DEMOCRATIC POPULAR ALGERIAN REPUBLIC
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
UNIVERSITY OF MOUHAMED KHEIDER-BISKRA
FACULTY OF EXACT SCIENCES, NATURAL SCIENCE AND LIFE
DEPARTMENT OF COMPUTER SCIENCE

Order Number: GLSD/M2/2020

Thesis

Presented to obtain the diploma of academic Master in

Computer Science

Option: **Software Engineering and Distributed Systems**

TRAJECTORY PLANING FOR A MOBILE ROBOT

By:
MEFTAH Nesrine

Defended the ./09/2020, in front of the jury composed of:

....	...	President
Kahloul Laid	MCA	Supervisor
....	...	Examiner

University Year: 2019/2020

Acknowledgements

first of all, I thank God the Almighty, for giving me the health, the courage and the will to study and for allowing me to carry out this modest work in the best conditions.

I strongly thank my supervisor **Pr. KAHLOUL Laid**, for having guided me and for been always available and attentive to my questions Specially with these conditions and epidemic that the world passes by. My deep respect and thanks for his commitment to giving me the opportunity to carry out this project. Your support and reflections on the work throughout the long-lasting process have been priceless.

I never forget to thank all my teachers at Computer Science Department who taught me the basic principles of computer science. Major thanks are also directed to the Head of Department: **Pr. BABAHNINI Mohamed Chawki**.

At last, but not least I would like to thank my parent for being always next to me and helping me with every thing i need, thanks to all my friends for all their support and understanding during this year of my studies.

Abstract

The field of autonomous mobile robots has recently gained the interests of many researchers. Due to the specific needs of different applications of mobile robots systems especially in navigation, designing a real time obstacle avoidance and a trajectory planning in robot system has become the backbone of controlling robots in unknown or known environments. Therefore, an efficient collision avoidance and path planning methodology is needed to develop an intelligent and effective autonomous mobile robot system. This project introduces some techniques for trajectory planning and collision avoidance in the mobile robotic systems. The used technique relies on the use of planning search algorithms, an electronic card "Arduino" adapted to be able to connect to the robot after developing the program in Python language and Arduino software, so that it can be easily used in real-time control applications. The simulation is implemented on multiple environment to show the ability to find the path, time needed to reach the desired goal, detect obstacles, and navigate around them to avoid collision. It also shows that the robot has successfully reached its desired goal and has avoided all obstacles that emerged on its path.

Key words: *Autonomous mobile robotics, Real time control, Trajectory planning, Unknown/known environment, Obstacle avoidance, Planning search algorithms, Arduino card*

Résumé

Le domaine des robots mobiles autonomes a récemment suscité l'intérêt de nombreux chercheurs. En raison des besoins spécifiques des différentes applications des systèmes de robots mobiles en particulier dans la navigation, la conception d'un évitement d'obstacles en temps réel et d'une planification de trajectoire dans un système de robot est devenue l'épine dorsale du contrôle de robots dans des environnements inconnus ou connus. Par conséquent, une méthodologie efficace d'évitement de collision et de planification de trajectoire est nécessaire pour développer un système de robot mobile autonome intelligent et efficace. Ce projet présente certaines techniques de planification de trajectoire et d'évitement des collisions dans les systèmes robotiques mobiles. La technique utilisée repose sur l'utilisation d'algorithmes de planification de recherche, une carte électronique «Arduino» adaptée pour pouvoir se connecter au robot après avoir développé le programme en Python logiciel de langage et Arduino, afin qu'il puisse être facilement utilisé dans les applications de contrôle en temps réel. La simulation est mise en œuvre sur plusieurs environnements pour montrer la capacité de trouver le chemin, le temps nécessaire pour atteindre l'objectif souhaité, détecter les obstacles et les contourner pour éviter les collisions. Il montre également que le robot a atteint avec succès son objectif souhaité et a évité tous les obstacles qui sont apparus sur son chemin.

Mots clés :*Robotique mobile autonome, Contrôle en temps réel, Planification de trajectoire, Environnement inconnu / connu, Évitement d'obstacles, Algorithmes de recherche de planification, Carte Arduino*

Contents

List of Tables

List of Figures

List of Algorithms

General Introduction

Nowadays, human activity in many fields is supported or replaced by robots, ranging from simple robots used for industrial applications to complex autonomous robots for space exploration. A reason for this diffusion is the excellent versatility and flexibility of robots, which make them suitable to perform different tasks.

Mobile robots are widely used in industrial environments for transporting products for example. Most often these tasks are repetitive and follow a well defined path, sometimes even well materialized like lines on the ground. Such a framework of use requires that the robotic system has a minimum level of autonomy and navigation facilities. To do this, the system generally has to accomplish three basic tasks which are location, planning and navigation. Mobile robotics aims to make an autonomous system in its movements. A robot, endowed with capacities of perception and information on its environment, must to be able to move independently, without getting lost and while avoiding obstacles. One of the tasks to be accomplished by is to plan its trajectory in the environment. The path planning for a mobile robot is an essential element of various significant fields, such as video games, robots, and wireless sensor networks. The aim of the problems that we can see in path planning for mobile robots is to calculate a feasible path from the starting point to the target point in a work-space, and how to control this mobile robot in real time after finding that path.

For a robot A evolving in a given environment W, the general planning problem consists to be determined for A a movement allowing it to move between two given configurations while respecting a certain number of constraints and criteria. These arise from several factors of various kinds and generally depend on the characteristics of the robot, the environment and the type of task to perform. In this case, the constraints relating to the robot concern its geometry, its kinematics and its dynamics and their consideration can be complex depending on the initial architecture considered. This architecture that can correspond to an articulated system of rigid objects such as a manipulator arm, a hand with several fingers or a vehicle towing trailers, or even several robot systems with coordinate such as manipulator arms or car-type mobile robots moving in a network road. The constraints emanating from the environment mainly concern the non-collision of obstacles fixed bulky W and taking into account contact interactions with the robot. Obstacle avoidance depends on the geometry of the environment and is common to all robotic tasks.

Thus planning a trajectory for a solid body in Cartesian space amounts to planning the trajectory from a point in the work-space[25]. If we now consider the obstacles of the environment, we realize that putting them into the space of system configurations is not an easy task, especially if the system has many configuration variables and therefore the dimension of space is high. After this brief presentation of the problem of trajectory planning, several questions remain unanswered:

- How to express the obstacles in the space of configurations
- How to find a way in free space.
- How to control a mobile robot in reel time to get throw the finding way.

In order to answer theses questions we made this thesis to see same algorithms that can help us to find the optimal path after expressing the obstacles, and to control a mobile robot in real time we will use a micro-controller called Arduino to determine the behavior of this robot.

This work is then organized into four chapters with 2 parts as follows:
For the first part which is the state of the art, we have two chapters as follow: The first chapter presents the trajectory planning in artificial intelligence and introduced some search algorithms.
The second chapter presents some generalities concerning robotics and mobile robots.
For the second part, it contains the chapters for the design and implementation of a system for the control of a mobile robot based on Arduino. We have two chapters as follow:
The third chapter present three path finding search algorithms and there implementation and tools for the simulation application development.
The fourth chapter deals with the practical part to make a car mobile robot, introduces the robot components and tools to use.

This thesis ends with a general conclusion where the emphasis will be on the contribution of suggested techniques for solving planning, ordering and controlling problems by using a set of techniques to find our optimal path and send the commands that's goes by a set of steps to control the mobile robot behavior.

Part I

State of the art

Chapter 1

Trajectory planning in artificial intelligence

1.1 Introduction

In this chapter, we present the general planning problem and review the main resolution approaches and methods proposed in the literature and relevant to the problem addressed in this manuscript, this will include, in addition to purely geometric techniques, a basic concepts, a summary of the work major in movement planning in the presence of holonomic and dynamic constraints. Next we present the theory of optimal graphs and paths, also we present the main search algorithms for the shortest path .

1.2 Basic concepts

We present the essential principles of trajectory planning .

1.2.1 Path and trajectory

Most of the trajectory planning work is based on the concept of space configurations of the robot introduced in [25] at the beginning of the 80s. A configuration denotes the set of parameters characterizing in a unique way the robot in its environment or work-space. The set of robot configurations is the configuration space Q, which has a differential variety structure. We denote by n the dimension of Q. A trajectory is a continuous function of $[Tinitial, Tfinal] \subset R$ in Q which has any value $t \in [Tinitial, Tfinal]$associates a configuration:

$$q : [Tinitial, Tfinal] \rightarrow Q$$

$$t \rightarrow q(t)$$

A trajectory is said to be admissible if it is a solution of the system of differential equations corresponding to the kinematic model of the robot, including the constraints on the controls, for initial and final conditions given. A path is the image of a path in Q. A path admissible is the image of an admissible trajectory[10].

1.2.2 Trajectory planning

The target of the trajectory planning is to generate the reference inputs for the manipulator control system that ensures the implementation of the desired movement, so we call trajectory planning, the calculation of an admissible trajectory and collision-free for a robot between a given start and end configuration[10].

1.2.3 Reactive obstacle avoidance

In this part, we make a brief state of the art of the main reactive avoidance methods obstacles in order to highlight their advantages and their limits in relation to the specifics of our problematic.

1. **Analytical methods**

   Even if there is no obstacle, ordering a non-holonomic system to bring it from an initial configuration to an arrival configuration is not an easy task. Indeed, there does not exist today general algorithm allowing to solve the problem for any system not holonomic. Analytical methods are known only for certain classes of systems. For the others, we only have numerical methods.

   In addition, the presence of obstacles makes analytical methods inapplicable to systems that are not holonomes[36].

2. **Cell Decomposition methods**

   A first approximate method for determining a collision-free path is based on the cellular breakdown of the environment [3]. It consists in partitioning the space of free configurations of the robot into a set of adjacent connected regions. Different techniques of space decomposition exist. We can cite for example the partition of Voronoı [8] or the visibility graphs [7] The decomposition obtained is then captured in a connectivity graph whose nodes correspond to the different regions and the arcs to the adjacency relationships between them. The problem of finding a path in the configuration space is then replaced by the problem of finding a path in a graph, problem which can be easily solved by classical algorithms based on dynamic programming[37].

However, these methods are based on the structuring of the configuration space and the modeling. A priory of its Interconnecting. In addition, the grid discretizations of the workspace, proposed by these methods, limit the space of admissible trajectories.

3. **Potential Fields**

The potential field methods for robotics navigation, originally proposed by [20] for a manipulator arm, consists in assimilating the robot to a particle constrained to move in a field of fictitious potential obtained by the composition of a first attractive field (reach the desired configuration) and a set of repulsive fields modeling the presence of tackles in robot space. At each position of the robot, a force resulting from the combined action obstacles and goal is calculated. It corresponds to the direction to follow. Many adaptations of this technique have been proposed. We can for example quote [4], which calculates a direction of movement from proxy information metric. It should be noted, however, that these purely reactive methods are subject to minimums. In addition, they can cause the robot to oscillate in certain situations (narrow passages for example).
These problems can be solved in different ways. It is for example possible to trigger a particular behavior when you meet such a minimum (random displacement, followed by walls) [2], [18], [38]. He is also possible to impose that the function representing the field of potential is a harmonic function [21], [34], which guarantees that there is no local minimum, but it greatly complicates its calculation, making its implementation difficult.

4. **Dynamic window method**

This technique proposed in [13] works in the robot control space. The size of the search area for accessible speeds (that is to say, not causing collisions) is reduced by taking explicit account of the kinematic model of the system. Orders sent to the robot are the result of the maximization in this area of research of a cost function related to the final configuration. The use of this method is very interesting for a moving robot quickly or for a robot with limited acceleration and deceleration capabilities. She permits then to produce a safe and regular robot movement. Its extension to the multi-robot framework is however very delicate due to its lack of flexibility.

5. **Rubber band**

The concept of elastic band has been proposed in [33] for mobile platforms. It has been extended to mobile manipulators in [6]. The goal is to distort locally a given trajectory to take into account possible obstacles in real time. This initially planned trajectory is represented by a series of adjacent balls in the space of configurations. The radius of a ball centered in a configuration is the distance from this configuration to the nearest obstacle. Thus a trajectory is without collision when the balls which compose it overlap. Developed for systems without kinematic constraint, this technique considers the trajectory like an elastic band changing under the action of repulsive forces generated by obstacles and internal forces of contraction or elasticity.

This technique has been extended to a car-type robot by [19]. However, this technique can only be applied at the cost of approximations to the shape of the trajectories (combinations arcs of circles and straight lines), which makes navigation in a highly constrained.

The presentation of these different methods, their hypotheses and their limitations makes immediately show their inadequacy to a flotilla of robots evolving in unknown environments strongly constrained. This motivates the development of a reactive obstacle avoidance method. for non-holonomic systems.

1.3 Summary of the main work in movement planning

Historically, the planning problem was initially addressed as part of a system of robots moving in an environment containing fixed obstacles and subject only to the constraint of non-collision. We will present some of work methods in planning work, most of these works are based on the concept of space configurations of the robot introduced by Lozano-Perez [25]. So we will see cellular decomposition and retraction-type resolution methods who called the global methods and next the local ones.

1.3.1 Cellular decomposition methods

A first approach for path planning is known as the decomposition approach cellular. It consists in partitioning the space of the robot's free (or working) configurations into one set of adjacent related regions. The description of the decomposition obtained is then captured in a connectivity graph whose nodes correspond to the different regions (or cells) and the arcs to adjacency relationships between them. The problem of planning a movement between two located configurations initially in two different cells is solved in two steps:

1. exploration of the connectivity graph and determination of a path connecting the cells containing the two initial configurations of the graph
2. search for the solution to the planning problem from the envelope defined by the list of adjacent cells found in step 1.

Due to its generality, the decomposition approach has been adopted in several works and has led to the implementation of many methods. These are generally classified into two categories the exact methods and approaches and are distinguish by the decomposition models they use and their completeness with regard to the resolution of the planning problem.

1. **Exact methods**

   The first category groups together the so-called exact resolution methods in the sense that the decomposition performed is based on an exact covering of the robot's free space of its contacts in convex cells. The completeness and the capacity of this type of method to provide a response to the resolution of the problem General planning have been demonstrated by Schwartz and Sharir [35]. The problem planning is formulated in algebraic form by considering a decomposition of free space of the robot in the form of semi-algebraic cylindrical components (Collins cells) [41]. By this formulation, the authors arrive at one of the important results on the complexity of planning, that it is polynomial in the complexity of the environment and doubly exponential in it dimension of the configuration space. The complexity of the environment is measured by the number and the degree of polynomials used for the semi-algebraic description of free space. It is important to note that this decomposition free space is a so-called exact method, which should be distinguished from non-exact methods.

2. **Approaches**

   In order to remedy this complexity, so-called approximate methods are generally applied. They are distinguished from the previous ones by the simplicity of the structure of the cells used for the decomposition and the approximation of the robot's free space. The structure of it is usually captured in a hierarchical representation in identical elementary cells and allowing the adaptation of the size of these to the geometry of the areas to be covered. Several types of cellularization of robot space are generally used in practice: representation in octrees, slicing, or even in polyhedral[41].

By comparing the two decomposition approaches, it emerges that the exact methods are more complete in theory than approximate methods. However, the complexity of their implementation, makes them less suitable in practice than the techniques approached even if these sometimes remain limited to small spaces. This finding is not general since it appears in the route planning using set methods that an exact method can lead in the plane case and for a polygonal modeling of the robot and its environment to an efficient implementation.

1.3.2 Retraction-type resolution methods

A second major approach to solving the planning problem is to bring back the research of the robot movement in a space smaller than that of the admissible space. That consists in representing the connectivity of the robot's free space by a network of one-dimensional curves can be entirely in the free space or contacts of the robot. Movement planning between two given configurations is then resolved in three stages: determining a path bringing the robot back of its initial configuration at a point located on one of the curves of the connectivity network, determination a path between the final configuration and the network, and finally exploration of the latter in order to extract one path connecting the two points connected to the initial and final configurations. A first method based on this concept was proposed by Nelsson for a two-dimensional work-space cluttered with

obstacles polygonal [31]. In this method, the network is described by a graph, called visibility graph, where the nodes correspond to the vertices of the obstacles and the edges to line segments connecting these summits in free space. The disadvantage of this method is that the solutions obtained lead to the robot to be in contact with obstacles at some of their points (the vertices) even in the presence of a solution entirely contained in free space. In order to remedy this, a second method, known as name of retraction and developed by Ô'Dùnlaing and Yap [28], consists in using a Voronoi diagram [42] to capture the connectivity of free space and generate solutions most distant from obstacles. Such a method, although it is general in theory, remains limited to small spaces. Another variant of this method is developed by Brooks for polygonal work-spaces and is based on the representation of these by generalized cylinders. The paths to be followed by the robot are then determined by considering the connectivity between the axes of these cylinders.

Finally, a last retraction type method was presented by Canny in the context of the solving the general planning problem [42]. The proposed algorithm is exponential in the dimension of the configuration space, and constitutes one of the most important results on the complexity of the motion planning problem.

1.3.3 Local methods

The methods exposed in the preceding paragraphs are generally said to be global knowing that they are all based on the structuring of the configuration space and the a priory modeling of its connectivity. In order to avoid the complexity of such a structuring step, other so-called local methods are generally used. Their principle consists in determining the movements of the robot by not considering as a local representation of the environment and to perceive movement planning as an optimization problem.

1. **Potential method**

   A first method consists in assimilating the robot to a particle constraint to move in a field of fictitious potential obtained by the composition of a first field attractive to the goal and a set of repulsive fields modeling the presence of obstacles in space of the robot. The displacements of this one are then calculated iterative by an algorithm of descent of the gradient of the potential obtained. If a potential type method can be easily implemented and applied in real time for manipulation or navigation tasks of a mobile robot, it remains sensitive to the occurrence of local minimums generating blocking or oscillation configurations.These minimums are generally linked to the geometry and the distribution of obstacles in the robot's work-space and especially to the penalty coefficients associated with them during the construction of the potential field. In order to remedy these problems, Barraquand and Latombe propose in [3] to integrate the minimization of the potential applied to the robot by exploring a hierarchical bitmap representation of the configuration space. The avoidance or rather the exit of a local minimum is carried out by the application of random movements, called Brownian.

Finally, the method can provide local solutions to the problem of minimization of the functional considered is sensitive to the position of the curve chosen initially before the start of the research. This problem presents a method combining variational calculus and a dynamic programming technique for finding paths that maintain a safe distance from obstacles.

2. **Constraints method**

As part of a local approach, Faverjon and Tournassoud propose a method, called constraints, which addresses the problem of obstacle avoidance by locally modeling each one of them by the whole of its tangent planes [12]. The generation of robot movements is obtained by minimizing a quadratic criterion on the task to be performed in the presence of constraints linear associated with the equations of the planes tangent to the obstacles. Unlike the potential method, the obstacles act on the robot during the minimization process only when it is very close to it and has tendency to enter it. However, the method remains sensitive to the presence of local minima. This issue is approached in [12] by combining the local method with a technique of learning.

1.4 The theory of optimal graphs and paths

In this part, we will explain why we use graphs to get the optimal path and we take as an example the Tas structure and some of the problems that we face to find that path.

1.4.1 Importance of graphs

Graphs represent a powerful instrument for modeling many Coimbatore problems, which would otherwise be difficult to approach by conventional techniques such as mathematical analysis.

In addition to its existence as a mathematical object, the graph is also a powerful data structure for computing.

Graphs are irreplaceable when it comes to describing the structure of a complex whole, by expressing relationships, dependencies between these elements. Examples are hierarchical diagrams in sociology, genetic trees, and diagrams of succession of tasks in project management[22]. Another large group of uses is the representation of connection and routing possibilities: determination of any connected network (electrical, water supply), calculation of shorter path in a road network, Graphs are finally precious for describing dynamic systems, that is say that evolve over time: transition diagrams, finite state automata, Markov chains. There are two families of graphs: oriented and non-oriented, that is to say that the relationships between the elements of a set are oriented or not.

A graph is defined by a couple $G = (X; U)$ of two sets:

- X is a set $(x_1, x_2, ..., x_N)$ vertices, also called nodes
- $U = (u_1, u_2, ..., u_M)$ is a family of ordered pairs of vertices called edges.
  A graph can be valued, or weighted, it is then provided with weight or cost on its edges.

1.4.2 Tas structure

Let us consider a set H, of which every element x, has a numerical value $W[x]$, and in which direct debits only concern the smallest element. A simple solution is to look first sample what is the small element, which costs $O(k)$ if H contains k elements. The heap is a structure powerful data which makes it possible to carry out these minimum withdrawals and the addition of any elements in just $O(\log k)$. We will use it to accelerate shortest path calculation algorithms in a graph.

The TAS structure (binary) [22] tabulated object that can be seen as a binary tree almost complete. Each node of the tree corresponds to an element of the array which contains the value of the node. The tree is completely filled on all levels, except sometimes the lowest one which is filled in from the left, to a certain point. An array A which represents a heap is an object with two attributes: $Longueur[A]$, which is the number of elements in the array, and $Taille[A]$, which is the number of elements of the pile stored in table A. In other words, although $A[1..Longueur[A]]$may contain numbers valid, no element after $A[taille[A]]$, where $Taille[A] \preceq Longueur[A]$, is an element of the heap.
The root of the tree is $A[1]$, and knowing the index i of a knot, the indices of his father $Pere(i)$, of his left son $Gauche(i)$, and his right son $Droit(i)$ can be easily calculated.
The following algorithm 1.1produces a TAS from an UN-ordered array.

```
Entrée(A)
Sortie(Tas(A))
begin
    taille[A] :=longueur[A]
    for i ← ⌊ longueur[A]//2⌋ à 1 do
        l ←Gauche(i) ; r ←Droit(i) ;
        if l ⩽ taille[A] et A[l]>A[i] then
            max ← l ;
            else max ← i ;
            if r ⩽taille[A] et A[r]>A[max] then
            max ← l ;
            if max ≠ i then échangerA[i]↔ A[max] ;
            i ← max
        end
    end
    return Tas(A)
end
```

Figure 1.1 – Construction of a TAS from an unordered array.

1.4.3 Optimal path problems

Optimal path problems are very commo n in practical applications. We meet them as soon as it is a question of routing an object between two points of a network, so as to minimize a cost, a distance or duration. They also appear in combined sub-problems, in particular the flows in the graphs and schedules. All this motivated the search for efficient algorithms very early on.

1.4.4 The main types of problems

Let us consider a valued graph $G = (X, A, W)$. X denotes a set of N vertices (or nodes) and A a set of M edges. $W(i, j)$ also noted $W_{i,j}$, is the valuation (also called weight or cost) of edge (i, j), for example a distance, a transport cost, or journey time. For the most economic function widespread, the cost of a path between two vertices is the sum of the costs of these edges. The problems associated consist in calculating paths of minimum cost (in short minimal paths, or shorter paths). They make sense if G does not have a negative circuit, otherwise we could infinitely decrease the cost of a path by turning in such a circuit, called for this reason absorbing circuit [9]. Good of course we can also look for paths with maximum values. In the absence of an absorbing circuit, we can limit the search for the shortest paths to only elementary paths, that is to say not passing not twice by the same vertex. Indeed, if a path follows a circuit, we can remove the portion going through the circuit without increasing the cost. The problem of finding an optimal path in the presence of absorbent circuits exist, but it is NP-difficult. There are other economic functions than the sum of edge costs, but they are less common in applications.
The literature distinguishes three types of problem, which we note 1, 2 and 3:

1. Shortest single-origin path: given a starting summit s. Find a shorter s path to any other peak.
2. Shortest path for a couple of vertices: given two vertices s and t. Find the shortest path from s to t.
3. Shortest path for any couple of vertices: find a shortest path between any couple vertices, i.e. compute a matrix $N \times N$ called distancer.

1.5 Shortest path search algorithms

Most of the shortest path search algorithms calculate a label for each vertex x $V[x]$, value of the shortest paths from the starting vertex to the vertex x. This value represents at start an estimate by excess (increasing) of the value of the shortest paths. Some algorithms definitively process a vertex at each iteration: they select a vertex x and calculate the final value of $V[x]$. These algorithms are said to be fixed to labels[22] and are represented by the Dijkstra algorithm and its derivatives. Other algorithms can refine up to the last iteration the label of each vertex. They are called label-correcting algorithms[22]. The Bellman algorithm, fifo, D'Esopo and Papeis, floyd a well-known label correction algorithms.

1.5.1 Label fixing algorithm

We present some of fixing algorithms that uses to find the shortest path.

1. **Djikstra algorithm**

Dijkstra's algorithm (see 1.2)[9] is only valid for graphs with positive valuation or null, which do not contain negative circuits. At each iteration, a vertex x receives its final label, it is said to be fixed.
The main iteration selects the minimum label vertex x among those already reached by a path provisional original s. For any successor y of x, we see if the path passing through x improves the path already found from s to y: if yes we replace $V[y]$ by $\min(V[y], V[x] + W(x, y))$ and we memorize that we succeed in y via x by setting $P[y] := x$.
If all the vertices are accessible from s, the algorithm proceeds in N iterations. In practice, summits may not be accessible. If we want to calculate only a shorter path from s to an other vertex t, just stop the algorithm as soon as t is closed, for example by modifying the end test in "until ($V \min = +\infty$) or $x = t$".

```
Entrée(G, list(Adj[G]), s)
Sortie(ℓ)
begin
    R ensemble de tous les noeuds ;
    Initialiser le tableau V à +∞ ;
    Initialiser le tableau P à 0 ;
    V[s] := 0
    P := s while R ≠ ∅ do
        u :=Extraire-Min(V) ;
        P := P ∪ u ;
        R := R − u ;
        for All Noeud v voisin de u do
            if V[v] > V[u] + W(u, v) then
            V[v] := V[u] + W(u, v) ;
        end
    end
    return (ℓ)
end
```

Figure 1.2 – DJIKSTRA algorithm

The major drawback of Dijkstra's algorithm is that it is insensitive to the density of the graph. The number iteration while loop, at most N, cannot be improved by constructing the algorithm. In however most of the work is due to the internal loop finding the next vertex i to fix. This loop costs $O(N)$. This suggests using a heap to get x faster.

In the case of related graphs, the most common in practice, we have $M \geq N-1$. Dijkstra's algorithm with heap is then in $O(M. \log N)$and it is therefore advantageous for sparse graphs. If all the arcs possible exist ($M = N^2$), it is therefore as expensive as the version without heap.

2. **Sedgewick and vitter algorithm**

This algorithm was designed for non-oriented graphs whose vertices are points of a space Euclidean, and the edges of the segments between the points valued by the Euclidean length of segment (distances between points). It is designed for problem A, that is, calculates a shorter path between two vertices s and t. Its general structure is similar to that of Dijkstra's algorithm.

3. **Bucket algorithm**

Bucket algorithms are interesting variants of Dijkstra's algorithm when costs arcs are whole and their maximum U is not too large. We partition the range of values of labels in B intervals of common width L, numbered from 0 to $B-1$, and each is associated with a set of peaks called bucket. This system can be coded as an array of B lists. To find the vertex x to be fixed in the buckets Dijkstra algorithm, we first seek the non-empty bucket more small clue k. We then scan the latter to locate and extract the minimal label vertex x. For each successor y whose label can be improved: we search for the bucket of y, we scan it to locate and remove y, we modify the label of y with $(V[y] := V[x] + W(x; y))$, we locate the new bucket of y and Finally, we insert y at the head of this bucket. The vertices in practice are well distributed and the buckets lists are short, which gives very good average performance.

1.5.2 Label correction algorithm

Now we present another type of algorithms that called fixing algorithm that uses to find the shortest path.

1. **Bellman algorithm**

This label correction algorithm was designed in the 1950s [9] by Bellman and Moore. It is intended for any valuations and can be adapted to detect a cost circuit negative. This is a dynamic programming method, i.e. recursive optimization, described by the following relation:

$$\begin{cases} V_0(x) = 0 \\ V_0(x) = +\infty, y \neq s \\ V_k(x) = \min_{s \in T^{-1}(y)}\{V_{k-1}(x) + W(x,y)\}, k > 0 \end{cases}$$

$V_k(x)$ denotes the value of the shortest paths of at most k arcs between the vertex s and the vertex x. Both first relationships are used to stop recursion. The vertex s can be considered as a path of 0 arc and zero cost. The third relation means that an optimal path of k arcs from s to y is obtained from optimal paths from $k-1$ arcs from s to any predecessor x from

y. Indeed, any optimal path is formed of optimal portions, otherwise we could improve the whole path by replacing a portion not optimal with a shorter portion.
The recursive formulation being not very effective, one calculates in practice the table V iteratively, for the values increasing from k. We give below a simple algorithm(see 1.3). The labels at the end of step k are calculated in a new table from the V of the labels available at the start of the step. For any vertex y, we see if $V[y]$ can be improved by coming from a predecessor of y. At the end of the stage we overwrite V with the new table and we go to the next step. In the absence of an absorbing circuit, we can restrict ourselves to the paths elementary to find a shorter path from s to any other vertex. Now, such a path has no more of $N-1$ arcs. The labels are therefore stabilized in at most $N-1$ iterations. In practice, they can be stabilize earlier, and a better end test is when $V_k = V_k - 1$. The complexity is in $O(N; M)$: there are at plus $N-1$ main iterations, consisting in consulting the predecessors of all the summits, that is to say the M arcs.

```
Entrée(G, s)
Sortie(ℓ)
begin
    R ensemble de tous les noeuds ;
    Initialiser le tableau V à +∞ ;
    V[s] := 0
    for i=1 jusqu'à Nombre de sommets de G do
        for All lien (u, v) du G do
            if V[v] > V[u] + W(u, v) then
                V[v] := V[u] + W(u, v) ;
            end
        end
    end
    for All lien (u, v) du G do
        if V[v] > V[u] + W(u, v) then
            return Faux
        end
    end
    return (ℓ = P)
end
```

Figure 1.3 – Bellman algorithm

2. **Fifo algorithm**

This simple label-correcting algorithm examines vertices in FIFO order using a queue Q of vertices. At the beginning, only s is in Q. A main iteration processes all the vertices

present in Q at the start of the iteration. Then the algorithm scans the successors of the vertices and places them, those whose the improved labels at the end of Q. The algorithm ends when Q is empty. In fact, the FIFO algorithm is a derivative of the Bellman algorithm, very interesting because it does not use the predecessors.

3. **Desopo and popes algorithm**

   This algorithm uses a Next queue-queue, it is a queue where we can add an element at the end (En Tail) or head (Push). As in FIFO, a peak is reached the first time and placed at the end of the queue. On the other hand, if we revisit it, we insert it at the head of the queue (provided that it is not already in the queue). This criterion management heuristics can be explained intuitively. When you reach a peak at the same time, it is not urgent to develop his successors because the paths obtained may be bad at first.
   On the other hand, a summit already visited and whose label has just decreased must be developed as a priority for spread the improvement.

4. **Floyd's algorithm**

   Floyd's algorithm calculates a distance $N \times N$ giving the values of the shortest paths between any couple of vertices. For this algorithm the table$V$ of the labels becomes a matrix $N \times N$, $V[i, j]$ denoting the cost of the shortest paths from i to j.

1.6 Conclusion

In this chapter we have presented same basic concepts, summary of the main work in movement planing, the theory of optimal graphs and paths and also some search algorithms for shortest path. As well we can see that most of the algorithms seen previously consist in scanning a search space (containing a infinitely uncountable of points) in order to find an optimal solution. A punctual approach allows us to analyze only a tiny portion of this search space. The latter can frequently be covered by a finite number of simple subsets, on which we know how to calculate, and which contain an infinity of points.
In the next chapter we will try to present some general information about robotics and there components we detail mobile robotic with wheels, its configuration, location, types, uses, and some advantages and inconveniences.

Chapter 2

Robotics

2.1 Introduction

Today, robotics is the art of automating more or less complex systems but based on the know-how acquired by studies on the design of robots. Know-how stemming from developments in a branch of general automatic. In this chapter we will present some concept about Robot, Robot components, and information about mobile robots with wheels such as its localisation, area of use, same of the advantages and inconveniences of it.

2.2 Basic Concept

Czech writer , Karel Capek, in his drama[29], introduced the word robot to the world in 1921. It is derived from the Czech word robota which means "forced labor". Isaac Asimov, the Russian science fiction writer, coined the word robotics in his story "Dress Up", published in 1942, to refer to the science of studying robots.
Before defining what is a robot we will cite the three laws which have been developed by Isaac Asimov, and which are governing the behavior of a robot.

1. **The three laws of robotics**

   - A robot cannot injure a human or, by its inaction, allow a human to be injured.
   - A robot must obey orders given by humans, unless such orders are in contradiction with the first law.
   - A robot must protect its own existence as long as such protection does not contradict either the first and / or the second law [44].

2. **Robot**

It is a machine that can manipulate objects by performing various movements dictated by an easily modifiable program.
Programming a robot consists first of all in specifying the sequence of movements it will have to perform.

Some robots have "sense", that is to say of a more or less important set of measurement and appreciation instruments camera, thermometer, rangefinder, ect) allowing the robot program to decide which movement is best suited to external conditions. For example if a mobile robot equipped with a camera ut has to move in an unknown room, it can be programmed so that it circumvents any obstacle which would obstruct its path.
We are also trying to equip robots with an artificial intelligence device so that they can face unforeseen and new situations (the robot could gain some "experience").

2.3 Robot components

A robot, as a system, consists of the elements, which are integrated together to form a whole. The most by robots contains the following:

1. **Manipulator**

It is the main body of the robot as that includes the joints, and other structural elements of the robot. It should be noted here that the manipulator alone is not a robot [30], (see 2.1 for a manipulator example like arm).

Figure 2.1 – A manipulator arm

2. **Final Effector**

This part is connected to the last junction (hand) of a manipulator which generally manages objects, establishes connections to other machines or performs the required tasks [30].

3. **Actuators**

The actuators (see 2.2)are the "muscles" of manipulators. The controller sends signals to the actuators, which, in turn, move the robot joints and junctions, the common types of actuators are servo motors, stepper motors, pneumatic actuators, and hydraulic cylinders. The actuators are under the control of the controller [30].

Figure 2.2 – Different actuators of a robot

4. **Sensors**

The sensors (2.3) are used to collect information on the internal state of the robot or to communicate with the external environment. As in humans, the robot controller must know the location of each link of the robot in order to know the configuration of the robot. Always like your main senses of sight, touch, hearing, taste, and speech, robots are equipped with external sensory devices such as a vision system, touch and touch sensors, speech synthesizer, and thanks to them the robot can communicate with the outside world [30].

Figure 2.3 – Different Robot sensors

5. **Controller**

The controller is rather close to your cerebellum; even if it doesn't have brain power; it always controls your movements. The controller receives data from the computer (the brain of the system), controls the movements of the actuators, and coordinates the movements with the information sent by the sensors [30].

6. **Processor**

The processor is the robot's brain(for an example we have the Arduino uno boeard, (see 2.4). It calculates the movements of the robot's joints, determines how much and at what speed each joint must move to reach the desired location and speed, and supervises the coordinated actions of the controller and the sensors. In some systems, the controller and processor are integrated together into a single unit, and in other cases, they are separate units[30].

Figure 2.4 – A controller used to control a robot

7. **Software**

Three groups of software are used in a robot. One is the operating system that operates the processor. The second is the robotic software which calculates the necessary motion of each joint of the robot based on kinematic equations. This information is sent to the

controller. This software can be at different levels, from the language of the machine to the sophisticated languages used by modern robots. The third group is the application collection - oriented routines and programs developed to use the robot or its peripherals for specific tasks such as assembly, loading machines, handling and vision routines[30].

2.4 Mobile robot

There are two main families of robots are:

1. Manipulating robots.
2. Mobile robots.

And we will focus on the second one.

A robot is a physical agent that performs tasks in the environment in which it evolves(see 2.5), it is an automaton equipped with sensors and actuators which offer it the ability to move and adapt close to autonomy [27].
In order to be autonomous the robot must implement:

- **A set of actuators**: these actuators allow the robot to move in other words they only serve to apply forces physical to the environment among the actuators we find: arms, wheels, clamps, etc.

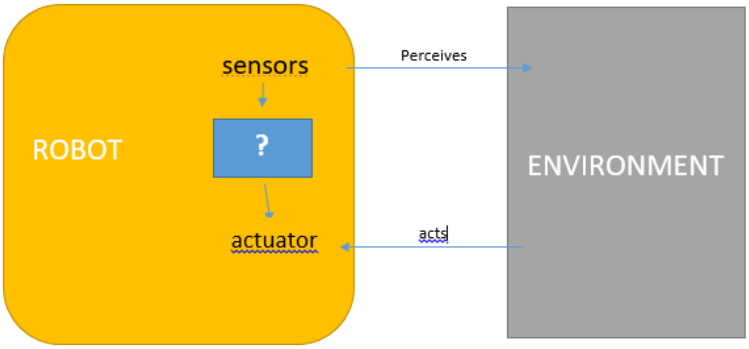

Figure 2.5 – Robot decision loop

- **A set of sensors**: these sensors allow the robot to perceive the environment in which it operates, robotics uses several types of sensors of two main categories [14]:
    1. **Proprioceptive sensors:** provide information on the internal state of the robot (GPS position sensors, speed sensors, etc).

2. **Exteroceptive sensor:** provide information on the environment which robot is moving (compass, radar, camera, etc).

Mobile robot must also have a certain intelligence implied by algorithm or regulator, which allows it to calculate the commands received by sensors and sent to the actuators to perform a given task from the data received by the sensors.

2.5 Wheeled robots

The choice of wheels and their arrangement give the robot its mode of locomotion, There are mainly three types of wheels for mobile robots(see 2.6):

- Fixed wheels or conventional wheel whose axis of rotation passes through the center of the wheel.
- Centered orientable wheels, the orientation axis of which passes through the center of the wheel.
- off-centered orientable wheels, called idler wheels, for this type of wheel the axis of orientation does not pass through the center of the wheel.

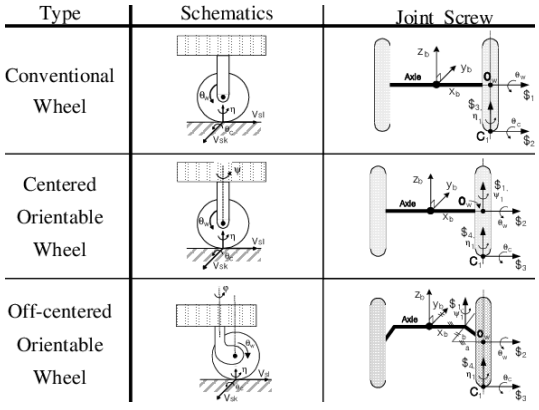

Figure 2.6 – Different types of wheels

2.5.1 The different configurations of wheeled mobile robots

There are several classes of wheeled robots determined mainly by the position and the number of used wheels.
We will cite here the main classes of wheeled robots.

1. **Configuration of a mobile robots with differential control**

A robot is said to command differential if the displacement of this last one is insured through two independent drive motors(as showing in 2.7), each of them is linked to a driving wheel The direction of the robot can be changed to playing on the speeds of two wheels drive wheels, the idle wheels are wheels free, their role is to ensure the stability of the robot platform.

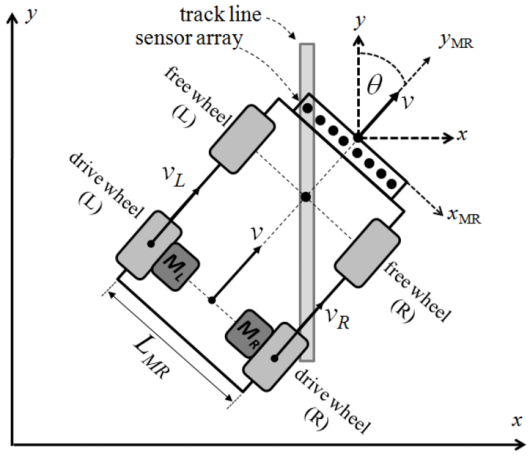

Figure 2.7 – Configuration of a mobile to differential drive

2. **Unicycle configuration**

A mobile robot is said to be a unicycle if it is powered by two independent wheels and that it has a number of wheels crazy people who provide stability. The figure (2.8) shows the diagram of a type robot unicycle. We omitted the idle wheels, which are not involved in the kinematics, as far as they have been judiciously placed. This type of robot is widespread due to its simplicity of construction and properties interesting cut-scenes.

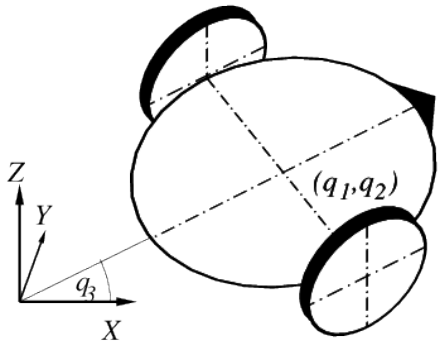

Figure 2.8 – Unicycle configuration

3. **Tricycle configuration**

The tricycle configuration robot illustrated in figure 2.9, consists of two passive rear wheels and a front drive wheel and vice versa.

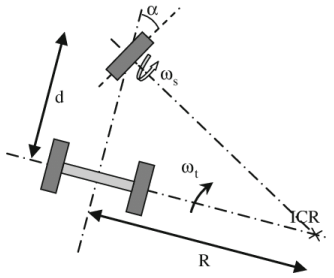

Figure 2.9 – Tricycle configuration

The robot movement conferred by two actions: the longitudinal speed and the orientation of the steerable wheel. To locate this type of robot, a sensor orientation is associated with the steerable wheel, and two encoders are associated with the wheels driving.

4. **Cars or the so-called ackerman configuration**

Used in the automotive industry, in this configuration the inner front wheel turns at a slightly narrow angle to the outside wheels when cornering which reduces tire slippage (see 2.10).

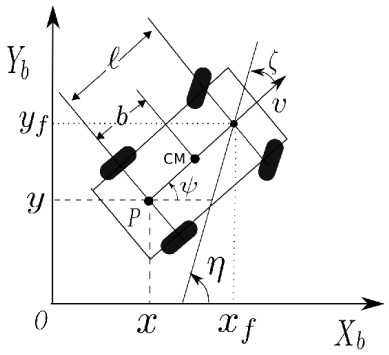

Figure 2.10 – Car configuration

5. **Synchronous traction robot**

Synchronous traction is a technique used to minimize the effect of slip and increase the pulling force. This configuration has three wheels or more (see 2.11), coupled so that they all rotate in the same direction and at the same speed. Mechanical synchronization can be done by different methods, the most used is through a belt.

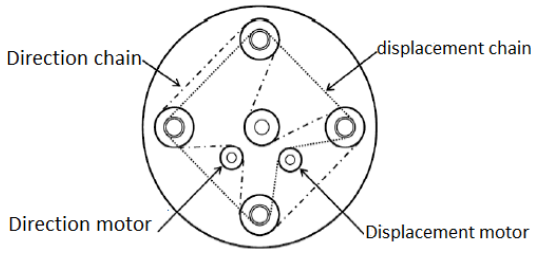

Figure 2.11 – Synchronous traction configuration

2.6 Location of a mobile robot

For the robot to be autonomous in performing certain given tasks, it must be able to locate itself in its environment, for example, a robot must move from point A to point B, the location of the latter is one of the major problems of mobile robotics, for the robot to be able to perform its movement it must necessarily know its current position to calculate its trajectory in order to go to the target position B, in addition to its movement, the robot must update its trajectory by calculating the position throughout its displacement, and this to know if it is arrived at its target destination or not, we will briefly present in this part the most used means of localization.

1. **Distances measure**

The most used navigation method for locating a mobile robot, this method provides good short-term accuracy, the principle of this method is to integrate incremental movement information over time, the weak point of this method is the accumulation of orientation errors, which will cause errors important on the position, the latter increases in proportion to the distance traveled, despite its weak point this method remains present is preferred by researchers [5].

2. **Distance sensors**

   Rangefinders or distance sensors are also widely used in the mobile robotics, these sensors give the possibility of having distance information between the nearest obstacle and the robot, for these sensors there are two distance sensor technologies:

   - **Sensors using ultrasound**: as its name suggests these sensors are based on the use of acoustic waves whose frequency is too high high to be heard by humans, it's ultrasound.
   - **The sensors using light**: these sensors detect the intensity of the light captured.

3. **Cameras**

   Referred to by visual localization, this method consists in identifying bitters and to locate themselves by using for example a characteristic map, the latter represents the environment through global bitter coordinates (points, lines, polygons, etc), or by a topological map, which characterizes the environment by a interesting graph between the different regions of the environment.

2.7 Area of use

- **Industrial robots**

  Industrial robots are robots used in an industrial manufacturing environment. They are used in the manufacture of automobiles, electronic components and parts, drugs and many products.

- **Domestic or household robots**

  Robots used at home. This type of robot includes many very different devices, such as robotic vacuum cleaners, pool cleaning robots, sweepers, gutter cleaners and other robots that can do different tasks. In addition, some surveillance and telepresence robots could be considered household robots if used in this environment.

- **Robots in medicine and surgery**

  Robots seem to have a future in the hospital. Robodoc helps to perform certain surgical operations. The nursing robot is still in the planning stage. The HAL cyberskeleton helps people get around. And the patient robot allows future dental surgeons to learn how to treat without causing damage.

2.8 The advantages and inconveniences

A robotic system consists not only of robots but also of other devices and systems which are used with the robot to perform the necessary task.

- **Advantages of robots**

  -Robotics and automation can in many situations increase productivity, safety, efficiency, quality and consistency of products.
  -Robots can work in a hazardous environment, without the need for life support, or safety concerns.
  -Robots do not need lighting, air conditioning, ventilation, and noise protection.
  -Robots work continuously, without feeling tired or bored, and do not require medical or vacation insurance.
  -Robots are repeatable precision at all times, unless something happens to them or they wear out.
  -Robots can be much more precise than humans. Linear accuracy of a robot typically is 20 to 10 microns.
  -The downside of robots is that they lack the ability to react in an emergency, unless the situations understood and the responses are included in the system. The safety measures necessary to ensure that they do not harm operators and do not damage the machines that work with them [1].

- **Disadvantages of robots**

  -Inadequate or wrong answer.
  -The lack of powers to make a decision.
  -Energy consumption.
  -They can cause damage to other devices, and injury to humans.
  -Although the robots have good certain characteristics but also have these limited characteristics like the ability to degree of freedom, the dexterity, sensors, vision system and the response in real time. Robots are expensive because of the initial cost of equipment, the cost of installation, the need for peripherals, the need for training, and the need for programming [1].

2.9 Conclusion

In this chapter we have presented a general definition of a robot and we have chosen the mobile robot type and we select this type with wheels. We detailed the different configuration for those wheels, location, area of use and pros and cons of robotics also we had as a little conclusion that mobile robotics plays a special role. Unlike manipulative industrial robots which work independently in a large number of automated factories, mobile robots are not very widespread. This situation is not due to the lack of possible applications, but as soon as we have mobility, we can imagine postmen robots, cleaners, guards, deminers, explorers, gardeners and many others. The low diffusion is mainly due to the fact that these tasks have a much higher complexity than those

carried out by industrial manipulator robots. The world in which a mobile robot must move is often very large, partially or totally unknown, difficult to characterize geometrically and having its own dynamics.
In the next chapter, we will have same implementation and conception of specific algorithms to compare also between them to use it in path planning for mobile robot .

Part II

Design and implementation of a system for the control of a mobile robot based on Arduino

Chapter 3

Simulation and evaluation of planning algorithms

3.1 Introduction

The graph search algorithms are the most known solutions for mobile robots in path planning, these algorithms use the directed or underacted graph trees, these algorithms were used in most computer games and GPS systems for finding the shortest and the lowest cost path [24].
The most known algorithms for the shortest path problem are Breadth-first, Dijkstra and A-star (A*) algorithms. Most studies focus on one of these algorithms and examines various types of selected algorithm, In this chapter, we will present the way how these algorithms works and an application to compare between theme, we will see also the part of implementation and the tools that we need to make this application, also see the results of this comparison to see which one is good for our work.

3.2 Breadth-first search algorithm

Breadth-first algorithm works with the method branching from the starting cell to the neighbour cells until the goal cell is found[26] [39]. Every traversable neighbour cell is added to an array which is called **open list**. **open list** is the array of neighbour cells which must be reviewed in order to find the goal cell. **open list** elements are reviewed if one is the goal cell or not. Then OPEN LIST grows up with the new neighbour cells of the old neighbour cells and this procedure goes on until the goal cell is added to the **open list**. The cost of the starting cell is 0. The cost of each neighbour cell is plus + one(or +defined constant cost) of the cell which added it to the **open list**. The costs of the cells are stored in a matrix(cost-matrix) with the same dimensions of the map. Then after adding the new neighbour cells, old reviewed cells are pulled out of **open list**. This prevents reviewing the reviewed cells again. When the goal cell is added to the **open list**, to find the shortest path you just follow from the goal cell to the starting cell step by step by the decreasing cost of the cells from the cost-matrix If the **open list** is empty anytime, this means

there is no possible paths[45].
The algorithm (1) is as follows:

Algorithm 1: Breadh-first algorithm

1. Define the starting and goal cells.
2. Load the map matrix.
3. Add the starting cell to **open list**.
4. Add the neighbour cells to **open list**.
5. If **open list** is empty , no possible path.
6. If goal cell is added to **open list** , define the path using map matrix. Else compute the cost of neighbour cells.
7. Pull out the reviewed cells from **open list**.
8. Go to step 4.

This algorithm is simple to implement, doesn't need too much matrix operations and also doesn't need to use the location of the goal cell (an advantage if the location of the goal isn't defined) but it has two major and important disadvantages:

1. You have to search the whole traversable cells until the goal cell is found. In large maps it needs very large computational space. It ignores the knowledge of the location of the goal cell.
2. It is impossible to define cells with different costs (just traversable and untraversable).

3.3 Djikstra algorithm

This algorithm is like Breadth-first algorithm but adds the computation of different cost cells (not only the shortest path but also the lowest cost path) [39], In this algorithm, again the neighbour cell array **open list** exists. First the neighbours of the starting cell are added to the **open list**. Then the costs of the neighbours are computed. These costs are the costs of moving from starting cell to these cells and it is the cost function. Neighbour cells are reviewed according to their computed costs. The lowest cost cell is found and first the neighbours of this cell is added to **open list**(lowest cost becomes comparison criterion) For these new cells the cost from starting cell to these cells are computed and again the neighbours of the lowest cost cell is added to the **open list**. This lowest cost criterion obtains the shortest path but it has two problems. First **open list** has to

be sorted according to the costs and the new neighbour cells have to be located in the right place in the **open list**. The parents of the neighbour cells have to stored in **parents** array in order to locate the new cells and in order to find the shortest and the lowest cost path. Furthermore if the cost of the neighbor cell is lower than its parent cell the neighbour becomes the parent and the costs have to re-computed. This procedure goes on until the goal cell is added to the **open list**. When the goal cell is added to the **open list**, following the parents of the cells from the goal cell to the starting cell gives the shortest and the lowest cost path. If the **open list** is empty anytime, it means that there is no possible paths.
The algorithm (2) is as follows:

Algorithm 2: Djikstra algorithm

1. Define the starting and goal cells.
2. Load map matrix.
3. Add the starting cell to **open list**.
4. Add the neighbour cells to **open list**, compute the costs, record their parent cell to **parents**.
5. If **open list** is empty, no possible path.
6. If goal cell is added to **open list** define the path using **parents** matrix.Else go on.
7. If neighbour cell is added **open list** before find its new cost and compare to its old cost. If it is lower, update the cost and **parents** matrix.
8. Pull out the reviewed cells from OPEN LIST.
9. Go to step 4.

Using the lowest cost criterion and the ability of computing different cost cells makes this algorithm efficient in large and different cost terrain maps. But it is lack of searching towards the direction of goal cell[45].

3.4 A-star algorithm

This is the most common and efficient used algorithm in shortest path finding problems [26], [39]. This algorithm has two list arrays **open list** and **closed list**. **open list** does the same work and **closed list** holds the cells that have to be saved. Again first the neighbours of the starting cell are added to the **open list**. And again these cells are reviewed according to their costs. But this time two cost functions exist. First the G cost function is the cost of moving from the starting cell to the current cell and the H cost function is the cost of moving from the current cell to the

goal cell. The G cost function can be computed but the H cost function can just be estimated. That's why this cost function is called heuristic cost function. There are several methods for this estimation. For 4-adjacent traversable cells Manhattan method is the most used method. Other methods can be found in the literature [24].

$$H(curent_cell) = abs(curentX - goalX) + abs(curentY - goalY)$$

This method directs the search to the goal cell. The total cost function $F = G + H$ is the comparison criterion for the cells. **open list** has to be sorted and in addition as the comparison criterion the F cost array has to be sorted. The parents of the neighbour cells are stored in **parents** array. Again in this algorithm if the cell exists in **open list** its new cost must be compared to the old cost. If it is lower the cell becomes the parent and G and F costs must be re-computed. The reviewed cells are placed in the **closed list**. Again after the goal cell is added to **open list**, following the parent cells gives the shortest path. If the **open list** is empty at anytime, it means that there is no possible path[45].
The algorithm (3) is as follows:

Algorithm 3: A-star algorithm

1. Define the starting and goal cells.
2. Load the map matrix.
3. Add the starting cell to **open list**.
4. Add the staring cell to **closed list**.
5. Add the neighbour cells to **open list**.
   - If traversable;
   - If not in **open list** before;
   - If not in **closed list**;
     With the order compute G , H and F cost function values. Record the parent to **parents** matrix. Locate the F cost function value in the right place.
   - If in **open list** before;
     compute the G cost function value. If it is better than the old value, chance the parent with this parent in **parents** matrix. Update G and F cost functions.
6. If **open list** is empty, no possible path.
7. If the goal cell is added to **open list** define the path using **parents** matrix.
8. Find the lowest cost neighbour cell. Add it to **closed list** and continue the search on this cell.
9. Pull out the reviewed cells from **open list**. Go to step 5.

This algorithm is the most efficient algorithm because it uses both the shortest path information from starting cell and the shortest path information to the goal cell. But if the location of the goal cell is not known, this algorithm can't be used[45].

3.5 Implementation and Test

After taking acknowledges of the three main algorithms for path planning that are mentioned in the previous sections, we have to pass to the next steps of the project, which are coding and test. These phases aim to implement the three algorithms, simulation, test, and discuss the results showing. This section introduces by the development of tools and languages that we need in the realisation of this application. Then presents the main implementation results of our application. Finally, we present and discuss some experimental results.

3.5.1 Development tools and language

In this part, we present different tools and languages, that helps us during the implementation of this application.

- **Python programming language**

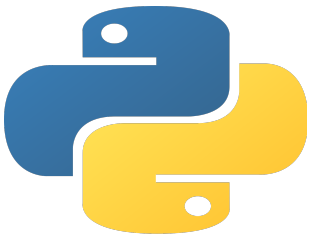

Figure 3.1 – Python logo

Python has grown into one of the most widely used general-purpose, high-level programming languages, and is supported by one of the largest open source developer communities. Python is an open source programming language that includes a lot of supporting libraries. These libraries are the best feature of Python, making it one of the most extensible platforms. Python is a dynamic programming language, and it uses an interpreter to execute code at run-time rather than using a compiler to compile and create executable byte codes.
The philosophy behind the development of Python was to create flexible, readable, and clear code to easily express concepts. Python supports functional, imperative, and object-oriented programming with automatic memory management[11].

- **PyCharm Programming editor**

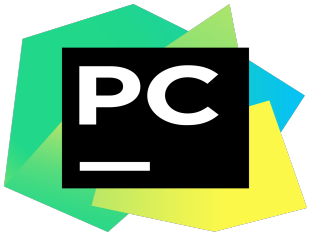

Figure 3.2 – PyCharm logo

PyCharm is an open source Integrated Development Environment (IDE), used for python programming. It is a powerful coding assistant, it can high- light errors and introduces quick fixes based on an integrated Python debug-ger. It is a suitable editor for writing and testing many lines of code and classes, since it offers a structural project view, and a quick files navigation.

- **PyGame**

Figure 3.3 – PyGame logo

PyGame is a set of Python modules designed for writing video games. PyGame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language. PyGame is highly portable and runs on nearly every platform and operating system.

3.5.2 Implementation

The studied algorithms in the previous sections are implemented using a set of software and hardware which are summarised in the following (4.2).

Software/Hardware	**Version**
OS	Microsoft Windows 10 Professional, 64bits, version version 10.0.18362.959
CPU	Intel(R) Core(TM) i5-7200U CPU @2.50GHz 2.70GHz
RAM	8.00Go
Python Interpreter	3.7.2
PyCharm	2018.1.4
Pygame	1.9.6

Table 3.1 – Software/Hardware versions

- **Main application**

1. **The Main GUI**
   The figure bellow 3.4 shows the main page.

Figure 3.4 – first window to choose one of the 3 algorithms

This window (3.4) allows us to chose from one of the 3 following by clicking on the name of algorithm we want to see there simulation,in each window we can see the time needed to get to the goal.

2. **A-star search algorithm**

Now we present the simulation in A-star algorithm(see (3.5), the home icon is for the start point and the cross icon is for the goal point, the black cells are consider as obstacles, the cells with red are the visited one, the cells with yellow are for the last path found. The necessary Time to find the path in this algorithm will be showing in the window caption.

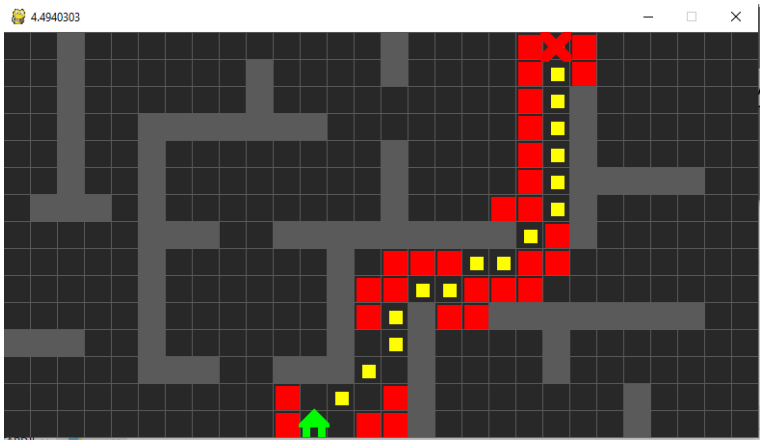

Figure 3.5 – Path from home to cross icon in A-star algorithm

3. **Breadth first search algorithm**

Now we present the simulation in BFS algorithm (see 3.6), the home icon is for the start point and the cross icon is for the goal point, the black cells are consider as obstacles, the cells with red are the visited one, the cells with yellow are for the last path found. The necessary Time to find the path in this algorithm will be showing in the window caption.

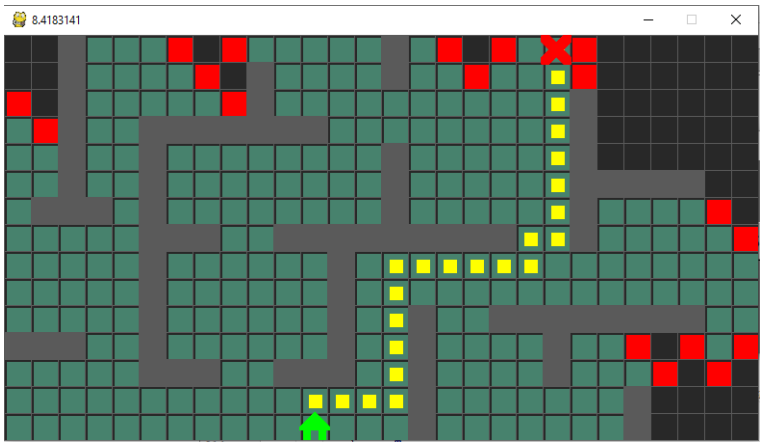

Figure 3.6 – Path from home to cross icon in BFS algorithm

4. **Djikstra algorithm**

Now we present the simulation in djikstra algorithm (see 3.7), the home icon is for the start point and the cross icon is for the goal point, the black cells are consider as obstacles, the cells with red are the visited one, the cells with yellow are for the last path found. The necessary Time to find the path in this algorithm will be showing in the window caption.

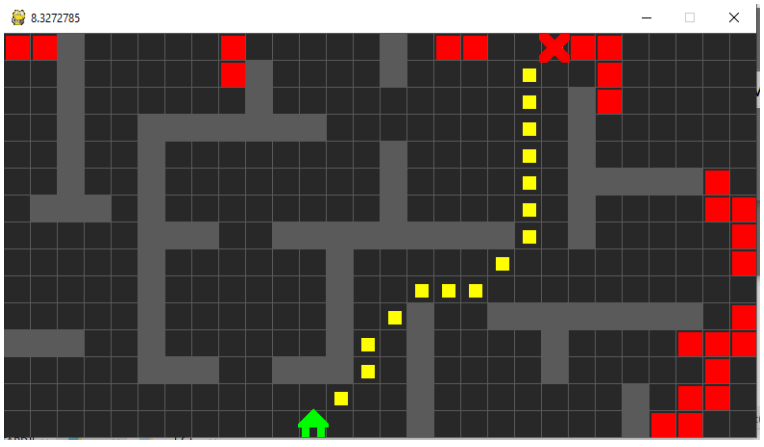

Figure 3.7 – Path from home to cross icon in djikstra algorithm

3.5.3 Experimental results

We made this application which make a simulation for the 3 algorithms that we talked about before to compare between them so we can know which one is the best for our work. Now we will go to the comparison part.

The comparison process was conducted by measuring 3 variables of path finding computation. These three variables are the measurement of the process time, the sum of the cells path, and the sum of visited cell to get the optimal path.

In the process of measuring time, time will be measured in seconds (s). The calculated time is the time needed from start node to destination node. The time will start when we press space so the simulation starts running. The sum of path cells measured by counting the number of final optimal and shortest results of cells that gets from search algorithm, the sum visited cells measured by counting each cell visited by the search algorithm to find the optimal path. The purpose of this research will determine the best algorithm to be applied in our mobile robot.

The algorithm which will be recommended to be applied in the next project is considered to be taken from various aspects of the comparison test results. The results of this research are expected to help to implements the best algorithm for making this robot.
The next table in table (3.2) will be showing the result of comparison:

Algorithms	**Time (seconds)**	**Sum of the cells visited**	**Sum of the cells path**
A-star	4.4940303	55	188
Djikstra	8.3272785	306	188
Breadth -first	8.4183141	286	188

Table 3.2 – The comparison results between the three algorithms

From table 3.2 above, the same results of 188 blocks (cells) were obtained from the distance traveled by the the start point (home icon) to the destination point (cross icon). With these three algorithms, we will get the same path length. As we can see that The computational time required by A-star algorithm is faster than djikstra and Bfs. Bfs has 0.1 s longer computation time than Dijkstra. The computational process of bfs algorithm is slightly slower than the a-star and djikstra algorithms. It can be deduced that A* algorithm consumes less memory for its computation process than any other algorithm.
Based on the results of this research and evaluation conducted, it can be concluded that:

1. A-star , Dijkstra, and Breadth First Search can be used to find the shortest path in path planning.
2. A-star is the best algorithm in path finding especially in Maze / grids. This is supported by the minimal computing process needed and a relatively short searching time.
3. The use of the right algorithm can make the work or the application better in terms of computing process, memory usage, and computing time.

3.6 Conclusion

This chapter presents three path planning algorithms for a mobile robot on grid based map for one starting-one goal, an application for comparison and to see which one of the 3 algorithms are the most appropriate. For maps with different cost cells and with one starting one goal cell A-star is best in computational time and size of memory. But the heuristic function H for A-star must be chosen carefully in order to make sure of the shortest and lowest cost path.
In the next chapter we will present the realisation of a robot mobile using python and the Arduino uno broad.

Chapter 4

Realisation

4.1 Introduction

As part of our attempt to produce a car with four tires that go to a certain destination by using path finding algorithm, and after taking acknowledge about mobile robot components and the algorithms can be used to reach our goal in other chapters we made this chapter for the robot production.
In this chapter, we will try to describe what the project exactly do, the goal that we want to reach, and see the list of components needed to make it reel. Also designing the software, hardware of the system. At the end we will have a look at the project setup.

4.2 Project analysis

The main goal of our work is to use python and Arduino to control a mobile robot. We must build a prototype which contains a car with four wheels have to reach the goal from a given start point with avoiding obstacles and choosing the shortest and optimal path, all this by using an algorithm for path planning implemented in python and the result will be sending to the Arduino board as a commands, this board contains the program that allows us to manage these commands and control the robot behavior.

We can describe the project goal in one sentence as follows: generate an optimal path for a car mobile robot from a start point to a destination one with avoiding any obstacles in the way and take the shortest path. In comprehensive list of goals, we will have to perform the following tasks to satisfy the mentioned project goal:

- knowledge of the environment by entering it as a picture into the system.
- knowledge of start point and end one from the same picture giving to the system.

- Detect all the obstacles in the environment.
- Find the shortest and optimal path between the start and goal by using a specific algorithm for path finding.
- Go to the goal with less time and power.

The project can be implemented as a DIY(do it yourself) application or as part of other projects with minor modifications. This project is composed by two parts, hardware part and software part.

4.3 Conceptual Model

The first step, before jumping to work on any hardware system, is to design the project model using logic. We use this diagram (see 4.1)as a flowchart to better understand how all the project works, the layout of the components and the flow of the code. The following diagram shows the flow of the project where you can see that the project runs in loops once motion is detected and the appropriate car actions are performed.

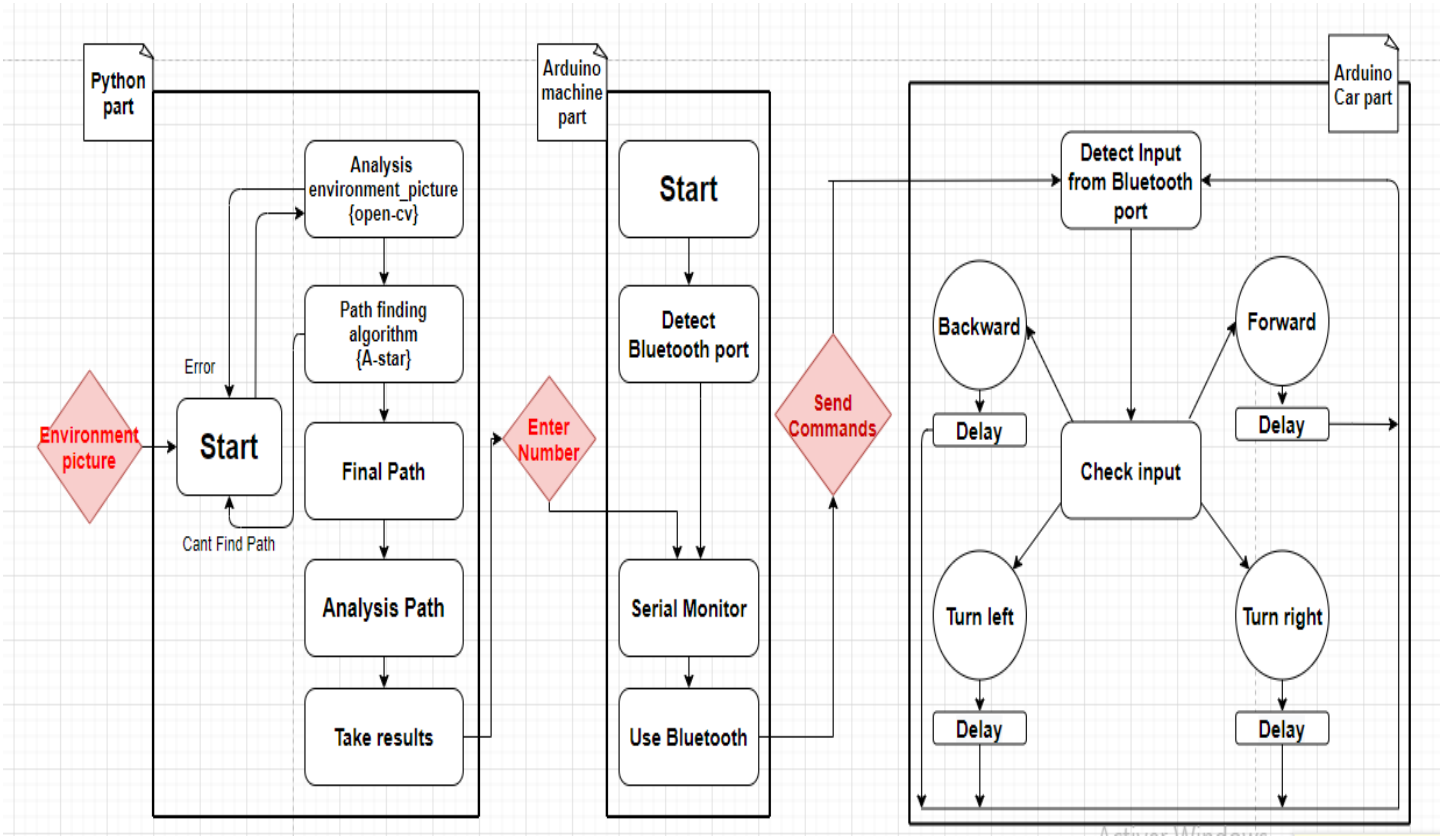

Figure 4.1 – Conceptual model design

As you can see, the program logic divided into three parts the first one start by program running with python, the last starts by entering a picture of the environment which is a visual grid of squares with obstacles in it. Then, we will use computer vision and a path planning algorithm to find the optimal route from point A(start) to point B(goal) in the grid. we will need the OpenCV, scikit-image, and numpy libraries installed for python.
The picture contains:

1. 10x10 grid, making 100 squares .
2. Obstacles marked as black square .
3. Start point defined by circle with S, the goal point defined by circle with G.

We choose the same shape of object so the open-cv can find a match between them. The squares are identified by the coordinate (x, y) where x is the column and y is the row to which the square belongs. Each square can be empty or have an Obstacle or have an Object. The picture entering to the system is as follow in (4.2), we try to embodiment this picture in reel environment for this project in (4.3).

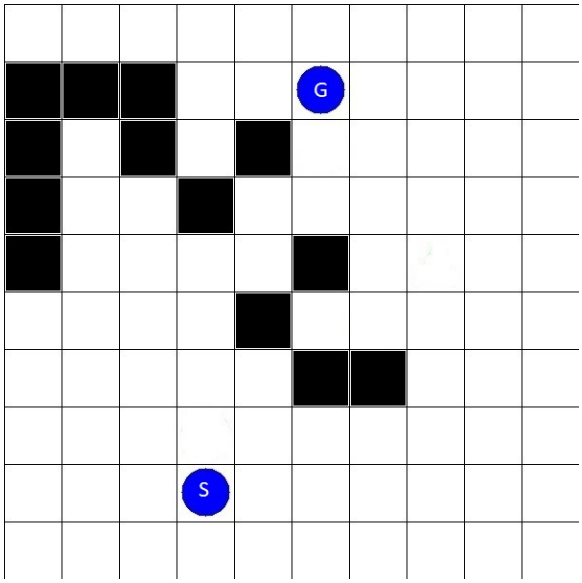

Figure 4.2 – Environment picture entering to the system

Figure 4.3 – Environment picture in reel life

After input the picture and analysis it with open-cv to find the start and goal point we send the coordinates to an algorithm of path finding and here we choose the A-star algorithm, the last one works as we said in the other chapters and in the end gives us a message of non path finding shows if its not possible to find this path and the program starts over again, if its possible we will have the final optimal path as list of couple. Each couple have the coordinate (x, y) of which square the path should goes.

Now we take this result list to diagnostics. We take this list and divided it, the first couple named (x, y) and the next couple named (e, f) for example $x = 4$, $y = 8$ for the first couple and $e = 4$, $f = 7$ for the next couple, ect, as showing the path result in figure (4.5), now we compare the 2 first couples and the second with the third, ect as showing in figure (4.4).

```
print(result)
for i in range(0, len(result)):
    # for G in range (1,len(result)):
    if ((i + 1) < len(result)):
        x, y = result[i]
        e, f = result[i + 1]
        print("x  ,  y", x, y)
        print("e  ,  f", e, f)
        if (x == e):
            if (y > f):
                print("forword")
                serialcomm.write("on".encode())
                time.sleep(2)
                print(serialcomm.readline().decode('ascii'))

            if (y < f):
                print("backword")
                serialcomm.write("off".encode())
                time.sleep(1)
                print(serialcomm.readline().decode('ascii'))
        else:
            if (x > e):
                print("go left")
                serialcomm.write("left".encode())
                time.sleep(5)
                print(serialcomm.readline().decode('ascii'))
            if (x < e):
                print("go right")
                serialcomm.write("right".encode())
                time.sleep(8.5)
                print(serialcomm.readline().decode('ascii'))
serialcomm.write("quit".encode())
print(serialcomm.readline().decode('ascii'))
serialcomm.close()
```

Figure 4.4 – Comparing code

```
print(result)
for i in range(0, len(result)):
    # for G in range (1,len(result)):
    if ((i + 1) < len(result)):
        x, y = result[i]
        e, f = result[i + 1]
        print("x  ,  y", x, y)
        print("e  ,  f", e, f)
        if (x == e):
            if (y > f):
                print("forword")

            if (y < f):
                print("backword")
                time.sleep(1)
        else:
            if (x > e):
                print("go left")
                time.sleep(5)
                #print(serialcomm.readline().decode('ascii'))
            if (x < e):
                print("go right")

                time.sleep(8.5)

if not result:  # If no path is found;
    planned_path[startimage] = list(["NO PATH", [], 0])
planned_path[startimage] = list([str(grid), result, len(result) + 1])
```

Figure 4.5 – Execution results

After finding the result path, now we go to the second part which is the Arduino machine part. This part contain the Arduino code that running in our machine, after getting the path from the python part we enter in the serial monitor one of the numbers, the follow table (4.1) will show the meaning of each number.

Numbers	0	1	2	3	4
Value	Turn right	Forward	Backward	Turn left	Stop

Table 4.1 – Numbers values

Here came the work of Bluetooth, after installing its pilots (4.6) in our machine and configure the send from the serial monitor to the Bluetooth port we start to enter the numbers and send it (see 4.7 for the execution results).

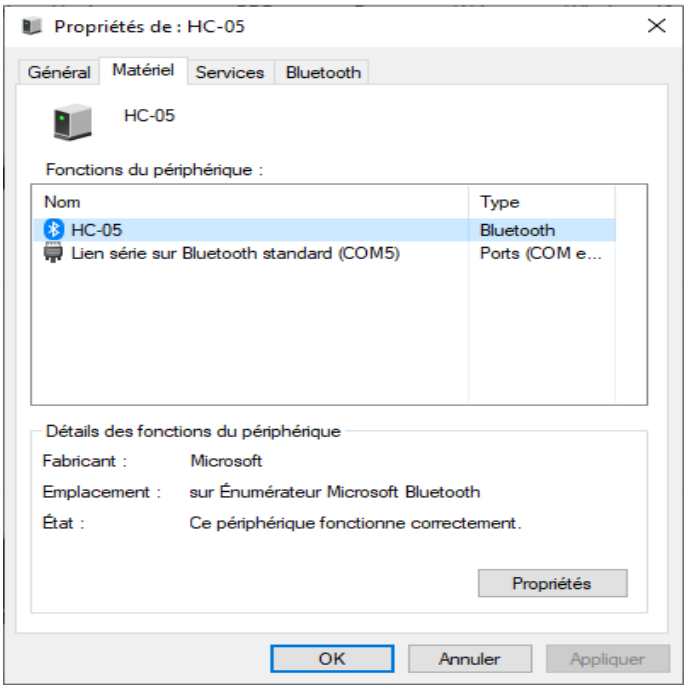

Figure 4.6 – Bluetooth installation in our machine

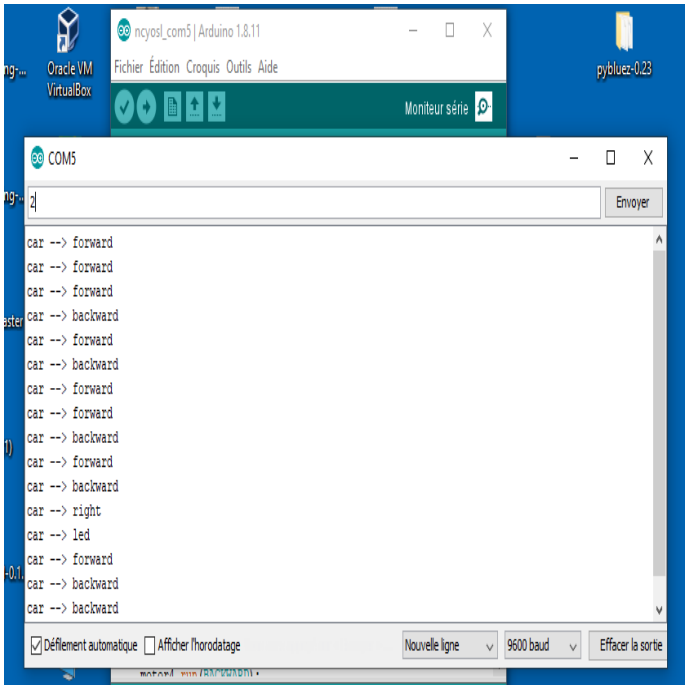

Figure 4.7 – Execution results from serial monitor

In here we go to the last part which is the Arduino board or the hardware part. After sending commands from the Arduino code thats running in our machine the Bluetooth starts by detecting the state of what the port hold by getting inputs. First check it and the code will be executed as it have to be in the Arduino board that we will see it in one of the next sections. We will also insert a delay between the execution of each successive loop so that the code still working and waiting for any other commands.

4.4 Software parts

In this part, we will see the 2 main parts of this project. But before entering to the two code and details we need to know the development tools that we need to make theses two codes.

4.4.1 Development tools and language

Beside to the tools that we already talked about in the previous chapter we need same others tools and libraries:

1. **For Python code**

- **Open-CV Library**

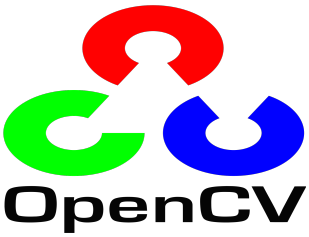

Figure 4.8 – OpenCv logo

Open Source Computer Vision Library is a library of programming functions mainly aimed at real-time computer vision[32].
OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

- **NumPy library**

Figure 4.9 – NumPy logo

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
NumPy is open-source software and has many contributors. NumPy is a python library used for working with arrays. It is an open source project. NumPy stands for Numerical Python.

- **Scikit-Image library**

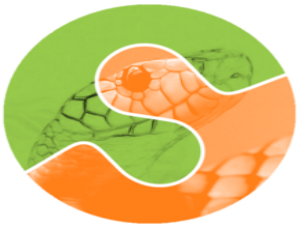

Figure 4.10 – Scikit-Image logo

Scikit-Image is an open-source image processing library for the Python programming language[43]. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more[15]. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

2. **For Arduino sketch**

- **Arduino IDE**

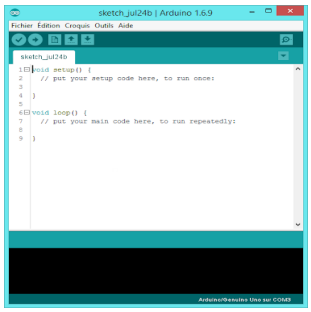

Figure 4.11 – Arduino IDE

The Arduino IDE is a cross-platform application developed in Java that can be used to develop, compile, and upload programs to the Arduino board, The IDE contains a text editor for coding, a menu bar to access the IDE components, a toolbar to easily access the most common functions, and a text console to check the compiler outputs. A status bar at the bottom shows the selected Arduino board and the port name that it is connected to, as shown in figure (4.11). An Arduino program that is developed using the IDE is called a sketch. Sketches are coded in Arduino language, which is based on a custom version of C/C++.

- **AFmotor library**

Figure 4.12 – AFmotor logo

The AF-DCMotor class provides speed and direction control for up to four DC motors when used with the Adafruit Motor Shield.

4.4.2 Implementation

The studied codes that we will discus next to this section(Python, Arduino code) are implemented using a set of software and hardware which are summarised in the following (4.2).

Software/Hardware	Version
OS	Microsoft Windows 10 Professional, 64bits, version version 10.0.18362.959
CPU	Intel(R) Core(TM) i5-7200U CPU @2.50GHz 2.70GHz
RAM	8.00Go
Python Interpreter	3.7.2
PyCharm	2018.1.4
Open-cv	4.1.2.30
Skimage-encoder-v2	0.0.1
Numpy	1.18.1
Scikit-image	0.16.2
Serail	0.0.97
Arduino	1.8.11

Table 4.2 – Software/Hardware versions for realisation

4.4.3 Python code

After getting familiar with the tools and libraries that we need now we start by detailing the program used for this project:
First we present the initial pseudo codes for this code, generation the initial grid (see 4), processing environment picture (see 5).

Algorithm 4: Generate initial grid

Result: Create the maze
initialization;

1. Define the window size.
2. Define the starting point.
3. Create a blank image.
4. create a list of blank images.
5. Generate the final maze.

After initialing the list of store coordinates of occupied grid, Dictionary to store information regarding path planning, List to store obstacles (black tiles), we go to generate the **list images** that represent an array of 100 blank images, and the maze that is an empty matrix to represent the grids of individual cropped images.

Algorithm 5: Picture processing

Result: Make the appropriate image initialization;

1. Load the image.
2. Copy the image.
3. Change the image format to square.
4. Crop the image.
5. Add the result image to **list images**.

After loading the image, we print its index and make a copy of it in **Clone**, image is our iterator, its where were at returns image matrix. Then we make this image as square format for openCv, crop it so it can be appropriate to add it in the array of images (**list images**).

The grid is to be considered occupied if either grid has an Obstacle or an Object. We need to check if white or not. The next pseudo code (6) represent the important steps to make the list of occupied grid of object. Next pseudo code (7) represent the important steps to make the list of occupied grid of obstacles.

Algorithm 6: pseudo code to create List of occupied grid of object

Result: Find object in image

1. Define list of average color in the grid.

for $i = 1 : averagecolor$ **do**
  if $i \leq 20$ **then**
    $obstacles \leftarrow i$
  end if
end for

Algorithm 7: pseudo code to create List of occupied grid of obstacles

Result: Find obstacles in images

1. Define list of average color in the grid.

for $i = 1 : averagecolor$ **do**
  if $i \leq 240$ **then**
    $occupied - grid \leftarrow i$
  end if
end for

In these two pseudo code (6), (7) we used the average color in the pictures to detect any object, obstacles in it, its mean that we want to check if every grid is white or not, if its not majorly white then it will be considered as an object, if its all black then it will be considered as an obstacle.

In the next pseudo code (8), we will try to find match between object so we can send it to the path finding algorithm to find the optimal path between the two matches.

Algorithm 8: pseudo code to find match between objects

Result: Match object

1. Get the object list.
2. Compare each image in the list of objects with every other image in the same list.
3. The most similar images will be sending to the a-star search algorithm.

Here we take every grid image in the list of objects and compare it to an other object in the same list to find if there is any match between them by using the compare structural similarity, if there is a match we send the index and the information of each image and her match image to the A-star algorithm (3) that we already talked about in other chapter so this last algorithm can find the optimal path between them if its possible.

Before Arduino, the domain of micro-controller-based applications was limited to hardware programmers. Arduino made it simple for developers that came from other software fields and even for the non-coding com-munity to develop micro-controller-based hardware applications. Arduino consists of a simple hardware design with a micro-controller and I/O pins to interface external devices. If one can write an Arduino sketch that can transfer the control of the micro-controller and these pins to an external software mechanism, then it will reduce one's efforts to upload Arduino sketches for every modification. This process can be performed by developing such an arduino program that can then be controlled using a serial port.

4.4.4 Arduino sketch

This section describes the Arduino code for the project. The arduino language has a set of library functions to simplify the programming experience. Although not all of these library functions are required by an Arduino sketch, setup() and loop() are mandatory functions and they are required to successfully compile the sketch. Here we will see the two functions we use for our Arduino sketch:

1. **Defining phase**

Before we go to see the two functions we need to call the library "AFMotor", the variable that store the data entering from the serial port, define the name of each motor and its input as follow:

include <AFMotor.h> //The library of motors.
String portdata ; //The variable to store data.

The constructor

The constructor for a DC motor is as showing in the four present lines. We call this constructor once for each motor in our sketch. Each motor instance must have a two parameter, port number and frequency (see the next items). The name for each motor should be different then the others as showing below in (see 4.13).

- **Port number** selects which channel (1-4) of the motor controller the motor will be connected to
- **Frequency** selects the PWM frequency. If no frequency is specified, 1KHz is used by default.

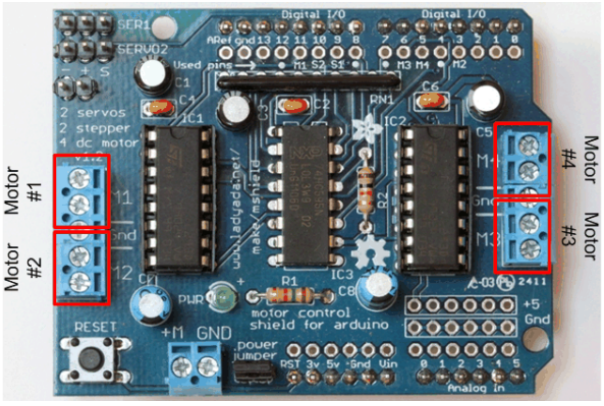

Figure 4.13 – Place of each motor

```
AF-DCMotor motor1(1,MOTOR12-1KHZ); //define motor1 on channel 1 with 1KHz PWM
AF-DCMotor motor2(2,MOTOR12-1KHZ); //define motor2 on channel 2 with 1KHz PWM
AF-DCMotor motor3(3,MOTOR12-1KHZ); //define motor3 on channel 3 with 1KHz PWM
AF-DCMotor motor4(4,MOTOR12-1KHZ); //define motor4 on channel 4 with 1KHz PWM
```

2. **The setup() function**

When Arduino runs a sketch, it first looks for the setup() function. The setup() function is used to execute important programming subroutines before the rest of the program, such as declaring constants, setting up pins, initializing serial communication, or initializing external libraries. When Arduino runs the program, it executes the setup() functions only once. the initialization of the setup() function, as displayed in the following code snippet:

```
void setup() {

//initialize the digital pin as an output.
pinMode(A1 , OUTPUT);
pinMode(A2 , OUTPUT);
pinMode(A3 , OUTPUT);
pinMode(A4 , OUTPUT);

//set the speed of each motor.
motor1.setSpeed(255);
motor2.setSpeed(255);
motor3.setSpeed(255);
motor4.setSpeed(255);

//the initial state of each motor will be in stop state.

motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
}
```

As you can see in our sketch, we used the pinMode() function to assign the role of each motor pin in the setup() function, set the speed and initialization the first state.

3. **The loop() function**

Once Arduino has executed the setup() function, it starts iterating the loop() function continuously. While setup() contains the initialization parameters, loop() contains the logical parameters of your program:

```
void loop() {

//this function run repeatedly

portdata = Serial.read(" n"); //read the input data and put in the portdata variable

if(portdata == '1') { //if the input data is 1 which is "Forward"

motor1.run(FORWARD);
motor2.run(FORWARD);
motor3.run(FORWARD);
motor4.run(FORWARD);
delay(1000); // run forward for 1 second
motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
delay(1000); // stop for 1 second
}

if(portdata == '2') { //if the input data is 2 which is "Backward".
motor1.run(BACKWARD);
motor2.run(BACKWARD);
motor3.run(BACKWARD);
motor4.run(BACKWARD);
delay(3000); // run backward for 3 second
motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
}

if(portdata == '3') { //if the input data is 3 which is "Left".
motor1.run(BACKWARD);
motor2.run(FORWARD);
motor3.run(FORWARD);
motor4.run(BACKWARD);
}

if(portdata == '0') { //if the input data is 0 which is "Right".
motor1.run(FORWARD);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(FORWARD);
delay(1500); // turn right for 1.5 second
```

```
motor1.run(RELEASE);
motor2.run(FORWARD);
motor3.run(FORWARD);
motor4.run(RELEASE);
delay(1300);// turn left for 1.3 second
motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
delay(1000);// stop for 1 second
}

if(portdata == '4') { //if the input data is 4 which is "Quit".
motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
}}
```

After send commands from the serial monitor through Bluetooth, the variable "portdata" will have one of theses data: 1 for "Forward", 2 for "Backward", 0 for "Right", 3 for "Left", 4 for "Quit" so the robot behavior will be accurate to one of the input data. The next table (4.3) will detail the desired run mode for the motor, we can take as a note that the motor shield does not implement dynamic breaking, so the motor may take some time to spin down.

Commands	**Value**
Forward	Run forward (actual direction of rotation will depend on motor wiring)
Backward	run backwards (rotation will be in the opposite direction from FORWARD)
Release	Stop the motor. This removes power from the motor and is equivalent to setSpeed(0).

Table 4.3 – Commands values

4.5 Hardware parts

This section assign to The Hardware side, from the hardware components to the system design and finally same tests and experiment the results obtained.

4.5.1 Hardware components

The major hardware component required for this project is a computer, an Arduino board, Bluetooth (HC-05) to connect with the laptop. we will also need four wheels connected to four dc motors gearbox in here we will need a L293D Motor Driver Shield so we can control all of these motors, we will be needing for same wires, a switch, battery for power.
The description of the necessary components is as follows:

1. **The Arduino uno board**

   Any electronic product that needs computation or interfacing with other computers first requires a quick prototyping of the concept using simple tools. Arduino is an open source hardware prototyping platform designed around a popular micro-controller family, and it includes a simple software development environment. The Arduino board that we will use is Uno board (see 4.14).

Figure 4.14 – Arduino uno

The latest revision of the uno board is based on Atmel's ATmega328 micro-controller. The board extends the I/O pins of the micro-controller to the peripheral, which can then be utilized to interface components using wires. The board has a total of 20 pins to interface, out of which 14 are digital I/O pins and 6 are analog input pins. From the 14 digital I/O pins, 6 pins also support pulse-width modulation (PWM), which supports the controlled delivery of power to connected components.

The board operates on 5V. The maximum current rating of the digital I/O pins is 40 mA, which is sufficient to drive most of the DIY electronic components, excluding motors with high current requirements.

While the previous image provided an overview of the uno board, the following diagram in figure (4.15) describes the pins on the uno board. As you can see, the digital pins are located on one side of the board while the analog pins are on the opposite side. The board also has a couple of power pins that can be used to provide 5V and 3.3V of power to external components. The board contains ground pins on both sides of the board as well. We will be extensively using 5V of power and ground pins for our projects. Digital pins D0 and D1 support serial interfacing through the Tx (transmission) and Rx (receiver) interfaces respectively. The USB port on the board can be used to connect Arduino with a computer[11].

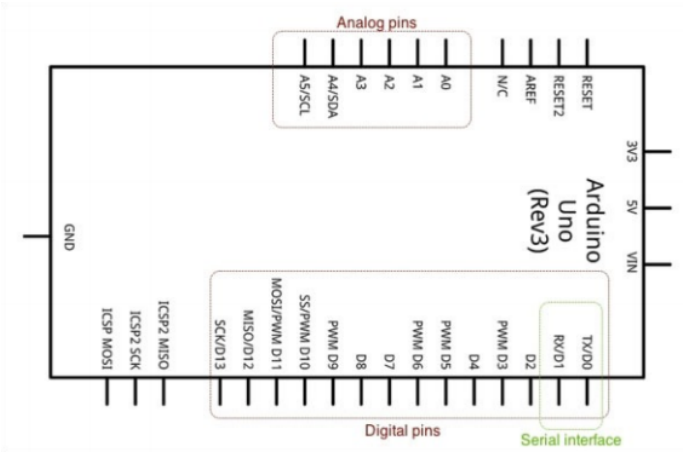

Figure 4.15 – The pins on uno board

We can present the characteristics of the Arduino uno used in the follow table (4.4):

Micro-controller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Table 4.4 – Arduino characteristic table

2. **Bluetooth HC-05**

We will need a Wireless communication between the machine and the mobile robot by using the Bluetooth 4.16 so we can run the car in real time.

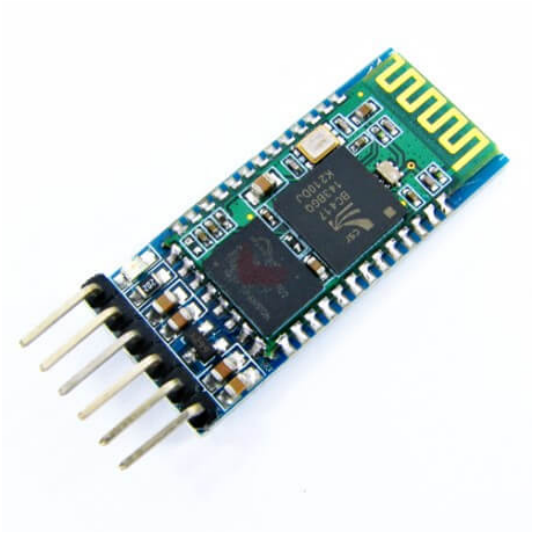

Figure 4.16 – Bluetooth HC-05

HC.05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. The HC-05 Bluetooth Module can be used in a Master or Slave configuration, making it a great solution for wireless communication. This serial port bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04[U+2010]External single chip Rluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature).

3. **A computer**

After configured a computer with Python and the Arduino ide. we will need this computer for the project to upload the application that generate for us the right path and by entering the path result to the Arduino program in our machine, this last will send commands to arduino board(car) by the bluetooth.

4. **Wheels**

In order for the robot to be able to move it is necessary to use the wheels, we used four of theme and each one is connected to one of the dc motors.

5. **Dc motors gearbox**

DC motor (see 4.17) is the most common type of engine that can be used for many applications. We can see it in remote control cars, robots, etc.
This motor has a simple structure. It will start rolling by applying proper voltage to its ends and change its direction by switching voltage polarity. DC motors speed is directly controlled by the applied voltage. When The voltage level is less than the maximum tolerable voltage, the speed would decrease.

Figure 4.17 – Dc motor gearbox

6. **L293D Motor Driver Shield**

We can't connect motors to micro-controllers or controller board such as Arduino directly in order to control them since they possibly need more current than a micro-controller can drive so we need drivers. The driver is an interface circuit between the motor and controlling unit to facilitate driving. Drives come in many different types.
L293D shield (4.18) is a driver board based on L293 IC, which can drive 4 DC motors and 2 stepper or Servo motors at the same time.

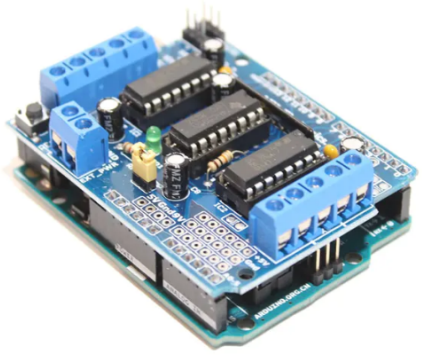

Figure 4.18 – L293D Motor Driver Shield

7. **Battery**

In order to give a power to the motors and to make the project portable we need same battery.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts (see 4.19).
In our case we are connecting the power to the L293D motor driver which can make a 6v .

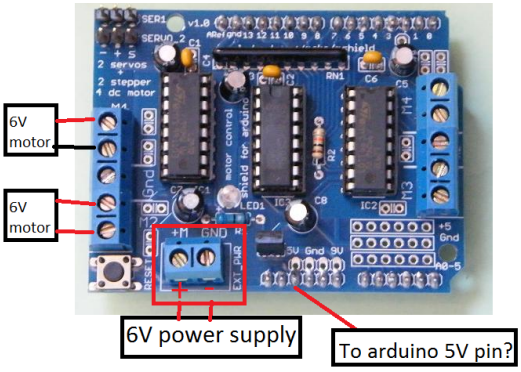

Figure 4.19 – L293D power

8. **Switch**

We used a switch so we can cut the power or make it in.

4.5.2 Hardware system design

Designing a diagram for a software flow helps to write the program and also assists in identifying actions and events for the project. The process of hardware system design includes circuit connections, schematic design, simulation, verification, and testing. This design process provides a detailed understanding of the project and the hardware components. It also helps in preliminary verification and testing of the project architecture. Before we jump to the hardware design process of this project, let's get ourselves familiar with the helpful tools.

- **Tinkercad**

Tinkercad is a free, online 3D modeling program that runs in a web browser, known for its simplicity and ease of use.
Tinkercad uses a simplified constructive solid geometry method of constructing models. A design is made up of primitive shapes that are either solid or hole. Combining solids and holes together, new shapes can be created, which in turn can be assigned the property of solid or hole. In addition to the standard library of primitive shapes, the user can create custom shape generators using a built-in JavaScript editor[16].
We will use this tool to design our hardware system. The follow figure (4.20) shows the result.

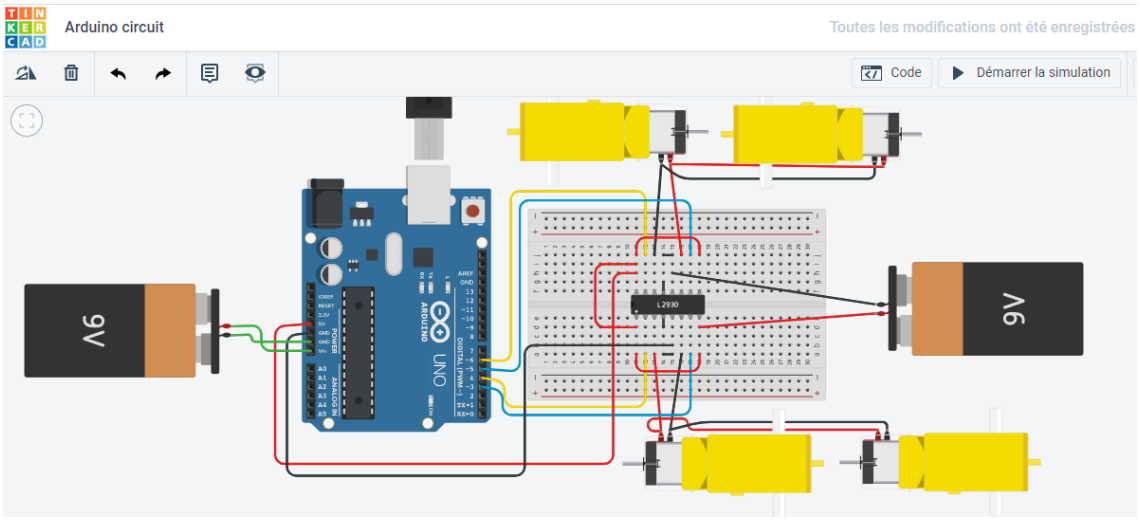

Figure 4.20 – Hardware system design

In our production we avoid using the board but in simulation we need it to make the communication so this is what we exactly did.

1. Connect the motor driver shield l293D directly to the Arduino uno board as Figure (4.21) shows.

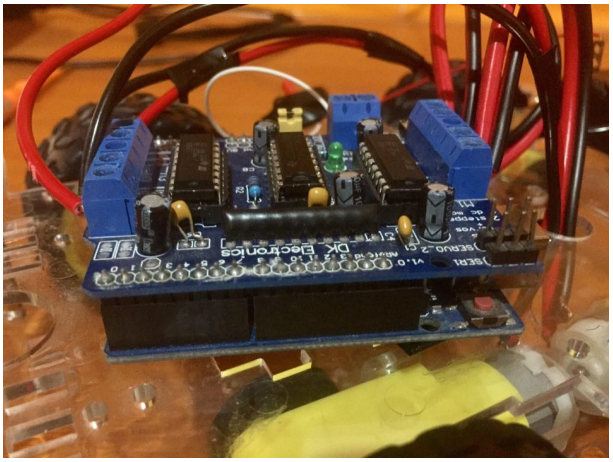

Figure 4.21 – The 2 boards

2. The way we connected wires of each motor in the l293D is like the two next figures (4.22) and (4.23) shows.

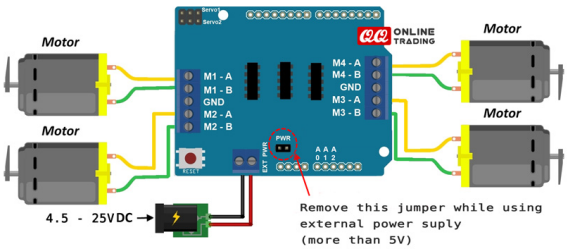

Figure 4.22 – Motor, power and driver sheild wires connection

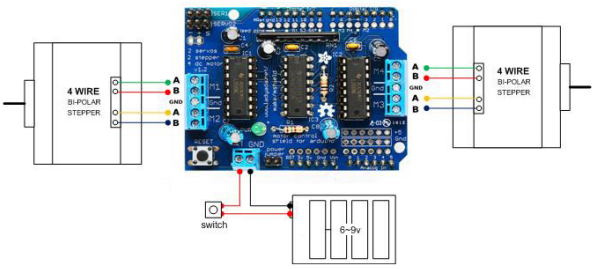

Figure 4.23 – Design motor, power and driver sheild wires connection

- The digital pins used by this shield for motors

  Motor1.PWM - pin 11
  Motor1.PWM - pin 3
  Motor1.PWM - pin 6
  Motor1.PWM - pin 5

- The analog pins used by this shield for motors

  A1 , A2 , A3 , A4 .

- The final result in Figure 4.24.

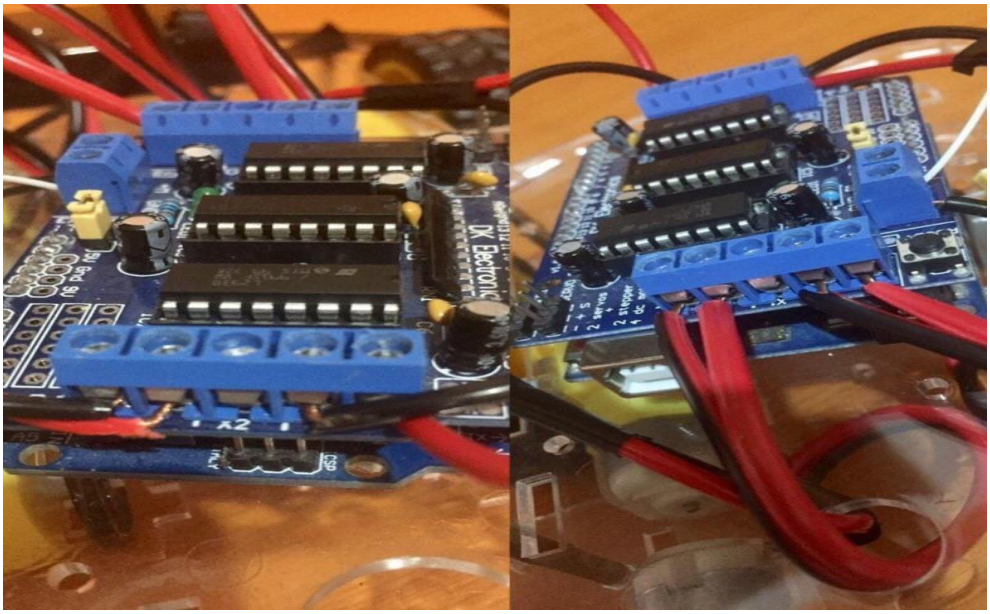

Figure 4.24 – Connection Results

3. The power will be supply as follow in (4.25) and (4.26).

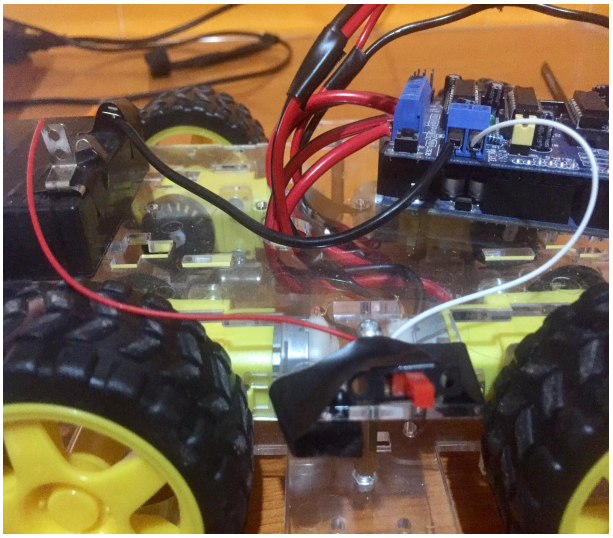

Figure 4.25 – Switch

Figure 4.26 – Battery

4. The Bluetooth will be connected to the Arduino board as showing in (4.27).

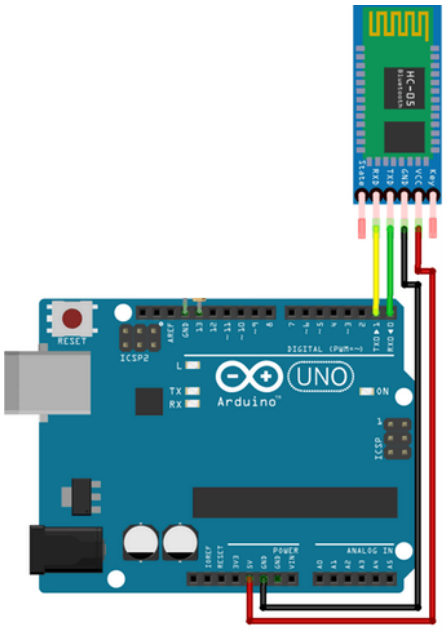

Figure 4.27 – Hc05 connection to Arduino

- **Connection pins**

  -Tx of the Arduino Board to Rx of the Bluetooth.
  -Rx of the Arduino Board to Tx of the Bluetooth.
  -5v of the Arduino Board to Vcc of the Bluetooth.
  -GND of the Arduino Board to GND of the Bluetooth.

5. The last form will be in Figure (4.28).

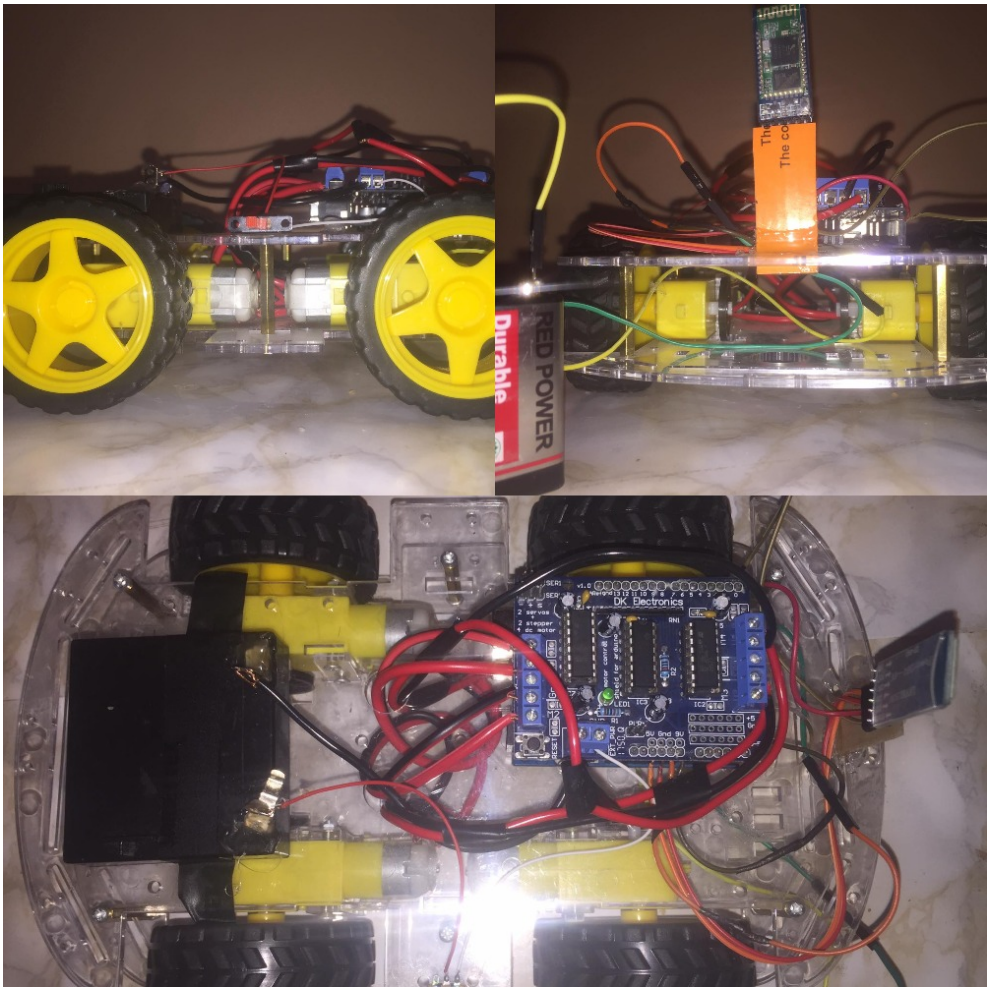

Figure 4.28 – Final car result

4.5.3 Hardware test and experimental results

In this section, we will present a comparison between a set of environment with different obstacles positions but the same start and goal position.

The comparison process was conducted by measuring in the same environment the process of time needed for the car to reach the desired goal in a specific speeds. The speed can be from 0 to the top speed 255.

We will take the next three environment for test (see the first in 4.29, second in 4.30, third in 4.31).

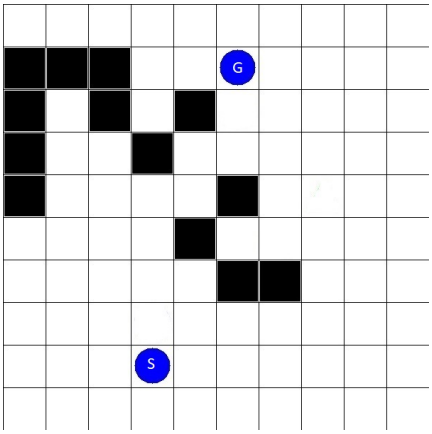

Figure 4.29 – First environment

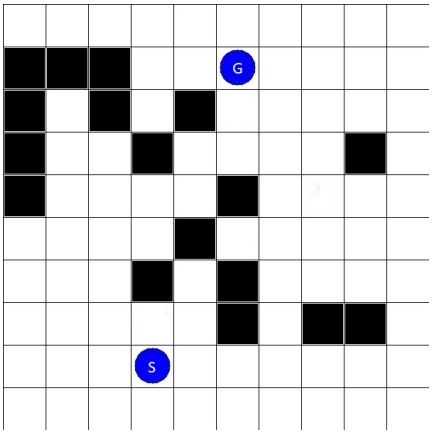

Figure 4.30 – Second environment

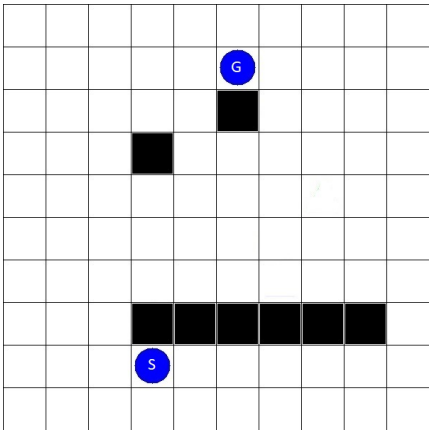

Figure 4.31 – Third environment

After entering environment pictures to the system studied, the next three figures (see 4.32, 4.33, 4.34) represent the trajectory planning result for each results.

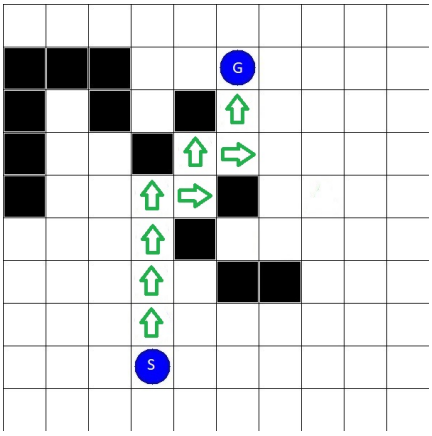

Figure 4.32 – Trajectory planning for the first environment

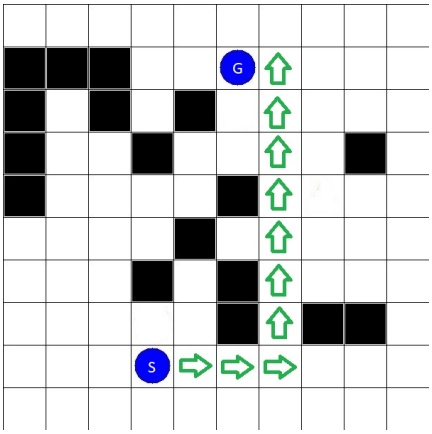

Figure 4.33 – Trajectory planning for the second environment

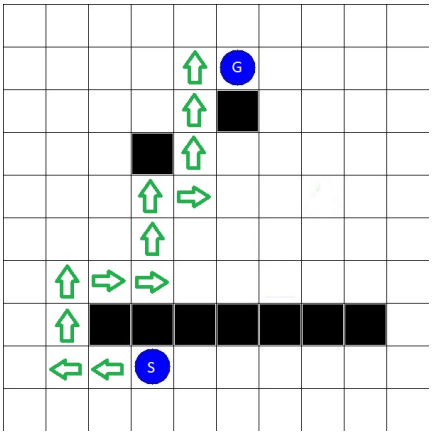

Figure 4.34 – Trajectory planning for the third environment

The next table (4.5) represent the comparison of time for each specific speed in the three environment.

Time needed to reach goal		
Environment name	**Speed**	**Time**
Environment 1	$V = 255$	00:28:58
	$V = 200$	00:29:18
	$V = 155$	00:32:40
	$V = 100$	00:24:50
Environment 2	$V = 255$	00:40:20
	$V = 200$	00:41:60
	$V = 155$	00:42:89
	$V = 100$	00:42:23
Environment 3	$V = 255$	00:51:46
	$V = 200$	00:54:54
	$V = 155$	00:56:67
	$V = 100$	00:56:80

Table 4.5 – Speed/Time in second each environment

According to the result we can conclude that the number of wire turns in an armature, the operating voltage of the motor, and the strength of the magnets all affect motor speed. If a DC motor is running on a 12 V battery, that is the maximum voltage available to the unit and the motor will only be able to perform at a speed rated for 12 V. If the battery is low and supplying less voltage, the speed will decrease accordingly. So the values of motor speed entering not only the one how control the rotation motor speed bu also the power of battery.

So the speed value, battery power affect the time, whenever The higher the speed value and battery power, the less time needed to reach the desired goal.

4.6 Project setup

After we became familiar with the hardware and software parts needs for this project, we need to make sure that we have the following set in place.

- The hardware components are set up, as described in the system design .
- The Arduino is connected to the computer using a USB cable .

- We have Python and the Python packages (pySerial) installed on The computer .
- There is no problem with the port for connection .
- The computer has the Arduino IDE and you can access the connected Arduino board through the IDE .

4.7 Conclusion

In this chapter, we have seen the mobile robot system realisation that has to go from a given start point to a given goal one by using path planning algorithm which is the a-star search algorithm. This prototype can have a lot of improvements to be done to this robot to develops it.

General conclusion

The trajectory planning is an important research field of mobile robot, which has aroused the interest of many researchers both at home and abroad. Good path planning technology of mobile robots can not only save time(users are always looking for the best results in less time), but also reduce the wear and capital investment of mobile robot.

During the project realization, we have learned knowledge about:

- Trajectory planning problems.
- Robotics and mobile robots.
- Some of planning search algorithms such as: BFS, Djikstra, A-star.
- The Arduino micro-controller and hardware components.
- How to make a bridge between a programming language such as Python and an electronic card "Arduino uno card".

We implemented BFS, Djisktra and A-star search algorithms to get the optimal set of results and to see which one is the most suitable in time, cost, path length. Without forgetting that the main ob-jective of this study is not limited just to implement the three algorithms, but also by taking the most appropriate one to corporate with the electronic card for the control of robot after implementing the necessary code to the Arduino board. We have implemented these algorithms using Python programming language, and the Arduino programming language.

In our future research, we intend to concentrate on addressing other questions which remain to resolve, some of which are:

- What if we use a dynamic environment or obstacles.
- Use a camera.
- Use a WiFi card.
- Considering an environment with multi mobile robot.

Bibliography

[1] KK Appuu Kuttan and IK Robotics. International publishing house pvt. *Ltd, India*, 2007.

[2] Jerome Barraquand, Bruno Langlois, and J-C Latombe. Numerical potential field techniques for robot path planning. *IEEE transactions on systems, man, and cybernetics*, 22(2):224–241, 1992.

[3] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[4] J Boernstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transaction on Robotics Automation*, 7(3):278–288, 1991.

[5] Johan Borenstein, HR Everett, Liqiang Feng, et al. Where am i? sensor and methods for mobile robot positioning. *University of Michigan*, 119(120):27, 1996.

[6] Oliver Brock and Oussama Khatib. Mobile manipulation: Collision-free path modification and motion coordination. In *Proceedings of the 2nd International Conference on Computational Engineering in Systems Applications*. Citeseer, 1998.

[7] Bernard Chazelle and Leonidas J Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4(6):551–581, 1989.

[8] Howie Choset and Joel Burdick. Sensor based motion planning: The hierarchical generalized voronoi graph. *Algorithms for Robot Motion and Manipulation*, pages 47–61, 1996.

[9] Thomas H Cormen, Charles Eric Leiserson, Ronald L Rivest, Philippe chetienne, and Xavier Cazin. *Introduction á l'algorithmique*. Dunod, 1994.

[10] Michael Defoort. *Contributions à la planification et à la commande pour les robots mobiles coopératifs*. PhD thesis, 2007.

[11] Pratik Desai. *Python programming for arduino*. Packt Publishing Ltd, 2015.

[12] Bernard Faverjon and Pierre Tournassoud. The mixed appraoch for path planning: Learning global strategies from a local planner. In *IJCAI*, pages 1131–1137, 1987.

[13] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[14] Rémy Guyonneau. *Méthodes ensemblistes pour la localisation en robotique mobile*. PhD thesis, 2013.

[15] Md Haque, Tanzeeb Samee, et al. *Facial recognition using dynamic image processing.* PhD thesis, BRAC University, 2017.

[16] Mark E Herrman, Dow K Hardy, Jhon F Acres, Jhon E Taylor Jr, Scott N Waller, and Francis Josef Lichtenberger. System and method for collecting and using player information, August 20 2013. US Patent 8,512,150.

[17] Daniel Ichbiah. *Robots, genése d'un peuple artificie.* Daniel Ichbiah, 2005.

[18] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[19] Maher Khatib, H Jaouni, Raja Chatila, and Jean-Paul Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 2920–2925. IEEE, 1997.

[20] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[21] Jin-Oh Kim and Pradeep Khosla. Real-time obstacle avoidance using harmonic potential functions. 1992.

[22] Philipe Lacomme, Christian Prins, and Marc Sevaux. *Algorithmes de graphes*, volume 28. Eyrolles Paris, 2003.

[23] Gerardo Lafferriere and Hector J Sussmann. A differential geometric approach to motion planning. In *Nonholonomic motion planning*, pages 235–270. Springer, 1993.

[24] Patrick lester. A* pathfinding for beginners. *online]. GameDev WebSite. http://WWW. gamedev.net/reference/articles/article2003. asp (Acesso em 08/02/2009)*, 2005.

[25] Tomas Lozano-Perez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

[26] Kelly Manley. Path-finding: from a* to lpa. *University of Minnesota*, 2003.

[27] Laetitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Cordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

[28] H Mitchell, c O'Dunlaing, and c Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[29] RK Mittal and IJ Nagrath. *Robotics and control.* Tata McGraw-Hill, 2003.

[30] Saeed B Niku. *Introduction to robotics: analysis, control, applications.* John Wiley & Sons, 2020.

[31] Nils J Nilsson. A mobile automaton : an application of artificial intelligence technique. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1969.

[32] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communication of the ACM*, 55(6):61–69, 2012.

[33] S Quinlan. Real-time path modification of collision-free paths. *These de Doctorat, Université de Stanford, Etats-Unis*, 1994.

[34] Elon Rimon and Daniel E Koditschek. Exact robot navigation using artificial potential functions. *Departmental Papers (ESE)*, page 323, 1992.

[35] Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–351, 1983.

[36] Jacob T. Schwartz and Micha Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, 37(1-3):157–169, 1988.

[37] Kang Shin and N McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control*, 31(6):491–500, 1986.

[38] Thomas Statheros, M Defoort, S Khola, Klaus D McDonald-Maier, Gareth Howells, Anne-Marie Kokosy, J Palos, W Perruquetti, Thierry Floquet, and D Boulinguez. Automated control and guidance system (acos): An overview. 2006.

[39] Bryan Stout. smart moves: intelligent path-finding. *Game developer magazine*, 10:28–35, 1996.

[40] KC Tan, Tong H Lee, and EF Khor. Evolutionary algorithms with goal and priority information for multi-objective optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 106–113. IEEE, 1999.

[41] Pierre Tournassoud. Géométrie et intelligence artificielle pour les robots. Technical report, 1988.

[42] Pierre Tournassoud. *Planification et controle en robotique,application aux robots mobiles et manipulateurs*. 1992.

[43] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[44] Jean-Jacques Wittezaele. *La double contrainte: L'influence des paradoxes de Bateson en Sciences humaines*. De Boeck Superieur, 2008.

[45] Tolga Yuksel and Abdullah Sezgin. An implementation of path planning algorithms for mobile robot based map. In *Proceedings of the 4th International Conference on Electrical and Electronics Enginnering (ELECO 205), Bursa, Turkey*, pages 7–11. Citeseer, 2005.