Development and Implementation of Even And Uneven Environment Navigation Strategy

BELKALEM JUGURTHA
ABED MOHAMMED AMINE

5 juin 2016

Dedication

To my lovely parents and grandparents who always supported me in my studies.
To my brothers Sofiane and Malik and my sister
To all my family.
To all my friends who have been there for me, especially : Lyes, Messaoud , Karim, Madjid « The Genius », Tayeb, Mokrane and Yacine.
To my first real life teacher *Karim* from « EPI_MEDIA »Entreprise.
To all people who helped and gave me the chance to be the person I am now.

BELKALEM Jugurtha

To my lovely parents, who have been always my source of inspiration and a big symbol of sacrifice.
To my grand father and grand mother « rabi yarahmoum »my teachers of this life.
To my brothers youcef, koki, wassim and especially my little angel meissa.
To all my family.
To my football team's players mamadou, adel, mounir and vitch and all who share this passion.
To my special friends Mehdi, mohcen, moncef, oussama, nassim, abdou, imad, hamza, okba, foued, seddam and Suki.
To all the people who have helped me to be here today.

ABED Mohamed Amine

Acknowledgments

We would like to express our gratitude to the many people who helped us through this project ; to all those who provided support, talked things over, read, wrote, offered comments, allowed us to quote their remarks and assisted in the editing, proofreading and design.
We would like to thank our supervisor, Dr. Hadjira Belaidi, for the patient guidance, encouragement and advice she has provided throughout our time as her students. We have been extremely lucky to have a supervisor who cared so much about our work, and who responded to our questions and queries so promptly. We would also like to thank all INELEC's staff who helped us financially. In particular we would like to thank signal and systems lab's responsibles(Room B413) who made this project realizable.
We would like to thank HADBI Karim from « EPI-MEDIA »entreprise for helping us in the process of selection and editing.
We would like to thank phd students BAADJI Boussadia and MATI Ayache for proofreading and comments.
Last and not least : we beg forgiveness of all those who have been with us over the course of the years and whose names we have failed to mention.

Abstract

Industrial robotics is evolving quickly around the world, a great interest is shown for these machines. The robot's age is close, new citizens will join our society. However, for these machines to operate complex algorithms are needed and required to handle various situations. An important capability for mobile robots is navigation, reaching a destination without colliding the obstacles, this problem is referred as Path Planning.

Path planning is required for an autonomous mobile robot to find an optimal trajectory to the destination, a human controller is not needed anymore. A new approach has been proposed in this work, built upon the *internet of objects*; computers are focused on path computation and robots on the execution, information is exchanged through the network (wireless local area network). The results outperform by far the classical implementations where robots were making every decision. Scalability is also achieved easily using the proposed topology, multiple robots can be controlled without impacting on the implementation.

In this project, intelligent and robust software has been written to automate every task, a simple input of the map will trigger the whole process, the computer will compute an optimal trajectory then sends it to the robot for execution through the wireless medium(WIFI). The path is preplanned and can be saved for further exploitation. The robot is also able to detect a change in the environment and report it to the computer to get a new path.

Table of Content

List of Figures

List of Tables

Introduction

Robots are rapidly evolving from factory workhorses, which are physically bound to their work-cells, to increasingly complex machines capable of performing challenging tasks in our daily environment. Robots are used everywhere like mass production of consumer goods, assembly and packing, transportation, space and underwater exploration, surgery laboratory research and army. Autonomous robots are special flavors of these which work without human operators with a high degree of autonomy.
Autonomous Robot has to be intelligent in such a way that knowing it's dynamics, initial state and the environment description (set of ***Goals*** and ***Obstacles***), it should be able to drive itself from an initial position to a destination point while avoiding obstacles (collision free path), this is known As ***Path Planning Problem***.
Unfortunately, complicated software and hardware must be combined to construct these machines which is not an obvious task. The software must be robust, fast and easy to use. By the same token, the hardware must meet the environment's irregularity and software compatibility requirements. Complicated math functions are involved to bring system compensation to face different situations. In this project we have constructed this machine, a user interactive system able to accomplish the demands.
Navigation strategy is a classical challenge but papers around the world does not seem to cover the programming aspects of this field. In this work we overcame this limitation and filled some gaps of this problem.

The first chapter provides a description of mobile robots, defines the path planning problem as well as the algorithms deployed in practice to solve it and gives a global image of our implementation of some them.
The second chapter is an overview of the hardware parts of our autonomous mobile robot, by the same mean it provides a quick picture of some technologies evolving in the wild.
Coming to the third chapter, it provides a detailed analysis of our implementation to get a collision free path with some sample codes illustrating this work.
Finally the last chapter shows optimization process made for faster execution and low memory consumption and demonstrates how our software is *resistive* to *reverse engineering and intellectual property* violations.
The remaining chapters dive into the world of path planning.

Chapter 1

Generalities

A new era is coming, a world where robots will live among us and cooperate with humans in our society. This reminds us computers 20 years ago, they invaded our lives suddenly and today they are everywhere (schools, universities, business buildings, railway stations and even in our homes). Robots are expected to be the next computers in the near future.
In this chapter, a special flavor of these are studied closely. Some robots are different from the others which attracted our attention, they are given the name « Autonomous Mobile Robots ».

1.1 Generalities on mobile robots

1.1.1 Definition

According To *McGraw-Hill Dictionary :* « a robot mounted on a movable platform that transports it to the area where it carries out tasks. »[1]

Informally speaking, a mobile robot is an automatic machine that is capable of locomotion(see figure 1.1), an intelligent physical agent that is built to simulate the actions of humans. When they are in an environment : they have self awareness and awareness of others in the same space, they can perceive it, note the changes that occur in the environment, then reacts to the changes, and takes appropriate decisions.
Almost every type of mobile robot operates in a different environment, has different behavior, and connects to different sensors and actuators. Mobile Robots can be classified as follow [2] :

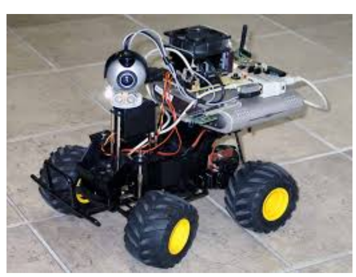

FIGURE 1.1 – Autonomous Mobile Robot

AMR - Autonomous Mobile Robot : can navigate without the need for physical or electro-mechanical guidance devices

AGV - Autonomous Guided Vehicle : rely on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space

1.1.2 Classification Of Mobile Robots

The possible types of mobile robots are unlimited but most often they fall into two categories[3] :

1.1.2.1 The environment in which they travel

Land or home robots : are usually referred to as Unmanned Ground Vehicles (UGVs).

Delivery & Transportation robots can move materials and supplies.

Aerial Robots (UAVs) : are usually referred to as Unmanned Aerial Vehicles (see figure 1.2).

FIGURE 1.2 – Aerial robot Robot

Underwater Robots (AUVs) : are usually called autonomous underwater vehicles (see example in figure 1.3).

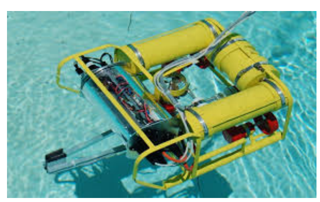

FIGURE 1.3 – Underwater Robot

Polar robots : designed to navigate in icy, crevasse filled environments.

1.1.2.2 The Way(Device) Used To Move

Legged robot : an increasing number of robots use legs for mobility. Legs are often preferred for robots that must navigate on very uneven terrain.
Examples : human-like legs (i.e. an android) or animal-like legs.

Wheeled robot : wheels are by far the most popular method of providing robot mobility and are used to propel many different sized robots and robotic platforms(An example is shown in figure 1.4).

Tracks : tracks (or treads) are similar to what tanks use. Track drive is best for robots used outdoors and on soft ground.(Consider figure 1.5)

FIGURE 1.4 – Wheeled Robot

FIGURE 1.5 – Tracked Robot

1.2 Environment modeling

To perform navigation, a robot needs to interact within its environment. It needs to know how to negotiate terrain, detect and avoid obstacles and sense the surrounding.

1.2.1 Continuous representation

A continuous-valued map is one method for *exact decomposition* of the environment. The position of environmental features can be described with high accuracy. Today's Mobile Robots use continuous map representation only for 2D space as increasing the dimension yields computational complexity.

1.2.2 Discrete representation

We can group them into the following :

1.2.2.1 Exact decomposition

In exact decomposition of a planar workspace populated by polygonal obstacles, the map representation tessellates the space into areas of free space. The representation can be extremely compact because each area is actually stored as a single node(method Shown in figure 1.6).
About Exact decomposition :

- A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries (cylindrical cell decomposition).
- Provides exact solution and leads to completeness.
- Expensive and difficult to implement in higher dimensions.

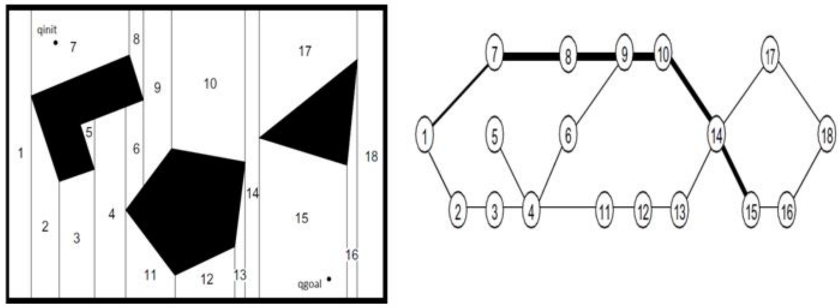

FIGURE 1.6 – Exact Cell Decomposition

1.2.2.2 Fixed decomposition or Approximate Cell Decomposition

In which the world is tessellated, transforming the continuous real environment into a discrete approximation for the map.The idea of Fixed cell decomposition approach is to keep subdividing the environment into subspaces of *equal size* recursively until each subspace is either completely occupied by some obstacle, or completely outside of any of the obstacles, or the pre-specified resolution limit is reached(see figure 1.7).

Approximate Cell Decomposition Advantages :

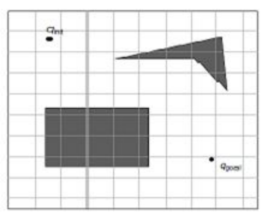

FIGURE 1.7 – Fixed Cell Decomposition

- Limited assumptions on obstacle configuration.
- Approach used in practice.
- Find obvious solutions quickly.

Approximate Cell Decomposition Drawbacks :

- No clear notion of optimality (« best »path).
- Trade-off completeness/computation.
- Still difficult to use in high dimensions.

1.2.2.3 Hybrid representation

It is a combination of the both representation mentioned above. The environment is taken discrete but the motion planning algorithm is considered as continuous environment. This strategy is used to avoid the difficulties of software implementation of continuous environment.

1.3 Path planning

The path planning problem is described as : finding a shortest or optimized path between start point and goal in a spatial configuration consisting of obstacles of various types(see figure 1.8). There are many fundamentally different approaches suitable for different environmental configurations. The various methods are Cell Decomposition, Sampling Method, Probabilistic Roadmap methods, Generalized Voronoi diagrams, etc...

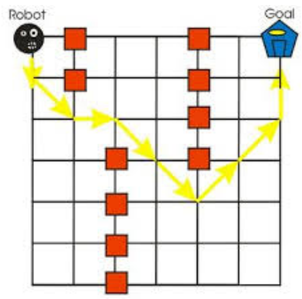

FIGURE 1.8 – Path Planning Problem

1.3.1 Road Map Method

The basic idea behind probabilistic roadmap planner(PRM) is to take random samples from the configuration space of the robot, testing them for whether they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations[4]. The starting and goal configurations are added in, and a graph search algorithm is applied to the resulting graph to determine a path between the starting and goal configurations.
Two different kinds can be Distinguished :

1.3.1.1 Visibility Graph

1. First, draw lines of sight from the start and goal to all « visible »vertices and corners of the world(step illustrated in figure 1.9).

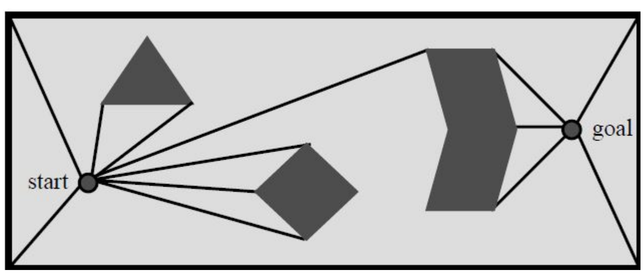

FIGURE 1.9 – Visibility Graph 1st Step

2. Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight(step illustrated in figure 1.10).
3. Repeat until you are done(step illustrated in figure 1.11).

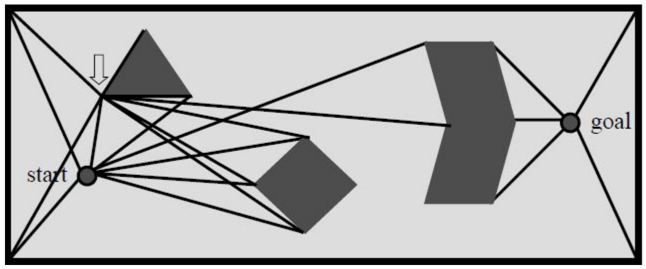

FIGURE 1.10 – Visibility Graph 2nd Step

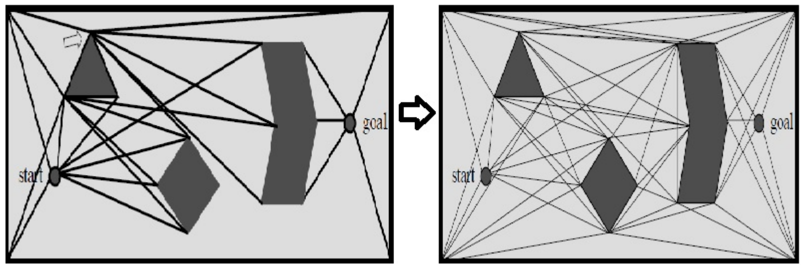

FIGURE 1.11 – Visibility Graph repeated For n Steps

Visibility Graphs : Weaknesses

- It produces a short path but :
  - Tries to stay as close as possible to obstacles.
  - Any execution error will lead to a collision.
  - Complicated in higher dimensions(greater then 2D).
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path.

1.3.1.2 Voronoi diagram(Maximum Clearance Roadmap)

Voronoi diagram of a set of sites in the plane is a collection of regions that divide up the plane. Each region corresponds to one of the sites and all the points in one region are closer to the site representing the region than to any other site.[5]

- Difficult to compute in higher dimensions or *nonpolygonal worlds*.
- Can be unstable because Small changes in obstacle configuration can lead to large changes in the diagram.
- Localization is hard (e.g. museums) if you stay away from known surfaces.

1.3.2 Graph Search

1.3.2.1 Breadth-first search (BFS)

Breadth First Search algorithm(BFS) traverses a graph in a breadthwards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration[6]. BFS was invented in the late of 1950s by **E. F. Moore**, who used it to find the shortest path out of a maze[7], and discovered independently by **C. Y. Lee** as a wire routing algorithm (published in 1961)[8]. This algorithm is summarized by the following :

1. Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
2. If no adjacent vertex found, remove the first vertex from queue.
3. Repeat Step 1 and Step 2 until queue is empty.

1.3.2.2 Depth First Search(DFS)

Depth First Search algorithm(DFS) traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration.
A version of depth-first search was investigated in the 19th century by French mathematician **Charles Pierre Tremaux** as a strategy for solving mazes[9][10]. It is described as follow :

1. Visit adjacent unvisited vertex. Mark it visited. Display it. Push it in a stack.
2. If no adjacent vertex found, pop up a vertex from stack(It will pop up all the vertices from the stack which do not have adjacent vertices).
3. Repeat Step 1 and Step 2 until stack is empty.

Important : DFS is the building block of Garbage Collection Languages(Like Java, Javascript, Python, VB.Net, etc...) [1].

1.3.2.3 Dijkstras Algorithm

Dijkstras algorithm, discovered by E. W. Dijkstra in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with *nonnegative edge weights* [2].
This problem is related to the spanning tree one. The graph representing all the paths from one vertex to all the others must be a spanning tree - it must include all vertices. There will also be no cycles as a cycle would define more than one path from the selected vertex to at least one other vertex [3](see figure 18 in the appendix page52).

1.3.3 Potential Field

The first formulation of artificial potential fields for autonomous robot navigation was proposed by Khatib (1986)[11]. Since then other potential fields formulation have been proposed (Canny 1990, Barraquand 1992, Guldner 1997, Ge 2000, Arambula 2004).[12]
The idea of a potential field is taken from nature[13].The main idea is to generate attraction and repulsion forces within the working environment of the robot to guide it to the target. The goal point has an attractive influence on the robot and each obstacle tends to push away the robot, in order to avoid collisions.
Here is a guide line for *Potential Field* :

1. Compute the attraction potential field(due to the goal) : $U_{att}(q) = \frac{1}{2} * \xi * d^2$(see figure1.12) [4]. Where $d = q_{robot} - q_{goal}$ Where q_{robot} is the current position of the robot, q_{goal} is the position of an attraction point, and ξ is an adjustable constant.

1. Mark And Sweep Phase can be found at : `http://www.brpreiss.com/books/opus5/html/page424.html` shows « Mark and Sweep phase »
2. Dijikstra tutorial is available on this page :`http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html`
3. Dijiktra's pseudo-code can be found on this link : `http://pearl.ics.hawaii.edu/~sugihara/course/ics241/notes/Graphs4.html`
4. You can download the potential field line simulator from : `http://www.physics-software.com/`

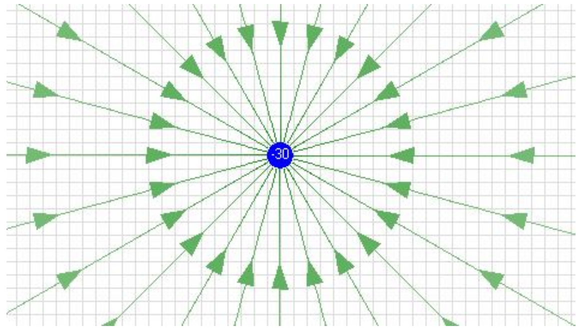

FIGURE 1.12 – Goal Potential Field

2. Synthesize a repulsive force generated by the obstacles. If the robot approaches the obstacle, a repulsive potential : $U_{rep}(q) = \sum_{i=0}^{allobstacles} U_{repO}[i]$ will act on it, pushing it away from It(see figure 1.13).

$$U_{repO}(q) = \begin{cases} \frac{1}{2} * \kappa * (\frac{1}{d} - \frac{1}{d_0})^2 & d \leq 0 \\ 0 & d > 0 \end{cases}$$

Where $d = q_{robot} - q_{obstacle}$ for the robot q_{robot} and the obstacle position $q_{obstacle}.d_0$ is the influence distance of the force and κ is an adjustable constant.

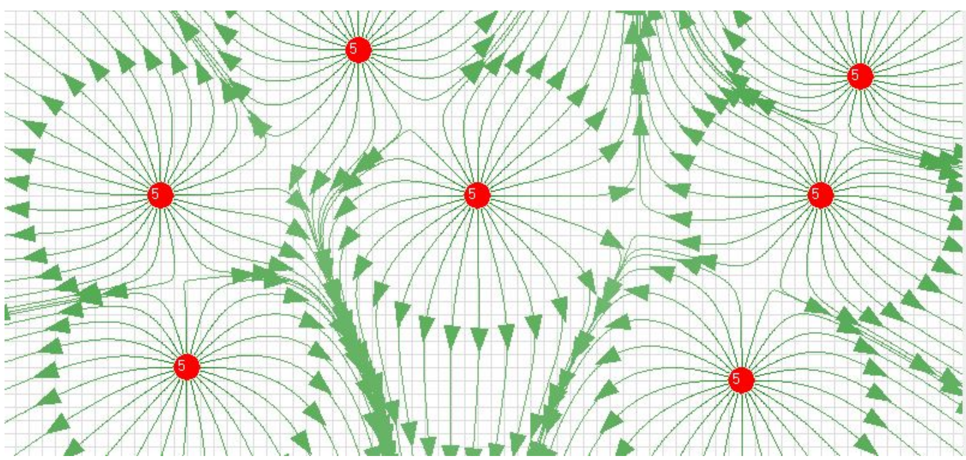

FIGURE 1.13 – Obstacle Potential field

3. Add the previous two behaviours(because a force is linear), the robot then can follow the potential induced by the new field to reach the goal while avoiding the obstacle(see figure 1.14).
Now the robot can be represented as a particle under the influence of a scalar potential field U(q) as follow : $U(q) = U_{att}(q) + U_{rep}(q)$ Where Uatt(q) and rep Urep(q) are the attractive and repulsive potentials respectively. Now we can define the vector field of artificial forces F(q) which is given by the gradient of U(q) : $F(q) = -\nabla U_{att}(q) + \nabla U_{rep}(q)$ where $\nabla U$is the gradient vector of U at robot position q(x, y) in a two dimensional map.

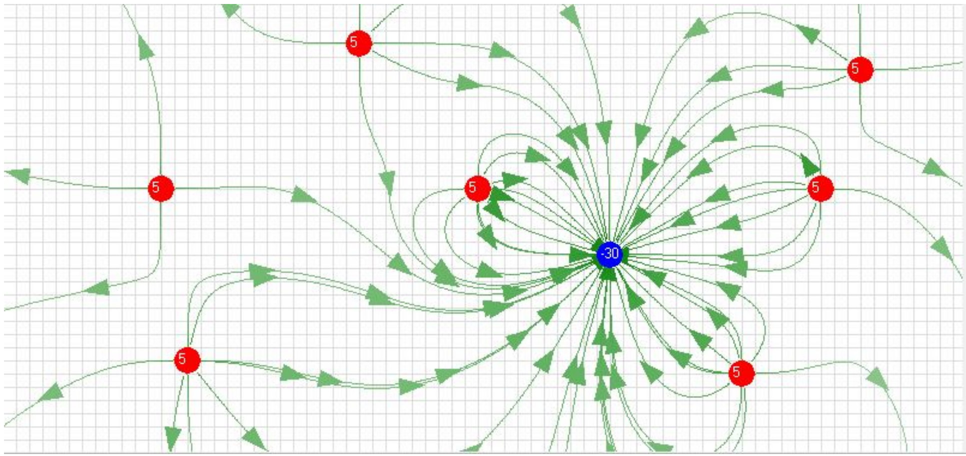

FIGURE 1.14 – Obstacle-Goal Combined Potential Fields

1.3.4 Sensor Based Methods

1.3.4.1 Bug Algorithms

The Bug algorithms are perhaps the simplest and earliest obstacle avoidance techniques one could imagine[14]. Perhaps the most straight forward path planning approach is to move toward the goal, unless an obstacle is encountered, in which case, circumnavigate the obstacle(follow the boundaries) until motion toward the goal is once again allowable [5].
Bug algorithm's family include a broad range of flavors from which we can state :

— **Bug0** *Pseudo Code :* The simplest one in this family(see Figure 1.15) :

```
WHILE (Goal Not Reached)
  head toward goal
  IF an obstacle is encountered,the robot will follow the obstacle
      boundaries until it can head toward the goal again
CONTINUE
```

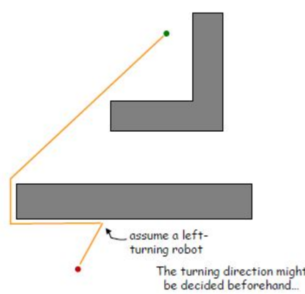

FIGURE 1.15 – bug0 Algorithm

— **Bug1** *Pseudo Code :* the evolution of bug0(see Figure1.16)

```
  WHILE (Goal Not Reached)
```

5. Bug Algorithm's Javascript simulation/implementation is available in this link : `http://barankahyaoglu.com/robotics/bug/`

```
    head toward goal
    IF an obstacle is encountered ,the robot will circumnavigate it and
        remember how close it gets to the goal.
    The robot will return to that closest point (by wall−following).
CONTINUE
```

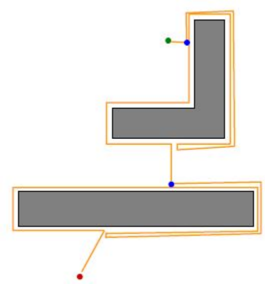

FIGURE 1.16 – bug1 Algorithm

— **Bug2** *Pseudo Code :* the evolution of bug1(see Figure 1.17)

```
WHILE (Goal Not Reached)
  head toward goal on the m−line
  IF an obstacle is in the way,the robot will follow its
      boundaries until it encounters the m−line again closer to the
       goal and will leave the obstacle.
CONTINUE
```

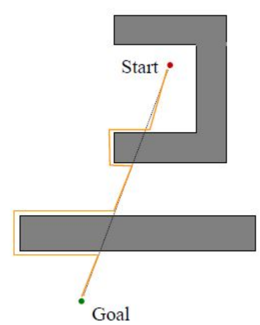

FIGURE 1.17 – bug2 Algorithm

1.3.4.2 D* Algorithm

D* (pronounced "D star") is any one of the following three related incremental search algorithms[15] :

— The original D*,[16] by Anthony Stentz, is an informed incremental search algorithm.

— Focused D*[17] is an informed incremental heuristic search algorithm by Anthony Stentz that combines ideas of A*[18] and the original D*. Focused D* resulted from a further development of the original D*.

— D* Lite [6][19] is an incremental heuristic search algorithm by Sven Koenig and Maxim Likhachev that builds on LPA*,[20] an incremental heuristic search algorithm that combines ideas of A* and Dynamic SWSF-FP.[21]

All three search algorithms solve the same assumption-based path planning problems [7], including planning with the freespace assumption,[22] where a robot has to navigate to given goal coordinates in unknown terrain.

1.3.4.3 Rapidly Exploring Random Trees

A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search non-convex[23], high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem [8]. RRTs were developed by Steven M. LaValle and James J. Kuffner Jr. [24].

1.4 Uneven Environment and Algorithm discretization

1.4.1 Uneven Environment

Uneven is the opposite of uniform and predictable [9].If the road is uneven, not regular, consistent or equal(see figure 1.18).

This introduces a new challenge in navigation as we must take a couple of other factors into

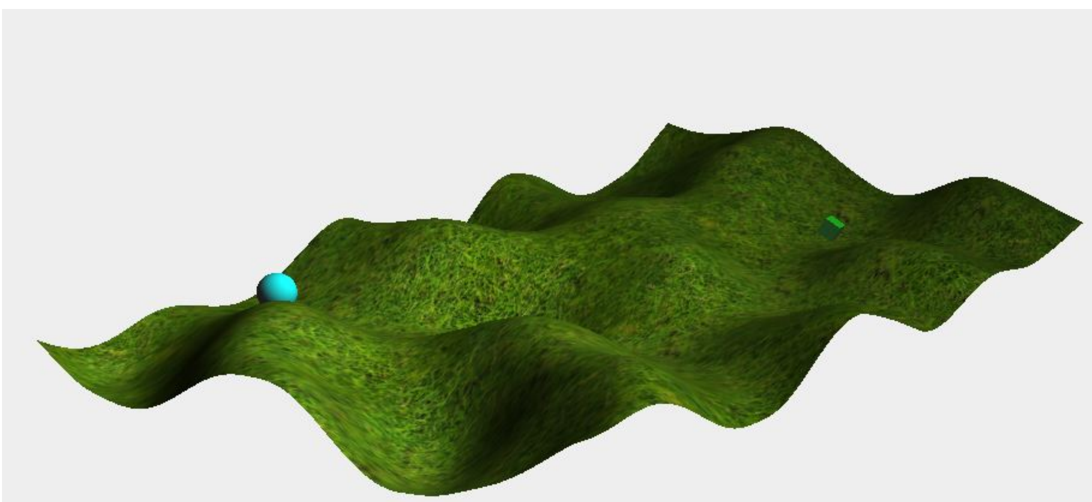

FIGURE 1.18 – Picture From Our Uneven 3D View Simulator Using Perlin Algorithm for terrain generation

consideration : gravity constrains, speed variation, fast algorithm response.

6. D* Lite implementation in C is available from Sven Koenig's page : `http://idm-lab.org/code/dstarlite.tar`
7. Learn A* and D* with Python programming : `http://letsmakerobots.com/node/40568`
8. Rapidly Exploring Random Trees animation can be found in this page : `https://www.jasondavies.com/rrt/`
9. uneven defintion : `https://www.vocabulary.com/dictionary/uneven`

1.4.2 Algorithm discretization

In this C and JavaScript programming languages are used along the way to build a powerful simulator. Unfortunately, the biggest issue with digital computers is the need to discretize.
In this project two different algorithms have been choosen to be manipulated : Bug0 and Potential Field using a *fixed cell decomposition for the environment*.

1.4.2.1 Bug0

Bug0 can be easily coded without difficulty, only a finite set of data are required to generate a Path. But we have provided an enhanced model of the old bug0 and we came with a version that outperforms other implementations on internet(see figure 1.19).

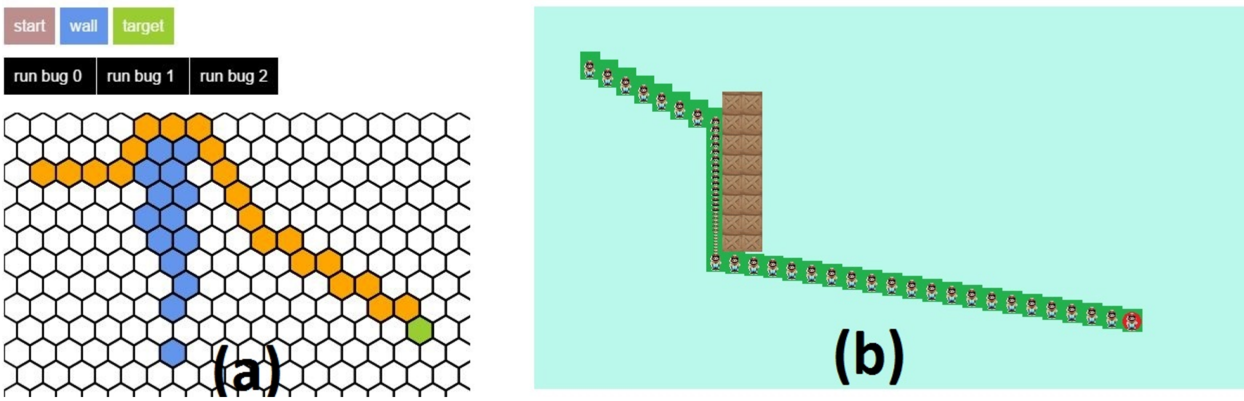

FIGURE 1.19 – bug0 implementation comparison (a)-Simulator from internet(no straight line) (b)-Illustration of our approach

1.4.2.2 Numerical Potential Field

Potential field evaluates an infinite number of points which makes it impossible to implement without sampling.
In 2005, Numerical Potential Field[25] was released making it suitable for computers(this method is implemented in this project).

1.5 Conclusion

This chapter has been an introduction to how mobile robots are impacting our lives and how they are gaining popularity, some of their uses have been exposed as well as their classification and evolution.
Competitors around the world are trying to produce efficient algorithms capable of moving the robot from a starting point to some destination without colliding the objects in it's environment. Another constrain would be reaching the goal as quickly as possible(optimal path), the reason why some algorithms are easy (like bug algorithms which require low computational power) and others are complicated (like dijikstra, A* , D* which needs high computational power). In this work bug0 and Numerical Potential Field have been choosen for our implementation.

Chapter 2

Hardware Implementation

Robots are bringing a revolution, a world's change, they are needed everywhere(industry, warfare, exploration, health, lifting and many more), they became a symbol of power.
However, stable software system and complex hardware are required to complete their daily tasks that they have to undergo.
Hardware industry evolved a lot, it became a competitive field(since the invention of transistor everything is made possible) ; a reason why we must choose the appropriate robot's parts carefully. Let's take a deeper look at the hardware part :

2.1 Mobile robot platform description

The robot's platform(wheeled Robot see figure 1.4 3) is described in figure 2.1, the measurements are in millimeters(mm). it is large enough to carry on the Arduino and the protoboard.

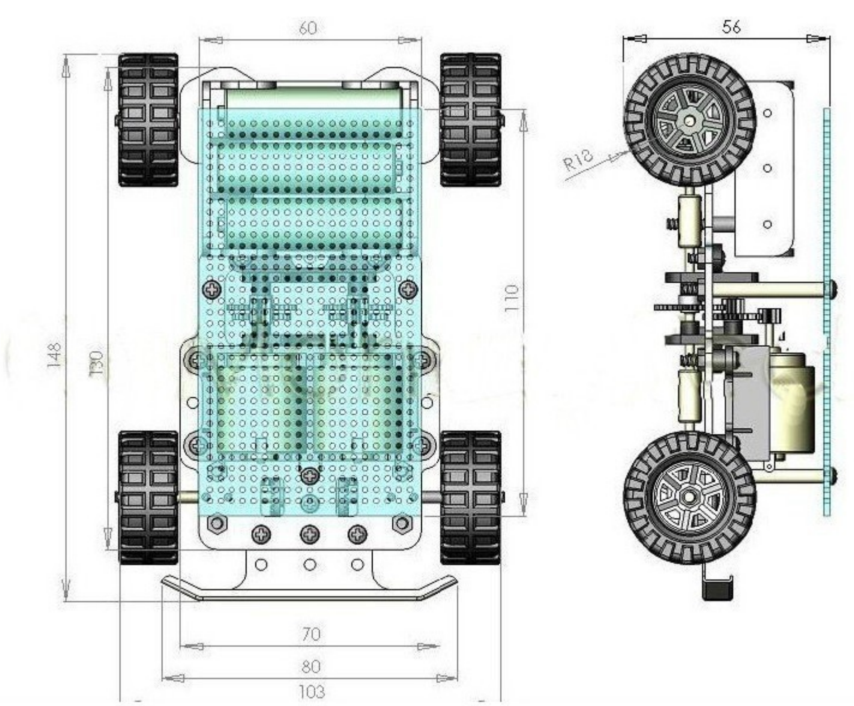

FIGURE 2.1 – Mobile Robot Platform Description (unit : mm)

2.2 Sensors Description

A sensor is an object whose purpose is to detect events or changes in its environment, and then provide a corresponding output [10].
Sensors can detect and respond to some type of input from the physical environment. The specific input could be *light*, *heat*, *motion*, *moisture*, *pressure*, or any one of a great number of other environmental phenomena. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.

2.2.1 Camera

Vision is one of the most powerful and popular sensing method used for autonomous navigation. Camera(see figure 2.2)is the best vision device. it allows large ranges of vision and delivers very detailed images of the environment.

FIGURE 2.2 – Camera Based

An example of these is the OV7670 Camera Module : This camera module (shown in figure 2.3)can perform image processing such as AWB (auto white balance), AE (automatic exposure) and AGC (automatic gain control), for the video signal coming from CMOS sensor. What is more, the fusion of other advanced technology such as image enhancement processing under low illumination, and image noise intelligent forecast and suppress, this module would output high quality digital video signals by standard CCIR656 interface.

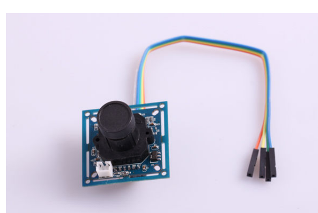

FIGURE 2.3 – OV7670 Camera Module

2.2.2 LADAR

LADAR (LAser Detection And Ranging) systems use light to determine the distance to an object. Since the speed of light is well known, LADAR can use a short pulsed laser to illuminate

10. `https://en.wikipedia.org/wiki/Sensor`

a target and then time how long it takes the light to return. The advantage of LADAR over RADAR (Radio Detection And Ranging) is that LADAR can also image the target at the same time as determine the distance. This allows a 3D view of the object in question. This provides long range reconnaissance with greater fidelity and thus greater recognition range than other technologies(figure 2.4 illustates this method).

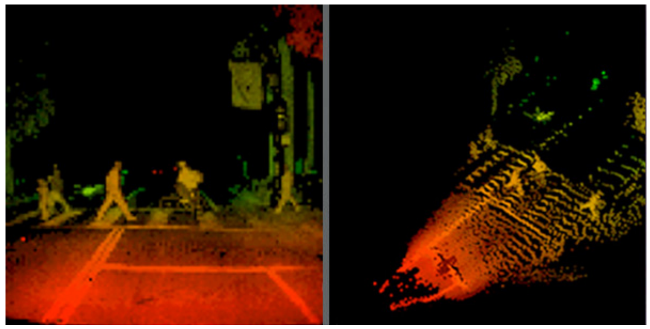

FIGURE 2.4 – In the image above, on the left is a LADAR image from the front of a vehicle stopped at a crosswalk in Santa Barbara, CA. On the right is the same LADAR data « viewed »from an overhead location highlights the 3-D nature of the data collected.

2.2.3 Infrared Sensors

An infrared sensor is a device that emits and/or receives infrared waves in the form of heat(figure 2.5 shows an instance of IR sensor). While most infrared sensors transmit and receive infrared waves, some can only receive them. These types of infrared sensors are known as *Passive Infrared Sensors (PIR sensors)* or motion detectors.

Although infrared sensors can be designed to perform different functions, all infrared sensors are made of pyroelectric materials, whether natural or artificial. A pyroelectric material produces an electrical voltage whenever it is heated or cooled. Most infrared sensors are coated with either parabolic mirrors or Fresnel lenses in order to retrieve infrared waves from an entire room or area. As infrared waves reach the sensor from different areas, they cause the sensor to generate a voltage in different waves, which can be used to trigger an alarm or activate another system.

FIGURE 2.5 – infrared Sensor- GP2Y0A41SK0F Module

2.2.4 Ultrasonic Sensors

A basic ultrasonic sensor (as shown in figure 2.6) consists of one or more ultrasonic transmitters (basically speakers), a receiver, and a control circuit. The transmitters emit a high frequency ultrasonic sound, which bounce off any nearby solid objects. Some of that ultrasonic noise is reflected and detected by the receiver on the sensor. That returned signal is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used, along with some clever math, to calculate the distance between the sensor and the reflecting object.

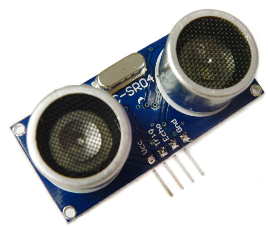

FIGURE 2.6 – Ultrasonic Sensor-HC-SR04

HC-SR04 Specifications :

- Working Voltage : DC 5V
- Working Current : 15mA
- Working Frequency : 40Hz
- Max Range : 4m
- Min Range : 2cm
- Measuring Angle : 15 degree
- Trigger Input Signal : 10μS TTL pulse
- Echo Output Signal Input TTL lever signal and the range in proportion
- Dimension $45 * 20 * 15$ mm

2.3 Necessary sensors which must be added to the platform

2.3.1 LM35 : Temperature Detector

The LM35 is an integrated circuit sensor [11] that can be used to measure temperature with an electrical output proportional to the temperature (in °C). In this work, it is used in *Master Mode* to report environment's temperature (see figure 4.3 in page 37).

2.3.1.1 Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric. In this project it has two usages [12] :

- trigger an alarm to signal that robot reached the destination.

11. Find LM35 Datasheet at : `www.ti.com/lit/ds/symlink/lm35.pdf`
12. Arduino-Buzzer connections and usage : `https://www.arduino.cc/en/Reference/Tone`

— to tell that the received path is inconsistent if it detects an obstacle not supposed to be in the trajectory.

2.4 Wireless LAN Communication Using WIFI

2.4.1 Wifi Overview

Wi-Fi is the name of a wireless networking technology that uses radio waves to provide high-speed network and Internet connections. The Wi-Fi Alliance [13], the organization that owns the Wi-Fi (registered trademark) term specifically defines Wi-Fi as « wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards ». Initially, Wi-Fi was used in place of only the 2.4GHz 802.11b standard, but the Wi-Fi Alliance has expanded the generic use of the Wi-Fi term to include any type of network or WLAN product based on any of the 802.11 standards, including 802.11b, 802.11a, dual-band, and so on, in an attempt to stop confusion about wireless LAN interoperability.
Precaution : in this work we tried to reduce interferences, we have made a quick site survey and verified which channels (see figure 2.7) are already in use by the help of spectrum frequency analyser (lot of them are free for Windows, Mac, Linux) [14].

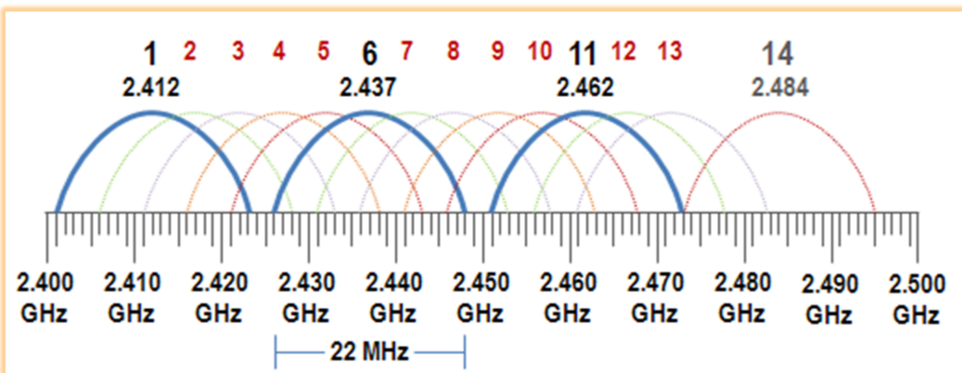

FIGURE 2.7 – Channels and interference on a WiFi network

Note : we can always keep things simple by issuing the command : « netsh wlan show all »in windows cmd and get some precious information (like channels used by every SSID [15]).

2.4.2 Interacting With Wifi

At this time of writing, the searching query ***wifi module*** returned « About 10,300,000 results (0.43 seconds) »which shows the variety of ways of Wifi interaction. Let's choose some of them in terms of : fidelity, transmission speed, cost and documentation.

13. Wi-fi Alliance official website : `http://www.wi-fi.org/`
14. Have a list of spectrum frequency analyser on this link : `http://blog.tanaza.com/blog/wifi-stumblers-the-complete-list-windows-mac-linux-android-ios`
15. More Information on SSID is in this link : `http://searchmobilecomputing.techtarget.com/definition/service-set-identifier`

2.4.2.1 SparkFun WiFly Shield

The WiFly Shield equips the Arduino with the ability to join an 802.11b/g wireless networks. The featured components of the shield are a Roving Networks RN-131C wireless module and an SC16IS750 SPI-to-UART chip. The SPI-to-UART bridge is used to allow for faster transmission speed and to free up the Arduinos UART(figure 2.8 show the WiFly Shield).

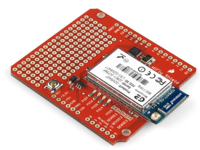

FIGURE 2.8 – SparkFun WiFly Shield

2.4.2.2 ESP8266

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to a WiFi network(see figure 2.9). The ESP8266 is capable of either hosting an application or offloading all WiFi networking functions from another application processor. Each ESP8266 module comes preprogrammed with an AT command set firmware(can be reprogrammed using AT Or Lua Firmware [16]), meaning, it can be simply connected to an Arduino device and get about as much WiFi ability as a WiFi Shield offers. The ESP8266-01 module is an extremely cost effective board(For less Than 3$) with a huge, and ever growing, community.

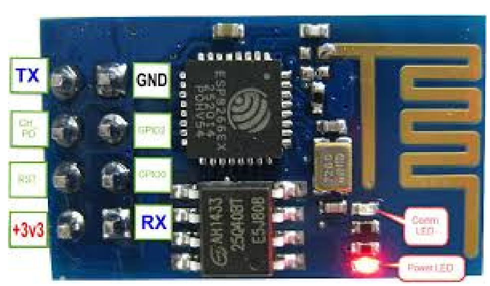

FIGURE 2.9 – ESP8266

Some features of this module are :
— 802.11 b/g/n
— WiFi Direct (P2P), softAP
— Integrated TCP/IP protocol stack
— 1MB Flash Memory
— Integrated low power 32bit CPU could be used as application processor

16. This link explains how to reprogram the ESP8266 `http://blog.randypatterson.com/esp8266-firmware-updates-and-options/`

2.4.3 Reprogramming the ESP8266

This step is quite difficult, it requires some special settings and considerations. The ESP8266 is known to be powerful but hard to use. For this reason, a detailed page[26] has been written in our website along this work at : `http://www.mrobot.netai.net/hardImplemntation.html` discussing this WIFI module and solutions for all problems encountered in this project.

Note : Many counterfeited ESP8266 chips exist, latest « serial com »drivers detect the fake versions and block it(error *code 10* is returned). Downloading an old driver will fix the issue [17].

2.5 Conclusion

At this point of the project, precious information have been gathered which allowed us to choose the right hardware components to build our autonomous mobile robot, the Arduino will be the robot's artificial intelligence controller which will guide all other peripherals (Ultrasonic sensors, motors , ESP8266 and buzzer).
Having such a configuration is very common among the educational and professional worlds , Arduino's community is wide enough to get maximum documentation and help. This was an important step as the choice of hardware will influence dramatically the software's model(discussed more in the next chapter).

17. Error *Code 10* can be fixed with an old driver : `http://www.ifamilysoftware.com/news37.html`

Chapter 3

Navigation strategy development

Navigation strategy must meet the user requirements, the previous works that have been seen in the institute till the date focused on the mathematical part of the problem. The programming part is discussed in this part.
RNCS(Robot Navigation Control/Simulation) is the C language Based program developed during this work. A software which is intended to be the building block of our implementation, *a complete path generator program*.
The following features can be identified in RNCS :

- User friendly : GUI(Graphical User Interface) is used to increase user experience.
- Robustness : the code is stable, clean and optimized (use very few resources) and can run for very long periods without crashing down.
- Fast in response : the program reacts quickly to the user and optimal path is computed in minimal time.
- Security : the software sanitizes user inputs and protects the communication to the robot(Arduino) to avoid eavesdroppers(hackers).

Let's dig deeper into the wonderful world of navigation strategies.

3.1 Environment recognition Using Environment's Map

A computer can be fed with a map and apply on it any algorithm that have been learned so far(bug0 and Numerical Potential Field in our study). One section of our software is the MAP EDITOR(see figure 3.1). The user can place any element (goal, robot, obstacles) *relative to the real world* with a simple mouse click in the environment map. Finally, using the map editor the space configuration of the field is available, by pressing « s »in the keyboard the topology will be saved into a file (maps.ja), so that both Bug0 and Numerical Potential Field can be applied on it to generate a path.

3.2 Obstacles detection and avoidance(Two Level ODA)

Many approaches have been known through the last decades in the field of obstacle detection and avoidance, from the simplest(sensor based like bug family) to the most complicated(image processing using libraries like openCV).

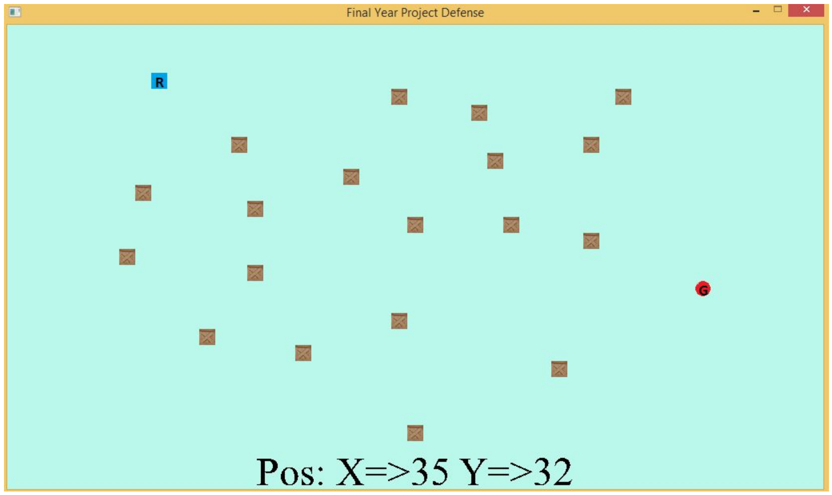

FIGURE 3.1 – Picture From Our Environment Map Editor / Pos : X and Y shows mouse cursor position

However, this work deals with known environment(even and uneven) and it proposes a different solution for the problem, a two steps solution :

1. Computer based computation which does the heavy work of producing the optimal path, then sending it to the robot.
2. the robot is the computer's slave but will not execute the received trajectory blindly, it will always check(using ultrasonic sensors) the correctness of the path.

3.2.1 Computer Level Obstacle Detection(Path Generation)

Having the space configuration(using the Map Editor) a **collision free path** can be *easily produced*.
How does this work ? : the map is a 2D integer array (int mapField[width][height])filled with numbers to indicate the different elements in each cell, for instance this line in C language :

```
int map[4][4] = {{1,0,1,2},{0,1,1,0},{0,3,0,0},{1,0,4,1}};
```

has a matrix representation :

$$\begin{Bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 \\ 0 & 3 & 0 & 0 \\ 1 & 0 & 4 & 1 \end{Bmatrix}$$

We can interpret the matrix as follow :
- Number 0 : Empty
- Number 1 : Obstacle
- Number 2 : Goal
- Number 3 : Uneven obstacle(ground irregularity)
- Number 4 : Robot

We can represent the above numbers in C using « enum »because constants are easier to deal with at source code level than numbers, a sample line from our c code :

```
enum {SAFE = 0, OBJECT, GOAL, UNEVENOBJECT, ROBOT, ROBOTPREVIOUS,
    ROBOTPREVIOUSLEFT, ROBOTPREVIOUSRIGHT, ROBOTPREVIOUSUP, ROBOTPREVIOUSDOWN
    , ROBOTPREVIOUSUPRIGHT, ROBOTPREVIOUSUPLEFT, ROBOTPREVIOUSDOWNRIGHT,
    ROBOTPREVIOUSDOWNLEFT};
```

At this point, we are able to synthesize a map with numbers, let's see a sample from our software (see figure 3.2). Now that we know the content of each cell in our map we can avoid the unwanted

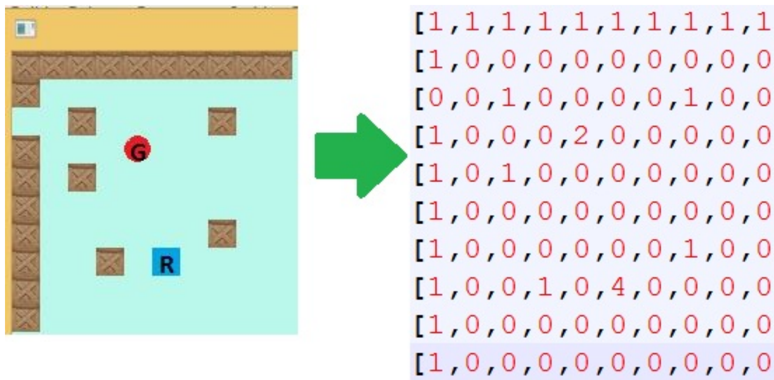

FIGURE 3.2 – Picture from our Map Editor with Matrix representation of the field

objects(Number 1 in our case) and reach the goal(Number 2 in our case), we can generate the path using either Bug0 or Numerical Potential Field(see figure 3.3) and save the path into a file « reportPath.txt ».

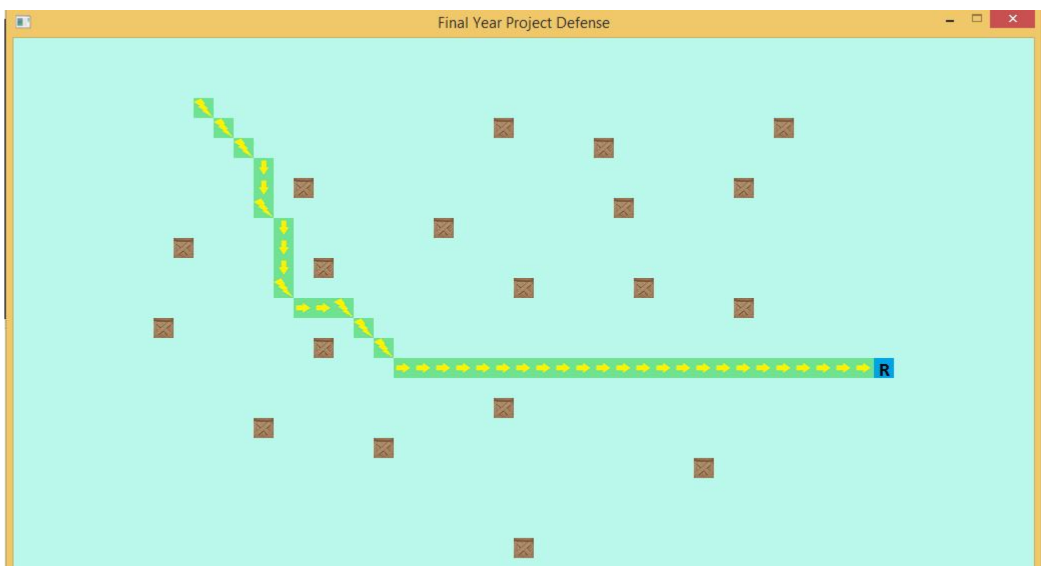

FIGURE 3.3 – Potential Field Applied On The Map(Path Generation Phase)

Path Transmission :
In order to transmit the trajectory across the network we need to extend our program's functionalities to manage Network Communication. The best library candidate from plenty in the wild (like : winHTTP or winINet) is « winsock32.h » [18] because :

18. An excellent tutorial on winSock32 is available on this page : `http://www.binarytides.com/winsock-socket-programming-tutorial/`

— It is low level compared to other libraries.
— The functions it uses are exactly the same as Linux networking libraries which has helped us to make cross-platform code(more in chapter 4).

This header gave our software more flexibility topology for sending data(path) to the robot .
« Wless Com »is the subprogram (see figure 3.4) responsible to send the path to the robot in WLAN(Wireless Local Area Network), secure the communication and detect transmission errors(as it uses TCP) through the wireless medium.

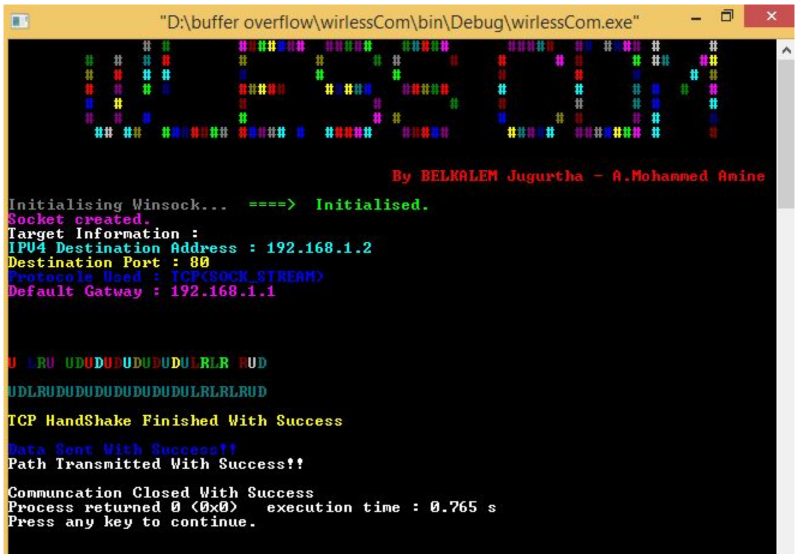

FIGURE 3.4 – Picture From Our Wireless Communication Program

Path Transmission Format :

— **Bug0 :** It is enough to send displaceWithX and displaceWithY which will be added to the current position of the robot. The format of the message {[displaceWithX0, displaceWithY0], [displaceWithX1, displaceWithY1], [displaceWithX2, displaceWithY2], [displaceWithX3, displaceWithY3], [displaceWithX4, displaceWithY4], etc...}
— **Numerical Potential Field :** this method relies on how we subdivided the map, we send first the map precision(cell size) than a stream of direction letters « UDDUR »which means : UP, DOWN, DOWN, UP, RIGHT. Having this we can easily move the robot till the destination. Though, some compression can be made to reduce network overhead, for insctance if the direction letters are of the form « UUUUUULDDDRDRRRRU »can be replaced using Run-Length Encoding[27] by « U6LD3RDR4U ».

3.2.2 Robot Level(Path Execution)

At the reception side (the robot), the Arduino will parse the formatted path and begins the journey to the destination.
Sensors will always be used even if the path is preplanned to check the consistency of the trajectory(there is no obstacle).

How the robot will behave when it encounters an obstacle ?

— Case of path consistency : normal path execution the robot will follow the preplanned trajectory but always checking the sensor data before moving.

— Case of path inconsistency : the Arduino will stop, make a backward move and triggers an alarm, then it will send a message to the computer with the updated obstacle position to compute a new way out.

3.3 Path planning algorithms

3.3.1 Bug0 Algorithm

Bug0 is one of the easiest algorithms in path planning, only a prior knowledge of the robot and goal locations is required.

3.3.1.1 Even environment(2D Environment)

From Linear Algebra : the shortest distance between 2 points(2D configuration space) is the straight line[28]. However, in game theory it is the hypotenuse of the right triangle[29] which can be computed with : $shortestdistance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (see figure 3.5). From this we can

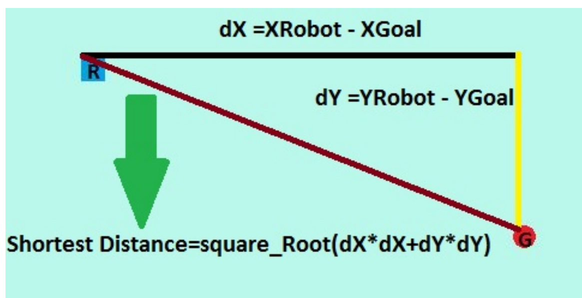

FIGURE 3.5 – Euclidean Shortest Distance in 2D Space(picture from our simulator)

build an extended pseudo-code of bug0 algorithm, take a look at the following pseudo-code :

```
WHILE (robot.position.x != goal.position.x OR robot.position.y != goal.
    position.y)
   Float dX = (robot.position.x >= goal.position.x) ? (robot.position.x -
      goal.position.x) : (goal.position.x - robot.position.x)
   Float dY = (robot.position.y >= goal.position.y) ? (robot.position.y -
      goal.position.y) : (goal.position.y - robot.position.y)
   Float hypothenus = square_Root(dX * dX + dY * dY)
   Float displaceWithX = dX / hypothenus
   Float displaceWithY = dY / hypothenus

   IF (NO Obstacle in the displaceWithX AND displaceWithY Direction)
        robot.position.x += displaceWithX
        robot.position.y += displaceWithY
   ELSE
        WHILE (Obstacle Boundaries Detected And Leaving Point Not Found)
             Follow Obstacle Boundaries In Anti-CLOCK WISE Manner
        End IF
End While
```

Practical Problem :
When we manipulate pixels, floating points cannot be used. Only unsigned integers(uInt32 with SDL.h) are allowed. We have tried :

— Forcing the C program to work with Floats : The application opens and terminates with « code execution 3 »(Segmentation fault).
— Rounding the results to the nearest integer (displaceWithX and displaceWithY) which was working but the trajectory was wrong (no stright line) (see figure 3.6).

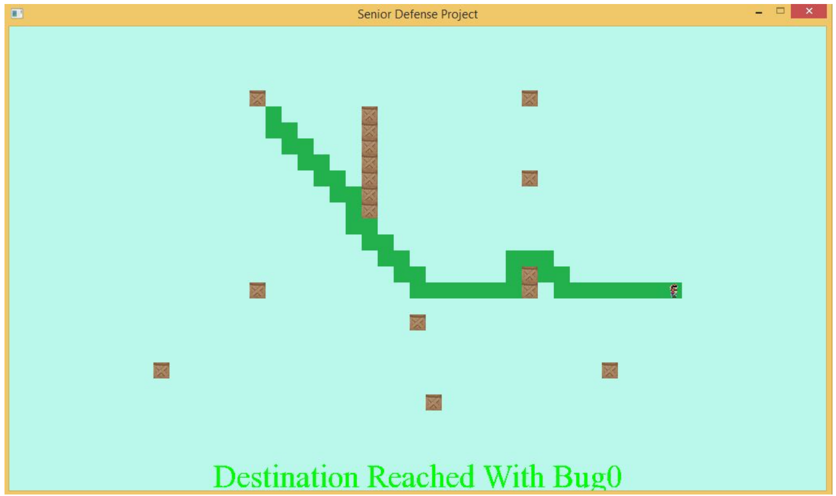

FIGURE 3.6 – Bug0 With Displacement Rounding

Finally we came with a trick for robot position discretization(see figure 3.7)

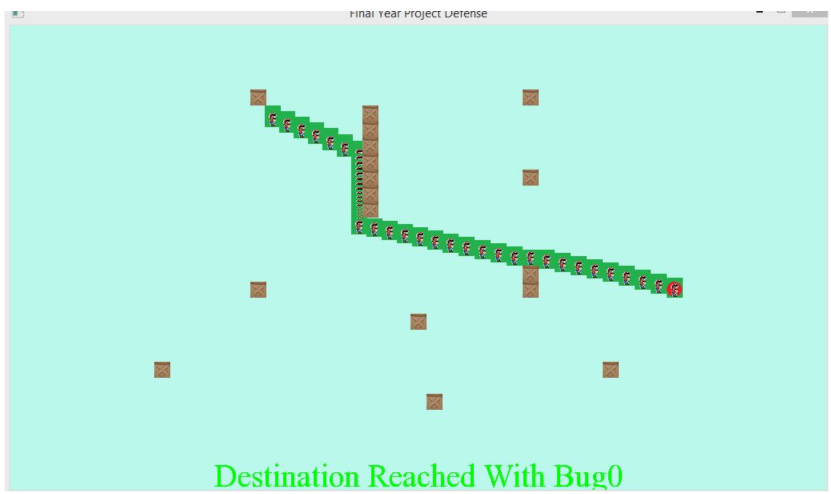

FIGURE 3.7 – Bug0 With Robot Position Discretization

3.3.1.2 Uneven environment

In this kind of environment, a distinction between obstacles and ground irregularity is needed. In this work, it has been kept simple with the assumptions that :
— *Obstacles are higher* than ultrasonic sensors level of the robot.
— *Ground is lower* than sensor level.
This works as follow :

```
WHILE destination not reached
    Head(straight line) To Destination
    IF Object Detected And Object is higher than ultrasonic placement
        Follow The Boundaries of the object till a leaving point appears.
    ELSE IF Object Detected And Object is less than ultrasonic placement(
        Ground Detected)
        IF Robot Can Circumnavigate the ground
            Circumnavigate Like For Obstacles
        ELSE IF Ground Is The Only Path To Destination
            Decrease Motor speed (Higher Torque) to a certain value and
                climb (always head toward destination)
        END IF
    END IF
CONTINUE
```

3.3.2 Numerical Potential Field

For these potentials to be useful the space must be discretized. While the objects themselves have continuous boundaries and smooth path is required, it is **not possible to evaluate the potential at an infinite number of points**. A rectangular grid is a simple discretization that places points at the corners of squares (in 2D) or cubes (in 3D)[25]. The resolution of the potential and the resulting path increases as the size of the squares decreases. As with all numerical methods there is a *tradeoff* between *computation time* (square size), *output accuracy* and resolution[30].

3.3.2.1 Even environment(2D Environment)

Let's take figure 3.8 as a working example :

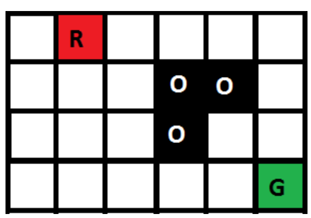

FIGURE 3.8 – Numerical Potential Field Working Example

Steps of obtaining the Numerical Potential Field(see figure 3.14)

1. Create the goal force. If a flat goal force is used then the force at every point except the goal is 1, and the force at the goal is 0(see figure 3.9).

```
For all points x in the space, FG(x) = 1
FG(xgoal) = 0
```

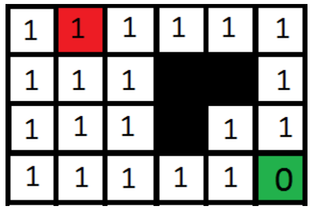

FIGURE 3.9 – Goal Force

2. For simplicity, the obstacle force is assigned with a value 1 in the surrounding grid points and 0 everywhere else. Remember that obstacle's forces add, so if a point is within one grid point of two obstacles then its force is 2(see figure 3.10).

```
For all points x in the space, FO(x) = 0
For all obstacles K,
For all points x surrounding obstacle k, FO(x) = FO(x) + 1
End obstacle loop
```

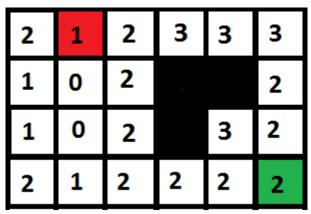

FIGURE 3.10 – Object Force

3. Next, sum the goal force and the obstacle force to get the total force(see figure 3.11).

```
For all points x in the space, F(x) = FG(x) + FO(x)
```

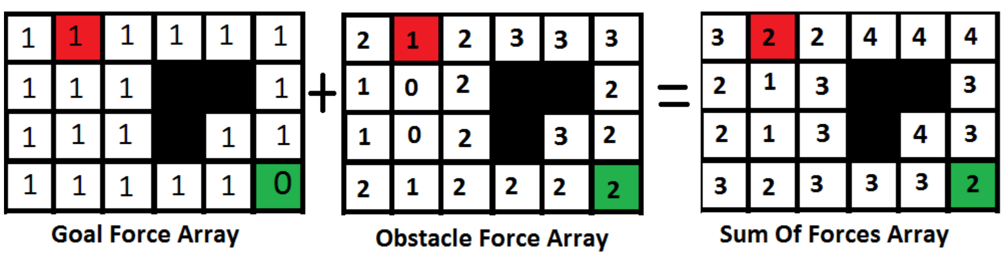

FIGURE 3.11 – Total Sum Of Forces

4. Now it is time to evaluate the potential at every point. Start by assuming that the potential at every point is infinity (or a very large number). Then, starting with the goal, find the points that have the lowest potential and look at their neighbors to update their potentials if necessary. Use a queue to keep track of the points that must be examined(see figure 3.12).

```
For all points x in the space, U(x) = 1,000
    U(xgoal) = 0
    Add xgoal to the queue Q
    While Q is not empty,
    Remove the point xi with the minimum U from Q
    For all points xj that surround xi,
```

```
        If U(xj) > U(xi) + F(xj)
        U(xj) = U(xi) + F(xj)
        Add xj to Q
        end if
        end neighbor for loop
    End while loop
```

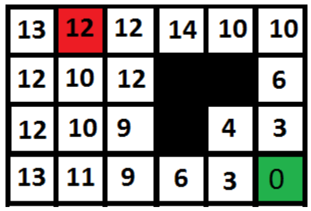

FIGURE 3.12 – Potential Field Array

5. The final step is to make the path from a point xstart.

```
xi = xstart
While xi != xgoal,
x(i+1) = argmin(for xj neighbors of xi) of U(xj)
end while loop
```

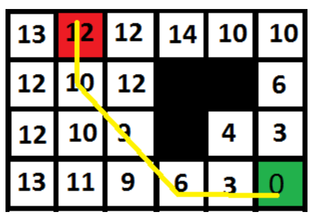

FIGURE 3.13 – Minimal Path Produced By Numerical Potential Field

In the last step(when moving the robot using *potential array*) of the algorithm two scenarios may be handled differently :

— Diagonals are not allowed which takes more time (see figure 3.14)

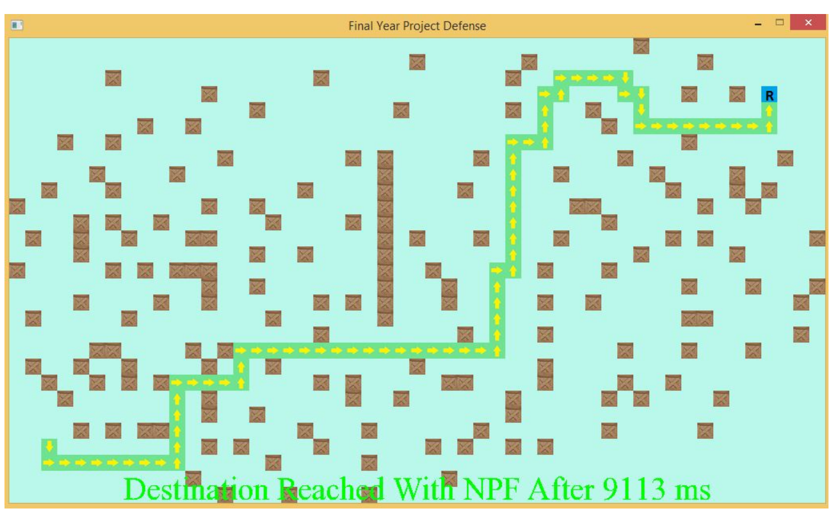

FIGURE 3.14 – Potential Field Implementation-Diagonals Are Not Allowed

— Diagnals are allowed which takes less time(see figure 3.15)

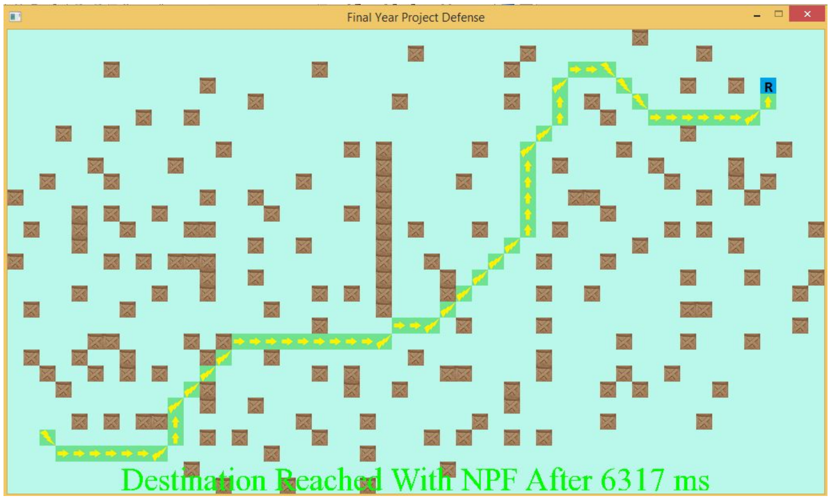

FIGURE 3.15 – Potential Field Implementation Results-Diagonals Are Allowed

3.3.2.2 Uneven environment

Same algorithm of Numerical Potential Field can be extended To 3D but ground and object distinction is added (same assumption as with 3D bug0) as follow :

— *Obstacles are higher* than ultrasonic sensors level of the robot.

— *Ground is lower* than sensor level.

The results are as follow :

— On a flat surface, same algorithm as 2D(see figure 3.16).

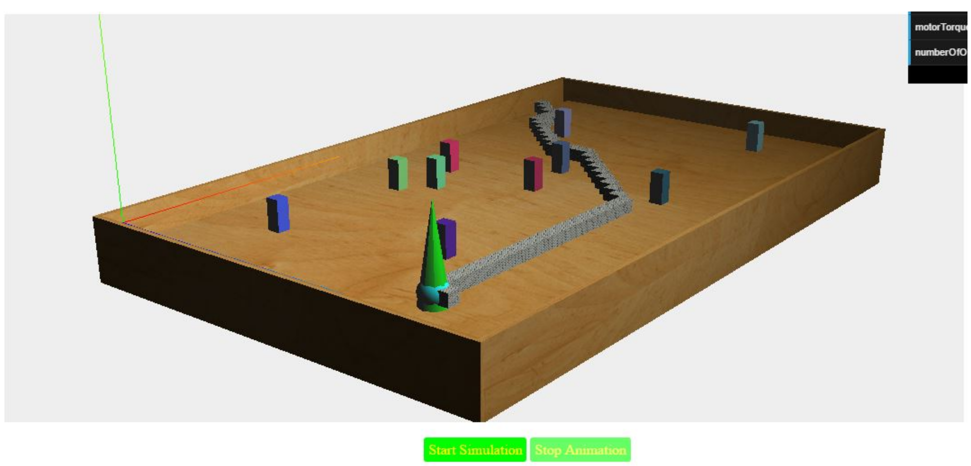

FIGURE 3.16 – Picture From Our 3D Simulator- Navigation with Numerical Potential Field on flat surface

— On Uneven surface the algorithm tries to avoid the irregular ground when possible(see figure 3.17).

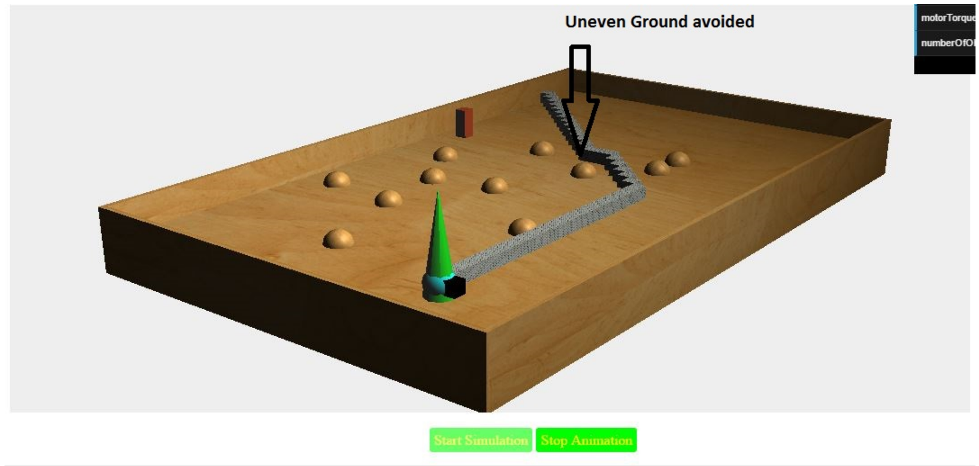

FIGURE 3.17 – Picture From Our 3D Simulator Uneven-terrain (ground circumvented using Numerical Potential Field)

— When no choice is possible the robot must climb over the ground (see figure 3.18).

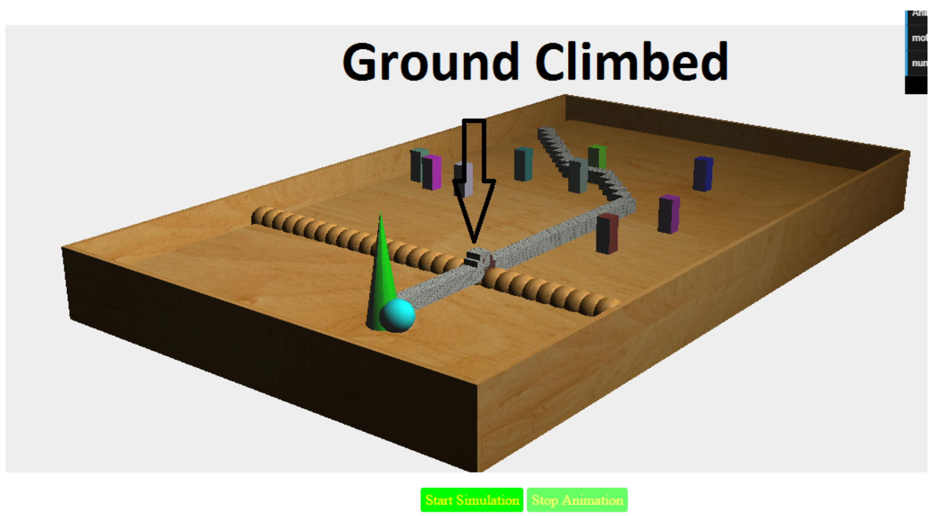

FIGURE 3.18 – Picture From Our 3D Simulator Uneven-terrain (ground climbed using Numerical Potential Field)

3.3.3 Local Minima Problem

Several methods have been suggested to deal with the local minimum phenomenon in path planning. However all of them can be broken into two main blocks :

3.3.3.1 Local Minimum Detection

The first step that must be handled is detecting the local minimum(see figure 3.19).
In our implementation a position based estimator is used to estimate the current position of the robot. If the current position does not change for a considerable amount of time (a predefined threshold) the robot is considered to be trapped.

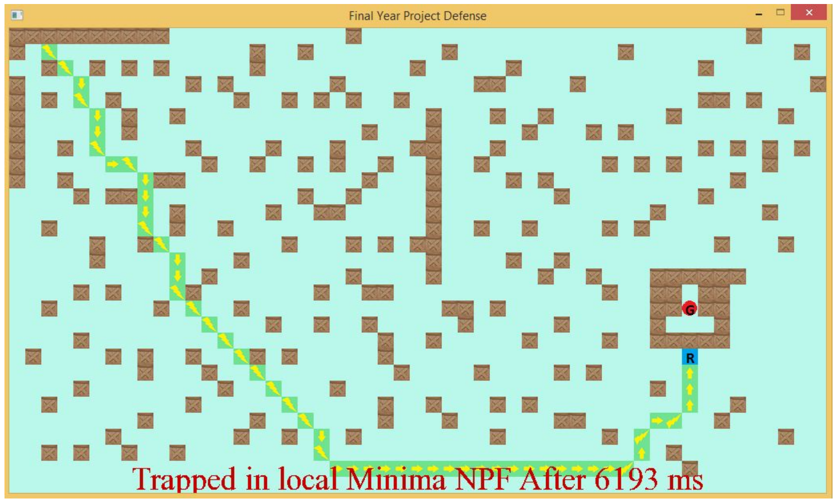

FIGURE 3.19 – Local Minima Detection Using Time Based Estimation

In C Programming different ways for time measurements exist :
- Using the rdtsc Instruction(Intel Assembly AI32) which returns the count of the number of ticks since the last system reboot as a 64-bit value placed into EDX :EAX registers [19].
- Win32 API to Acquire high-resolution time stamps [20].
- Use function SDL_GetTicks() to get the number of milliseconds since the SDL library initialization (Must include SDL.h) [21](this option was used in our work).

The time required for our software to detect the local minima is around 115ms(see figure 3.20).

FIGURE 3.20 – Local Minima Detection Time

3.3.3.2 Escaping Local Minima

A set of approaches can be applied to try fixing the local minima Problem by :

19. More On rdtsc Instruction : https://www.aldeid.com/wiki/X86-assembly/Instructions/rdtsc
20. Win32 API for time measurements : https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408(v=vs.85).aspx
21. More information on SDL_GetTicks() can be found at : https://wiki.libsdl.org/SDL_GetTicks

— Backtracking from the local Minimum and then using another strategy to avoid the local minimum.
— Doing some random movements, with the hope that these movements will help escaping the local minimum.
— Using one of the bug algorithms to avoid the obstacle where the local Minimum exists.
— Using more complex potential fields that are **guaranteed to be local minimum free**, like harmonic potential fields.

3.4 Conclusion

In this chapter, path planning generation problem has been solved for both 2D(Even field) and 3D(Uneven field) based on some assumptions. The mathematical part has been translated to model pseudo-codes, then the latest has led to the software that automates the task of finding a collision free path. RNCS(our software) is also able to send the trajectory through the wireless medium(WIFI) using it's sub-program « Wless Com »to be executed by the robot.
At this time, the first part of the challenge was accomplished with success (*how to get a collision free path ?*), the remaining task will be the exploitation of this path for real execution (at the robot side).

Chapter 4

Hardware and software implementation

Generating the path was the first step in navigation strategy, path execution must follow it quickly and effectively. However, real world perspective is usually different from discrete view(computer) of the environment.
Both softwares (computer and robot sides) must be fully optimized, compressed with very few resources utilization(especially the robot) and immune against internal and external malicious entities and attacks that are rising in the hood. Packet injection is commonly used and require very limited knowledge due to the availability of tools(Wireless Network is more vulnerable to wired network), encryption is used along the way in this work.
The computer software must also be rewritten to work on most platforms (Windows, Linux, Mac OS). Removing dependencies is a hard challenge that we have undertaken.
These points have been investigated in this project, let's explore them more in this chapter.

4.1 Navigation strategy execution

In Chapter 3, the computer side and it's software have been discussed. This part will bring us to the robot and path execution in the real world.

4.1.1 Circuit Overview

Now It is time to take a look at the circuit diagram of this project (see figure 4.1 [22]) :

1 - Arduino Mega 2560 : is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button(see figure 17 page 51).

2 - DC Motors : converts direct electrical current power into mechanical power to move the robot, one motor is used for direction(left or right) and one for forward or backward motion.

3 - ESP8266-01 Module : Wi-Fi chip with full TCP/IP stack and microcontroller capability produced by Shanghai-based Chinese manufacturer [23] in August 2014.

22. Visit The Official Web Site Of Fritzing : `http://fritzing.org/home/`
23. Complete Documentation on ESP8266 on `http://www.esp8266.com/`

4 - **L293d :** The L293D works on the concept of typical H-bridge, a circuit which allows the high voltage to be flown in either direction [24]. In a single L293D IC there are two H-bridge circuits which can rotate two DC motors independently.

5 - **Ultrasonic Sensors :** Ultrasonic sensors use sound waves rather than light, making them *ideal* for stable detection of *uneven surfaces*, liquids, clear objects, and objects in dirty environments. These sensors work well for applications that require precise measurements between stationary and moving objects.

6 - **9V-Battery :** to power-up the hole system.

7 - **Buzzer :** to signal path execution completion or the presence of a dynamic obstacle(obstacle not supposed to exist at that location).

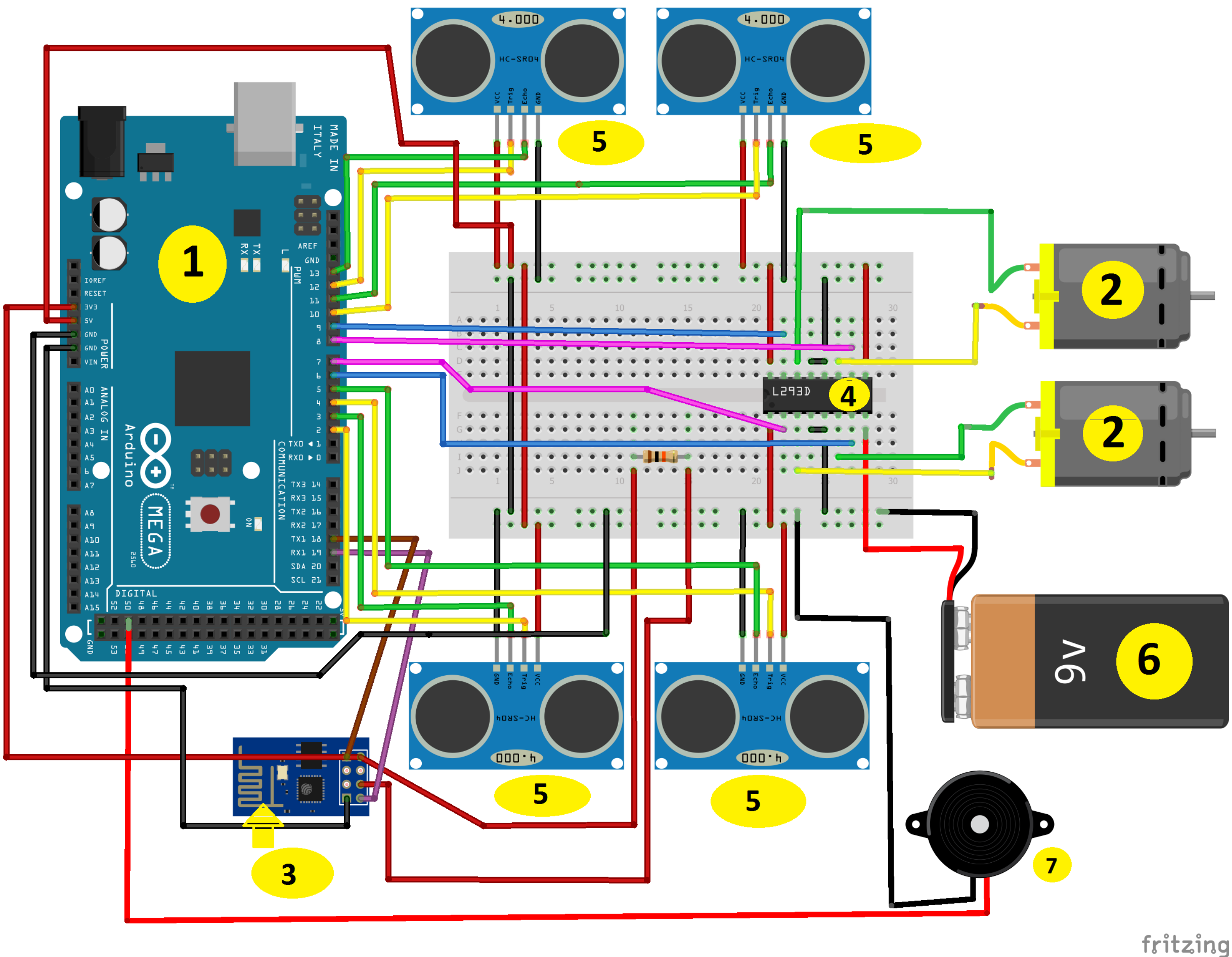

FIGURE 4.1 – Project Circuit Diagram Made With Fritzing

The following corresponds to the circuit diagram above :

24. L293d data sheet : `http://www.ti.com/lit/ds/symlink/l293.pdf`

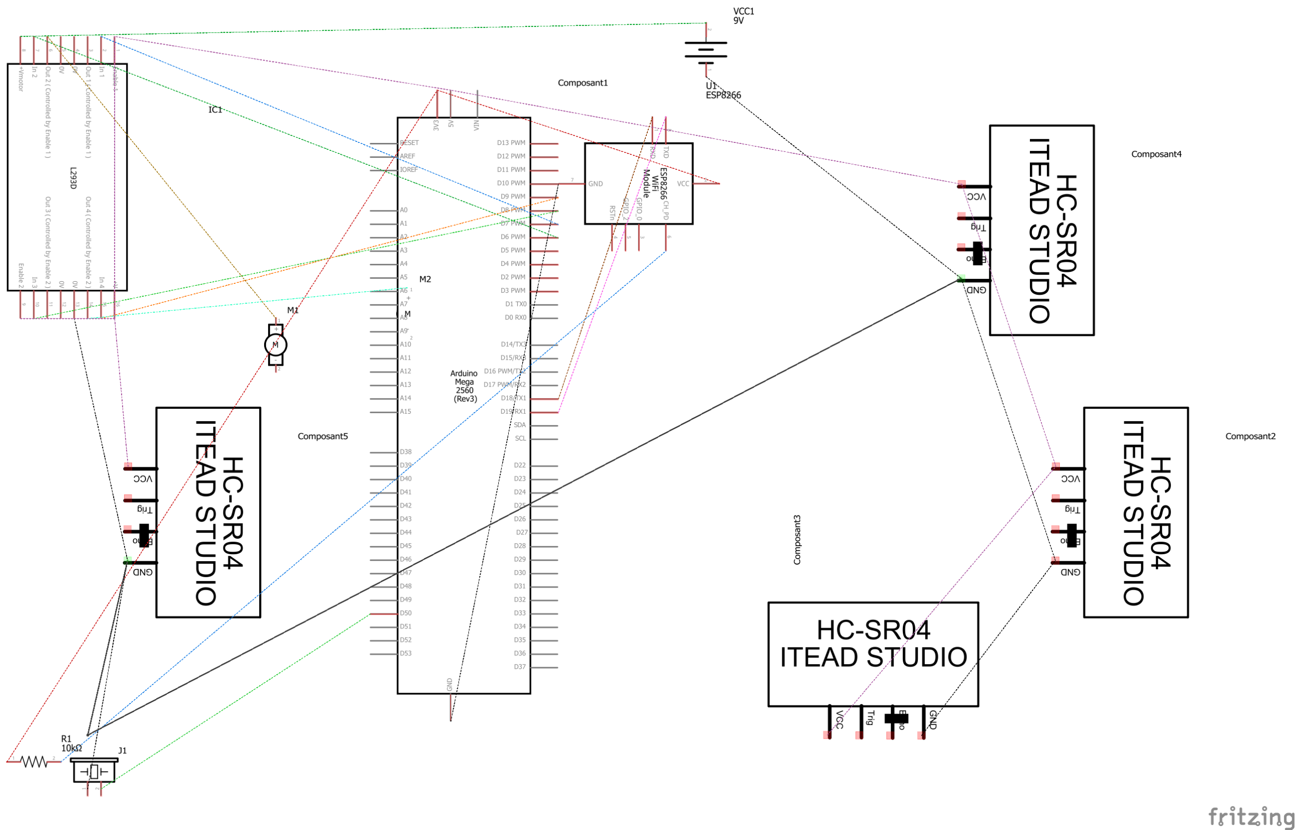

FIGURE 4.2 – Project Schematic Diagram Made With Fritzing

4.1.2 Path Parsing And Execution

The last step of path planning is path execution ; but in order to follow the trajectory, the robot must extract the information inside the received packets which depends on the algorithm being used.

4.1.2.1 Parsing Received Path

Bug0 : the received path is defined as {[x0,y0], [x1,y1], [x2,y2], [x3,y3], [x4,y4], etc, ...} where [x0,y0] are robot's initial position(the robot must know it's position), then the execution strategy is explained by the following Pseudo-Code :

```
String path = readFromArduinoSerialPort()
robotPosition.x = path[0][0]
robotPosition.y = path[0][1]
counter = 1
WHILE path[counter] != '\0'
    ACTIVATE the Ultrasonic Sensor closest to path[counter][0] and
        path[counter][1] direction
    IF NO OBSTACLE DETECTED
      robotPosition.x += path[counter][0]
      robotPosition.y += path[counter][1]
    ELSE
      STOP the Robot
      Use Buzzer to signal path inconsistency
```

```
            Send new obstacle position to the computer to compute the
                path again
            break
          END IF
          Deactivate Ultrasonic Sensor
          counter++
      END WHILE
```

Numerical Potential Field : the received path is simple of the form [mapSize/URLUUL], this can divided using « / »character which yields to [mapSize] and [URLUUL], now the robot will execute it as [go Up with mapSize, go Right with mapSize, go Left with mapSize, go Up with mapSize, go Up with mapSize, go Left with mapSize], in our work mapSize = 20cm.

4.1.2.2 Path Execution

After parsing the path, the robot deduces the algorithm being used and start moving the motors toward the goal.
Special Precaution :
The internal buffer of Arduino accepts up to 64Bytes of data.The received path should not exceed this amount of space otherwise buffer overflow would result[31].
In case of long path(more than 64Bytes), the computer program divides it into chunks and sends them across the network.The Arduino will reconstruct the path easily because the communication is built on the top of reliable protocol TCP(sequence and acknowledgment numbers are used to reorder the received packets).

4.1.3 Master Mode(Web Interface)

Even if the algorithms seen so far are powerful and efficient, no program can replace a human controller ; for this reason, a master mode or full control mode is provided and open for customizations depending on the specific application (see figure 4.3).

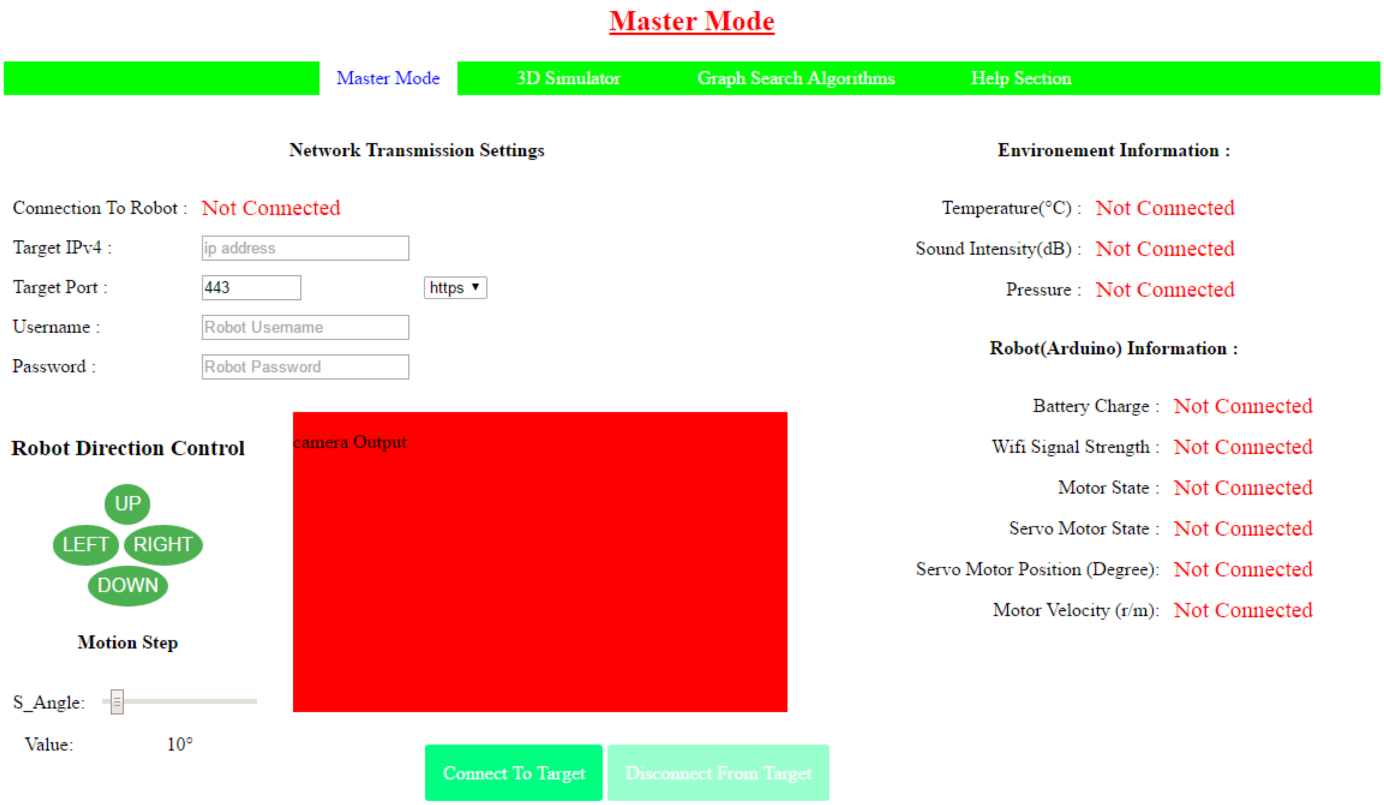

FIGURE 4.3 – Picture From Master Mode Web Interface

4.2 Experimental tests

4.2.1 Software Optimization

4.2.1.1 Cross-Platform Code (Removing Software OS Dependencies)

One of the most useful pieces of information that can be gathered about an executable is the list of functions that it imports. Code libraries can be connected to the main executable by linking. This linking process has been studied because libraries makes a program operating system's specific. This work started from analyzing our windows executable in order to add multiplatform features to our software(rncs.exe).
The program executable(.exe) file header stores information about *most* libraries and functions(see table 4.1) that will be loaded to be used by the program.

dynamically linked libraries : When libraries are dynamically linked, the host Operating System searches for the necessary libraries(.idata PE part) when the program is loaded. This section of our program can be easily reversed Using Dependency Walker(see figure 4.4) [25]. In this way, operating system specific libraries can be determined and provided with their equivalents on other platforms.

25. Explore PE imported functions using Dependency Walker : `http://www.dependencywalker.com/`

TABLE 4.1 – Sections of a PE File for a Windows Executable

Executable(.exe) sections	Description
.text	Contains the executable code
.rdata	Holds read-only data that is globally accessible within the program
.data	Stores global data accessed throughout the program
.idata	Sometimes present and stores the import function information ; if this section is not present, the import function information is stored in the .rdata section
.edata	Sometimes present and stores the export function information ; if this section is not present, the export function information is stored in the .rdata section
.pdata	Present only in 64-bit executables and stores exception-handling information
.rsrc	Stores resources needed by the executable
.reloc	Contains information for relocation of library files

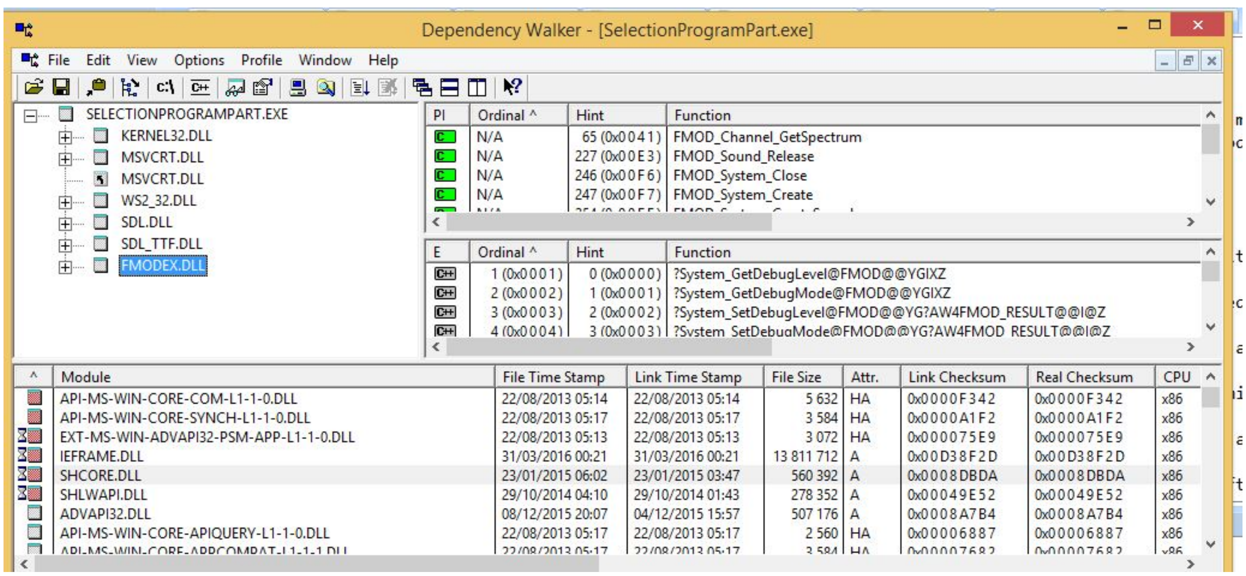

FIGURE 4.4 – Our Software Dependencies Using Dependency Walker

From the list shown in figure 4.4 the following is concluded : SDL, SDL_ttf and fmodex are all cross platform libraries. However WS2_32.dll(winsock2.h) works only on windows[32]. The source code must be rewritten as :

```
#ifdef _WIN32 /*if WINDOWS is detected*/
    #include <winsock2.h> /* for socket(), connect(), send(), and recv
        () *, sockaddr_in and inet_addr(), closesocket() */
  #else /* UNIX, LINUX, MAC OS*/
    #include <sys/socket.h> /* for socket(), connect(), send(), and
        recv() */
    #include <arpa/inet.h> /* for sockaddr_in and inet_addr() */
    #include <unistd.h> /* for close() */
#endif
```

Running time linking libraries : which cannot be viewed using Dependency walker. Several Microsoft Windows functions allow programmers to import linked functions not listed in a program's file header. The two most commonly used are *LoadLibrary* and *GetProcAddress*. *LdrGetProcAddress* and *LdrLoadDll* are also used. Hopefully, these functions(*LoadLibrary and GetProcAddress*) do not exist in the header (see figure 4.4) as we saw using Dependency Walker, the program will never import any library at run-time.

Checking for program's start-up external program dependencies : sometimes decompression, decryption or unpacking programs must be used on a program before the OS loads it(another form of OS dependency). We must have a greater insight to the « .text »section of our program to get the answer. Let's use PEview (see figure 4.5).

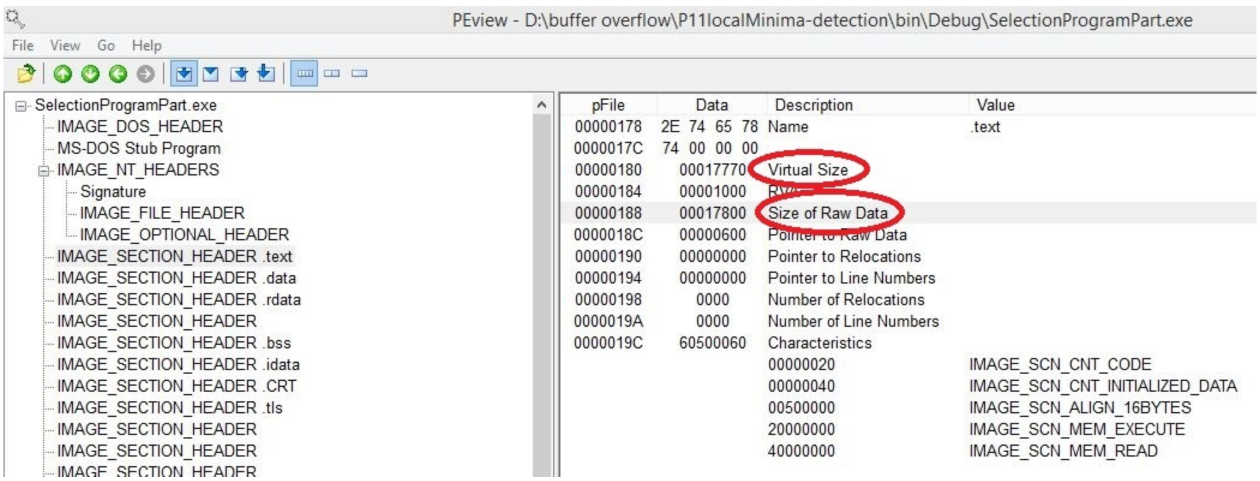

FIGURE 4.5 – Our Software Under PEview- Virtual Size=Raw Data

Virtual Size(found in .text section in table 4.1) tells us how much space is allocated for a section during the loading process. The Size of Raw Data shows how big the section is on disk[33]. These two values should usually be equal, because data should take up just as much space on the disk as it does in memory. *Small differences are normal, and are due to differences between alignment in memory and on disk*. It can be seen clearly that the *Virtual Size is almost equal to raw data size* so the program is neither compressed nor encrypted.
Note : Computing program's entropy can also lead to the same conclusion.

More with 32/64 bits version : Mingw(C++ Compiler) is a 32 bits Compiler, but there exists a 64 bits version(it can be downloaded from : `http://www.mingw-w64.org/`). When a 32 bits version program is running inside a 64 bits machine, the operating system is rushing for backward compatibility[34] which results in slow software's execution. After setting-up our IDE(Code-Blocks for windows and Linux is shown in) we produced the 64 bits software versions for Windows, Unix, Linux and Mac OS.

Now, our C code can be compiled for most known 32/64 bits Platforms (Windows, Unix, Linux, Mac Os).

4.2.1.2 Code Minification (minimization)

C/C++ Code : C++ compiler(recommended over traditional C Compiler) is making a lot of optimizations on the code, gcc supports more than 1500 different options that change it's behavior

toward the source code. In order to reduce the code size the following arguments have been added to the compiler settings : *« gcc myProgram.cpp -o myProgram -O2 -s »*.
Where -O2 reduces program assembly output size (there is also -O3 which is the most effective but error prone) and -s removes debugging options from the program's header for further minification (most today's softwares are compiled using this parameter). The results are shown in table 4.2.
Javascript : Javascript is one of the most popular programming languages, many tools have been

TABLE 4.2 – Program Size with different compiling options

Option	**Size(Ko)**
gcc myProgram.cpp -o myProgram	272
gcc myProgram.cpp -o myProgram **-O1**	289
gcc myProgram.cpp -o myProgram **-O2**	281
gcc myProgram.cpp -o myProgram **-s**	113
gcc myProgram.cpp -o myProgram **-O1 -s**	93
gcc myProgram.cpp -o myProgram **-O2 -s**	90

designed through the last years to check [26] and compress it for faster transmission over the network(like : JSMin YUI Compressor). In our work the « Google Closure Compiler is used [27] »(see figure 4.6).

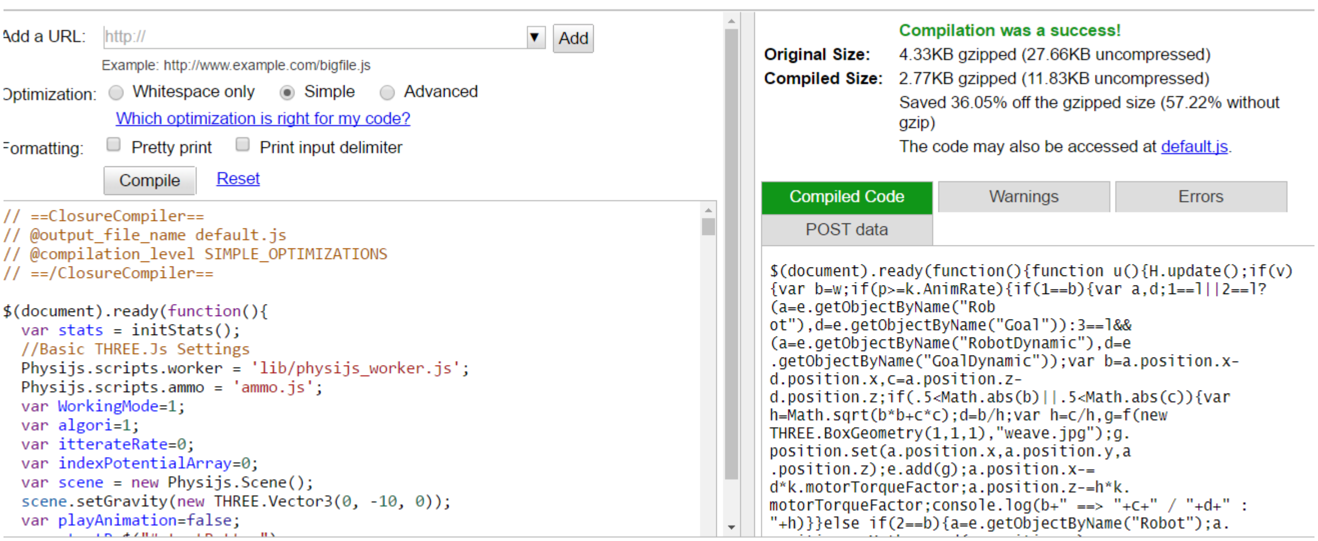

FIGURE 4.6 – Compressing Our Javascript code using Google Closure Compiler yields 36.05% compression efficiency

Our programs are optimized to run with high efficiency.

4.2.2 Numerical Potential Field Computation Complexity

The time complexity and memory usages required for numerical potential field to compute the potential array(which is used to move the Robot) have been studied, the environment has been

26. JSLint is the best Javascript validator : `http://www.jslint.com/`
27. You can access Google Closure Compiler in this link : `https://closure-compiler.appspot.com/home`

subdivided into different number of cells and tests have been made on « AMD C-60 APU with Radeon(tm) HD Graphics 1GHZ »(very low processing capacity), the results are shown on table 4.3.

TABLE 4.3 – Cell Number VS Time Complexity and Memory Complexity

Map Decomposition(Cells)	Time Complexity To Compute(ms)	Memory Consumption
2 * 2	0.041047	4 bytes
4 * 4	0.082094	16 bytes
8 * 4	0.135456	32 bytes
8 * 6	0.141613	48 bytes
8 * 8	0.167267	64 bytes
10 * 8	0.204210	80 bytes
10 * 10	0.221655	100 bytes
11 * 10	0.273990	110 bytes
11 * 11	0.311959	121 bytes
12 * 11	0.356085	132 bytes
12 * 12	0.368399	144 bytes
13 * 12	0.397132	156 bytes
13 * 13	0.405341	169 bytes
14 * 13	0.412525	182 bytes
14 * 14	0.448441	196 bytes
15 * 14	0.456651	210 bytes
15 * 15	0.462808	225 bytes
16 * 15	0.477174	240 bytes
16 * 16	0.483331	256 bytes

4.2.3 Adding Software Security

Security became an important task which is not well understood by engineers. A crucial step has to be made(most programmers fail to do it) for our software to survive reverse engineering and prove the ownership of the code.
To achieve this different protection mechanisms are introduced to our program which can be stated :

Anti Virtual Machine Techniques : Generally, reverse engineering(especially : malware analysis) is made inside a virtual machine to avoid any infection and have more control on the program.

Anti Debugging Techniques : The Debugger is the most used tool to understand the dynamic behavior of the software. Examples are : OllyDbg, an x86 debugger developed by Oleh Yuschuk [28](see figure 4.7).

28. This link shows OllyDbg official website `http://www.ollydbg.de/`

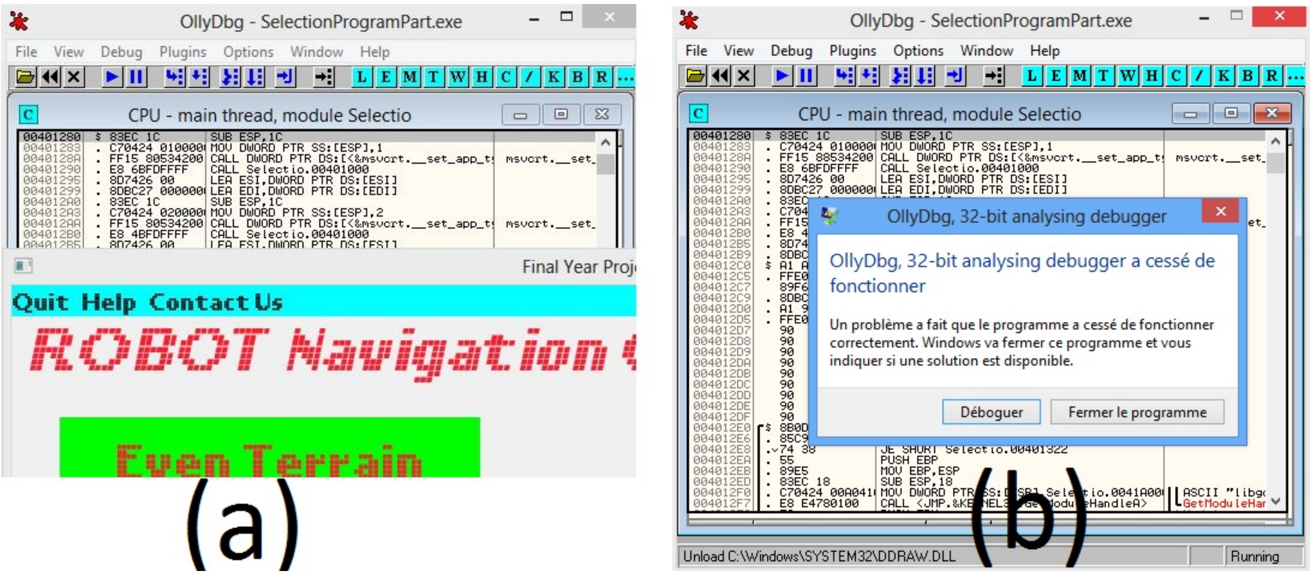

FIGURE 4.7 – Our Program Under OllyDbg(Debugger) : (a)-before AntiDebug / (b)-after AntiDebug

Anti Disassembly Techniques : Disassembler come to the second position of tools used against softwares, an example of these is IDAPro [29].

Extensive use of digital Steganography : hidden data are also inserted into the software(see figure 4.8) to prove the ownership[35].

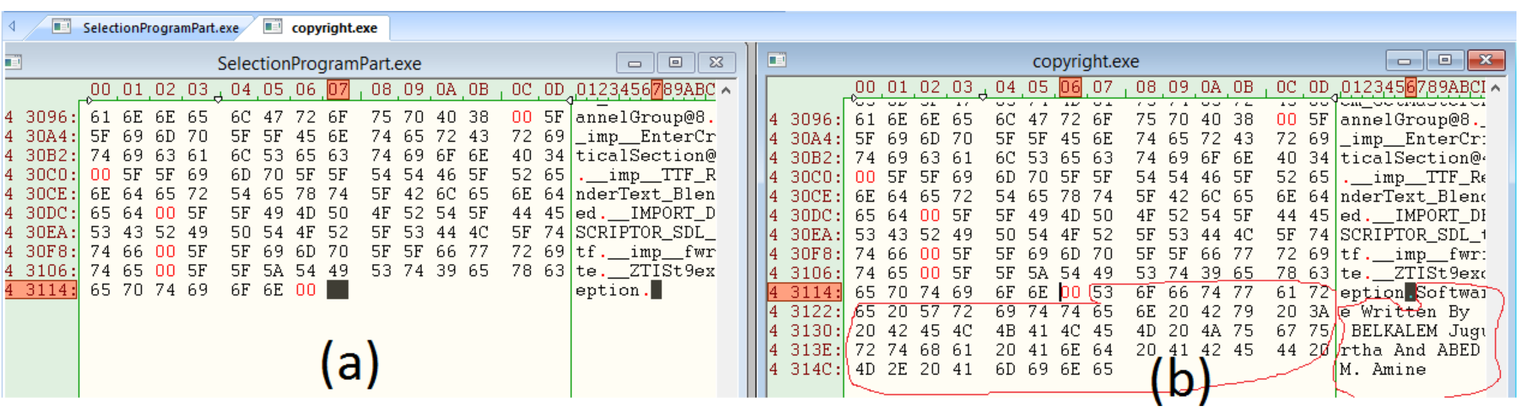

FIGURE 4.8 – Our Program Under HexEditor : (a)-before Steganography / (b)-after Steganography

Code Obfuscation : (At source code level not at Assembly level[36]) was introduced into the code to make difficult to reverse and understand [30].

4.2.4 Real World Results And Discussion

At this stage, it is time to take a look at the robot. Figure 4.9 gives a real picture of the discussed circuit (figure 4.1 at page 34).

29. IDAPro official website : `https://www.hex-rays.com/products/ida/`
30. Learn about Top Down Obfuscation : `https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-sean_taylor-binary_obfuscation.pdf`

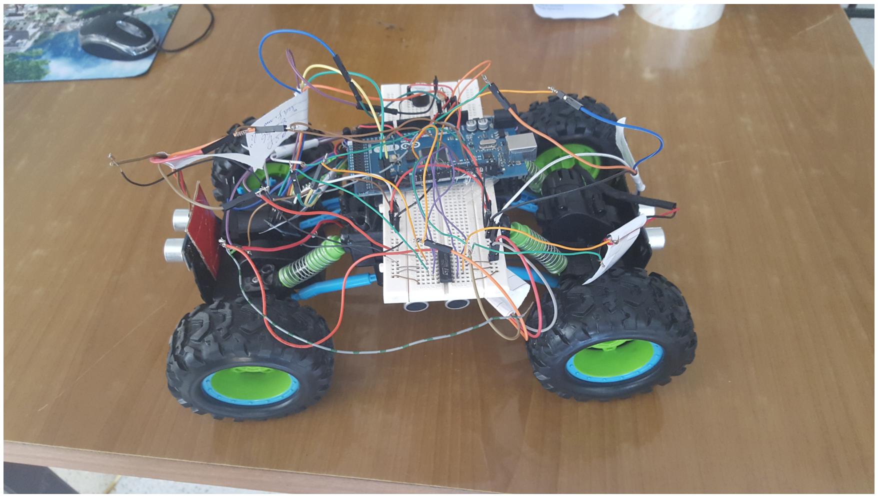

FIGURE 4.9 – Robot Implementation

4.2.4.1 Path Execution

A map is constructed in our software(rncs) which will serve as a reference test or ideal path(see figure 4.10).

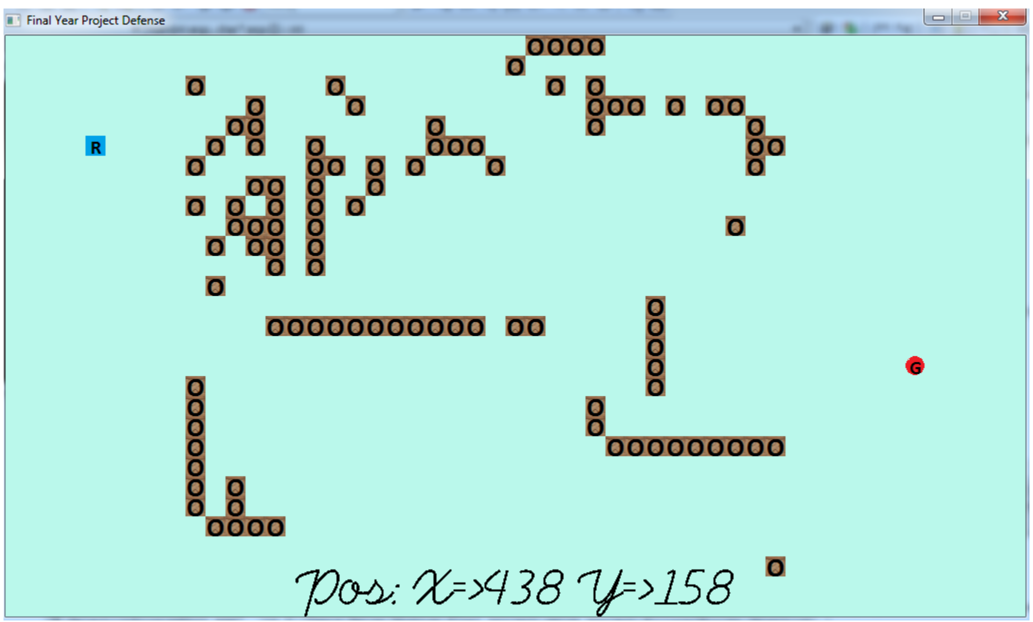

FIGURE 4.10 – Initial state configuration of the environment

Let's use Numerical Potential Field in order to generate a path which will be compared to the real trajectory followed by the robot in the real world(see figure 4.11).

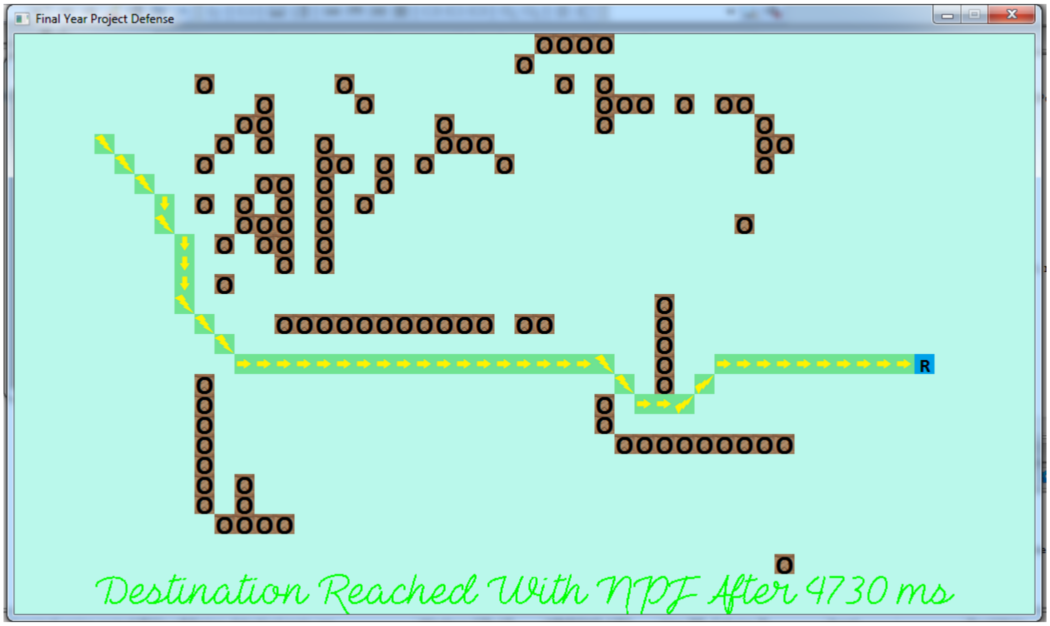

FIGURE 4.11 – Desired path produced using Numerical Potential Field

Figure 4.12 shows the robot while executing the reference path(see figure 4.11), each move is followed purposely by an important delay to make it possible for us to take precise pictures.

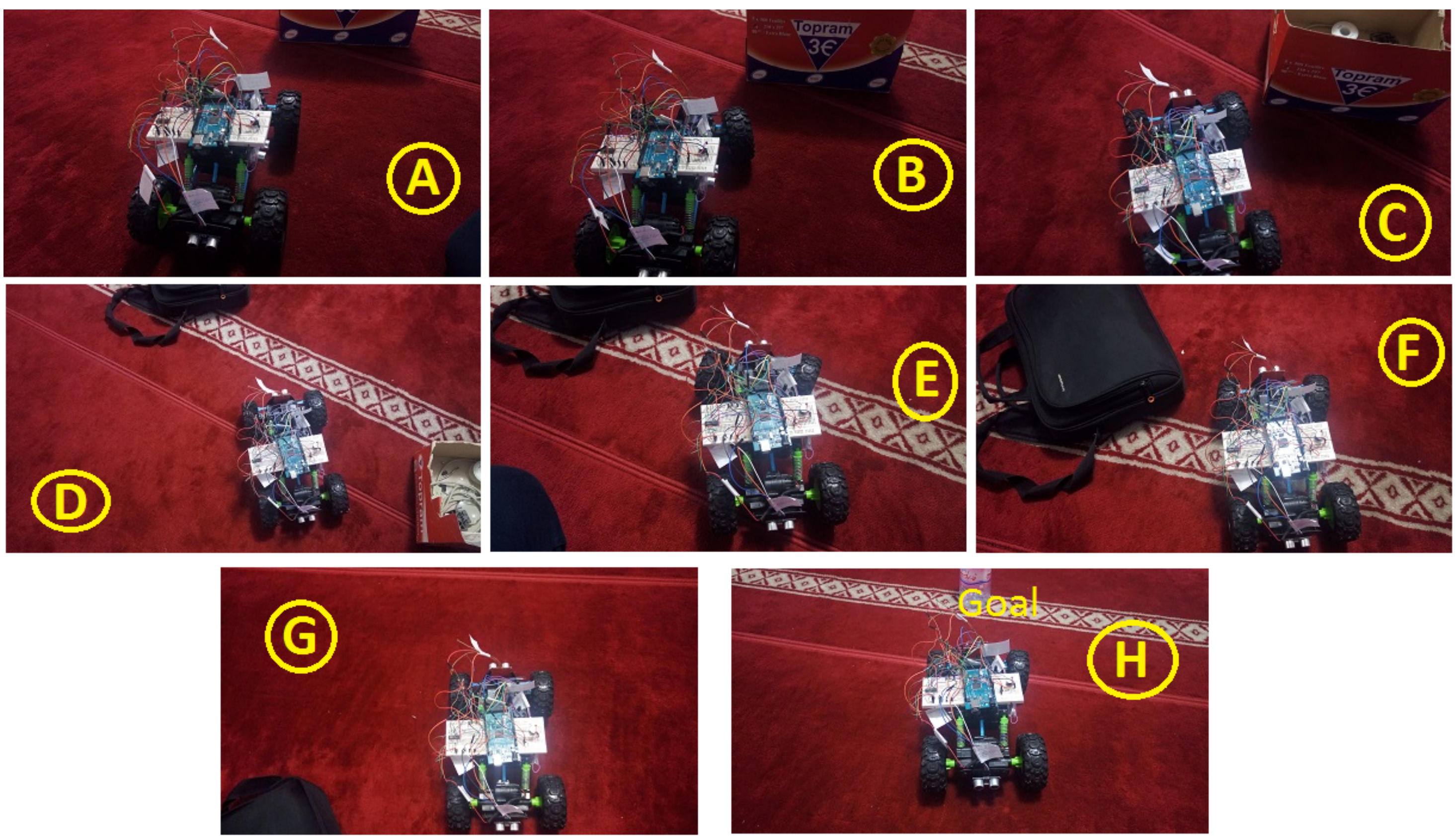

FIGURE 4.12 – Robot executing the received trajectory

4.2.4.2 Measurements Error (Simulation Vs Real Robot)

For the same map, tests were repeated twelve times, let's consider two particular cases :

Best case : for this situation, an approximate drawing has been made representing the robot's real path execution(see figure 4.13), the red line is the real robot and the green squares are the simulated path.

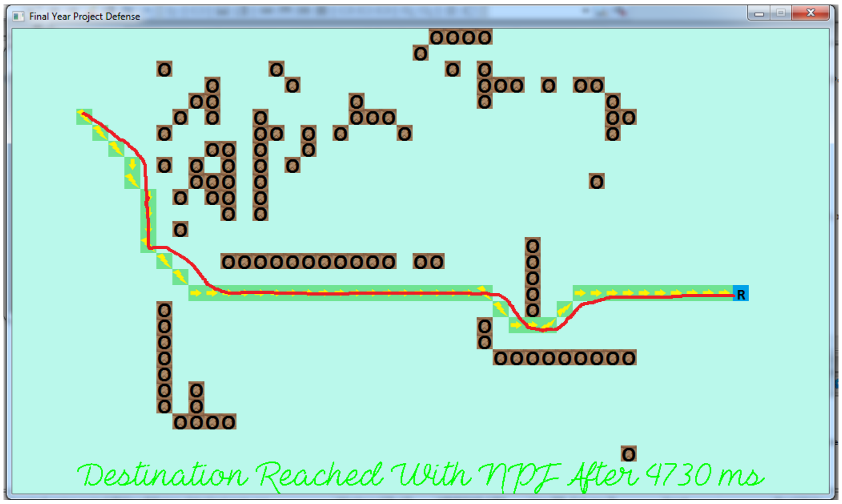

FIGURE 4.13 – Robot Real Trajectory Approximation - Best Case

Worst case : due to some limitations in the hardware, if we have to reconstruct the path for this situation, the results will be as shown in figure 4.14.

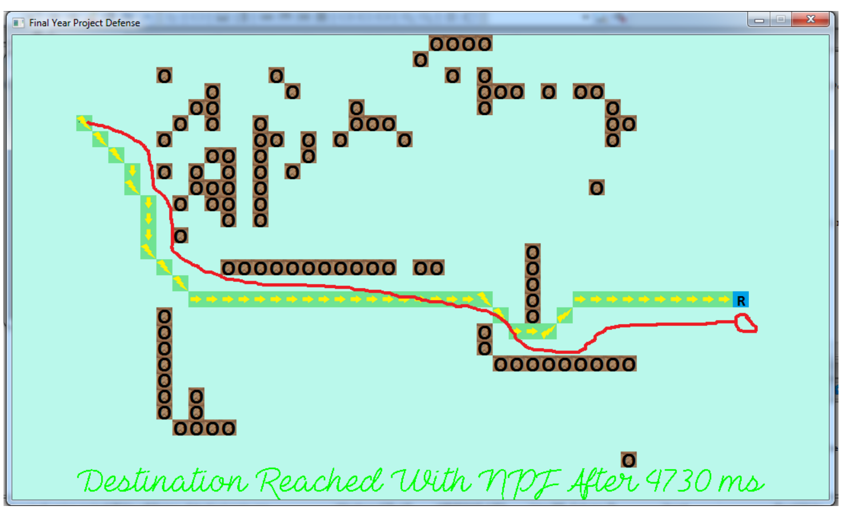

FIGURE 4.14 – Robot Real Trajectory Approximation - Worst Case

Error Origins :

Robot's avoiding constrains : Robot's wheels can turn to a maximum of 30 degrees (right or left), but our simulator assumes a turning of 45 degrees (because we move from cell to cell).

ESP8266 chip counterfeit : the esp8266 introduces some errors in the received path, a filter has been added to Arduino program to minimize the effect, though long use of this chip loses some part of the path.

Slippery ground : the robot is designed to navigate in outdoor environment not indoor, a reason why tests were mostly made on high friction surface.

Stepping Errors : Even if we are moving the robot by 20 centimeters each time(remember : 1pixel=1cm ; and each cell in the software is 20pixel by 20pixel), it is impossible to get this displacement value with 100% accuracy.

4.2.5 Future Expansions

4.2.5.1 Multiple robots control

Single robot control can be extended to multiple robots because the structure of potential field allows us to have such a flexibility, an experimental version has been already started(see figure 4.15).

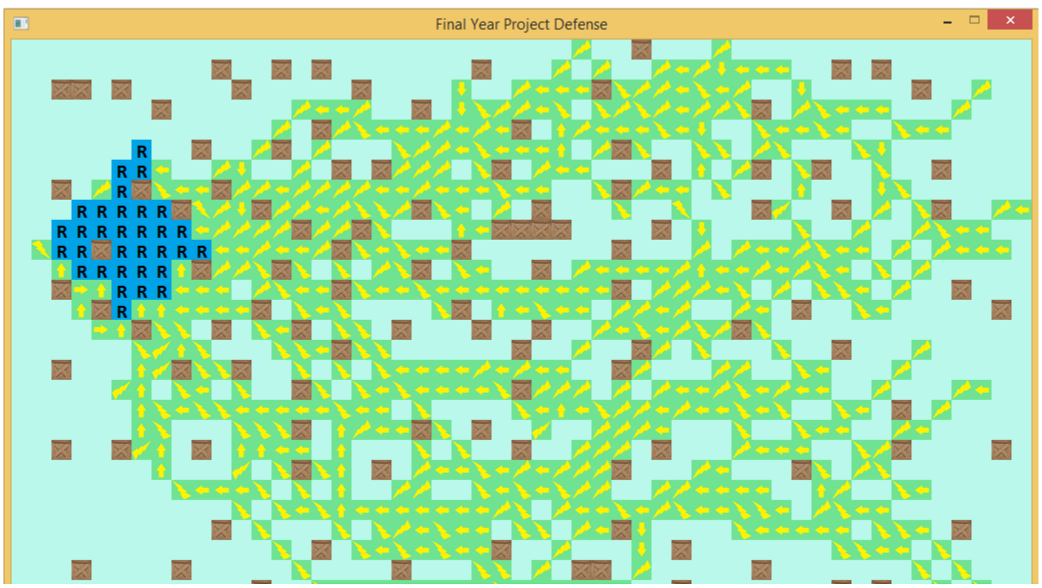

FIGURE 4.15 – Multiple Robots Control Using Potential Field

4.2.5.2 Implementing More Enhanced Algorithms

More robust methods must be also implemented in particular for 3D, an overview of Graph Search algorithms (BFS, DFS and Dijikstra) has been released as a sample of our future work(see figure 4.16).

4.2.5.3 Multiprocessing architecture Execution

Parallel code execution running on multiple processors will be added :

— for C language SDL can already handle this(for thread management `https://wiki.libsdl.org/CategoryThread`).

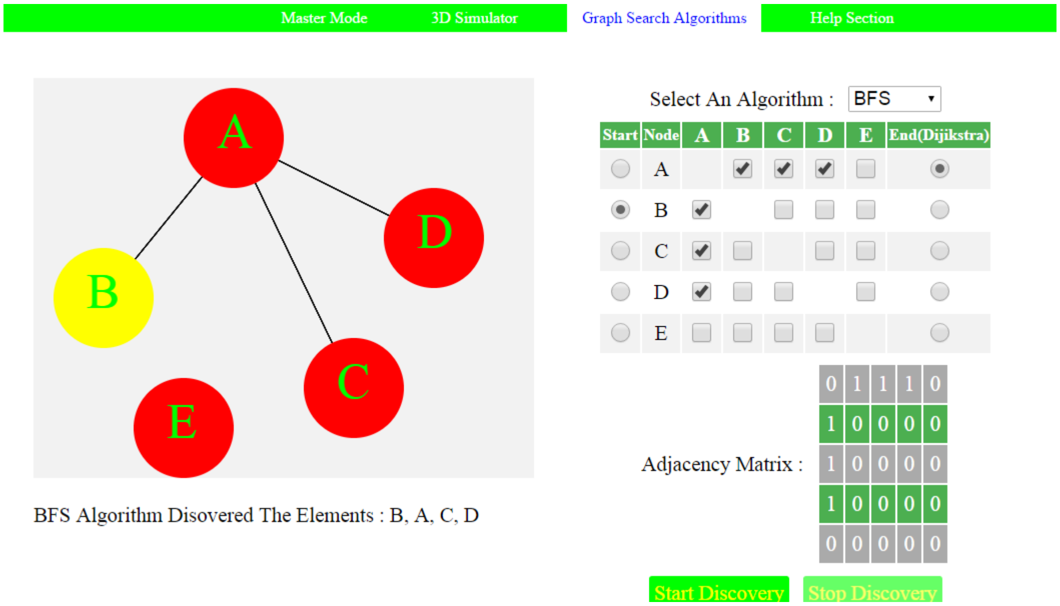

FIGURE 4.16 – Our Implementation of Graph Search Algorithms

— and from Javascript side webworkers can be used for a such purpose (you can dive more in : `http://www.w3schools.com/html/html5_webworkers.asp`), however the 3D will be rewritten with pure C++ (using OPENGL Library).

4.2.5.4 Adding digital certificate

Our program's executable has been uploaded to `https://www.virustotal.com/` in order be scanned with more than 50 different anti-viruses, the result was it was neither malicious nor trusted. In order to fix this issue a digital certificate must be purchased to white-list our application and gain full customer's trust.

4.3 Conclusion

At the end of this chapter, the expected results have been obtained ; another view of path planning and navigation strategy has been proposed. The robot is executing the preplanned path and avoiding the obstacles till it reaches the destination. Dividing the software between the computer and the robot helps to create, maintain, manage the code and increases the user experience (at the computer side). New algorithms can be added easily due to the flexibility of our implementation, the core of the code has been already written.
Our software is secure and hard to reverse(even in the wireless network), a source of trustfulness and fidelity.

Final Conclusion

Path-planning is an important primitive for autonomous mobile robots that lets robots find the shortest (or optimal) path between two points or even ways that minimize the amount of turning, the amount of braking or whatever a specific application requires. Algorithms to find a shortest path are important not only in robotics, but also in network routing, video games and gene sequencing.

Path-planning requires a map of the environment and the robot to be aware of its location in it. In this project, a solution was provided to construct the field (through a GUI « the MAP Editor ») and place the robot, goal and obstacles in user-friendly fashion. After this step, bug0 and Numerical Potential Field can be applied on the map to find a collision free path from an initial to a destination point. Wireless communication(Using WIFI) is important in today's systems, a reason why networking functionalities have been added to our software in order to send the trajectory to the robot. As a last step, the robot will parse the received path and deduce which algorithm must be used and how to move and reach the goal. Moreover, the human can master the robot when necessary.
Software optimization was also taken into consideration, dependency analysis has been shown and fixed to produce cross platform code. Various ways to minify the code have been demonstrated (for C and JavaScript), and target specific applications(32/64 bits) was produced for further execution speed enhancement(no need for backward compatibility).
Finally, security precautions were integrated into the code, many protection mechanisms against various attacks (including reverse engineering) have been discussed. It is a mandatory skill in modern time that we have to master in order to face the *dark art* and withstand their attempts and stop them. Everyone must keep in mind : *there are ghosts in the wires, security is our concern*[37].
The requested task has been completed but this opens new opportunities to generalize the results to *multiple robots and real time controlling systems*. Eventually, camera can be added allowing the human controller to take the full control when precision is needed, target tracking can also be used when the destination is reached(OpenCV is an excellent choice in the professional world).

After this work, some questions should get an answer « *how smart will be future robots ?, Can they be more intelligent than we are ?* ». Renowned physicist Stephen Hawking said at the Zeitgeist [31] conference in London that : « robots powered by artificial intelligence (A.I.), could overtake humans in the next 100 years [32]. When that happens, we need to make sure that computers have goals aligned with ours ». Path planning was the first step that we have taken to understand the robots, more challenges are waiting but as engineers we should always keep our creation's impacts positive on the society, *never create a machine able to rule a man*.

31. Follow the Zeitgeist Movement - London at : `http://www.meetup.com/fr-FR/Zeitgeist-Movement-London/?chapter_analytics_code=UA-12837668-1`
32. Stephen Hawking predicts robot apocalypse coming within 100 years : `http://www.geek.com/news/stephen-hawking-predicts-robopocalypse-in-next-century-1622734/`

References

[1] McGraw-Hill Companies. *McGraw-Hill Dictionary of Scientific & Technical Terms*. McGraw-Hill, 2003.

[2] Wikipedia. https ://en.wikipedia.org/wiki/mobile_robot, May 2016.

[3] Coleman Benson. http ://www.robotshop.com/blog/en/what-types-of-mobile-robots-are-there-3652, Mar 2012.

[4] Kavraki L. E., J.-C. Svestka, P. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 1996.

[5] M.de Berg, M.van Kreveld, M.Overmars, and O.Schwarzkopf. Computational geometry : Algorithms and applications. *Springer-Verlag :Berlin*, 2000.

[6] tutorialspoint. http ://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm, Mar 2012.

[7] Steven Skiena. The algorithm design manual. *Springer. p. 480*, 2008.

[8] C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, 1961.

[9] Even Shimon. *Graph Algorithms (2nd ed.)*. Cambridge University Press, 2011.

[10] Sedgewick Robert. *Algorithms in C++ : Graph Algorithms (3rd ed.)*. Pearson Education, 2002.

[11] Oussama Khatib. Reat-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 1986.

[12] Miguel A. Padilla Castaneda, Jesus Savage, Adalberto Hernandez, and Fernando Arambula Cosio. *Local Autonomous Robot Navigation using Potential Fields*. intechopen, 2008.

[13] Hani Safadi. Local path planning using virtual potential field. apr 2007.

[14] Shmuel Wimer. Bug algorithms. Apr 2011.

[15] Wikipedia. https ://en.wikipedia.org/wiki/d*, May 2016.

[16] Stentz Anthony. Optimal and efficient path planning for partially-known environments. *Proceedings of the International Conference on Robotics and Automation*, 1994.

[17] Stentz Anthony. The focussed d* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[18] Hart P., Nilsson N., and Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Science and Cybernetics*, 1968.

[19] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *Transactions on Robotics 21*, 2005.

[20] S. Koenig, M. Likhachev, and D Furcy. Lifelong planning a*. *Artificial Intelligence Journal 155*, 2004.

[21] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms 21*, 1996.

[22] S. Koenig, Smirnov Y., and Tovey C. Performance bounds for planning in unknown terrain. *Artificial Intelligence Journal 147*, 2003.

[23] Wikipedia. https ://en.wikipedia.org/wiki/rapidly_exploring_random_tree, May 2016.

[24] David Ferguson and Anthony (Tony) Stentz . Rapidly-exploring random trees : A new tool for path planning. Technical report, Computer Science Department, Iowa State University, October 1998.

[25] Matt Greytak. Numerical potential field path planning tutorial. dec 2005.

[26] BELKALEM Jugurtha and ABED Mohamed Amine. Our website : http ://mrobot.netai.net/, May 2016.

[27] David Salomon. *Data Compression The Complete Reference (Third Edition)*. Springer-Verlag New York, Inc., 2004.

[28] Saturnino Salas, Einar Hille, and Garrett Etgen. *CALCULUS ONE AND SEVERAL VARIABLES*. WileyPLUS, 2003.

[29] Rob HAWKES. *Foundation HTML5 Canvas for Games and Entertainment*. friendsof, 2003.

[30] G. Shanker Rao. *Numerical Analysis*. New Age International (P) Ltd., Publishers, 2006.

[31] Chris Anley, John Heasman, Felix Linder, and Gerardo Richarte. *The Shellcoders Handbook : Discovering and Exploiting Security Holes(Second Edition)*. Wiley Publishing, Inc., 2007.

[32] Michael J. Donahoo and Kenneth L. Calvert. *TCP/IP Sockets in C : Practical Guide for Programmers*. Morgan Kaufmann Publishers is, 2001.

[33] Michael Sikorski and Andrew Honig. *PRACTICAL MALWARE ANALYSIS*. No Starch Press, 2012.

[34] Mark Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals (Sixth Edition)*. Microsoft Press, 2012.

[35] Eric Cole. *Hiding in Plain Sight : Steganography and the Art of Covert Communication*. Wiley Publishing, Inc., 2003.

[36] Sean "Frank2" Taylor. Binary obfuscation from the top down. jul 2009.

[37] Kevin D. Mitnick and William L. Simon. *Ghost in the Wires : My Adventures as the World's Most Wanted Hacker*. Little, Brown and Company, 2011.

Appendices

.1 Arduino Mega 2560

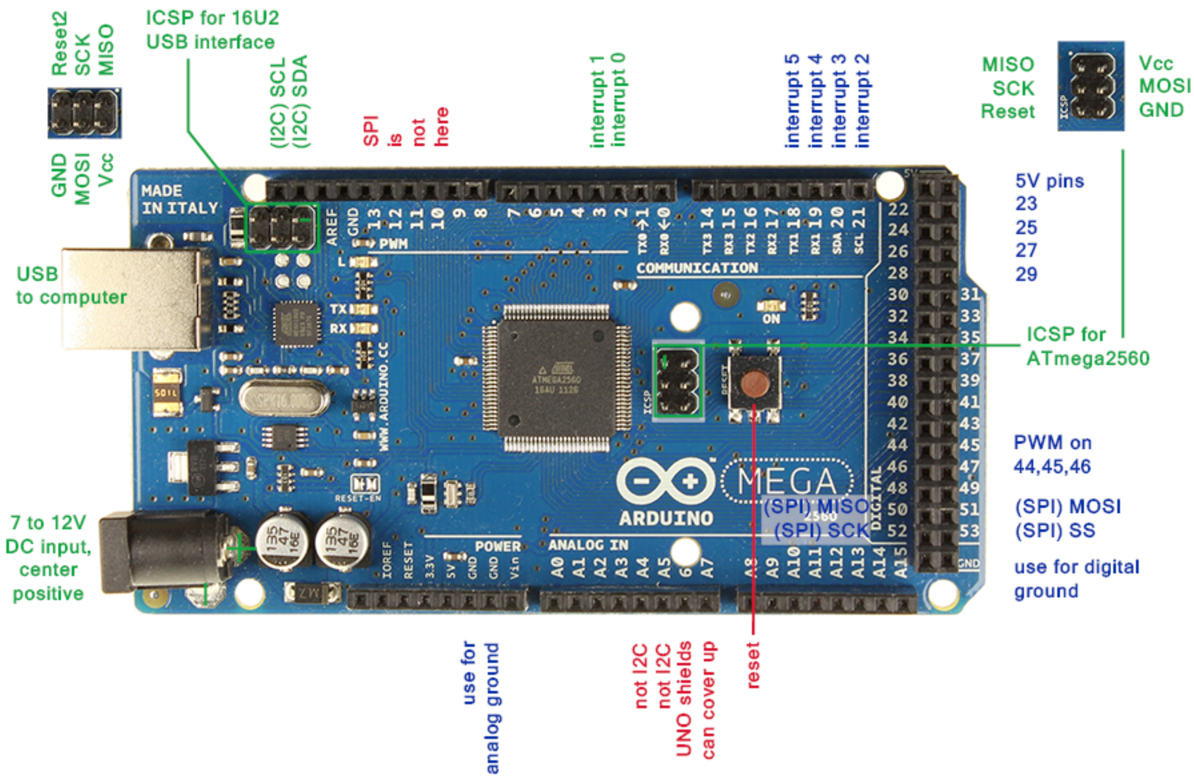

FIGURE 17 – Arduino Mega 2560

- Based on Atmega2560 microcontroller
- 4 UARTs
- 54 digital I/O pins 15 of them are PWM
- Voltage input range(Vin pin) : 7-12V

.2 Dijisktra's Algorithm

Algorithm : A general template for Dijkstra's algorithm.

Input: An undirected or directed graph $G = (V, E)$ that is weighted and has no self-loops. The order of G is $n > 0$. A vertex $s \in V$ from which to start the search. Vertices are numbered from 1 to n, i.e. $V = \{1, 2, \ldots, n\}$.

Output: A list D of distances such that $D[v]$ is the distance of a shortest path from s to v. A list P of vertex parents such that $P[v]$ is the parent of v, i.e. v is adjacent from $P[v]$.

```
D ← [∞, ∞, ..., ∞]                    /* n copies of ∞ */
D[s] ← 0
P ← []
Q ← V                          /* list of nodes to visit */
while length(Q) > 0 do
    find v ∈ Q such that D[v] is minimal
    Q ← remove(Q, v)
    for each u ∈ adj(v) ∩ Q do
        if D[u] > D[v] + w(vu) then
            D[u] ← D[v] + w(vu)
            P[u] ← v
return (D, P)
```

FIGURE 18 – Dijiktra Algorithm Pseudo-Code