MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

AUTONOMOUS VACUUM CLEANER

LICENSE THESIS

Graduate: Alexandru Viorel BONDOR
Supervisor: Assist. Prof. Dr. Eng. Mihai Negru
Assist. Eng. Mircea Mureşan Paul

2016

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. Dr. Eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. Dr. Eng. Rodica POTOLEA

Graduate: **Alexandru Viorel BONDOR**

AUTONOMOUS VACUUM CLEANER

1. **Project proposal:** *Autonomous vacuum cleaner with infra-red proximity sensors and path planning computed with A^* algorithm.*
2. **Project contents:** *Project Context, Project Objectives and Specifications, Bibliographic research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's Manual, Conclusions, Bibliography and Appendices.*
3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:** Assist. Mircea Paul Mureşan
5. **Date of issue of the proposal:** November 1, 2015
6. **Date of delivery:** June 30, 2016

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) _____, legitimat(ă) cu _____ seria _____ nr. _____ CNP _____, autorul lucrării _____

elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă.*

Data _____

Nume, Prenume _____

Semnătura

Contents

List of Tables

List of Figures

Chapter 1

Introduction - Project Context

1.1 Project Context

The concerned project was developed under the fields of *Robotics, System Automation and Artificial Intelligence*. Specifically, the the goal and narrowed down context of the project is described in brief in the subsection 1.1.1.

We are the ones witnessing the evolution of technology at an exponential pace. Robots are evolving around us more and more and artificial intelligence is flourishing. We are living the era of the so called *Internet of Things*. Needless to say, thousands of smart and internet connected devices are born every day. It feels like the revolution of technology is happening and it is easier than ever to build a system that will turn your lights off at the clap of your hands or water your plants at a click of a button.

Given such a context, software together with hardware are evolving and thus robust processing power can be found in smaller and smaller devices. All the above mentioned facts represent the perfect surrounding for the people to automatize every day routines. Automation becomes more and more reliable and less costly. People get accommodated to seeing and using *robots*(automated systems in general) around and thus they win precious time that would have been lost performing all sorts of repetitive tasks.

The demand for smart and connected devices is real and people are willing to pay if a device is useful for them. The evolution of such devices seems to be a bit chaotic as there are a lot of companies competing with each other to provide the best choice for developers.

1.1.1 Narrowed Down Context

The field of above mentioned *Internet of things* is wide. However, the current project can be placed inside a thinner layer from the IoT world: **Home automation systems**. Even deeper, the project places itself in the branch of *Autonomous vacuum cleaners* as it can be seen in figure 1.1.

The structure presented in figure 1.1 represents my personal logical point of view with respect to the *Internet of things* world.

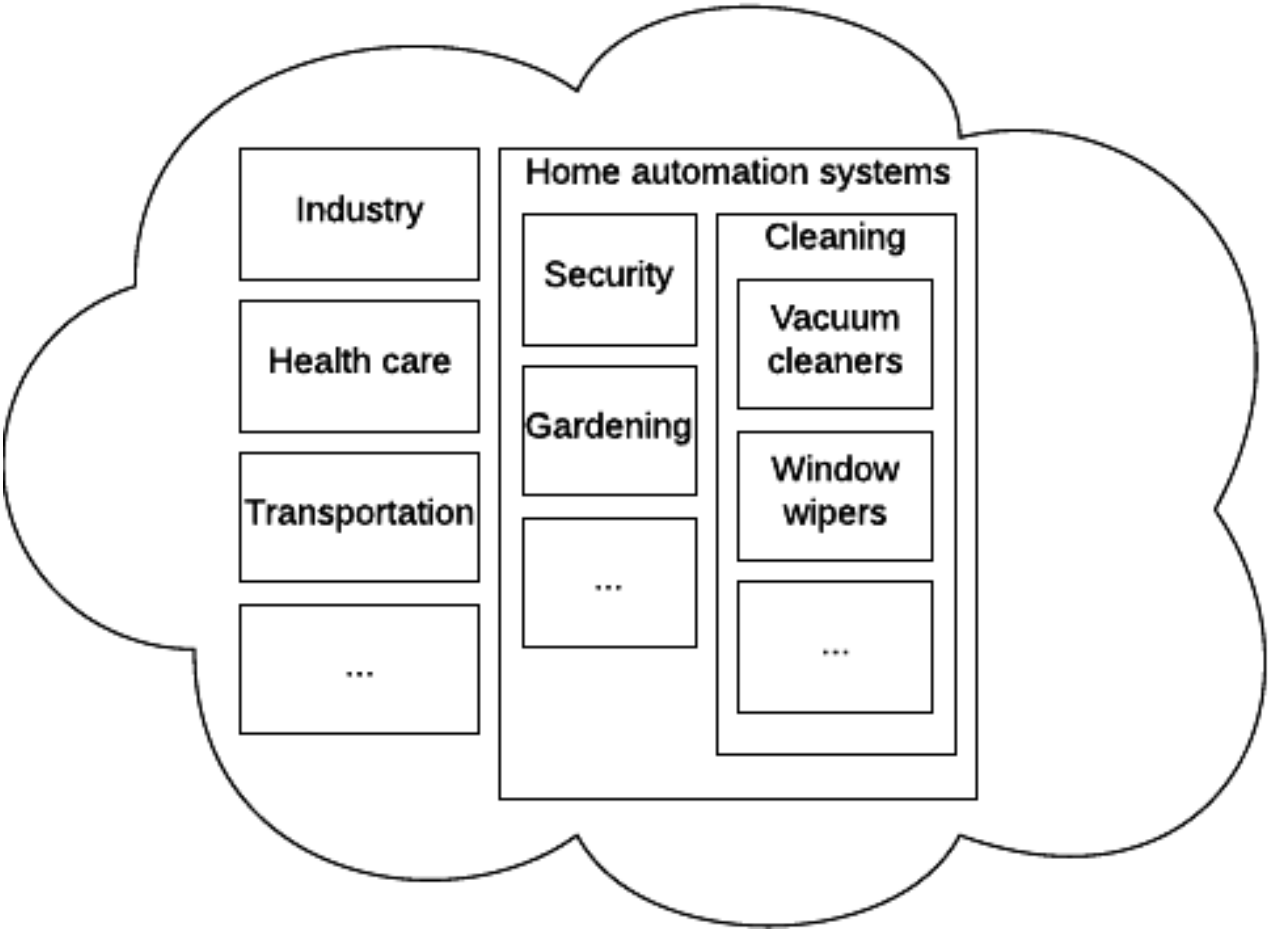


Figure 1.1: IoT logical structure.

The tackled context targets the end users directly and thus the project represents a product that should fit the user needs and expectations.

The purpose of the project was to build an autonomous vacuum cleaner. The end product will be a functional vacuum cleaner which will have the ability to move around a room by its own without any exterior intervention. The functionality of the product involves both the vacuuming and autonomous control part. However, the design and mechanical implementation of the vacuuming component of the product will be kept to a minimal functionality as the other component will be the main target of the whole project.

1.1.2 Motivation

Autonomous vacuum cleaners like Roomba are around us for some time now. However, it was only year 2015 that they considered taking the robots' *brain* to a whole new level. Until that year the robot was finding its way around a room only by using some rudimentary sensors placed in the bumper of the robot. This way, the robot was spending

quite a lot of time cleaning a whole room as it was passing over the same area multiple times without realizing it. Starting from 2015, the new Roomba has an on-board camera which is being used in a SLAM algorithm so that the robot maps the camera while cleaning, thus improving the battery life and radically decreasing the required cleaning time.

Given the above described context and taking into account that a Roomba costs $899.99 at the moment, which seems an exaggerated price from my point of view, I considered that it is worth building such a product.

The goal of the project is to build a functional SLAM based autonomous vacuum cleaner using minimal hardware resources in order to obtain a better product cost while keeping the performance as high as possible.

1.1.3 Competition Overview

Unfortunately the autonomous vacuum cleaning robots on the market are really expensive and not that smart for the imposed price tag. Of course my robot cannot compete against any professional vacuum cleaners as its vacuum system is not the main concern of this project. My robot will compete against other robots with its intelligence. Fortunately, the whole system will act more intelligent than current solutions on the market in different room setups.

Some features of **Roomba iRobot** and **Neato botvac** are presented in table 1.1.

	Roomba iRobot	**Neato botvac**
Self-charging	✓	✓
Web interface	✓	✓
Schedule	✓	✓
Real time control	✗	✗
Interactive display	✗	✓
Systematical cleaning	✗	✓
Multiple rooms	✓	✓
Stair avoidance	✓	✓
Price	$899.99 (on store.irobot.com)	$699.99 (on amazon.com)

Table 1.1: Some features of Roomba iRobot in comparison to Neato botvac.

As you can see from the above table the two analyzed robots have pretty nice features and you might almost say it is something worth buying.. until you see the price. The project I am working on aims to give birth to another smart vacuum cleaner which will try to learn all the features highlighted in the table 1.1 but at the lowest price possible.

Chapter 2

Project Objectives and Specifications

The concerned project tackles some key points of interest in the robotics field. The issues taken into account by this project were meant to be treated as simple as possible while trying to keep the quality of the obtained results at a high level. Some key features that were tried: simultaneous localization and mapping using cheap infrared proximity sensors and magnetic encoders, path planning on a map represented as a matrix, remote Bluetooth mobile device-robot control, WiFi robot-PC communication, I^2C communication between two microcontrollers so that required processing power is met and adaptation of a car vacuum cleaner to a functional Roomba like robot vacuum cleaner.

The robot itself represented a challenge from the point of view of mechanics and hardware design as well as from the point of view of the strong and robust control software it needed in order to operate close to expected standards. The scope of the project was to build a vacuum cleaning robot which should: be able to do the vacuum cleaning task at an acceptable level (e.g. to be able to vacuum some rice grains), have the ability to *sense* and create a virtual 2D map of the environment it is placed in (e.g. detect obstacles such as walls or furniture and mark it accordingly on the 2D map), allow remote control from a mobile device via Bluetooth technology, send real time data to a server hosted on a PC so that the 2D map could be displayed and be able to move autonomously in the mapped environment in order to do the vacuum cleaning task by itself. The software which sends commands to the robot so that it acts autonomously was written on a separate more powerful controller and via I^2C communication the commands are sent to a slave microcontroller which acts directly on end parts of the robot. Given that the control software for all the robots' parts was written on a single core microcontroller it was a challenge to simulate the multithreading effect in order to operate all the devices (sensors, motors, Bluetooth and WiFi module, I^2C communication) in real time.

2.1 Research Proposal

Pursuing a project in the field of *Robotics and Artificial Intelligence*, as it is the case of the current proposed project, requires research in both the hardware and software directions. There are several aspects that have been taken into account from the point of view of research before building the project. The mentioned aspects will be briefly summarized in the following paragraphs.

The project was created from scratch and the robot was custom build in order to fulfill the needs the proposed diploma project was looking for. The initial phase of the building requires prior knowledge about basic mechanics in order to create a good looking and functional frame of the robot. Additional research on the vacuum cleaners working principle is required when trying to build the vacuuming component of the robot.

Moreover, knowledge about electronic components is needed in order to fulfill the task of crafting the electronic custom made components of the robot. Apart from the ready-to-buy microcontrollers there is also a custom made PCB which houses the needed circuit for controlling the vacuum pump and controlling the forward/backward movement of the motors.

In addition to physically connecting the electronic components there are other aspects that needed to be taken into account from the field of communication protocols. Such aspects include the wireless communication between the robot and the PC or phone and the wired communication between the two on-board microcontrollers.

Taking the research one step further but still highly coupled to the hardware component we have some key features from the field of *Systems Theory*. A closed loop system is build on top of the motion ability of the robot in order to fulfill an important specification of the project.

From the point of view of software development there are multiple fields that correlate with each other. Path planning algorithms represent the key element of the autonomous behavior of the robot. Concepts of *Operation System Design* and *Design with Microprocessors* are used for describing the working principle of the final product *brain*.

2.2 Design Proposal

Previous experience with building automated systems as well as research on the Internet influenced the design decisions taken for the project in cause. In order to broaden my personal knowledge I have decided to build the robot from scratch instead of building on top of a ready to buy solution.

The design of the proposed solution aims to provide a high quality standard from both the mechanical and software points of view. As I have already mentioned, the vacuuming component of the robot was not the main concern of the project however, a functional system has been provided after research on vacuum cleaner working principle has been pursued.

Figure 2.1 presents the overview of the electronic circuit of the robot as well as the links between all the composing components of the robot. Every component presented on the mentioned diagram will be thoroughly explained in upcoming section 5 of this document.

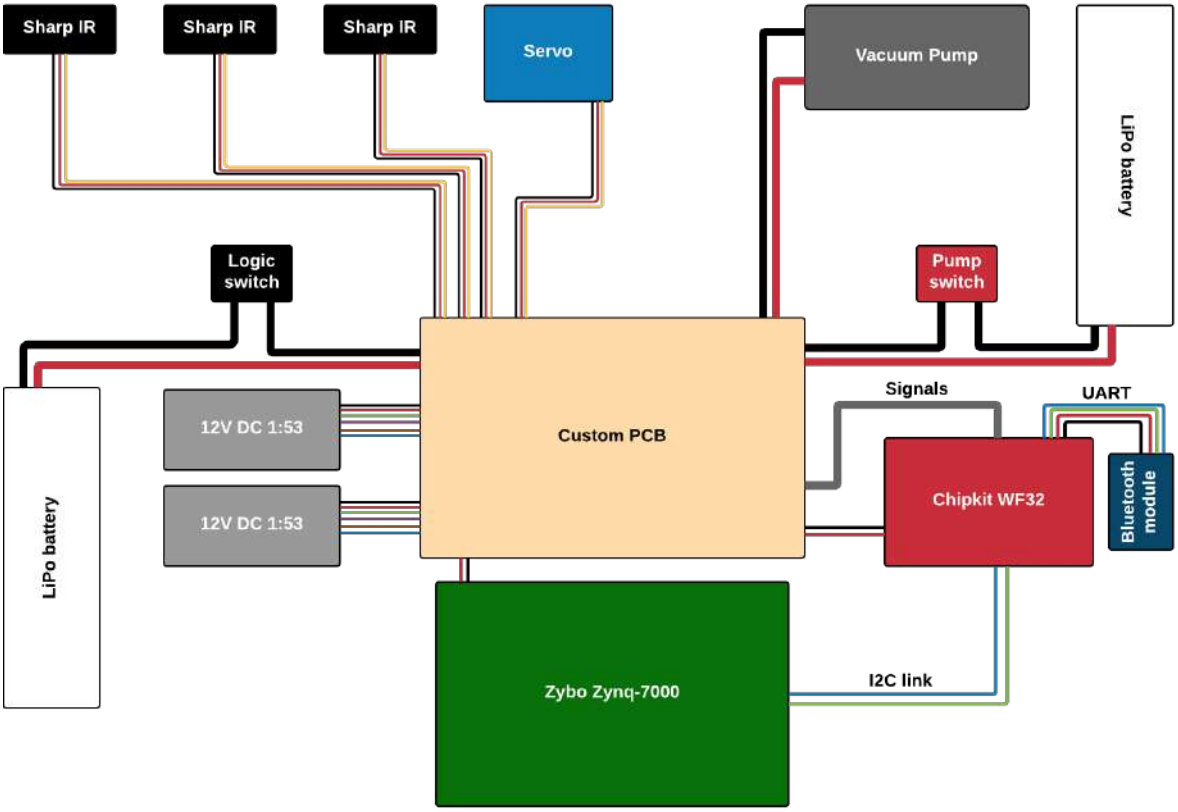


Figure 2.1: Project design overview.

In the following paragraphs the design is broken down in logical components so that a well overview is provided before jumping to the in detail explanations provided in sections 4 and 5.

The design of the robot is split between the hardware and software part. The hardware part is further split between the mechanical and electronic components while the software is split between two microcontrollers. In the following lines the already mentioned components will be briefly explained.

Hardware

Mechanics. The robot is a two-wheel drive one, with the two active wheels placed equidistant to the robots' center in the back and two passive wheels placed in front. The active wheels are driven by two geared motors. The vacuuming component of the robot has a motor and a turbine taken out from a car vacuum cleaner. On top of the robot there

is a servo motor used for rotating the proximity sensors to left and right.

Electronics. The power for the electronics is supplied by two LiPo batteries, one for the two microcontrollers, Bluetooth module, driving motors, sensors servo and proximity sensors while the other one is dedicated entirely to the vacuum pump. Power distribution is made through a custom built PCB and a DC-DC step down converter. One of the microcontrollers control the motors via an H bridge.

Software

Chipkit WF32. This microcontroller is responsible for controlling all the hardware components of the robot as it has direct connection to all of them. The software present on this board is aware of reading and writing data to peripherals so that they operate normally. Different modules like I^2C, Bluetooth & WiFi communication, motor driving and sensors reading is done through the software present on this development board.

Zybo Zynq-7000. Implementing the environment mapping and localization of the robot as well as the further path planning and obstacle avoidance would have been an overkill for the not so powerful Chipkit WF32. That is why it was considered that the implementation of this modules would suit better of the more powerful Zybo Zynq-7000. As the peripherals were already connected to the Chipkit WF32, the second more robust microcontroller uses it as a slave to send the commands required for the completion of the autonomous vacuuming task via its I^2C interface.

2.2.1 Design Decisions

I cannot say that all the build-versus-buy decisions I made for the current project were based on a logical and common sense approach. Some of the decisions I made were simply to fulfill my own knowledge satisfaction and calm down my creative side of the brain. In table 2.1 I have highlighted the build versus buy decisions I have made and the reason for each of them.

Buy alternative	**Built**	**Reason**
Hack a cheap version of autonomous vacuum cleaner	Custom design of autonomous vacuum cleaner made out of plexiglass and custom components	Flexibility and know-your-tool feeling
Expensive LIDAR sensor	Infra-red proximity sensors	Just to prove a point that robust results can be obtained by using cheaper sensors

Table 2.1: Build-versus-buy decisions.

The current project was more of a challenge to build something which was already awesome, an autonomous vacuum cleaner, with less budget that a product you can buy requires. My passion for robots and software got together again and proposed some ideas for my brain to have fun with.

During the development of the project I have encountered some issues that made me change the initial design. The initial design combined with the on-the-spot changes resulted in the final design that is specifically implemented. In table 2.2 I will highlight some particularities of the project and the reason they were implemented this way.

Particularity	**Decision**
Two LiPo batteries instead of only one	The 12V vacuum pump consumes a high amount of current during operation and thus, sensitive electronic components could have encountered problems in normal operation. Moreover, as the robot is a two-wheel drive one, it is crucial that the robots' weight is balanced between the two wheels.
Square like shape of the chassis of the robot	Inspiration from already on the market products, ease of operation and proper fitting of all the components inside the body of the robot.
Sharp sensors mounted on a servo	In order to obtain a similar output to the one a LIDAR sensor would have been provided the existing sharp sensors are mounted on a servo that turns left and right.

Table 2.2: Design particularities.

2.3 Objectives

At a first glance, the objective of the proposed diploma project can be formulated straight forward as follows: build an autonomous vacuum cleaner. A little bit more detailed, the final product will be an automated system that, through sensors, will identify the particularities of the surrounding environment and, upon further processing, will be able to perform the vacuum cleaning procedure by itself.

One of the most important objective of the project was to build a system that will do the job of a vacuum cleaner in the first place. Given so, the vacuum component of the final product represented one of the main points of interest during development.

Furthermore, the robot had to have the ability of autonomous motion in the end. Thus, another aspect of great importance was to create the proper hardware + software

system that would enable the mentioned feature. Algorithms for path planning had to be considered as well as the logic for obstacle avoidance.

Another objective for the project settled in backed up by the nature of the development boards used as part of the hardware components of the robot. Namely, the final phase of the project should provide a way of monitoring the status of the robot and controlling the actions of the robot through a wireless connection.

2.4 Specifications

During the designing phase of the project some key specifications were set. The number and complexity of the features were kept at a manageable level taking into account the time and working power constants. In the following subsection there can be found table 2.3 which highlights the key elements of the proposed diploma project.

The set of specifications were set right after the idea of the project got born. This step represented an important phase in the early development of the project. As for any other project, noting down the specifications provide a clear understanding of the complexity and scheduling of the work to be performed.

The specifications of the proposed solution are represented in table 2.3 as features of the final product. The logic behind the choice of the mentioned specifications was either necessity or comparison with the already existing products.

2.4.1 Features in Brief

Feature	Explanation
Robot simultaneous localization and mapping	The ability of the robot to localize itself while also mapping an unknown environment. The result of this process is a virtual map that the robot is aware of and which correlates one to one to the initially unknown environment.
Bluetooth remote control	An Android application is provided so that the robots' movement, vacuum pump and sensors' rotation can be controlled real-time from any Android device that supports Bluetooth communication.
WiFi feedback	The robot is able to send data real-time via WiFi to a sever hosted on a PC. The sent data represents information about robots' current position, sensor readings and heading. Based on this information a real-time visualizing tool can be created.
Vacuuming component	The robot has a vacuuming component consisting of dust and waste collector bin, a dust filter and vacuum pump which is based on the same principle that a real vacuum cleaner implements.
Autonomous mapping and cleaning	Based on the simultaneous localization and mapping principle, the robot will create a 2D map of any unknown environment and then, by choosing the right path and avoiding any obstacles, will start performing the vacuum cleaning task all by its own.
Fuzzy controller based motion	It is a key feature that the robot moves in a straight line when requested to do so. Due to characteristics of the surface the wheels of the robot can either slip or remain locked for a small period of time. Based on the input taken from the motor encoders a fuzzy controller will compute the output for the motors so that, overall, the robot moves linearly and smooth.

Table 2.3: Features in brief.

Chapter 3

Bibliographic Research

Every projects' development involves a phase in which information is gathered. Information about working principles and algorithms that are considered to be used for the development of the proposed solution were gathered during this phase. In the following sections I will sketch the conclusions I have drawn after doing some research. All the information noted down in the following sections were used for implementation afterwards.

3.1 Odometry

The very first component of the robot that required some research was the odometry component. According to [1] odometry is the basic method of navigation used by most if not all of the robots in order to estimate the vehicle's position.

Paper [1] aims to obtain the position of the robot taking into account the velocity and rate of turn of a differential driven robot. A differential drive robot has two motor wheels; one located on the left side of the robot and the other one on the right side of the robot.

3.1.1 Derivation

From the speeds of the two motors one can draw the following useful information: the rate at which the vehicle in turning with respect to the middle point in between the motors and the rate at which the vehicle is moving forward/backward. The idea mentioned by [1] is that if one has knowledge about the initial position $(x, y, 0)$ and the motions of the motors for a θ period of time, then the position $(x^{'}, y^{'}, \theta)$ can be easily determined.

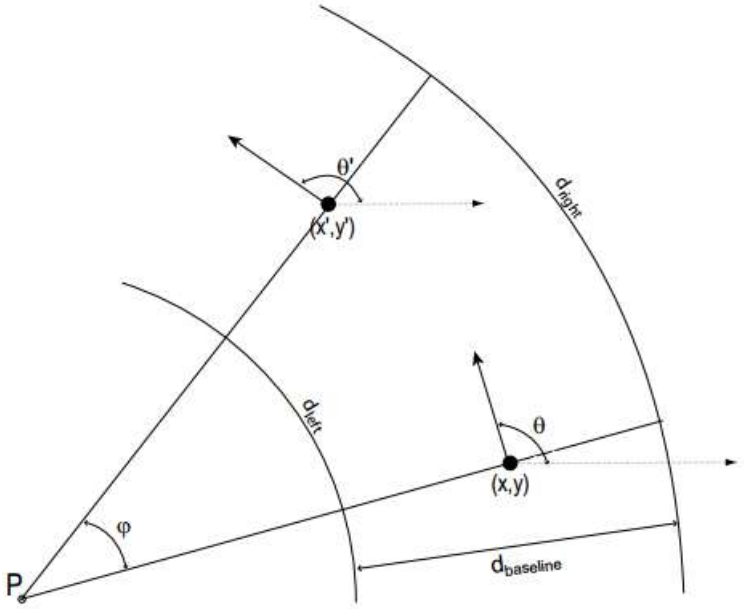


Figure 3.1: Odometry geometry. As found in [1].

The problem that the figure 3.1 poses is to determine the new position of the robot given the initial one and the motion of the robot. From [1] the following mathematical equations are drawn:

$$d_{\text{center}} = \frac{d_{\text{left}} + d_{\text{right}}}{2} \tag{3.1}$$

$$\phi = \frac{d_{\text{right}} - d_{\text{left}}}{d_{\text{baseline}}} \tag{3.2}$$

$$\theta^{'} = \theta + \phi; \tag{3.3}$$

$$x^{'} = x + d_{\text{center}} cos(\theta) \tag{3.4}$$

$$y^{'} = y + d_{\text{center}} sin(\theta) \tag{3.5}$$

3.1.2 Magnetic Encoders

Same paper [1] suggests the ways for measuring the angular velocity of a motor. From the multiple number of ways to perform the mentioned action I have chosen quadrature phase magnetic encoders.

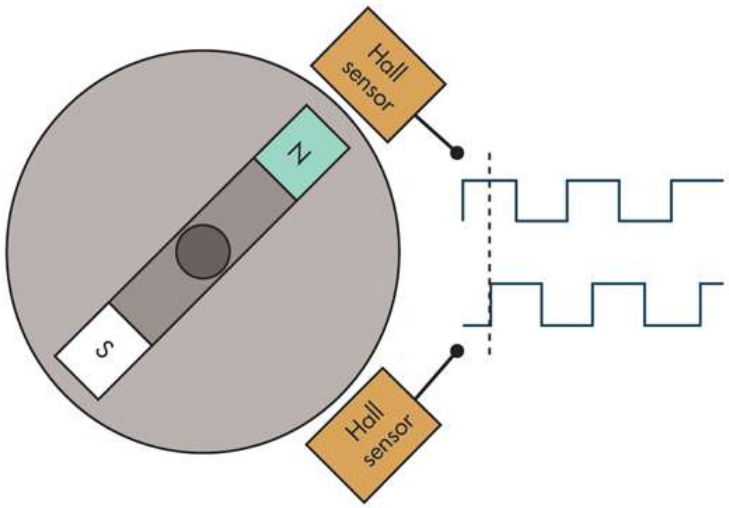


Figure 3.2: Quadrature phase magnetic encoders. Image present in [8].

Figure 3.2 sketches the internal composition of a magnetic encoder. As you can see, two Hall effect sensors and two permanent magnets of different polarities are used. The two sensors are placed in such a manner that they produce waveforms 90 degrees out-of-phase as you can see in picture 3.2. When the motor shaft rotates in one direction then the signal from one lead will be shifted with 90 degrees in front of the other one. When the motor is rotating in the other direction then the opposite will happen.

By taking into account the frequency of the generated signal we can determine the speed of the shaft. If we take into account the phase difference between the two signals generated by the two Hall effect sensors then we can determine the direction of rotation of the motor shaft.

3.2 SLAM

SLAM stands for Simultaneous Localization And Mapping and it is an important and widespread problem in the world of *Robotics*. According to reference [2] SLAM is more of a concept than a single and concise algorithm. Given that, there are many ways in which this concept can be implemented and each of the solutions has its pros and cons.

3.2.1 Autonomous Navigation

Reference [3] states that an autonomous robot is a mechanical piece of equipment that can perform a finite set of specific tasks without continuous human guidance. From this definition we can draw the conclusion that most robotic tasks can be considered autonomous. Generally speaking, for a robot that performs SLAM, autonomy is defined as the ability to identify particularities of the surrounding environment in order to perform the required actions accordingly.

3.2.2 Dead Reckoning Localization

The most basic form of computing the position of the robot at a specific period in time is to update its position taking into account the sensed motion. By following this path, the concept of SLAM is still implemented even though the system is more error prone than other more complex implementation.

Paper [3] defines odometry as being one of the methods to be chosen when doing dead reckoning localization. When the robot moves the distance traveled is computed and the position of the robot is updated accordingly.

3.2.3 Occupancy Grid

Another important aspect when performing the SLAM algorithm is to gather and interpret data from proximity sensors. The raw data is gathered and then interpreted. Figure 3.3 displays a matrix of cells in which the color of each cell represents the probability of the cell of being a real obstacle. White cells represent a probability of 0% and black cells represent a probability of 100% of being an obstacle.

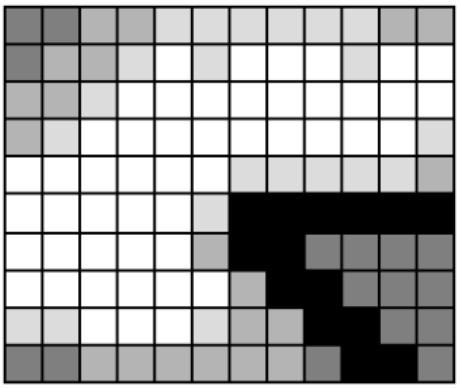


Figure 3.3: An occupancy grid as presented in [3].

3.3 Fuzzy Controller

It is often the case in real world situations that members of the same set to be impossible to be filtered in only two specific and discrete values like *white* - *black* or *0* - *1*. For example, lets take into account the value of the temperature outside: we cannot say for values under 0°C that it is *cold* and for the ones above this value that it is *hot*. There are definitely more adjectives under which we can place the range of values outside temperatures can take. This is the time and place where we know that the need for a Fuzzy Controller is real.

Fuzzy controller comes with a new concept according to which the values from a set can be separated under more than two features; for our example with temperature we can have values under the incidence of *freezing cold*, *very cold* and so on.

3.3.1 Heuristic Method

Paper [4] denotes Fuzzy Controller as being the heuristic method of the traditional control approach that requires modeling of the physical reality. A heuristic rule is a logical implication of the form: **If** <condition> **Then** <consequence> or, in a typical control situation: **If** <condition> **Then** <action>.

Fuzzy control strategies are based more on experience and experiments rather than on a mathematical model. Fuzzy control consists of several inputs that, based on several cause and effect prior reasonings, are interpreted and the output is computed accordingly. For example, if it was to turn the heater on when the temperature drops a certain level then a human like behavior would be performed by the system implemented with a fuzzy controller. The system would most probably have a rule like: If temperature_level=TOO_COLD Then TURN_ON_HEATER. For the mentioned example the key aspect is to provide the suitable sets and rules. More about fuzzy sets is explained in the following section.

3.3.2 Fuzzy Sets

Reference [5] describes fuzzy sets as follows:

Let X be a space of points. A *fuzzy set* A in X is characterized by a *membership function f(x)* which associates with each point in X a real number in the interval [0, 1], with the value of *f(x)* at x representing the "grade of membership" of x in A. Moreover, in the case of fuzzy controller, the whole space of points is separated in several fuzzy sets such that the sum of "grades of membership" of each point x from X in all of the sets is always 1. The separation of the whole domain can be better understood if taking into account figure 3.4 shown below.

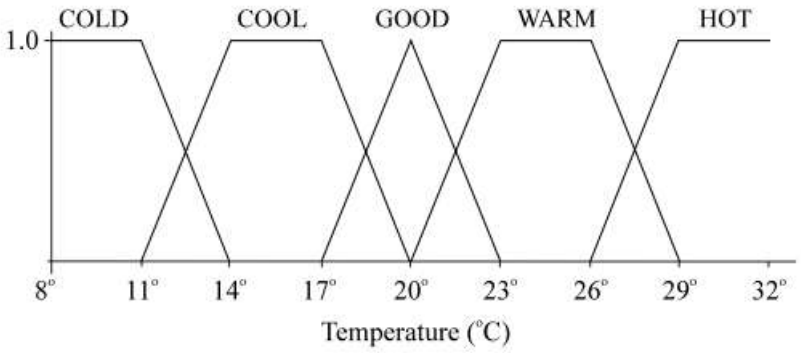


Figure 3.4: Fuzzy sets example. Image extracted from [4].

3.3.3 Working Principle

In [4] the figure 3.5 is displayed to provide a better understanding of the working principle of a fuzzy controller. The following steps are performed by such a controller in order to transform the crisp input in the required defuzzified output:

- Convert crisp input into fuzzy value
- Based on decision making logic and knowledge base the output of each fuzzy IF-THEN rules is computed
- The values obtained at the previous step are combined to obtain the requested output

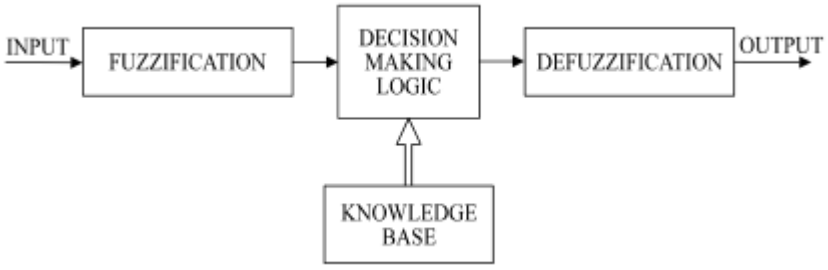


Figure 3.5: Fuzzy controller block diagram. Image extracted from [4].

The steps to be followed when creating a rule based fuzzy system:

1. Identify the inputs and their ranges and name them
2. Identify the outputs and their ranges and name them
3. Create the fuzzification function for each input-output pair of values

4. Determine the rules the system will work with

5. Combine the rules and deffuzify the output

3.4 A* Algorithm

Book [6] names A^* *algorithm* as *Dijkstra's Algorithm with a Twist*. Taking this into account I won't explain the already well known Dijkstra's algorithm for path planning but only mention the particularities the A^* *algorithm* has.

3.4.1 Dijkstra's Algorithm Minus

Figure 3.6 taken from book [6] highlights the most important defect that Dijkstra's algorithm has. Taking into account that in real time applications the speed of computation is really important and the fact that embedded platforms do not provide powerful resources Dijkstra's algorithm is not suitable for being implemented on embedded platforms. As you can see in picture 3.6 the algorithm analyses an awful amount of edges before it can get to the requested destination.

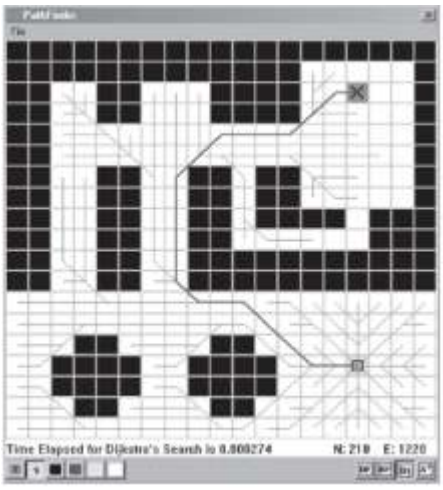


Figure 3.6: Dijkstra's algorithm generated path and considered nodes. Figure taken from [6].

3.4.2 Working Principle

The solution A^* *algorithm* is proposing to the problem encountered by Dijkstra's algorithm when searching for a path is to also take into account an *estimated* distance from neighbor node to destination before adding it to the frontier. This way, the algorithm ensures that it will not consider nodes on the frontier that will make the path get farther

away from the destination point. The result of the applied algorithm can be seen in figure 3.7. We can definitely see that the number of nodes considered by the A* implementation is considerably smaller than the implementation of Dijkstra's algorithm. At a first glance this might not appear to represent such a big deal, however, taking into account that such a computation is done at each a few milliseconds on an embedded system, then we will see that the computation time is greatly reduced.

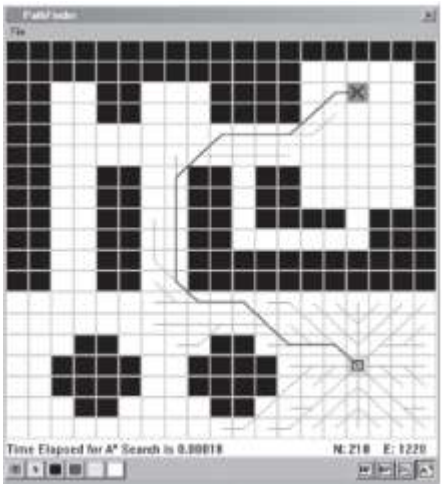


Figure 3.7: A* algorithm generated path and considered nodes. Figure taken from [6].

3.4.3 Algorithm Heuristic

As a heuristic for the *A* algorithm* it is recommended to use Manhattan distance over Euclidean one due to computation time. As you can see in picture 3.8 for computing the distance between x and y in the case of Euclidean distance it is required to do the computation of a square root which is resource consuming.

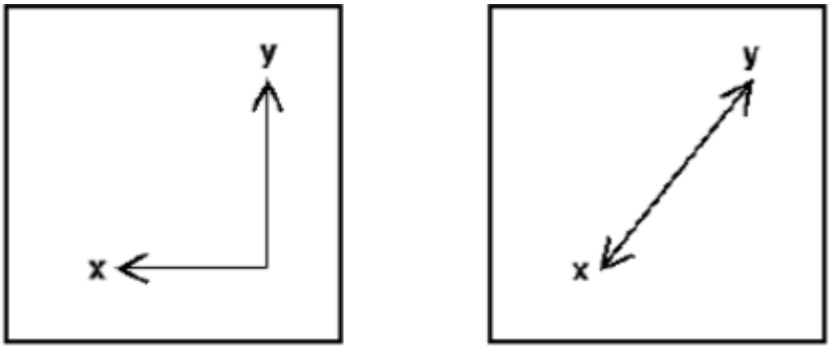


Figure 3.8: Manhattan(left) vs Euclidean(right) distances. Image from [9].

Chapter 4

Analysis and Theoretical Foundation

In the following sections and subsections I will detail the proposed solution's key aspects from a theoretical point of view. The next sections will highlight some aspects such as algorithms and communication protocols used as well as an overview of the structure of the software application.

4.1 Structure Overview

The application running on the Chipkit WF32 controller was written in C/C++ and is responsible for controlling and gathering information from the hardware components of the robot. Due to its nature, the microcontroller makes it possible for the developers to use Object Oriented Programming concepts. For a better understanding and better organized code I have chosen to create and link in controllers objects that are an abstraction of physical components of the robot.

In figure 4.1 you can see the UML diagram of the software running on the Chipkit development board. As you can see the hardware components were abstractized in objects inside the context of the software. This was done for a better manageability and understandability of the code. The concepts of high cohesion and low coupling were taken into account when developing this application.

As you can see, the components were assigned to manager controllers which are responsible for controlling and gathering information from the components. As one can see, each of the controllers have a method called *process*. This method is called every a few milliseconds and each of the controllers does the required steps at each call.

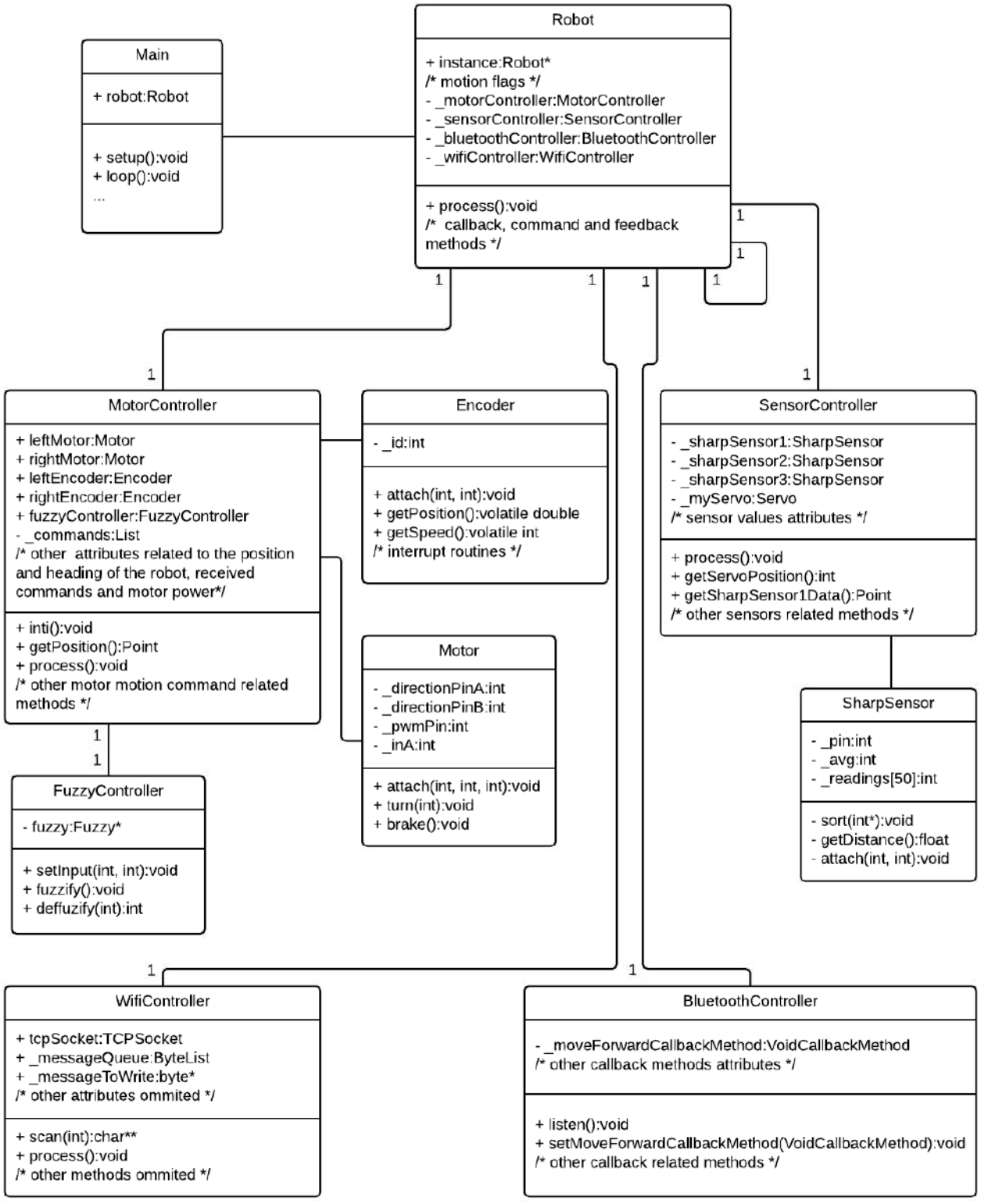


Figure 4.1: UML diagram of software running on Chipkit WF32.

As the Chipkit WF32 is not so powerful and taking into account that a good amount of code has to be handled only for controlling and monitoring the hardware components, I have chosen to implement the autonomous behavior on the Zynq board. This is more powerful than the already mentioned one and thus the implementation the the path planning was considered to be implemented on this specific board.

Apart from the software running on the two development boards the robot is provided with, there is also a part written in Java as what is called *Processing Server*. A client-server architecture was considered in the provided solution in which the application hosted on the PC acts as the TCP server and the robot acts as the client. The connection is established wireless thanks to the WiFi chip present on the Chipkit WF32 board.

4.2 Algorithms Used

In the next two subsections I will conclude on the decision making and applicability of the algorithms used in implementation of the software.

4.2.1 Fuzzy Controller

The reason a closed feedback loop was needed was tied to the requirement of the robot to move in a straight line when commanded so. If the motors are controlled at the same PWM value then, due to motor winding imperfections, the motors will not execute the same angular motion. This was the main issue that made me think that a sort of closed feedback loop system such those studied at *Systems Theory* was needed.

From previous experience I have first considered the PID (Proportional Integral Derivative) controller. However, it seemed impossible to set the right K_p, K_i *and* K_d parameters such that the system behaved as expected.

At the moment of consideration my diploma project advisor suggested I should take into account implementing a *Fuzzy Controller* instead.

The main objective of the *Fuzzy Controller* is to output the right PWM value to be written on each of the two motors of the robot given as input the values provided by the magnetic encoders located at the end of the motors. The encoders are recording the angular motion of the shaft of each of the motor and provide some information of its rotation.

The main challenge of the *Fuzzy Controller* was to find the right ranges for the *Fuzzy Sets*. Fuzzy sets are the most important components of a Fuzzy controller before defining the as well important Fuzzy rules for the controller. The advantage of the Fuzzy controller over the PID controller was that the ranges for the Fuzzy sets are more easy to be found and set accordingly than the values required by the PID controller.

4.2.2 Fuzzy Sets

The Fuzzy controller takes *decisions* based on the initial sets. As the components that needed to be controlled were the motors of the robot I decided that, based on the RPM (Revolutions Per Minute) each of the motor describes a PWM (Pulse Width Modulation) value should be computed by the Fuzzy controller.

Given the above mentioned situation the solution was to create multiple sets for both RPM and PWM values based on which the Fuzzy rules could further be set.

The whole range of values for both of the features that were taken into account were separated in sets where, for each of the sets, one the following attributes are representative: *low, little_low, right, little_high, high.*

4.2.3 Fuzzy Rules

The input of the Fuzzy controller is put against a set of rules based on which the output is then generated. The rules were set based on human-like language and logical thinking. Given so, in table 4.1, the reasoning of the Fuzzy controller is presented.

RPM attribute (input)	PWM attribute (output)
low	high
little_low	little_high
right	right
little_high	little_low
high	low

Table 4.1: Fuzzy rules used by the fuzzy controller.

4.2.4 A*

According to [7] there is a family of heuristic-based algorithms used for path planning in the real world. There are several graph search algorithms used for calculating the least-cost paths on a weighted graph and, according to [7], two popular ones are Dijkstra's algorithm and A*.

In the proposed solution the bare A* algorithm is used. On the other hand, it was not sufficient to provide the optimal path as the robot will gradually identify the environment in order to fulfill the SLAM concept. This represents an issue in the case of bare implementation as, by following the initial computed path, the robot might bump into an obstacle at any moment in time as it moves on the map.

A solution to the above mentioned problem would have been to use an incremental replanning algorithm as suggested by [7]. The most basic way of performing the replanning is to simply recompute the path using the same A* algorithm but starting from the updated position of the robot. There are also other considerations into the direction of replanning

algorithms but, in order to avoid complexity, I have chosen to go forward with the above mentioned solution.

The cons of doing the replanning the way I have described above is that it might be computational intensive for an embedded system. However, the overhead of the computation is dodged by not doing the recomputation continuously but at every few seconds in time when requested.

4.3 Communication Protocols Used

Several communication protocols were considered for implementation of the proposed solution. There were many components that needed to communicate one with each other. Most of the communications are provided by third party libraries and so, the development was done at a higher level even though the proposed solution has a hardware component. Some particularities of communication protocols used are mentioned in the next subsections.

4.3.1 UART

Straight forward communication protocol widely used is also spread into the proposed solution. It consists of two hardware lines **RX** and **TX** for receiving and transmitting data respectively. The UART (Universal asynchronous receiver/transmitter) communication is based on sending and receiving data frames between the two communication ends.

This specific communication protocol was mainly chosen because it only requires two lines of data and the environment in which the software application of the robot was implemented already provides the so called *Serial* library. Through this library the setup of the communication is done in one line of code and, if hardware connections are in place, the two ends can start communicating with each other.

UART is used for the communication between the Bluetooth module and the Chipkit WF32 as that is the only interface provided by the Bluetooth module for communication. The baud rate specific to this communication is 9600 as stated by the official documentation of the Bluetooth module. The datasheet of the Bluetooth module is referenced in [14].

4.3.2 TCP/IP

One of the specifications of the current project was the ability of remote monitoring of the robot. This is achieved by wireless communication from the robot to an application hosted on a PC. The simple infrastructure is sketched in figure 4.2.

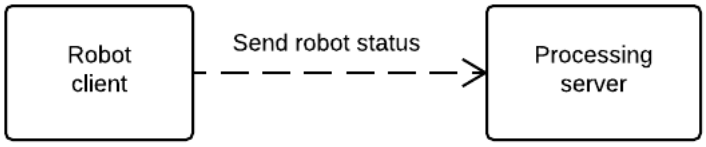


Figure 4.2: Simple client-server architecture.

The connection between the robot and the application hosted on the PC is based on a TCP (Transmission Control Protocol) communication protocol. This is widely used in network communications as its strong point represents the resending of lost packages in contrast to the UDP (User Datagram Protocol) protocol. The monitoring of the robot represents an important feature and all the states of the robot must be logged. Given so, we could not agree to lose intermediary states by using a UDP connection and so we chose the TCP communication protocol.

4.3.3 I^2C

This communication protocol is also widely used in the embedded system world. It consists of only two hardware lines as it was the case of previously mentioned UART. The two lines this protocol uses are **SDA** (data line) and **SCL** (clock line). The provided clock line assures the synchronization between the communication lines. One of the strong points of this specific protocol is the ability to have more slaves on the same line. In this case, the two lines act as a bus in which the slaves are identified by a unique address.

The main reason I have chosen this protocol over its competitor SPI (Serial Peripheral Interface) was the low level of complexity in implementing it on the Zynq development board. As it was already the case of UART communication protocol, a third party library was also provided in the case of I^2C protocol.

This specific protocol was chosen for communication between the Zynq and Chipkit WF32 boards.

4.4 Vacuum System 3D model

It might not have been my duty to build the vacuum system but, as the purpose of my project is to build an autonomous *vacuum cleaner*, then it should have the basic usage of a vacuum cleaner as well. In the next sections I will compare the three versions of vacuum system I came up with and decide which is the best, thus remaining the vacuum system of the final robot.

4.4.1 Versions

In the next pictures I will present the three versions of the vacuum system as 3D models and highlight the main components.

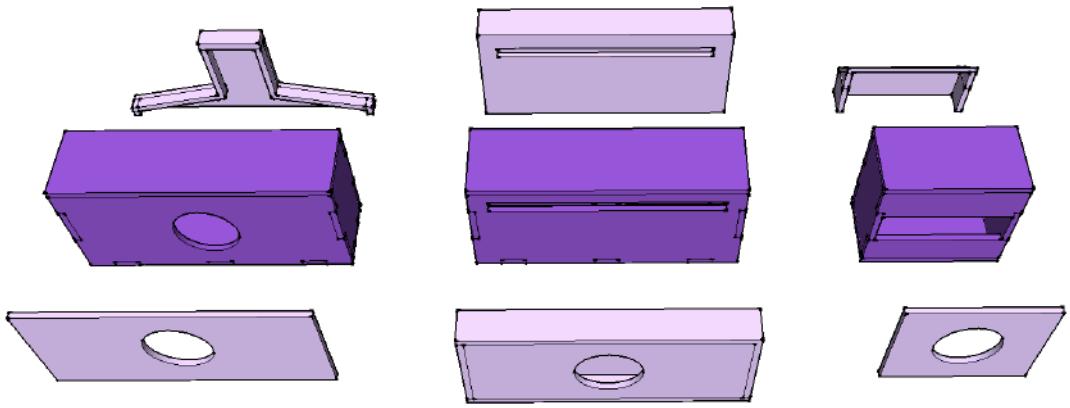


Figure 4.3: Back view of **v3**, **v2**, **v1**.

The main reason the system is modular as you can see in pictures 4.3 and 4.4 is because of the need to empty the dust tray every now and then. The dust tray is the dark purple box located in the middle of each of the systems' versions. The back of the structure presents an adapter between the vacuum pump and the dust tray. In the front of the system there is the hose through which the dust and garbage are collected.

In order to test the prototypes show above I have build them out of cardboard and tested whether the system could suck a bunch of grains of rice.

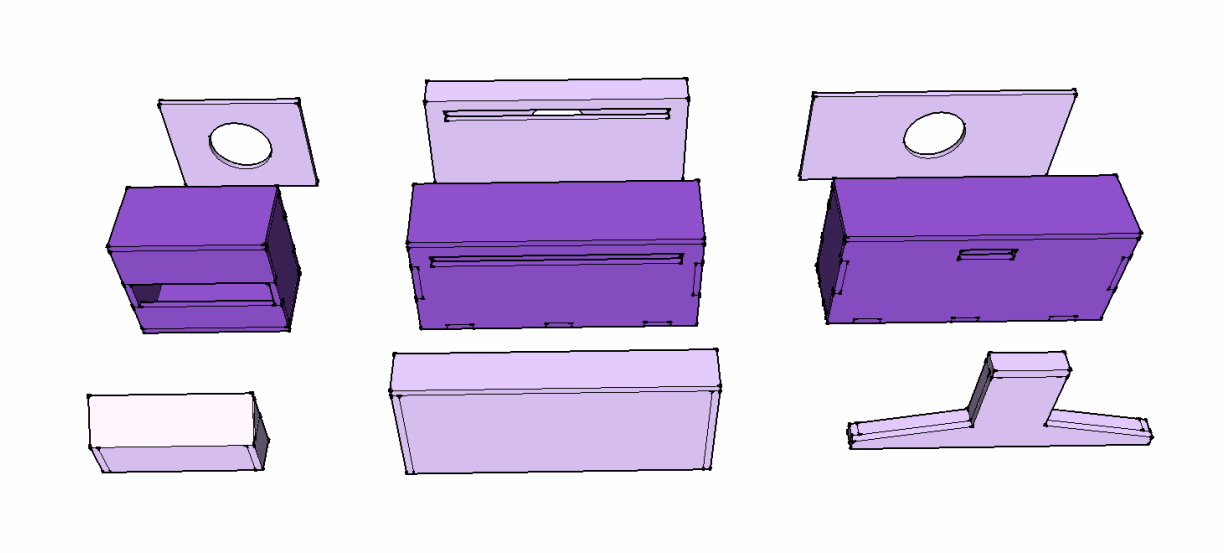
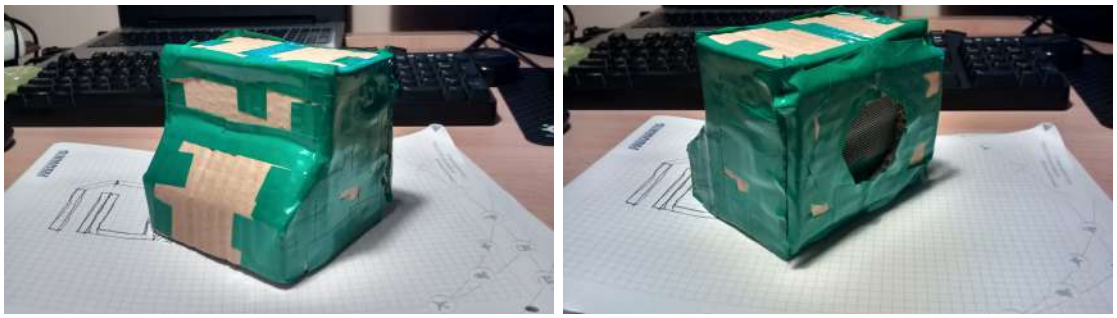


Figure 4.4: Front view of **v1**, **v2**, **v3**.

4.4.2 Vacuum System v1



(a) Version 1 side-front view.

(b) Version 1 side-back view.

The first version of the sucking system did quite a good job when testing it. However, there were some issues that made this system not feasible. The issues will be explained by taking the next picture as reference:

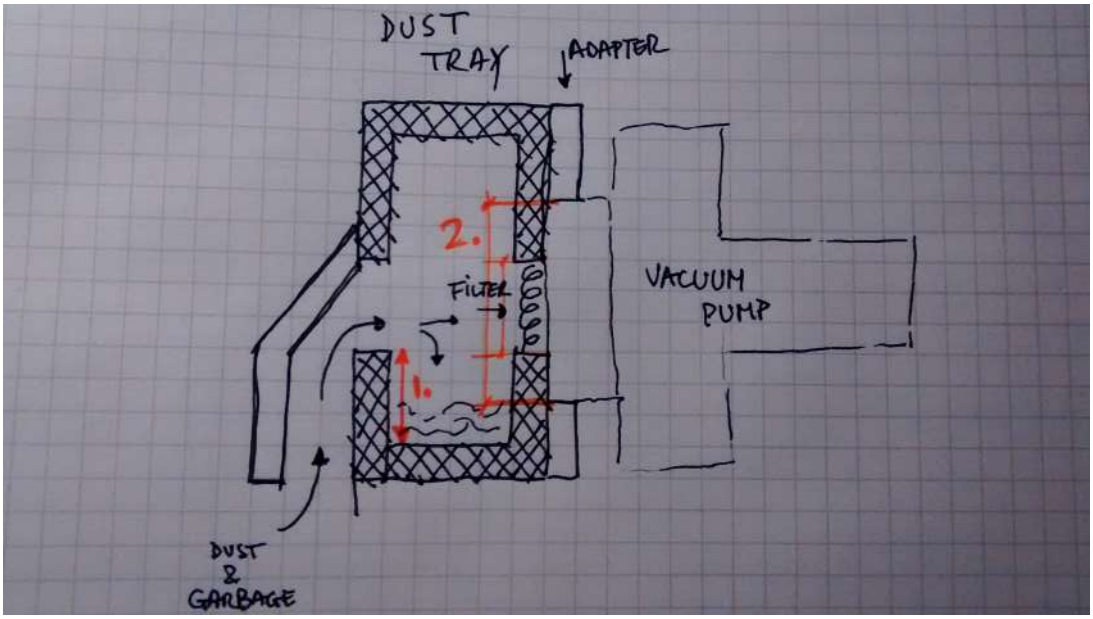


Figure 4.6: Version 1 side view section.

Issue **no. 1** is highlighted with orange and labeled with **1.** in figure 4.6. This dimension was too small and it was unpractical to empty the dust tray as all the collected garbage would have got out through that opening. This way the dust tray wouldn't have been able to be cleaned properly inside.

Issue **no. 2** is highlighted with orange and labeled with **2.** in figure 4.6. The opening from the adapter and the one from the back of the dust tray won't match in size and shape as it can be better seen on the right of 4.3. This way, the flow created by the vacuum pump wouldn't have been used at its' maximum capacity.

4.4.3 Vacuum System v2

(a) Version 2 side-front view.

(b) Version 2 side-back view.

Obviously, with version **no. 2** I have tried to solve the issues encountered at the previous step as described in the subsection 4.4.2.

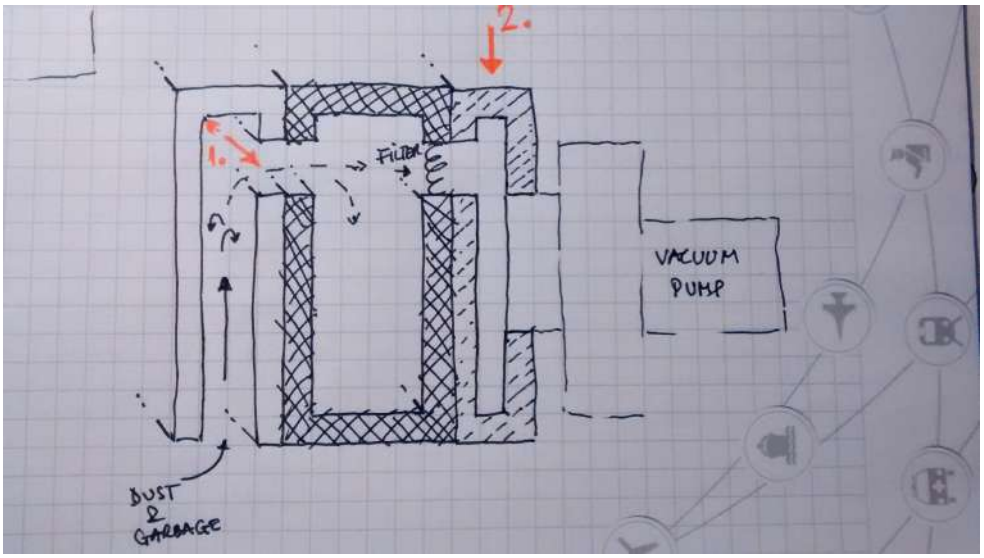


Figure 4.8: Version 2 side view section.

Unfortunately, the new design did not work as expected. After seeing how well the first version was performing when absorbing garbage I said that I could make the hose wider at it was only 10cm, yet impractical for cleaning the floor fast. I've decided to make it 20cm this time and another issue appeared. As you can see in the figure 4.8 the issue is highlighted with orange and labeled with **1**. The front opening of the dust collector and the back opening of the front hose were too wide so the pump did not have enough pulling force to lift the rice grains all the way inside the dust collector.

Moreover, when trying to solve the second issue of the first version I have introduced a redundant "chamber" at the adapter which presented the same issue as the one describer above.

4.4.4 Vacuum System v3

The last and final version of the vacuum system combines the first two versions together in order to obtain a feasible sucking structure for the final shape of Wilkie.

(a) Version 3 side-front view.

(b) Version 3 side-back view.

Inspired by the real vacuum cleaners, I have reshaped the front component of the system. The transformation is highlighted in the next picture:

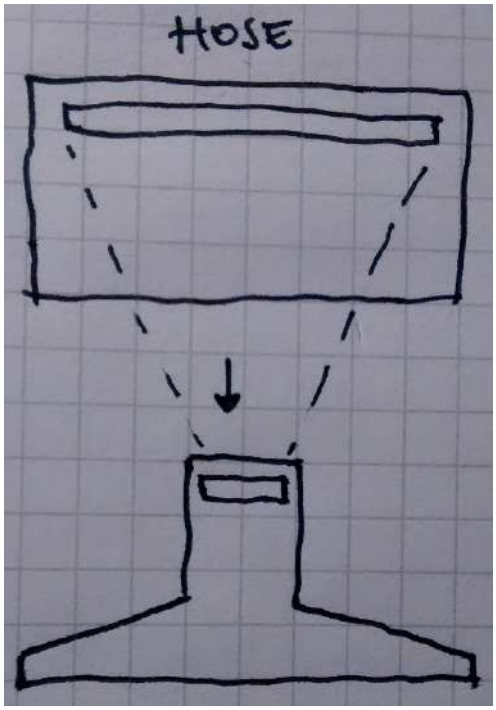


Figure 4.10: Hose improvement from v2 to v3.

As the first version of the system was performing very well at absorbing garbage I've decided to make the absorption hose even narrower than the one in the first version. In order to keep the wide range of sucking tried with the second version the hose gets wider towards bottom.

Another issue that had to be resolved with the third try was the adapter between the vacuum pump and the dust collector. This was simply achieved by getting rid of the "adapter chamber" (fig. 4.8) introduced in version two.

The changes made at the level of dust collector can be seen in the next picture:

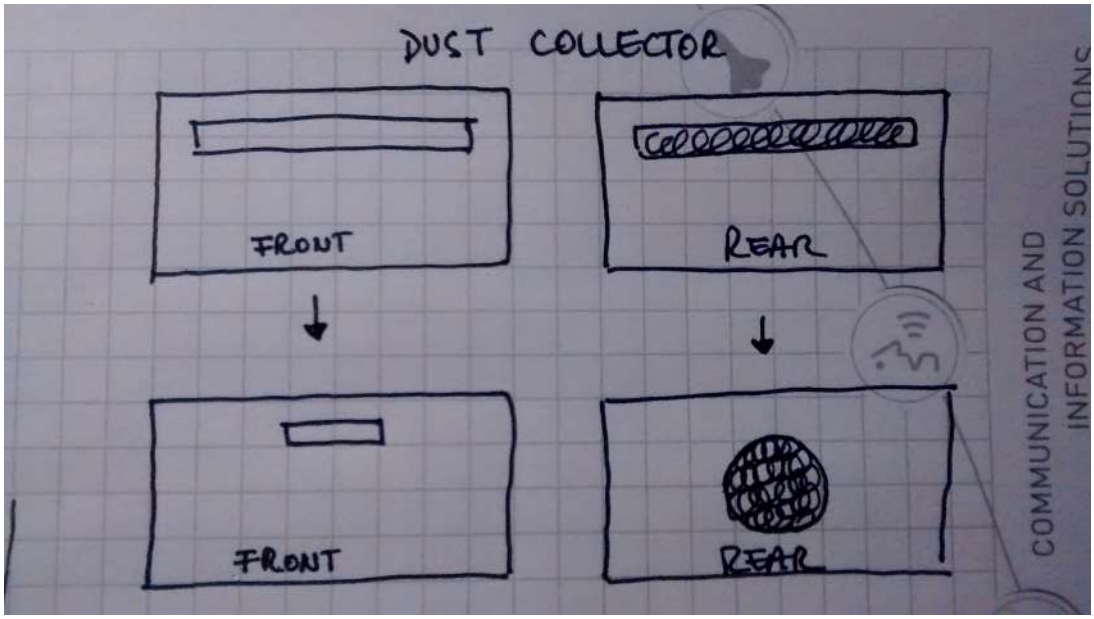


Figure 4.11: Dust collector improvement from v2 to v3.

4.4.5 Testing

For testing out the systems described in the previous subsection 4.4.1 I have used the vacuum pump of a 12V car vacuum cleaner which will be the pump used in the final version of the robot. In the final version of the robot only the motor and propeller of the device presented in the next picture will be used:

Figure 4.12: 12V car vacuum pump.

The test consisted in building the systems described in the previous subsections 4.4.1 and then trying them out in cleaning a floor full of rice grains. The structures were

build out of cardboard at their real dimensions with negligible dimension variations due to human faults during cutting and assembling of the structure.

4.4.6 Results

The winner of the tests was the third version of the vacuum system presented in subsection 4.4.4. The final version of the system will be CNC cut using the dimensions and shapes used for building 4.4.4. Although the reference structure performed very well on tests there are still some issues to be taken into consideration. In the next figure you can see the inside of the dust and garbage collector after tests performed with version 4.4.4:

Figure 4.13: Inside of dust and garbage collector.

From the above picture you can notice that the dust got stuck into the ”filter” I have used and in short time the pump wouldn't have had enough force to absorb any other garbage. A solution like using a thicker filter will be considered in the upcoming models of the robot.

The final vacuum system is presented in the next picture as a snapshot of the 3D model:

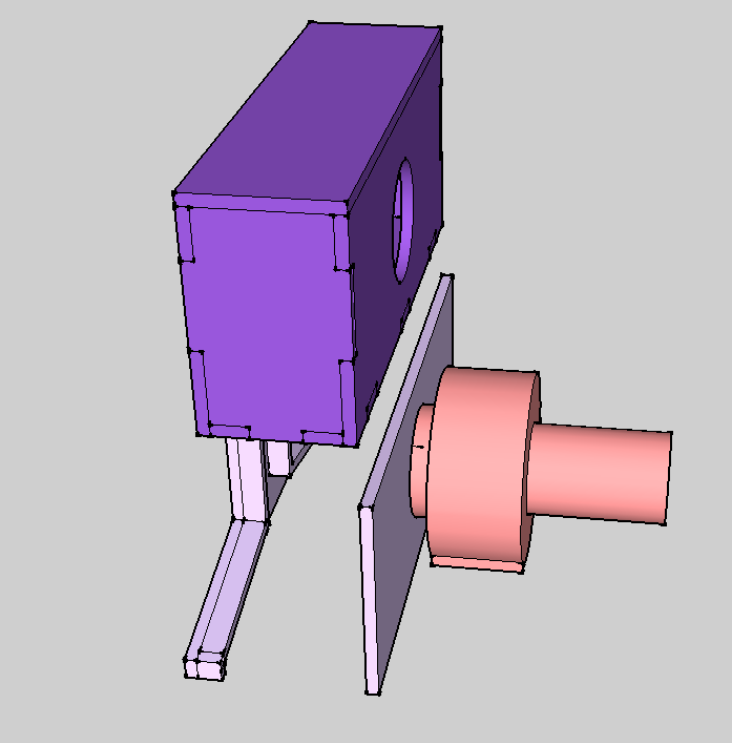


Figure 4.14: Dust collector removal.

As you can see in picture 4.14, the dust collector will be removable in order to facilitate the cleaning of the inside of the tray. The other two parts of the system will be fixed on the body of the robot.

Chapter 5

Detailed Design and Implementation

5.1 Design Overview

In order to have a proper understanding of the whole system, a top level design diagram is showed in picture 5.1.

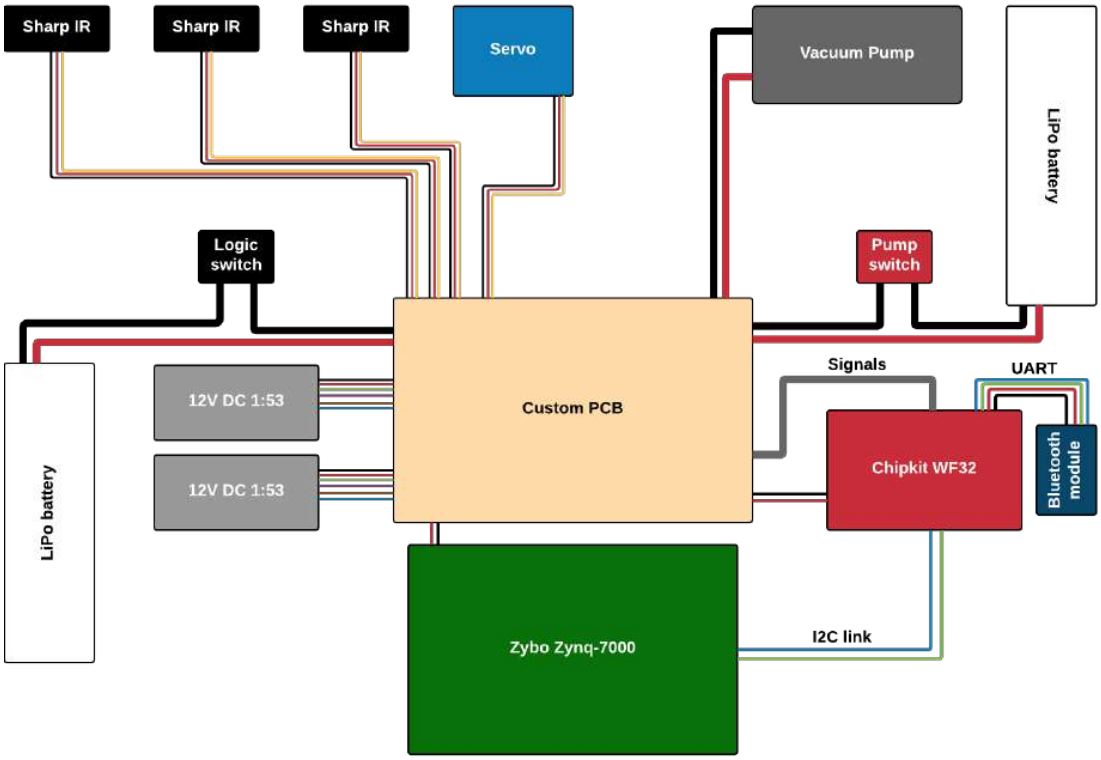


Figure 5.1: Project design overview.

In the upcoming section I will take apart the main components of the whole system and explain them in detail.

5.2 Detailed Design Description

In the following subsections I will detail all the components of the robot so that anything I know about the project will be written down in what is going to be its documentation.

5.2.1 Robot Body

Before jumping straight to crafting the robot I decided it is better to make a 3D model of it first. This approach is the best you can make when building something custom because you can always redesign on-the-spot and it is easily to observe and correct any mistakes.

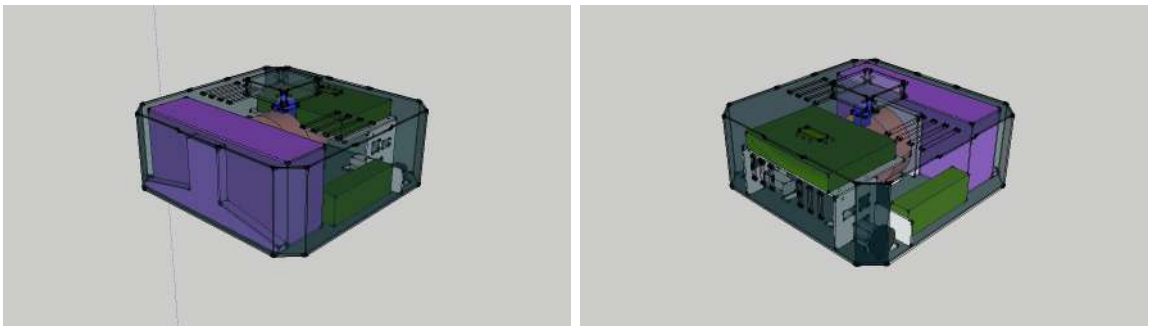


Figure 5.2: 3D model of the robot.

Once the final version of the 3D model was met, I transposed all the body parts of the robot to a 2D drawing. The 2D drawing was then handed over to a CNC machine and the parts were cut. For the body of the robot a 2mm transparent plexiglas was used.

After the parts were finally cut I continued with assembling all the parts together. As I have already had a 3D drawing it was easier for me to glue together the pieces. I have used some paper tape to hold the pieces in place while I was gluing. At this stage of the project the variable speed rotary tool kit really came in handy

Figure 5.3: Frame assembly.

as there were some parts of the body that
needed adjustment.

At a first glance it might seem the frame is not stiff enough for the weight of the inside components. Contradictory, the frame holds just fine and the transparency of the plastic gives the robot a geeky look (as you can see its inside cables and circuits).

5.2.2 Batteries

Figure 5.4: 3S 11.1V LiPo battery. Image taken from [20].

The systems' power supply is provided by two 11.1V 2800mAh LiPo batteries. One of the batteries is entirely dedicated for powering the vacuum pump of the robot. The reason being so is because the motor of the pump consumes approximately 10Ah which, given the current capacity of the battery, can run continuously for about 16 minutes. This time range was considered to be satisfactory even from the early designing phase of the project.

The remaining battery is split between all other components: motors, sensors and microcontrollers. Even so, the capacity of the battery provides way more running time than the 16 minutes it takes for the vacuum pump motor to empty the other one.

5.2.3 Vacuuming Component

The scope of the project was not to build a vacuum cleaner however, if it didn't have this component, I couldn't have named it a vacuum cleaner anymore. I considered building a custom version of a small vacuum cleaner by adapting a 12V car vacuum cleaner and building my own version of vacuuming system. Inspired by real vacuum cleaners, my robot has a 20cm wide opening to the front through which the dust and waste is collected. The system is composed of four parts: the already mentioned opening that narrows to the top, the dust collector bin, the dust filter and the turbine-motor system taken from the car vacuum cleaner. The scope of the filter is that no dust or waste to get inside the turbine.

The system behaves as expected as it was previously designed, prototyped and tested. The robot was successfully tested with picking up rice grains from the surface of a carpet. Below you can see attached a picture of an early prototype that was finally implemented and the motor-turbine component that I have recovered from a working car vacuum cleaner.

Figure 5.5: Opening-dust bin-filter component and turbine-motor system.

5.2.4 Motors and Wheels

The muscles of the robot are two 12V DC motors which have an 1:53 hub gearbox. The motors also came with a magnetic encoder already mounted. The need of the encoder is crucial for real time localization of the robot. The high gear ratio provides enough torque for the robot to move on carpet.

The wheels have a diameter of 42mm and a width of 19mm. The tire of the wheels is made out of rubber and this aspect, together with the deep dents, maximizes the friction force at forward or backward operation. In the front of the robot there are two passive 9.5mm ball casters mounted so that the front of the robot won't sink into the carpet at forward motion.

Figure 5.6: Robot motor (image from [21]), active wheel and passive ball caster (images from [22]).

From the datasheet of the magnetic encoder I have come up with the following table which highlights the colors and usages of the wires that come from each of the motors:

Wire color	Meaning
Black	Motor -
Red	Motor +
Brown	Hall Sensor VCC (5V)
Green	Hall Sensor GND
Blue	Hall Sensor A VOUT
Purple	Hall Sensor B VOUT

Table 5.1: Motor wires meaning

Important note: The datasheet of the magnetic encoder states that a resistor of 1KΩ must be provided between *Hall Sensor VCC* and each of the *Hall Sensor VOUT* in order to receive steady values from the encoder.

5.2.5 Infra-red Sensors and Servo

For implementation of the 2D mapping I have chosen to rely on the output given by three cheap IR proximity sensors mounted on a platform which is rotated to the left and right by a servo motor. The sensors detect reflective objects which are 10 to 80cm away. The outputs of the sensors connect to the ADCs of the Chipkit WF32.

Figure 5.7: IR proximity sensor (image from [21]) and servo motor (image from [23]).

5.2.6 Custom PCB

As there are a lot of components that need to be controller and some of them also require specific circuits there was the need to build an intermediate board. This board makes the connection between most of the hardware components of the robot and the

Wire	**Meaning**
Sensor **B**lack	Sensor GND
Sensor **R**ed	Sensor VCC (5V)
Sensor **Y**ellow	Sensor VOUT
Servo **B**lack	Servo motor GND
Servo **R**ed	Servo motor VCC (5V)
Servo **O**range	Servo motor CMD

Table 5.2: Sensors and servo wires meaning.

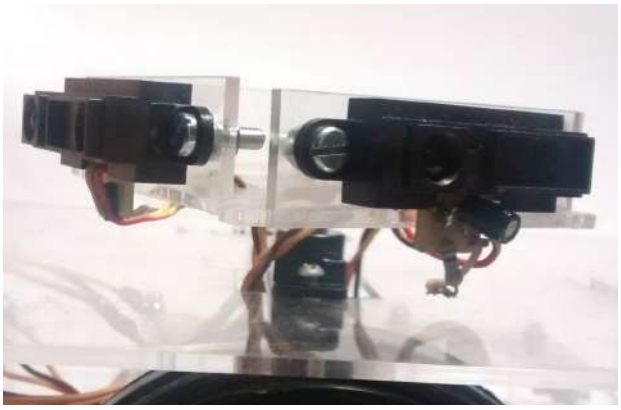


Figure 5.8: Sensors platform on top of the robot.

main controller board (Chipkit WF32). Below you can see a picture of the custom PCB I am detailing about:

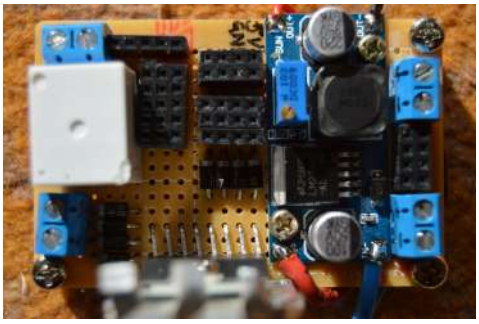
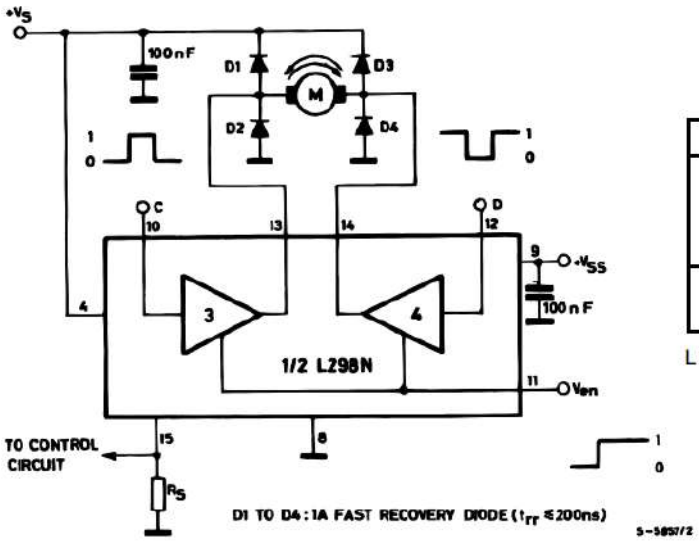


Figure 5.9: Custom PCB top view: vacuum pump relay (left), L298N H-bridge IC (center-bottom) and buck converter (right).

The variable voltage buck converter is used for transforming the 12V taken from the LiPo battery down to 5V which is the needed operating voltage for the Zybo Zynq-7000, Chipkit WF32, sensors, servo motor, relay command and H-bridge logic level input voltage.

L298N integrated circuit was chosen mainly because it is a double H-bridge, thus fitting the two-motor drive in the case of this project. Moreover, it is easy to set up and requires only a few other components for the final working circuit. The circuit I have build on this board that includes the L298N IC is the same as the one described in the datasheet of the product. Below you can see attached a snapshot from the mentioned datasheet:



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure 5.10: L298N IC motor circuit. Picture extracted from [13].

Important note: The signal denoted in figure 5.10 as *TO CONTROL CIRCUIT* is ignored. The signals C and D are tied to the leads of the motor and the V_{en} signal goes to the microcontroller.

For turning the vacuum pump motor on and off I am using a relay. The relay has the nominal voltage on the coil 5V and is capable of driving 12V signals. The exact one I am using is *RM50-P-05* and comes for a small price. The circuit I have implemented using this relay can be seen in the picture below:

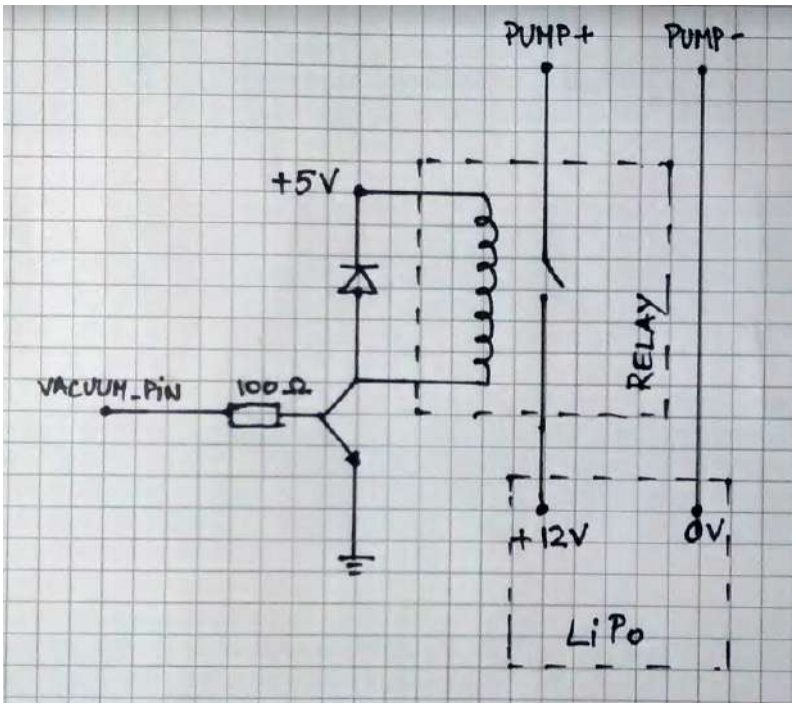


Figure 5.11: Vacuum pump motor control circuit.

Important note: The battery mentioned in figure 5.11 is the LiPo battery dedicated entirely for the vacuum pump motor as already mentioned in subsection 5.2.2. The end denoted as *VACUUM_PIN* goes to the microcontroller and is the signal that toggles the power on the vacuum pump motor.

The rest of the parts present on the custom PCB are just some sockets. The blue ones are used for connecting the power supply from the batteries, connecting the vacuum pump motor or taking 5V as external power supply for the two microcontrollers. The black sockets are used for a better cable management inside the robot: all the peripherals are linked to their own pins on the custom PCB and from some other sockets a single PCI cable with all the signals leaves towards the Chipkit WF32. In order to have a better understanding about the design of the custom PCB I denoted the pins in picture 5.12 and in table 5.3 I have explained the meaning for all of this pins.

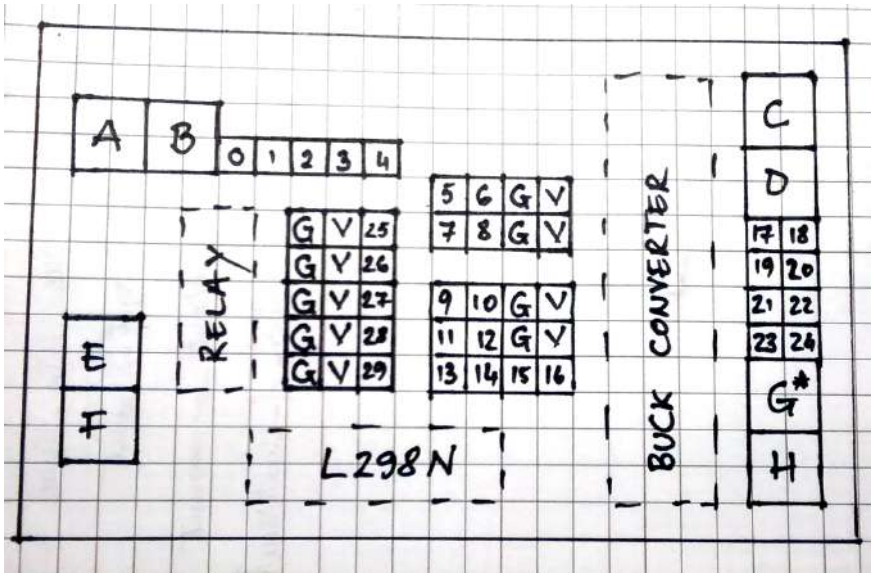


Figure 5.12: Custom PCB pins notation.

Pin notation	**Meaning**	**Chipkit pin no.**
0	(output) Open for 4th IR sensor	18
1	(output) Left IR sensor output	15
2	(output) Front IR sensor output	16
3	(output) Right IR sensor output	17
4	(output) Servo control	14
5	(output) Left encoder digital	28
6	(output) Left encoder interrupt	2
7	(output) Right encoder digital	33
8	(output) Right encoder interrupt	7
9	(input) Right encoder digital	N/A
10	(input) Right encoder interrupt	N/A
11	(input) Left encoder digital	N/A
12	(input) Left encoder interrupt	N/A
13	(input) Right motor -	N/A
14	(input) Right motor +	N/A
15	(input) Left motor +	N/A
16	(input) Left motor -	N/A
17	(output) Vacuum pump control	29
18	(output) Open for any purpose	32
19	(output) Left motor PWM	3
20	(output) Right motor PWM	6
21	(output) Left motor direction A	4
22	(output) Right motor direction A	5
23	(output) Left motor direction B	30
24	(output) Right motor direction B	31
25	(input) Servo control	N/A
26	(input) Left IR sensor output	N/A
27	(input) Front IR sensor output	N/A
28	(input) Right IR sensor output	N/A
29	(input) Open for 4th IR sensor	N/A
A	(output) Vacuum pump motor +	N/A
B	(output) Vacuum pump motor -	N/A
C	(output) +5V	J17 - +5V
D	(output) Common ground	J17 - GND
E	(input) 1st LiPo GND	N/A
F	(input) 1st LiPo VCC	N/A
G*	(input) 2nd LiPo VCC	N/A
H	(input) 2nd LiPo GND	N/A
G	Common ground	N/A
V	+5V	N/A

Table 5.3: Custom PCB pinout and Chipkit WF32 connection.

Important note: Explanation for table 5.3 right column terms: *output* - signal goes from PCB to microcontroller and *input* - signal comes from peripheral to PCB.

5.2.7 Chipkit WF32

As it was already mentioned in section 2.2, this development board is utilized for direct interaction with the peripherals. Most of the pins and usages were already explained in table 5.3. Apart from those there are also the Bluetooth module communication explained in section 5.2.8 and the I^2C communication with Zybo Zynq-7000 which will be explained in this section.

In the following paragraphs I will detail the software modules loaded on this board, their scope and operating principle. The development language is C/C++ and, because C++ makes it possible to interact with objects, the software is mapped one to one to the hardware components.

Peripheral components

Encoder.cpp

This class is responsible for attaching the two digital and interrupt pins of the left and right motor encoders. In addition to this, on the interrupt pin the interrupt handling method increments the *position* of the encoder. The most crucial part of this class is to return the RPM of the motor by taking into account the difference in time between two consecutive interrupts. The formula of computing the just mentioned transformation is explained in detail in the listing A.2 in section A.

Motor.cpp

Motor class has the scope to attach the three pins (direction A, direction B and enable) of each of the left and right motors. In addition to this, it is also capable of sending the right commands to either turn at a specified speed or brake the motor. More information in the code snippet from listing A.3 in section A.

SharpSensor.cpp

As its name suggests, it is responsible for attaching an IR sensor by the specified pin number. In addition, it returns the distance in cm to the sensed obstacle. The formula to transform from raw sensor output to actual cm was found out the following way:

1. Read sensor output at different object distance and write down the values (10 to 40 cm with a 5 cm step);

2. Load experimental data in Matlab;

3. Make Matlab approximate the function that for the given sensor output data returns the length in cm. Matlab *fit* function was used for this step and the input for this function was the experimental data loaded at the previous step;

4. For the range starting from 200 up to 730 (sensor raw data) use the function generated at previous step to compute the output in cm;

5. The results of the previous step were then stored in a look-up table and further used in the method that returns the cm to the sensed object.

In order to get a steady and reliable sensor reading before returning the actual requested value the sensor is read several times and the mean value of the last 20 readings is returned. More information about this part can be found in listing A.4 from section A.

Peripheral controllers

SensorController.cpp

Wrapper class for all the sensors. Calls the attach method on each of the sensors during setup phase. It is also responsible for controlling the motion of the servo motor on top of which the sensors platform is mounted as you could see in figure 5.8 from subsection 5.2.5. The snippet of code which shows how the rotation of the servo is handled in time is shown in listing A.5 in section A.

At each call of the *process* method the position of the servo is updated in the case the free move mode of the servo is enabled. Moreover, sensors readings are requested by the controller in order to keep the array of each of the sensors up to date such as, when actually requested, the returned value is up to date as well.

BluetoothController.cpp

Class responsible for driving the Bluetooth module mentioned in subsection 5.2.8. Implements simple communication with the module through the Serial1 pins located on Chipkit WF32. Baud rate for this communication is 9600 as highlighted on line 15 in listing A.1 from section A.

At each call of the *listen* method the Serial1 is checked for any incoming data. If it is present then the data is parsed and code proceeds accordingly (e.g. send move/turn command, toggle vacuum motor pump, etc.).

WifiController.cpp

Code that handles WiFi connection and communication. The code is based on the same principle used by the example available here (or **search phrase**: chipkit wf32 wifi library).

The class has an internal *state* variable and based on the value of this variable, at every *process* method call the necessary steps are performed (e.g. connect to WiFi/server, send message, wait acknowledge from server, etc.).

MotorController.cpp

This class is the most complex of all and requires patience for proper understanding. As in the case of the past mentioned controllers this one is responsible for initializing the communication and control of the two motors.

A queue variable present at the level of this controller keeps track of requested commands. At each *process* method call the command from the first position in queue gets executed. It requires more calls for such a command to be processed; for example, if a turn left 30 degrees command arrives then, every *process* method call, the robot might only turn 1 degree out of the requested 30 degrees. Thus, it requires 30 calls to perform the whole operation. Once a command is executed it is removed from the queue and the

next one is processed. If the queue gets empty then the robot brakes waiting for new commands to come.

Some other features this class has is the communication with the FuzzyController class. By this means the motor controller sends information gathered from motor encoders to the FuzzyController and waits for data that it then sends as commands to motors. By doing so, the specification which says that the robot must move in a straight line when requested so is then met.

More information about this class can be found in the source code which is referenced in [10].

FuzzyController.cpp

Here is implemented the fuzzy controller used by the MotorController to drive the robot in a straight line. I won't insist on the code itself as it is not the case. However, I would like to point out the fuzzy sets I am currently using and which, in my case, provide close to perfection results. I have come up with the values present in figures 5.13 and 5.14 by doing a simple trick: write 50 (zero value on the fuzzy set from figure 5.13) on the PWM outputs of the motors and record the RPM at which the motors were turning. That value is 60 (zero value on the fuzzy set from figure 5.14). Finally, by playing around with intermediate values I came up with the next sets:

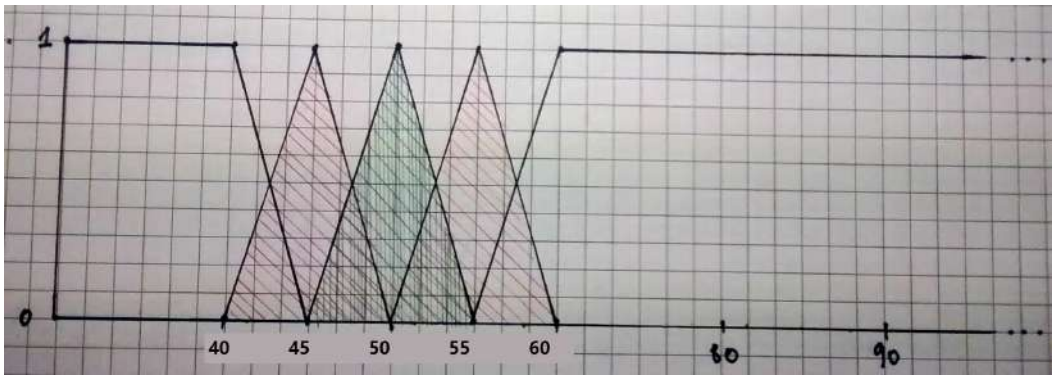


Figure 5.13: PWM fuzzy sets.

The centers of the green triangles of both figures 5.13 and 5.14 represent the values the fuzzy controller is trying to get the system to when the robot is moving. For a smooth transition between values more sets were considered close to the target one. The small range of the sets assures that the response of the system is sharp and steady.

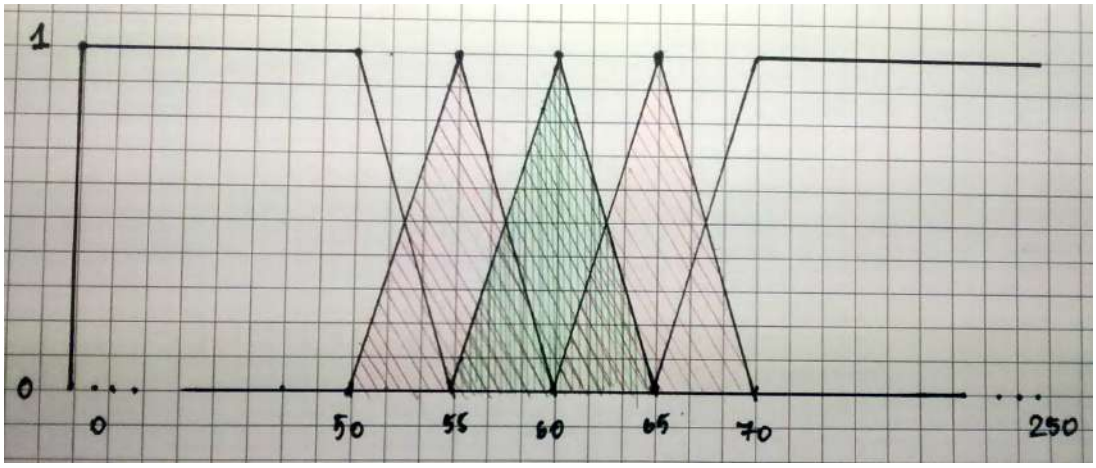


Figure 5.14: RPM fuzzy sets.

Multithreading simulation on microcontroller

In order for the robot to be responsive to external events there is a high demand for skipping long processor time consuming sequential tasks inside the continuous loop of the software. Tasks such as connecting to WiFi are only done once in the beginning of the program. In order to simulate multithreading, thus allowing the robot to be responsive to outside events, the processor time is split between the controllers in a sort of round-robin manner. However, the time slice is not fixed, but let the processor be taken on trust by each of the controller. The instructions that need to be executed during the time each of the controller get to have the processor are just a few and guaranteed to take no more than a few microseconds. All other interactions like I^2C communication is done either on external or timer interrupts. A simple flow diagram of the code is provided in the following picture:

Important note: At any moment in time while executing the continuous loop the flow could be interrupted by an external event but not for a long period of time. However, after the event is finished the execution is resumed. This happens a large number of times each loop but it happens that fast that it isn't noticeable.

Due to performance considerations, as it was already mentioned about listing A.4, I have taken into account to build some lookup tables for *sin* and *cos* mathematical functions. I decided doing so because these functions are called quite often at they would have definitely represented a drawback in performance of the microcontroller.

I^2C communication with Zybo

From the point of view of Chipkit WF32, the I^2C communication with Zybo is done by using the available *Wire* library. The board is set in slave mode with address *8* and the callback methods for sending and receiving data during the I^2C setup phase. On receiving a command from the master board it is first parsed and then executed. When requested from master, the microcontroller gathers requested information from peripherals, packages it in a message and sends it right away. More information about the above mentioned

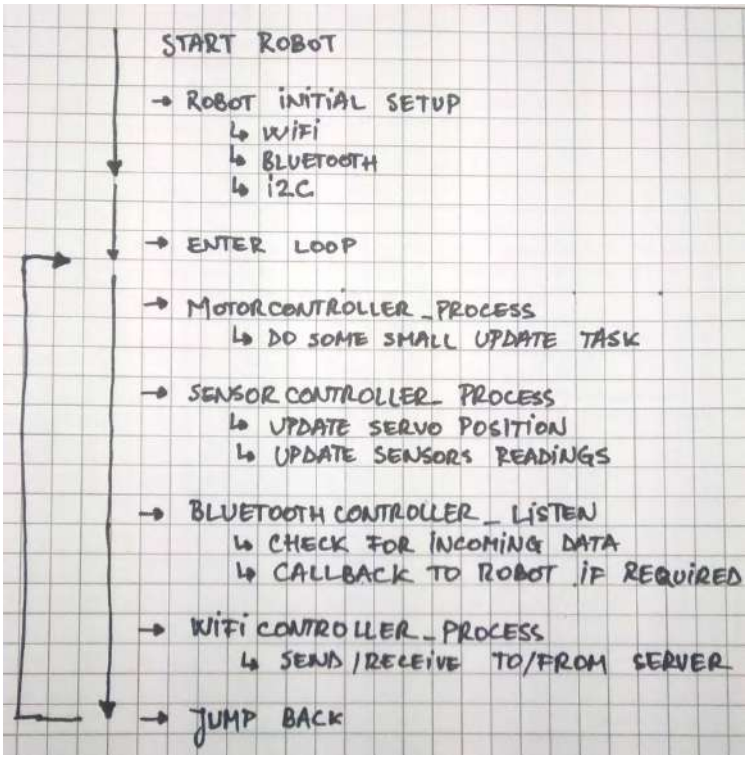


Figure 5.15: Chipkit WF32 code flow.

processes can be found inside the code snippets from listing A.1 in section A. The pins used for I^2C communication on Chipkit WF32 board are the ones that can be found on J6 as mentioned in the datasheet of the board.

5.2.8 Bluetooth Module

HC - 05 Bluetooth Serial Module is perfect for hobby projects. It is really easy to interface with any microcontroller that supports UART protocol. Input voltage can range between 3.6V to 6V thus perfect for the 5V output from the Chipkit WF32. The Bluetooth module is connected to the Chipkit WF32 board as shown in the following table:

Bluetooth module pin	**Chipkit WF32 pin no.**
RX	40
TX	39
GND	GND
VCC	+5V

Table 5.4: Bluetooth module - Chipkit WF32 connection.

5.2.9 WiFi Module

Fortunately, the Chipkit WF32 comes with a WiFi module soldered on board. The main reason this board was chosen for development was because it could provide a WiFi connection and communication without any external cables required; just uploading the right code did all the setup and communication.

5.2.10 Zybo Zynq-7000

I was a newcomer to the Xilinx tools used for hardware design and, given so, I decided to keep the design to minimum so I could get something functional in the limited amount of time I had allocated for this project. The block design presented below clearly shows that I have only considered making external both I^2C and SPI interfaces and some general purpose I/O pins. All of the pins required by the mentioned interfaces are linked to the external Pmods available on Zybo development board.

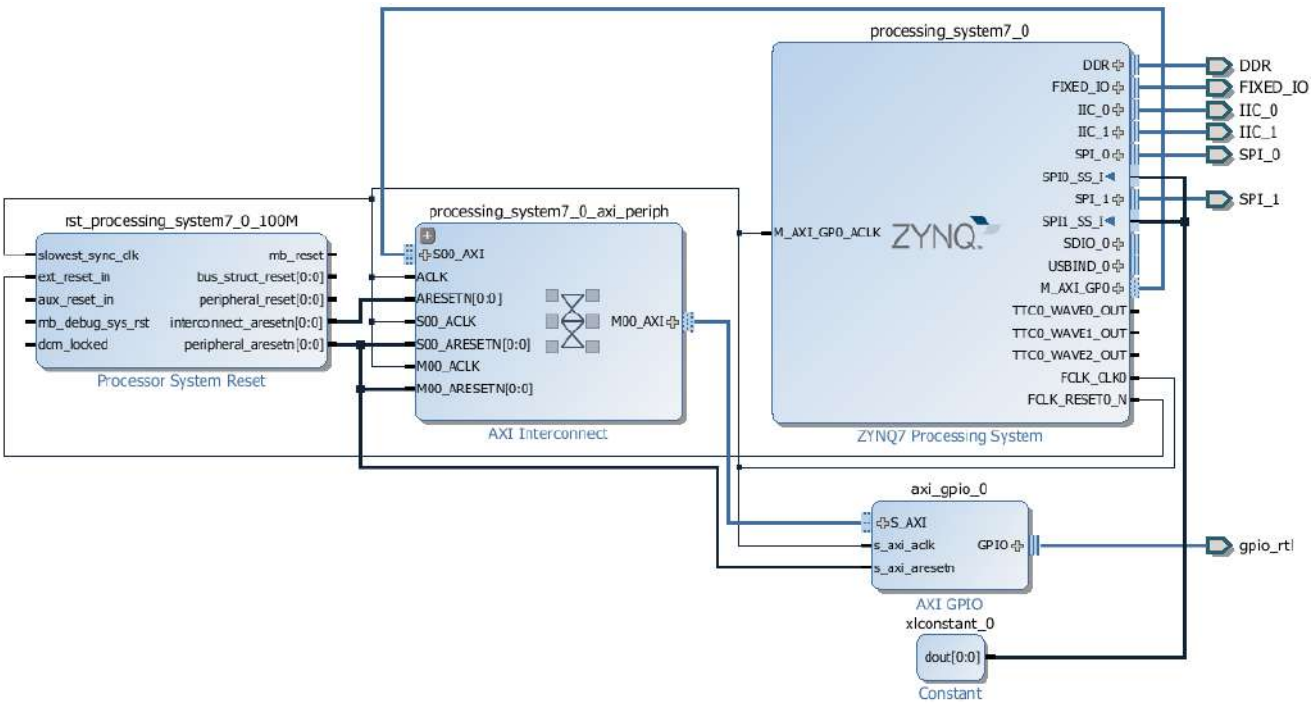


Figure 5.16: Zybo Zynq-7000 block design in Vivado.

Based on the constraints file generated in the Vivado project I have created table **??**. This table highlights the pin number in correspondence to its purpose and way to access when used in Xilinx SDK.

Pin location	**Purpose**	**SDK**
JE1	SPI0 SS0	Accessed through PS
JE2	SPI0 MISO	Accessed through PS
JE3	SPI0 MOSI	Accessed through PS
JE4	SPI0 SCK	Accessed through PS
JE7	GPIO 0	Pin 0
JE8	GPIO 1	Pin 1
JE9	GPIO 2	Pin 2
JE10	GPIO 3	Pin 3
JD1	I^2C0 SCL	Accessed through PS
JD2	I^2C0 SDA	Accessed through PS
JD3	I^2C1 SCL 3	Accessed through PS
JD4	I^2C0 SDA 3	Accessed through PS
JD7	SPI1 SS0	Accessed through PS
JD8	SPI1 MISO	Accessed through PS
JD9	SPI1 MOSI	Accessed through PS
JD10	SPI1 SCK	Accessed through PS
JC1	GPIO 4	Pin 4
JC2	GPIO 5	Pin 5
JC3	GPIO 6	Pin 6
JC4	GPIO 7	Pin 7
JC7	GPIO 8	Pin 8
JC8	GPIO 9	Pin 9
JC9	GPIO 10	Pin 10
JC10	GPIO 11	Pin 11
JB1	SPI0 SS1	Accessed through PS
JB2	SPI0 SS2	Accessed through PS
JB3	SPI0 SST	Accessed through PS
JB4	UNUSED	-
JB7	SPI1 SS1	Accessed through PS
JB8	SPI1 SS2	Accessed through PS
JB9	SPI1 SST	Accessed through PS
JB10	UNUSED	-
R18	BTN0	Pin 12
P16	BTN1	Pin 13
V16	BTN2	Pin 14
Y16	BTN3	Pin 15
G15	SW0	Pin 16
P15	SW1	Pin 17
W13	SW2	Pin 18
T16	SW3	Pin 19
M14	LED0	Pin 20
M15	LED1	Pin 21
G14	LED2	Pin 22
D18	LED3	Pin 23

Table 5.5: Zybo Zynq-7000 external pins and their correspondent in code.

The application which runs on the Zybo Zynq-7000 is written in C using the Xilinx SDK and burnt on the flash of the controller. The code is split in several modules as follows: main controller, I^2C communication and map controller.

I^2C communication controller

Based on the examples provided in the Xilinx SDK I have modeled the code to suit my own needs. Check code snippets in listing A.7 from section A for more information.

Map controller

Once the data received via I^2C from Chipkit WF32 is parsed it is time to update the virtual map held in memory by Zybo. The structure containing information about the current state of the robot is sent straight to the map controller which is responsible of updating the map. Several 2D point transformations are performed during this step as the data comes in raw format. For the position and the reading of a sensor is sent from the robot and it is the job of the map controller to further process it in order to obtain the final point on the map which corresponds to the position of the obstacle sensed by the sensor.

Initially the map is represented as a grid of free cells and the robot positioned in the center of the map. As the robot moves on the surface in reality the grid map is also updated. If a sensor detects some obstacle at any moment in time, then the map is updated accordingly. For each computed point in which the sensors sensed an obstacle the corresponding grid map block is returned. The block is then updated by incrementing the number of points that have fallen in the region of that specific block. If the number of points that have fallen in a block pass above a threshold then the block is marked as obstacle.

More details about how the above mentioned flow is implemented can be found by looking into the source code of the robot referenced in [10].

Main controller

This part of code is responsible for calling the initializing methods of the other methods and then, by jumping into a continuous loop, to manage the proper interaction between modules.

Autonomous behavior

Integrated in the same module with the map controller comes another piece of code which is responsible for sending the right commands to the robot such as it completes the mapping of the environment by itself and then does the vacuum cleaning task.

Initially, the virtual map of the robot is created as empty and the robot is placed in the middle of the map. For the purpose of demonstration, the current map is limited to a square of a defined length. The map is represented internally as a matrix. The first step of the autonomous algorithm is to set the intermediate points that will make the robot cover all the surface of the map by touching all the points. The just mentioned step is better explained by figure 5.17.

The path between the current robots' position and the position of the intermediate point is computed using the A^* algorithm that has Manhattan distance between points as a metric. The robot can only move horizontally or vertically as there are no virtual

diagonal edges so that A* will consider. The most powerful fact about A* is that it is adaptable to environment changes. The robot will perform in small steps as follows:

1. Look around for 3 seconds
2. Compute path from current position to target intermediate point taking into account previous and current observations
3. Perform action (move forward 5 cm, turn left or turn right)
4. If still got intermediate points jump to step 1. Finish otherwise.

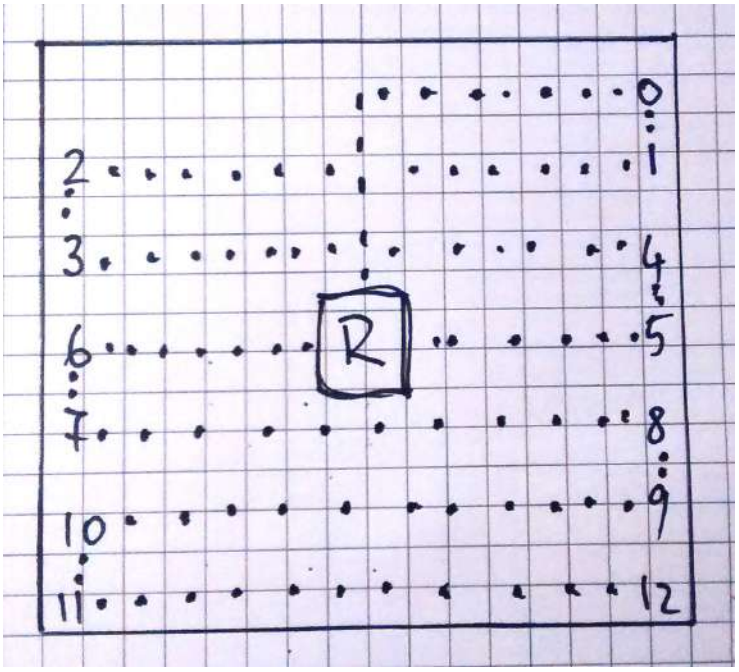


Figure 5.17: Algorithm behind autonomous behavior.

5.2.11 Android Application

One of the features the robot has is the ability to be controlled remotely via Bluetooth from an Android device through a customized application. As the project was complex enough already I chose not to write an Android application from scratch but to use a third party application which could do the same thing for me. Thus, the development of the Android application only last for one afternoon. For building the application I have used this tool (or **search phrase**: mit app inventor 2). You can find more information about building Android application directly on their website.

I wanted a simple and easy to use application which features some buttons and when pressed, sends a *string* corresponding to the pressed button to the Bluetooth module mentioned at subsection 5.2.8. The interface of the application is presented in picture 5.18 and controls are explained in detail in table 8.1.

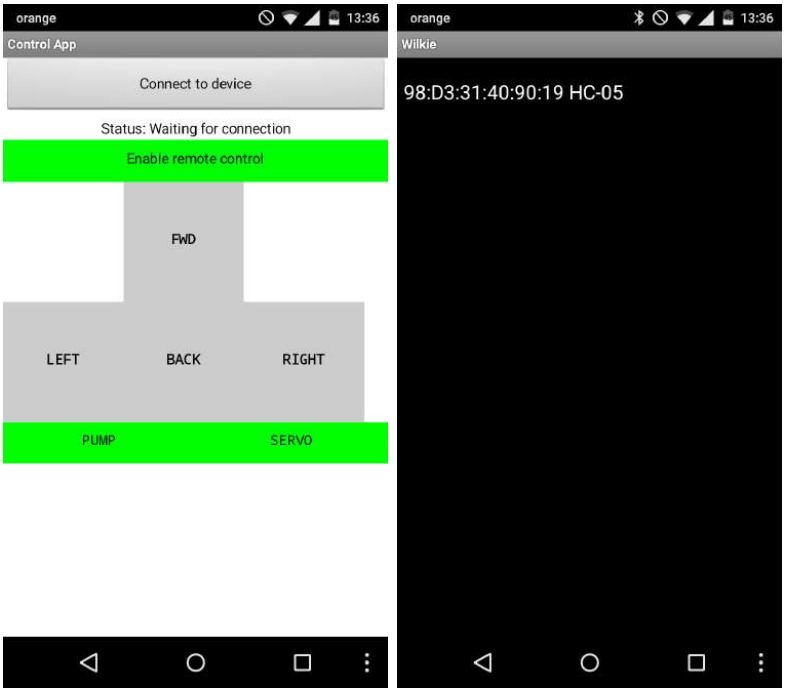


Figure 5.18: Android application interface and list of Bluetooth devices.

Button label	**Command**
Connect to device	Opens the list of available Bluetooth devices
Enable remote control	Sends $0\backslash n$ via Bluetooth
FWD	Sends $1\backslash n$ via Bluetooth
LEFT	Sends $3\backslash n$ via Bluetooth
BACK	Sends $2\backslash n$ via Bluetooth
RIGHT	Sends $4\backslash n$ via Bluetooth
PUMP	Sends $5\backslash n$ via Bluetooth
SERVO	Sends $6\backslash n$ via Bluetooth

Table 5.6: Android applications button presses commands.

Application operation:

1. Power on robot
2. Turn Bluetooth on from mobile device
3. Enter Android application
4. Press ”Connect to device”
5. Select robot bluetooth modules’ MAC address from the list
6. Press ”Enable remote control” for gaining remote control over robot
7. Have fun!

Chapter 6

Testing and Validation

Considering the fact that the proposed solution for the diploma project nearly represents an end-user consumer product thorough testing needed to be performed during the development phase of the project.

It is worth mentioning that most of the code is written in C/C++ on a microcontroller using an environment that did not provide a debugger. Taking this into account it was quite hard to find out the issues that arose most of the bugs encountered through the development process of the project. However, at the time of completion of the project there were no bugs discovered that might block the normal functioning of the final product.

Most of the code was written using *Arduino IDE* using C/C++ as programming language. The software written on the Zybo board was developed under the SDK provided by Xilinx. The server used for WiFi communication between the robot and the PC was written in Java using the *Processing IDE*.

The majority of tests were performed by having a wired connection between the PC and the microcontroller and following the print information transmitted by the board to the development environment on PC. On the other side, if it was the case to test wireless communication, then through WiFi, information was sent to the webserver hosted on PC.

6.1 Wired Testing

In order to test functionalities that required a wired connection the next setup needed to be done:

1. Make sure the batteries voltage level is enough for normal operation of the robot
2. Make sure that all the connections on the robot are in place and there are no loose ends
3. Connect the microcontroller board to PC through USB cable
4. Check if the robot is powered on

5. Open the Serial Monitor on the host PC and follow the on screen information

Once the setup is done, in an iterative manner throughout the development process, several tests were performed. This tests include: proper commands sent and performed by motors, encoder readings are accurate and consistent, proximity sensors values are accurate and consistent, servo motor is performing as commanded, WiFi connection is working, remote control through Bluetooth is working and vacuum pump is operating as expected. The above mentioned tests represent just the main ones as dictated by the initial set objectives of the project.

6.2 Wireless Testing

The steps to be performed in the case of wireless testing differ from the ones mentioned in the previous section just by the absence of the wired connection and the certainty that the WiFi connection between the robot and PC is working properly.

For the proposed solution, in the case of wireless testing, the following tests were performed:

- Make sure the system will operate properly even after several minutes
- Check if the robot can position itself in space properly
- Check whether the virtual map generated by the robot corresponds to the real environment

6.3 Validation

As the project developed in a more complex one than expected in the beginning, the set of validation constraints was reduced as well. Given so, the next validation aspects were considered for the proposed solution:

- Ability of the robot to detect and correctly mark obstacles
- Ability of Bluetooth remote control through a mobile application
- Remote feedback from robot to a PC hosted application through WiFi
- Ability of the robot to autonomously move from point A to B and avoid obstacles

During the testing process held inside the developing environment all the aspects mentioned above were passed. However, it might be the case that because surface properties, lightning conditions or other environment aspects that the product will not behave the same as in the controlled environment.

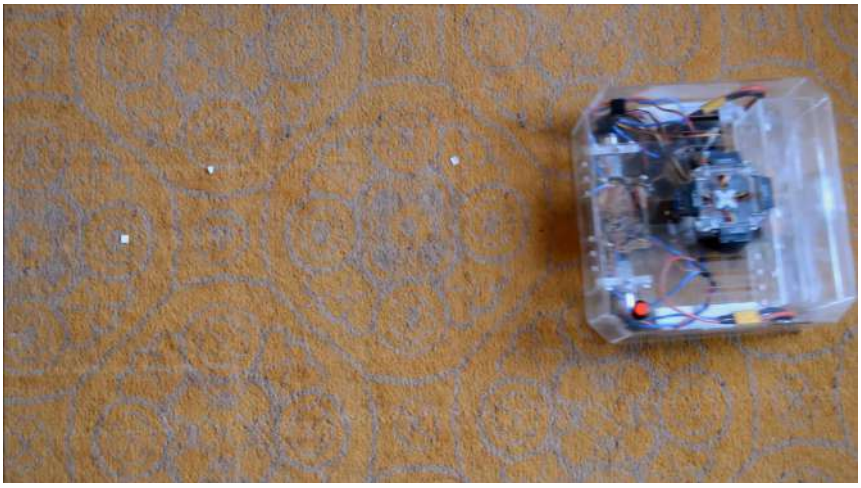
In the following subsections the obtained results of the aspects presented above are detailed.

6.3.1 Vacuuming system in action

The definition of done for the vacuuming component of the product was set at an early phase of development as already mentioned in chapter 2. Some early validation and testing was performed during the early designing phase of the project as it was already mentioned in subsection 4.4.6 of section 4.4.

As part of presenting the project in cause at a local hardware design competition I have edited a presentation video of the product which can be seen following the link referenced by [11]. From the above mentioned video I have extracted some results which I am going to present in the following paragraphs.

(a) Vacuum system test **before** state.



(b) Vacuum system test **after** state.

Figure 6.1a presents the state of the carpet before the robot passed that area. The white text ”written” on the carpet was done using paper bits. As you can see in picture 6.1b, the robot has successfully cleaned the carpet except three pieces of paper. It is not

visible in the above presented pictures but the dust bin collector of the robot also presented traces of dust which furthermore strengthens the quality of the vacuuming system.

6.3.2 Bluetooth remote control

The tutorial about HC-05 bluetooth module referenced in **??** specifies that the operating range is 10m. This is taken into account as long as the robot is operated in open space. However, I have tested the robot indoor. For a room with dimensions of 4 by 5 meters I had no problems operating the robot via bluetooth mobile application. I was performing the control actions from the same room where the robot was located in. On the other side I have came to the conclusion that if there is a wall obscuring the direct interaction between the mobile device and robot then the bluetooth connection gets interrupted if the distance between the mobile and robot gets more than 5m.

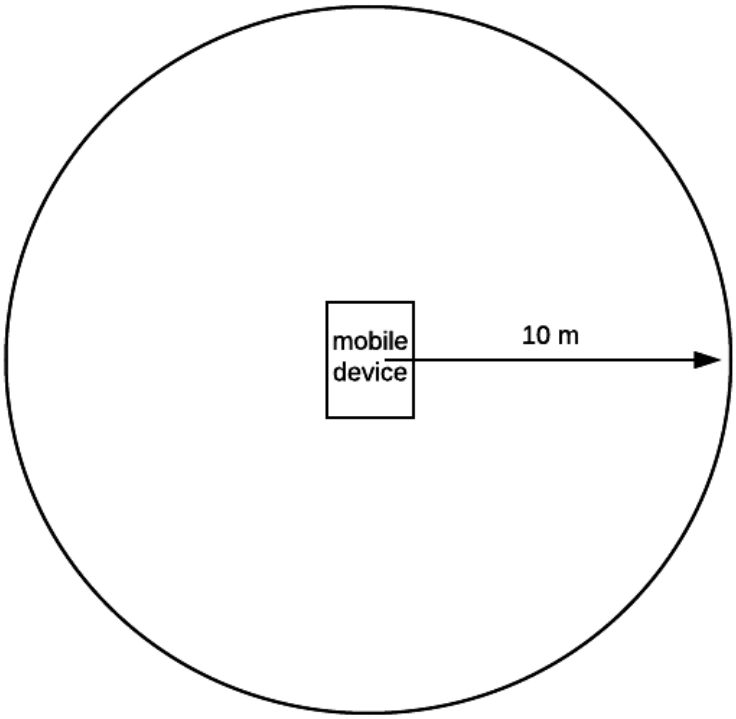
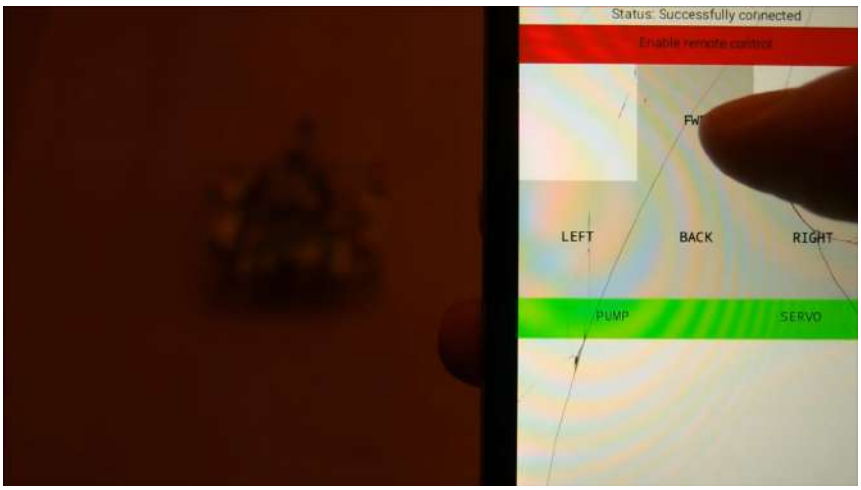


Figure 6.2: 10m radius of Bluetooth connectivity in open air.

Another important aspect that needed thorough testing was the proper execution of commands sent from the mobile device. As presented in subsection 5.2.11 from section 5.1 there are several commands that can be sent from the Android application. Specifically, the user can control the motion of the robot (move forward/backward and turn left/right)

and other aspect such as toggling the vacuum pump or the free move of the on-top of the robot servo.

Out of the commands the robot is able to perform I have chosen to illustrate the testing of the forward command executed by the robot. Picture 6.3a presents the initial state of the robot in the background and the state of the application before pressing the *FWD* button. Once the button was pressed it changed the color from gray to green indicating that the command was sent to the robot. In real time the robot received the *move forward* command and after approximately 1s we can see it changed its initial position. Specifically, image 6.3b displays the new position of the robot and, as we can see, the robot moved forward from its initial position presented in figure 6.3a.



(a) Before pressing the move forward button.

(b) After pressing the move forward button.

The other features provided by the mobile application were tested in the same manner and the results can be seen in the video referenced by [11].

6.3.3 Obstacle detection and occupancy grid map

In order to validate the feature of the product which states that it should be able to *sense* its surrounding environment I have built a corner out of boxes and examined the feedback given by the robot.

The initial state of the robot is presented in figure 6.4. In the top left corner of the image there is the occupancy grid sent as feedback by the robot and logged on the *Processing server* from the host PC. The red arrows from figure 6.4 indicate the forward orientation of the robot for both the real case scenario and virtual occupation grid map.

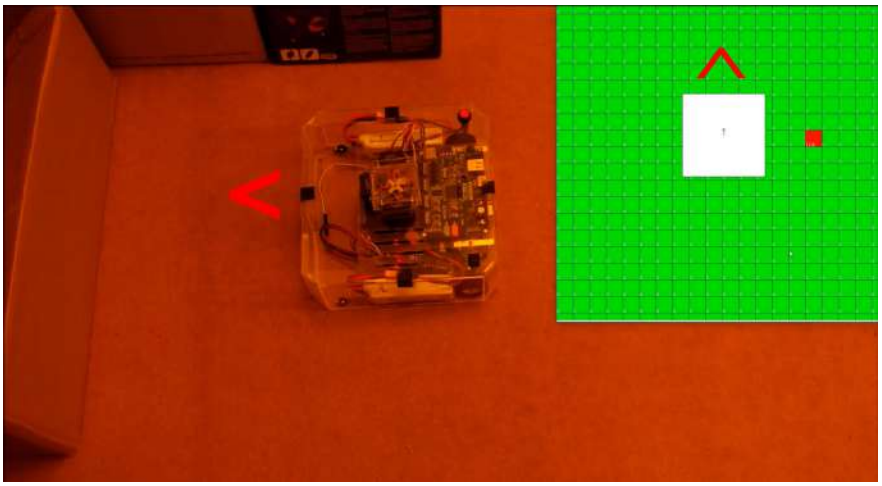


Figure 6.4: Corner testing **initial** robot position and occupancy grid based on sensors feedback

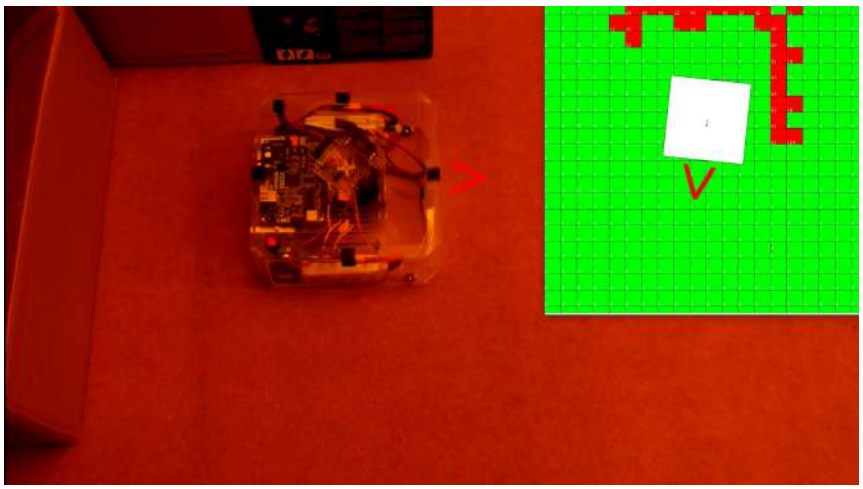


Figure 6.5: Corner testing **final** robot position and occupancy grid based on sensors feedback

The actions the robot has performed during this test can be seen in the video referenced by [11]. The test took around 30s and consisted of the following steps:

1. The free move of the on-top servo was powered on
2. The *FWD* button was pressed twice with a wait interval of 5s in between consecutive and after the final button press
3. The *LEFT* button was pressed twice with a wait interval of 5s in between consecutive and after the final button press
4. The *FWD* button was pressed once

The final position of the robot after performing the steps described above is presented in figure 6.5. We can see the robot has successfully mapped the corner with remarkable accuracy. The red squares present on the occupancy grid map located in top-left corner of image 6.5 indicate that the robot identified a possible obstacle in those positions relative to its personal position and direction.

Chapter 7

User's Manual

The following sections will describe in detail the steps a user has to perform in order to obtain normal operation of the product. Moreover, the operation of all the features the product provides are described thoroughly for the best user experience possible.

7.1 Setup

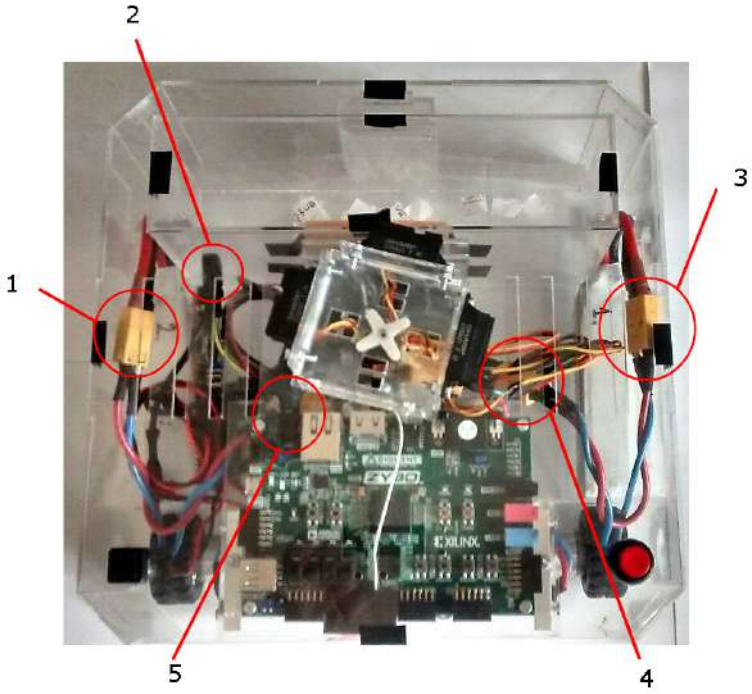


Figure 7.1: Robot top-view connections.

In order for the robot to operate properly the initial setup must be done right. In the next few lines I will make reference to the labels on picture 7.1 to explain the initial connections that must securely be done.

The robot features two LiPo batteries, one on each side of the robot. The battery on the left is used by the logic of the robot while the one on the right is used by the vacuum pump. The connections must be done as follows:

1. Make the connection labeled with 2 in figure 7.1. This is the power supply for the Chipkit WF32 board.
2. Connect the power supply of the Zynq board labeled with 5 in figure 7.1
3. Connect the plug of the pump in the socket labeled with 4 in picture 7.1
4. Perform the connection between the left battery and the socket labeled with 1 in figure 7.1
5. Finally connect the battery on the right with the socket labeled with 3 in picture 7.1

Once the above enumerated steps are performed the initial setup of the robot concerning the electronics part of the product is done.

7.2 Remote Control

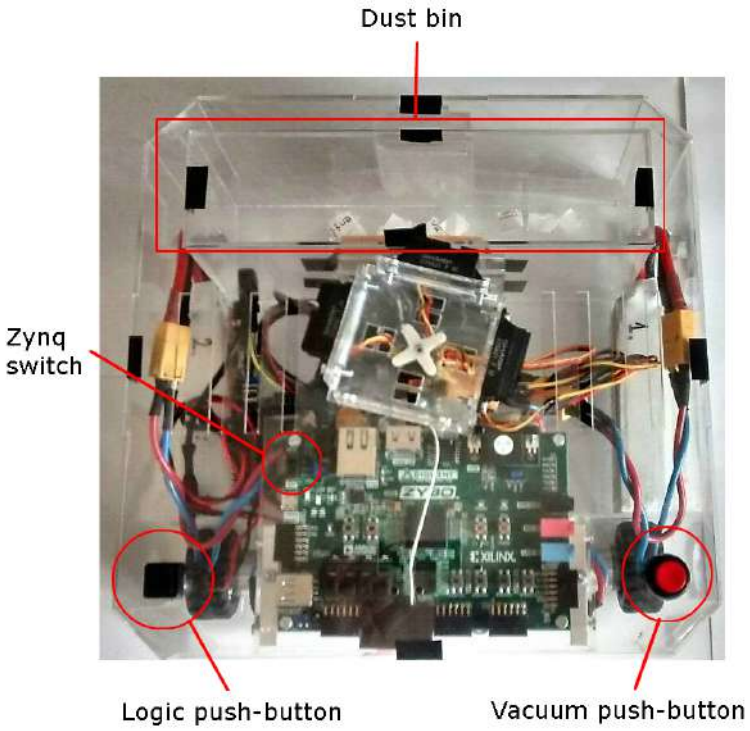


Figure 7.2: Robot top-view power buttons and dust bin.

Important note: Before starting operation always make sure that the dust bin is properly in place. The dust bin is located in the front of the vacuum cleaner as you can see it labeled in picture 7.2.

In order to power on the robot so that remote control is enabled the **Black logic push-button** must be pressed once. The location of the button can be identified as seen in figure 7.2.

To enable the vacuum pump the **Red vacuum push-button** must be pressed once. The location of the button can be identified as seen in figure 7.2.

Remote control of the robot requires a mobile phone running Android capable of Bluetooth communication. The application that need to be installed can be found on the CD provided with the current document as *Wilkie.apk* or can be downloaded from the link of the source code provided in [10].

Once the application is installed on the device that will act as a remote control for the robot the screen present in figure 7.3 will be displayed as the user enters the application.

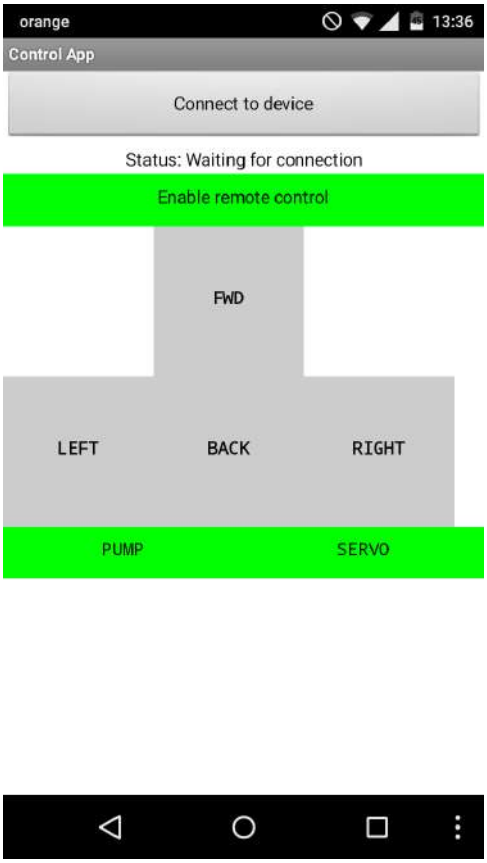


Figure 7.3: Application start screen.

The first step that needs to be performed in order to enable Bluetooth communication between mobile phone and robot is to press the button labeled with *"Connect to device"* as you can see in picture 7.3. Next, a list of available devices will be displayed as you can see in the picture 7.4.

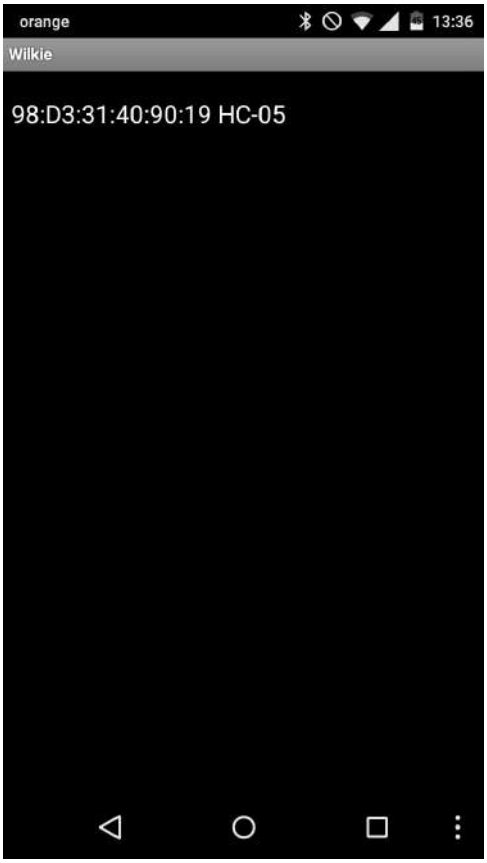


Figure 7.4: List of Bluetooth devices.

Usually there is only one device listed if there aren't any devices with Bluetooth turned on in range of the mobile phone. Identify the MAC address of the vacuum cleaner usually by looking to the item containing *"HC-05"* text.

Once the connection is established we can control the vacuum cleaner through the application interface. Table **??** presents the role of the buttons the app provides.

		Button background color	
		Red	Green
Button label	Enable remote control	Remote control is enabled	Remote control is disabled
	FWD	N/A	Robot moves forward 10 cm
	BWD	N/A	Robot moves backward 10 cm
	LEFT	N/A	Robot turns left 90°
	RIGHT	N/A	Robot turns right 90°
	PUMP	Vacuum pump turned on	Vacuum pump turned off
	SERVO	Top servo turned on	Top servo turned off

Table 7.1: Mobile application buttons meaning.

7.3 Autonomous Behavior

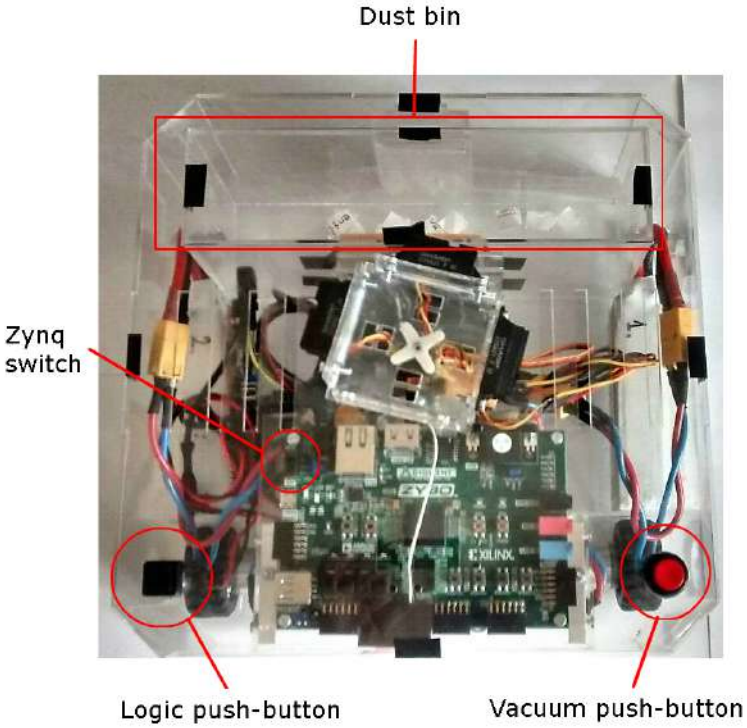


Figure 7.5: Robot top-view power buttons and dust bin.

For autonomous behavior of the robot the user must first follow the same steps as for previous section 7.2 excepting the steps for application configuration. Moreover, the user must assure that the Zynq switch labeled in figure 7.5 accordingly is set in the on position.

7.4 Monitoring

In order to monitor the robot remotely the user must install the Processing IDE which can be downloaded from their official website. The code of the Processing software can be found in the resources provided with this document.

Once the project is opened the user sees the screen presented in figure 7.6.

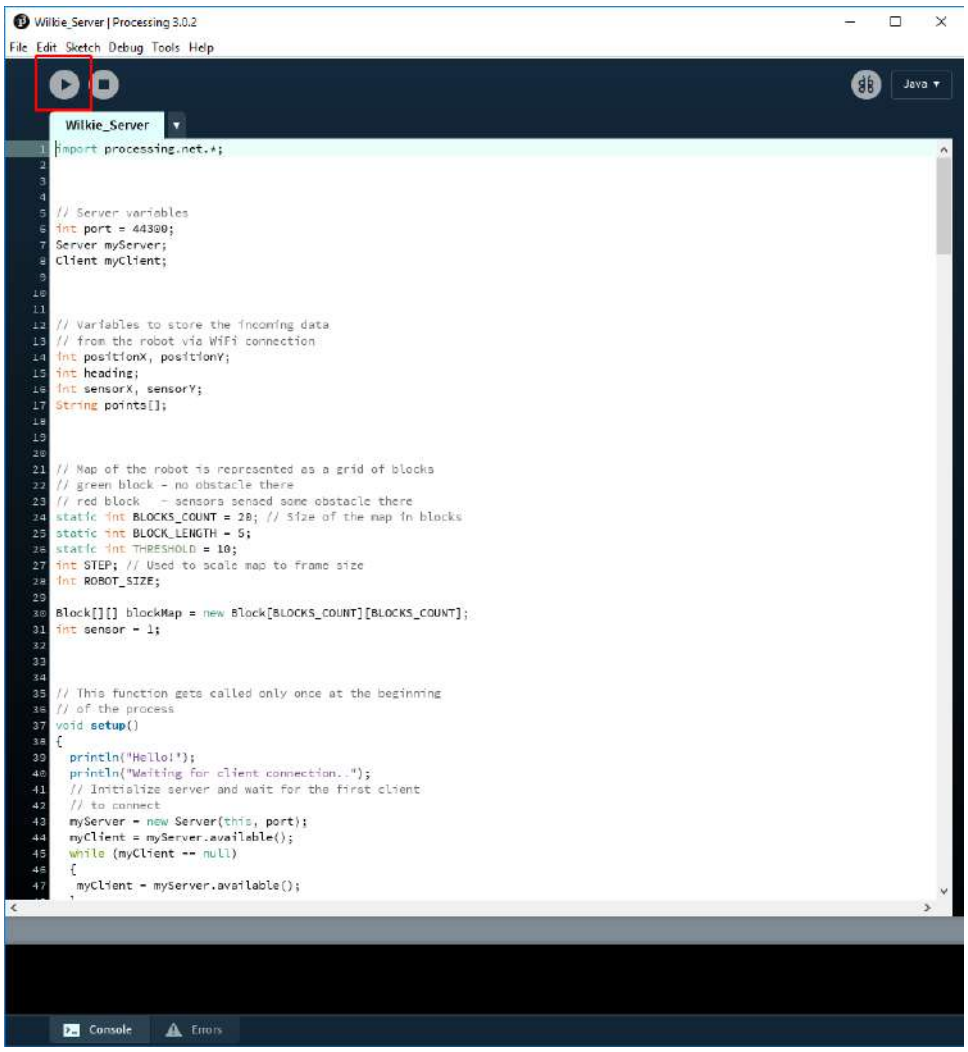


Figure 7.6: Processing IDE initial screen.

From the initial screen of the application the user has to press the start button highlighted with a red square in figure 7.6.

Figure 7.7: Processing server initial screen.

Once the application starts a gray screen will be presented as you can see in figure 7.7. The application now waits for the robot to connect. The user can proceed with powering on the robot as it was already explained in previous sections from current chapter.

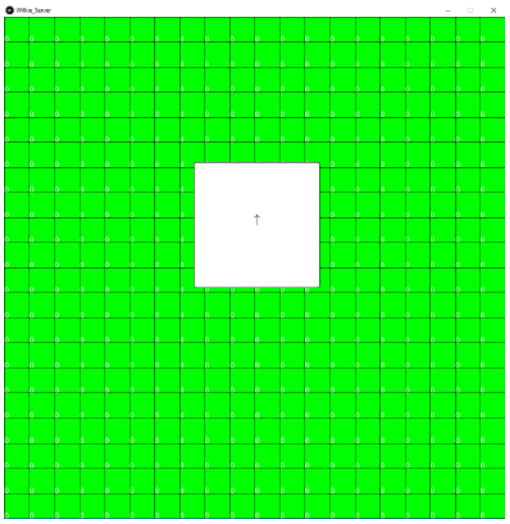


Figure 7.8: Processing server displaying robots' feedback.

Figure 7.8 presents a snapshot of the feedback sent by the robot which is displayed by the Processing server.

Chapter 8

Conclusions

The proposed solution for the diploma project definitely boosted my knowledge in the domain of electronics, software development and problem solving as non of the projects I have performed before did. On one side I am a bit concerned about the fact that I could have invested more time in research so that I could have build a better products. On the other side, the time available for development was quite short so I had to find the right balance between research and development such that in the end I could present a working prototype.

Overall status of the project got quite close to the predicted one in the early phase of design and implementation of the project. Thorough testing of the project from both hardware and software points of view lead to a robust solution in the end of development. Moreover, major bugs were discovered in time because of that and thus, the scheduled work did not suffer in any non-recoverable way.

8.1 Achievements

The end of implementation and testing phase came with a series of achievements I have managed to attain. From this it is worth mentioning the fact that I have managed to build a working prototype of an autonomous vacuum cleaner designing and implementing it all from scratch.

I have managed to craft the mechanics and put together all the electronic components such a device requires. From the same sphere of hardware components, as already presented in section 4.4, I have managed to build a functional vacuuming system using the motor and pump of a 12V car vacuum cleaner together with some other custom built parts.

Above all, I have also managed to write the required software such that the user can control remotely the robot via a mobile application. Moreover, the robot has the ability to sent real time data gathered from its hardware components back to a server hosted on a PC. In addition, the final solution also presents the feature of autonomous behavior, which

makes the robot to be able to detect, avoid obstacles and re-plan its path on the go.

8.2 Engineering Resources Used

The concerned project is a complex one and requires more engineering skills. Knowledge about electronic circuits, software programming, artificial intelligence, computer graphics, operating systems, systems control and signals processing is required in order to understand and reproduce the documented project. Roughly three months were spent on designing and implementing the project and there are still parts that will be considered for further improvements. I cannot state at the moment the amount of hours I have spent on this project but I can assure you that the last two months were intense, spending even 10 hours a day on implementation.

8.3 Problems Encountered

The problems that are worth mentioning and that I have encountered along the development phase of the project together with their solution are mentioned in the table below:

Problem	**Solution**
The initial motors had a gearbox with a 1:19 ratio. The robot was unable to move at low speed on any surface.	Replace the motors with the 1:53 gear ratio ones.
The old wheels would slip under the full weight of the robot when moving on carpet.	Replace the wheels with bigger ones. The new wheels also provide some additional features which were already mentioned in the subsection 5.2.4
Couldn't get consistent readings on the interrupt pins of the encoders	Followed the instructions present in the datasheet of the encoders as already mentioned in subsection 5.2.4
The robot would behave normally for about one minute but then suddenly freezes and no code is executed anymore	Had a bug that would keep allocating memory for a buffer until the memory was full an thus causing the serious failure. Be cautious when working with memory!
The visualizing application would lose connection with the robot.	The server requests were too fast thus causing the server to fail. Added a delay between consecutive requests.

Table 8.1: Problems encountered and their solutions.

8.4 Marketability

In my opinion the current project has a high potential for becoming a viable product. If not in entirety at least some components could be used for development of further ideas. Moreover, the fact that the products that are available on the market already are quite expensive represent a gap that might be filled with a cheaper and versatile product as the current prototype tries to mimic.

8.5 Community Feedback

The only feedback I have received from an outside person was from a user on www.instructables.com who stated that it might be difficult to implement simultaneous localization and mapping using the cheap sensors and that I should consider using a lidar. However, one of the main targets of the project was to implement exactly this feature so it did not influence the course of the project in any way.

8.6 Further Considerations

Of course that there are a lot of things I would like to consider if it was to continue developing the current project. It is only the case you either have a really easy project or a well organized team that you can declare a project to be finished. Taking the above mentioned into account I would like to present the list of features I would like to further implement as follows:

- On-board LCD display and buttons for user interaction
- Web interface for customers so that they can control and monitor the vacuum cleaner
- Add the voltage level indicator so that the vacuum cleaner will have the ability to recharge itself
- Improve the SLAM algorithm by using a more complex solution
- Replace the cheap Sharp proximity sensors with a decent LIDAR
- Improve the mechanics of the robot such that it behaves well on normal carpet as well

Bibliography

[1] E. Olson, *A Primer on Odometry and Motor Control*, January, 2009.

[2] S. Riisgaard, M.R. Blas, *SLAM for Dummies - A Tutorial Approach to Simultaneous Localization and Mapping*.

[3] T.M. Aycock, *A Simultaneous Localization And Mapping Implementation using Inexpensive Hardware*, Tuscaloosa, Alabama, 2010.

[4] M.G. Simones, *Introduction to Fuzzy Control*, Colorado School of Mines, USA.

[5] L.A. Zadeh, *Fuzzy Sets*, Information and Control, 8, 338-353, 1965.

[6] M. Buckland, *Programming Game AI by Example*, Texas, USA, Wordware Publishing, Inc., 2005.

[7] D. Ferguson, M. Likhacev, A. Stentz *A Guide to Heuristic-based Path Planning*, Carnegie Mellon University, Pittsburgh, PA, USA.

[8] G. Monari, *Understanding Resolution In Optical And Magnetic Encoders*, Electronic Design, http://electronicdesign.com/, 2013.

[9] Manhattan distance, http://goo.gl/Y5tQlH

[10] Project source code, https://github.com/AlexBondor/wilkie-wf32

[11] Digilent Contest project presentation video, https://goo.gl/xDoAYr

[12] Instructable page, http://goo.gl/wKvDjF.

[13] L298N H-bridge datasheet, https://goo.gl/1g6BHs.

[14] Bluetooth module datasheet, http://goo.gl/OMFWts.

[15] Bluetooth module tutorial, https://goo.gl/uogCZD

[16] Magnetic encoder datasheet, http://goo.gl/0IEHSY.

[17] Chipkit WF32 datasheet, https://goo.gl/WpRPn6.

[18] Zybo Zynq-700 datasheet, https://goo.gl/fNvvsE.

[19] IR sensor datasheet, https://goo.gl/K318gY.

[20] HPI racing website, https://www.hpi-racing.ro/

[21] Digilent Inc. website, http://store.digilentinc.com/

[22] Robofun website, https://www.robofun.ro/

[23] Gearbest website, http://www.gearbest.com/

Appendix A

Relevant code

Listing A.1: Main.ino

```
1
2 ... includes and definitions omitted ...
3
4 /*
5  *  Gather data from encoders and sensors and send as
6  *  IIC data when requested from iicController on Zybo Zynq-7000
7  */
8 void iicSendData();
9 void iicRecvData(int numBytes);
10 void initPaddingString();
11
12 void setup()
13 {
14   // Serial1 is used by Bluetooth module to receive data
15   Serial1.begin(9600);
16
17   ...
18
19   // IIC initialization
20
21   ...
22
23   robot.connectToWifi("wifiConnectionName", "wifiConnectionPassword");
24   robot.connectToServer("visualizationServerIp", visualizationServerPort);
25
26   start = millis();
27 }
28
29 void loop()
30 {
31   robot.process();
32   if (millis() - start > 50) // Send data every 50 ms not to block the
      server
```

```
33   {
34     start = millis();
35
36     sprintf(headDeg, "h%d", (int)robot.getHeading());
37
38     ... omitted lines of sending data to remote server ...
39
40     robot.writeToServer(headDeg, strlen(headDeg));
41   }
42 }
43
44 void iicSendData()
45 {
46   ... data initialization omitted ...
47
48   sprintf(message, "%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s%*.*s", noCmds, vacF,
       svoF, posX, posY, headDeg, xs1, ys1, xs2, ys2, xs3, ys3,
     IIC_MESSAGE_LENGTH - messageLen, IIC_MESSAGE_LENGTH - messageLen, padding
     );
49
50   Wire.write(message);
51 }
52
53 /*
54  *  Commands list:
55  *  10 - no operation
56  *  20 - brake
57  *  31 - move forward
58  *  32 - move backward
59  *  43 - turn left
60  *  44 - turn right
61  *  55 - toggle vacuum
62  *  56 - toggle sensors free move
63  */
64 void executeCommandFirst()
65 {
66   Serial.println(cmd);
67   switch(cmd)
68   {
69
70   ... omitted switch cases ...
71
72     case 43:
73       robot.turnLeft(atoi(amount));
74       break;
75
76   ... omitted switch cases ...
77
78     default:
79       robot.brake();
```

```
80        break;
81    }
82  }
83
84  /*
85   *  Receive commands from master Zybo
86   *  Commands types:
87   *    Actions:
88   *      m - move
89   *      t - turn
90   *      n - no operation
91   *      s - stop
92   *      p - power
93   *    Directions:
94   *      f - forward
95   *      b - backward
96   *      l - left
97   *      r - right
98   *    Amount:
99   *      XXX - a number represented with digits
100  *    Devices:
101  *      v - vacuum
102  *      o - servo
103  *
104  *  Example of valid commands from master:
105  *  mf10 - move forward 10 cm
106  *  tl45 - turn left 45 cm
107  *  s    - stop; robot brakes
108  *  n    - no operation; robot proceeds normally
109  *  pv   - power vacuum
110  */
111 void iicRecvData(int numBytes)
112 {
113   char x;
114   while(Wire.available() != 0)
115   {
116     x = Wire.read();
117     switch(x)
118     {
119       case 'n':
120         if (cmd != 0)
121         {
122           executeCommandFirst();
123         }
124         cmd = 10;
125         amountIndex = 0;
126         break;
127
128     ... omitted switch cases ...
129
```

```
130       case 'f':
131         if (cmd % 10 != 0)
132         {
133           executeCommandFirst();
134         }
135         cmd += 1;
136         break;
137 
138     ... omitted switch cases ...
139 
140       default:
141         if (x - '0' >= 0 && x - '0' <= 9)
142         {
143           amount[amountIndex++] = x;
144         }
145         else
146         {
147           cmd = 20; // No known operation
148         }
149         break;
150     }
151   }
152 }
```

Listing A.2: Encoder.h

```
// Needed to obtain the RPM value given the time
// between consecutive interrupts in microseconds
//
// T_INT   - time between consecutive interrupts (in microseconds)
// T_EWRus - time the encoder wheel makes a rotation (in microseconds)
// T_EWRs  - time the encoder wheel makes a rotation (in seconds)
// T_WRs   - time the robot wheel makes a rotation (in seconds)
// N_WR/s  - number of robot wheel rotations per second
// RPM     - robot wheel rotations per minute
//
// Number obtained in the following way:
//
// T_EWRus = 3 * T_INT;          // 3 encoder wheel interrupts per encoder wheel rotation
// T_EWRs  = T_EWRus / 1000000; // 1s = 1000000us
// T_WRs   = 53 * T_EWRs;        // it takes 53 encoder wheel rotation for the robot wheel
//                               // to make one revolution (53:1 gear ratio)
// N_WR/s  = 1 / T_WRs
// RPM     = 60 * N_WR / s
//
// So..
//                        1 * 60
// DINT_TO_RPM = ----------------
//                 T_INT * 3 * 53
//                 ------------
//                    1000000
//
```

Listing A.3: Motor.cpp

```
/*
 * Updates the fields of current motor and sets the direction
 * of its pins
 */
void Motor::attach(int directionPinA, int directionPinB, int pwmPin)
{
  _directionPinA = directionPinA;
  _directionPinB = directionPinB;
  _pwmPin = pwmPin;

  pinMode(_directionPinA, OUTPUT);
  pinMode(_directionPinB, OUTPUT);
  pinMode(_pwmPin, OUTPUT);
}

/*
*   Turns the motor in a desired direction
*   forward/backward with a desired speed
*/
void Motor::turn(int speed)
{
  if (speed >= 0)
  {
    _inA = HIGH;
  }
  else
  {
    _inA = LOW;
    speed = -speed;
  }
  // Adjust speed so it won't exceed imposed limits by PWM
  speed < 0 ? speed = 0 : speed > 255 ? speed = 255 : speed = speed;

  digitalWrite(_directionPinA, _inA);
  digitalWrite(_directionPinB, !_inA);
  analogWrite(_pwmPin, speed);
}

/*
*   Brakes the current motor
*/
void Motor::brake()
{
  analogWrite(_pwmPin, 255);
  digitalWrite(_directionPinA, LOW);
  digitalWrite(_directionPinB, LOW);
}
\newpage
```

Listing A.4: SharpSensor.h/SharpSensor.cpp

```
// 4cm from sensor led to center of the robot
#define DISTANCE_FROM_SENSOR_TO_ROBOT_CENTER 4

/*
 * Raw analogRead values translated to
 * cm values. Values obtained by manual
 * testing of the proximity sensor.
 */
const float RAW_TO_CM [531][2] = {
  {200, 50.3},
  {201, 49.9},
  {202, 49.5},
  {203, 49.2},
  {204, 48.8},
  {205, 48.5},
  {206, 48.1},
  {207, 47.8},
  {208, 47.4},
  {209, 47.1},
  {210, 46.8},
  {211, 46.4},
  ...
  {719, 9.9},
  {720, 9.9},
  {721, 9.9},
  {722, 9.8},
  {723, 9.8},
  {724, 9.8},
  {725, 9.8},
  {726, 9.8},
  {727, 9.8},
  {728, 9.7},
  {729, 9.7},
  {730, 9.7}
};

...

/*
 *  Returns the distance in cm if the sensor has
 *  anything blocking its viewing ray. Returns 0 otherwise.
 */
float SharpSensor::getDistance()
{
  _readings[_readingsCount] = getRaw();
  _readingsCount += 1;
  if (_readingsCount >= _avg)
  {
    sort(_readings);
```

```
50      _lastIndex = _readings[_avg / 2] - MIN_RAW;
51      _readingsCount = 0;
52    }
53
54    return _lastIndex < 0 ? 0 : RAW_TO_CM[_lastIndex][1] +
       DISTANCE_FROM_SENSOR_TO_ROBOT_CENTER;
55  }
56
57  void SharpSensor::sort(int* readings)
58  {
59    int temp, swap;
60    for (int i = 1 ; i < _avg; i++)
61    {
62      temp = i;
63
64      while (temp > 0 && readings[temp] < readings[temp - 1])
65      {
66        swap             = readings[temp];
67        readings[temp]   = readings[temp - 1];
68        readings[temp - 1] = swap;
69
70        temp--;
71      }
72    }
73  }
```

Listing A.5: SensorController.cpp

```
/*
 *  This method must be called every loop cycle in order to make
 *  the servo on top of which the sensors are located turn left and right.
 *  As in the case of other peripherals, we make the call each loop
 *  instead of waiting for it because we still want that other peripherals
 *  get the CPU as well.. Sharing is caring!
 */
void SensorController::process()
{
  if (_freeMove)
  {
    if (millis() - _startTime > _myServoTurnDelay)
    {
      _startTime = millis();

      if (_myServoPosition > ZERO_POSITION + ROTATION_ANGLE)
      {
        _myServoDirection = -1;
      }
      if (_myServoPosition < ZERO_POSITION - ROTATION_ANGLE)
      {
        _myServoDirection = 1;
      }

      _myServoPosition += _myServoDirection;
      setServoPosition(_myServoPosition);
    }
  }
  _myServo.write(_myServoPosition);
}
```

Listing A.6: BluetoothController.cpp

```
/*
 *  This method must be called every loop cycle in
 *  order for the Bluetooth module to react to new
 *  commands.
 */
void BluetoothController::listen() {
  if (Serial1.available() > 0) {
    val = Serial1.read();

    if (val == 10) // New line character
    {
      _last = _last - 48; // ASCII code for 0 is 48

      if (_last == 0)
      {
        _isEnabled = !_isEnabled;
        _enableRemoteControlCallbackMethod();
      }
      if (_isEnabled)
      {
        if (_last == 1)
        {
          _moveForwardCallbackMethod();
        }
        if (_last == 2)
        {
          _moveBackwardCallbackMethod();
        }
        if (_last == 3)
        {
          _turnLeftCallbackMethod();
        }
        if (_last == 4)
        {
          _turnRightCallbackMethod();
        }
        if (_last == 5)
        {
          _enablePumpCallbackMethod();
        }
        if (_last == 6)
        {
          _enableServoFreeMoveCallbackMethod();
        }
      }
    }
    _last = val;
  }
}
```

Listing A.7: iic_slave_polling.cpp

```
// This function simply initializes the iic component of the board
int IicInit()
{
  ... initializing code omitted ...
}

/**
 * Returns a structure containing the position, heading and data read from sensors
 * of the robot. The structure contains the parsed data requested by IIC from the
 * slave board.
 */
robot_state IicGetRobotState()
{
  int Status, Index;
  // phase used for parsing the input taken from iic buffer
  // 0 - positionX
  // 1 - positionY
  // 2 - headingAngle
  // 3 - leftSensorX
  // 4 - leftSensorY
  // 5 - frontSensorX
  // 6 - frontSensorY
  // 7 - rightSensorX
  // 8 - rightSensorY
  int Phase = 0;
  int Sign = 1;

  ... read data from slave ...

  /*
   * Verify that received data is correct.
   */
  char CurrentChar;
  int CurrentNumber = 0;
  int StoreData = 0;
  for (Index = 0; Index < RECV_BUFFER_SIZE; Index++)
  {
    StoreData = 0;
    CurrentChar = RecvBuffer[Index];
    switch (CurrentChar)
    {
      case '-':
        Sign = -1;
        break;
      case ',':
        CurrentNumber *= Sign;
        StoreData = 1;
```

```
48          break;
49        default:
50          CurrentNumber = CurrentNumber * 10 + (CurrentChar - '0');
51          break;
52      }
53      if (StoreData == 1)
54      {
55        switch(Phase)
56        {
57          case 0:
58            Result.numberOfCommands = CurrentNumber;
59            break;
60          ... other switch cases omitted ...
61          default:
62            break;
63        }
64        CurrentNumber = 0;
65        Phase ++;
66        Sign = 1;
67      }
68    }
69    return Result;
70  }
71  // Simply sends a command via IIC
72  int IicSendCommand(u8* message, int messageSize)
73  {
74    while (XIicPs_BusIsBusy(&Iic)) {
75      /* NOP */
76    }
77
78    int Status, i;
79    Status = XIicPs_MasterSendPolled(&Iic, message, messageSize,
       IIC_SLAVE_ADDR);
80    if (Status != XST_SUCCESS)
81    {
82      // failed at this point..
83      // do something about it
84      return -1;
85    }
86
87    return 0;
88  }
89  robot_state IicGetInvalidRobotState()
90  {
91    ... sends a robot state initialized with invalid values ...
92  }
93  void IicPrintRobotState(robot_state RobotState)
94  {
95    ... prints the state received as parameter ...
96  }
```