People’s Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M’Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
The Requirements for the Degree of

MASTER

In **Electrical and Electronic Engineering**

Option: **Computer Engineering**

Title:

Design & Implementation of a navigation Robot

Presented by:

- **DJEZAIRI Salim**
- **FALI Thamila**

Supervisor:

Dr. KHALIFA

Registration Number:………/2016

We dedicate this work for our parents who supported us through our entire curriculum, and pushed us to be the WHO we are today; and to our beloved friends who supported and motivated us during the last semester.

To my sisters and brother

May god keep you all.

Acknowledgement

We would like to express our deepest thanks to our supervisor Dr Khalifa for his guidance, support and encouragement during the work; and a special thanks to our beloved friends that helped us achieve this work. To souleyman, Ilyes, Feriel, Meriem and Amira,This semester wouldn’t have been the same without you.

Abstract

This project goes through the design and implementation process of an autonomous navigation robot, capable of exploring an unknown environment and perform numerous tasks such as, 2D reconstruction of the environment, self-location inside the map, finding shortest path to a given destination, map inspection and target search.

The robot uses a front camera and two sensors to observe the environment and navigate inside it. The System is built around the Raspberry PI microcomputer running Raspbian operating system and programmed with Java and Python languages.

The different tasks' algorithms are first simulated in a software application, without hardware restriction in an error free environment, then implemented on the hardware design and tested inside a maze created in the laboratory.

The result obtained were satisfactory, since even with some limitations, the robot performed the requested tasks.

Table of Contents

- CPU Central Processing Unit
- DC Direct Current
- DSP Digital Signal Processor
- FET Field Effect Transistor
- GND Ground
- GPIO General Purpose Input Output
- GUI Graphical User Interface
- HDMI High Definition Multimedia Interface
- I2C Inter-Integrated Circuit
- LAN Local Area Network
- LED Light Emitting Diode
- MCU Micro-Controller Unit
- MOSFET Metal Oxide Semiconductor Field Effect Transistor
- RAM Random Access Memory
- SD Secure Digital
- SDHC Secure Digital High Capacity
- SOC System On Chip
- SPI Serial Peripheral Interface
- UART Universal Asynchronous Transmitter Receiver
- UML Unified Modeling Language
- USB Universal Serial Bus
- YANA You Are Not Alone

1. INTRODUCTION

Nowadays, automation is taking an increasingly important role in human’s environment. Whether in daily life or in industries, the last years proved that autonomous robots are taking the lead in many fields such as manufacturing, utilities, transportation and exploration.

Automata have always been a fascinating research field for engineers. Over the centuries, they went from purely mechanical and human depending tools to fully autonomous robots.

An autonomous robot is an electro-mechanical machine guided by computer software that can perform behaviors or tasks with a high degree of autonomy. In other words, it is a machine that can acquire information about the environment, work for an extended period with no dependency, move all or part of itself throughout its operation environment without human assistance and avoid situations that are harmful to people, properties or itself. Thus they are particularly desirable in fields such as space exploration, household maintenance and delivering goods and services. Robots can be classified depending on the degree of autonomy they have, the task they perform and their design complexity.

All industries and fields can benefit from autonomous machines but, they are most suited for exploration and transport, where they can replace man in dangerous or hostile environment, such as underwater, underground or space. For this purpose self-driving, obstacle avoiding and self-locating cars are attracting developers and engineers.

This project aim is to design and implement an autonomous navigation robot capable of exploring an unknown environment, reconstructing it in a 2D map, and navigating through it. In order to illustrate and test the application, a maze is constructed, the robot explores it and send the reconstruction to the main interface panel in real time. After map exploration, the robot must be able to locate itself and go to any location through a shortest path. In addition, it must be able to search for a given target and locate it on the map. For this purpose a control system with a camera and sensors is used; depending on the feedback data received from the sensing unit, the robot must be able to adjust its position and avoid obstacles.

This topic is an opportunity for us to deepen our knowledge in the robotics field, make use of the past five years acquired techniques and merge two indivisible aspect of modern technologies, namely software programming of applications and hardware implementation of systems. Robotics is a domain that encompasses most of the engineering fields. This project will allow us to build a fully functioning system.

As stated previously, the main aim of this project is to design an autonomous navigation robot. The digital controller system is based on the raspberry PI microcomputer and is developed using Java and Python languages. The robot consists of a four motorized wheels car and two sensing peripherals: distance sensors and camera. This project is divided into two parts, the first one simulates the robot behavior in a software, error free environment, while the second one implements the actual system and tests its functionality in laboratory conditions. The main functions the robot must achieve are:

- Exploring and constructing a 2D mapping of the environment.
- Finding an optimal path to a given destination.
- Inspecting the environment and searching for changes.
- Finding a target inside the environment and returning its location.

The report is organized into five chapters. Considering the first chapter to be this present introduction. Chapter two presents the theoretical background and technical details of the different components forming the Raspberry system. Hardware system design and implementation is discussed in chapter three, it represents the bulk of the report where each system block is discussed and treated. Chapter four on the other hand, gives a software design of the whole system. Finally, chapter five implements the actual software algorithms, on the simulation software and hardware system and tests its functionality. In the end, a conclusion terminates the work with remarks and discussions including further works suggestions.

2. THEORETICAL BACKGROUND

2.1 INTRODUCTION

This chapter introduces the theoretical background needed to design the robot's hardware system and the technical details of the different components. In the first section, we introduce the robot brain and main control part, while in the second one we present the actuating and sensing parts.

2.2 RASPBERRY PI

Nowadays, embedded systems are used in most modern electronic designs and span all aspects of modern life. For years, design engineers have largely relied on programmed technology such as general-purpose microprocessors, microcontrollers and digital signal processors (DSPs) as their main processing engines. But with todays' technology advances, more performant devices are available. The Raspberry PI is a step above old programmed technologies, since it is one of those all in one System On a Chip (SOC) that regroups all the features of a single board computer in a credit card sized platform[1].

2.2.1 HISTORICAL BACKGROUND

Developed in the United Kingdom by the Raspberry Pi Foundation with the intent to promote the teaching of basic computer science in schools and developing countries.[1][36] Several generations of Raspberry Pi's have been released. The first generation (PI 1) was released in February 2012 in basic model A and a higher specification model B. A+ and B+ models were released a year later. Raspberry Pi 2 model B was released in February 2015 and Raspberry PI 3 model B in February 2016. A cut down computer model was released in April 2014 and a PI Zero with smaller footprint and limited IO (GPIO) capabilities released in November 2015.

2.2.2 DESIGN AND IMPLEMENTATION

All models of Raspberry PI feature a Broadcom system on a chip (SOC) which include an ARM compatible CPU and an on chip graphics processing unit GPU (a VideoCore IV). CPU speed range from 700 MHz to 1.2 GHz for the PI 3 and on board memory range from 256 MB to 1 GB RAM. Secure Digital SD cards are used to store the operating system and program memory in either the SDHC or MicroSDHC sizes. Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phono jack for audio. Lower level output is provided by a number of General input Output pins (GPIO) which support common protocols like I2C. Some models have an RJ45 Ethernet port and the Pi 3 has on board WiFi 802.11n and Bluetooth [2] [3] [4].

The Foundation provides Debian and Arch Linux ARM distributions for download [5], and promotes Python as the main programming language.

Relative to its size, the Raspberry PI is a powerhorse of a computer – it can drive HDMI displays, process mouse, keyboard, and camera inputs, connect to the Internet, and run fullfeatured Linux distributions. But it’s more than just a small computer, it’s a hardware prototyping tool. The PI has bi-directional I/O pins, which can be used to drive LEDs, spin motors, or read button presses.

2.2.3 RASPBERRY PI 2 MODEL B

Regarding our system’s needs, we opted for a Raspberry PI 2 modal B. It is similar the previous ones but with double sized RAM (1GB) and a much faster processor. Which are the most suited specifications for our application.

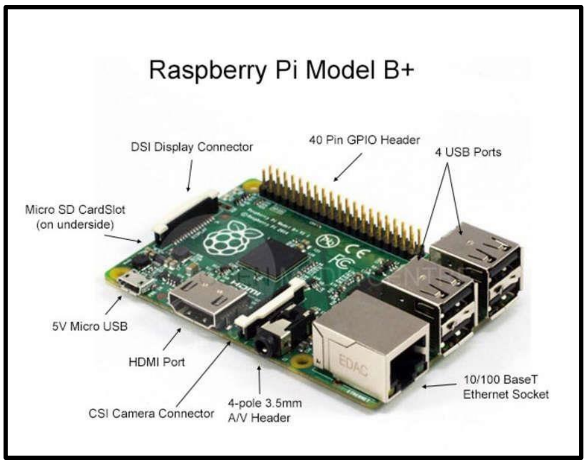


***FIGURE 2.1**. RASPBERRY PI 2 MODEL B [38]*

The PI 2 modal B board as shown in figure 2.1, contains:

- Microprocessor Broadcom BCM 2836 SOC with a 900MHz quad –core ARM CortesA7 CPU.
- 1 GB RAM
- 4 USB ports

- 40 GPIO pins (27 GPIO with UART I2C bus, SPI bus with two chip selects, +3.3V, +5V and grounds)
- Full HDMI port
- Ethernet port
- Combined 3.5 mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10 [6].

2.3 ROBOT'S BODY

The first step toward implementing our robot is to choose a chassis with the desired specifications, to create the robot body carrying all the circuitry and components. The chassis includes four independently motorized wheels (figure.2.2).

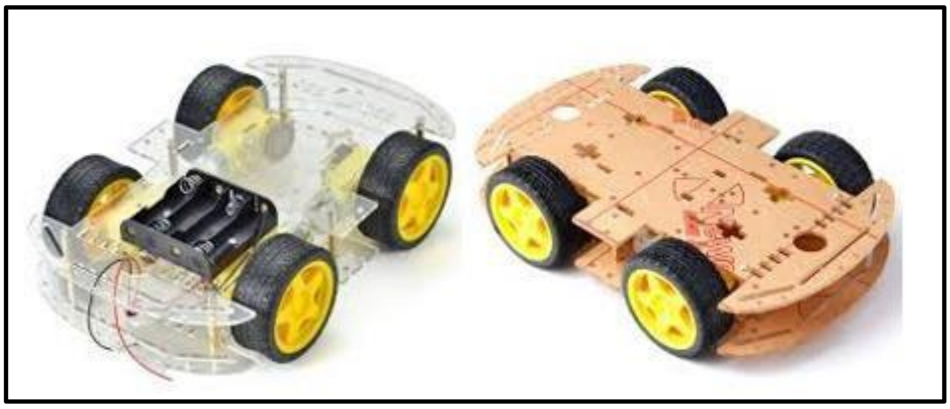


FIGURE 2.2. *ROBOT CHASSIS [39]*

All what needs to be controlled is the input to the DC motors. A positive voltage moves the car forward while a negative one moves it backward.

Since the chassis does not contain a differential drive mechanism, (the direction drive mechanism), the left and right manoeuver are operated using the four motorized wheels. So that a right turn is achieved with the two left motor rotating forward and the two right ones rotating backward, while a left turn is achieved with the right wheels rotating forward and the left ones rotating backward. This causes the car to turn over its gravitational center with a specific angle depending on the time power is supplied.

2.3.1 DC MOTOR

A DC motor is a device that converts electrical energy into a mechanical one, it generates torque by creating an interaction between a fixed and rotating magnetic field. The rotating field is created by passing a DC current through several different windings on the armature (rotating part) and timing which winding is powered through a device called a commutator. Power is applied to the armature by brushes which ride on the commutator [7] (figure.2.3). It is composed of several major components, which includes the following:

- Frame
- Shaft
- Bearings
- Main field windings (stator)
- Armature (rotor)
- Commutator
- Brush assembly

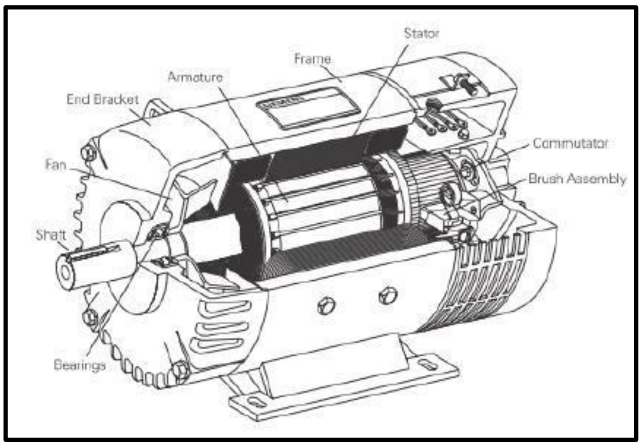


FIGURE 2.3 *DC MOTOR MAIN COMPONENTS [7]*

2.3.2 H-BRIDGE

The moving aspect of a robot is what is most important. Attaching a motor directly to a chip that controls its movements would be great. But most chips cannot supply enough current or voltage to spin a motor. The current supplied by an average MCU is of the order of 5 mA compared to the 500mA required by a simple motor to rotate. In addition, motors tend to create electro-magnetic noise when they change direction of rotation or speed which is a damaging process to any close electronic circuit.

Motor drivers have then been developed to specifically solve these problems and be able to control the speed and rotation's direction of the motors. Thus, they represent the link between the digital circuitry and the robot's mechanical parts.

2.3.2.1 OPERATING PRINCIPLES

In general, an H-bridge is a rather simple motor drive circuit containing four switching element, with the load at the center, in an H-like configuration (figure 2.4).

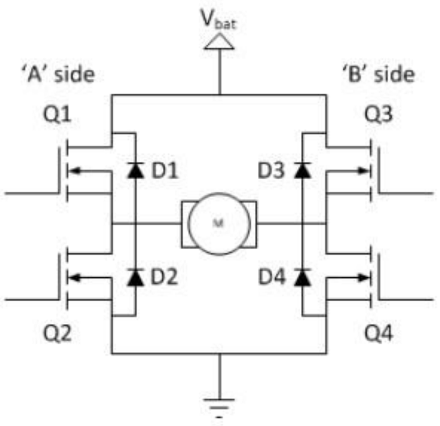


FIGURE 2.4 *H-BRIDGE CONFIGURATION [8]*

The switching elements (Q1...Q4) are usually bi-polar or FET transistors, in some high voltage applications IGBTs. The diodes (D1...D4) are called catch diodes and are usually of a Schottsky type. The top-end of the bridge is connected to a power supply (battery for example) and the bottom end is grounded. In general, all four switching elements can be turned on and off independently, though there are some obvious restrictions. The load can in theory be anything wanted, by far the most used application of an H bridge is with a brushed DC or bipolar stepper motor (steppers need two H-bridges per motor) load [26].

The basic operating mode of an H-bridge is fairly simple: if Q1 and Q4 are turned on, the left lead of the motor will be connected to the power supply, while the right lead is connected to ground. Current starts flowing through the motor which energizes it in (let's say) the forward direction and the motor shaft starts spinning.

If Q2 and Q3 are turned on, the reverse will happen, the motor gets energized in the reverse direction, and the shaft will start spinning backwards. The principle is illustrated in figure 2.5.

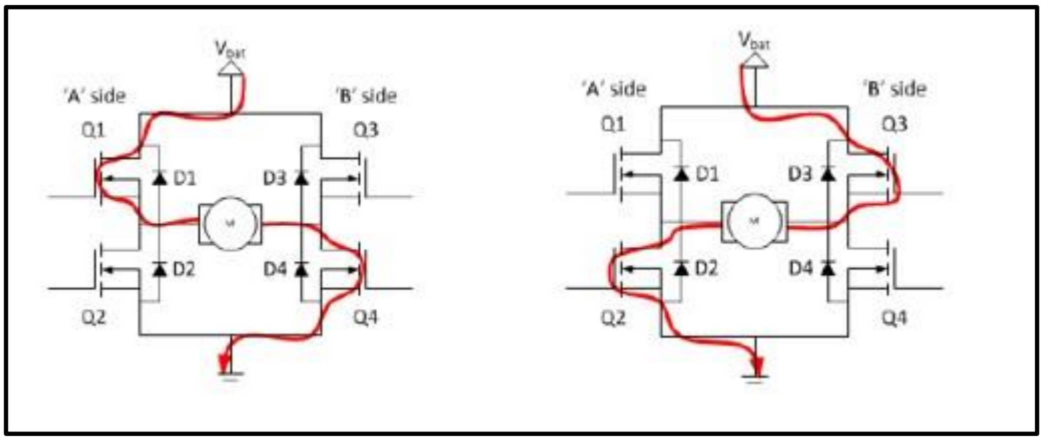
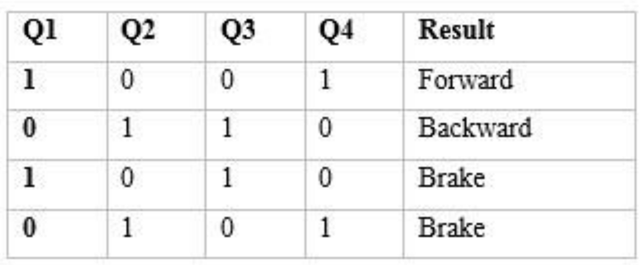


FIGURE 2.5 *H-BRIDGE MOTOR DIRECTION OF ROTATION CHANGE [8]*

In a bridge, both Q1 and Q2 (or Q3 and Q4) should never be closed at the same time. If so, a really low-resistance path between power and GND is created, effectively short-circuiting the power supply. This condition is called 'shoot-through' and is an almost guaranteed way to quickly destroy the bridge, or something else in the circuit [26]

There exist 16 possible combinations of paired transistors. Knowing that turning on the switches of the same side is not advisable leaves 4 combinations that are summarized in the following table.



Q1	**Q2**	**Q3**	**Q4**	**Result**
1	0	0	1	Forward
0	1	1	0	Backward
1	0	1	0	Brake
0	1	0	1	Brake

TABLE 2.1 EFFECTIVE TRANSISTOR COMBINATONS

The first two rows show the two state of motor rotation, forward and backward. Whereas the two last rows show an interesting combination called braking. It is when two transistors facing each other are turned on simultaneously. This causes a short circuit across the motor which leads to the motor's generator effect to works against itself. The turning motor generates a voltage which tries to force the motor to turn the opposite direction and causes it to rapidly stop spinning. The Braking circuitry is needed if very good motors are used in order to keep them from spinning freely when no control signal is being applied [8].

2.3.2.2 IMPLEMENTATION

An H-bridge can be designed using discrete components such as relays [27] and transistors, or integrated circuits. Relays are best for turning on and off a device, while transistors, feats when smaller physical size, low driving voltage and high speed switching is needed. A solidstate H-bridge is typically constructed using opposite polarity devices, such as P-channel MOSFETs connected to the high voltage bus and N-channel MOSFETs connected to the low voltage bus.

H-bridges are also available as integrated circuits. The most efficient ones use NPNs on both the high side and low side because they typically have one third of the ON resistance of PNPs.

2.3.2.3 CRITERIA OF SELECTION

Different criteria must be observed and analyzed before choosing an appropriate H-bridge. First, the voltage and current ratings of the motor and its output power capability. Second, the number of circuit connections and last, the bridge size, such that more compact circuits are easier to fit into arbitrary designs.

For safety purposes a bridge design needs to be "smoke proof" [8]. Most drivers have two "Smoke proof" control inputs such that designs won't self-destruct if both control inputs are "low", or if both inputs are "high".

2.3.3 STEPPER MOTOR

In contrast to a regular motor that spins when powered on, a stepper motor rotates by taking a fixed number of steps, the thing that allows it to be used when a rotation by a certain angle is needed [28]. Hence, a stepper motor is a brushless DC motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed. They have multiple coils that are organized in groups called "phases". By energizing each phase in sequence, the motor will rotate, one step at a time.

Stepper motors are practically good for positioning, speed control and low speed torque, even so they have a number of limitations, such as low efficiency, limited high speed torque and no feedback [29].

2.3.3.1 STEPPER MOTOR TYPES

Stepper motors come in sizes ranging from smaller than a peanut to big NEMA 57 monsters. One of the first things to consider while purchasing a stepper motor is the work that the motor has to do, since larger motor are capable of delivering more power, the motor size will depend on the application.

The next thing to consider is the positioning resolution required. The number of steps per revolution ranges from 4 to 400. Resolution is often expressed as degrees per step. A 1.8° motor is the same as a 200 step/revolution motor.

2.3.3.2 UNIPOLAR VS BIPOLAR

Unipolar drivers, always energize the phases in the same way. One lead, the "common" lead, will always be negative. The other lead will always be positive. Unipolar drivers can be implemented with simple transistor circuitry. The disadvantage is that there is less available torque because only half of the coils can be energized at a time.

Bipolar drivers use H-bridge circuitry to actually reverse the current flow through the phases. By energizing the phases with alternating the polarity, all the coils can be put to work to turn the motor.

A two phase bipolar motor has two groups of coils. A four phase unipolar motor has four. A two-phase bipolar motor will have four wires - two for each phase. Some motors come with flexible wiring that allows to run the motor as either bipolar or unipolar.

2.3.3.3 THE 28 BYJ-48 STEPPER MOTOR

The 28 BYJ-48 (figure.2.6) Is a specific unipolar stepper motor with 5V input voltage and a gear reduction of 1/64 which leads to a total of 2048 steps in full stepping (since one revolution takes 32 real steps (32*64)), and 4096 steps in half stepping (2*2048) [30]. It is driven using the ULN2003 driver, and fits exactly the projects requirement. The motor will serve as a rotation shaft for the camera, it will be programmed to rotate steps of 90 degrees to enable the camera to see in all direction giving it the time to take clear pictures.

FIGURE 2.6 *28 BYJ-48 STEPPER MOTOR [30]*

Inside the stepper, we can find two fixed coils with 4 sets of 8 teeth and a central rotating core, which is made up of 16 north/south poles. Each of these teeth can be magnetized to act like tiny norths or tiny souths. The teeth are magnetized when the coil has an electric field going through it, either upward or downward [30].

2.3.3.4 FULL VS HALF STEPPING

Figure 2.8 depicts the 28BYJ-48 stepper motor's teeth. For the sake of illustration, we keep track of one of the north poles of the central core, represented in the figure by a red bar.

When the first coil is energized the core's north pole will move to a south pole creating the first step, then the upper coil is off and the bottom one is on, so that the bottom teeth become north poles. This causes the core's North Pole to shift again toward the closest south, rotating the core a small amount and ending the second step. In the third step the upper coil is on again but with upper teeth being souths, causing the core to move. Lastly bottom coil is on with bottom teeth being souths, inducing motion. This ends the cycle which brings back to first step again (figure.2.7).

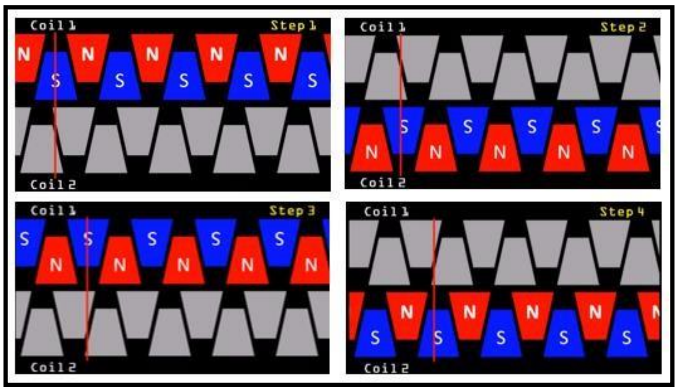
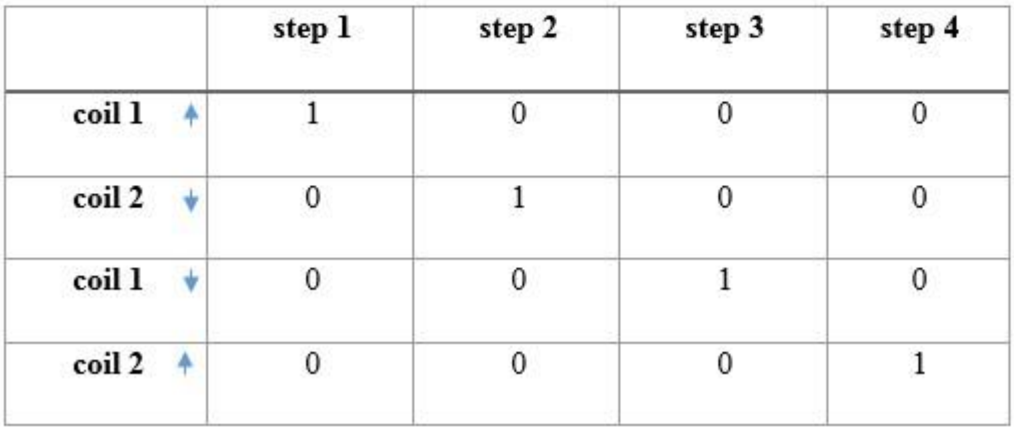


FIGURE.2.7. *MAGNETIC ROTATION STEPS [30]*

There are in the end four steps for each cycle which makes sense as there are four sets of teeth, a pair for each coil. And since 32 steps are needed for one revolution of the core means there are 8 cycles taking place.

Single phase stepping discussed above is resumed in table.2.2, each arrow in the table represent the north pointing direction.



	step 1	step 2	step 3	step 4
coil 1 ↑	1	0	0	0
coil 2 ↓	0	1	0	0
coil 1 ↓	0	0	1	0
coil 2 ↑	0	0	0	1

TABLE 2.2 SINGLE PHASE STEPPING

Stepping can better be represented as seen in figure.2.8. Two coils in different colors (green and orange) are represented. The two coils have two cases each, first case is with north pointing upward, while second case has its north pointing downward. The red arrow in the middle shows in which case we are.

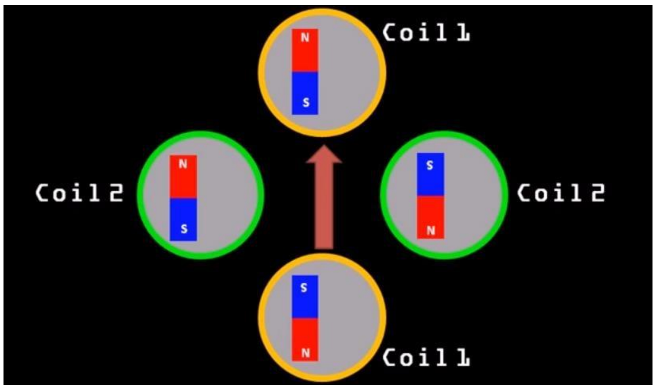


FIGURE.2.8. *STEPPING, ARROW REPRESENTATION [30]*

Single phase stepping works perfectly, but the spinning force it generates is low. And this is where dual phase stepping or full stepping gets the advantage, such that, two coils are ON at a time increasing significantly the turning force. Figure 2.9 illustrates dual stepping, while table.2.3 resumes its steps.

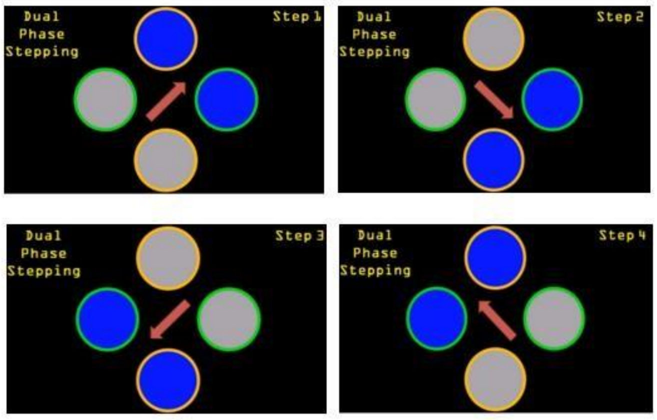
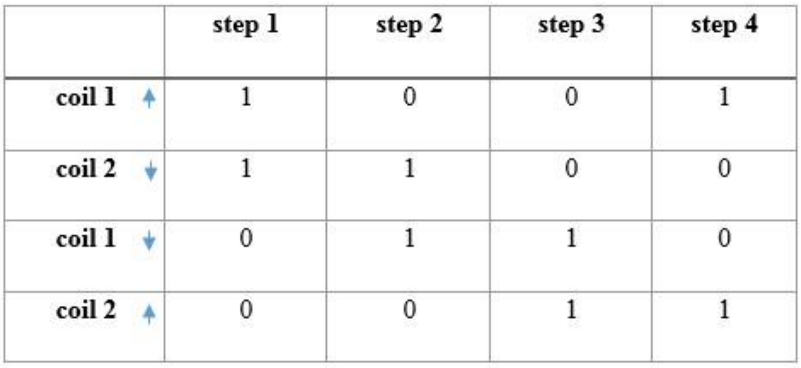


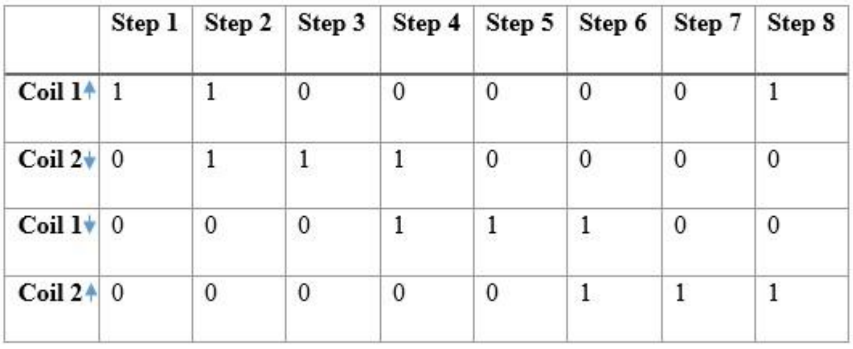
FIGURE.2.9. *FULL STEPPING, ARROW REPRESENTATION [30]*



	step 1	step 2	step 3	step 4
coil 1 ↑	1	0	0	1
coil 2 ↓	1	1	0	0
coil 1 ↓	0	1	1	0
coil 2 ↑	0	0	1	1

TABLE 2.3 *DUAL PHASE STEPPING*

The drawback of this technique is that it requires twice the electro-current of a single phase stepping, which lead us to half stepping. It is a combination of full and single phase stepping resulting in 8 steps that alternate between one and two coils being ON. But the downside of it, is that the turning force (resolution) is about 70% to that of a full stepping and it also appears to be twice as slow as full stepping [30]. Results are best resumed in table 2.4.



	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
Coil 1↑	1	1	0	0	0	0	0	1
Coil 2↓	0	1	1	1	0	0	0	0
Coil 1↓	0	0	0	1	1	1	0	0
Coil 2↑	0	0	0	0	0	1	1	1

TABLE 2.4 *HALF STEPPING*

2.3.4 DISTANCE SENSORS

In contrast to its moving aspects, the second most important feature of an autonomous robot is its capacity to sense the outside environment. This is achieved via sensors that get information from the environment and convert them to different signals. One of the most extensively used sensor in robotic and automotive applications is the proximity (distance) sensor. A proximity sensor is a sensor capable to detect the presence of nearby objects in a defined range without any physical contact.

2.3.4.1 TYPES OF DISTANCE SENSORS

Different targets and different objectives demand different sensors. Depending on the application and the principle of operation, each type of sensor will give different performances.

Common types of non-contact proximity sensors include inductive proximity sensors, capacitive proximity sensors, ultrasonic proximity sensors, and photoelectric sensors. Only two of these will be discussed in this part due to their popularity and suitability for this kind of project namely, ultrasonic and infrared sensors.

2.3.4.2 INFRARED SENSORS

Photoelectric sensors are widely used in the fields of electrical and electronic controls. An infrared sensor emits an electromagnetic field or a beam of electromagnetic radiation (infrared), and looks for changes in the field of the return signal. IR sensors working principle is based on triangulation: a pulse of light (wavelength range of 850nm +/-70nm) is emitted and then reflected back (or not reflected at all). When the light returns it comes back at an angle that depend on the distance of the reflecting object [9]. Triangulation works by detecting this reflected beam angle - by knowing the angle, distance can then be determined. (figure.2.10).

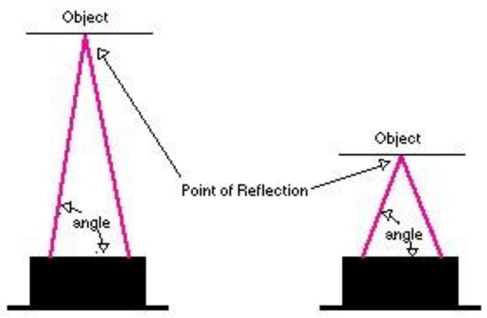


FIGURE.2.10. *TRIANGULATIO PRINCIPLE OF IR SENSOR [9]*

2.3.4.3 ULTRASONIC SENSORS

Another kind of proximity sensors uses sound waves to sense the presence or absence of an object. Sound consists of oscillating waves through a medium such as air. Very low frequency sound below Acoustic is defined as “Infrasound” and high frequency sounds above are called “Ultrasound”. Ultrasonic sensors are designed to sense object proximity using ultrasound reflection, similar to radar. It is based on the calculation of the time it takes to reflect ultrasound waves between the sensor and a solid object. Ultrasound is mainly used because it’s inaudible to the human ear and is relatively accurate within short distances.

A basic ultrasonic sensor consists of one or more ultrasonic transmitters (basically speakers), a receiver, and a control circuit. The transmitters emit a high frequency ultrasonic sound, which bounces off any nearby solid objects. Some of that ultrasonic noise is reflected and detected by the receiver on the sensor (figure 2.11).

That returned signal is then processed by the control circuit to calculate the time difference between the signal being transmitted and received.

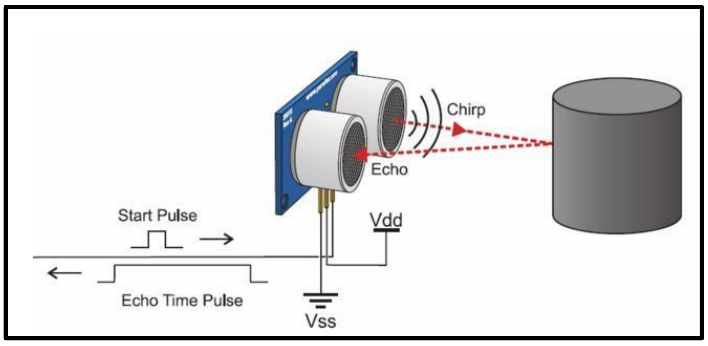


FIGURE.2.11. *ULTRASONIC SENSOR WORK PRNCIPLE [10]*

2.3.4.4 IR VS ULTRASONIC SENSORS

Considering the current application, it is important to treat the advantages and drawbacks of each sensor’s type.

Infrared sensors, emit infrared light, and therefore the sensors cannot work accurately outdoors or even indoors, if there is direct or indirect sunlight. In addition, the working principle of such sensors are based on triangulation. Now since light does not reflect the same way off every surface, the infrared sensor reading will be different for different surfaces , different colors, and different shades even if the range is the same.

Conversely, ultrasonic sensors use sound instead of light for ranging and thus are not affected by outside light. Hence, they can be used outside in bright sunlight. These sensors are amazingly accurate and work for larger ranges, though they may be thrown off by a sound absorbing obstacle, like a sponge. But the real issue that arises is the "ghost echo" issue [10]. As can be seen in figure 2.12, the walls bounce off in a strange pattern causing a ghost effect.

In the case of this project, the robot will be freely moving in a maze where most obstacles have square shapes which eliminate the possible ghost echo effect. Thus, the ultrasonic sensor imposed itself as the best choice.

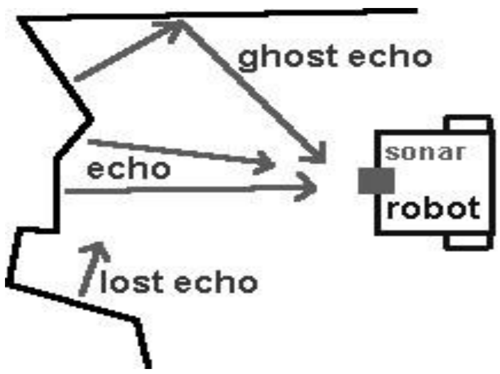


FIGURE.2.12. *GHOST EFFECT OF ULTRASONIC SENSORS [9]*

2.3.5 USB CAMERA

In order to ensure the image processing and voice command part of the system it is necessary to use a cam (to record video and images) and a mic (to record sound). Since the Raspberry PI contains four USB ports, the choice for a USB camera and mic seemed most suited. The camera is a 5 mega pixel webcam with a built-in mic.

FIGURE.2.13. *USB CAMERA*

3. HARDWARE DESIGN

3.1 INTRODUCTION

This chapter covers in some details the application's hardware design and structure. The self-navigating robot starts with a user's voice command, the robot begins by reading data from the outside environment through proximity sensors and camera (sense the obstacle, check wall color). Data is then processed by the Raspberry PI to generate the necessary movement decisions, send the appropriate commands to the motor drivers and display the necessary outputs on the user interface.

We can thus, divide the hardware system into three main blocks, each of these blocks' functionality will be treated and explained in details in the following sections. The first section, highlights the moving aspects of the robot, while the second one will focus on its sensing part with the sensor unit examination. The third section, on the other hand, will regroup the two latter ones in the controlling and processing unit.

3.2 MOTOR DRIVING UNIT

As stated in the previous chapter, the robot chassis comes with four motorized wheels. Each DC motor has a voltage/current range of 3V to 12V and 70mA-250mA. In this application the ratings used are 5V-100mA.

In order to interface the motors with the overall system and ensure various speed and direction control, H-bridges are used. To choose the correct chip, peak currents are measured when motors are first powered on with 5V. Results are tabulated in Table 3.1.

	Reverse peak current	**Forward peak current**
Motor 1	0.540 A	0.582 A
Motor 2	0.591 A	0.599 A
Motor 3	0.469 A	0.554 A
Motor 4	0.559 A	0.589 A

***TABLE 3.1**.MOTORS PEAK CURRENTS*

Relying on these motor’s characteristics, it is safe to choose a double half bridge L293D chip, since it is designed to provide bidirectional drive currents of up to 600 mA at voltages rate from 4.5V to 36V. Figure 3.1 shows the internal structure and pin distribution of the chip.

Remark: the chip contains four grounds, they actually serve as heat sink to protect the chip from damages.

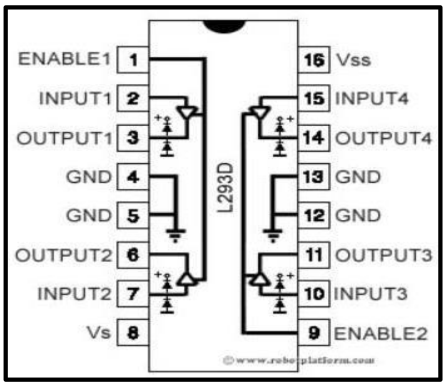


FIGURE 3.1.*L293D CHIP [13]*

The chip enables a voltage to be applied across a load in either directions. Therefore, it allows full control over a standard electric DC motor. Hence with this motor drive, the Raspberry PI can electronically control the robot movement to be forward, backward, left or right.

Since we have four DC motors, the motor drive circuit will contain two L293D chips, each chip will control two motors. The drivers can be enabled at the same time or independently using their enable pins. When an enable input is high, the associated driver is enabled. The outputs vary according to the values of the inputs, and direction is controlled by asserting one of them high and the other one low.

The drivers are powered from the PI’s 5V pin (Vss), while the motors are externally powered by a 5V battery (Vs) (figure 3.2). Drivers’ outputs connect to motors, while its inputs connect to the PI’s GPIO pins. Each motor has a pair of control inputs from the PI (blue lines) which leads to eight control GPIO pins being used. Since our application requires the motors to always be enabled at the same time, only one control pin from the PI is used for chip enable (yellow line).

In addition, all the devices' grounds are connected together.

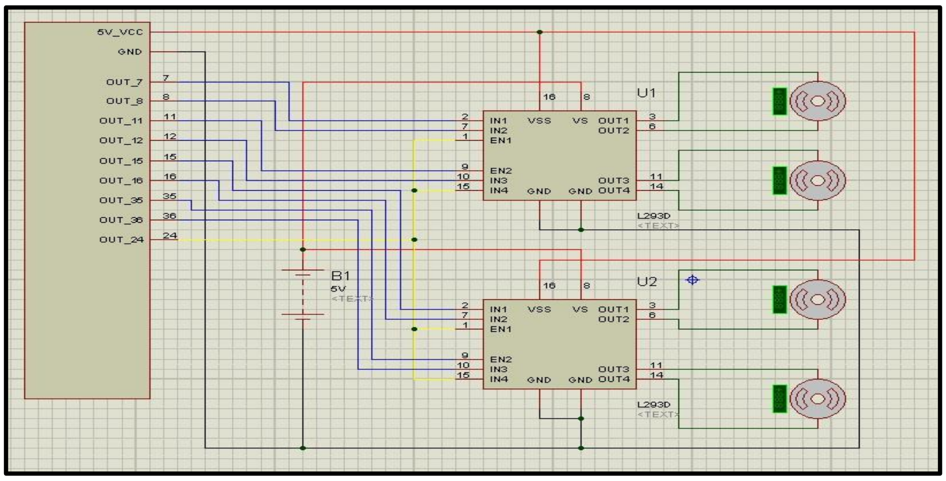
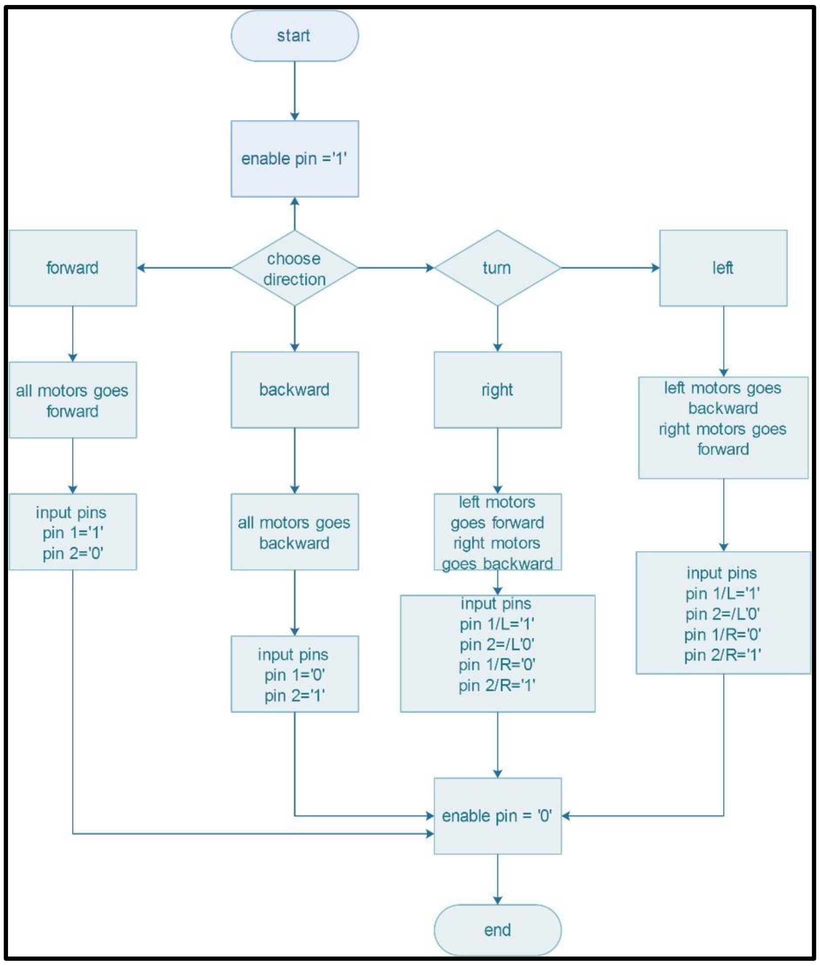


***FIGURE 3.2**.MOTOR DRIVING UNIT CIRCUIT*

The following schematics, illustrates the pin states to generate the different required displacements.



start
enable pin ='1'
forward
choose direction
turn
left
all motors goes forward
backward
right
left motors goes backward right motors goes forward
input pins pin 1='1' pin 2='0'
all motors goes backward
left motors goes forward right motors goes backward
input pins pin 1/L='1' pin 2=/L'0' pin 1/R='0' pin 2/R='1'
input pins pin 1='0' pin 2='1'
input pins pin 1/L='1' pin 2=/L'0' pin 1/R='0' pin 2/R='1'
enable pin = '0'
end

***FIGURE 3.3**.MOTOR DRIVING UNIT FLOWCHART*

3.2.1 SPEED CONTROL WITH PULSE WIDTH MODULATION (PWM)

In order to control the motors' rotation speed, pulse width modulation (PWM) is implemented inside the Raspberry PI in software, and applied to the enable pin. PWM is a technique that allows control over the total power supplied to a load (motor). This is done by varying the on and off times of the signal, in other words, varying the duty cycle of the enable signal will then vary the power supplied to the motor, resulting in a precise motor's speed rate control. Knowing that a 100% duty cycle means setting enable high (5V) all the time and 0% duty cycle means grounding the signal all the time, the speed is then proportionally related to the duty cycle's percentage; the longer the signal is "on" compared to "off" time, the higher the total power supplied to the load and the faster is the motor's rotation [32].

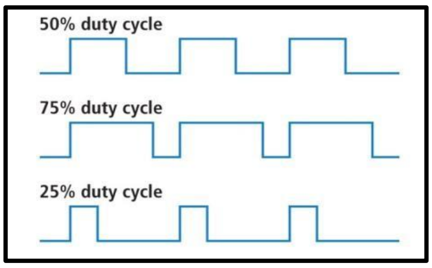


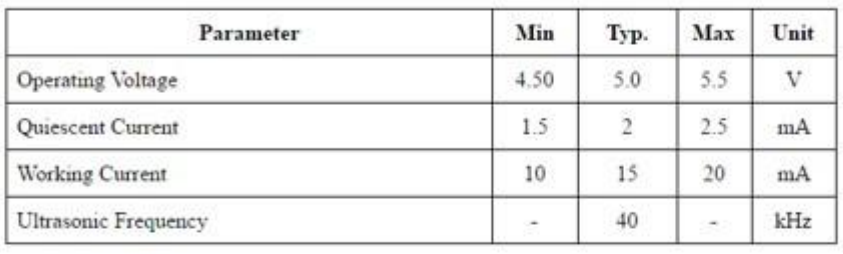
***FIGURE 3.4**.PWM DUTY CYCLES [32]*

3.3 SENSOR UNIT

To determine the distance between the robot and nearby objects, an ultrasonic proximity sensor is used. The camera is used to determine the color of the obstacle.

3.3.1 HC-SR04 ULTRASONIC SENSOR

The HC –SR04 ultrasonic sensor provides 2cm- 400cm non-contact measurement function and a ranging accuracy of three mm [33][34]. The module includes ultrasonic transmitter, receiver and control circuit. Table.3.2 contains the sensor specifications.



Parameter	Min	Typ.	Max	Unit
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz

***TABLE 3.2**.HC-SR04 VOLTAGE & CURRENT SPCIFICATIONS*

The module has four pins:

- VCC -5V-from the PI
- TRIG –trigger the measurement starting
- ECHO –outputs an echo signal to the controller
- GND –from the PI

To start measurement, Trig pin must receive a high pulse for at least 10 μs, this will initiate the sensor to send out an eight cycles of ultrasonic burst at 40KHz and wait for the reflected ultrasonic burst. When the latter is received, the ECHO pin is set high for a period proportional to the distance (figure 3.5).

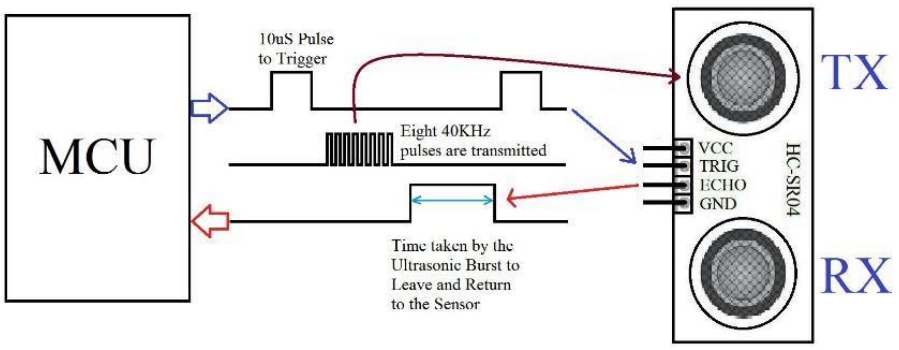

MCU
10uS Pulse to Trigger
Eight 40KHz pulses are transmitted
Time taken by the Ultrasonic Burst to Leave and Return to the Sensor
TX
RX
VCC
TRIG
ECHO
GND
HC-SR04

***FIGURE 3.5**.SENSING PRINCIPLE [33]*

3.3.2 SENSING UNIT'S IMPLEMENTATION

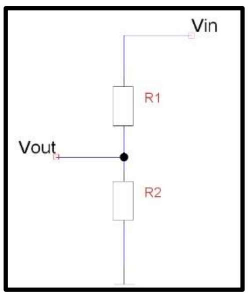


***FIGURE 3.6**.VOLTAGE DIVIDER*

The sensor output signal (ECHO) on the HC-SR04 is rated at 5V. However, the input pin on the Raspberry PI GPIO is rated at 3.3V. Sending a 5V signal into that unprotected 3.3V input port could damage the GPIO pins. Hence, in order to interface the two devices together, the use of a voltage divider circuit is needed. The circuit consists of two resistors, to lower the sensor output voltage to a level that the Raspberry PI can handle.

A voltage divider consists of two resistors in series connected to an input voltage (Vin), which needs to be reduced to our voltage (Vout). In our circuit, Vin is ECHO which needs to be decreased from 5V to 3.3V. We need to calculate the value of the resistors R1 and R2, considering that

$$Vout = \frac{R2}{R1+R2} Vin \text{ Yields to, } \frac{Vout}{Vin} = \frac{R2}{R1+R2}$$

We set a value for R2, R2=10kΩ and search for R1.

$$\frac{3.3}{5} = \frac{10000}{R1+10000} \Rightarrow \text{Hence, } R1 = 5151.51\Omega$$

Practically we have used a combination of a 4.7KΩ resistor with a 330Ω and 360Ω ones which yields to a total of R1 = 5390Ω. Figure.3.6 depicts the sensing circuitry.

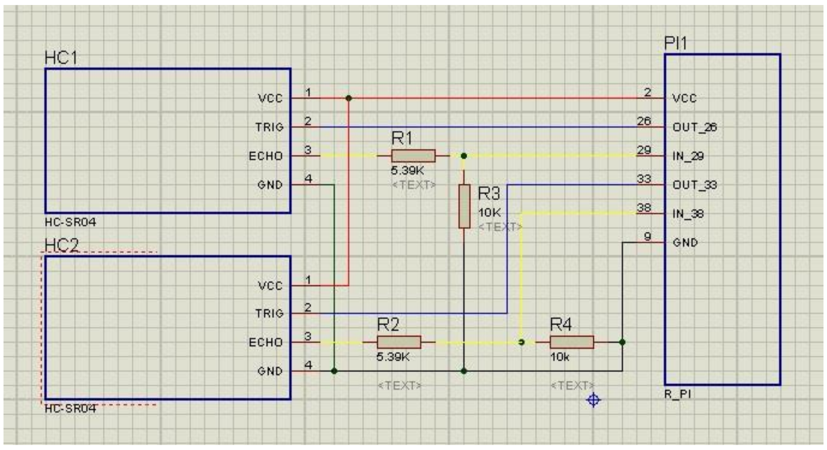


FIGURE 3.7.*SENSING UNIT IMPLEMENTATION*

The software program goes as follow: whenever sensing is needed the TRIG pin is set high for at least 10µs and set low again. Now that our pulse signal is sent, we need to listen to our input pin, which is connected to ECHO. The sensor sets ECHO high for the amount of time it takes for the pulse to go and come back, so our code needs to measure the amount of time that the ECHO pin stays high. We use the “while” loop to ensure that each signal timestamp is recorded in the correct order.

Our first step must be to record the last low timestamp for ECHO (pulse_start) e.g. just before the return signal is received and the pin goes high. Once a signal is received, the value changes from low (0) to high (1), and remains high for the duration of the ECHO pulse. Therefore we also need the last high timestamp for ECHO (pulse_end).

We can now calculate the difference between the two-recorded timestamps, and hence the duration of pulse (pulse_duration).With the time it takes for the signal to travel to an object and back again, we can calculate the distance using the following formula.

Pulse_duration = pulse_end – pulse_start

Distance = pulse_duration x 17150 (speed of sound = 34300cm/s)

3.3.3 USB CAMERA

The robots needs to take pictures of the surrounding environment at specific angles in order to determine the map's wall colors and be able to locate itself and navigate inside the map. For this purpose, a camera mounted on a stepper motor is used.

The camera is a simple USB webcam that can be plugged through the PI's USB port. The stepper motor is a 28BYJ-48 motor. It is a unipolar motor best driven with a ULN2003A driver. The ULN2003A is an array of seven NPN Darlington transistors capable of 500mA, 50V output. It features common-cathode flyback diodes for switching inductive loads [11][12]. Figure.3.8 shows how the stepper mottor is interfaced with the driver and the PI.

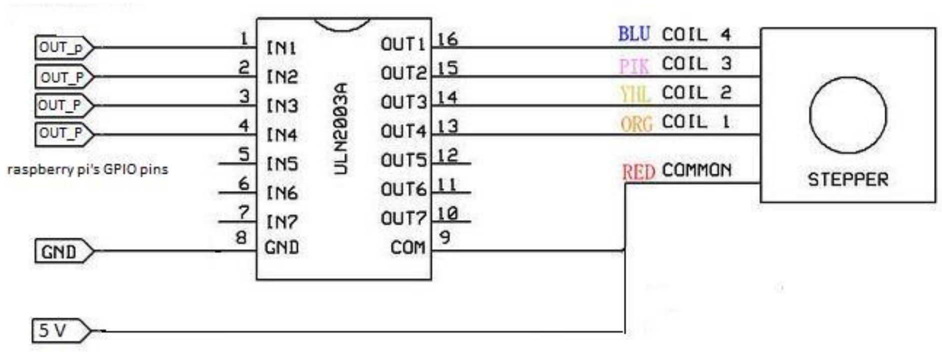


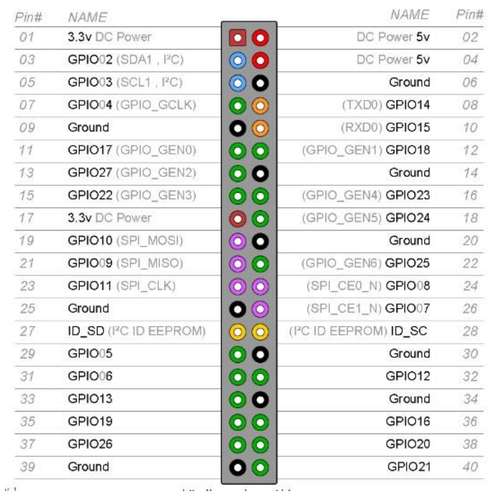
FIGURE 3.8 .STEPPER MOTOR INTERFACED WITH THE PI

In order to rotate the motor, all we need is to go through the eight steps of half stepping, send one sequence at a time to the output pins, and repeat this in a while loop a specific amount of times depending on the angle of rotation we need.

3.4 CONTROL AND PROCESSING UNIT

This project has been built around the micro-computer, Raspberry PI 2 modal B. It is a credit card sized computer that allows processing and hardware control through its GPIO pins.

GPIO stands for general purpose input/Output. These pins are the physical interface between the PI and the outside world. Out of the 40 pins that the PI 2 has, only 26 are GPIO pins and the others are power or grounds. Figure 3.9 shows the pins' distribution.



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I^2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I^2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I^2C ID EEPROM)	(I^2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

FIGURE 3.9.*PI'S PIN DISTRIBUTION*

We can thus program the GPIO pins to be inputs or outputs and assert them to a high value or a low one. High value means 3.3V, any value bigger than this will damage the pin, and low means 0V.

Aside from the pins that allow connection to the outside world, the PI is a full computer running the Raspbian operating system (a Debian distribution specially made for the PI).In general the PI will have to accomplish the following task: when powered on the PI will start in an idle mode waiting for an order. The order will be emitted by the user, as a voice command. YANA client [35] on the desktop computer will receive the voice command, use windows voice to text library to convert it to text and process it into JavaScript Object Notation (JSON) format. Yana client will then send the JSON command through LAN, to YANA server (installed on the PI), YANA server will process the command and assert the PI to proceed with the corresponding action. In our case, execute a system command that will start the robot main script.

The PI will then execute the main Java program, reading sensors and taking images. Received data is processed (distance will be compared with a default value, and image will be processed to check its color) and actions are asserted accordingly (motors start or stop, move forward or backward, left or right rotation). In addition, the PI will send the results to the display interface on the user computer via LAN.

3.4.1 CONTROL UNIT DESIGN SCHEMATICS

The control unit working principle can thus be summarized as shown in figure 3.10

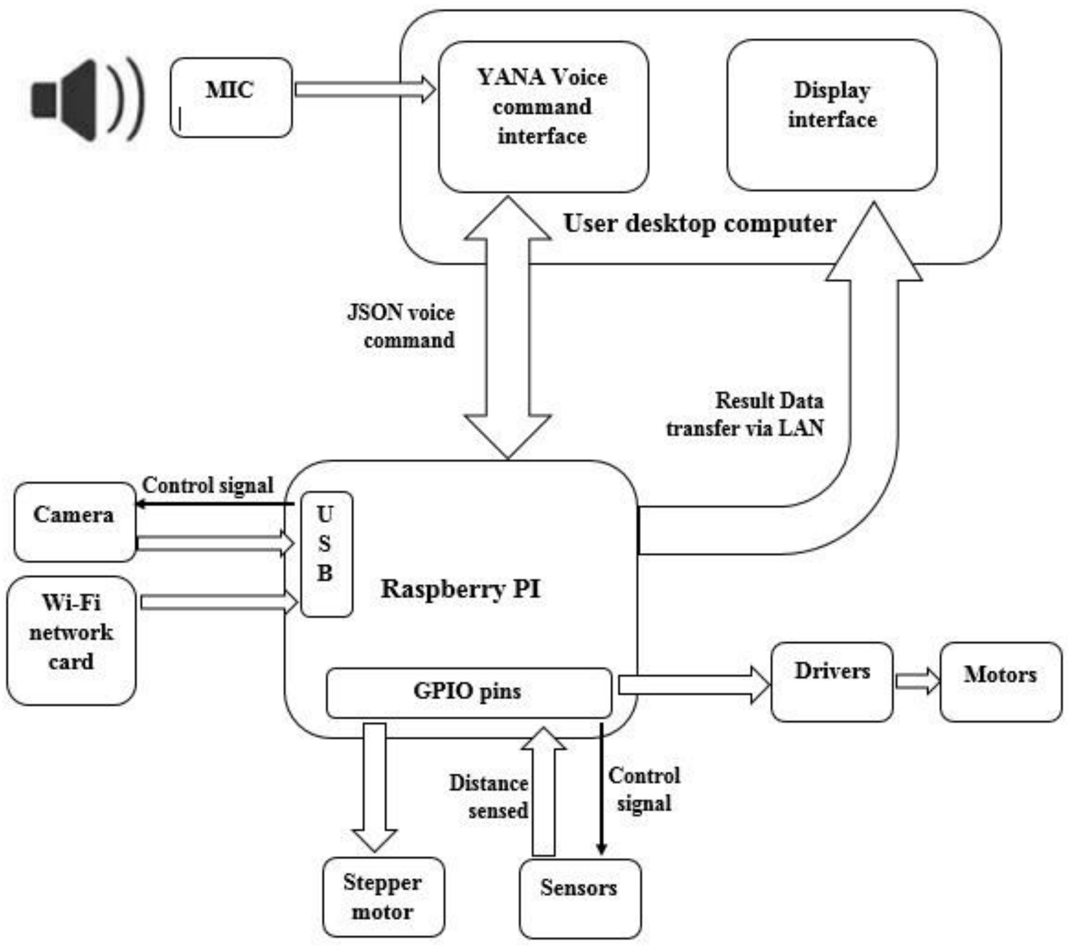


FIGURE 3.10.*CONTROL UNIT SYSTEM*

3.4.2 VOICE COMMAND PRINCIPLES

The voice command system used is based on the YANA domestic automation, a web interface with a client created by idleman [35]. Yana-server is used in smart house applications. It is a home server for Raspberry PI that includes a voice-controlled assistance called Yana. The voice-controlled assistance needs a client (Windows or Android). .

The system functions as a nucleus (the server) that can accept an infinite number of plugins.

These plugins can in turn be queried by multiple interfaces. Figure 3.11 illustrates the system’s working principle.

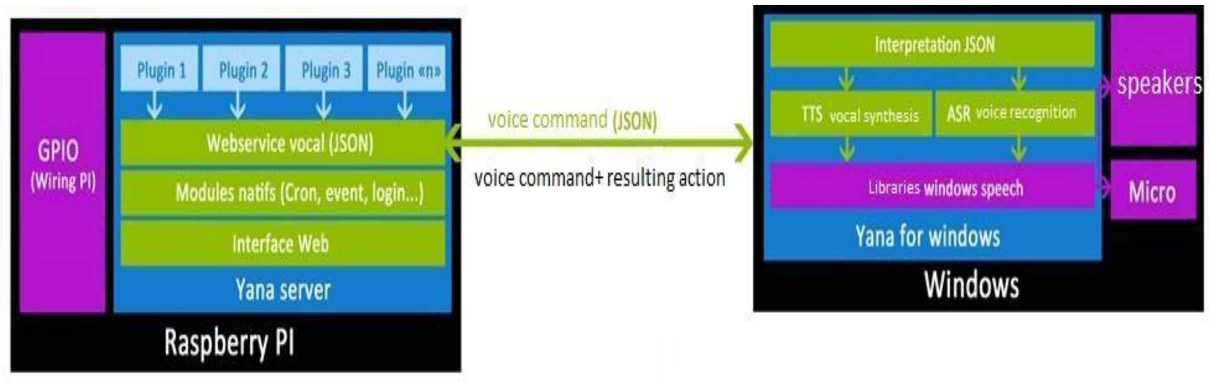


FIGURE 3.11.*YANA VOICE COMMAND PRINCIPLE [35]*

4. SOFTWARE DESIGN

4.1 INTRODUCTION

In this project, we created a simulation software to provide an error free environment in which our algorithms can be tested. Another software is created for the robot. This chapter covers the software design of the system. It introduces the algorithms and design patterns used in both simulation and hardware.

The first part presents the design patterns used in the simulation and hardware system. In the second one we set the ground by defining the used data structures and the computational representations of the problem. The third part gives an overall view of the software implementation using the Unified Modeling Language (UML) Class Diagram. The fourth part introduces the algorithms used in the software with detailed explanations, while the last one explains error handling.

4.2 DESIGN PATTERNS

In order to implement the desktop application used for simulation, two different design patterns have been used: The Model Controller View (MCV) [14] used to separate different parts of the application and the Observer design pattern [15] used to insure proper communication between the Model and the View by using interfaces.

Concerning the robot system software, the Observer design pattern is also used to insure communication between different parts of the program. The Model and Controller of the MCV packages design pattern are kept with small changes, and the View is replaced with Actuator-Sensor classes.

4.2.1 THE MODEL CONTROLLER VIEW DESIGN PATTERN

Any desktop application can be divided into three components: The View, which consists of the graphical interface that end users are going to interact with; the Controller, which is responsible of checking the input from the user, and the Model that does all the processing and decisions.

The only component that is visible for the user is the View. The user interacts with the graphical interface and the inputs (clicks, keyboard events) are passed to the Controller. This latter will check the correctness of the inputs. If these inputs are valid the model is asked to perform some action accordingly; otherwise, an exception is raised. When the model receives a task from the controller, a set of instructions is executed according to the task. Once the task is terminated, the Model informs the View, and if the GUI is to be updated, the changes are communicated. The View needs to have access to the state of the model so that it can be updated correctly. Figure 4.1 illustrates how the Design pattern functions.

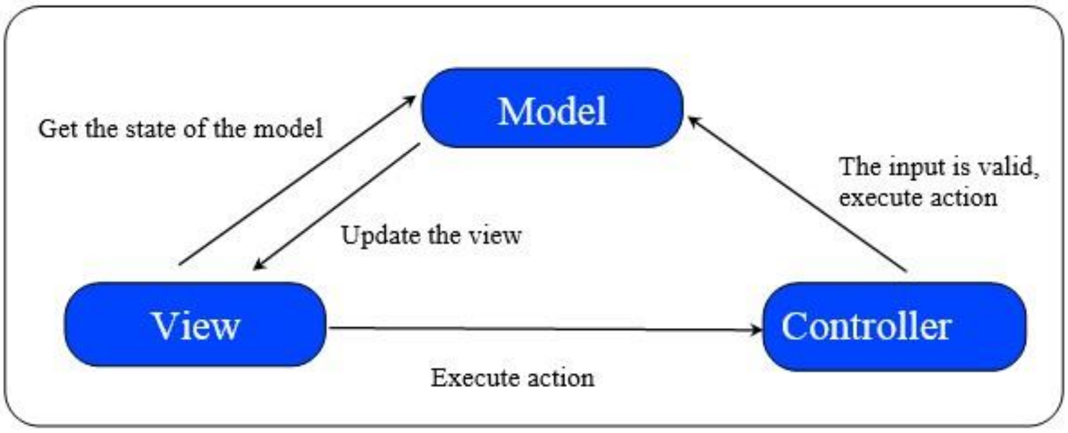

Model
Get the state of the model
Update the view
The input is valid, execute action
View
Controller
Execute action

FIGURE 4.1. Communication between different components of the MVC design pattern

4.2.2 THE OBSERVER DESIGN PATTERN

Figure 4.1 Shows that there is a bidirectional communication between the view and the model. The communication from the View to Model can be insured by using the controller as a channel. However, communication in the other direction cannot be achieved using the same technique. The solution to this problem is to use the Observer design pattern.

The Observer design pattern consists of using two interfaces: Observable, and Observer. Each Observable Object has a set of Observer objects. Whenever a change occurs in the Observable, it notifies all the Observers. These Observers are then updated according to the received notification.

The Model implements the Observable interface, the View implements the Observer interface. Whenever the change occurs in the Model, it notifies the View, this latter will be updated. Figure 4.2 shows how this design pattern works.

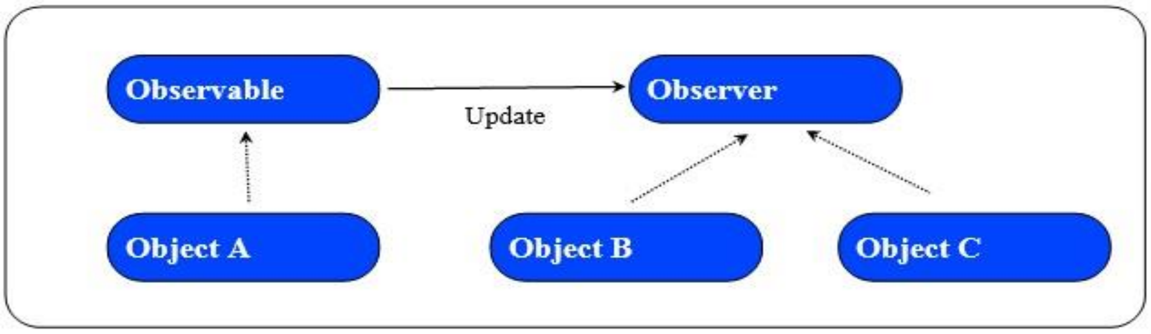


FIGURE 4.2. COMMUNICATION BETWEEN DIFFERENT COMPONENTS OF THE OBSERVER DESIGN PATTERN

4.2.3 PROBLEM PRESENTATION

This section depicts the representations and data structures used to picture the problem in the software. It explains mainly how the maze, graphs, robot and cells are represented.

4.2.3.1 THE MAZE

The robot will navigate through a maze similar to the one shown in figure 4.4. The maze has a width M and a height N. It is divided into N×M cells. Each cell has at most four walls (north, west, south and east) of different colors (Red, Green or Blue). It also has a specific position (x,y). The UML class diagram of the Cell is shown in figure 4.3.

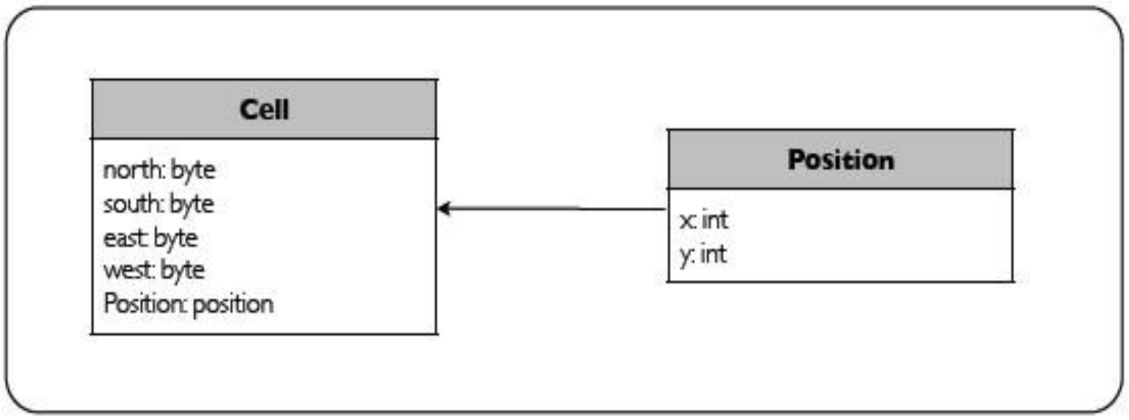


FIGURE 4.3. *UML CLASS OF CELL AND POSITION*

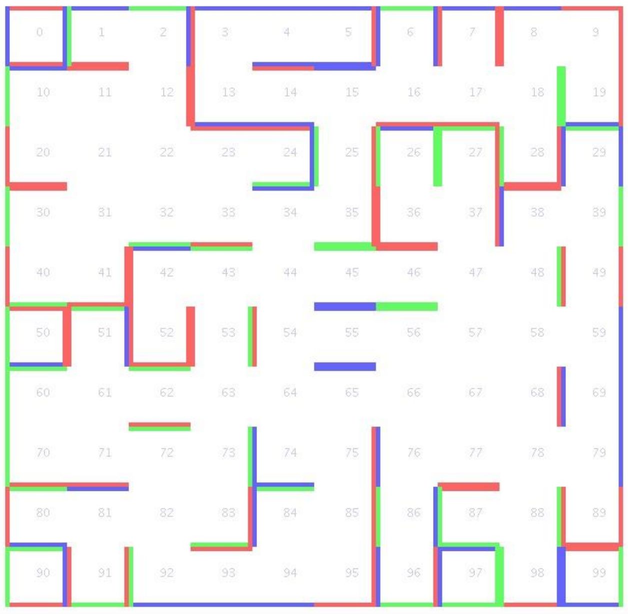


FIGURE 4.4. *EXAMPLE OF A MAZA MAP*

The maze can thus be represented by a collection of cells.

4.2.3.2 THE GRAPH

In order to do operations in this environment such as map search, auto-location and calculating optimal path, a simpler representation is needed. The graph representation has been chosen because most of the problems cited above would be much easier to solve in such a representation.

Each cell of the map will be represented as a node, making an undirected graph (figure 4.5) with N×M node. Each node have N×M-1 possible vertices. Making a total of $(N \times M - 1) \times (N \times M)/2$ maximum number of edges. The memory cost of this implementation is $((N \times M)^2)$. However, it can be reduced to $(N \times M)$.

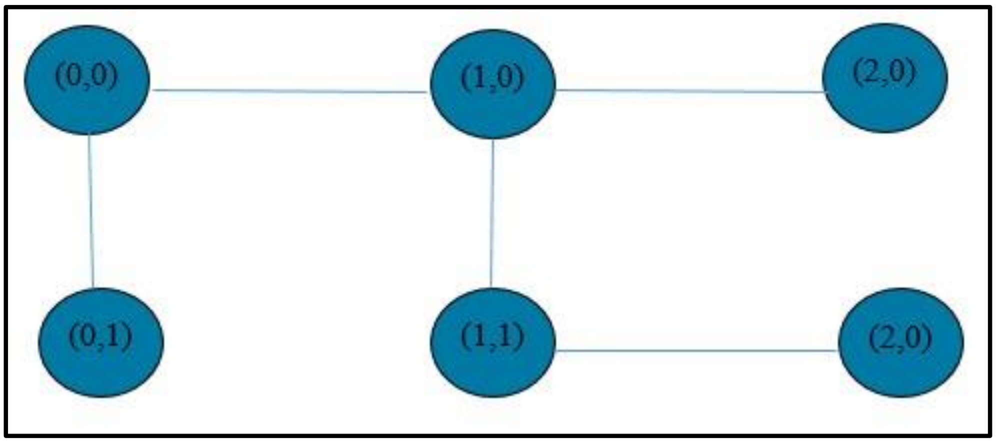


FIGURE 4.5. *THE UNDIRECTED GRAPH REPRESENTATION OF THE MAP*

4.2.3.3 REDUCING GRAPHAS MEMORY COST

The memory cost can be reduced by noticing that each cell cannot have more than four adjacent cells which are the ones situated to its north, south, east and west. All the other cells cannot be adjacent to it. The number of possible edges can be reduced to 2*N*M-(N+M).

This representation can be implemented using an (M×N) ×4 matrix. However, the nodes cannot be added randomly. We have to add them in such a way that the four possible adjacent cells can be found easily.

One way to do that, is adding them according to their coordinates, where the cell in position (x,y) will be the position y×M+x in the array. It is easy to recover the possibly adjacent cells as follows:

- **North** has coordinates (y-1)×M+x= y×M+x-1 = p-M. The northern cell has the position p-M in the array.
- **South** has coordinates (y+1)×M+x= y×M+x+M= p+M. The southern cell has the position p+M in the array.
- **West** has coordinates y×M+x-1= p-1. The western cell has the position p-1.
- **East** it has coordinates y×M+X+1= p+1. The eastern cell has position p+1.

It is no longer needed to keep information about where is the northern, southern, eastern or western cell, since they can be calculated from the current position. The matrix will be of the form shown in table 4.1, where X can take two values namely 1 for adjacent and infinity for non-adjacent.

Cell number	North	South	East	West
0	X	X	X	X
1	X	X	X	X
...	......	.....	.....	...
(N×M)-1	X	X	X	X

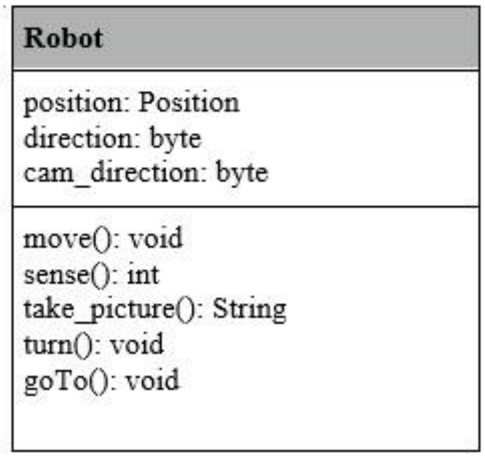
TABLE 4.1. *MATRIX REPRESENTATION OF ADJACENT CELLS*

4.2.3.4 ROBOT

In this project, the cell sizes were taken in such a way that the robot can be considered as a point object. The robot is represented by the position of its center, its direction, and the direction of its camera as shown in figure 4.7. Figure 4.6 shows the graphical representation of the robot. In order to be able to represent the robot by a point object, the following condition should be satisfied.

Max(robot_width, robot_hight) < = Min(cell_width, cell_hight).

In other words, the robot should fit into one cell regardless of its orientation. In this experimentation, the robot size is 15cm X 22.5cm and the cell size is 30cm X 30.cm The condition is satisfied since 22.5cm <30cm. The 7.5cm difference will be taken in case of robot displacement errors. Since the robot is not provided with a feedback control system, the error is handled in the algorithm. For the sake of experimental restriction, the robot is only allowed to move horizontally or vertically making turns of 90 degrees.



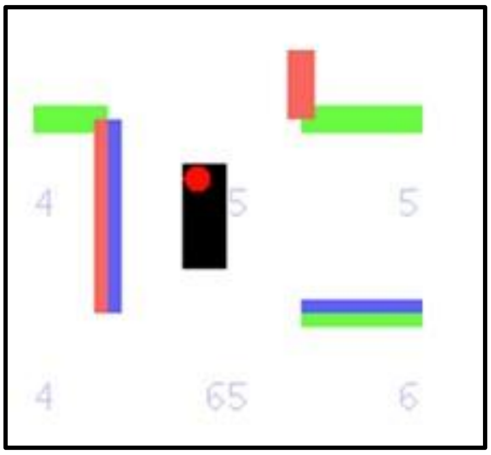


FIGURE 4.7. *ROBOT UML CLASS DIAGRAM* ***FIGURE 4.6.*** *ROBOT'S GRAPHICAL REPRESENTATION*

4.3 UML CLASS DIAGRAM

In this section the UML Class Diagram [16] is introduced. Details such as irrelevant fields and methods have been removed to insure diagram simplicity and readability. The Graphical User Interface (GUI) implementation details have been removed as well.

4.3.1 UML DEFINITION

The Unified Modeling Language (UML) is a general-purpose visual modeling language used to specify, visualize, construct, and document the artifacts of a software system [38]. UML class diagram has been chosen in order to illustrate the overall structure of the software because it is the most used modeling language, and offers many advantages over flow charts. When a complex software is developed using an Object Oriented approach, understanding the structure becomes of great importance. Although flow charts are suitable to describe the behavior of the software, they fail to describe its structure. The following class diagram are drawn using ArgoUML freeware [36].

4.3.2 SIMULATION CLASS DIAGRAM

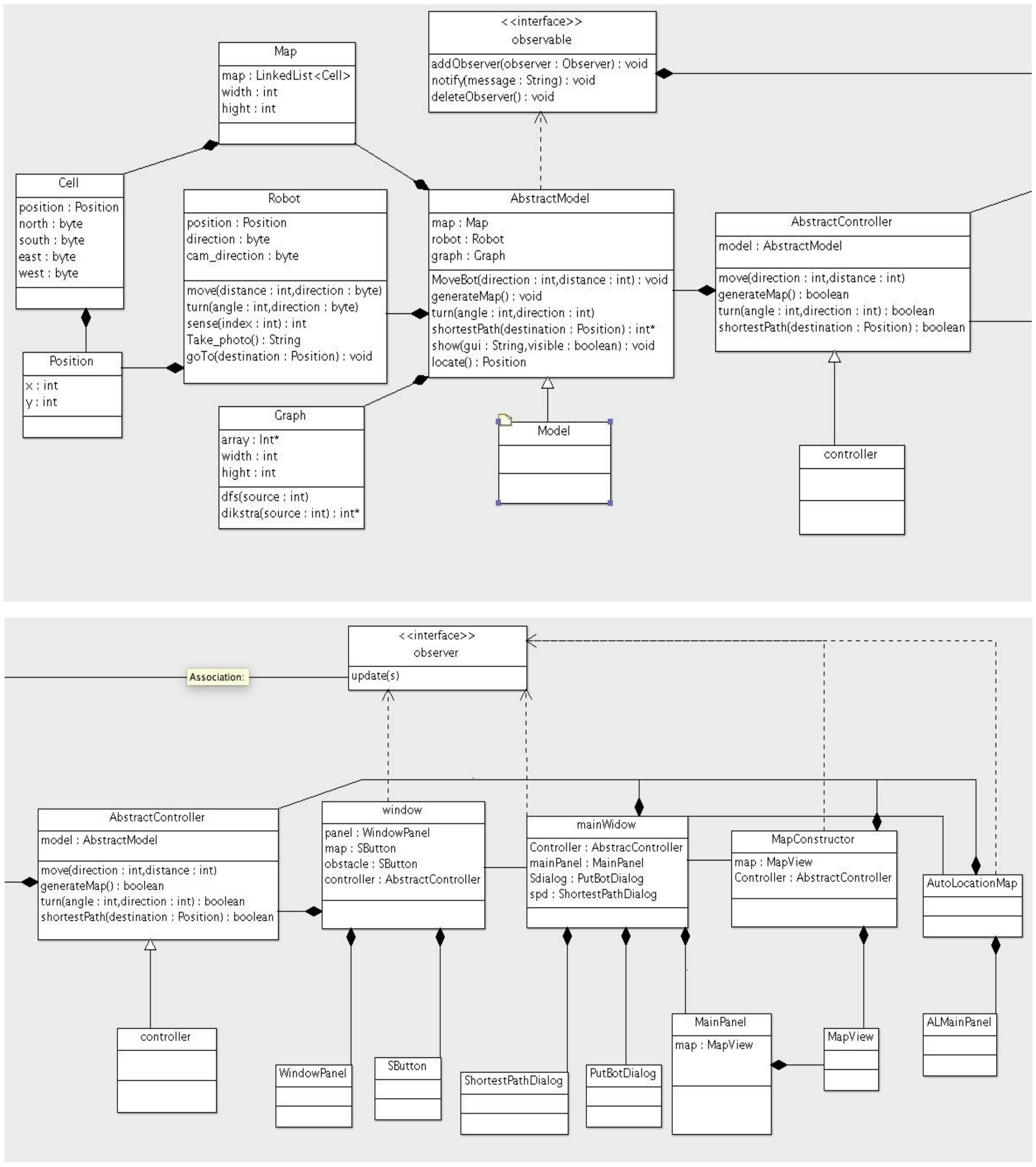
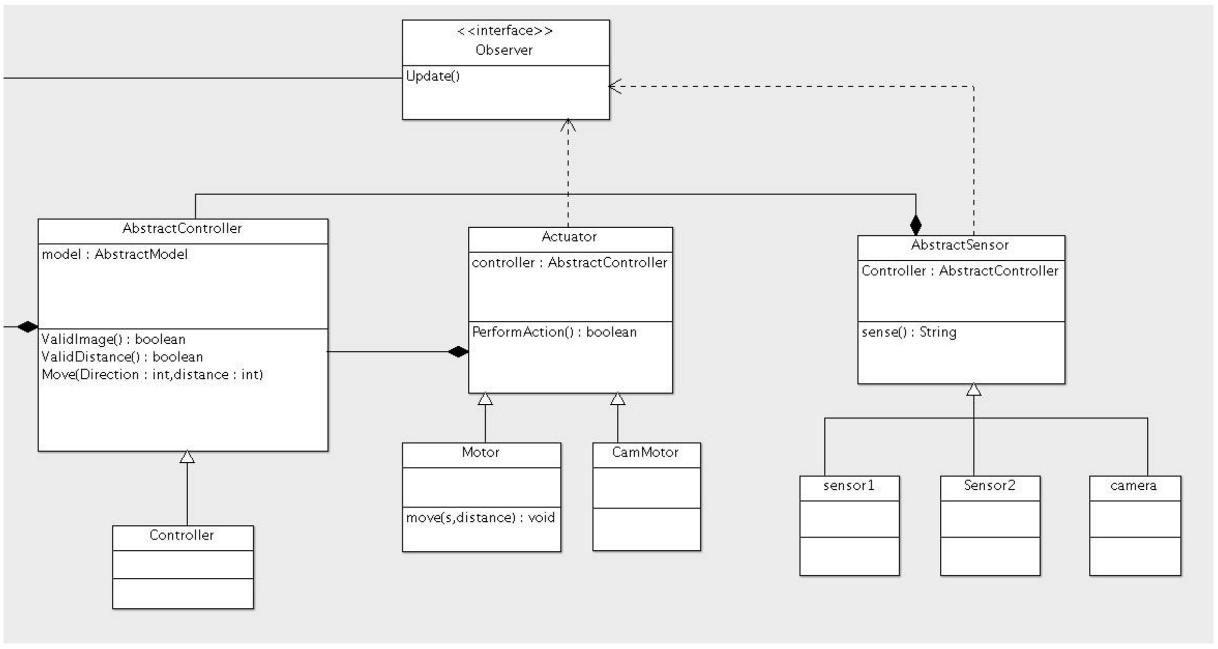


FIGURE 4.8.A *SOFTWARE SIMULATION'S UML DIAGRAM*

4.3.3 ROBOT CLASS DIAGRAM



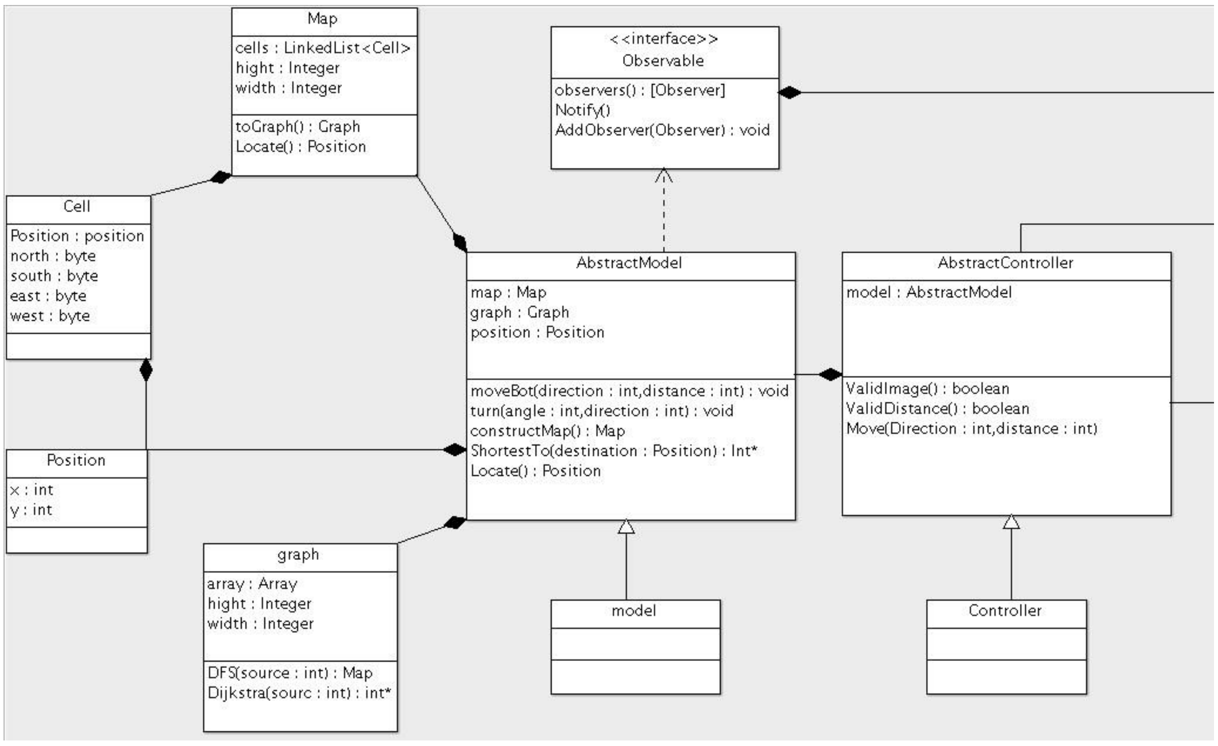


***FIGURE 4.8.B** UML CLASS DIAGRAM OF THE ROBOT SYSTEM SOFTWARE*

4.4 ROBOT ALGORITHM

This section presents the algorithms used in the simulation software. Most of these algorithms are common between the simulation and the robot software. The exceptions are the algorithms used to generate a virtual environment that are used solely in the simulation and the error handling which are used exclusively in the robot software.

4.4.1 MAP GENERATION

The aim of this algorithm is to generate a virtual equivalent of the maze. The actual maze has the following constraints:

- The maze is divided into N×M cells where M is the width of the maze and N is the height.
- All the boundaries of the maze are closed.
- If cell A is not adjacent to cell B, then cell B is not adjacent to cell A.
- Each cell has four possible walls. The walls can be either red, green or blue.

In order to generate an N×M map, N×M cells are randomly generated. The function GenerateCell () of Cell object insures that if the cell is located on the outer border of the maze, this border is a solid wall. Otherwise, the walls are generated randomly with a given probability for each wall.

red wall: 10%   blue wall: 10%   green wall: 10%   no wall: 70%

The probability of no wall is high to insure the connectivity between different cells. The generated map does not respect the third constraint because the cell are generated independently. In order to make sure that this constraint is respected we iterate through the cells and for each cell in position (x, y), the northern and eastern cells are checked. If these are adjacent to the current cell, the current cell is made adjacent to them.

4.4.2 GRAPH GENERATION

The graph representation has been introduced in section 4.2. To transform this map into a graph represented by an adjacency matrix, cells are added according to their position. For each cell, the four directions are checked. If there is a wall in that direction, the cell in the same direction is not adjacent, otherwise, it is. Figure 4.9 shows a 2x2 map, while figure 4.10 shows its corresponding graph.

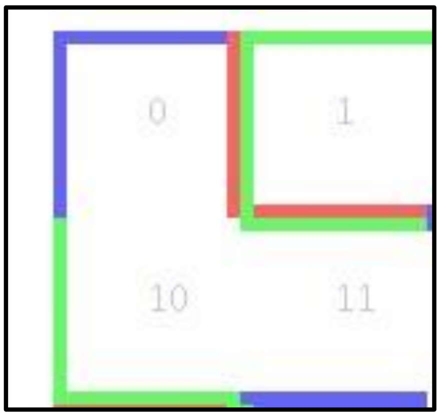


***FIGURE 4.10.** SAMPLE MAP*

Cell	north	west	south	east
0	NA	NA	A	NA
1	NA	NA	NA	A
10	A	NA	NA	A
11	NA	A	NA	A

***FIGURE 4.9.** CORRESPONDING ADJACENCY MATRIX REPRESENTATION*

4.4.3 AUTO LOCATION

The auto location algorithm works by an elimination process. To locate itself in the maze, the robot starts by assuming that the probability for it to be in any cell (x,y) is the same as in cell (x’,y’) since it has no information yet. This is the maximum confusion state. Afterwards, it starts observing the environment, and eliminating the positions that do not correspond to the observation.

If the robot is put in the 2x2 sample map of figure 4.9, it starts initially by assuming that it has equal probabilities to be in each one of the four cells.

Anterior probability= (0.25, 0.25, 0.25, 0.25).

The robot makes an observation to the north and sees a green wall. The cells 0 and 10 do not correspond to this observation, so their probability is dropped to 0, and cells 1 and 11 correspond to the observation, so the probability of the robot being there is bigger.

Observation (0, 1, 0, 1).

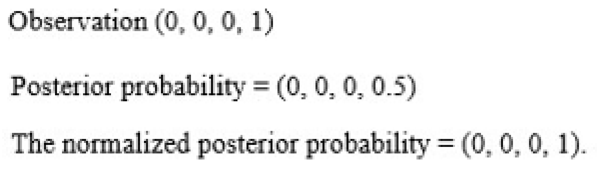
Each element of the anterior probability is multiplied by the corresponding element of the observation.

Posterior probability (0, 0.25, 0, 0.25).

The probability is then normalized.

Posterior probability = (0, 0.5, 0, 0.5)

The robot makes another observation to the west, and sees no wall.



Observation (0, 0, 0, 1)

Posterior probability = (0, 0, 0, 0.5)

The normalized posterior probability = (0, 0, 0, 1).

The algorithm has then the following form:

1. Initialize prior probability (1/L, 1/L .... 1/L)   L= N×M.
2. While position not found
3. Make observation
4. Multiply prior probability by observation
5. normalize the product to get Posterior probability
6. Prior probability = posterior probability.

4.4.4 MAP CONTRUCTION

The Map construction algorithm is used to construct a virtual map of the external environment (whether physical or virtual).The Depth First Search is used to explore the map [17].

4.4.5 SHORTEST PATH

This Shortest path function is used to find the optimal path to go to a given destination. The problem is addressed as a single source multiple destination graph problem, and the Dijkstra Algorithm [18] is used to solve this problem.

4.4.6 MAP SEARCH

Map search is used to look for an object in a map. The object can be anywhere in the maze, and since the map can be represented by a graph, the problem becomes simply a graph search problem. The DFS algorithm is used again for this problem.

4.4.7 MAP INSPECTION

Map Inspection consists of walking through the graph, by visiting each node at least once, and taking pictures in the four directions. If any change occurred in the graph, the robot will detect it.

Map inspection is a task that is repeated many times, reducing its cost is of great importance. The problem can be seen as Traveling Salesman problem (TSP) [19]. However in the graph, some nodes need to be visited more than once, and in most cases a Hamiltonian tour [20] does not exist. We assume that the graph is connected.

To overcome this problem, a complete Graph (each node in the graph is connected to all the other nodes) is constructed from the maze graph by adding edges between the nodes that are not adjacent initially with weights that are equal to the shortest path cost between these nodes. The Algorithm below explains the process.

1. Run the Floyd-Warshall [21] algorithm on the initial graph.
2. Start with cell k= 0.
3. Go to cell h=k+1.
4. Create an edge between cells k and h with the weight equal to the shortest paths cost between k and h (calculated with the Floyd-Warshall algorithm).
5. increment h
6. If h <= N×M go to step 4.
7. Increment k.
8. If k < N×M go to step 3.

Figure 4.11 shows a simple example where only two edges are added.

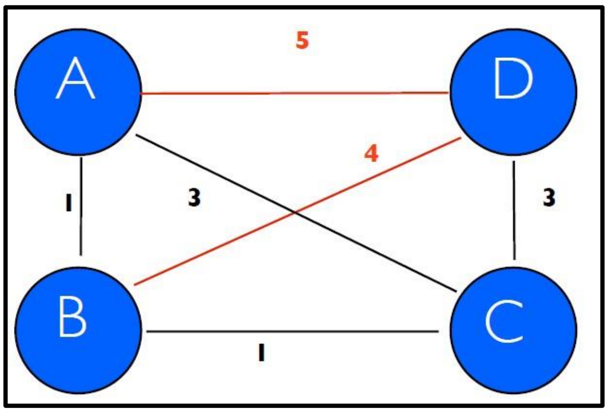


FIGURE 4.11. *MAKING A COMPLETE GRAPH BY ADDING EDGES BETWEEN NON ADJACENT CELLS*

4.5 ERROR HANDLING

Strict error handling is not necessary in the simulation software, since the inputs are known precisely and the environment is error-free. However, when implementing the robot software, it is possible that errors occur. The sensors may not respond, the camera may be unable to create a file, the robot may travel 8 cm instead of 10 cm if the battery is low. To handle such errors, a package containing the possible errors that might rise has been created.

Only the most relevant errors will be handled in this section.

4.5.1 RECOVERING THE DISTANCE ERROR

When the robot moves from a cell to cell, an error may occur resulting in the robot traveling a smaller distance. The error is small enough to be neglected for a single displacement, however it either cancels (moving in opposite directions) or cumulate (if moving in the same direction) when multiple displacements are performed as shown in figure 5.3. If the cumulative error is bigger than the cell margin error, the robot can hit an obstacle. As we can see in the figure, the blue arrow is the desired displacement, the green one is the actual displacement, and the red one is the error.

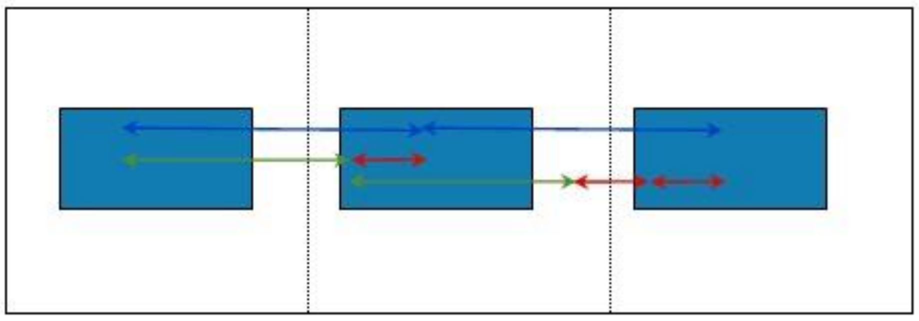


FIGURE 4.12. *THE CUMMULATIVE ERROR IN ROBOT DISPLACEMENT*

This error is handled in the Motor wrapping class. The algorithm is presented below:

- Move the robot distance X, in direction D.
- Check if there is a wall in the direction D. If there is no wall, stop.
- Measure the distance (d) to the wall in direction D.
- If (d – M) > e

   d is the measured distance in step 3.

- M is the cell margin error
- e is the maximum accepted error

- Move the robot in (d-M) in direction D, and go to step 4

The algorithm requires the presence of a wall in the next cell in the same travelled direction. If there is no wall, the error will not be corrected. If the robot moves in one direction, the error will cumulate until a wall is met in that direction.

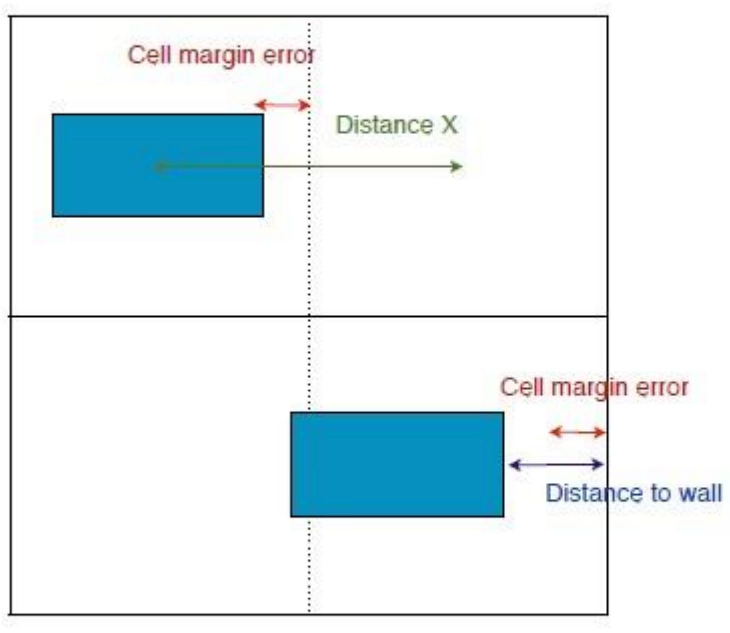


FIGURE 4.13 *ERROR CORRECTING ALGORITHM*

4.5.2 ROTATION ERROR CORRECTION

When the robot is asked to rotate n degrees, it may rotate (n-e) degrees. e is the rotation error. If the robot is rotating in one direction, the error will cumulate, if it rotates in opposite directions, the error will cancel. Correcting this rotation error is not as straightforward as in the distance error. In order to do that, image processing will be used. The algorithm below explains the procedure of correcting the error. Figure 5.5 illustrates the algorithm.

- Rotate to direction D
- Take an image of the wall in direction B. If there is no wall, stop, if there is a wall, determine the color C of the wall.
- Calculate X the center of the pixels having color C, on the central raw.

- If (image_width/2 - e < x < image_width /2 + e), stop If (X > image_width/2 + e) rotate counter clockwise.
    - e is half the maximum accepted error.
- If (X<I mage_width /2 - e) rotate clockwise Repeat step 5.

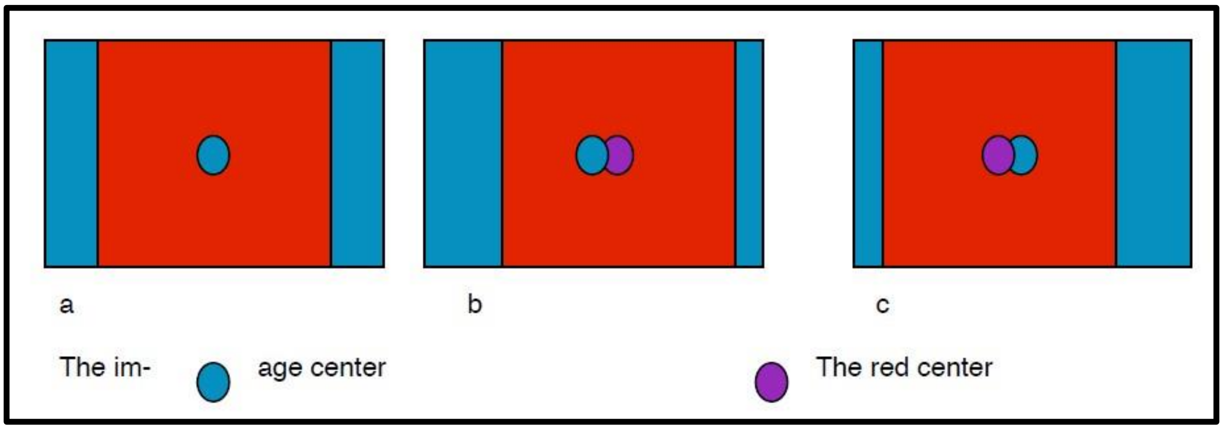


FIGURE 4.14 *.ROTATE ERROR CORRECTION ALGORITHM -A- NO ERROR -B- ROTATE COUNTER CLOCKWISE FOR CORRECTION -C- ROTATE CLOCKWISE FOR CORRECTION*

5. SOFTWARE IMPLEMENTATION

5.1 INTRODUCTION

This chapter covers the software implementation of the robot system and the simulation results. The programs are implemented using Java and Python languages. Java has been chosen because it is a familiar language and makes desktop applications' implementation with Graphical User interface simple. Python has been chosen because it makes the handling of the PI IO signals simple and is also its native programming language.

The software is downloaded and installed on the Raspberry PI memory in order to be executed when the robot is turned on.

The first part covers the Python functions that control the robot movements and sensors. The second one introduces the wrapping classes which constitute the interface between the Python codes and the main program, while the last section deals with simulation software results.

5.2 PYTHON FUNCTIONS

In order to perform I/O control operation, Python code have been used since it is the native programming language on Raspbian OS. Java could be used by installing a library called "Pi4J" [22] However using python code was preferred because of memory limitation.
There are five basic Python functions:

1. MoveRobot (Distance: Int, direction: Int): void
2. TurnRobot(angle: int , direction: int): void
3. TurnCamera(angle: int, direction: int): void
4. TakePicture(): String
5. Sense(SensorID: int): int

5.2.1 MOVEBOT FUNCTION

The function is given two parameters; movement's distance which is an integer, and the direction which is either backward or forward. After reading the parameters, the function sets the correct I/O signals that are sent to the motor drivers.

The motors have no controller to correct errors thus, this function does not handle error recovery. For example, if the robot is asked to move 10 cm and it moves 8 cm, no correction will be made by the function; the correction will be handled in the wrapping class.

5.2.2 THE TURN FUNCTION

This function is given two parameters: the angle of rotation, which is an integer, and a direction which is either clockwise, or counterclockwise. Since the robot moves only vertically or horizontally, the angles are given as multiples of 90°.

Once the inputs are read, the function sets the appropriate I/O signals and sends them to DC motors drivers. This function is not too accurate, and does not handle error correction either.

5.2.3 THE TURNCAM FUNCTION

This function is given two parameters: the angle of rotation, which is an integer, and a direction which is either clockwise, or counterclockwise and turns the camera accordingly. Since the maze is represented as a collection of square cells, with each cell having four walls, the angles are given as multiples of 90°. Once the inputs are read, the function sets the appropriate I/O signals and sends them to stepper motor driver.

5.2.4 TAKEPICTURE FUNCTION

This function sends an I/O signal to the USB camera. This latter takes a picture and saves it in a file named [X]_[Y]_[DIRECTION], where x and y are the coordinates of the robot where the picture is taken, and DIRECTION is the direction to which the camera is pointed. The name of the file is returned in a String form. The file is then read by the read () function which uses BufferedImage [23] and ImageIO [24]. The image is processed in a matrix form to determine the color of the obstacle.

5.2.5 THE SENSE FUNCTION

This function takes as the input the ID number of the ultrasonic sensor (0 for the front sensor, and 1 for the back sensor). The I/O signals are sent to the corresponding sensor. The sensor makes the needed measurement and displays the distance of the nearest obstacle. An InputStreamReader [26] is used to get the distance from the Python process.

5.3 WRAPPING CLASSES

The wrapping classes are the interface of communication between the Java main program and the Python functions i.e. they wrap the Python code. There are two actuator classes, one is used for the four DC motors, and the other is used for the stepper motor. Both of these classes inherit from the AbstractActuator class. There are two sensor classes as well, one is used for the camera, and the other is used for the ultra-sonic sensors. These two classes inherit from AbstractSensor class.

To run the Python code, the functions are written into main files, each file has the function name. These files are gathered in a folder named functions. These functions can be called from the wrapping classes by using the RunTime [26] class from Java.

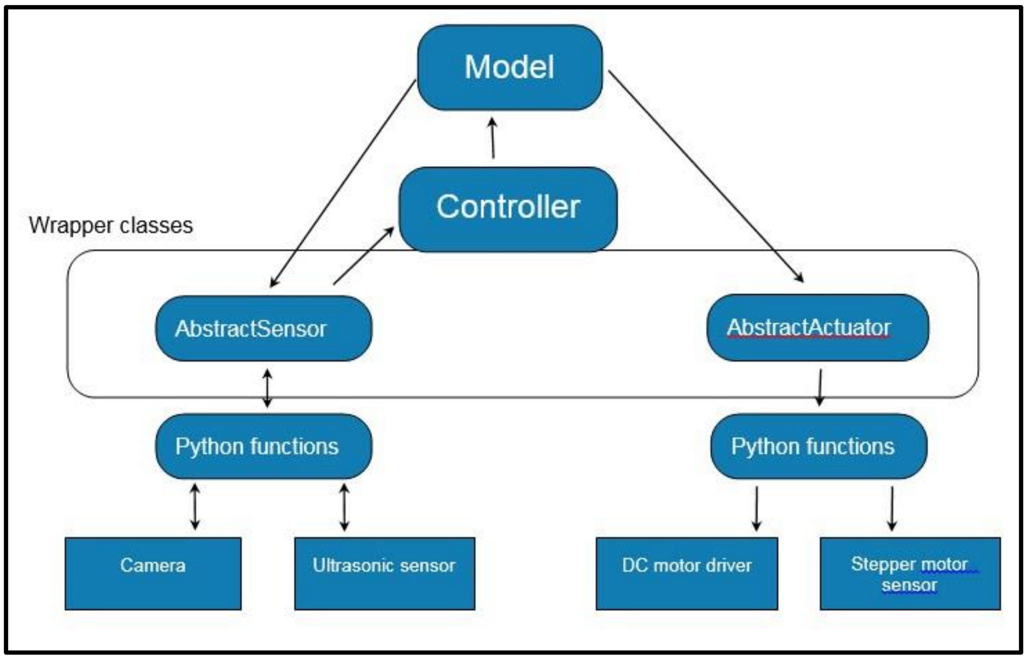


***FIGURE 5.1.** COMMUNICATION PATTERN BETWEEN THE HARDWARE AND THE SOFTWARE*

5.4 SIMULATION RESULTS

In this section, the simulation results of the algorithms stated in the previous section are shown.

5.4.1 THE SHORTEST PATH SIMULATION

The robot is asked from its current position to go to location 45. The shortest path is shown in figure 5.3.

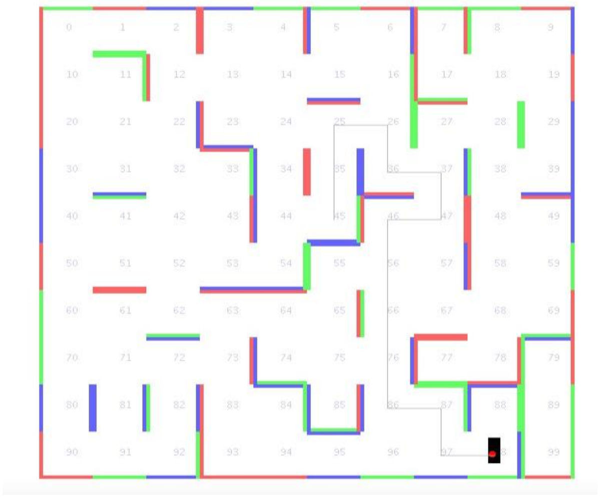


FIGURE 5.2 *SHORTEST PATH TO POSITION 45*

5.4.2 MAP CONSTRUCTION

Figure 4.13, shows the robot while it is exploring the map. The map on the left is the actual map, and the one on the right is the map being constructed by the robot, then the entire map is shown after the robot finishes exploration.

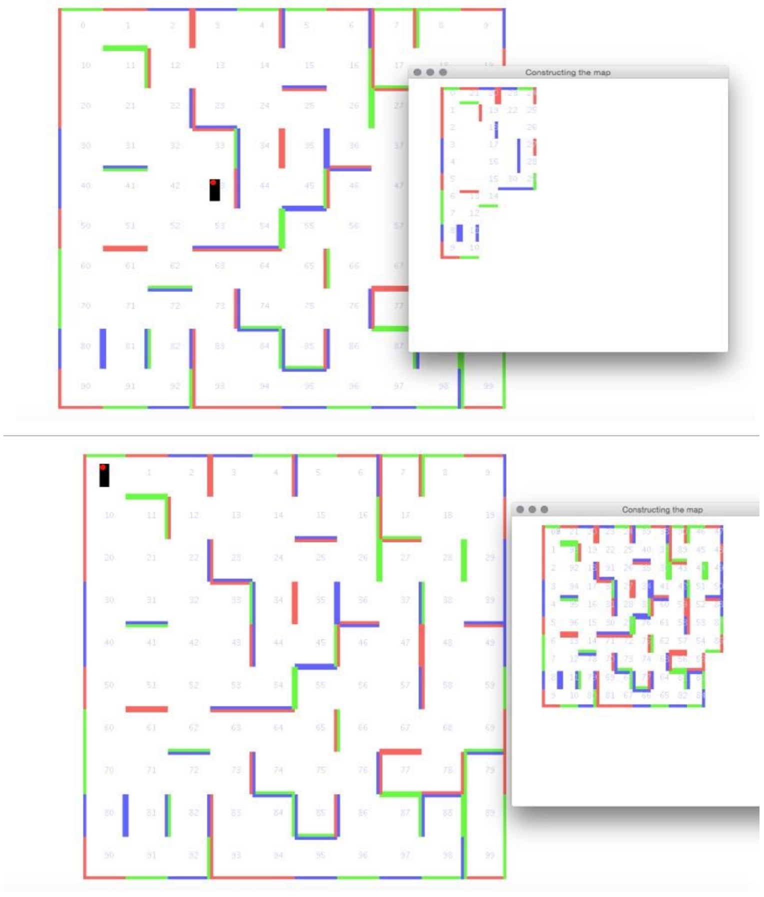


FIGURE 5.3. *MAP CONSTRUCTION INTERFACE*

5.4.3 AUTO LOCATION

Figure 4.14 shows the different steps of auto location process.

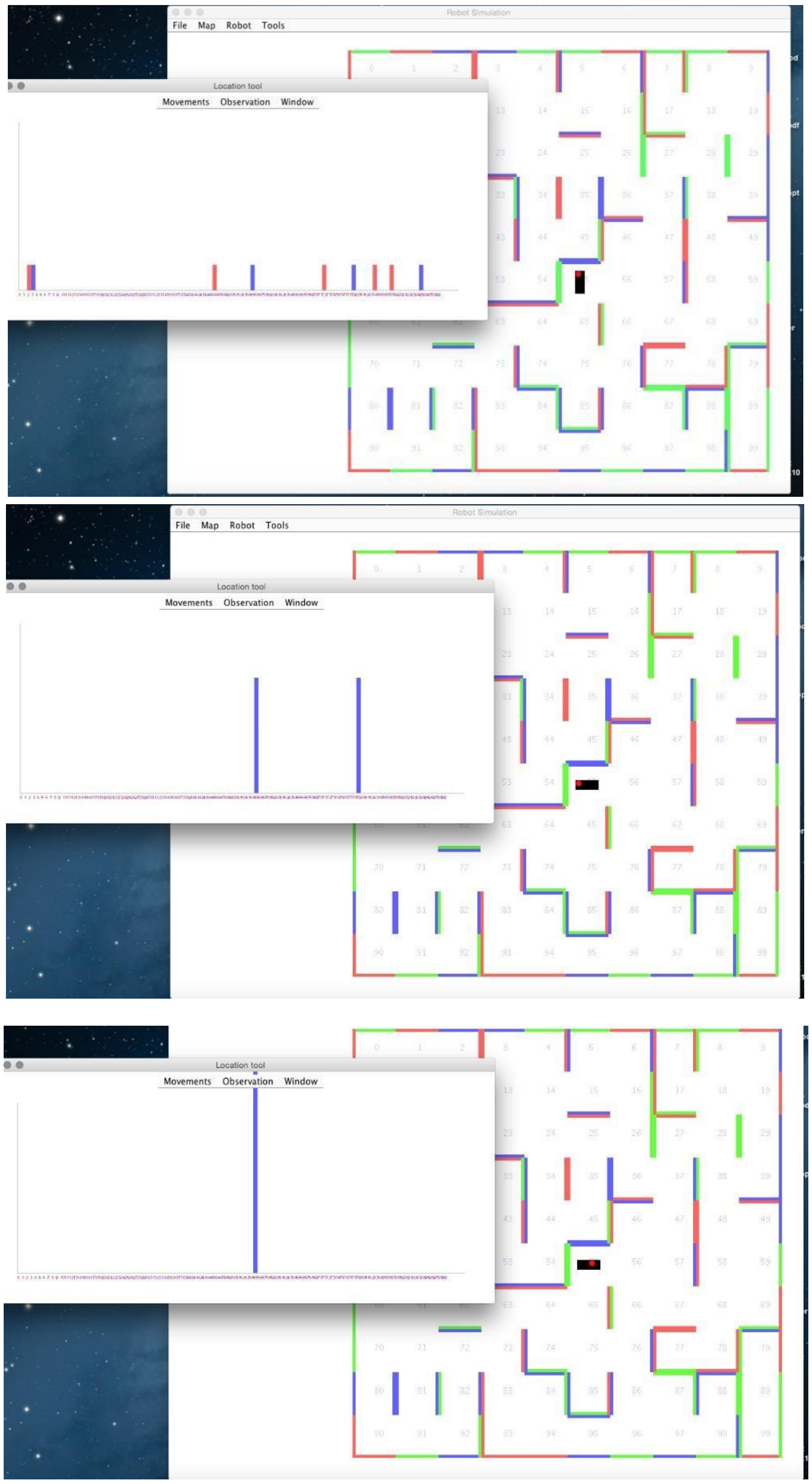


***FIGURE 5.4** AUTO LOCATION PROCESS'S STEPS*

6. CONCLUSION

The main concern of our project was to design an autonomous navigating robot capable of navigating inside an unknown environment and reconstructing its mapping. This work underwent the different design aspects and had a look to the problem from a software and a hardware perspective.

The control system was built around the Raspberry PI microcomputer and the algorithms developed under Java and Python languages. At the end, the robot performs the following tasks: freely navigate inside a maze, reconstruct its 2D mapping, locate itself inside the map, find the shortest path to a given location and undergo map search for a target, or map inspection for potential environment changes.

The project was first tested on a simulation software in an error free environment, then implemented on the hardware design. A great deal of brainstorming and time was spent on debugging recurring hardware problems and software bugs. The components restrictions and low quality gave us a hard time during the implementation process. The first problem we faced concerned the hard drive memory of the PI. As discussed previously, the PI uses a micro SD card as its main storage memory; and since SD cards cannot support extensive reads and writes, the complete system crashed while performing program testing and some GPIO pins got damaged on the recovery process. The second big problem we faced concerned the power supply. The used motor required more current than expected and finding a convenient battery light enough to be put on the robot was a big issue. On the other hand, debugging software took a lot of time. Working on image processing for color and object detection as well as creating the software interface was a challenge by itself. This offered a tremendous learning experience. It cultivated our knowledge about hardware circuitry such as motor drivers, sensors and embedded systems. Moreover, it gave us the opportunity to learn Python as a new programming language and work with the raspbian OS.

Nevertheless, a technical project is never complete. The groundwork has been already laid down, all that is needed is further developments and improvement, such as:

- A better control of the robot motion with the implementation of a feedback control system by extending the raspberry pi's GPIO pins, and adding sensors to the robots right and left sides
- Adding an obstacle avoidance algorithm and optimal path follower. i.e. the robot might be able to avoid any shaped obstacle and join the target point following an optimal path.

- Adding object tracking; since the robot already search for an object using the camera, an algorithm can be implemented to use the camera stream and follow the given object.
- Develop the image processing part of the software and implement 3D mapping of the environment.

We would like to conclude this work with a cote from Albert Einstein “The more I learn, the more I realize how much I don’t know”, a feeling that accompanied us during the whole realization process of this work.

Bibliography

1. "Dongle computer lets kids discover programming on a TV". Bush, Steve. Electronics Weekly. 25 May 2011
2. "BCM2835 Media Processor; Broadcom". Broadcom.com. 1 September 2011.
3. "Raspberry Pi Model A+ 512MB". Farnell 2016-05-04.
4. "Broadcom BCM2835 SoC has the most powerful mobile GPU in the world?" Brose, Moses. Grand MAX. 30 January 2012
5. "Raspberry Pi downloads".
6. "Windows 10 for IoT". Raspberry Pi Foundation. 30 April 2015
7. "Controlling a line follower robot", laboratory project report | DCI, May 2015) supervised by Mr Guernane. IGEE, University of Boumerdes
8. "H-bridges: theory and practice" Chuck McManis,December 23rd, 2006
9. "SOPC-Based Automated self-parking Car", "Ingenieur d'etat" final year project report, by Hafnaoui Imane and Hadji Isma supervised by Mr Benzekri, IGEE. June 2011
10. Ultrasonic methods of nondestructive testing, j.blitz. simpson
11. Texas, Instruments. "ULN2003A Datasheet" (PDF)
12. TOSHIBA, CORPORATION. "ULN2003 Datasheet".
13. Texas instrument L293D datasheet
14. Cysboy- Apprenez à programmer en Java- OpenClassroom.com 2013
15. Cysboy- Apprenez à programmer en Java- OpenClassroom.com 2013
16. Gardy Booch, James Rumbaugh, Ivar Jacobson The Unified Modeling Language User Guide second Edition
17. Steven M.LaValle- Planning algorithms - University of Illinois – 2006
18. Steven M.LaValle- Planning algorithms - University of Illinois – 2006
19. Applegate, D.L;;Bixby, R.M.;Chvatal, V, Cook, W.J. (2006), The traveling Salesman Problem, ISBN 0-691-12993-2
20. DeKeon, Melissa (200), "A study of sufficient conditions for Hamiltonian cycles", Rose-Hulman Undergraduate Math Journal 1.
21. Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L. (1990). Introduction to Algorithms. MIT press and MCGrawHill. ISBN 0-262-03141-8.

Webography

22. www.pi4j.com, 18/04/2016
23. https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html, 18/04/2016
24. https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html, 03/05/2016
25. https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html, 19/04/2016
26. Https://docs.oracle.com/javase/7/docs/api/lang/RunTime.html, 15/03/2016
27. www.modularcircuits.com/blog author andras tantos, 15/03/2016
28. wordpress.com 05/03/2016
29. Www.learn.adafruit.com/all-about-stepper-motors/what is a stepper motor 02/05/2016.
30. http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf, 04/2016
31. Gaven MacDonald stepper motor control with the raspberry PI youtube video 15/03/2016
32. https://learn.sparkfun.com/tutorials/pulse-width-modulation, 04/2016
33. http://www.micropik.com/PDF/HCSR04.pdf, 20/04/2016
34. www.elecfreak.com, 04/2016
35. http://blog.idleman.fr, 02/05/2016
36. argouml/tigris.org/documentation/umlsupport/index.html, 05/2016
37. https://fr.wikipedia.org/wiki/Raspberry_Pi, 19/05/2016
38. The Unified Modeling Language Reference Manual Second Edition / James Rumbaugh Ivar Jacobson Grady Booch,
39. http://www.gizmojunkee.com/product/raspberry-pi-2-model-b-quadcore-1gb-ram-2/ 05/2016
40. https://www.amazon.fr/Raspberry-compatible-Arduino-ch%C3%A2ssis-dentra%C3%AEnement/dp/B0195HXH44, 16/04/2016