case in driving two DC motors. The H bridge we use is an L298 Dual H bridge Driver. The basic operation of the H bridge for our DC motor control is based on the enable pin and the inputs, the connection of these can be seen in table 2.1

Table 2-1 H bridge Control

Enable	Input1	Input2	Result
Off	X	X	Motor is Decoupled
On	Off	Off	Full Stop
On	Off	On	Rotate Left
On	On	Off	Full Stop
On	On	On	Rotate Right

The Control of the speed of motors is done through the Enable pins of the H bridge, it is possible to control the speed by turning on and off the direction from full stop to a direction but this is not practical and better control can be attained this way. A schematic of the connections is presented in figure 2.11:

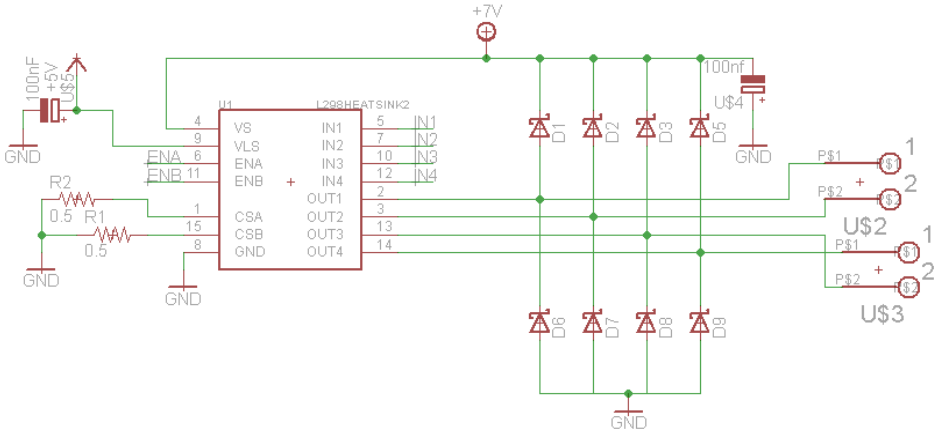


Figure 2.11 H Bridge Driver Connections

The U$5 100nF capacitance is routed as a decoupling capacitance to the 5V reference of the H bridge driver, The CSA and CSB tie the motor current sense to the ground through 0.5 Ohm Resistances R1 and R2. The output of the Driver is decoupled against noise using U$4 100nF capacitance and over and undercurrent protection is given using the D1-D9 Shottky Diodes. This is done to protect the driver and the system. An additional Heat-sink is required due to the high current required by the motors. This heatsink is a standard one used for this driver allowing it to work with 2Amps of current passing through it.

For the PWM control of the system we have to take into account some factors from the data-sheet that say that the highest frequency that is allowed for the commutation of the Enable pins is 20kHz.

2.1.9 Microcontroller and the Connections

The microcontroller used for this pcb is a PIC18f48k80 with 8-bit MCU with Integrated ECAN. The Operating Voltage Range is between 1.8V and 5.5V with an On-Chip 3.3V Regulator, it has an Operating Speed up to 64 MHz with up to 64 Kbytes On-Chip Flash Program Memory, a typical of 10,000 erase/write cycle, 1,024 Bytes of Data EEPROM, as well as 3.6 Kbytes of General Purpose Registers , Three Internal Oscillators: LF-INTOSC (31 KHz), MF-INTOSC (500 kHz) and HF-INTOSC (16 MHz), Self-Programmable under Software Control, priority Levels for Interrupts, Extended Watchdog Timer (WDT) with Programmable period from 4 ms to 4,194s, In-Circuit Serial Programming™ (ICSP™) via Two Pins and In-Circuit Debug via the same Two Pins. The microcontroller is also equipped with multiple MSSP modules, ADC-s, Comparators and UART channels.

The connection scheme of the microcontroller can be seen is figure 2.11:

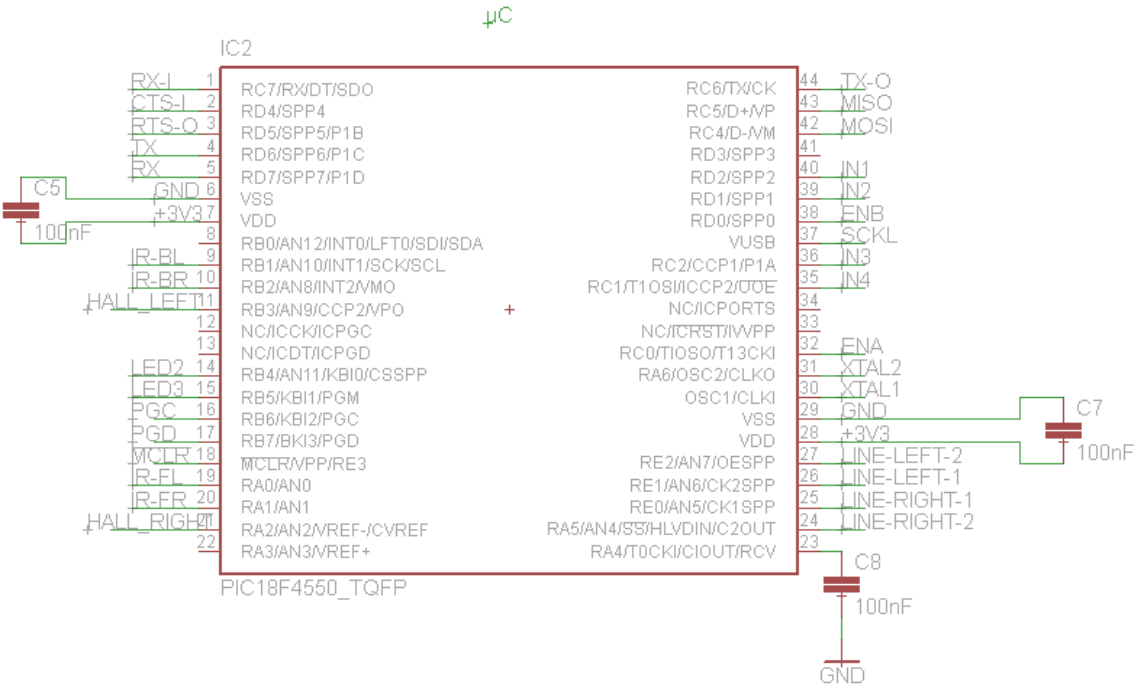


Figure 2.12 connections to the microcontroller

The microcontroller has decoupling capacitances on the power supplies both on pins 6, 7 and pins 28, 29. The microcontroller also has a 100nF capacitance connected to the ground and the RCV pin which is used for synchronization in case of multiple systems and a master slave connection.

The table 2.1 shows the function and connection of each pin, to the adjacent figure:

Table 2-2 function and connection of each Label

Pin Nr.	Label	Function on Microcontroller	Function on System	Adjacent Figure
44.	TX-O	UART1 connection	Bluetooth Communication	2.6
1.	RX-I	UART1 connection	Bluetooth Communication	2.6
2.	CTS-I	Digital Input	Bluetooth Communication	2.6
3.	RTS-O	Digital Output	Bluetooth Communication	2.6
4.	TX	UART2 connection	Debugging	2.6
5.	RX	UART2 connection	Debugging	2.6
9.	IR-BL	Digital Input	Infrared Sensor - Back Left	2.10
10.	IR-BR	Digital Input	Infrared Sensor - Back Right	2.10
19.	IR-FL	Digital Input	Infrared Sensor - Front Left	2.10
20.	IR-FR	Digital Input	Infrared Sensor - Front Right	2.10
11.	HALL_LEFT	External Interrupt	Hall Effect Sensor - Left	2.9
21.	HALL_RIGHT	External Interrupt	Hall Effect Sensor - Right	2.9
14.	LED2	Digital Output	Orange LED control	2.3
15.	LED3	Digital Output	Green LED control	2.3
16.	PGD	Programming pin	Programmer input	2.7
17.	PGC	Programming pin	Programmer input	2.7
18.	MCCR	Reset Pin	Programmer input	2.7
24.	LINE-RIGHT2	Digital Input	Line Sensor - Right2	2.10
25.	LINE-RIGHT1	Digital Input	Line Sensor - Right1	2.10
26.	LINE-LEFT1	Digital Input	Line Sensor - Left1	2.10
27.	LINE-LEFT2	Digital Input	Line Sensor - Left2	2.10
30.	XTAL1	External Oscilator	Oscilator	2.5
31.	XTAL2	External Oscilator	Oscilator	2.5
32.	ENA	PWM output	H bridge Enable Mot1	2.11
38.	ENB	PWM output	H bridge Enable Mot2	2.11
40.	IN1	Digital Output	H bridge direction Mot1	2.11
39.	IN2	Digital Output	H bridge direction Mot1	2.11
36.	IN3	Digital Output	H bridge direction Mot2	2.11
35.	IN4	Digital Output	H bridge direction Mot2	2.11
37.	SCKL	MSSPI In/Out	Clock for SPI	2.8
42.	MOSI	Master Out Slave In	Output for SPI	2.8
43.	MISO	Master In Slave Out	Input for spi	2.8

A full schematic and the routing for the system is available at the annexes part of the paper.

2.2 Configuration of the Microcontroller

Programming the microcontroller was done using the MPLAB X IDE provided by the Microchip Corporation and PICKit 3 programmer from the same company. The programming itself can be done from multiple IDE-s, even Eclipse and similar, the only thing required is the compiler. The advantages of using their IDE is that you have Debugging features and certain prebuilt configurations like auto configuration file builder.

All the information needed to program the microcontroller as well as the connections and the configuration of the Registers is presented in the datasheet [11]. The programming of the microcontroller was done in Embedded C using the microcontroller, uart.h and delay.h library that was available.

The programming of the microcontroller was grouped into 3 main files. The hardware configuration, where the input, output and configuration registers are given names and grouped, as well as containing static variables used throughout the system is available in the annexes of the thesis. The board file contains all the functions that are used to control the microcontroller as well as the functions needed for communication. The Main file only has the includes from the board.h file that already has all the includes required for the system and this way of programming is in concordance with the standers for programming in Embedded C.

2.2.1 Basic Configurations

The full configuration of the board is visible in the appendix on the main file. The explanation of how and why certain registers are set how they are set is the following.

In order to have to microcontroller work properly we need to adjust certain parameters of the config files to suit our needs and the electric hook-up of the system. The Watchdog Timer having the Register WDTEN needs to be turned off, because we are not using it, and it would reset the system. In order to use certain inputs we need to turn the external oscillator input into a Digital pin, this is done by setting SOSCSEL to DIG which is digital. For the Crystal settings we need to set the FOSC register to HS2 which is the high speed external oscillator we have connected. IESO and PLLCFG disable the divider that is available on the input and guarantees 20 MHz Clock signal. The last configuration we have to set is the XINST register that we need to turn off, in order to not work with the special instruction set.

2.2.2 UART configuration and use

The universal asynchronous receiver/transmitter or UART takes bytes of data, splits them up into bits in a sequential manner and sends them over a channel. This is done because serial transmission of data through one wire is less costly than using multiple wires. The receiver assembles the bits into a byte and makes it available for the user usually triggering an interrupt.

The UART communication is mainly useful for two things, on the first channel the microcontroller communicates with a Bluetooth Silver board that allows it to communicate through Bluetooth with other board of the same type. The second use is through the second UART channel that is used for debugging. Debugging is especially important to see data and information while the system is running, this function can be bypassed using the programmer but it's easier to work with it this way.

The configurations for the Channels require the usart.h library to be included. The basic settings require us to set the TX pin as an output and the RX pin for both channels as an input. This is done by writing 0 on the register for output and 1 for the input.

In order to have more control over the UART port we set the BRG16 bit of the BAUDCON bits to 1 in order to have a 16 bit register that works the UART channel divider. We then need to open the usart port using Open2USART giving as inputs the parameters: Turn of interrupts on rx, turn off interrupts on tx; use usart in asynch mode; the message sent is 8 bits long; set the BRGH bit as high to enable the long register for the divider. The final setting is the delay that we need to calculate from the baud rate of the microcontroller frequency. This is done as seen in equation (2.2). For this equation we need to know that for 4 clock signals the microcontroller can do one execution cycle so we consider that a 20Mhz microcontroller has a 5 Mhz execution cycle, we then take the execution cycle and divide it with the baud rate and subtract 1 for the start bit. The resulting equation is the one below.

$$Delay_{UART} = \frac{Frequency\ \ uC}{4 \cdot Baud\ \ Rate} - 1 \qquad (2.2)$$

Using the equation we can calculate that for the baud rates 9600,19200 and 115000 we get a delay of 520, 259 and 42. These are used as input for the opening function. The errors on the timing for the uart communication for 9600,19200 and 115000 is 0.032%, 0.16% and 0.937%. This is due to the fact that the baud rate is not exactly the value mentioned above.

Messages through the UART channels can be sent in two ways, the first one uses the function putsXUsart() where the X is replaced with the channel number, and the input can be any string directly added as an input. The second way of writing to the uart where we can actually insert variables is using a character buffer, sprintf and putrsXUsart in the same way, just adding the character buffer as input. If these functions are not used correctly the receiver will receive gibberish.

2.2.3 Proxy and Line follower and LEDs

The Settings for these digital inputs/outputs are usually very simple it's a matter of setting the right value for the tris configuration 1 for input 0 for output after this we can either read the value using the port registers and write values using the lat registers.

This first step is simple but some pins have higher functions like comparator or analog digital converters or any other function even UART or external clock connections. In these cases we need to disable these options and make sure that the pins are set to be used as digital inputs.

2.2.4 Hall Sensors the Timer and Interrupts

The Hall sensors work by giving a pulse like signal each time the magnet passes in front of the sensor. In order to capture this data it is necessary to be able to react to the spike as soon as it happens and it's not acceptable to have the microcontroller always watching the input. The solution is having the spike be the input for an external interrupt.

An interrupt is defined in the literature as being a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor that it needs to interrupt the current routine and enter a ISR or interrupt service routine. In the microcontroller used here there are two levels of interrupts high level and low level interrupts, this is important because certain interrupts might need to be handled before others.

In order to initialize the sensor inputs and make them work as digital inputs we only need to set their tris bits and disable one connected analog to digital converter. In order to enable high and low interrupts we need to set the interrupt enable registers. In order to set the external interrupts for the device we need to enable them and set the edge to falling, select them as high priority interrupt and finally reset the register. In order to read an interrupt after it has been read the interrupt flag needs to be reset in order to be able to read other interrupt. When the interrupt is triggered we can increment the distance variable and read the speed by reading the time delay between two passes.

The interrupts on the inputs allow us to capture the distance the device took but in order to see the speed of the device we need to have a timer that gets reset every time the flag is triggered. In order to use the timer first we have to enable it, after that we need to set the type of the timer which can be 16 bit or 8 bit, depending on the precision we want and if the prescalar is enough. Furthermore we need to set the clock for the timer to be the internal clock, as well as the prescalar to be enabled. The prescalar can be a value between 4 and 256 with the steps being of the power two. In our case the 256 prescalar is not enough so we have to use a little trick and only use the top 8 bits of the timer register so we get another 256 prescalar. The resulting prescalar allows us to have a precision of 12ms when counting the time, usually having values around 260, 440ms for the time between pulses generated.

In addition the reading of the valued of the timer is important because the timer has multiple buffers, when the top value of the buffer is read the bottom half is saved at the same

time making sure the value is correct. When the bottom value is read as well, the timer is reset and the buffers are cleared.

2.2.5 SPI configuration and Interrupt

SPI or Serial Peripheral Interface is one of the key elements in the workings of the microcontroller, being the part that communicates with the raspberry pi. This bus is a synchronous serial communication interface specification used primarily in embedded systems for short distances. SPI devices can communicate using full duplex mode in a master-slave architecture where the master originates the frame for reading and writing. Multiple slave devices can be connected to the same Master using the slave select control. In our application there is only one slave and one master the microcontroller being the slave and the raspberry pi being the master. An example of how these devices and how the devices are connected in real life is visible in figure 2.13:

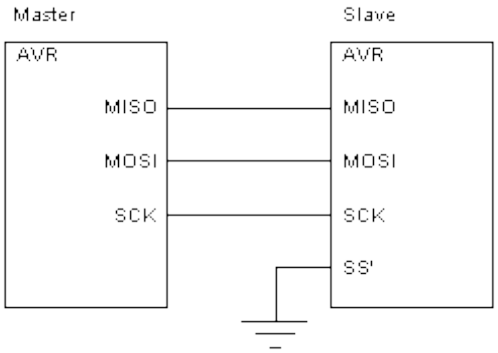


Figure 2.13 SPI Connection

The SPI is considered a four-wired serial bus considering the selection bit. The microcontroller has two Master Synchronous Serial Ports MSSP modules, these allow for SPI and I2C communication with other devices depending on which we want to use.

The final form of a message sent on the channel can be seen in figure 2.14 from [11].

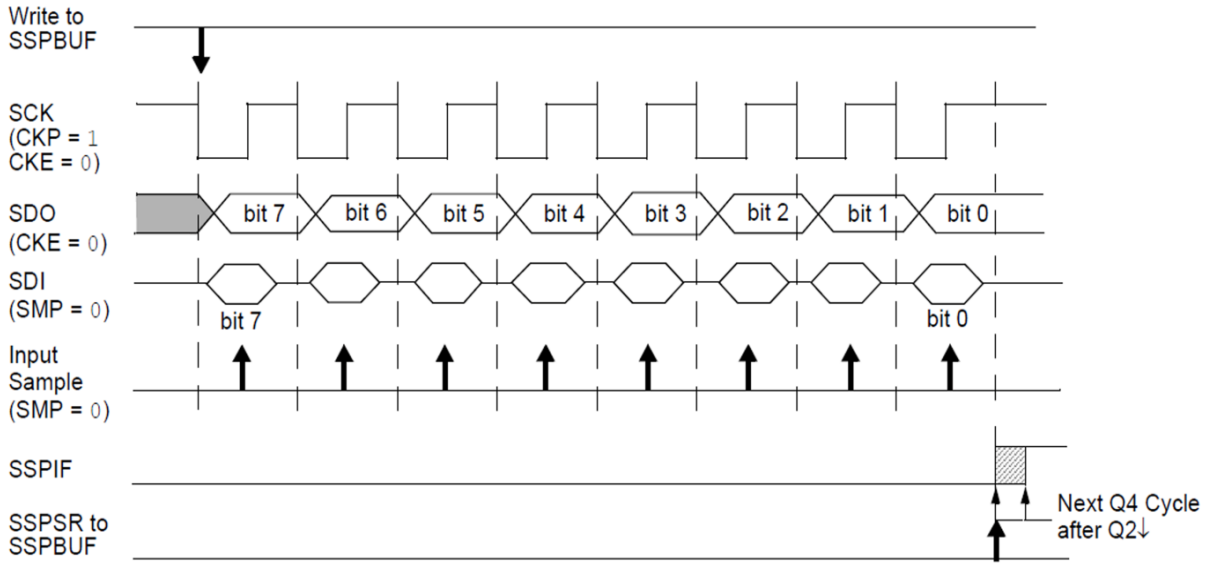


Figure 2.14 SPI Waveform

For the correct use of the SPI ports we need to configure the The MOSI as an output and SCK and MISO as an input. Furthermore we need to configure an array of registers, enabling the channel, overflow and other register, but out of which the most important being

the SMP, CKE and CKP registers and SPI communication type register. The SMP register which needs to be cleared in slave mode. The CKE register which is the clock select bit, that sets whether the data is sampled on the rising edge or falling edge. The CKP bit that is the clock polarity bit which sets the idle time of the SPI. The final register that we need to set is the SSPM register which selects the Port mode. There are only two Slave modes, with the difference that in one of them the SS pin can be used as a Digital I/O, which is essential to us.

2.2.6 Motor control configuration

The control of the motors is done it two ways, the direction as well as the stop is controlled by the input1, 2, 3 and 4 these are set by setting the specific tris registers as 0 making the outputs, these do not need to be pwm outputs. In order for them to work properly the adjacent analog to digital converters, comparators and oscillator input needs to be turned off to allow digital use of the pins.

The control of the speed of the motors should be done using PWM outputs, due to errors in the design of the connections the outputs assigned to the H bridge enable bits that should run the speed modulation are only digital pins. A solution is mentioned in the Programming part that involves bit-banging.

2.3 Programs and Protocols on the Microcontroller

The programming and hence the protocols on the microcontroller are designed in such a way that the microcontroller can receive commands from the raspberry pi through the spi a and run the right subroutines depending on the commands. This is done by having routines that run one instance and gets rerun depending on the variables.

These functions are made in such a way that their running time takes the absolute minimum required time to run.

2.3.1 Motors Control Sequences

All control options for the motor are in the board.c file having their headers and explanation on what is set how in the board.h, the values of certain delays and variables is set in the HardwareProfile.h file.

The board controls can be placed into 3 categories, the basic controls, pwm and Positioning controls. Using these controls, in theory the 4 functionalities of the mobile robot can be fulfilled.

Table 2-3 Motor control and functions

EnableA	EnableB	Input1	Input2	Input3	Input 4	Function Name
0	0	0	0	0	0	InitMotors()
0	0	0	0	0	0	MotorsReset()
X	X	1	0	0	1	MotorsRight()
X	X	0	1	1	0	MotorsLeft()
X	X	1	0	1	0	MotorsBack()
X	X	0	1	0	1	MotorsForward()
0	0	X	X	X	X	MotorsStop()
1	1	X	X	X	X	MotorsFullSpeed()

The basic control for the motor includes all the possible direction and the full enabling of the motors without the use of PWM. These functions are two kinds. The first type sets the direction of the car. In order to understand how these work we need to consider the workings of the workings of the H Bridge presented in 2.1. The motors are connected to the board is such a fashion that if we put a high value on the even number inputs will make the

corresponding motors turn forward, for example if we want to have both motors turn Forward, input 2 and input 4 need to have a positive value. The exact values and the name of the function that work these values can be seen in table 2.3

The control of the basic function in the pwm is done in such a way that allows the user to control each aspect of the motors separately the direction and the activation of the motor. In such a way it is possible to use a later defined pwm function to control rotating movements speed and other controls.

The pwm control function is only based on controlling the Enable bits, which are designed for pwm. The name of the actual function is MotorsDiff() and has as input a char value ranging from 0 to 255 which is the extra delay. This function is a bit banging version of pwm, because the pins are not connected to digital outputs as they should be, this is an improvisation. The way this function works is that it runts a cycle 20 times, the cycle itself puts the enable bits to a high value waits for a minimum delay specified in the hardware files and then waits the delay time specified in the input and afterwards turns off the enable bits and waits for the Rest of the time minus the delay before restarting.

It is important to make sure the length of the pwm cycle is the same each time. The period of the function is 2ms or a frequency of 500Hz, where the minimum duty cycle is 1.15ms which is needed because a voltage value below that will make the motor enter a state where it could do damage to itself and the system and burn out. A smaller period would not allow the H Bridge to commute properly and would have the result of the motors twitching.

The third set of functionalities that the motor has is the positioning. Because the distance between the magnets on the wheel is pretty high and there are errors due to this and also due to the fact that there is a slip of the wheels when the robot is moving it is best to have the robot rotate an amount dictated by time. There are two types of rotations the robot can do, clockwise and counter-clockwise, both of these function have as input a count that specifies how many rotations of 90 degrees they have to do.

2.3.2 Safety Procedures and Debugging

The Safety procedures in place are mostly based on the proximity sensors placed on the front and back of the board. These procedures can be enabled and disabled based on a safety variable that is controllable through the spi port. If the safety is turned on, before running the main function routine the device tests whether any of the proximity sensors are triggered.

In the loop, with the direct controls that will be described more in detail in section 2.3.5, the safety simply makes the robot go forward for 0.3s if the back proxy is triggered and the go into the stop routine or make the car go backward for 0.3s if the front proxy is triggered and the go into the stop routine as well.

In the case of the second routine because the main idea is to reach certain positions and this is done without obstacle avoidance algorithms it is important to not continue in the same direction that caused the problem because it would do the same thing, and it's important in both cases to give feedback to the user, but in this part it's crucial. This protocol has the mobile robot always moving forward in order not to render the camera useless. Taking this into account, we only check if the front proximity sensors sense anything. If the safety protocol is triggered the motor will go backward for 0.2s, the status of the position is changed to an error and the required positions are set to the current position all of these things informing the controller that an error had occurred. The start_stop variable is disabled as well not allowing the command for the new location to be sent without an additional option of starting the movement.

The safety procedures can be shut of and sometimes they need to be shut of depending on the sensors that are very sensitive to light and might cause errors giving back positive values when there is no object in the area.

2.3.3 Hall sensors Interrupt Routine

The Hall sensor is very important part of the first two routines, for the first it's the one that gives the speed of the device and for the second, its the one that can calculate the distance travelled. The interrupt routine is triggered when a transition from low to high is made and in that point INT3IF flag is triggered and the high_isr() routine is started.

Whenever this routine is started the first thing that has to be done is to clear the flag to allow it to be triggered again, afterwards the high and low bits of the timer are read clearing them and the value of the high byte is saved and can be queried by the SPI module. After this, the problem of calculating the position is done by only incrementing the position if the motors are enabled and they are either moving forwards or backwards, we do not need to worry about the direction at this point. Furthermore, this functionality only comes into play if we are in the positioning functionality of the system.

If the interrupt occurs in one of these conditions then we take into account the current direction variable of the system that is changed when in positioning part turn clockwise or counter-clockwise. Depending on the current direction values to the current position in system x and y are added or subtracted.

2.3.4 Spi Communication using Interrupt Routines

The whole spi communication is based on interrupts. To be more specific, when a character is sent through the spi a flag, the SSPIF or the Synchronous Serial Port Interrupt Flag is set to 1 and a service routine is started depending on the settings low or high, in our case the spi communication is on the low priority interrupts list having the function low_isr() mange this interrupt. When the interrupt is triggered the interrupt bit is reset and the Green led is turned on in order for us to know that the first communication was made. The SSPIF flag is triggered when the master has either finished reading or writing values from the SPI channel, due to this it's possible to add values you want to send to the spi mutually and have an exchange in one clock where the values of the buffers are exchanged. An example of how a simple and a complicated message is sent can be seen in figure 2.15.

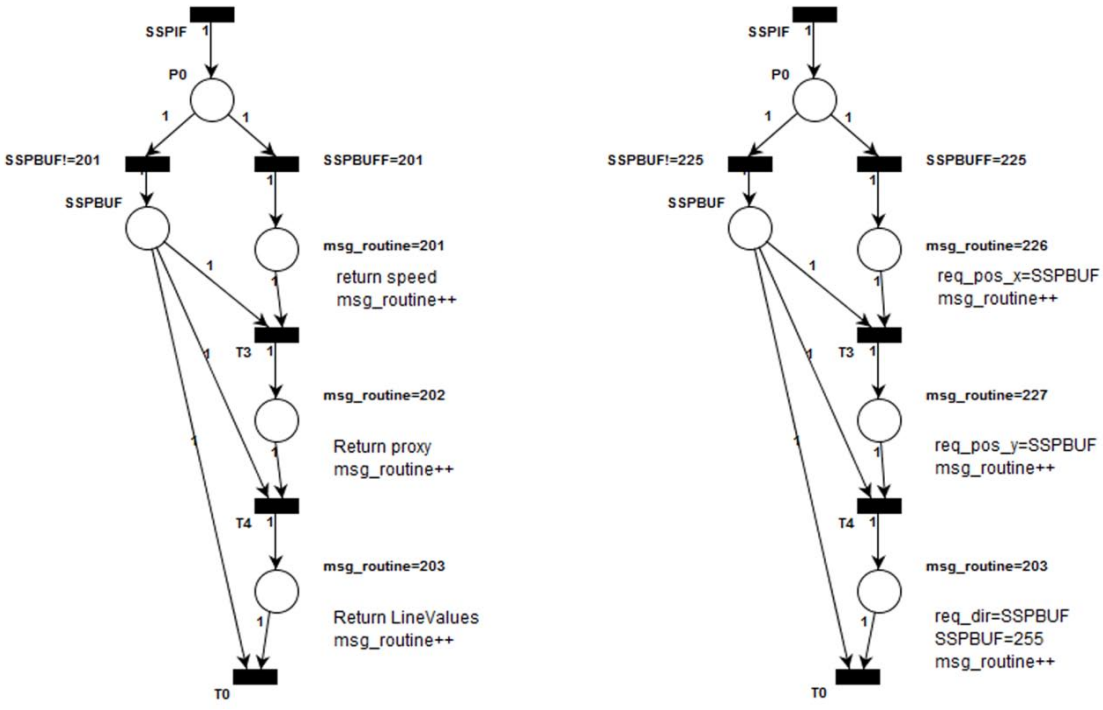


Figure 2.15 HLPN example of SPI communication

The 2.15 example is mentioned for two cases the first is when the sensor data is required from the proxy and the second is when the controller sends the new position data and requires a confirmation.

The request of sensor value starts by receiving the value 201 on the SSPBUF register. If the microcontroller receives this value it sets the msg_routine value to 201, this starts the routine that sends the values it requires, as the routine is started the values of the speed variable is written to the SSPBUF and the msg_routine is incremented. At the next setting of the SSPIF flag the speed variable is read by the raspberry pi. After this is done, the value of the proxy sensors shifted with 1,2,3 and 4 bits is written to the SSPBUF. Finally the value of the line following sensors is written to the SSPBUF in the same way.

The command of a new position the microcontroller has to travel to is done in a similar way as the sensors, with the difference that here the microcontroller receives data and not just sends them. Verification of the data is important here as well as making sure the received data is confirmed to make sure the necessary changes are made. In figure 2.15 the right petri network is that of the new position command. In these when the SSPIF is triggered values are read from the register and not written, but the rest is similar.

If at any point a value is received that is a new message routine this one is aborted and a new routine is started, if at any point values are received through the SSPBUF that are not what was expected the whole process is reset.

The length and exact function differs for these routines, an assembly of how they work can be seen in table 2.4.

Table 2-4 SPI Message Routines

Routine Name	SSPBUF starter	Length	Description
Sensors	201	4	Sends the values of the proxy, speed and line sensors
Direction	205	3	Sets the direction of the motors, can be forward backwards, left and right
Motor pwm	210	3	Sets the pwm value for the motor and starts the pwm subroutine
Motors Full	215	1	Sets the motors to full on without pwm
Motors Halt	217	1	Stops the motors
Function	220	3	Set the function routine of the microcontroller
Set Position	225	5	The microcontroller receives the next position required
Request Position	230	4	Requires the current position of the microcontroller
Request Status	235	2	Requests the status of the controller, which includes errors and if it reached the final position
Set Safety	237	3	Turns on or off the proxy based safety procedures
Start	240	2	Starts the routine
Stop	242	2	Stops the routine
Reset	244	2	Resets the whole board

The SSPBUF starter represents the value that the Serial Synchronous Buffer needs to have in order to start a message routine; the message routine will have the same value itself and will continue for the length mentioned in the next column.

Certain routines that send information to the raspberry pi don't send a confirmation because the data itself serves as proof for that. For other functions at the end of receiving a set

of data the value 255 is written to the buffer as a confirmation of receiving data and as a confirmation that the microcontroller is indeed turned on.

The response at the end of every command is important, this command lets the raspberry pi test whether the microcontroller can receive the information or is even turned on. If the microcontroller is turned off the raspberry pi doesn't receive any error reports from the SPI driver, the only hint is that no matter what happens the value read from the buffer the value is 0.

2.3.5 Programmed Execution Routines

Currently there are 4 subroutines programmed into the robot, these represent basic functionalities of the robots of this type. The most basic functionality is the first which basically reacts to the controls received from the raspberry pi. It is possible to change the direction of the movement back and forth and left and right rotation and change the speed of it as well. The problem is because of the construction of the wheels and the slip the robot won't rotate if I just change the speed of each motor. In order to be able to rotate we need to stop the robot and only focus on the rotation, either left or right. If we use pwm to set the speed of the rotation the user will not be able to control the robot, to assist this, a delay between rotations is the one that is being changed and not the pwm speed.

In order to speed up the working of the subroutine, it is put in a while loop where the condition is the function value remaining the same, 1. The first thing the function does is iterate the safety loop if it's activated. The next step is a switch that sets the correct motor direction values based a command direction variable. The final step is setting the speed based on the direction of the motors. There are 3 cases, where the final case has 2 extra cases: the first is when the motor needs to stop; the second is full throttle; the third case is the pwm control of the motors where the simple case just gives a pwm control value but in the case of rotation besides these pwm values the motors need an off period to allow the user to react, the off period is 2,000,000 cycles which is about 0.33 seconds in real life.

The final procedure is done after the loop exits; in this case the motors are reset in order to make sure they do not interfere with other processes.

The second execution routine is for the positioning of the robot. In the positioning we consider that the robot should always go head first towards a point and that due to the way the robot turns and the directions are counted certain errors might occur due to slip and the positioning of the magnets relative to the start positions, so bigger values give better results. An example of how the directions and everything is done is presented in figure 2.16.

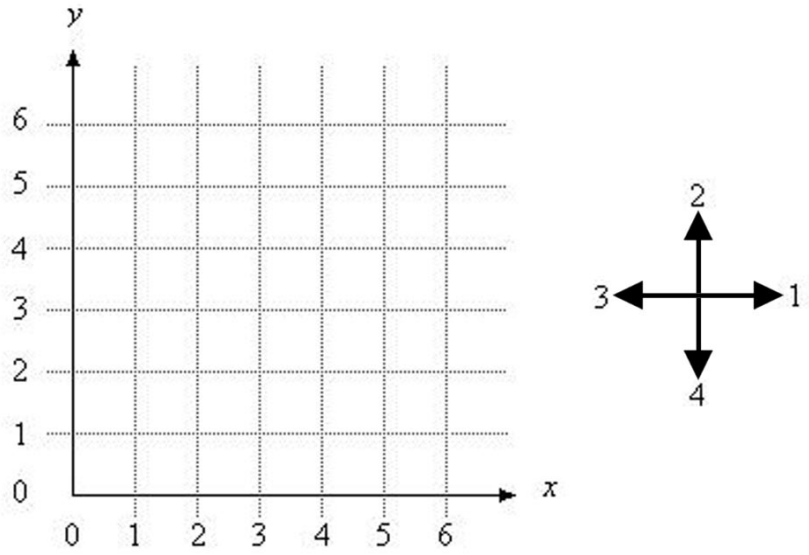


Figure 2.16 The positioning and directions of the Robot

The algorithm for the program is very simple, at each iteration of the program it checks whether the required position in the specified direction is the one that is required and if it's not the it goes to the position, and finally it turn to the direction required. The microcontroller receives 4 parameters from the raspberry pi, position x, position y required direction and whether to start or stop.

The function is written inside a while loop where the condition the stopping condition is whether the function is started, whether it's the required function and whether the required positions and directions are met.

The first part of the function is the same as before the safety that depends on the sensors and whether the safety is enabled or not. The difference in this part is that because the robot only goes forward only the case where the front proxies are triggered are interesting.

The algorithms for the two directions are mostly the same, having differences in certain values but work with the same idea. The first step is to check whether the algorithm needs to enter at all by checking if the position of the robot on that direction is the required one or not, if the required position is not met then the first thing to check afterwards is whether the orientation of the robot is proper in order to move towards the goal position. If the orientation is not right, then it needs to turn in that direction and afterwards it moves to that direction while the current position meets the required position.

After the required positions are met the device checks whether the required orientation is met, if not it turn the robot to the direction specified.

There are 4 directions mentioned in 2.16 each going forward and back on one of the axis x and y, where 1 and 3 are forwards and backwards on x and 2 and 4 are the same on y. The positioning on directions is connected, one positioning doesn't start while the other is not done.

The third functionality is that of the maze solver. It follows the basic algorithm of maze solvers. It goes forward while one of the front sensors are triggered considering that it reached a wall. Afterwards it turns either left, right or back depending on whether the way is clear or not. In all of this time, the direction and positioning of the car is monitors in order to see its progress through the maze. This functionality is not put into the example due to the unreliable results from the sensors.

The final functionality is that of a line follower. The program is set to run while there is a line under the sensors, or while there is a line detected under the sensors. The robot needs to go left while only the right sensors are triggered and to go right while the left sensors are triggered, in the case where only the middle sensors are triggered it should go forward and while only the extreme sensors are triggered it should turn more. This functionality is not used in the example die to the unreliable sensors.

Chapter 3. Design of the Structure and Electric Supply

3.1 The structure

The structure of the robot requires it to perfectly fithe a coupld of objects, the microcontroller, the raspberry pi, the motors, the voltageregulator required to supply both and to make sure all sensors have are well placed.

The size of each object that needs to be fit inside the structure can be seen in table 3.1.

Table 3-1 Structure components size and requirements

Component	Length(mm)	Width(mm)	Height(mm)	Description
Board	120	80	4	The Microcontroller that needs to have its sensors close to the edge
Raspberry Pi	86	56	21	The Raspberry Pi needs to connect to the Microcontroller through a butterfly cable and to the raspberry pi and have space for the Wi-Fi module
Raspberry Pi Camera	25	20	9	Mounted on the Pi Camera Holder
Pi Camera Holder	45	35	30	Hold the Camera should not be obstructed in any way
DC Motors	52	24.8	24.8	Mounted on the sided hold the wheels
Raspberry Pi Power Supply	45	35	15	Power supply to controller and Raspberry Pi, needs easy access to battery as well.
Batteries	85;55	65;40	23;15	Connectors can't be modified, mounted close to ground
Magnets	4	3	3	Magnets put in wheels for Hall

Each of the components presented have their own requirements, for orientation mounting and other features that are not presented in the description. Furthermore, one of the key elements of the design was to make it as cheap as possible and furthermore, make it easy to put together. In order to do this, two possible solutions were thought out, the first would be to use a 3d Printer, which would allow any design and easy testing. The problem with 3d printers is that they are expensive to buy and having someone print something out is a lot more expensive. Due partly to this and taking into account the possibilities in the local environment the best solution is cutting the parts out of Plexiglas.

The Plexiglas is quite a brittle product so we need to have a pretty thick version of it calculating it into the total mass of the robot, the best compromise is a plexiglas that is 4mm thick.

Further requirements for the system are based on mostly the sensors and certain electrical connections that need to be made. The proximity sensors require that they should be one of the furthest things in front and behind the system, with the constraint that nothing can block their view, rendering them useless. In the case of the Line following sensors require that they have a clear view of the floor undisturbed by any mechanical devices, furthermore the distance of these sensors from the ground is of main importance.

One of the most restrictive aspects of the whole system is the placement of the magnets on the front wheels and the hall sensors. These sensors are on the bottom of the

microcontroller and should be very close to the passing magnets, this in itself localizes the wheels of the robot. The resulting design can be seen in figure 3.1.

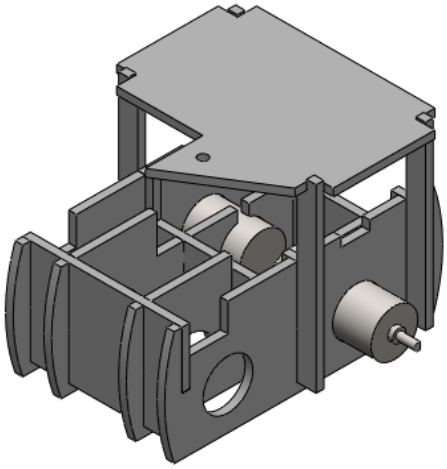


Figure 3.1 Design of Structure

Never should the wheels be more than 0.3 mm away from the sensors because they are going to vibrate. With other errors on the wheels and it would render the positioning useless to have the certain passes skipped. The other important parameter is the heavy batteries that give power to the whole system, these need to be low to the ground, otherwise these can make you have a robot that can turn over on a sudden start or forward backward movement. Going up we consider the placement of Batteries adjacent to the raspberry pi power regulator, and never should it be considered to put the batteries bellow the proximity sensors. Furthermore, going to the next step the microcontroller and the raspberry pi should be placed in such a way to allow the butterfly cable to be easily mounted without twisting the cable. Furthermore we let the heatsink from both the pcb and the power regulator for the raspberry pi enough space, you need to do this to protect the rest of the components from the heat and allow the cooling down of the heatsink.

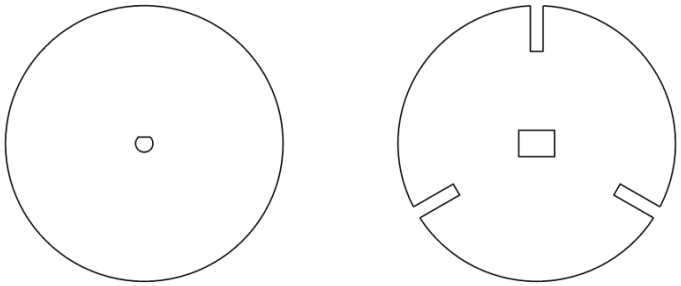


Figure 3.2 Wheel Design

The design of the wheels is presented in figure 3.2, where the left wheel is the back wheel, having the inside made in order to be put on the motors. The second wheel is the one that is designed to hold the magnets; the cuts from the exterior are designed so the location of the magnets can be changed. The center is designed in such a way that two 4mm strips of Plexiglas can be glued together and used as a drive shaft. The driveshaft is then hold on by two ball bearings, two on each side. Initially there was only two in total because a long enough shaft could connect them, but the two wheels need to be able to move independently and the middle needs to be clear in order for the line following sensors to work properly.

3.2 Electric Supply Design

The Supply for the microcontroller can come directly from the batteries having protection and other things connected to it, and even needing the 7.3 volt input to power the mtoros correctly. The Raspberry pi on the other hand needs a stable 5V input or it might burd out, and we cannot power it from the GPIO pins either, so the only solution is building a voltage stabilizer for the raspberry pi.

A possible solution for a voltage stabilizer can be seen in the schematics presented in figure 3.3.

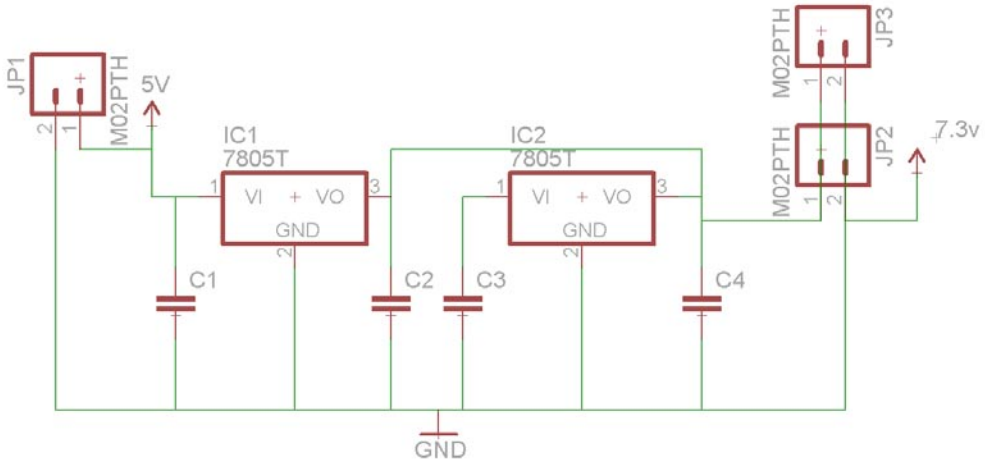


Figure 3.3 5V Regulator for Raspberry Pi

The design is done using two 7805 voltage regulators mentioned as IC1 and IC2. These are connected in parallel two give the same voltage output but to be able to give the double amount of current with the same Heat dissipated for each device. To make sure these don't overheat after all a Heatsink is connected to these as well.

On the right side we have JP2 and JP3 which hold the place for the connections of the batteries and the input for the microcontroller. On the left side we have the connections for the raspberry pi which is a micro usb plug where we only have connected the ground and the 5v Vcc. Each voltage regulator requires two capacitances for stability. The capacitances C4 and C2 have to have the value 0.1uF or 100nF in and the Capacitances C1 and C3 need to have the values 0.33uF or 330nF in order to stabilize the system.

In the datasheet it's mentioned that one voltage regulator can work with a current throuput of 1 amps this is only true if it has a heatsink, and taking into account that the raspberry pi consumes around 400mA and the Dongle another 248ma while the raspberry camera another 320mA we need to make sure that they can support the load.

The Batteries needs to supply around 1 amps of voltage to the raspbery pi and could possibly need to supply even 1.4 amps to the motors alone and another 200mA to the microcontroller, so at its peek it should be comfortable to supply at least 3Amps in order to not have the microcontroller or the raspberry pi restart. The property of the battery that shows how fast it can be discharged is the C rating which shows how fast can the battery dump its full load of 1.8amps, for example a battery of 1.8amps with a C rating if 1 can supply 1.8amps maximum. The battery for this project is a 20C battery being well and enough for the requirements.

Chapter 4. Programming the Raspberry Pi

The Raspberry pi is roughly a card sized board hath acts like a single board computer. It is developed in the United Kingdom by the foundation that wears the same name with the initial intention of using it to make computers available for impoverished countries making a personal computer a cheap alternative.

The original raspberry pi had a Broadcom BCM2835 system on chip or SOC which included and Arm Processor a Video Core IV GPU and was shipped with 256 or 512 megabits of ram for the + version. The version used here is a Raspberry Pi + model that has the 512 mb of ram and SDHC slot with a possible of 16 Gb memory. Furthermore the Raspberry pi has Three possible video outs: The RCA video that is used for most of the old monitors screens; and HDMI output which is mostly used for TV screens and certain monitors and finally a CSI parallel port that can be used to output the video as well. The Raspbery pi also has status leds two USB outputs, a set of GPIO pins with SPi and I2C options and also an Ethernet port that can be used for internet connections. Finally the raspberry pi has a CSI Input for a video feed that allows fast video capture.

A better view of the system can be seen in figure 4.1

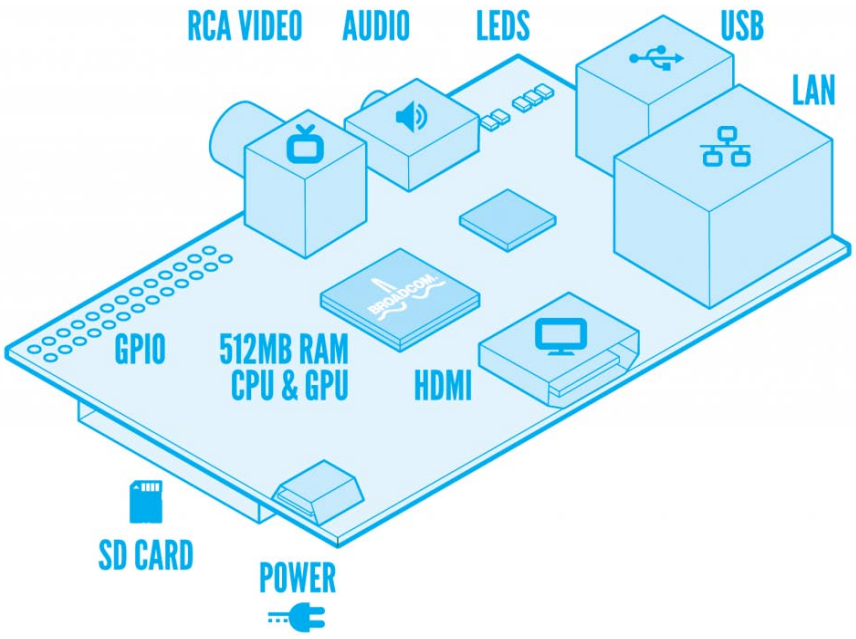


Figure 4.1 Raspberry Pi +

The current configuration of the raspberry pi doesn't require us to use most of its video output features having the video streamed, so the HDMI and the RCI as well as the CSI outputs could be disabled to increase speed and reduce the load on the processor. Furthermore the LAN connections are not used, rather a Wi-Fi dongle is used to allow extra mobility to the mobile robot.

For the GPIO pins they are linked to the microcontroller and a butterfly cable is attached to them. The sd card used needs to be a minimum of 4GB having OpenCV and other software installed on it to allow for possible Image processing to be done on the robot.

The power input of the board is through the microusb from the power supply mentioned in chapter 3 and is regulated at a stable 5v. The board and the configurations are compatible with better versions of the Board like the B+ that has 4 usb-s and 7 cores that would allow for a lot more advanced operations on it.

4.1 The Raspberry pi Camera and the video feed

The purpose of the video feed is to relay a picture of what the robot sees to the user with as short of a time delay as possible. In order to achieve this we need to take two things into account, the first is the speed of the port and the ease of the connections use and the second is the software used to forward the video feed and finally the format. The format needs to be a format that would allow the user to view it without the need of any special software of program package.

Initial attempts at this were realized using the VLC packages video streaming options and even raspivid which can directly stream video but only in a couple of formats. The VLC package had two major defects: the first was the delay which was cumulative and could reach 4-5 seconds and the second is the format which was not changeable and could only be viewed from another vlc plug-in or player. The Raspivid is a better option but the format is not changeable, better then the vlc, but still almost fixed and the delay was quite high with this as well 1-2 seconds, the benefits of raspivid were that you could start it from the command line that you couldn't with the VLC and that you could alter the camera settings through the Raspicam driver.

The final solution was done using Motion software and setting done using the raspicam, and bcm2835-v4l2 the connection scheme can be seen in detail in figure 4.2.

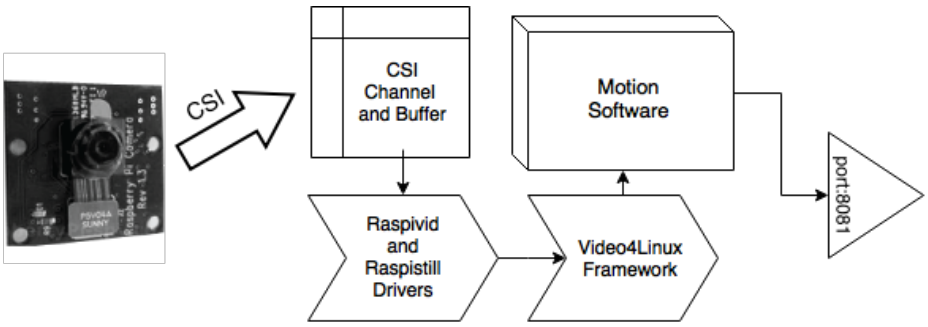


Figure 4.2 Raspberry Camera Workings

The Raspberry pi Camera is not directly compatible with the Motion software so the intermediary bmv2835-v4l2 which is a video driver for Linux needs to be used to make the camera available as an input video device. The command used to make it available is *"sudo modprobe bcm2835-v4l2"* where the sudo is the main user in order to allow the changes to be made, modprobe is the command to allow the device to be set as input and the final part is the driver used to achieve this.

The Motion software is installed simply from the bash but then needs to be configured, the exact configuration depends on how high a quality feed we want and how much of the processing power of the pi do we actually want to use up. The exact setting will be available in the annexes but the important ones are: the port needs to be set to 8081; we need to chose the video device which is /dev/video0; we need to set the max frame rate to 20 and finally turn off any local saving of the images or local previews.

The motion normally can be enabled using its demon, but because the modprobe command needs to be run beforehand it's not possible in this case. In order to do this a pre-installed software called crontab which is a demon on the raspberry pi that can perform actions at certain sets of intervals. This allows me to set that it should run the configuration command 10 seconds after booting and the motion command 30 seconds after booting allowing everything to work.

4.2 REST Service and SPI

The Purpose of the REST service is to transform the requests from the client into messages and requests sent through the SPI to the microcontroller, to handle errors like wrong type of data is sent or if the microcontroller is offline.

The SPI connection to the microcontroller is done through the GPIO pins, to be more precise, pins 19,21 and 23 for MOSI, MISO and CLK. These are the Connections for the spi device 0 as in the main device. The raspberry pi has another SPI device connected to it which only works on two wires.

In order to make the spi communication on the SPI we need to take a few steps. First we need to enable the SPI pins through the modprobe's blacklist file where we need to remove the blacklisting for spi-bcm2708. After this we need to restart the device and update it to make sure everything is working fine.

In order to work with the SPI registers we need to install one of three drivers. The most basic driver that accesses the port is the spi-bcm2708 that grants the user access directly to the registers and everything. Built on top of this driver we have the spidev and the wiringpi. The wiringpi is a good driver mostly used for digital outputs and things like that. The documentation for the wiringpi is sketchy at best so the best option is the spidev driver which has good documentation and examples.

The programming of the SPI communication will be done in python, where we need to initialize the driver and then we need to set 3 things: the mode needs to be set to mode 3 to match that of the microcontroller; the communication needs to be opened with the configuration (0,0) which stands for the first Spi Port set's first port; the max speed of the spi which is 1Mhz in our case;

There are more version of communicating, of sending and receiving data on the same clock and other functions but for this application we only is the writebytes and the readbytes function the first of which accepts an array of bytes that it sends on the spi. The second receives an integer which is the number of bytes it should read and return an array of bytes. The detailed use of this is presented in the class Comm which is presented in the annexed code.

The Rest service is realized using a Flask library that allows the user to process request sent to a certain port. In order to do this the app needs to be run mentioning that the *"host='0.0.0.0'"* this tells the object to host the service in such a way that it's available from the outside on the devices ip having the port 5000. The service allows the user to use Http protocols like Post and Request to send or receive data.

The use of the Flask and the Comm class can be seen in the class diagram presented in figure 4.2.

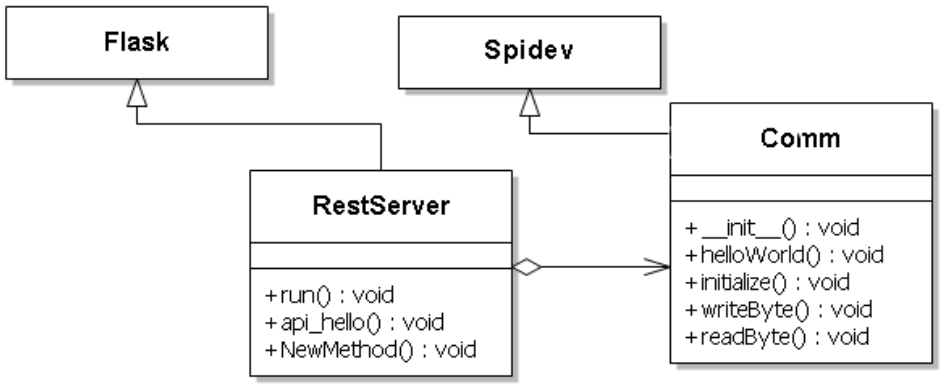


Figure 4.3 Class diagram for Python Rest Service

The function of the Rest server is to translate the Get and Posts request and to send these through the spi. The workings of the Spi and the programming of each of the functions on the microcontroller is explained, details will be in the code in the attachments. Details about what each function does and how which function communicates with which and what are the directories can be seen in the table 4.1

Table 4-1 Rest and Spi function use

SPI function	REST function	http location	Type	Description
--	api_hello():	/	-	Tests the connection to the Controller
setFunction()	api_func()	/basics/function	POST	Sets the function/subroutine of the microcontroller
setSafety()	api_safety()	/basics/safety	POST	Turns the safety on or off
resetBoard()	api_reset()	/basics/reset	POST	Resets the board
setStart()	api_start()	/basics/start	POST	Starts the subroutine
setStop()	api_stop()	/basics/stop	POST	Stops the subroutine
sensorRead()	api_sensors()	/basics/sensors	GET	Requests the proxy, line and speed values
isReady()	api_isReady()	/basics/ready	GET	Requests the status of the subroutine
setFullMotor()	api_full()	/control/full	POST	Sets the motors running on full speed
setHaltMotor()	api_halt()	/control/stop	POST	Stops the Motors
setDirection()	api_direction()	/control/dir	POST	Sets the direction of the motors
setPwmMotor()	api_pwm()	/control/pwm	POST	Sets the motors running on the PWM value
getPosition()	api_getPos()	/pos/get	GET	Requires the current position
setPosition()	api_setPos()	/pos/set	POST	Sets the next position

The structure of the function used in the main file mostly follow the same form. Based on the location of the request a certain function is selected that processes the request. The selected function splits the body of the request, if there are more than 1 inputs, or simply takes the first character if we are looking for a 1 character command or even skips the step if only the location of the request is important.

The next step of the sequence is to send the data to the Spi Communication Class where based on the data and the response of the microcontroller an answer is sent. When it comes to answers there are two types, the first responses with either 0 for a correct value and an accepted command, with -1 in case there is no connection to the board or with -2 in case the given value is not in range. Some commands like the reset only have 0 and -1 and an answer. Finally some functions require the data of sensors and other important information like location, in these cases these are returned as strings

The python application needs to be started from the terminal in order to work which is not acceptable for our case because all the services should be available on startup. There are two solutions for this problem, the first is installing a Service that can accept the Flask and use it as a server which is the proper way of doing it. Another way is using the crontab to start the python script on startup which is a simpler way of doing it. An important thing is to add it to the sudo user's crontab and run it as such. Otherwise the SPI can't work.

Chapter 5.The Client and User Interface

The Mobile robot is designed in such a way that it can be accessed by multiple types of clients and be controlled in multiple ways, the only requirement is that the client and the robot should be on the same network or the router should have port forwarding installed and have access to the robot.

The robot needs to know which network it need to connect to, and in order to do this, the user needs to edit the sd card on the robot or connect it to a monitor, keyboard and mouse and input the settings of the network/networks the user wants the raspberry pi to connect to.

Because this connection needs to be done and the user needs to program the raspberry pi to be ale to connect to it, no extra security measures are needed to protect the data from the device and requests can go directly to the microcontroller without the need of an additional layer for security.

The communication diagram of how the POST and GET requests are sent from the client and how they are interpreted by the raspberry pi is presented in figure 5.1

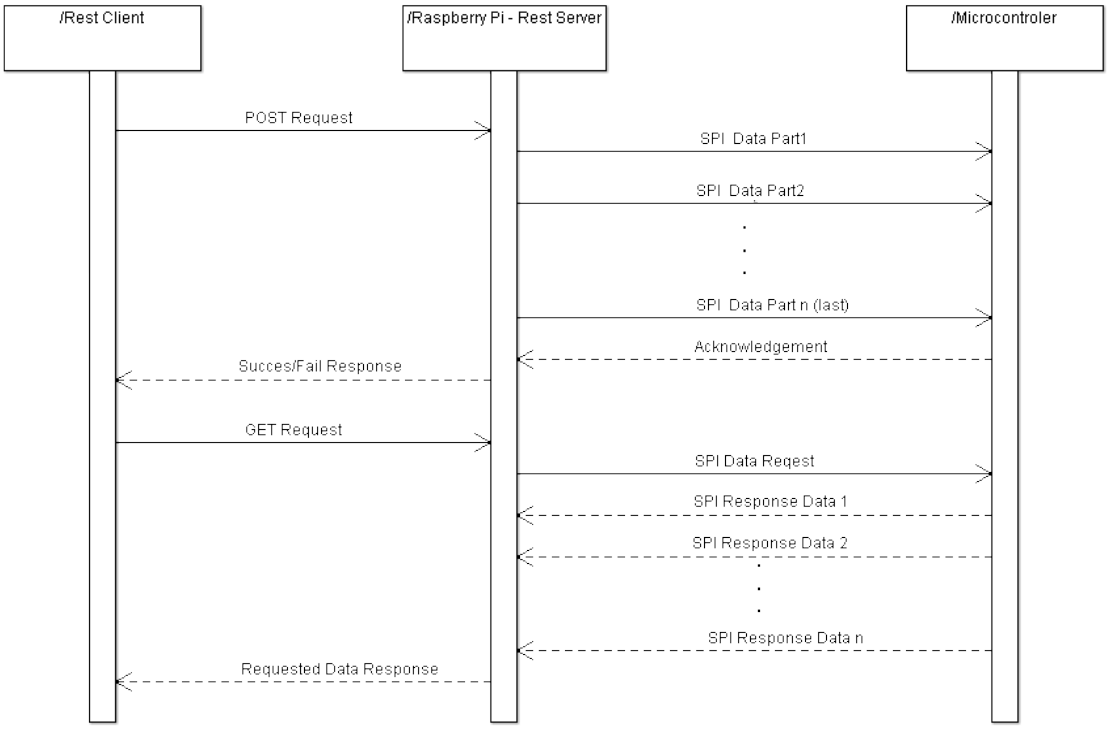


Figure 5.1 Sequence Diagram between Client, Raspberry pi and Controller

We can see from the sequence diagram how a message is transmitted from the Client, interpreted by the Raspberry pi and sent to the Microcontroller. There are twi distinct types of services, the first, the Post request is typically used to send data to the microcontroller, this is done by sending all the command inside one POST request which the raspberry pi interprets, splits up into little peaches of one bytes an sends to the microcontroller through the SPI channel. This is done though one clock cycle for each Data Part.

The GET request is used to receive status and sensor parameters from the microcontroller. These are done is a similar fashion to the POST with the difference that only one command is sent to the microcontroller telling it the type of data it needs to send and afterwards data is read from it until the required number is reached. The Raspberry pi takes all of the information from the controller ad packages it into a Response for the GET request. If

the microcontroller is turned off the response will be zero, in order to determine when the value is due to the lack of power or due to the actual values, at least one or more parameters sent can't have the value 0.

5.1 Overview of the Android Application

The Android application is designed to be a presentation of the possibilities of the Mobil robot, how it's supposed to be controlled and what data you can gather from it. The application is built in a way that the user can connect to the board and send control data and gather sensor data.

The way the user interact with the application is build on the idea that the application should work much like a RC car remote control should work, or a steering wheel, with extra buttons and info, furthermore the second option allows the user to control the car on the x and y axis and in direction.

A detailed view of what the user can modify and how these modifications work can be seen in the use-case diagram seen in figure 5.2

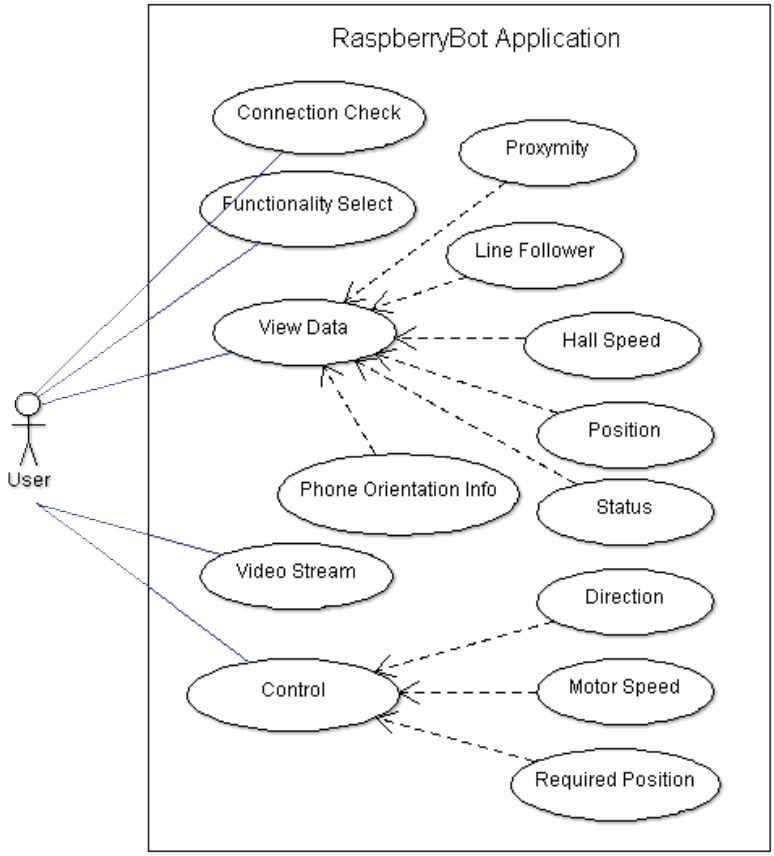


Figure 5.2 Use Case diagram for the Android Application

The use-case diagram represents all the actions a user can take as well all the data the user has access to while using the application. The figure represents all the actions that can be taken by all the Activities of the Android application. The Connection check allows the user to input an IP address and the application will check if it's a viable address for the robot, any other functionality is blocked before this is checked. The Functionality Select allows the user to select which type of control he wants to use, whether it's the manual control of the car or it's the positioning.

The View Data refers to all the displays that are available to the user regarding the data in play. This part is an extension of data windows like the proximity sensor led displays as well as the Hall sensor displays, the position status and phone orientation data.

The Video Stream is the video stream from the raspberry pi Displayed in a View on the two control activities. The size on the device does not change and there is no possibility to rewind or record the video stream.

The Control represents all the control actions a user can take to move the robot. The first activity allows the user to control the direction of the robot, forward, back, left, right and the motor speed which can be 0, full or controlled by a pwm signal.

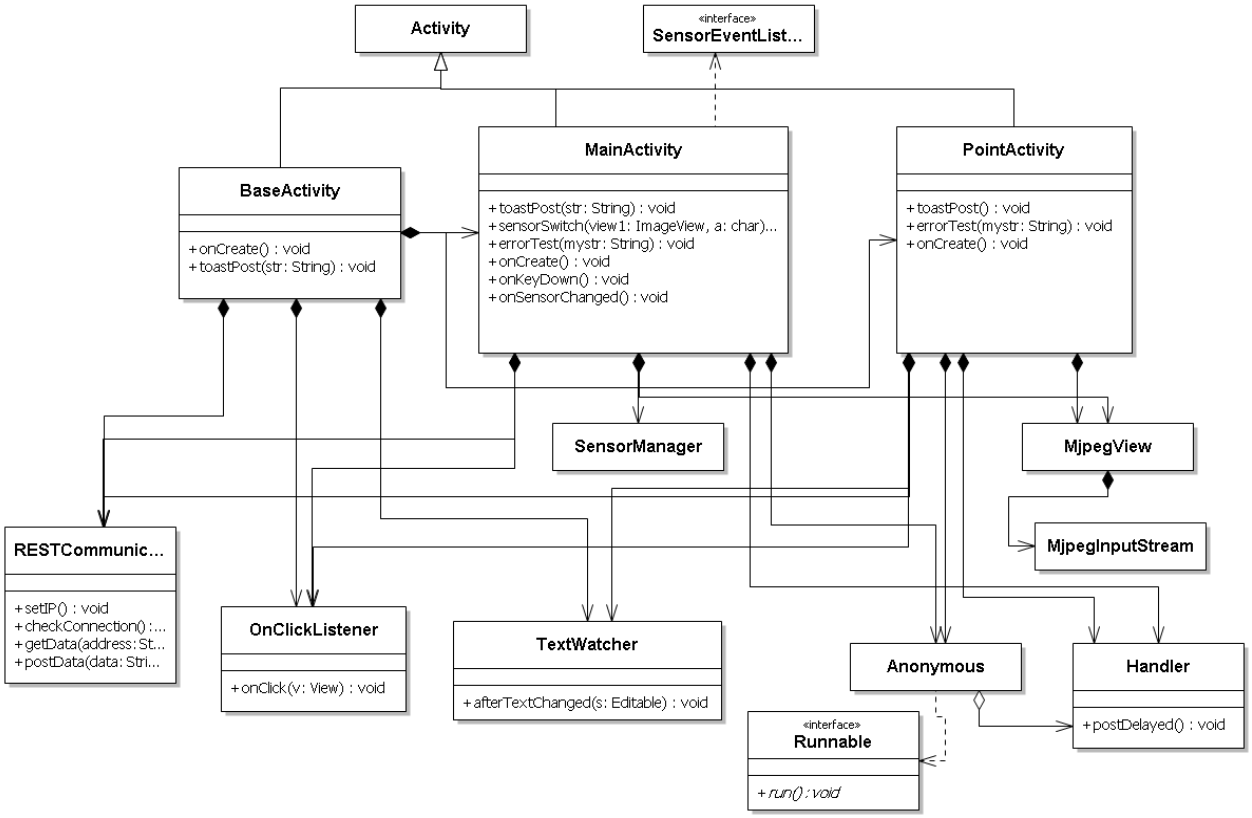


Figure 5.3 Class Diagram for the Android Application

The class diagram displays most of the connections from the Application, leaving out details that would make for a too big diagram that would be hard to understand. The main parts of the application are the Activity Classes, where BaseActivity is the initial class, which means that when we open the application the first view that we see is the BaseActivity, and consequently as we can see on the diagram the rest of the Activities are opened or created from that first activity.

The Structure in an android Application is based on the idea that the design of the Activities and the Objects inside them are saved into xml files that can be read at the point of execution and this control design and functionality parameters like linking resources to id-s and defining Titles names basic texts and other.

The Activities is the Application are the essential workings, where the Buttons and Text fields are updated using Interrupts and other functionalities like the OnClickListeners and TextWatchers, or even Asynchronous Event Handlers can be implemented with time delays or events in the case a back button is pressed or disabling, enabling button in order to be able to fully control the application. Most of the features are initialized on the create command of the Activity.

5.2 The Base Activity and REST Communication Classes

The base activity is the first Activity of the Application, the one that is opened with the application, in this case this is the simplest of the three activities only needing to test the connection with the mobile robot, give an initial user interface to the user and select which type of control the user want to use after making sure the connection exists.

In order to test the connection the user needs to connect to the specified ip address and see if there is a response. This is done using an instance of the RESTConnection Class which is tasked with uses the Http Client class and all the adjacent classes in order to connect and wait for a response.

The RESTConnection class is created using a basic IP address that is most specific for the device. When the ip is changed on the activity the setIP() method is called that changes the base ip of the class to that one.

There are 3 methods that are used in this class, the method used for checking the connection is the checkConnection() method that returns a String variable. This method creates a new HttpClient and a new HttpGet object, for which the input location is a concatenation of strings, the first being *"http://"* the second being the ip address given by the activity and the third being the port and the location, which in this case is *":5000/"*.This is the location of the Hello World part of the REST service on the mobile robot. Due to the fact that errors and problems may occur due to the connections nature the rest of the code needs to be surrounded by a ClientProtocolException and an IOException for interrupted connection. The code in the try section executes the httpGet request that we made using the client that we created and places it in an instance of the HttpResponse class. The part we are interested in at this point is the status code of the connection, which should be 200. This status is received by requesting the statusLine and the StatusCode from the response we just received. If there are any errors instead of the code an "*error"* string is returned.

For the other instances the address of the service is not as simple as for the *"Hello World"*, and to make searching for the addresses they are made into Static strings defined by the Class, these can be seen in table 5.1.

Table 5-1 Static Strings of the addresses

String Name	String Value
URL_SENSOR_GET	"basics/sensors"
URL_FUNCTION_POST	"basics/function"
URL_SAFETY_POST	"basics/safety"
URL_RESET_POST	"basics/reset"
URL_START_POST	"basics/start"
URL_STOP_POST	"basics/stop"
URL_READY_GET	"basics/ready"
URL_FULL_POST	"control/full"
URL_HALT_POST	"control/full"
URL_DIRECTION_POST	"control/stop"
URL_PWM_POST	"control/dir"
URL_GETPOS_GET	"pos/get"
URL_SETPOS_POST	"pos/set"

The name of the string also gives us hints about the way the address should be used whether it's a post or a Get request and in the comments we have a description of the format the body needs to have for Posts.

The second important method of the class is the getData(String address) method which is used to request data from a certain address and return it to the application. This is done in a

similar way to the test connection method with the exception that the initialization of the HttpGet method requests the extra address which is one of the static strings after the *":5000/"*.

The part that is inserted into the exception bracketed region is similar up to the part where it receives the status from the response. After this line the status is checked, and if it's the OK status which is 200 the rest of the code is continued otherwise an *"error"* string is returned. If the result is 200 then the entity is retrieved from the response which is then converted into a String using the EntityUtils which is the returned to the user.

The third and final important method of the class is the postData function which returns a string and receives and an input the address and a string containing the data the user wants to send to the server. This method differs from the get and the test methods because an Entity needs to be constructed that is sent to the service. The first step is verifying if the given Data String can be encoded to be a String Entity, afterwards the content type of the entity is set using a basic Header and *"text/plain"* type. After this we set the entity as the HttpPosts own entity after which we enter the same exception brackets we did in the previous method. Inside the exception handling try we execute the same http Response as before only using the before generated httppost after which we check for the status to be okay otherwise we send back the error message so the user knows that the sending was not successful.

In order for the requests to work we need to set one extra thing, the android-manifest file needs to be changed to add permissions for the device to connect to the internet, this is the file that contains and sets the launch icon and the start activity.

The Main Activty and all other activities perform an errorTest on all the http responses, this is done to ensure the case where the robot is disconnected is handled by the application. The error test basically checks whether the response is *"error"* and if it is, the it shows a toast of the error and cancels all handlers that cycle through instructions, as well as stops the mjpegview instance. I the string is not error then it returns the actual string.

The onCreate() method of the class contains all the instantiations of the buttons and all the actual work. In order for the rest work to be allowed we need to change the Strict mode of the device so it would allow the main Thread to work with http actions. After all the initial values an setting are done, like disabeling the buttons for the two activities we have an onClickListener that is attached to the Submit button that starts the CheckConnection method after inserting an ip to check whether the robot is available, if it's available the text underneath is is colored green and the value *"Connected"* is shown, otherwise the text is colored red and the value *"Connection Failed!"* is presented. This also enables the buttons that can start the new activities.

In order to check for a new url we need a textChangeListener on the text field with the ip, this is triggered whenever the text changes. After it's triggered we need to check if the changed value is an acceptable ip address, this is done using a matcher and a pattern, the pattern is *"^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])$"*. After a valid ip address is given a toast message confirming this is shown and the value for the ip is sent to the rest instance.

The final two onClickListeners are designed to open the two activities if the connection is made. These create a new intent using the current activity and the one we plan to open after which we insert the extra data, which in this case is the correct ip, and afterwards we start the activity given the intent.

5.3 The MainActivity or the Manual Control

The MainActivity is the most complex part of the Android application and is designed to allow the user to control the mobile robot like one would a little remote controlled car.

This activity is designed to get the pitch yaw and roll data from the mobile device and allows the user to control the mobile robot using these and a few other buttons and information.

The activity implements the SensorEventListener this means that the method onSensorChanged needs to be implemented on the Activity. This method is triggered when the value of the sensor changes. The sensor gives us a number of values like the gyroscope values uncalybrated, calybrated , the acceleration and the type rotation vectors as well. The values we are using are the rotation on axis x, y and z while the coordinate system and how it works is displayed in figure 5.4.

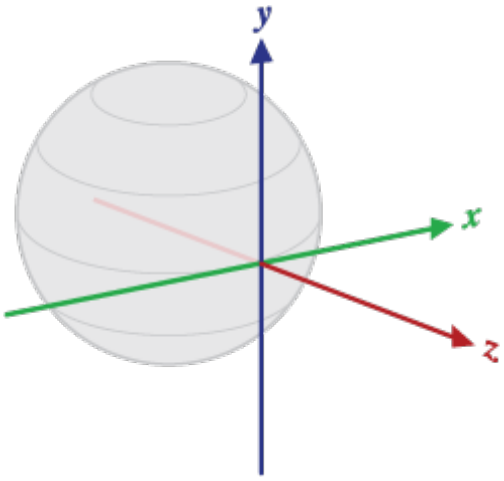


Figure 5.4 Coordinate System used by the rotation vector sensor

The application at hand requires the rotation values of the mobile device so we need to use the *"TYPE_ROTATION_VECTOR"* as an input for the sensors. For this settings the first three values of the occurred event are the rotations on each axis, where for x, y and z the first, second and third values of the event correspond. There are other more complex values given but we are not interested in those.

From these three values we are only interested in axis Y and axis Z for which if he hold the phone horizontally the values of rotation on the Y axis corresponds to the Pitch while the value of rotation on the Z axis corresponds to the Roll of the device. These values are displayed on the screen as user information. These values are very circumstantial because depending on whether we tilt the phone forward or backwards we get different values. The table 5.2 explains how the values change and what values are used for the upcoming calculations.

Table 5-2 Pitch and Roll values and Normalization

Pitch Value	Normalized Pitch	Roll Value	Normalized Roll	Control Case
>50	Min(30,180-x)	-	-	Rotate direction 3, pwm=(5.3)
<-50	Min(30,180+x)	-	-	Rotate direction 4, pwm=(5.3)
(-50,-5)	Min(30,x)	-	-	Rotate direction 4, pwm=(5.3)
(5,50)	Min(30,x)	-	-	Rotate direction 3, pwm=(5.3)
(-5,5), (-175,175)	-	-	Min(50,10)	Rotate direction=1,pwm=(5.2)

After the values are normalized and the values are put into a certain case, we need to set besides the direction the value of the pwm, this is done using the equations 5.2 and 5.3. These equations are calculated using the simple calculation of line using 2 points, we

consider the maximum and minimum values given to the pwm and the values received from the sensors and then we calculate the slope using two sets of equations with the form in 5.1.

$$a \cdot x + b = y \tag{5.1}$$

This equation is taken into account having for the pitch pwm calculation the minimum and maximum values for the pwm [0,65] and for the sensor data [10,50] using these the values for a and b are -1.225 and 76.25 and give the equation 5.2.

$$-1.225 \cdot x + 76.25 = y \tag{5.2}$$

This equation is taken into account having for the roll pwm calculation the minimum and maximum values for the pwm [0,65] and for the sensor data [0,30] using these the values for a and b are 1.96 and 5.2 and give the equation 5.3.

$$1.96 \cdot x + 5.2 = y \tag{5.3}$$

The values obtained from these equations are then displayed for the user to see.

Besides the control through the rotation of the phone the user has 4 buttons that he can use to control the robot to go at full speed, to stop, to reverse and to enable or disable the safety values. All of these buttons are toggle buttons and are used with the onClikListeners where a postData is sent to the robot put through the errorTest from before and the values are confirmed and logged and displayed.

In the case where the user presses the Back key on the phone the application needs to go back to the main activity and cancel all timerHandlers and stop the mjpegView playback, this is done using the onKeyDown function.\

The sensorSwitch method allows the user to change the led shaped images that signify the proximity and line following sensors values this function accepts two inputs, the ImageView that you want to change and a character which is either 1 or 0 depending on the value of the sensor, it then changes the drawable reference according to the character. A character is used because the split string values received through the http request can easily be split to characters.

In order to view an MJpeg video stream we require the Mjpeg stream library and a class that extends an Android View that we can place in the Activity. In order to setup the MjpegView we need to instantiate it and assign the corresponding view from the xml files after which we need to set the MjpegInputStream as a source having the url made out of an initial string *"http://"* after which having the ip address of the board and finally the port of the stream which is *":8081"*. After this is done we set the display mode to full screen and we set the showing of the frames per second or fps to false.

In order to control the mobile robot we need to send http request to the device about the control variables and request data about the device. In order to do this we need to send periodic requests to the device. These requests must be done on a separate thread from the main one to assure that it doesn't block the main thread. This is done using a Handler and a Runnable implementation. When the Runnable is called it executes the new routine in the background after which is called the specific timer Handler with a delay that restarts the process. For this Activity two such Threads are required, one for the sensor data and one to send the control data.

The Runnable process for sending the control parameters to the microcontroller is optimized in a way such that it doesn't send the same control data twice, burdening the connection. The first test is whether it's inside a normal routine, if any of the extra buttons are activated it should not send new data. The second test is regarding the required pwm, if the required pwm is the same with the last sent data or very close to it, the data should not be modified. The same is true for the direction, we don't send the same direction instructions twice. Another control variable is regarding the Pwm value, if this value is the minimal value

or close to it we consider that the device should actually stop. If the value is larger than the minimum then the value is sent as a pwm command.

Sensor data request Runnable is a process that is run periodically with no possibilities of improving the request data. The data is requested in the previously mentioned ways. After the Data is requested a string is received which looks like this: *"[proxy],[line],[speed]"*. The speed of the microcontroller is given in integer so there is no modification needed there, but the values of the proxy and line sensors are given in integers and the actual state of the sensors is in binary so these values need to converted into binary and then we use the sensor switch and the corresponding character to change the display leds to the correct values. The speed of the robot is printed out to the display and in the case when it's zero the progress bar is sent to 1% so we can see a minimum value. In the case the value of the speed is not 0 we need to convert the speed, which is actually the delay value into cm/s, this is done using with the same method as presented in equation (5.1) having the max delay 21 and the min delay 11 with the corresponding speeds calculated by converting the delays into seconds and the distance between sensors we get a minimum speed of 45cm/s and a maximum speed of 83cm/s. The resulting equation for the speed can be seen in equation 5.4.

$$-3.7 \cdot x + 123.7 = y \tag{5.4}$$

The values calculated by 5.4 are displayed as the speed and knowing the maximum and the minimum speed we can use the equation 5.5 and return a value in percentage that we can put on the progress bar so we have a visual for the speed.

$$progress = \frac{calculatedSpeed \ \ -minimumSpeed}{maxmimumSpeed \ \ -minimumSpeed} * 100 \tag{5.5}$$

5.4 The Point-to-Point Activity

The point to point activity is quite a simple activity, needing only to display the video feed, update the current position and send new position data to the board. This activity includes the common errorTest and toastPost methods that the other activities do as well.

The application holds a toggle button that is designated for the Safety protocol. It works in a similar way as the one in the other activity and when the error protocol is triggered the position is reset and when a new position wants to be submitted an error report is generated as well as the routine is stopped and a start is required to go to a new position.

The MjpegView work exactly the same as it does in the previous application and so do the Start and Stop buttons which in turn send the start and stop commands to the microcontroller.

The Activity also contains A ListView that is used to save all the positions the user submitted to the server. The use of such a view requires the creation of an ArrayList and an adapter that is created like in code segment 5.1:

```
final ArrayAdapter<String> listAdapter =
new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, items);
```
(5.1)

This adapter then can be used as the base adapter of the list view. When a new point is submitted the adapter is modified adding the pint with the current format, which needs to be a String for the textView which is the basic format of the ListView. Modifying the adapter automatically modifies the list view.

The Handler and the Runnable connection that periodically requests data from the mobile robot work in the same was as in the previous examples with the difference that they request data regarding the position of the robot. The position is given in the format

"[x],[y],[Dir]", this string needs to be split using the character *","* after which they are assigned to the corresponding textViews that display the data.

When the submit button is clicked two things have to be checked, the first thing is whether there is any data inserted, and on the case of the contrary a warning toast message is display. The second test is whether the data inserted is within the required parameters, knowing the x and y need to be between 0 and 255 and the value of the direction can only be between 1 and 4. If the data is incorrect another toast is sent. After the data is tested we send an isReady() request to see whether the microcontroller is ready to accept new set of coordinates. The response can be of 4 types explained in table 5.3

Table 5-3 Error messages and Cause

Error Value	Cause of Error	Toast Response	Next Step
"OK"	Last requested position is the current one, device is Ready	"Position Submitted"	Continue
"NOTDONE"	The device hasn't yet reached the last requested position	"Not Done"	Stop
"POS"	The Safety has been triggered, last position reset	"Position Safety Error"	Stop
"CONN"	Connection error to the board	"Connection Error"	Stop
"error"	Connection error with the Raspberry pi	-	Disconnect

If the return message is *"OK"* the new positions are sent to the microcontroller and the values are added to the List.

It is possible to select the next position by clicking on an element of the list adapter, this takes the string inside, splits it to its elements and puts the values inside in the textViews that are designated for the next location awaiting a Submit.

Chapter 6. Testing and Validation

The testing and validation period of the design consisted in testing separately all the features of the microcontroller as well as all the electrical connections and functionalities, as well as all the communication channels, rest servers, spi communication. The last period of testing being the one using the android application and testing the functionalities and working in a real life situation as well as measuring the limits of the robot, range, charge-time and how much time can it hold a charge.

6.1 Electrical Testing of the PCB and Debugging

The pcb board was ordered from a Chinese company that already did short circuit testing and continuity testing on most of the routings on the board but despite this another test was done before mounting the devices on the board. The Devices were mounted and tested in order of complexity and difficulty to remove, due to this fact the first thing soldered on the device was the microcontroller and we can see in figure 6.1 the first attempt where two pins are erroneously connected together and the pcb with the microcontroller had to be thrown away.

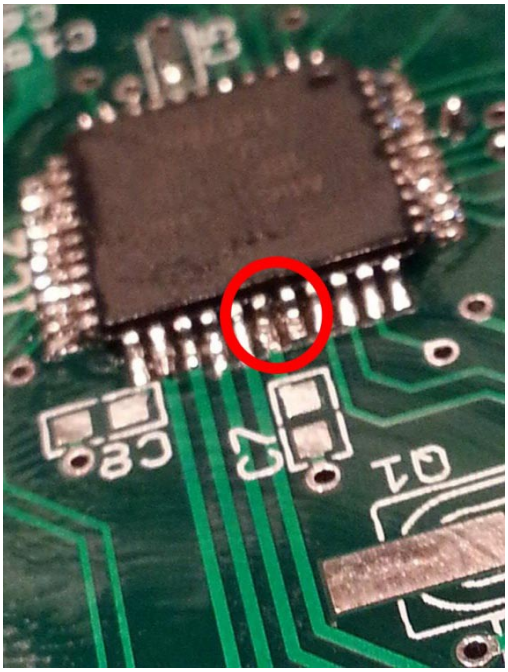


Figure 6.1 Soldering Error between two pins of the uC

All of the components of the microcontroller have been checked for connectivity problems and for unwanted connections as well.

Tests on the IR sensors had to be made in order to determine the value of their resistance, the resulting circuit and values were ultimately incorrect because there is no protection against ambient light making the sensors highly sensitive to interference from outer light. One of the sensors, the back-left has a problem because of the op-amp that broke and the value is always 0 form that sensor.

Another error that came out from connecting the pieces and programming the board was due to the slightly incorrect connection of certain pins. Only one of the Hall sensors is connected to an Interrupt pin and none of the Enables on the Motor Driver are connected to a PWM output of the board, these cause certain difficulties to the proper working of the robot

but workarounds have been made and a suitable compromised has been made so that the device can achieve all its functionalities.

The debugging of the device is cruicial to it's programming, but can sometimes slow it down, so to allow the better use of the microcontroller a DEBUG define is made which if deleted all the debugging parts are skipped.

6.2 SPI Communication and Testing

The SPI communication was tested separately through the creation of a python file that can access the SPICommunication classes functions and send values and request values from the microcontroller. The used interface can be seen in figure 6.2

```
Insert new Command
1
Sensor Values
0,0,0
[1] Read Sensor Values
[a] Safety Settings 1-On 0-Off
[s] OFF start_stop
[w] ON start_stop
[2] Set Direction
[3] Set PWM for Motor
[4] Full Motor
[5] Halt Motor
[6] Set Functionality
[7] Set Position
[8] Get Position
[9] is Ready
[r] Reset
[x] Exit
Insert new Command:
```

Figure 6.2 Python based SPI test Program Interface

The main challange of the spi communication was the configuration and afterwards it was a matter of testing cases where partial commands are sent and messages not received, to which the written algorithm responded well, having only errors when the sent commands were not executable by the microcontroller.

The tests on the SPI showed that this is a valid and usable form of communication betwene the microcontroller and the Raspberry pi.

6.3 Rest Services with spi

The REST services of the raspberry pi were tested using a REST client UI dependency from the course, which allows the user to construct Post and Get request insert a body a type and sent them to a specified location, this allowed for the testing of each individual function and using the debugger on the microcontroller to see whether the modification were made and from the command window of the raspberry pi if the message got through.

The only part that needed taking care of is the error handling and making sure everything works up to spec

For the number of request the raspberry pi can handle and still be functioning optimally, the cpu usage goes up to about 60% when more than 20 requests are made a second so this needs to be taken into account when making the rest of the application. Furthermore if the cpu usage goes up so does the power consumption and that is an important factor in this case.

The Tests on the rest services showed that it can be used to transmit control data and request sensor information as well with the idea that a better solution, or a faster one might

arise using a tcp connection, but that wouldn't be as general as this solution is, and would require more dedicated software on the client side.

6.4 Testing the system with the Android Client

The testing of the Android client system at first is tested only using the Raspberry pi and replacing the Java based rest service test with the application and seeing whether the received information on the Raspberry pi is correct or not, errors or any miscorrelations can be taken out here.

The parameters we need to take into account are the delay between the requests of the sensors values, positions and the sending of the control parameters and requested positions. From the tests the delay value for the first Activity, that controls the car like an rc vehicle the value for the sensor request delay is 500ms and the value for the controls has a delay 200ms, a lower delay would make the android application crash after a while. For the second activity we only need a delay for the current position values, for which a value of 300ms is more than plenty.

Despite these settings the android application sometimes freezes for a few seconds, this is due to errors is the main screen and the weight of the video stream. The video stream has a lowered frame per second value of 10 to allow the phone to stream the video lighter.

The Android application is a decent solution for the control, having the bonus that it can generate a wifi network directly and safely connecting to the mobile robot.

The resulting version of the device can be seen in figure 6.3

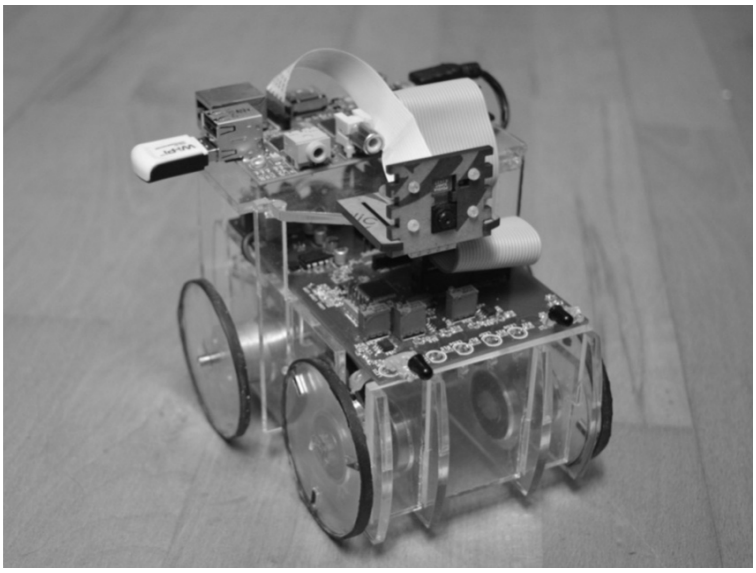


Figure 6.3 Final form of the Mobile Robot

Chapter 7.Conclusion and Future Work

7.1 Conclusion

In conclusion, in the diploma thesis there was presented the theoretical background to the whole project, that of the use of Distributed Systems and mobile Robotics in the creation of an IOT Device. Furthermore, the arduino cloud laboratory is presented as a state of the art system for the control and monitoring of the device as well as other projects where the focus was set on machine to machine communication. The newly booming field of Internet of Things is the driving force of the thesis, constructing most of the components of the thesis to fit in the requirements of the IOT devices and making it act like one. The defining factor is making the device available on the network and having control procedures implemented that would allow it to be controlled from it, as well as devising a client to control the device.

The upcoming chapters describe the work of designing the pcb and structure as well as programming the microcontroller, raspberry pi and the Android Application. The presentation starts by stating the requirements of each of the components of the system, and discussing what possible solutions there might be to solve their problems. Within this idea, the electrical design of the pcb is presented in detail giving the use and utility of each of the components and mentioning design flaws as well as trade-offs. After this chapter, the structure of the board is presented, which needs to hold the microcontroller and all the adjacent components and be able to fulfill multiple roles. The next step was programming the microcontroller that had to be able to communicate with the control board and also be able to communicate with other devices through Bluetooth, rf434 or Zigbee. The next step was to program the raspberry pi in order for it to connect to the microcontroller through the spi port and test the commands. Furthermore, the Raspberry pi needs to connect to the Raspberry Pi Camera and stream the video and also have a REST service running. These services need to be tested and made sure that they operate with the set input and give correct error reports when needed. The final step was devising a client application that can use these features to control and monitor the device in a useful manner. Following this tests have been made on the device regarding how the functions work in a real life situation, as well as tests regarding range and how it responds to going out of range.

The purpose of this diploma thesis is to design an IOT system that corresponds to most of the requirements and results a Device controllable from the internet. The resulting system allows the user to control the mobile robot from two different aspects of the device. The first aspects has a control delay of 200ms and is the manual control, where due to the construction of the device the rotation of the robot can't be done while moving forward and it has a signal range limit of around 40m without obstacles and a time range limit of 20-30 minutes depending on the use of the vehicle. The second aspect of control for the mobile robot is based on the idea of giving the robot coordinates and having it go to the requested coordinates.

7.2 Future Work

There are many aspects of the project that can be improved, as well as aspects that can be taken out in the entirety or replaced by modules, or even have functionalities added an improved.

One of the first steps that would be interesting to implement is to build a second robot and have the machine to machine communication implemented where one robot can ask the other about its status, and current functionality. Furthermore this would allow for only one device to communicate with a central hub and the others to pass data through it.

Another improvement region would be regarding the structure of the vehicles that are quite limited at this point, an Ackerman steering or and improved control should be useful for future work.

The most interesting advancement would be to take out the need of the Raspberry pi and replace it with a wifi module that would allow the microcontroller to connect directly with the central hub. This would make the use of a video stream almost impossible but would add energy efficiency to the mobile robot, maybe even improving the time range to up to 1 hour. Furthermore, this advancement would be closer to the core concept of internet of things and would allow the protocols like CoAP and even ThingieSpeak to be implemented on them.

Another improvement would be on the line of the Client which could be made bigger and more advanced as well as creating a central server that the mobile robots can register to, which would allow them to be controlled from any point of the world by making the request commands and not have commands sent to them.

References

[1] Tiberiu Leţia and Mihai Hulea, "Distributed Control Systems," 2005.

[2] James Aspnes, "Notes on Theory of Dsitributed Systems," 2014.

[3] Marcu Cosmin, "Robot Control Systems - Mobile Robots," 2015.

[4] Simon Janos, "Concepts of Internet of Things from the aspect of autonomous mobile robots," *Interdisciplinary Description of Complex Systems 13*, pp. 34-40, 2015.

[5] Ionut Muntean, Sergiu-Dan Stan, Mihail Abrudean, and Kuo-Ming Chao, "An architecture for Integrated Remote Management and Schedueling," *Future Gener Comp Sy*, 2014 April.

[6] Nandor Verba and Nagy Zoltan, "Arduin Tester," *ACTA Mechanica*, 2013.

[7] Nandor Verba and Nagy Zoltan, "Online Arduino Workshop," *XXIV.-th edition of the SzámOKT*, 2014.

[8] Nandor Verba, Birs Isabela, and Nagy Zoltan, "Microcontroller and RF 434Mhz Communication based Monitoring and Control System," *IEEE International Conference on Automation, Quality and Testing, Robotics AQTR*, 2014.

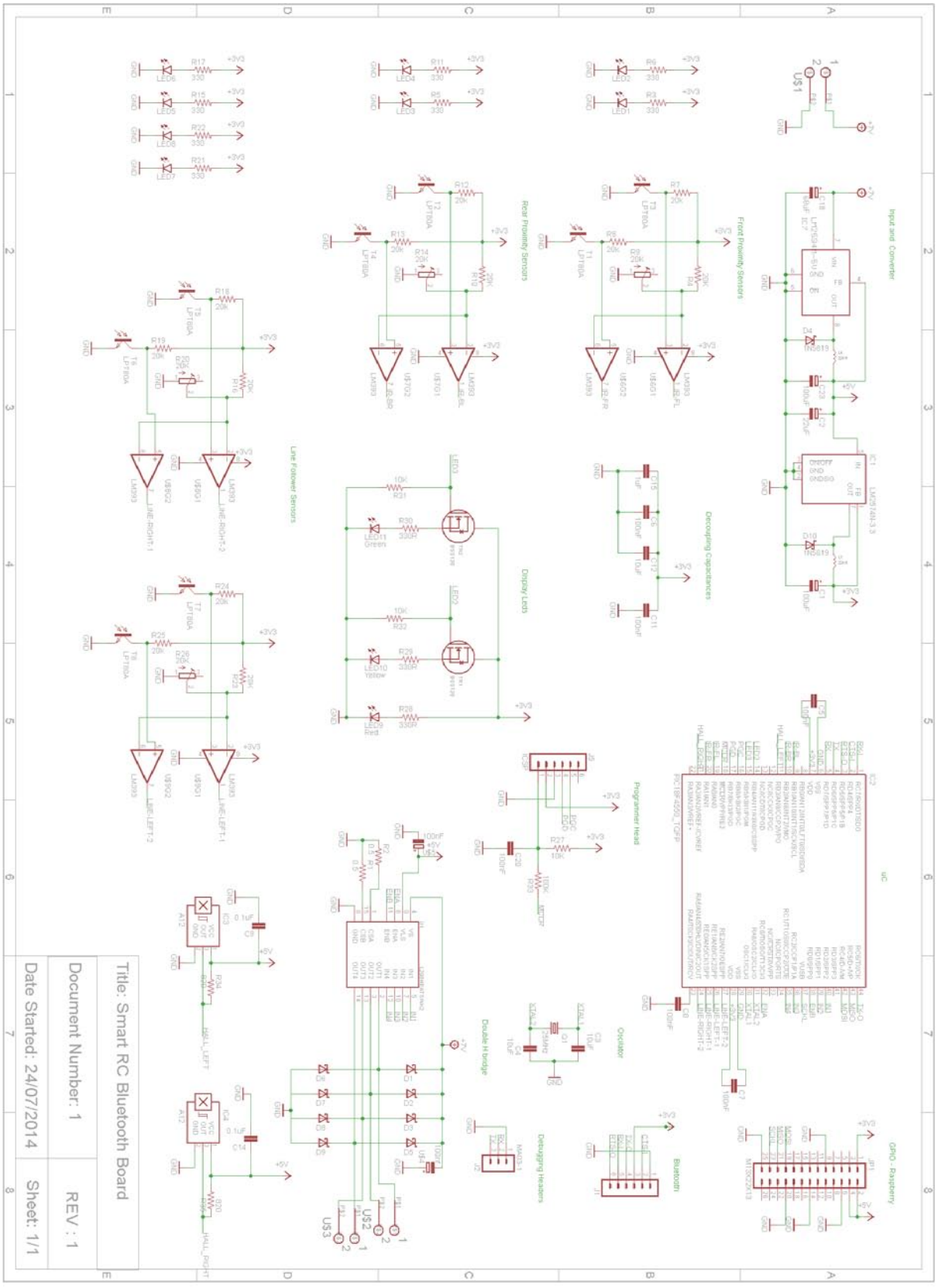
[9] Joan Navaro et al., "A Cloud Robotics Architecture for Foster Individual Child Partnership in Medical Facilities," 2011.

[10] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig, "Implementation for the Project of the Future: Industry 4.0," 2013.

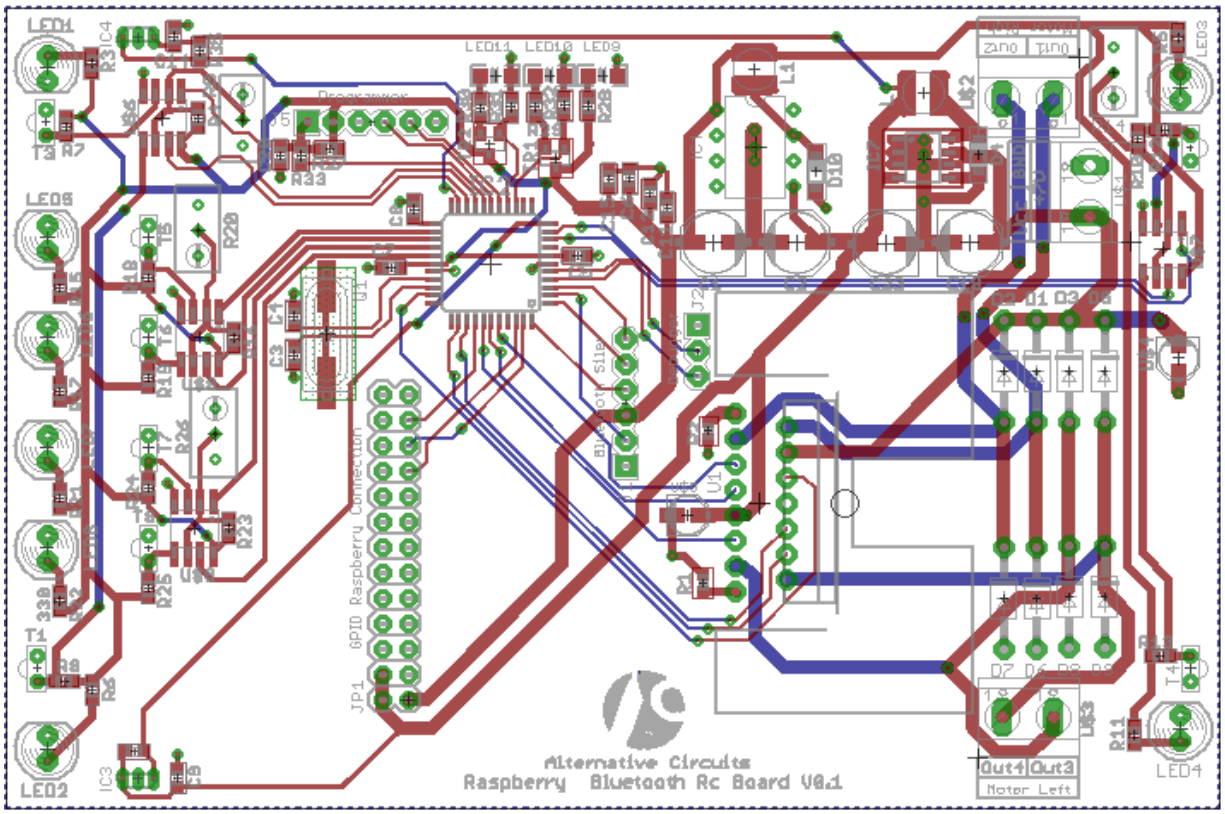
[11] Microchip Coorporation, "PIC18F66K80 Family Data Sheet," 2012.

Appendix A - EAGLE - board and schematics

Board.sch - Schematic file in eagle represents all the connections



Board.brd - Board file that represents the connections and the design



Appendix B - Android Application - Activity Design

BaseActivity - Design of the base activity

MainActivity - Design of the Main activity

PointActivity - Design of the Point to Point control activity

Appendix C - Publications

[1] Nandor Verba, Gabriela Mocanu,*"Equivalent Electric Circuit for a Harmonically Perturbed Accretion Disk around Supermassive Black Holes"*, Studia Unversitatis Babeş-Bolyai Physica, CNCSIS 519, Categorie B+, http://www.studia.ubbcluj.ro/arhiva/cuprins.php?id_editie=799&serie=PHYSICA&nr=1&an=2013

[2] Nandor Verba, Nagy Zoltan, *"ArduinTester"*, Accepted în ACTA Mechanica Universitatea Tehincă din Cluj-Napoca at the Communication Session for the Students of the Faculty of Mechanics, 16 mai 2013, IX-th edition

[3] Nagy Zoltan, Nandor Verba, *"Online Arduino Műhely"* - Published in the XXIV.-th edition of SzámOKT in 2014

[4] Nandor Verba, Nagy Zoltan, Birs Isabela, *"Microcontroller and RF 434Mhz Communication based Monitoring and Control System"* - Accepted in the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics AQTR 2014