University of Crete

Department of Physics

Computational Vision and Robotics Laboratory

Bachelor Thesis

Autonomous Robot Navigation with Artificial Potential Fields

Author:
Michail Maravgakis

Supervisor:
Prof. Panos Trahanias

Heraklion, September 2020

Abstract

The objective of this thesis is the development of an algorithm for autonomous robot navigation in a well-known environment. The method that was used was Artificial Potential Fields (APF), which is an online reactive navigation method that has been studied extensively over the years. After an overview of the basic theory of APF, its mathematical structure is presented and explained analytically.

The development was done by using the Robot Operating System (ROS) and the testing occurred in a sophisticated simulation platform, the Gazebo. The robot that the algorithm was implemented to, was an Unmanned Aerial Vehicle (UAV), the Firefly, part of the Ascending Technologies research line. Moreover, a real-time 3D environment mapping application was constructed by using a Visual-Inertial (VI) sensor.

Although the algorithm is limited due to the assumption of prior knowledge of the environment, it manages to safely reach the goal point by creating a collision-free path. It works fast and finds the optimal path in almost every configuration that it has been tested on. Additionally, in order to solve a pretty common problem of the APF method -the local minimum trap-, an indoors navigation algorithm was developed that manages to escape and plans an alternative path. Finally, the 3D mapping produces satisfying results, by mapping all the navigated area perfectly.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, robots possess an essential part in our everyday life; both directly and indirectly, with their impact becoming greater day by day. From the production line to self-driving cars, it is certain that our life wouldn't be the same without them. More and more companies are investing in automation with the purpose of increasing the productivity and minimizing the cost. This fast-growing industry is expected to reach just under 210 billion U.S. dollars by 2025.[1]

Aerial robotics is a field of robotics that is rapidly growing popularity. In the last decade, UAVs have been researched extensively since they can be used in many fields such as agriculture, healthcare, military etc. Unmanned Aerial Vehicles (UAVs) are becoming standard platform for research in robotics because of their high maneuverability, sufficient flight endurance and the ability to move in 3D space. There is a vast variety of applications that they are already being used in, such as search and rescue missions [2], 3D environment mapping [3], crop spraying [4] and many more.

Autonomy is an issue that many researchers are trying to resolve, nevertheless it is still an open problem. There are many difficulties with fully autonomous robots and specifically UAVs, that need to be addressed to ensure that no human supervision is required. First and foremost, safety is the number one priority since some robots can cause serious injuries to humans. Moreover, due to their fragile nature, hardware damages usually occur in case of collision and such damages are expensive to fix and they can even prove to be fatal for the robot's functionality. Limited computational power is also considered to be an issue, especially in dynamic environments where complicated calculations(e.g. obstacle detection, pose estimation, trajectory planning) need to be done in real-time, in order to avoid collision with a moving object. In the case of UAVs, computational power is even more limited because of the restrictions in size and weight of the on-board hardware.

Hopefully, as technology advances, with hardware getting smaller and more powerful these restrictions will seize to exist.

Motion planning by the robot itself without any external intervention, is essential in order to achieve autonomy. A motion planning algorithm is considered to be *complete* if and only if it finds a path when one exists and *optimal* when it finds the optimal path with respect to some criterion.[5] There are numerous algorithms for path planning in aerial vehicles, with each having advantages and disadvantages. In this thesis, a motion planning algorithm is developed and implemented in a simulated UAV, with the purpose of achieving fully autonomous flight in a well-known environment.

Chapter 2

Autonomous navigation

Navigation is the process or activity to plan and direct a route or path. It is a task that an autonomous robot must do correctly in order to move safely from one location to another without getting lost or colliding with other objects. There are three general problems with the navigation process: localization, path planning and motion control.

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot, as the knowledge of the robot's own location is an essential precursor to making decisions about future actions. In a typical robot localization scenario, a map of the environment is available and the robot is equipped with sensors that observe the environment as well as monitor its own motion. Robot localization techniques need to be able to deal with noisy observations and generate not only an estimation of the robot's location but also a measure of its uncertainty [6]. The most famous and most used localization sensor for outdoor activity is the Global Position System (GPS), however the uncertainty of the output position is approximately one meter.

Path planning or find-path problem is well known in robotics and it plays an important role in the navigation of autonomous mobile robots [7]. An ideal path planner must be able to handle uncertainties in the sensed world model, to minimize the impact of objects to the robot and to find the optimum path as fast as possible especially if the path is to be negotiated regularly [7]. Path-planning problem belongs to a class of non-deterministic polynomial-time (NP) hard problems which is usually solved for realistic problems by making some assumptions and using heuristics to reduce the complexity [8]. Global path planning is a relatively well-studied research area supplied with many thorough reviews. Some common global path-planning algorithms are summarized as follows [9]:

1. *Rapidly-exploring random trees* [10]. This method is based on building a tree of possible actions to connect initial and goal configurations.

2. *Graph search algorithms.* Graph search algorithms explore a graph either for general discovery or explicit search. These algorithms carve paths through the graph, but there is no expectation that those paths are computationally optimal [11]. Some famous graph search algorithms are Dijkstra's algorithm and the A* algorithm [12].

3. *Artificial potential field methods.* They are ideally suited to online reactive navigation of robots (without path planning). These can also be used as path planning approaches, essentially by using more information about the environment. There are some variations of these methods such as adaptive potential fields [13] and evolutionary potential fields [14]

Chapter 3

Artificial potential fields

3.1 Theory

Artificial Potential Fields (APF) was first introduced in 1985 [15] by O. Khatib. He suggested this idea where the robot was treated as a point under the influence of the field generated by goals and obstacles in search spaces. Nowadays, APF is commonly used in path planning by many researchers because of its advantages such as high safety, simplicity and elegance. It is also widely applied to overcome unknown dynamic scenarios, by taking into account the state of the current environment and the robot's motion. APF is suitable for real-time applications even with slight modifications [16]. The basic concept behind artificial potential fields is to treat the robot's configuration as a point inside a potential field. More specifically, the autonomous navigation in unknown environments is being done by assigning an attractive potential to the desired goal position and a repulsive one to every obstacle. The APF approach provides a simple and effective motion planning method for practical purposes, but has a major problem which is that the robot sometimes gets trapped to local minima before reaching the goal. This problem will be explained later in detail. The forces applied on the robot are the negative gradients of the potential fields, which point towards the global minimum potential value, in other words the goal point. These forces are used to determine the direction of the robot's motion and speed of travel while avoiding collision. In the following figure , the potential fields with their gradients respectively are illustrated graphically (fig. 3.1).

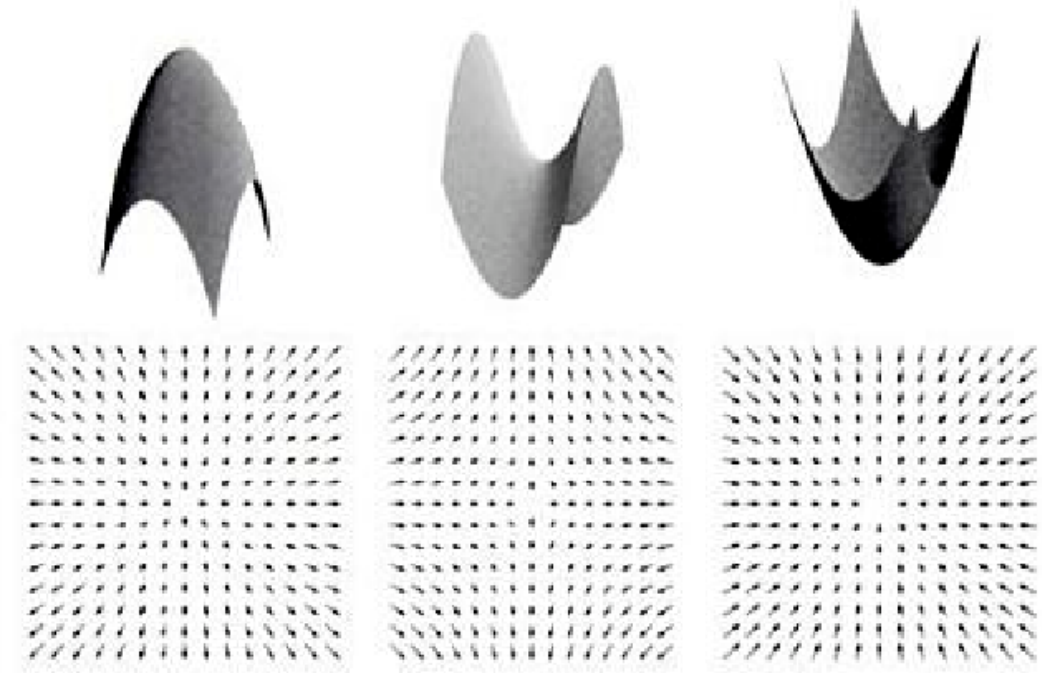

Figure 3.1: Single potentials and forces [17]

The upper graphs represent the potential fields and the bottom ones their negative gradients. The left image typifies a repulsive potential field (an obstacle). The right one shows an attractive potential field (goal point) and the middle one is a saddle somewhere in the total field where the other two co-exist. The saddle is usually formed between two close repulsive fields. The bottom graphs are the directions of the forces that are applied to the robot with respect to where the robot is located.

By combining all of the information above, it's easy to comprehend the basic principals of how a robot would be able to navigate safely through the environment and reach the goal. On top of that, in figure 3.2, the full potential field is illustrated in a space with two obstacles. The line demonstrates the trajectory that the robot will follow to find the minimum.

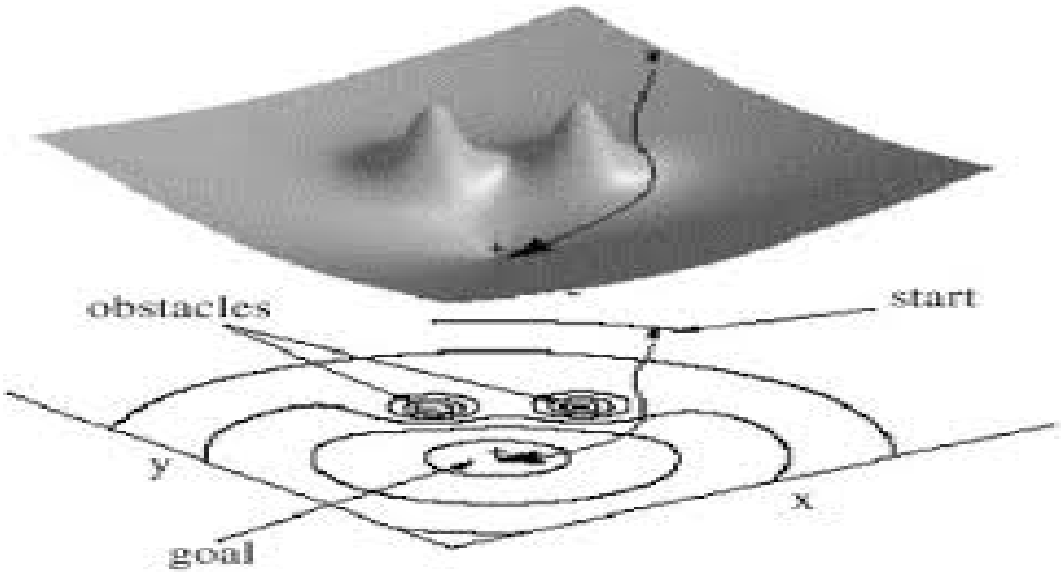

Figure 3.2: 3D illustration of a total potential field [18]

3.2 Mathematical representation

3.2.1 Potentials

In the path planning of a robot, potentials are often expressed in Cartesian workspace. Obstacles that have to be avoided are surrounded by repulsive potential fields and the goal point is surrounded by an attractive field. Consider the Cartesian coordinate of a robot in two dimensions is, $\mathbf{q} = (x, y)^T$. The APF function can be represented as

$$U(\mathbf{q}) = U_{att}(\mathbf{q}) + U_{rep}(\mathbf{q}) \tag{3.1}$$

where

$$\begin{array}{rcl} U(\mathbf{q}) & = & \text{Artificial potential field.} \\ U_{att}(\mathbf{q}) & = & \text{Attractive field.} \\ U_{rep}(\mathbf{q}) & = & \text{Repulsive field.} \end{array}$$

Note that $U_{rep}(\mathbf{q})$ is the total potential of all the obstacles that affect the robot. More precisely,

$$U_{rep}(\mathbf{q}) = \sum_{i=1}^{n} U_{rep}^{i}(\mathbf{q}) \tag{3.2}$$

with n being the number of obstacles.

The most common attractive field is the quadratic well [19], which is described as

$$U_{att}(\mathbf{q}) = \frac{1}{2} k_a |\mathbf{q} - \mathbf{q_d}|^2 \tag{3.3}$$

where

k_a	=	Attractive coefficient.
$\mathbf{q}$	=	Current position vector of the robot.
$\mathbf{q_d}$	=	Goal position vector.

Before the repulsive potential is formulated, it must be noted that when the robot is far away from an obstacle, it shouldn't get repelled by it at all. In order for this to happen, the repulsive potential function should be branched. Now, the repulsive potential [20] will be

$$U_{rep}(\mathbf{q}) = \begin{cases} \frac{1}{2} k_r (\frac{1}{d(\mathbf{q},\mathbf{q}_{obs})} - \frac{1}{d_0})^2 & if \quad d(\mathbf{q}, \mathbf{q}_{obs}) \leq d_0 \\ 0 & if \quad d(\mathbf{q}, \mathbf{q}_{obs}) > d_0 \end{cases} \tag{3.4}$$

where

k_r	=	Repulsive coefficient.
$d(\mathbf{q}, \mathbf{q}_{obs})$	=	Current distance between the robot and an obstacle.
d_0	=	Distance threshold of the repulsive force field.

As the distance between the robot and the obstacle is getting smaller the repulsive field is getting bigger with the purpose of avoiding potential collision. Finally, the distance threshold is set manually by the user and its value depends on the robot's speed, the available space and other parameters with regard of the given circumstances (e.g. wind speed, human presence etc.).

In figure 3.4, it is represented step by step how the final visualization of (3.1) is derived by the above mathematical equations (3.3) and (3.4).

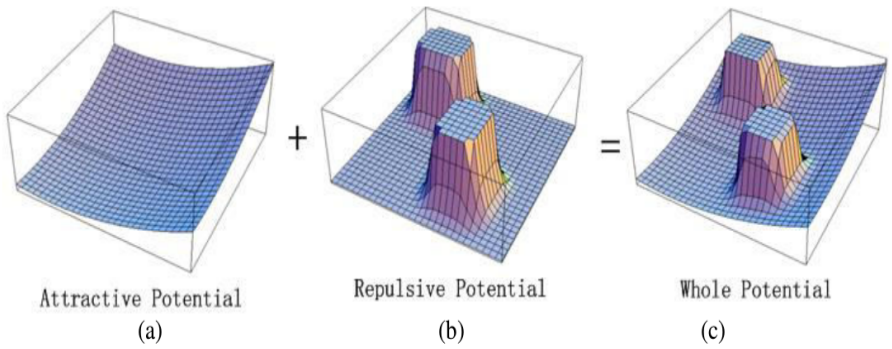

Figure 3.3: Components of total potential field [16]

3.2.2 Forces

The forces applied on the robot, are the negative gradients of each potential respectively. Without further ado, the total force is

$$\begin{aligned} \mathbf{F}(\mathbf{q}) &= -\boldsymbol{\nabla} U(\mathbf{q}) \\ &= -\boldsymbol{\nabla} U_{att}(\mathbf{q}) - \boldsymbol{\nabla} U_{rep}(\mathbf{q}) \\ &= \mathbf{F}_{att}(\mathbf{q}) + \mathbf{F}_{rep}(\mathbf{q}) \end{aligned} \tag{3.5}$$

where

$$\begin{array}{rcl} \mathbf{F}(\mathbf{q}) & = & \text{Total force.} \\ \mathbf{F}_{att}(\mathbf{q}) & = & \text{Attractive force.} \\ \mathbf{F}_{rep}(\mathbf{q}) & = & \text{Repulsive force.} \end{array}$$

The components of $\mathbf{F}(\mathbf{q})$ are computed as follows
Attractive force:

$$\begin{aligned} \mathbf{F}_{att}(\mathbf{q}) &= -\boldsymbol{\nabla} U_{att}(\mathbf{q}) \\ &= -k_a(\mathbf{q} - \mathbf{q}_d) \end{aligned} \tag{3.6}$$

$\mathbf{F}_{att}(\mathbf{q})$ is a direct vector towards $\mathbf{q}_d$ with magnitude proportional to the distance between $\mathbf{q}$ and $\mathbf{q}_d$.

Repulsive force:

$$\mathbf{F}_{rep}(\mathbf{q}) = \begin{cases} k_r(\frac{1}{d(\mathbf{q},\mathbf{q}_{obs})} - \frac{1}{d_0})\frac{1}{d^2(\mathbf{q})}\boldsymbol{\nabla} d(\mathbf{q},\mathbf{q}_{obs}) & if \quad d(\mathbf{q},\mathbf{q}_{obs}) \leq d_0 \\ 0 & if \quad d(\mathbf{q},\mathbf{q}_{obs}) > d_0 \end{cases} \tag{3.7}$$

with $d(\mathbf{q},\mathbf{q}_{obs})$ being the Euclidean distance between the robot and an obstacle. By assuming that $\mathbf{q} = (x,y,z)^T$ and $\mathbf{q}_{obs} = (x',y',z')^T$ then $d(\mathbf{q},\mathbf{q}_{obs})$ will be

$$d(\mathbf{q},\mathbf{q}_{obs}) = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2} \tag{3.8}$$

With that in mind, it is now feasible to compute the gradient term of (3.7)

$$\begin{aligned} \boldsymbol{\nabla} d(\mathbf{q},\mathbf{q}_{obs}) &= \frac{\partial d}{\partial x}\hat{i} + \frac{\partial d}{\partial y}\hat{j} + \frac{\partial d}{\partial z}\hat{k} \\ &= \frac{x-x'}{d(\mathbf{q},\mathbf{q}_{obs})}\hat{i} + \frac{y-y'}{d(\mathbf{q},\mathbf{q}_{obs})}\hat{j} + \frac{z-z'}{d(\mathbf{q},\mathbf{q}_{obs})}\hat{k} \end{aligned} \tag{3.9}$$

The repulsive force points to the opposite direction of the obstacle's position. The forces mentioned above (3.5),(3.6) and (3.7), are displayed in the following simple figure in two dimensions

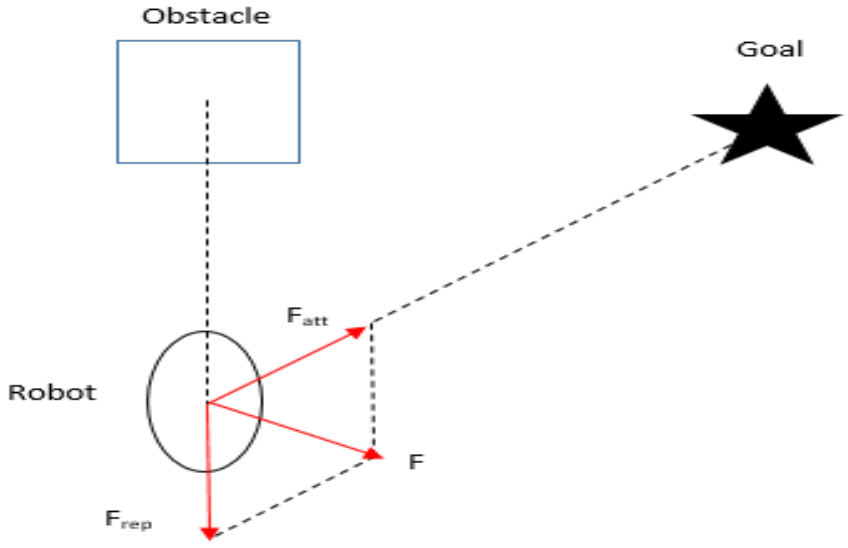

Figure 3.4: Forces applied on a robot

3.3 Local minimum problem

APF methods provide simple and efficient motion planners for practical purposes, however it is widely known that sometimes it is possible to be trapped in local minimum situations [21]. This problem occurs due to the fact that in certain environment configurations, the gradient of the field becomes zero in some locations even though none of these are the goal point. This causes the robot to stand there still or to oscillate around the minimum with no way of escaping and get back on track. Despite the fact that this problem is known for a long time and a significant amount of research has been put on solving it, an optimal way of dealing with it, is yet to be discovered. In the following figure the problem of local minimum is presented for a random configuration (fig. 3.5).

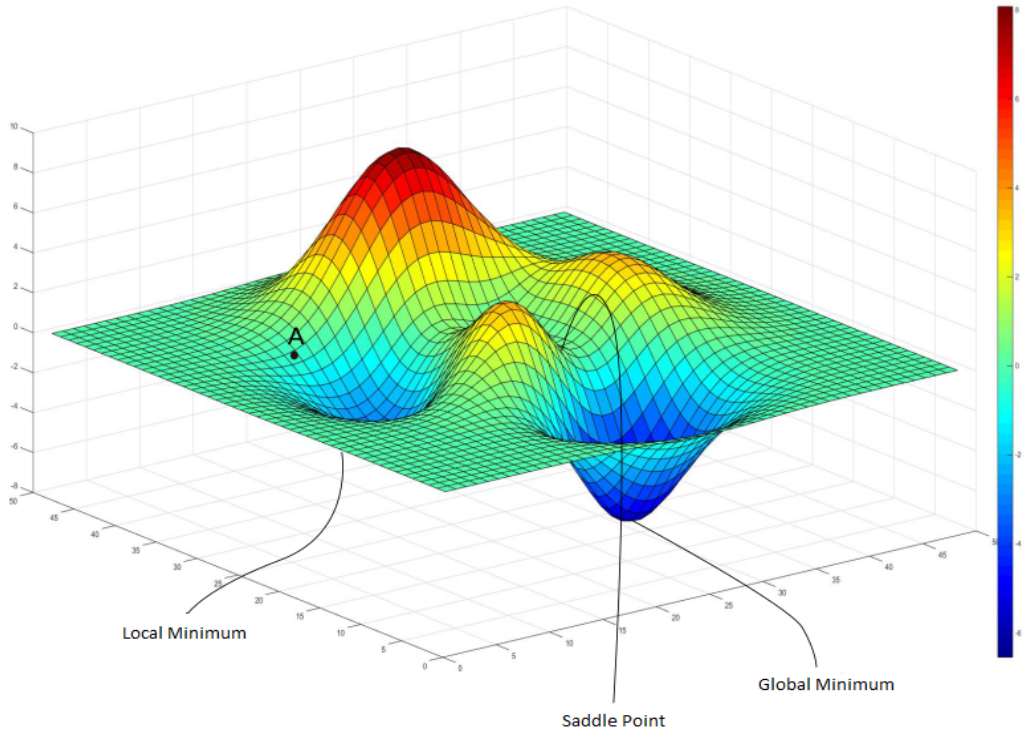

Figure 3.5: Local minimum demonstration [22]

Chapter 4

Software and platform

In this chapter, the software that has been used throughout this thesis, along with the platform, are presented. More specifically, the following sections contain all the tools used for the development of the algorithm, the simulation environment, the visualization tools for 3D mapping and finally the operating system as well as the programming languages.

4.1 Robot Operating System

The Robot Operating System[1] (ROS) is an open source flexible framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It is extremely useful and has many reliable drivers for communicating with the hardware of the platform. The latter is very helpful, because it makes it easy to directly create high level control applications without concerning about all the necessary low level firmware. Software developed in ROS, is easily integrated from one platform to another and thus it has a very active ecosystem with members all around the globe, sharing their software for everyone to use. Moreover there are many tutorials and information, that help a lot to grasp the structure and functions, especially for newer users.

ROS processes are represented as nodes in a graph structure, connected by edges called topics. ROS nodes can pass messages to one another through topics, make service calls to other nodes, provide a service for other nodes, or retrieve shared data from a communal database called the parameter server. A process called the ROS Master makes all of this possible by registering nodes to itself, setting up node-to-node communication for topics, and con-

[1] https://www.ros.org

trolling parameter server updates [23]. Messages and service calls do not pass through the master, rather the master sets up peer-to-peer communication between all node processes after they register themselves with the master. Next, a brief review of each component of ROS is listed.

4.1.1 Master

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another [24].

4.1.2 Nodes

Nodes are single processes that perform computations. They are combined together into a graph and communicate with one another using streaming topics, services, and the Parameter Server [25]. They are executables usually written in C++ or Python.

4.1.3 Topics

Topics are the way that nodes exchange messages. When a node needs information, it subscribes to a certain topic, when it wants to give information, it publishes to a topic. Topics are essential for connecting individual nodes in order to perform complicated tasks [26].

4.1.4 Messages

ROS uses a simplified messages description language for describing the data values that nodes publish. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages [27].

4.1.5 Services

The publish / subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for Remote Procedure Call (RPC) request / reply interactions, which are often required in a distributed system. Request / reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply. A providing

ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply [28].

4.2 Gazebo

Gazebo[2] is an open-source 3D robotics simulator, used by many researchers and companies all around the world. It provides a simple and robust platform, where anyone can rapidly test algorithms, design robots and even train AI algorithms in realistic environment. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Lastly, it supplies its users with a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.

4.3 RotorS

RotorS [29] is a ROS package created by ETH Zurich, it contains very useful tools for development such as low level controller, ROS msgs, URDFs, waypoint publisher nodes etc. It is built on top of gazebo and has all the necessary files to begin working on the drone, both in simulation and in real-life. The transition from the simulated UAV to real one, is supposed to be straightforward by only making some minor changes. Finally, this package contains a variety of examples and tutorials on how to use, making easy to implement and get a quick start on high level programming without having to worry about the dynamics of flight and the hardware.

4.4 Rviz

Rviz[3] (ROS Visualization) is a ROS graphical interface, that allows the user to visualize data obtained from ROS topics. These topics usually contain data from a variety of sensors. It is a very useful tool especially in projects that include computer vision. Rviz displays 3D sensor data from stereo cameras, lasers, Kinects, and other 3D devices in the form of point clouds or depth images. 2D sensor data from webcams, RGB cameras, and 2D laser rangefinders can be viewed in rviz as image data [30].

[2] http://gazebosim.org
[3] http://wiki.ros.org/rviz

4.5 Versions

In the following table are listed all the versions for the software mentioned above as well as the operating system and the programming languages used.

Table 4.1: Software versions

Software	Version
ROS	Kinetic Kame
Gazebo	7.16.0
Rviz	1.12.17
Ubuntu	Xenial Xerus 16.04.6
C++	C++11
Python	2.7.12

4.6 Platform

Although the algorithm was developed and implemented in a simulated environment, the UAV exists and its specifications were transferred to the simulation via a Unified Robot Description Format (URDF) file. The URDF contains all the necessary information of the robot model (e.g. size, weight, rotors etc.). The simulated UAV is the Firefly (fig 5.1), a hexacopter created by Ascending Technologies mainly for research purposes. It is equipped with a low level controller, a GPS, an Inertial Measurement Unit (IMU) sensor and finally an on-board processor for high level development.

Figure 4.1: AscTec Firefly

Chapter 5

Implementation

In this chapter, after a brief overview of the reasons that led to the development of autonomous navigation software with the APF method, the created algorithm will be presented and explained extensively. Moreover, some other noteworthy attempts and ideas will be mentioned and compared with the main algorithm and lastly the tools used for the 3D environment mapping application will be briefly mentioned.

Firstly, the objective of this thesis was to create an algorithm with the purpose of making a UAV fly autonomously in a well-known environment. By "well-known environment", is meant the knowledge not only of the positions of every obstacle but also the drone's. This of course, is ideal and even thought it is possible to recreate in real life scenarios, it is constrained to highly controlled and static environments, with limited applications. Nevertheless, it is often very useful in the context of developing and testing a motion planning algorithm. The APF method was chosen because it is simple, elegant and mainly because it has low demands on computational power. Moreover, it has high maneuverability, meaning that the adjustment of this method to new situations can be made effortlessly. For example, if the purpose is to go from point A to point B where A and B are far away in outdoor environment, one could tweak the constants ($k_r, d0of eq.3.4$) to make the UAV fly furthest from the detected obstacles. On the contrary, for indoors navigation the drone should fly closer to the obstacles due to space limitations.

Finally, it should be mentioned here that the APF method as described in chapter 3, was slightly reformulated. Instead of computing the forces applied on the UAV, the algorithm only computes several total potential field values and finds the minimum in each move. This process will be over when it finds the global minimum. How this happens, will be explained further in the following sections.

5.1 Grid construction

For the UAV to move in the right direction in each instance, it needs to know the value of the total artificial potential field. This value is computed and stored inside a grid each time the UAV changes its position. To be more precise, the algorithm computes twenty six values of the potential field and stores them inside a 3x3x3 grid where each cell represents a point in the 3D space. The drone's position is the central cell (2,2,2) and that is the reason it computes twenty six instead of twenty seven values. The following figure (fig 5.1), presents a visualization of the grid.

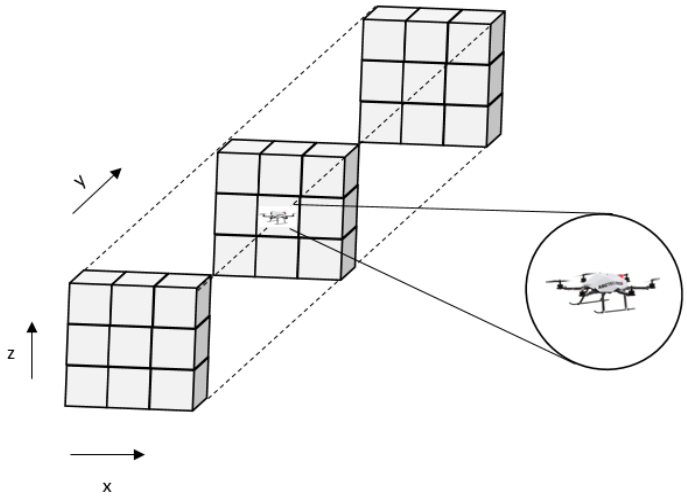

Figure 5.1: 3D grid around the UAV

Each one of the 27 cells (except the central one) contains a float value inside it, which represents the total potential field $U_{tot}^{(i)}(\mathbf{q}_i)$ in that point. This value is computed by adjusting equation 3.1 to fit the above layout.

$$U_{tot}^{(i)}(\mathbf{q}_i) = U_{att}^{(i)}(\mathbf{q}_i) + \sum_{j=1}^{n} U_{rep,j}^{(i)}(\mathbf{q}_i) \tag{5.1}$$

where

i = A specific cell of the grid.
$\mathbf{q}_i$ = Position vector of the ith*cell*.
n = The number of obstacles that interact with the robot.

Each cell represents a fixed-size shift with respect to the UAV's position, this shift is just a parameter that could be changed by the user. For example, let $\mathbf{q} = (x_d, y_d, z_d)^T$ be the position vector of the UAV and L the distance between two adjacent cells. By cutting out a slice of the grid, a plane 3x3 is formed and let's assume that this plane is the $x - y$ with z fixed at z_d. The following figure (fig. 5.2) demonstrates the equivalence between each cell of the 2D grid and its spatial coordinates with respect to the UAV's position.

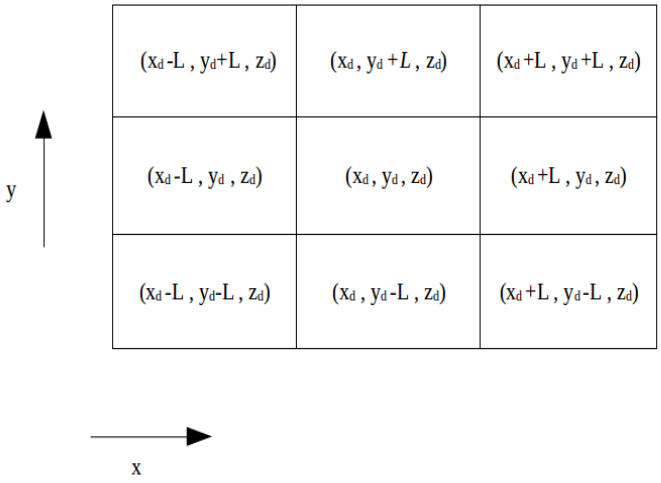

Figure 5.2: Slice of grid with displacements

According to figure 5.2, the central cell contains the UAV's position vector, and the other eight its closest displacements. The other two levels, $z_d + L$ and $z_d - L$ can be constructed accordingly.

5.2 Algorithmic procedure

Now that the notation has been well established, the procedure that the algorithm follows in order to find in which cell to move will be explained extensively. Once it finds out which cell this is, it publishes the correspondent waypoint with respect to the UAV's position and it moves there.

To start with, the software created consists of three ROS nodes. The task of the first node is only to extract the initial UAV position from the gazebo simulator. This is done by subscribing to a topic created by the gazebo itself, where it publishes the positions of all the objects inside the simulation. The second one does a similar job; it extracts the number of the obstacles that exist in the simulator (added by the user) and their positions.

The positions are just the Cartesian coordinates of the center of mass of each object. The last node is the main algorithm; it uses all the information obtained by the other two, also stores the desired goal position (user input), the set of parameters for the APF method and the cell displacement's length from a file. These APF parameters are the two constants for repulsive and attractive potential k_r and k_a respectively and the distance threshold d_0 for the repulsion. The cell displacement is L according to the notation of section 5.1. These parameters are stored in a file with purpose of making it easier to change them and test the algorithm without having to interact with the code itself. After obtaining all the aforementioned information, the goal is to output the desired coordinates in which the UAV will move to, and also the desired orientation.

After acquiring all the necessary information, a loop is created. The iterations of this loop will stop when the UAV's position vector is equal to the goal's one. Inside this loop, the first step is to create a 3x3x3 grid as mentioned in section 5.1. This specific grid is used only for one step and then a new one replaces it. After that, only the attractive potential field, generated by the goal point, is computed and stored inside the grid for each cell's position (see fig. 5.2). Since each cell expresses a different location, the values inside them are not identical. Afterwards, it checks if any of the twenty six cells is inside the range of any obstacle. If it is, the repulsive potential will be computed and added to the current value. Finally, it searches for the minimum value and after it finds it, the UAV moves to that specific cell's location and the process is repeated until the central cell (2,2,2) is equal to the goal's position.

This heuristic approach on implementing the artificial potential fields method for autonomous UAV navigation was made with the purpose of creating a fast and robust algorithm for real-time exploration. It could be possible to analyze a lot more data from a variety of available sensors, but this would have made the algorithm much more computational demanding and thus very restricted.

The whole procedure described above is presented below in a more compact way by using pseudocode.

Algorithm 1: Autonomous navigation with APF

```
GetParameters(k_a, k_r, d_0, L);

obstacle_pos ← GetObstaclePos();
drone_pos ← GetDronePos();
goal_pos ← GetGoalPos();

while goal_pos ≠ drone_pos do
    array grid(3x3x3) ;
    addAttractivePotential(grid, k_a, L) ;

    for i ← 1 to 27 do
        distance ← FromObstacle(drone_pos, obstacle_pos, L);
        if distance ≤ d_0 then
            addRepulsivePotential(grid,i,k_r,d_0,obstacle_pos);
        else
            continue;
        end
    end
    New_Position ← FindMinimumValuePos(grid);
    drone_pos ← New_Position
end
```

Finally, an rqt graph (fig. 5.3) provided by ROS, illustrates all the ROS nodes and topics that are up and running, including their connections, while the algorithm is executed. Rqt graph is a Graphical User Interface (GUI) plugin used for visualization. The square boxes indicate a topic while oval shaped ones are the nodes. The output from this procedure will be discussed further below.

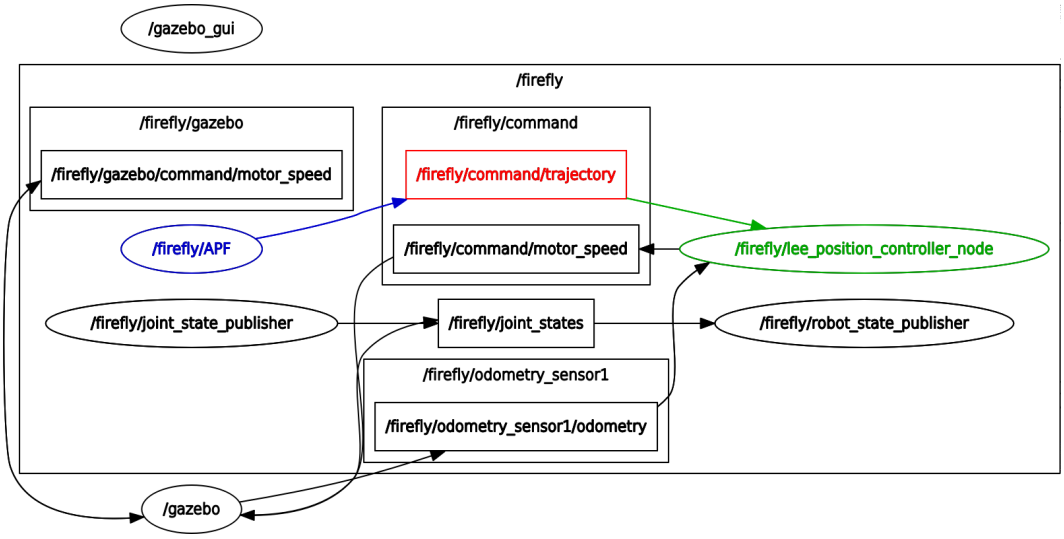

Figure 5.3: rqt graph

5.3 Indoors navigation algorithm

Throughout the development process, in an effort to fix the local minimum problem (section 3.3), a very similar algorithm to the original was constructed. This method creates again a 3D grid, but this time the size of the grid is larger and user dependent. This implementation works explicitly for indoors navigation or more generally, in small-sized areas. The user must give as input the dimensions of the room, then the algorithm creates a grid with dimensions proportional to the size of the room divided by the desired displacement between two neighbouring cells. Let's assume that the input dimensions of the room are X_R, Y_R, Z_R and L the displacement as described in section 5.1. The size of the grid would be:

$$Grid\_Size = \frac{X_R \cdot Y_R \cdot Z_R}{L^3} \tag{5.2}$$

where $Grid\_Size$ is the number of elements that can be stored inside the grid and L represents the length of displacement in each step. As L gets smaller the accuracy of the navigation gets bigger and thus this method is very limited, because the size of the grid (eq. 5.2) becomes enormous either by big inputs or small L.

Nevertheless, it is quite efficient dealing with the local minimum problem, since the grid remains constant throughout the execution and the information

inside it is not lost after each iteration. With that in mind, after each step, a constant is added to the existing value of the previous cell. By doing that, in case the UAV gets trapped in a local minimum, it will oscillate a few times and each time it will add a constant to its previous position (i.e. cell in the grid) and eventually escape by following a less optimum trajectory.

5.4 3D environment mapping

In this section, an application for real-time 3D environment mapping is presented. For the mapping process to be feasible, an estimation of depth is required. With that in mind, a Visual Inertial (VI) sensor (fig. 5.4) was added to the simulated UAV with the purpose of generating a point cloud dataset and using it to map the environment.

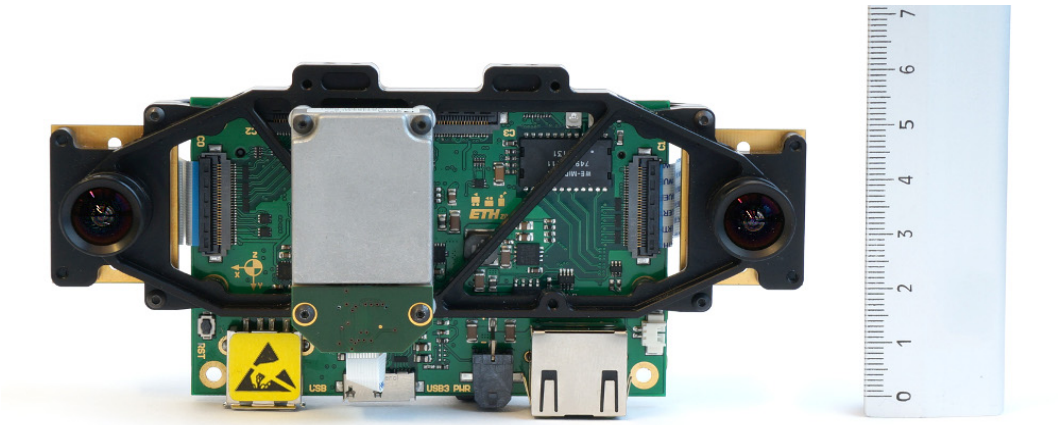

Figure 5.4: VI sensor

The VI sensor was developed by Autonomous Systems Lab (ASL), ETH Zurich and Skybotix, both the hardware and software [31]. This sensor provides fully time-synchronized and factory calibrated IMU and stereo-camera data streams [32]. Stereo cameras are able to calculate distance, by computing the disparity of each pixel of the two images taken by each camera.

The package used for the 3D mapping is *octomap* [33]. It takes the data from the stereo camera and builds a 3D occupancy grid mapping approach and also provides data structures and mapping algorithms in C++ particularly suited for robotics. It does all of the above without having any prior knowledge of the environment and in real time.

Chapter 6

Results and discussion

After extensive testing of the algorithm in a variety of different configurations, it is safe to say that in almost every case, it manages to achieve the goal without collisions.[1]. The linked video of the footnote demonstrates the algorithm in action. These particular configurations were created randomly in order to display the maneuverability of the UAV in avoiding obstacles by moving both vertically and horizontally. It is fast, optimal and complete because it finds the shortest possible trajectory to the goal position.

Even thought there is prior knowledge of the obstacle positions, something that's impracticable in real life, it could be used in a laboratory or a highly controlled environment for various testing experiments such as low level controller functionality, velocity control, maneuverability testing, battery endurance etc. Although, the algorithm is robust, it gets trapped sometimes in local minima[2], due to the nature of the APF method. It is unlikely to predict when this is going to happen or in what precise configuration it is possible to occur, thus testing for this particular problem is very difficult.

The 3D environment mapping procedure, works perfectly and it should be trivial to implement in a real camera as long as it is stereo. It successfully recreates a map of the navigated area, making it very useful for a variety of applications. Some results of the mapping are presented in the next figures.

[1] https://www.youtube.com/watch?v=Pul8T9lTZyI
[2] https://www.youtube.com/watch?v=wVnaOq3JV_o

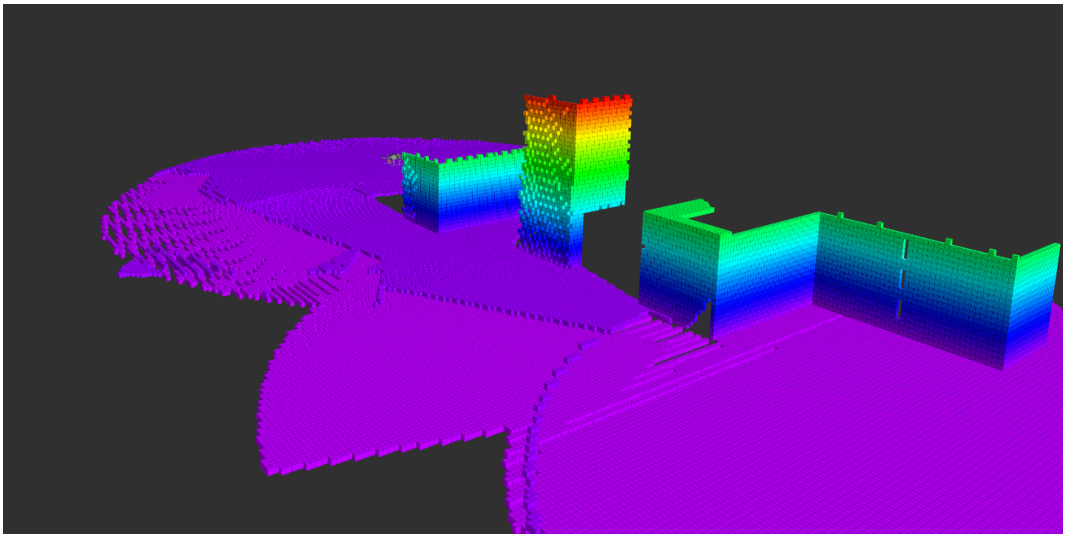

Figure 6.1: 3D environment mapping (1)

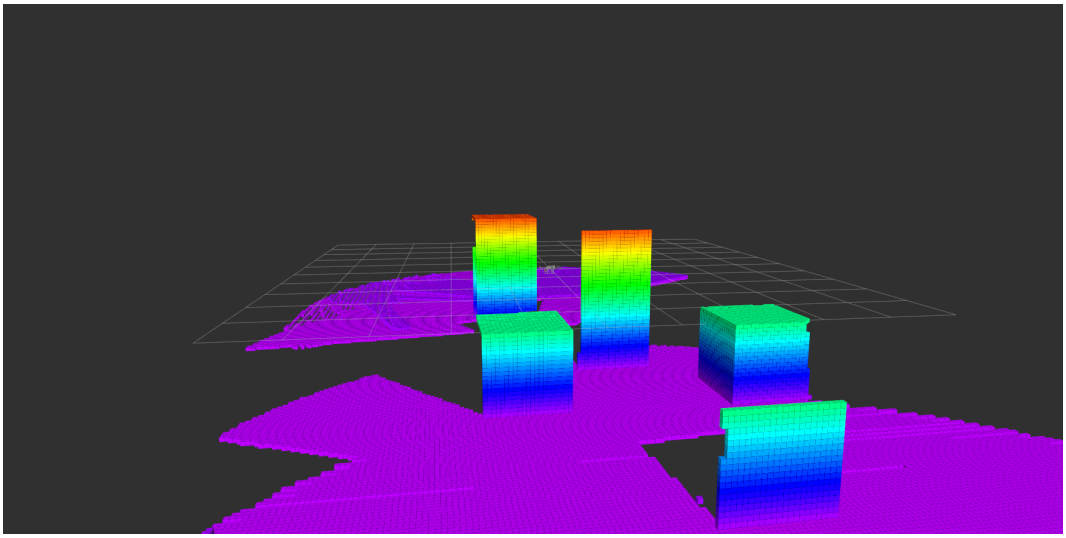

Figure 6.2: 3D environment mapping (2)

The procedure of creating the mapping in figure 6.1 is presented in this video (https://www.youtube.com/watch?v=LuHsqYgnkdg). It was done by using the rviz as mentioned in section 4.4.

Except from the local minimum problem, there is something more that needs to be addressed. Because of the fact that vision has not been added to the simulated UAV, it is impossible to know the exact size of the obstacle. As mentioned before, the input of each obstacle position is a set of Cartesian coordinates (x,y,z), representing the center of mass. According to this, it is impossible for the algorithm to avoid obstacles -regardless of their size- by using only a single set of parameters (k_r, d_0). For example, let's assume that the parameter d_0 is big enough in order to avoid efficiently small objects, but if an object with size greater than d_0 exists in the environment, a collision will occur. This problem could be fixed by using vision for 3D object detection and localization. Then, it would be feasible to extract some key points on the surface of each obstacle, let's say the edges, and add a repulsive potential to each one of them. Even thought this would be more computationally demanding due to the fact that there will be more repulsive potentials instead of one for each obstacle, this approach could ensure a collision-free path. To address this problem, during the testing of the algorithm all the obstacles where one-sized boxes.

Finally, the indoors navigation algorithm manages to escape from local minima traps very efficiently. After some oscillations around the trap point, it recreates a different path and manages to find the goal point successfully. In this video (https://www.youtube.com/watch?v=z6BQ7kkrmQI), there is a demonstration of how it manages to deal with a situation like this, in reasonable amount of time.

Further research

At this point, it should be clear the idea behind the algorithm and how it manages to find the goal position without colliding. All of the work that has been done in this thesis, should be transferred to the real UAV in order to see how close or how far apart is from autonomy and test its robustness more extensively. According to Fadri Furrer et al. [29], the implementation from the simulated to the real UAV should be very reliable and work almost the same as it did in the simulation.

Although the algorithm is a good step towards autonomy, it still lacks a very basic feature, the vision. By using input data from cameras, it would be able to detect not only the position of the obstacles, but also their sizes, making the algorithm much more efficient. Also by using vision techniques, the UAV would be able to locate its own position by using the Simultaneous Localization and Mapping technique (SLAM).

Bibliography

[1] Melania Scerra. *Size of the global market for industrial and non-industrial robots between 2018 and 2025*. URL: https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/. (Apr 16, 2020).

[2] A. A. Meera et al. "Obstacle-aware Adaptive Informative Path Planning for UAV-based Target Search". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 718–724.

[3] Fabio Remondino et al. "UAV photogrammetry for mapping and 3D modeling-Current status and future perspectives". In: vol. XXXVIII-1/C22. Jan. 2011. DOI: 10.5194/isprsarchives-XXXVIII-1-C22-25-2011.

[4] Rahul Desale et al. "Unmanned Aerial Vehicle For Pesticides Spraying". In: 5 (Apr. 2019), pp. 79–82.

[5] C. Goerzen, Z. Kong, and B. Mettler. "A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance". In: 57 (Nov. 2009).

[6] Shoudong Huang and Gamini Dissanayake. "Robot Localization: An Introduction". In: Aug. 2016. DOI: 10.1002/047134608X.W8318.

[7] N. Sariff and N. Buniyamin. "An Overview of Autonomous Mobile Robot Path Planning Algorithms". In: *2006 4th Student Conference on Research and Development*. 2006, pp. 183–188.

[8] Mohammadreza Radmanesh et al. "Overview of Path Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study". In: *Unmanned Systems* 6 (Apr. 2018), pp. 1–24. DOI: 10.1142/S2301385018400022.

[9] A.S. Matveev et al. *Safe Robot Navigation Among Moving and Steady Obstacles*. Jan. 2015, pp. 21–49.

[10] S. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[11] M. Needham and A.E. Hodler. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, Incorporated, 2019. ISBN: 9781492047681. URL: https://books.google.gr/books?id=UwIevgEACAAJ.

[12] Akshay Kumar Guruji, Himansh Agarwal, and D.K. Parsediya. "Time-efficient A* Algorithm for Robot Path Planning". In: *Procedia Technology* 23 (2016). 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05-06 February, 2016, pp. 144–149. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2016.03.010. URL: http://www.sciencedirect.com/science/article/pii/S2212017316300111.

[13] F. Arambula Cosío and M.A. Padilla Castañeda. "Autonomous robot navigation using adaptive potential fields". In: *Mathematical and Computer Modelling* 40.9 (2004), pp. 1141–1156. ISSN: 0895-7177. DOI: https://doi.org/10.1016/j.mcm.2004.05.001. URL: http://www.sciencedirect.com/science/article/pii/S0895717704003097.

[14] P. Vadakkepat, Kay Chen Tan, and Wang Ming-Liang. "Evolutionary artificial potential fields and their application in real time robot path planning". In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*. Vol. 1. 2000, 256–263 vol.1.

[15] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, pp. 500–505.

[16] Sabudin elia nadira, Rosli Omar, and Che Ku Nor Hailma. "Potential field methods and their inherent approaches for path planning". In: 11 (Jan. 2016), pp. 10801–10805.

[17] Kadir Firat Uyanik. "A study on Artificial Potential Fields". In: 2011.

[18] Jean Bosco Mbede, Xinhan Huang, and Min Wang. "Fuzzy motion planning among dynamic obstacles using artificial potential fields for robot manipulators". In: *Robotics and Autonomous Systems* 32.1 (2000), pp. 61–72. ISSN: 0921-8890. DOI: https://doi.org/10.1016/S0921-8890(00)00073-7. URL: http://www.sciencedirect.com/science/article/pii/S0921889000000737.

[19] Q. Zhu, Y. Yan, and Z. Xing. "Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing". In: *Sixth International Conference on Intelligent Systems Design and Applications*. Vol. 2. 2006, pp. 622–627.

[20] G. Li et al. “An efficient improved artificial potential field based regression search method for robot path planning”. In: *2012 IEEE International Conference on Mechatronics and Automation*. 2012, pp. 1227–1232.

[21] M.G. Park and M.C Lee. “A new technique to escape local minimum in artificial potential field based path planning”. In: *KSME International Journal* 17.9 (2003), pp. 1876–1885. ISSN: 0895-7177. DOI: https://doi.org/10.1007/BF02982426.

[22] Ayoosh Kathuria. *Intro to optimization in deep learning: Gradient Descent*. URL: https://mc.ai/intro-to-optimization-in-deep-learning-gradient-descent/. (Jun 1, 2018).

[23] URL: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes. (Sep 01, 2020).

[24] URL: http://www.wiki.ros.org/Master. (Sep 01, 2020).

[25] URL: http://wiki.ros.org/Nodes. (Sep 01, 2020).

[26] URL: http://wiki.ros.org/Topics. (Sep 01, 2020).

[27] URL: http://wiki.ros.org/msg. (Sep 01, 2020).

[28] URL: http://wiki.ros.org/Services. (Sep 01, 2020).

[29] Fadri Furrer et al. “Robot Operating System (ROS): The Complete Reference (Volume 1)”. In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[30] Carol Fairchild and Thomas L. Harman. *ROS Robotics By Example*. Packt Publishing, 2016. ISBN: 1782175199.

[31] Janosch Nikolic et al. “A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM”. en. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014 IEEE International Conference on Robotics and Automation (ICRA 2014); Conference Location: Hong Kong, China; Conference Date: May 31 - June 7, 2014. Piscataway, NJ: IEEE, 2014, pp. 431–437. ISBN: 978-1-4799-3685-4. DOI: 10.3929/ethz-a-010061790.

[32] URL: http://wiki.ros.org/vi_sensor. (2017-10-16).

[33] Armin Hornung et al. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013). Software available at http://octomap.github.com. DOI: 10.1007/s10514-012-9321-0. URL: http://octomap.github.com.