! POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master Degree Thesis

Multirotor Design and Modeling for Indoor Applications

Supervisors
Prof. Elisa Capello
Dr. Davide Carminati
Dr. Iris David Du Mutel

Candidate
Giorgio Baudino
266295

July 2021

Acknowledgments

Vorrei ringraziare tutti coloro che mi hanno supportato in questo percorso, sarò breve perché come tutti quelli che mi conoscono sanno sono tirato coi tempi. Per prima cosa vorrei ringraziare la Professoressa Elisa Capello che mi ha permesso di fare questo interessantissimo percorso di tesi e tutti i dottorandi passati e prossimi futuri del gruppo di ricerca che mi hanno aiutato in qualsiasi istante della tesi e hanno permesso che riuscissi a concludere questo percorso: Matteo Scanavino, Davide Carminati, Iris du Mutel e Enza Trombetta ex tesista con me e poi diventata mio superiore negli ultimi due mesi senza la quuale non avrei potuto concludere la tesi. Vorrei poi ringraziare tutta la mia famiglia che mi ha sempre supportato e sono sicuro continuerà a fare. Vorrei altrettanto ringraziare tutti i miei amici partendo dai miei fidati compagni di lezione del poli, dal gruppo vacanze piemonte capitanato da Meg e tutti i miei ex compagni del liceo. In particolar modo vorrei ringraziare gli Amitici, un gruppo granitico che ne ha passate talmente tante insieme che penso non si scioglierà mai. Vorrei citarvi tutti per nome ragazzi ma non c'è proprio il tempo scusate.

Abstract

In recent years, multirotor UAVs (Unmanned Aerial Vehicles) came to the attention of the scientific world. Ingenuity experiment, the first drone to fly on another planet is their last success. UAVs are widely used in different sectors thanks to the small dimensions, the possibility to fly on hovering, the great precision of maneuvers and the minor costs. In fact, UAVs are employed for military and scientific applications. They are also used in rescue and healthcare operations, precision agriculture, shipping and delivery services and for hobby to take pictures or in FPV races.
The purpose of this thesis is designing, modeling and assembling a multirotor for indoor applications and designing and testing guidance algorithms on it. The first part consists of modeling, to design a small drone. The motors, the propellers, the battery and the ESCs are dimensioned to have a good endurance and a low weight. Then, 3D printing is used to produce the UAV frame. This technology allows rapid prototyping, low costs, reduced weight. The plates are printed in PLA (PolyLactic Acid) whereas the arms are made from aluminum. Finally, the whole chassis is tested through a static analysis to verify that it can endure the loads. Motors and propellers are the principal part of a drone because they produce the thrust that allows the drone to fly. It is therefore important to properly characterize them with thrust tests. Experimental thrust tests are carried out on the propellers to obtain a model of the actuators to be used in simulation.
In the second part of the thesis project, guidance and on-board algorithms are analyzed. A trajectory planner is implemented as guidance algorithm. Two improved versions of classic algorithms are considered and compared: an improved APF (Artificial Potential Field) and an RRT*FNDAPF (Rapidly exploring Random Tree Star Fixed Nodes with APF) that is a revisited RRT algorithm for unstructured environments. Both algorithms are implemented in the MATLAB Simulink model of the drone. Khatib's APF has a low computational effort and can avoid static or moving obstacles and new obstacles that were not predicted before the flight. However, it cannot map all the environment before the flight, and it could present the local minima problem and the GNRON (Goal Non-Reachable with Obstacles Nearby) problem. These problems can be solved by modifying the repulsive potential field of APF. The latter path planning algorithm is derived from a

classic RRT. The classic RRT (Rapidly exploring Random Tree) can map the environment before the flight, and it is simple to implement. However, it has a computational effort that depends on the dimensions of the environment, it cannot avoid dynamic obstacles, and it cannot find the optimal path. In addition, the convergence could be slow because the search of nodes is completely random. RRT*FNDAPF can solve these problems. For both algorithms, simulations are performed in MATLAB Simulink. The simulations are also made using Unity as plant to have more realistic results.

table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

An Unmanned Arial Vehicle (UAV), also called drone, is an aircraft that flies without a pilot. UAVs can be divided into two different categories, fixed wing UAV and multirotor UAV the second of which will be considered in this work. Multirotors obtain the thrust to sustain the weight and to fly from a certain number of propellers, each attached to an electric motor. The Electric motor spins the propeller that pushes down the air and produces the thrust. Multirotors can have from two to eight propellers. The advantages of having a larger number of propellers are a better stability and a bigger thrust that allows to carry a higher payload. On the down side the cost of production increases quickly.[**1**] Multirotors have several advantages:

- Hovering: they can fly on hovering, so they can stay fixed in the same point while fixed wing UAVs must move constantly to be able to fly.

- maneuverability: they have a greater maneuverability than fixed wing UAV, so they can reach zones that other drones cannot.

- Vertical take-off and landing: they do not need to move in an horizontal plane to take off or to land so they can take off and land in zones with limited dimensions where fixed wing UAV cannot.

- Compact structure: they do not have wings, so they take up less space than a fixed wing UAVs.

- Low cost: they have a simple structure with a small number of spinning parts and the largest part is rigid and fixed, so they are cheaper than fixed wing UAV.

- Payload: they can transport more payload than fixed wing UAVs.

However they have also some disadvantages:

- Endurance: they have a limited endurance because they spend more power than fixed wing UAVs.
- Maximum velocity attainable: they can reach a smaller velocity fixed wing UAV due to the fact that the horizontal component of force is little.
- Stability: They are less stable than fixed wing UAVs. In fact, they are more sensitive to wind so in determined outdoor applications it could be a problem.

In the last years, drones came to the attention of the scientific, military and civil world because of the decrease of components price, the increase of performance of on-board hardware and the enormity of sectors with possible applications. Drones are used for dangerous operations where the crew's life could be in danger. Usually, these operations are military applications like bomb detection or air strike but also scientific applications like exploration of unreachable zones. They can also be used for disaster management. They can map the zone immediately following the disaster to allow to focalize the manpower where it is necessary and reach zones that are not safe yet. Rescue and healthcare operations can also benefit from the use of drones. In fact, they could be used to locate lost people through thermal sensors. They also can take medical supplies and food to people in unreachable locations. A monitoring function can also be used in agriculture for crop analyses, for safety inspections of infrastructures and for weather forecasting. Another application could be shipping and delivery services, in which they could allow to reduce time and costs. But this is at an embryonal stage due to the legislation that regulates drones' flight. They are also used for hobby and for drone races. In particular, small drones are used for these races so it begins to be difficult to find components for small drones that do not have racing drones' requirements.
The focus of this thesis is an indoor application of a multirotor drone, therefore the drone must be able to fly in a GPS-denied environment. For this reason, a CamBoard pico flexx is mounted as sensor, the camera can rotate around its axis tanks to servomotors. Through the camera, the drone can identify the destination and the obstacles.

1.1 State of Art

This work is focused on the design and modeling of a small drone and on the implementation of the trajectory planning as guidance algorithm.
There are various types of UAVs, they can be classified based on the principle of generation of the lifting force into fixed wing UAVs, rotating and flapping wing UAVs, hybrid wing UAVs and gas envelope UAVs. Then, rotating and flapping wing UAVs can be divided into multirotor UAVs, ornithopter UAVs and cyclorotor UAVs [**2**]. In this thesis, the focus is on multirotor UAVs. Multirotor UAVs can be classified according to the number of rotors

[**3**]. Quadrotors are the most diffused drones because they balance stability, thrust and production cost so the subject of this work is a quadrotor. According to European Aviation Safety Agency (EASA) legislation drones are classified by their Maximum Take-Off Weight (MTOW) and these for leisure activities and research can be divided in three categories: A1, A2, A3. Small drones belong to category A1 because they are drones that can fly over people. The principal components of a drone are:

- Motors: The motors allow the propellers to spin. Electrical motors are used on drones because of the reduced overall dimension. Brushed motors are mounted on microdrones, whereas brushless motors are used for small drones and bigger.[**4**]

- Propellers: The propellers are mounted on motors. When the propellers spin they push down the air and produce the thrust that allow the drone to fly.[**5**]

- ESC: The ESC are electronic speed controller. They receive as input the throttle signals from the controller and regulate the rotation speed of the motors.[**6**]

- Battery: The battery is the power source of the drone. It gives the energy to all the components of the drone and with the motors determines the time of flight (TOF) of the drone. [**7**]

- Sensor: The typology of the sensor depends on the application for which the drone is designed. For example, drones for indoor application need sensors replacing the GPS.

- Microcontroller: The microcontroller elaborates the information received from the on-board computer and from the sensors and returns the throttle signals for the ESC.

- On-board computer: the on-board computer is the brain of the drone. It elaborates the guidance information used as reference by the microcontroller.

- Frame: The frame is the skeleton of the drone. It holds together all the other components.

Ultralight drones have a Maximum Take-Off Weight (MTOW) smaller than 250 g, an example of them is modeled in [**8**]. This drone is a quadrotor with a frame in carbon fiber, it has a weight of 249 gr and a time of flight of 10-15 minutes. An optimization of the precedent ultralight drone can be found in [**9**], the frame of this one is 3D printed. Usually the frame is composed by carbon fiber but for small drones 3D printing can be a useful method to build the frame. Small drones have to resist to relatively low loads so also 3D printed parts can be used. 3D printing the frame allows to reduce the weight, the construction cost and the construction time. A project based on nano Arduino where it is explained step by step how to build a quadcopter can be found in [**10**]. Also in this project

the frame is 3D printed.
A great variety of guidance algorithms has been designed and implemented for quadrotors. The guidance algorithm allows the drone to move from a starting point to a goal avoiding obstacles without the help of humans. Drones with guidance algorithms have several possible applications. In precision agriculture, for example, it can be used to distribute medicine to all the infected areas. That allows to reduce the working time and the amount of medicine. [**11**] Another possible application is area surveillance using a swarm of drones. Thanks to their small size, their easy operational procedure and the easiness with which they can access from one place to another they can cover also zones non reachable by humans. [**12**]. They can also be used in search and rescue missions. In this case, the algorithm must provide a fast area coverage to maximize the probability to find the target and a high connectivity because the target position must be communicable. [**10**]. An interesting application is the package delivery. Because of the large zone to cover the path must be the optimal path with a reliable communication and high packet delivery rate depending on its battery autonomy.[**13**]
The guidance system returns the profiles of reference position, velocity and acceleration used by the controller. Guidance algorithms are composed of a path planning algorithm and a trajectory generation algorithm. The objective of path planning is to find a path free of obstacle. Path planning can be divided into five categories [**14**]:

- Sampling based algorithms
- Node based algorithms
- Mathematic model based algorithms
- Bioinspired algorithms
- Multifusion based algorithms

Each category includes different families of algorithms. In [**15**] the authors design a modified Artificial Potential Field (APF), which is a sampling-based algorithm, to allow a quadrotor to escape from a local minimum and to reach the goal even if it is in the influence radius of an obstacle. The first problem is solved using a virtual potential that is placed in front of the obstacle. The second one introducing the distance between the UAV and the goal in the repulsive potential. In [**16**] a method to avoid local minimum in APF is implemented. In this case the UAV is not point-like so the virtual obstacle is placed into the UAV to increment the repulsive force. An APF for UAV also valid in dynamic environments can be found in [**17**]. In this case they modified the repulsive force by adding a term dependent from the obstacle speed. The term was modeled as a vector pointing from the UAV to the moving obstacle. A different example of dynamic APF with a moving

goal can be found in[**18**], in which a dependence with the goal speed was introduced in the attractive terms.
In [**19**] a BIT*(Batch Informed Trees) is designed and implemented, as a sampling-based planning algorithm this algorithm performs better than graph-based search algorithms in a high dimensional environment. Even if this is a sampling-based algorithm, it can find the optimal path without using a node-based algorithm as optimal search.
Another sampling-based algorithm is RRT (Rapidly exploring Random Tree), implemented in[**20**] for a formation landing of quadrotors, this algorithm has the disadvantage that it does not find the optimum path. In[**21**] An improved RRT path planning algorithm to accelerate the convergence rate and avoid random sampling characteristics is proposed. In [**22**] a modified RRT called RRT*FND (Rapidly-exploring Random Tree Fixed Nodes Dynamics) is implemented. It has three advantages with respect to classic RRT. The first one is that it can find the optimal path. The second one is that the nodes are fixed so the computational cost is smaller. The third is that if a dynamic obstacle invalidates a part of the tree, those nodes are removed but the nodes on the solution path not colliding with the obstacle are kept intact. Then it reconnects the tree to one of the solution path nodes disconnected.
Some node-based algorithms are Dijkstra and A* [**23**], these algorithms can find the optimal path. A* is a modified version of Dijkstra algorithm because it uses a heuristic approach so it can find the optimal path in a minor time. Another node-based algorithm is CD (cell decomposition) [**24**] which can be further subdivided in adaptive and exact. Adaptive cell decomposition has the advantage with respect to exact cell decomposition that it divides the cells only if they include an obstacle. A RR-MACD (Recursive Rewarding Modified Adaptive Cell Decomposition) is implemented in [**25**]. This kind of algorithm does not need to invoke any additional planner to search for an optimal path generation and the reduction in the number of nodes generated improves the computational effort.
In [**26**] a GA (Genetic Algorithm) is implemented as path planning. It is a bioinspired algorithm that can run tasks with high performance when searching for complex space solutions and is less computationally expensive than other bioinspired algorithms but it cannot be implemented on-line. In [**27**] a GSO (Glowworm Swarm Optimization) algorithm is implemented as path planning and shows promising results after comparison with deterministic algorithms and several meta-heuristic algorithms but it implemented only in static environment. In [**28**] an ABC (Artificial Bee Colony) algorithm is implemented which is relatively simple in use, fast in processing and is a population-based stochastic search approach in the field of swarm algorithms.
Multifusion based algorithms are composed of several algorithms in order to avoid disadvantages of single algorithms. For example, the path planning implemented in [**29**] is a P-PRM that combines a PRM (probabilistic road map) with an APF. It uses the PRM

to obtain information about the environment and the APF to find the final path, and a heuristic approach is used to avoid falling into a local minimum. In[**30**] an APF-RRT* algorithm is implemented, which differentiates itself from a traditional RRT by using APF as a search algorithm to accelerate the convergence and improves the planning efficiency.
In [**31**] a path planning for complex environments is implemented. A PRM is used to find the admissible space reducing the time that the planner requires to find a collision-free path so static obstacles are identified. Then an A* algorithm is used as optimal path search to find the minimum cost path that connects the initial and desired state in the probabilistic graph. As cost function an APF is used, that allow the algorithm to function also in dynamic environment.
The path has only information about the position and the waypoints to reach so a trajectory generation algorithm is needed to obtain the others kinematics constraints, two polynomial methods are presented.
In [**32**] a minimum snap trajectory is implemented which produces a segmental seventh order polynomial trajectory. Trajectories generated with this algorithm are optimal in the way that they minimize cost functionals that are derived from the square of the norm of the snap (the fourth derivative of position). These cost functionals are meaningful since the input variables are algebraically related to the snap.
In [**33**] a minimum-acceleration trajectory is implemented which produces a segmental third order polynomial trajectory and achieves the balance between speed, safety and computational economy.

In this thesis, a quadrotor is designed and modelled, firstly by choosing the components and making a 3D CAD model of the quadrotor. To accomplish simulations, mass and inertia moments are taken from a CAD model in the first approximation. Then a prototype of the quadrotor is built, after which the measured data are adjusted in the simulations. The chassis is made in part in PLA and in part in aluminum: arms are in aluminum and the rest of the chassis is 3D printed to reduce cost and weight. Since 3D printed parts have an infill of twenty percent, to have mass and inertia moments on the 3D CAD, a virtual density is used on the model. Benchmark tests were made to decide which propellers and which battery to use.
Two guidance algorithms are designed and implemented in Matlab/Simulink: a modified APF and an RRT*FND. These algorithms are designed in a way that a code generation can be made without modifying the algorithm. That is because the guidance algorithm will be then deployed on the autopilot. These guidance algorithms are simulated in a simulation environment modelled on Matlab Simulink [**34**], firstly using a PID as controller for attitude and altitude control and a PDD (a PD with a derivative action on the derivate

channel) for the position control. Then simulations are made with a first order SMC (sliding mode control) as controller and compared with the previous. To perform more realistic simulations a second set of simulations is made using a plant implemented on Unity [**35**]. The signal passes from Simulink to ROS and from ROS to Unity and vice versa [**36**]. Finally experimental tests of these guidance algorithms are made using a CUAV V5 nano as autopilot hardware.

1.2 Outline

This work is organized as follows:

- Chapter 2 defines the drone modeling. Firstly, the mathematical model is recalled, then the chapter describes how to design a small drone and the final configuration with the chosen components. Then it illustrates the construction process. Two versions are proposed: the first is the one originally planned whereas the second is the one modified to experimentally test guidance algorithms. This second version has instead of the board STM Nucleo L432KC the CUAV V5 nano. Lastly tests and analyses made on the components are explained. Benchmark tests to evaluate the thrust curve with different propellers and batteries are made to properly choose these components. Basic structural analyses on 3d printed parts are made to ensure that these parts resist to principal loads.

- Chapter 3 treats the theory and the design of two guidance algorithms. Firstly, an Artificial Potential Field is implemented, which was modified to solve two typical problems of APF: Goal Non-Reachable with Obstacles Nearby (GNRON) problem and local minimum problem. Then a guidance algorithm with an RRT*FNDAPF as path planning is implemented. It is a modified version of classic RRT that also works in dynamic environment, using a fixed number of nodes to limit the computational effort and has an APF as heuristic search algorithm to reduce computational time and to converge faster at the solution. Therefore, this algorithm can find the optimal path.

- Chapter 4 illustrates the two simulation environments, the first implemented all in Matlab/Simulink and the second with the plant implemented in Unity, underling the differences. Principal numerical parameters for controller block are reported for both environments.Then results of simulations are exposed and compared.

- Chapter 5 summarizes the principal points and conclusions of the thesis and presents possible future works on this project.

Chapter 2

Drone Modeling

In this section, the drone model and operative decisions are described. Firstly, basic and mathematical notions that determine the quadrotors model in StateSpace representation are presented. Then, the typical components of a quadrotor are described, the choice of components is presented and the construction process is illustrated. Finally, tests and analysis about drone crucial components are exposed.

2.1 Mathematical Model

In this part, the flight mechanism of quadrotors and some notions about the reference frame used are described. Then the derivation of the mathematical model is presented in the State-Space representation. [**34**]

2.1.1 Refence Frame

In this work two different reference frames are used, NED and SEZ:

- The North-East-Down (NED) frame reference is adopted in the simulation with the Matlab/Simulink model. It is a right-handed reference frame and the "Down" arrow points toward the centre of the Earth.

- The South-East-Z (SEZ) frame reference is adopted in the simulation with Unity. It is a right-handed reference frame and the "Z" arrow points away from the centre of the Earth. This reference frame is used because unity adopts a left-handed reference frame "XYZ" with the "Y" arrow that points away from the centre of the Earth and for the conversation the SEZ was much convenient.

To describe the motion of the quadrotor besides of the inertial reference fame another reference frame is needed. The body reference frame that is centered in the quadrotor

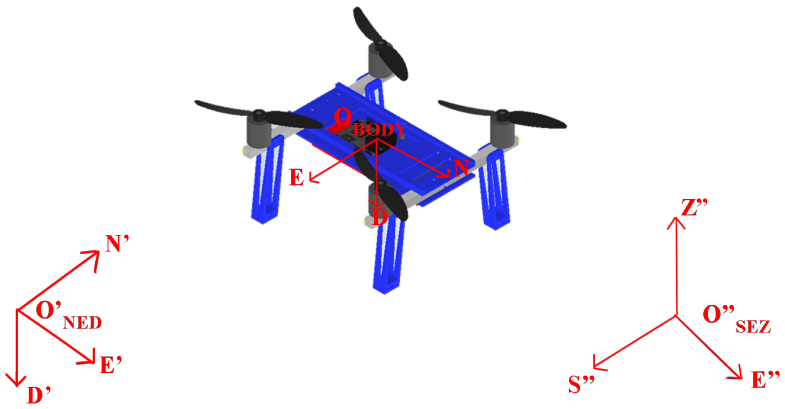


Figure 2.1: Reference frames used

Centre of Gravity (CoG) of the drone. (figure 2.1)

2.1.2 Flight mechanisms

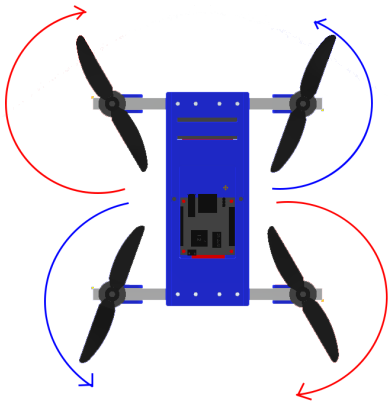
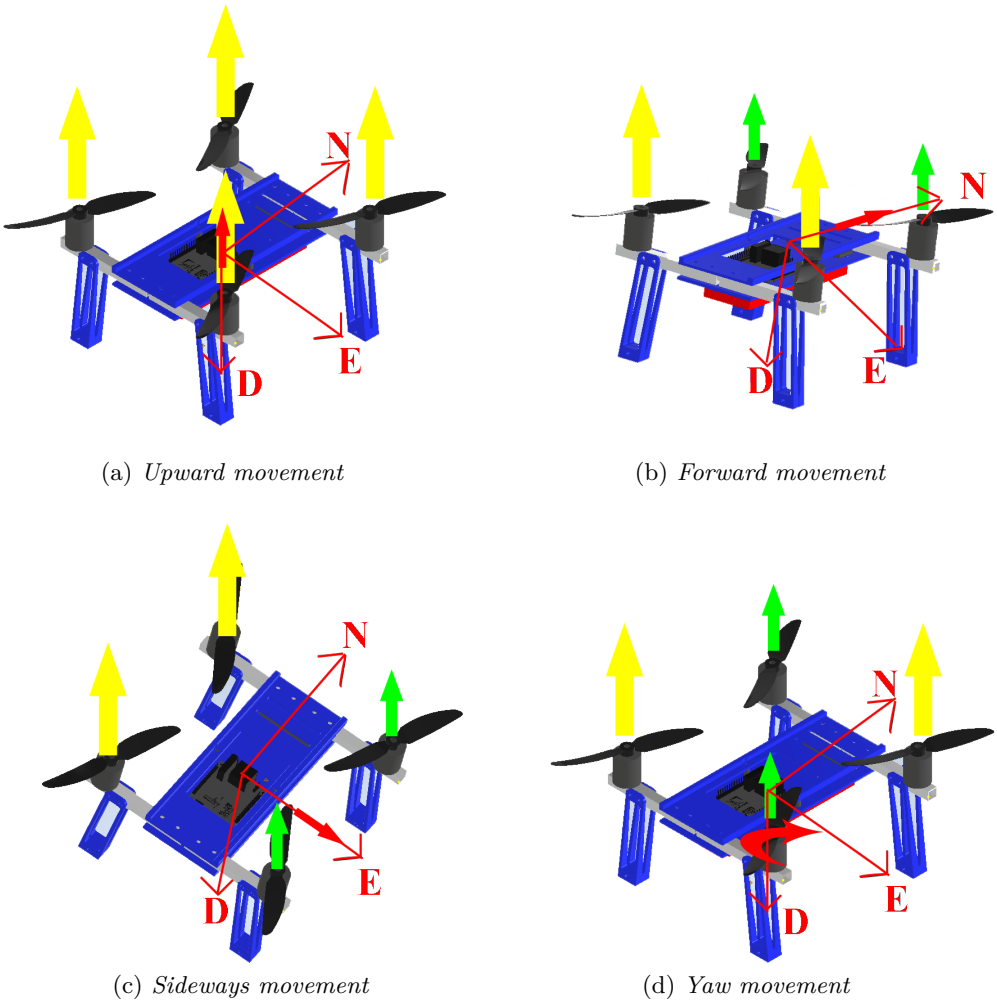


Figure 2.2: Directions of rotation of the propellers

The name quadrotor derives from the fact that these UAVs have four rotors that generate the thrust. If the total thrust is higher than the sum of the weight and the drag force the drone can fly. That allows the quadrotor to be a VTOL (Vertical take-off landing) UAV. The fact that they have four symmetrical rotors allows them to have an improved flight safety and manoeuvrability. [articolo nuovo che non riesco a scaricare] Two configurations of rotors can be used on quadrotor, X (cross) configuration or + (plus) configuration. The

Cross configuration maximizes the moments generated by the motor thrust, and as will be described in the following this configuration is the one used in this thesis. Two propellers spin clockwise while the others spin counter clockwise (figure 2.2 on the previous page). The reason is that the torque produced by each propeller affects the drone so it will rotate around its Vertical (z) axis if all the rotors spin in the same direction. Using two propellers that spin clockwise and the other two that spin counter clockwise the torques produced cancel each other out.

UAV multicopters can fly in hover and take off and lift off in vertical when all the



(a) *Upward movement*

(b) *Forward movement*

(c) *Sideways movement*

(d) *Yaw movement*

Figure 2.3: The basic maneuvers of a quadrotor

propellers spin at the same speed (figure 2.3a) but they have the disadvantage that they are under-actuated. The Translational motion is coupled with the rotational motion so to move along the x and the y axes they need to rotate around its body axes:

- To move forward and backward they have to rotate around the East axis (figure 2.3b on the previous page)

- To move sideways they have to rotate around the North axis (figure 2.3c on the preceding page)

- To move around the vertical axis and to vary the yaw angle the propellers that rotate in the same direction have to vary their speed simultaneously (figure 2.3d on the previous page)

2.1.3 Reference frame transformation

Two methods to pass between the inertial reference frame and the body reference frame and to describe the orientation of the UAV in the three-dimensional space are used in this work, the Euler angles and the quaternions:

- Euler angles describe a three-dimensional rotation with a sequence of elementary rotations around the axes. These rotations can be described with the following matrices:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.1}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.2}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

  The matrix that represents the total rotation around the axes is obtained by Multiplying them. The order of rotations can modify the final matrix. In this case the order ZYX is used so the final rotation matrix is:

$$\mathbf{R}_{\phi,\theta,\psi}(1,:) = \mathbf{R}_{body}^{NED}(1,:) = \begin{bmatrix} \cos(\theta) * \cos(\psi) \\ \cos(\theta) * \sin(\psi) \\ -\sin(\theta) \end{bmatrix} \tag{2.4}$$

$$\mathbf{R}_{\phi,\theta,\psi}(2,:) = \mathbf{R}_{body}^{NED}(2,:) = \begin{bmatrix} \sin(\phi) * \sin(\theta) * \cos(\psi) - \cos(\phi) * \sin(\psi) \\ \sin(\phi) * \sin(\theta) * \sin(\psi) + \cos(\phi) * \cos(\psi) \\ \sin(\phi) * \cos(\theta) \end{bmatrix} \tag{2.5}$$

$$\mathbf{R}_{\phi,\theta,\psi}(3,:) = \mathbf{R}_{body}^{NED}(3,:) = \begin{bmatrix} \cos(\phi) * \sin(\theta) * \cos(\psi) + \sin(\phi) * \sin(\psi) \\ \cos(\phi) * \sin(\theta) * \sin(\psi) + \cos(\phi) * \cos(\psi) \\ \cos(\phi) * \cos(\theta) \end{bmatrix} \tag{2.6}$$

These angles can also be used to characterize the quadrotor attitude with respect to the inertial frame, so it is necessary to find a relationship between these angles and the angular velocity p, q and r in the body reference frame:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi) * \tan(\theta) & \cos(\phi) * \tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{bmatrix} * \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.7}$$

- Quaternions are more practical to describe three-dimensional rotations with respect to Euler angles. In fact, Euler angles can suffer the problem of gimbal lock. For the ZYX rotation, when θ is ninety degrees the system loses one degree of freedom and ϕ and ψ can not be found uniquely. Moreover, they need a low computational power to be handled. The negative aspect of quaternion with respect to Euler angles is that they are less intuitive. Quaternions can be represented as follow:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.8}$$

Some programs prefer the representation with q_0 as the forth element of the vector. To define the error quaternion, that is the distance between the reference quaternion and the estimated quaternion, the Hamilton product is needed:

$$\mathbf{q} \otimes \mathbf{p} = q_0 p_0 - \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} * \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + q_0 \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + p_0 \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} + \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{2.9}$$

With this definition the error quaternion can be defined as:

$$\tilde{\mathbf{q}} = \hat{\mathbf{q}}^{-1} \otimes \mathbf{q}_{ref} \tag{2.10}$$

where $\hat{\mathbf{q}}$ is the current estimated quaternion and $\mathbf{q_{ref}}$ is the quaternion reference. Finally, the evolution of the quaternions can be described as the follow:

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega\mathbf{q} = \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.11}$$

2.1.4 Forces and Moments

In this part the mathematical model of the quadrotor is derived. Some assumptions are made to simplify the mathematical model and make it as a second order ordinary differential equation (ODE):

- The quadrotor and all its components are considered as a rigid body
- The CoG of the quadrotor and the origin of the body reference frame are coincident
- The actuators are not modelled
- The aerodynamic forces are not considered, in fact they are negligible in indoor applications
- The Earth is considered as flat because of the small dimension of the environment with respect to the earth radius and its rotation is negligible with respect to drone body angular speeds

Everything starts from the Newton's second law:

$$\mathbf{F}_B + \mathbf{R}_{NED}^B * mg = m\left[\frac{d\mathbf{v}_B}{dt} + \boldsymbol{\omega}_B \times \mathbf{v}_B\right] \tag{2.12}$$

where $\mathbf{v}_B$, that is the relative velocity of the multirotor CoM, and $\boldsymbol{\omega}_B\times$ are defined as follows:

$$\mathbf{v}_B = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \boldsymbol{\omega}_B\times = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

Making the differentiation the forces equation becomes:

$$\frac{\mathbf{F}_B}{m} + \mathbf{R}_{NED}^B * g = \left[\dot{\mathbf{v}}_B + \boldsymbol{\omega}_B \times \mathbf{v}_B\right] \tag{2.13}$$

Regarding the moments, one starts from the Newton's second law applied to rotating bodies:

$$\boldsymbol{\tau} = (\frac{d\mathbf{H}}{dt} + \boldsymbol{\omega}_B \times \mathbf{H}) \tag{2.14}$$

where:

$$\mathbf{H} = \mathbf{J}\boldsymbol{\omega}_B = \begin{bmatrix} J_x & J_{xy} & J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & -J_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.15}$$

So, isolating $\dot{\boldsymbol{\omega}}_B$ the equation becomes:

$$\dot{\boldsymbol{\omega}}_B = -\mathbf{J}^{-1}(\boldsymbol{\omega}_B \times (\mathbf{J}^{-1}\boldsymbol{\omega}_B)) + \mathbf{J}^{-1}\boldsymbol{\tau} \tag{2.16}$$

2.1.5 State-Space Model

The mathematical model of the quadrotor can be written as a nonlinear differential equation using the equation of the precedent section. The form of the equation is:

$$\dot{\mathbf{x}} = \mathbf{f}_{(x)} + \mathbf{g} \cdot \mathbf{u} \tag{2.17}$$

This form is convenient when it is used with controllers, in particular with controllers that require the knowledge of the model to be controlled. Sliding Mode Controller is one of them, so since SMC is used in this work, this form is adopted here.
The vector state x and the input u are defined as follows:

$$\mathbf{x} = \begin{bmatrix} p_N \\ p_E \\ h \\ \phi \\ \theta \\ \psi \\ u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.18}$$

A conversion from the linear velocity in the body reference to the inertial reference frame is needed to describe the first three states. using (2.4), (2.5), (2.6) to rotate $\mathbf{v}_B$ as follow:

$$\begin{bmatrix} \dot{p}_N \\ \dot{p}_E \\ \dot{h} \end{bmatrix} = \mathbf{R}_{body}^{NED} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.19}$$

For the next three states, that are the attitude variation in the inertial reference frame, (2.7) is used, so the result is:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} p + \tan(\theta)(q\sin(\phi) + r\cos(\phi) \\ q\cos(\phi) - r\sin(\phi) \\ \frac{1}{\cos(\theta)}(q\sin(\phi) + r\cos(\phi) \end{bmatrix} \tag{2.20}$$

Then for the linear acceleration in the body reference frame (2.13) is used. By isolating $\dot{\mathbf{v}}_B$, solving the vector product and multiplying the gravity vector with the rotation matrix the result is:

$$\dot{\mathbf{v}}_B = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw - g\sin(\theta) + 0 \\ -ru + pw + g\sin(\phi)\cos(\theta) + 0 \\ qu - pv + g\cos(\phi)\cos(\theta) + \frac{F_D}{m} \end{bmatrix} \tag{2.21}$$

For the three last states (4.62) is used, and introducing some constants c_i that describe the inertia properties the result is:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} q(c_1 r + c2p) + c_3\tau_x + c_4\tau z \\ prc_5 - (p^2 - r^2)c_6 + c_7\tau_y \\ q(c_8 p - c_2 r) + c_4\tau_x + c_9\tau_z \end{bmatrix} \tag{2.22}$$

where:

$$\begin{aligned} c_1 &= \frac{(J_y - J_z)J_z - J_{xz}^2}{J_x J_z - J_{xz}^2} & c_2 &= \frac{(J_x - J_y + J_z)J_{xz}}{J_x J_z - J_{xz}^2} & c_3 &= \frac{J_z}{J_x J_z - J_{xz}^2} \\ c_4 &= \frac{J_{xz}}{J_x J_z - J_{xz}^2} & c_5 &= \frac{J_z - J_x}{J_y} & c_6 &= \frac{J_{xz}}{J_y} \\ c_7 &= \frac{1}{J_y} & c_8 &= \frac{(J_x - J_y)J_x + J_{xz}^2}{J_x J_z - J_{xz}^2} & c_9 &= \frac{J_x}{J_x J_z - J_{xz}^2} \end{aligned}$$

Assembling (2.19), (2.20), (2.21) and (2.22) and using (2.17) as reference the dynamical model in state-space form is:

$$\mathbf{f}_x = \begin{bmatrix} \mathbf{R}_{body}^{NED} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\ p + \tan(\theta)(q\sin(\phi) + r\cos(\phi)) \\ q\cos(\phi) - r\sin(\phi) \\ \frac{1}{\cos(\theta)}(q\sin(\phi) + r\cos(\phi) \\ rv - qw - g\sin(\theta) \\ -ru + pw + g\sin(\phi)\cos(\theta) \\ qu - pv + g\cos(\phi)\cos(\theta) \\ q(c_1 r + c2p) \\ prc_5 - (p^2 - r^2)c_6 \\ q(c_8 p - c_2 r) \end{bmatrix} \qquad \mathbf{g} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_3 & 0 & c_4 \\ 0 & 0 & 0 & 0 & c_7 & 0 \\ 0 & 0 & 0 & c_4 & 0 & c_9 \end{bmatrix} \tag{2.23}$$

2.2 Design of a small drone

In this part, after a brief description of the UAV classification, the quadrotor anatomy is presented describing all the components of a quadrotor. After that, the motivations for the choices of each component are given.

2.2.1 UAV classification

The European Aviation Safety Agency (EASA) coordinates the civil aviation security management. A new release of the European regulation on the use of UAVs determines that from December 31, 2020, new European drone rules are in force [**37**].

Drones for leisure activities and low risk commercial activities are included in the Open

UAS		Operation		Drone Operator/pilot		
Class	MTOM	Subcategory	Operational restrictions	Drone Operator registration	Remote pilot competence	Remote pilot minimum age
Privately built	< 250 g	A1 (can also fly in subcategory A3)	- may fly over uninvolved people (should be avoided when possible) - no flying over assemblies of people	No, unless camera / sensor on board **and** a drone is not a toy	- no training needed	No minimum age
C0					- read user manual	16*, no minimum age if drone is a toy
C1	< 900 g		- No flying expected over uninvolved people (if it happens, should be minimised) - no flying over assemblies of people	Yes	- read user manual - complete online training - pass online theoretical exam	16*
C2	< 4 kg	A2 (can also fly in subcategory A3)	- no flying over uninvolved people - keep horizontal distance of 30 m from uninvolved people (this can be reduced to 5 m if low speed function is activated)	Yes	- read user manual - complete online training - pass online theoretical exam - conduct and declare a self-practical training - pass a written exam at the NAA (or at recognized entity)	16*
C3	< 25 kg	A3	- do not fly near people - fly outside of urban areas (150 m distance)	Yes	- read user manual - complete online training - pass online theoretical exam	16*
C4						
Privately built						

Figure 2.4: Open Category after january 1,2023

category. The Open category (figure 2.4) is divided in three subcategories:

- A1: drones that can fly over people but not over assemblies of people
- A2: drones that can fly only close to people
- A3: drones that have to fly far from people

These subcategories are divided in classes depending on drone weigh:

- C0: it is a subcategory of A1, includes drone with a MTOW smaller than 250 grams. They do not require a registration or a possession of an electronic on-board identification system
- C1: it is a subcategory of A1, includes drone with a MTOW smaller than 900 grams. They require a registration or a possession of an electronic on-board identification system

- C2: it is a subcategory of A2, includes drone with a MTOW smaller than 4 kilograms. They require a registration or a possession of an electronic on-board identification system

- C3 and C4: they are a subcategory of A3, includes drone with a MTOW smaller than 25 kilograms. They require a registration or a possession of an electronic on-board identification system

The objective of this thesis is to design and model a small drone namely a drone of the category A1, better if it belongs to the subcategory C0.

2.2.2 Motors

Motors are the principal component of a drone because the combination of them and the propellers allows to generate the thrust. Therefore, the choice of motors and propellers is critical. To allow the drone to fly stably and with a good response to command the throttle should be around 50% in hovering so the total thrust should be approximately the double of the total drone weight. There are two types of motors used on quadcopters, brushed and brushless motors:

- Brushed motor: It has a rotating stator and a stationary shell. The stator act as an electromagnet with two poles. This motor,with respect to the brushed one, is cheaper, simpler, does not need ESC and sometimes can be mounted without using any screws and can be smaller.

- Brushless motor: it uses a permanent magnet and switches the polarity electronically, so it does not use brushes. The advantages are that it is more efficient, has a longer durability, a better heat dissipation and a better speed and torque.

Usually brushless motors are used for micro and nano drones but for bigger drones brushless motors are preferred. In this work brushed motors are adopted. Brushless motors are identified using a 4-digit number: the first two are the stator diameter, the last two are the stator height. Motors with a taller stator have more power at high RPM instead of one with a wider stator that has more torque at lower RPM. So taller stators are useful if the objective is to have a fast drone, but for more efficient drones wider stators are a better choice.
The KV factor indicates how much the rotations per minute (RPM) increase with the increase of voltage. Higher KV allows the propeller to spin faster and it draws more current. For this reason, usually a good pairing is between larger props and low KV motors and vice versa. If a high KV motor is coupled with a propeller too large the motor risks to generate too much heat and burn the motor.

Since the speed drone was not a principal requirement, the efficiency was privileged in the choice of the motors. This choice complicated the research, in fact with the increase of popularity of FPV drones races, recent motors for small drones have a low efficiency in favour of high RPM.
The MN1806 kv2300 belonging to navigation series of T-motor was identified as ideal but

Figure 2.5: Tarrot MT1806 kv2300

due to the difficulty to find it in commerce, the Tarrot MT1806 kv2300 (figure 2.5) was chosen instead. A Tarrot MT1806 kv2300 has a weight of 18 grams.

2.2.3 Propellers

The propellers are strictly related to the motors, as they allow to generate the thrust thanks to the rotation of the motor stator.
A propeller is characterized by three elements, the length, the pitch and the number of blades:

- The length is the diameter of the disc the propeller makes when it rotates
- The pitch is the travel distance of one single propeller rotation
- The number of blades can be between two and six

The bigger the length and the pitch are, the more thrust the propeller generates but the more current it draws so a trade-off is necessary. Increasing the number of blades can add more surface that will generate more thrust, but it will also draw more current and

generate more drag.
Like the motors, the propellers are also identified with an analogue format: the first two numbers represent the length and the last two represent the pitch. Sometimes the name also includes the number of blades at the end.
Propellers and motors can be chosen from motor datasheets, but often these datasheets have performances for a few combinations so it is useful to buy more set of propellers and to test them with the chosen motors.

In this thesis 5035x2 (figure 2.6a) and 5045x3 (figure 2.6b) were tested and 5035x2 is

(a) *Propellers T-Motor 5035x2* (b) *Propellers Tarrot 5045x3*

Figure 2.6: Propellers

finally chosen. A 5035 propeller has a weight of 2.2 grams.

2.2.4 ESC

An ESC is an Electronic Speed Controller, which receives as input the throttle signals from the controller and regulates the rotation speed of the motor. The fundamental characteristic of ESC is the Current Rating. There are two types of current rating:

- Continuous current rating: it represents the maximum amount of continuous current that the ESC can safely handle
- Burst current rating: it represents a higher current which the ESC can withstand for short periods of time

If the ESC is badly chosen it can burn and seriously damage the quadrotor, so the ESC must be chosen controlling from motor datasheet the maximum current absorbed by the couple propeller-motor.
There are also ESC 4-in-1, which are 4 ESC integrated in the same board. That allows to reduce the total weight of the ESC but in case of damage all the ESCs are lost. Usually, it is installed under the flight controller but it may produce interference that affects the

flight performance.

For these reasons in this thesis four HobbyWing ESC 10A (figure 2.7 are chosen. A

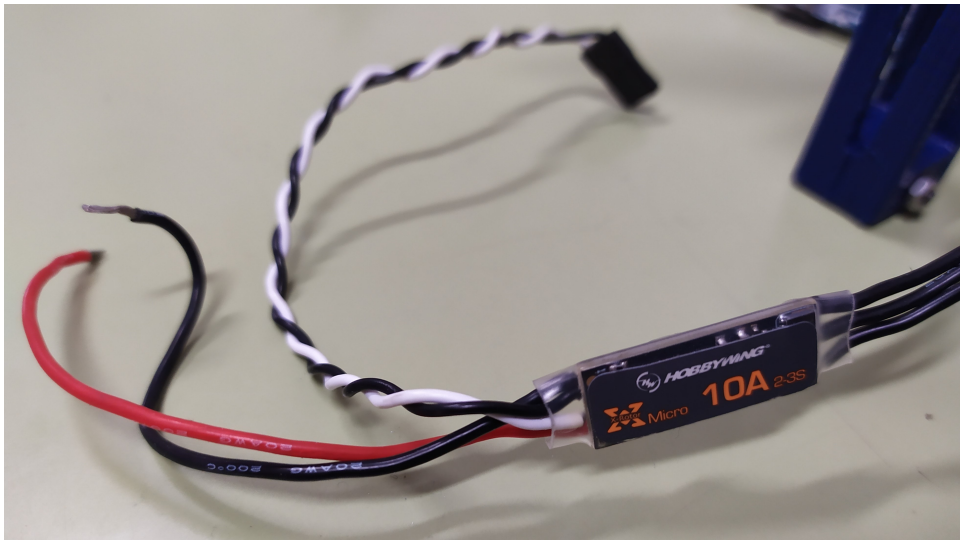


Figure 2.7: HobbyWing ESC 10A

HobbyWing ESC 10A has a weight of 6.5 grams.

2.2.5 Battery

The battery is the energy source of the quadrotor. Two types of batteries are taken into consideration, Li-Ion battery and Li-Po. The parameters that characterize the battery are:

- capacity: it is a measure of the charge stored by the battery and is determined by how much active material is in the battery. It is measured in mAh that indicates how much current can be drained from the battery in an hour until it is empty.
- Discharge rate (C Rating): it allows to calculate the maximum discharge current of the battery with the following equation:

$$maximum\ Discharge\ Current = C\ Rating * capacity \tag{2.24}$$

- Number of cells: It determines the voltage of the battery (each cell has $3.7V$).

Some batteries have two C rating indicated: the continuous and the burst. The differences are the same as for the ESC.

The maximum discharge current is fundamental to choose the type of battery. In fact, Li-ion batteries have a lower C rating with respect to Li-Po batteries. That implicates that the maximum discharge rate of this type of batteries should be lower than the current

needed by the motors. So, the battery chosen is a Li-Po battery.
The more cells the battery has the more power it can deliver to motors but the heavier the battery is, so it is better to use the minimum number of cells possible.
The capacity allows to calculate the Time of Flight (TOF) with the following equation:

$$TOF = \frac{Battery\ capacity}{Current\ absorbed\ by\ motors} * 0.8 \tag{2.25}$$

But the more capacity the battery has the heavier it is, so to choose the capacity of the

Figure 2.8: Fullpower Li-po 3S 2200 mAh 35C

battery a trade-off is needed. Thanks to thrust tests that will be illustrated in the following section, a 3S Li-Po battery with a capacity of 2200 mAh (figure 2.8) is chosen. This battery allows to have a TOF of 15 minutes and has a weight of 177 grams.

2.2.6 Sensor

This quadrotor is designed to be used in indoor applications, which means it operates in GPS-denied environments. To accomplish it, this drone uses as sensor a CamBoard pico flexx (figure 2.9a on the following page). The camboard pico flexx is a small 3D camera based on pmd TOF technology. The small form and the low power consumption makes it valid for 3D depth sensing applications. These are the characteristic of this camera [**38**]:

- Dimensions: 68mm x 17mm x 7.35 mm
- Weight 8g
- ToF-Sensor: IRS1145C Infineon REAL3 3D image Sensor IC based on pmd intelligence
- Measurement range: 0.1-4 m
- Framerate: Up to 45fps (3D frames); 8 pre-defined operation modes
- Acquisition time per frame: 4.8 ms typ. at 45 fps / 30 ms typ. at 5 fps

- Power consumption: USB2.0 compliant, average 300 mW for IRS chip and illumination
- Software: Royale SDK (C/C++ base, support Matlab, DotNet, CAPI, OpenCV, OpenNI2 and ROS)
- Resolution: 224 x 171 (38K) px
- Viewing angle (H x V): 62° x 45°
- Interface data: USB2.0/USB3.0
- Depth resolution: <= 1% (0.5-4m at 5fps) and <=2% (0.1-1m at 45fps)

(a) *Pico Flexx Camboard* (b) *Sg90 Servomotor*

Figure 2.9: Pico Flexx Camboard & Sg90 Servomotor

This camboard will be mounted using a support 3d printed that will be mounted on a Sg90 (figure 2.9b), a servomotor that allows the camera to rotate. A camboard pico flexx has a weight of 8 grams and a Sg90 has a weight of 9 grams.

2.2.7 Microcontroller

The microcontroller elaborates information received as input from the On-board computer and from the sensors to obtain as output PWM signals for motors. In this thesis the microcontroller used is a STM32 nucleo L432KC in a 32-pin package (figure 2.10 on the following page). The principal features are:[**39**]

- 1 user LED
- 1 reset push-button
- Board connectors: Arduino Nano V3 expansion connector and a Micro_AB USB connector for the ST-LINK
- Power supply: ST-LINK, USB V_bus or external sources

- On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port and debug port
- Free software libraries
- Support of a wide choice of integrated Development Environments (IDEs) including IAR, Keil and GCC based IDEs
- 24MHz crystal oscillator
- Arm Mbed Enabled compliant

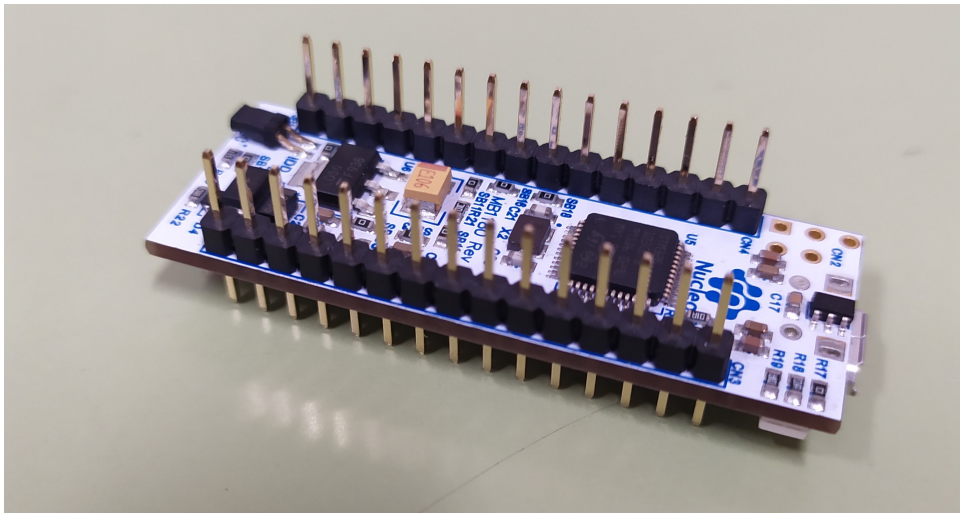


Figure 2.10: Microcontroller STM32 nucleo L432KC

This ST microcontroller has a weight of 7 grams.

2.2.8 On-board Computer

The on-board computer elaborates the guidance information for the microcontroller. Two On-board computers were used in this thesis, in a first moment the Orange Pi Zero LTS (figure 2.11a on page 25), then afterwards an Orange Pi Zero2 (figure 2.11b on page 25) was adopted because it was more performing. The hardware specifications of the orange pi Zero 2 are [**40**]:

- CPU: Allwinner H616 64-bit high-performance Quad-core Cortex-A53 processor
- GPU: Mali G31 MP2, Supports OpenGL ES 1.0/2.0/3.2 and OpenCL 2.0

- Video decoding: H.265 Main10 at L5.1 decoder up to 4K at 60fps or 6K at 30fps, VP9 Profile 2 decoder up to 4K at 60fps and AVS2 JiZhun 10bit decoder up to 4K at 60fps, H.264 BP/MP/HP at L4.2 decoder up to 4K at 30fps

- Video encoding: H.264 BP/MP/HP encoder up to 4K at 25fps or 1080p at 60fps and JPEG snapshot performance of 1080p at 60fps

- Memory(SDRAM): 512MB/1GB DDR3 (Shared with GPU)

- Onboard Storage: TF card slot and 2MB SPI Flash

- Onboard Network: Support 1000M/100M/10M Ethernet

- Onboard WIFI+BT: AW859A Chip, Support IEEE 802.11 a/b/g/n/ac and Support BT5.0

- Video Outputs: Micro HDMI 2.0a up to 4K at 60fps and TV CVBS output, Support PAL/NTSC (Via 13pin interface board)

- Audio output: Micro HDMI and 3.5mm audio port (Via 13pin interface board)

- Power Source: Type-C interface 5V2A input

- USB 2.0 Ports: 3*USB 2.0 HOST(Two of them are via 13pin interface board)

- Low-level peripherals: 26pin header with I2C, SPI, UART and multiple GPIO ports and 13pin header with 2*USB Host, IR pin, Tv-out, AUDIO (no MIC) and 3 GPIO ports

- Debug serial port: UART-TX, UART-RX and GND

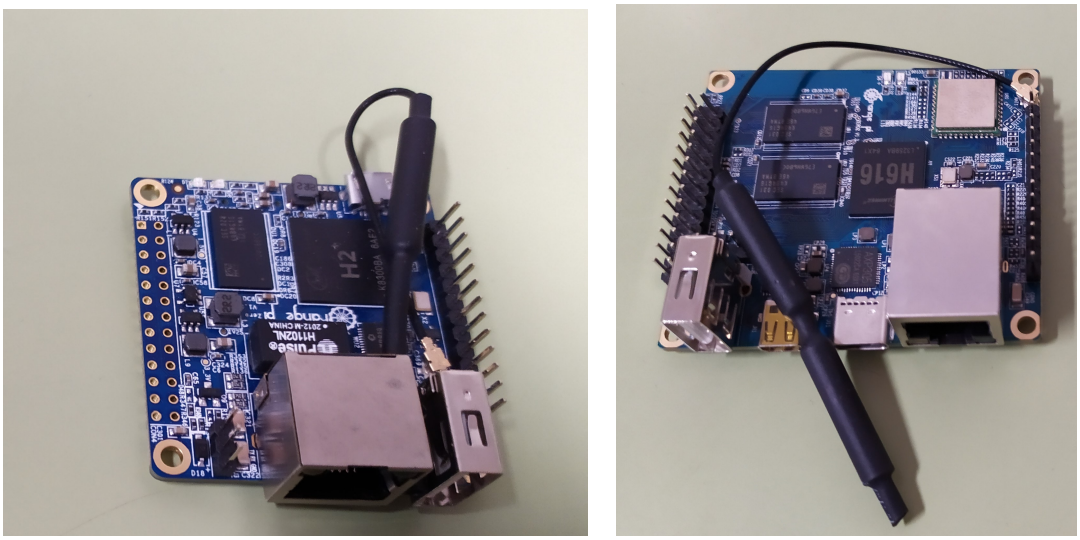
- LED: Power led and Status led

- IR receiver: Support IR remote control (via 13pin interface board)

- Supported OS: Android10, Ubuntu, Debian

- Dimensions: 53mm×60mm

The Orange Pi Zero LTS has a weight of 26 grams and the orange pi Zero2 has a weight of 30 grams.



(a) *Orange Pi Zero LTS* (b) *Orange Pi Zero2*

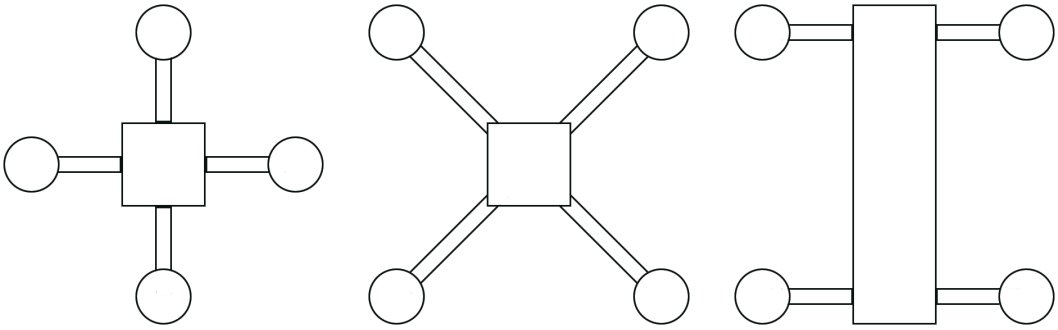
Figure 2.11: On board Computers

2.2.9 Frame

The frame is the skeleton of the drone, it supports all the components and must guarantee a correct weight balance and a good rigidity [tesi chiaberge]. There are different layouts of frames depending on how the arms are positioned with respect to the central core:

- + configuration (figure 2.12a on the next page)
- X configuration (figure 2.12b on the following page)
- H configuration (figure 2.12c on the next page)

X and H with respect to + allow to maximize the moments generated by the motor thrust, in this thesis the configuration adopted was the H.
The frame can be made in different materials depending on how intense the solicitations, that it receives, are. The most common material used is carbon fiber because it is a lightweight rigid and resistant material, but it is also expensive and complicated to process. The frame can be bought but these frames have standard dimensions and form so in this thesis the frame has been designed from the start and then 3D printed in PLA.
The arms are the most stressed part of the frame because they are directly connected with the motors which are the origins of the biggest strain. Because of that, arms are not 3d printed but are made with tubular bars in aluminum with a thickness of 1 mm.
The design of the frame is strictly connected with the choice of the previous components, so before the frame design, a preliminary design of the complete configuration is needed.

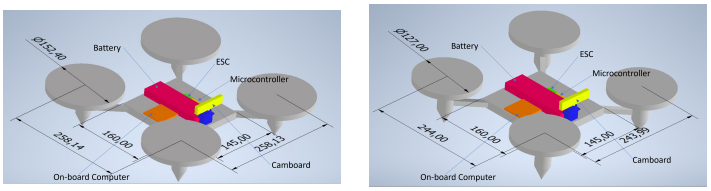


(a) *+ configuration* (b) *X configuration* (c) *H configuration*

Figure 2.12: Frame configurations

Principal requirements are a symmetrical position of rotors and a symmetrical weight distribution with respect to the CoG.

Five preliminary configurations were hypothesized (figure 2.13). Because the design pro-



(a) *Configuration 1* (b) *Configuration 2*

(c) *Configuration 3* (d) *Configuration 4* (e) *Configuration 5*

Figure 2.13: Possible drone configurations

cess is an iterative process, when these configurations were thought the chosen components were not the ones used in the final configuration but they are indicative of the overall dimensions. The differences are:

- All the batteries supposed are Li-Po 2S because of the reduction of weight with respect to Li-Po 3S

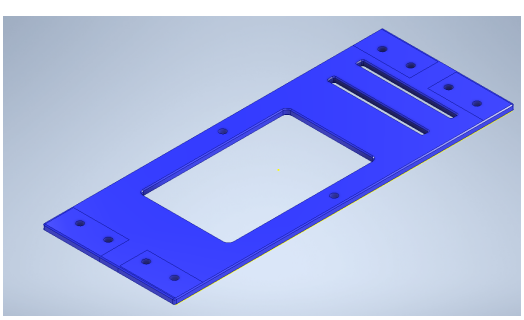
- The ESC supposed is a 4-in-1 ESC
- The propellers supposed for configuration 1 has a length of 6 inches
- Configuration 4 and 5 have two batteries with a minor capacity

Configuration	Advantages	Disadvantages
Configuration 1 (2.13a)	throttle in hover <50% symmetrical weight distribution symmetrical position of rotors	higher MTOW high maximum measure low endurance
Configuration 2 (2.13b)	symmetrical weight distribution symmetrical position of rotors high endurance	throttle in hover >50% high MTOW high maximum measure
Configuration 3 (2.13c)	symmetrical weight distribution symmetrical position of rotors high endurance low maximum measure low MTOW	throttle in hover >50%
Configuration 4 (2.13d)	symmetrical position of rotors higher endurance low maximum measure	throttle in hover >50% asymmetrical weight distribution high MTOW
Configuration 5 (2.13e)	symmetrical position of rotors low maximum measure lower MTOW	throttle in hover >50% asymmetrical weight distribution low endurance

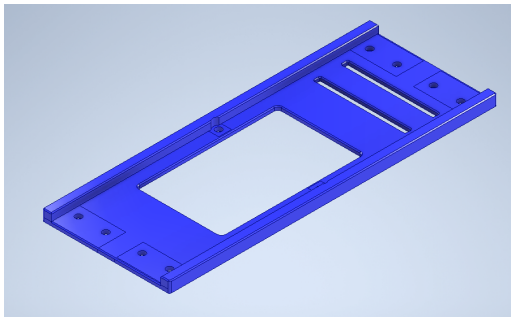
Table 2.1: Advantages and Disadvantages of the configurations

Each possible configuration has advantages and disadvantages as presented in table **??**. Thanks to this analysis, configuration 3 is chosen as reference.

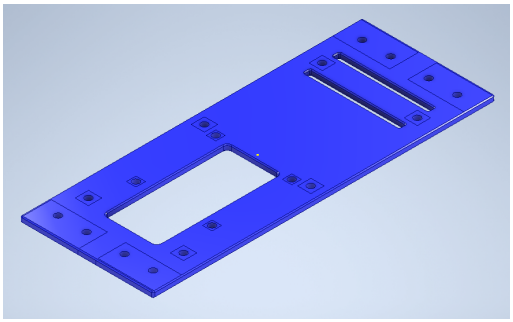
The frame is composed by two plates with aluminum arms fixed between them. The plates were designed with Autodesk Inventor. To reduce the weight of plates "Shape Generator" was used. It is a topology optimizer of Inventor that produces a 3D mesh useful to help in the design refinement. After setting the forces applied, the contact surface and the zones that must be preserved on the design of the first version of plates, the function calculates which part of the plates can be removed because structurally useless. The final CAD file of the lower plate is figure 2.14c on the next page Regarding the upper plate, two versions were designed because the first version (figure 2.14a on the following page), as will be exposed in the follow section, did not pass the static analysis. So, in this second version (figure 2.14b on the next page) two ribs with a thickness of 3 mm were added over boundaries plate and the general thickness was reduced from 3 to 2 mm to compensate the weight augmentation.



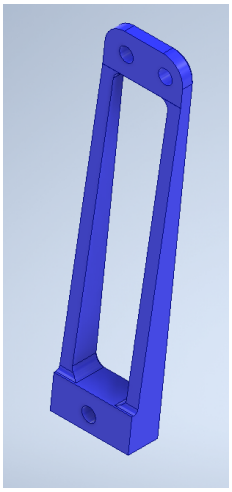
(a) *First version of the upper plate*



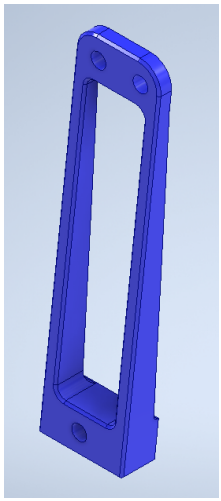
(b) *Second version of the upper plate*



(c) *Lower plate*



(d) *First half of the landing gear*



(e) *Second half of the landing gear*

Figure 2.14: CAD of the componets of the frame

As landing gear four structures were designed with the same procedure of the plates. Because of the constraints related to the 3D printing process, which will be explained in a future section, each one of these structures is divided in two parts (figure 2.14d and figure 2.14e):

2.2.10 Final configuration

In this section, the final configuration is summarized.

With respect to the CAD configuration in the precedent section the structure is different because of the design process that allows to define all the details. The final CAD is figure 2.15 on the following page The location of the components is changed for the same reason. The battery now is under the lower plate in order to improve the drone stability. Orange pi and STM are between the two plates in order to protect them and the cables

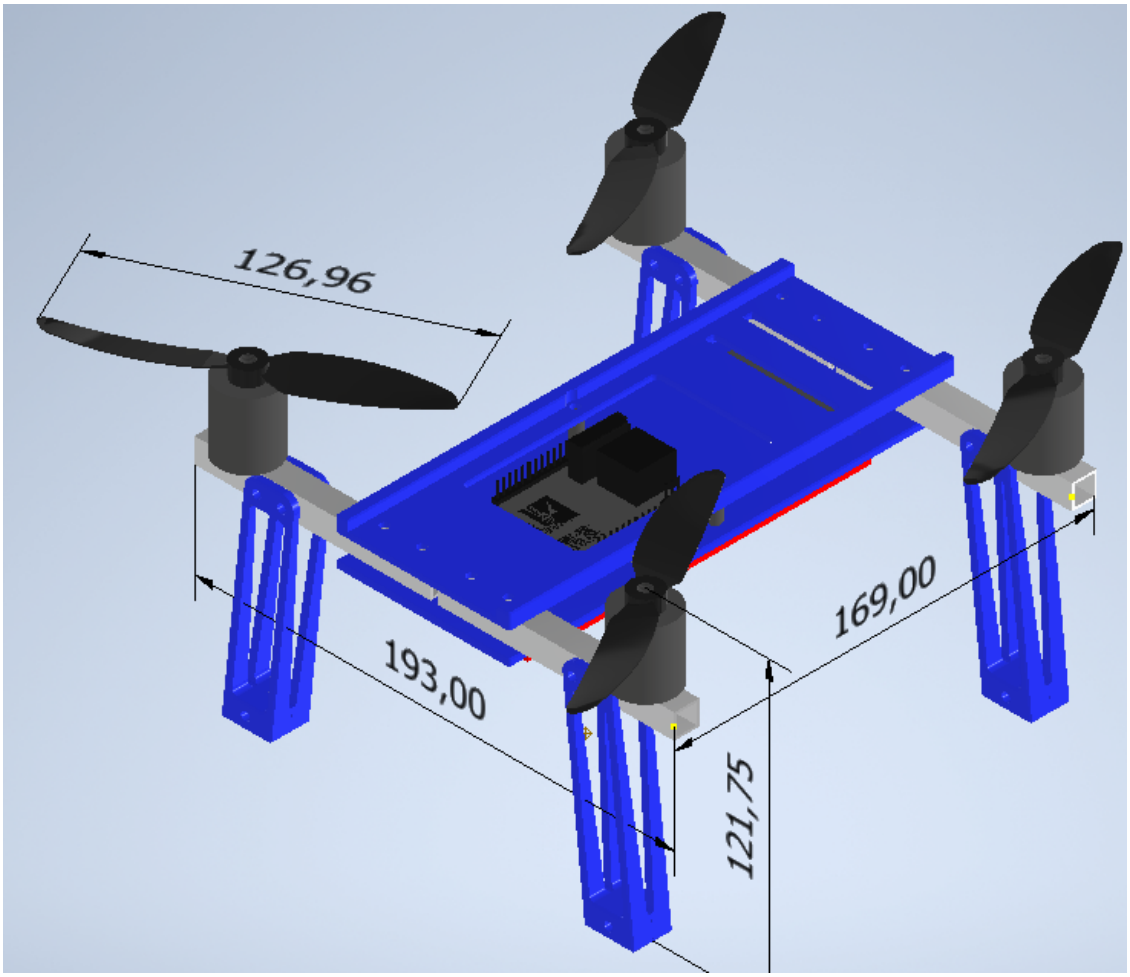


Figure 2.15: Complete CAD of the final configuration

connected from the propellers that rotate. MTOW in grams is exposed in detail in 2.2. The total MTOW is 504 g, therefore, each motor has to generate 126 gf. From thrust tests explained in the following section the following information are extrapolated:

- Operative voltage: 12.1 V

- Power per motor: 21 W

- throttle on hovering: 55%

Thus, the endurance, calculated with (2.25), will be of about 15 minutes.

Component	Quantity	Weight [g]
Avionics		
STM Nucleo l432kc	1	7
Orange Pi	1	26
Propulsion system		
Motor Tarrot MT1806	4	18
Propeller 5035x2	4	2.2
HobbyWing ESC 10A	4	6.5
Structure		
Upper plate	1	22.9
Lower plate	1	22.7
landing structure	4	9.2
aluminum arms	2	17
nuts and bolts	20	1.7
cables (estimated)		40
Battery		
Li-Po 3S 2200 mAh 35C	1	177
Sensors		
Camboard pico flexx	1	8
servo sg90	1	9
total		504

Table 2.2: MTOW of drone in detail

2.3 Tests and analyses on components

In this section, thrust tests and Structural analyses on CAD plates are presented. Thanks to thrust tests, batteries and propellers were chosen. Some basic structure analyses were made to verify that the plates do not break under the loads.

2.3.1 Thrust tests

Thrust tests allow to find an analytical relationship between the PWM signal from the controller and the thrust produced by a motor, so it is a fundamental step in simulink model building. These information are also provided by sellers in the datasheet (figure 2.16 on the next page) but they are often incomplete. So it is useful to make thrust tests to check the data and to calculate the thrust analytical expression that connects the controller and the plant in the simulink model.

The test bench used is a RCBenchmark Series 1520 Thrust Stand [**41**]. The thrust

TAROT产品测试记录表(Sample Test Report of the life)

Spindle specifications 锭子规格 (直径*内径*高度)	18*6*6	Spindle materials 锭子材料	川崎1200# 0.2MM
The magnet material 磁铁材料	N42SH	trunnion 轴径	2MM

负载测试数据The load test data:

The voltage 电压 (V)	Paddle size 桨尺寸	current 电流 (A)	thrust 推力 (G)	power功率 (W)	efficiency 效率 (G/W)	speed 转速 (RPM)	Working temperature 工作温度 (°C)
7.4	5030碳桨	1	70	7.4	9.5	7570	20
		2	120	14.8	8.1	10160	
		3	160	22.2	7.2	11910	
		4.4	210	30.0	7.0	13530	
	5*4.5三叶桨	1	50	7.4	6.8	6330	22
		2	100	14.8	6.8	8350	
		3	140	22.2	6.3	9600	
		4	170	29.6	5.7	10540	
		5	200	37.0	5.4	11450	
		6.2	240	42.0	5.7	12330	
11.1	5030碳桨	1	80	11.1	7.2	8870	28
		2	140	22.2	6.3	11500	
		3	180	33.3	5.4	13220	
		4	220	44.4	5.0	14680	
		5	260	55.5	4.7	16010	
		6	300	66.6	4.5	16890	
		7	320	77.7	4.1	17990	
		8	380	88.8	4.3	18510	
	5*4.5三叶桨	1	70	11.1	6.3	7120	35
		2	130	22.2	5.9	9330	
		3	180	33.3	5.4	10780	
		4	210	44.4	4.7	11810	
		5	250	55.5	4.5	12820	
		6	290	66.6	4.4	13600	
		7	320	77.7	4.1	14030	
		8	340	88.8	3.8	14660	
		9	360	99.9	3.6	15210	
		10.6	410	105.5	3.9	15910	

Figure 2.16: Datasheet Tarrot MT1806

generated is measured by an extensometer. The extensometer is connected to a board that sends the signal to the motor ESC and connects the bench to a PC with a USB cable. With the software of RCBenchmark the PWM signal can be varied manually or with scripts to collect data (figure 2.17 on the following page).

In this thesis, a script that provides a step test where the ESC signal varies from 1000 and 2000 μs was used to test the motor with 5035x2 and 5045x3 propellers and with 2S and 3S Li-Po batteries. Data collected from these tests can be fitted with a third order polynomial function. The results are shown in figure 2.18 on the next page, figure 2.19 on page 33, figure 2.20 on page 33 and figure 2.21 on page 34. Thanks to these results Li-Po 2S batteries were rejected because the throttle in hovering would be too high. Between the two propellers, the 5035x2 was preferred because the three-blades one draws more current and a throttle of 55% is low enough.

This bench tester cannot measure the moment, but it can be calculated indirectly from

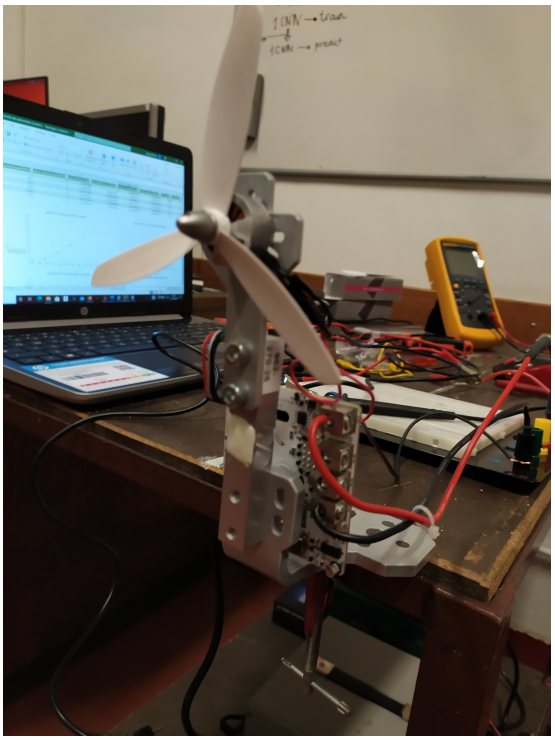


Figure 2.17: Thrust test bench

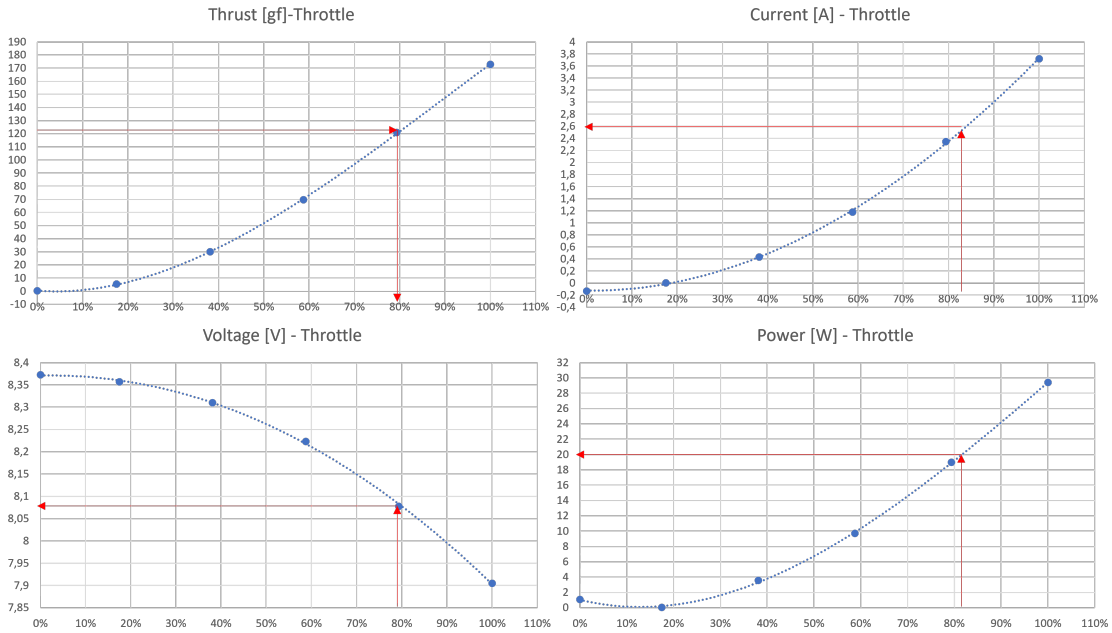


Figure 2.18: Results of thurst test bench with battery 2S and propeller 5035x2

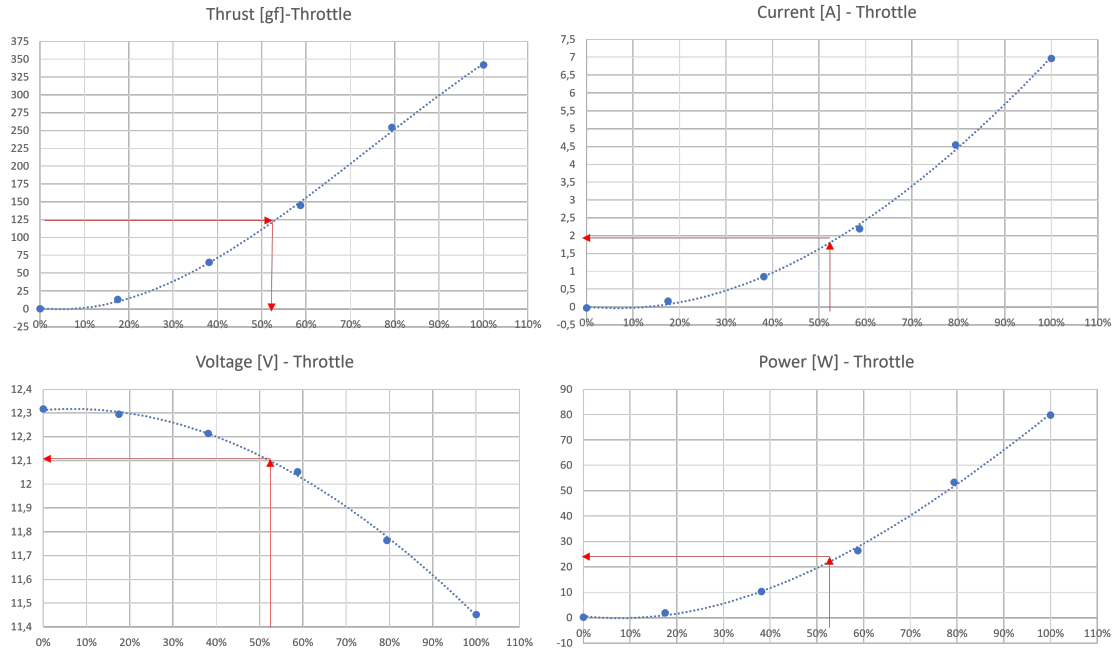


Figure 2.19: Results of thurst test bench with battery 3S and propeller 5035x2

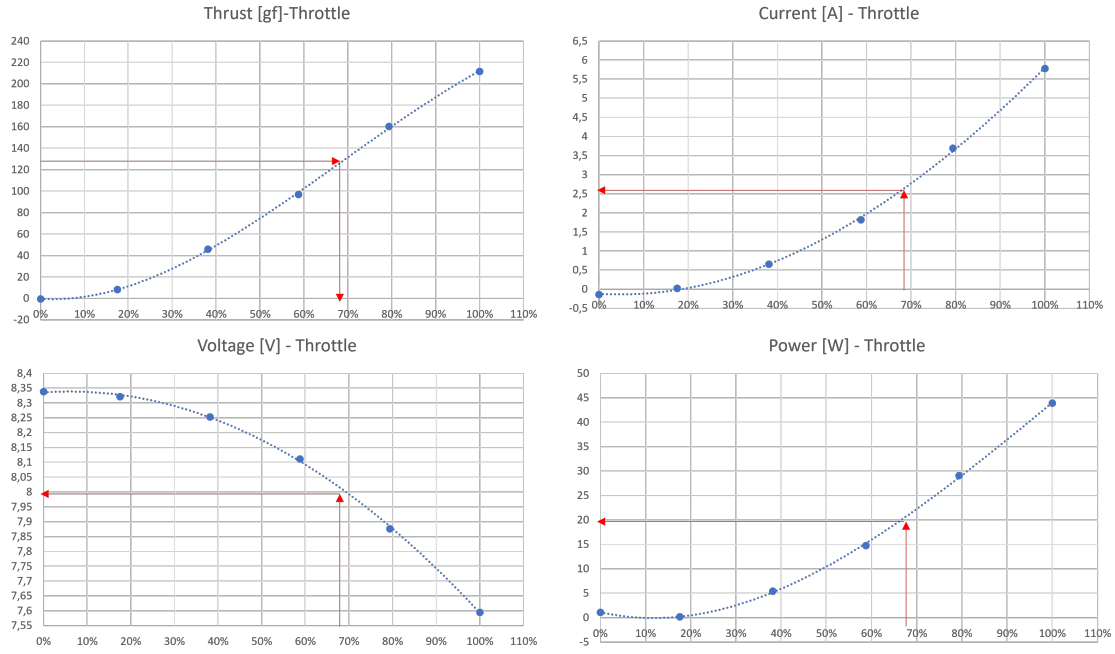


Figure 2.20: Results of thurst test bench with battery 2S and propeller 5045x3

the electrical power and the speed of the motor with the following equation:

$$\tau = \frac{P * RPM * 0.7}{2\pi}[Nm] \tag{2.26}$$

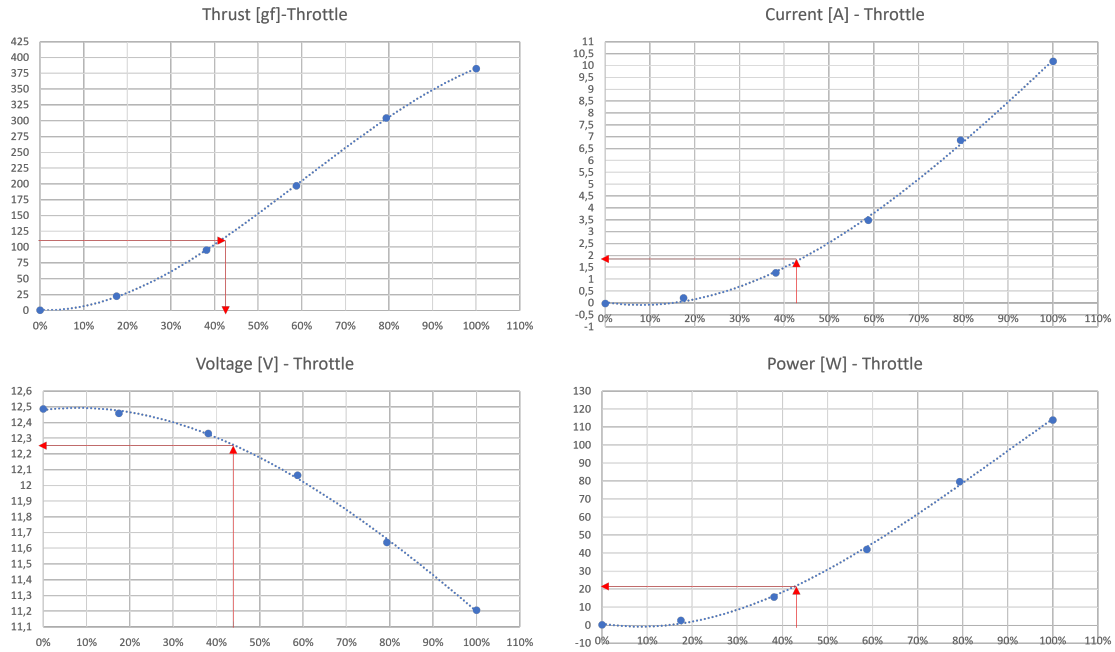


Figure 2.21: Results of thurst test bench with battery 3S and propeller 5045x3

where 0.7 is a coefficient to simulate the losses from electrical power to mechanical power. The thrust curve is:

$$T(x) = (2.382 * 10^{-6})x^2 - (3.628 * 10^{-3})x + 1.177 \tag{2.27}$$

and the moment curve is:

$$M(x) = (1.045 * 10^{-8})x^2 - (5.332 * 10^{-3})x - 5.277 * 10^{-3} \tag{2.28}$$

These curves, the maximum thrust and the maximum moment will be also used to update the parameters of the motors in the simulations.

2.3.2 Structural analyses

To verify that the plates and the gear structures could resist the loads, simplified static analyses were made. These analyses were made with the Inventor Stress Analysis tool. Firstly, the material must be set, because polylactic acid (PLA) is not in their database, Acrylonitrile butadiene styrene (ABS) is used instead because their characteristics are similar. Then constraints and loads must be set:

- The principal loads on the drone correspond to the thrust generated. Weighs of components are negligible with respect to them. Because these are rough analyses,

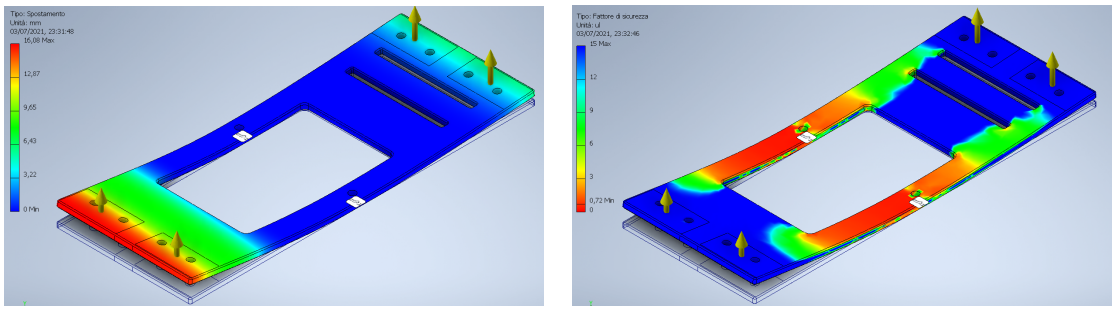
considering the difference between ABS and PLA and considering inventor does not keep in consideration the infill of 3d printed plates, loads were multiplicated with a safety factor of 10.

- Moments around the z axis are avoided because there are only two arms, each arm has two motors mounted, so regarding moments there are only the ones around x axis.

- Constraints are set where plates are connected with spacers.

- Analysis on gear structures simulate the landing so loads are set where the bolts are and the constraints are set in the lower zone.

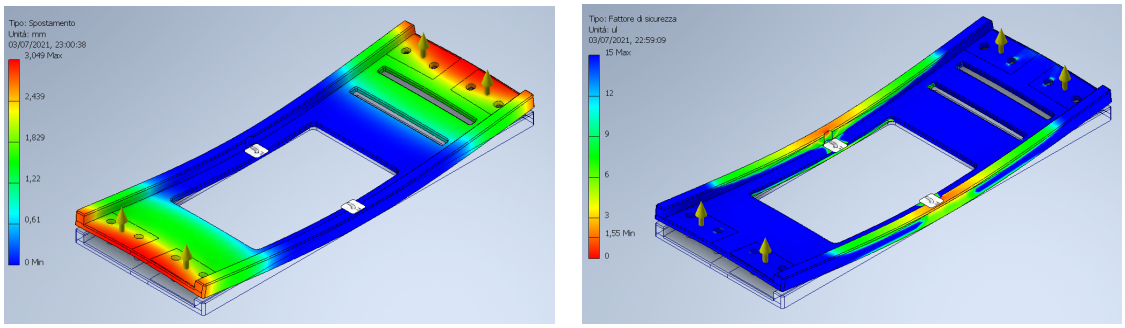
After making the automatic contact detection and generating the mesh the model can be solved.
As seen in the precedent section, for the upper plate two versions were made. With the first, these results (figure 2.22) were obtained. Displacements are too high and the safety



(a) *Displacements* (b) *Safety factor*

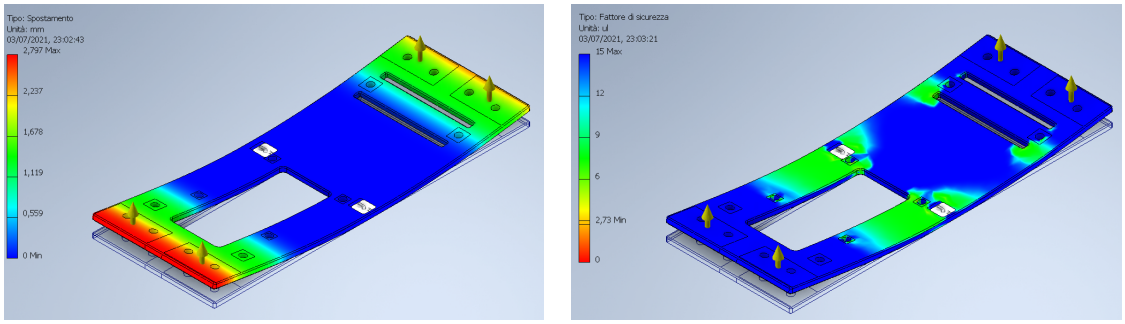
Figure 2.22: Analyses on first version of the upper plate

factor too low so the plate risks to break because of the moment generated around the x axis. For this reason, a second version was made. In order to increase the stiffness of the plate two ribs with a thickness of 3 mm were added over the boundaries plate and the general thickness was reduced from 3 to 2 mm to compensate the weight augmentation. Now the upper plate has a maximum displacement of 3 mm and a minimum safety factor of 1.55 (figure 2.23 on the following page), so the plate does not break under the loads.
The lower plate has a maximum displacement of 2.8 mm and a minimum safety factor of 2.73 (figure 2.24 on the next page), so the plate does not break under the loads. The landing gear structure has a maximum displacement of 0.6 mm and a minimum safety factor of 4.81 (figure 2.27 on page 38), so the structure does not break under the loads.



(a) *Displacements* (b) *Safety factor*

Figure 2.23: Analyses on second version of the upper plate



(a) *Displacements* (b) *Safety factor*

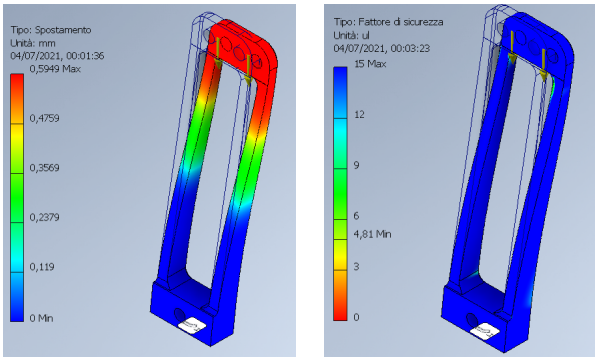
Figure 2.24: Analyses on the lower plate

2.4 Construction of the drone

In this section, the construction of the drone is described. Firstly, the 3d print process of of the plates and the construction of the frame are presented. Then the two different configurations assembled are described: the first one is the designed configuration, the second one is the version that will be used to experimentally test the guidance algorithms. This second version has a PX4 cuav v5 nano and a transmitter instead of ST microcontroller and a camboard pico flexx.

2.4.1 Construction of the frame

The plates which compose the frame are 3D printed. 3D printing is an additive manufacturing process where an object can be created starting from its CAD model. In this process the material is extruded layer by layer to create the final object. Unlike classic manufacturing process 3D printing does not remove material so the quantity of waste material is reduced at minimum and production times are lower with respect to the precedents.



(a) *Displacements* (b) *Safety factor*

Figure 2.25: Analyses on strucutures for the landing gear

In this thesis, PLA is used as printing material. PLA is one of the most popular materials used in this process. That is because it can be printed at low temperature and does not need the heated bed. Moreover, it is easy to print, inexpensive and one of the most environmentally friendly materials for 3d printing because it is biodegradable.[**42**]

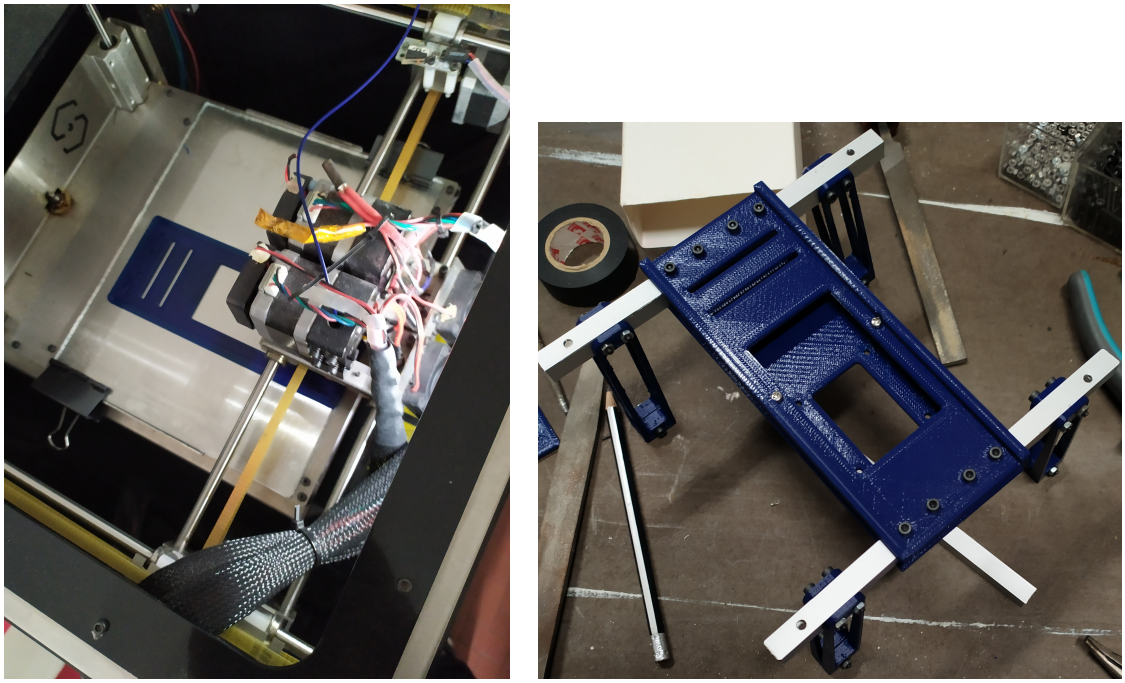
Firstly, the CAD models of the plates are saved as STL file, then these files are added

(a) *Upper view* (b) *lateral view*

Figure 2.26: Upper and lateral views of an arm

to simplify3D. Simplify3D is a program where the model is divided in layers, and it prepares the file to be ready to be printed. The model is positioned on the plate and the configuration is set. It slices the models in various layers. Our models are divided in layers with a height of 0.2 mm, the three top and bottom layers are fully printed, the remaining central layers are printed with an infill of 20% to reduce the weigh. After setting all the parameters the program creates a gcode file that must be inserted with a micro sd card in 3d printer. The glass plate of the printer is cleaned because it must be completely smooth and, after appllying a layer of lacquer to help the remotion of the object from the plate after the printing, the printing process can be started. (figure 2.27a on the next page). Because of how the material is extruded, a layer could be printed badly where the layer below does not have material. Thus, the structures for the landing gear cannot be printed whole but each one must be divided in two halfs. the arms have on the upper and lower

sides two pass-through holes to attach them with the plates and three pass through holes to mount the motors on them (figure 2.26a on the preceding page). In addition, they have two pass-through holes in the lateral sides to attach the landing gears to them (figure 2.26b on the previous page). After printing the plates and machining the arms, the frame can be assembled (figure 2.27b).



(a) *3D printing process* (b) *Complete frame*

Figure 2.27: 3D printing process & complete frame

2.4.2 Final assembly

Finally, the drone can be built. The scheme in figure 2.28 on the next page is been followed to connect all the components. The UBECs are needed because the boards need a voltage of 5V and the voltage of the battery is 11.4 V. The result of the assembly is in figure 2.29 on the following page.

To test the guidance algorithm an alternative version is mounted. In this case, instead of the microcontroller STM and the camboard pico flexx a PX4 CUAV v5 nano is used as autopilot and a trasmitter is added (figure 2.30 on page 40). That is because PX4 have sensors included and the camboard is not needed. The characteristics of PX4 CUAV v5 nano are reported in table 2.3 [**43**]. The new scheme to follow to connect all the components is (figure 2.31 on page 41) and the new version of the drone is in figure 2.32 on page 41.

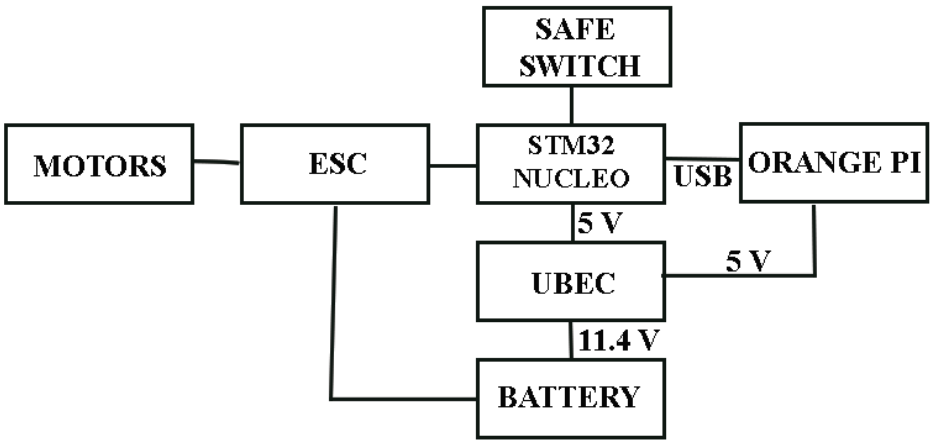


Figure 2.28: Schema of the components

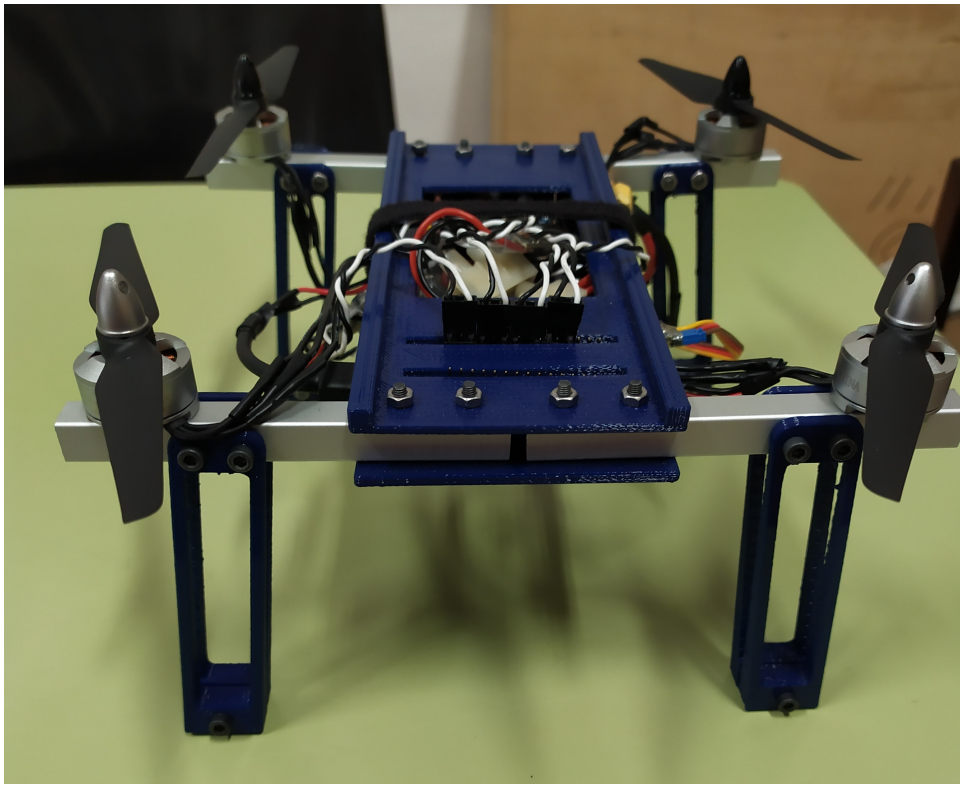


Figure 2.29: Final version of the drone

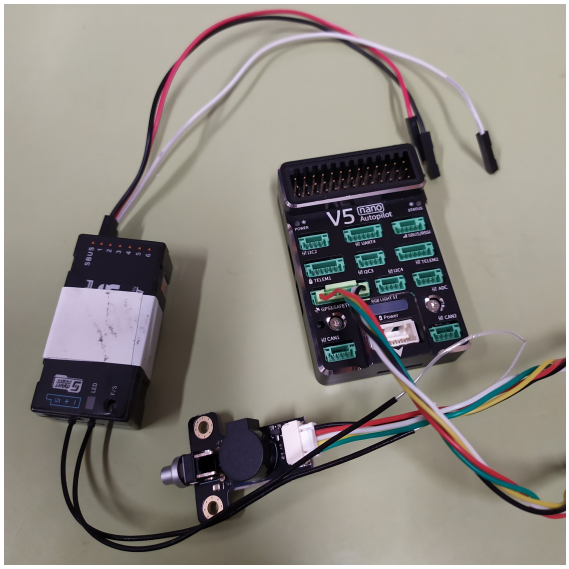


Figure 2.30: PX4 CUAV v5 nano & transmitter

Main FMU Processor	STM32F765 ◦ 32 Bit Arm® Cortex®-M7, 216MHz, 2MB memory, 512KB RAM
On-board sensors	Accel/Gyro: ICM-20689 Accel/Gyro: ICM-20602 Accel/Gyro: BMI055 Magnetometer: IST8310 Barometer: MS5611
Interfaces	8 PWM outputs: 3 dedicated PWM/Capture inputs on FMU Dedicated R/C input for CPPM Dedicated R/C input for Spektrum / DSM and S.Bus Analog / PWM RSSI input 4 general purpose serial ports 3 I2C ports 4 SPI buses 2 CAN Buses Analog inputs for voltage / current of battery 2 additional analog inputs Supports nARMED
Power System USB Power Input	Power Brick Input: 4.75 5.5V 4.75 5.25V
Dimensions Weight	60*40*14mm 40g
Operating temperature	-20 85°C

Table 2.3: Characteristics of PX4 CUAV v5 nano

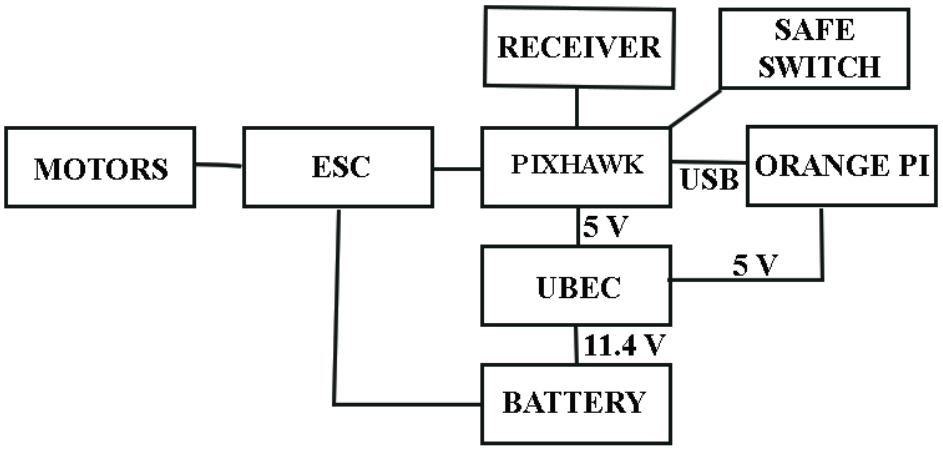


Figure 2.31: Schema of the components for experimental test

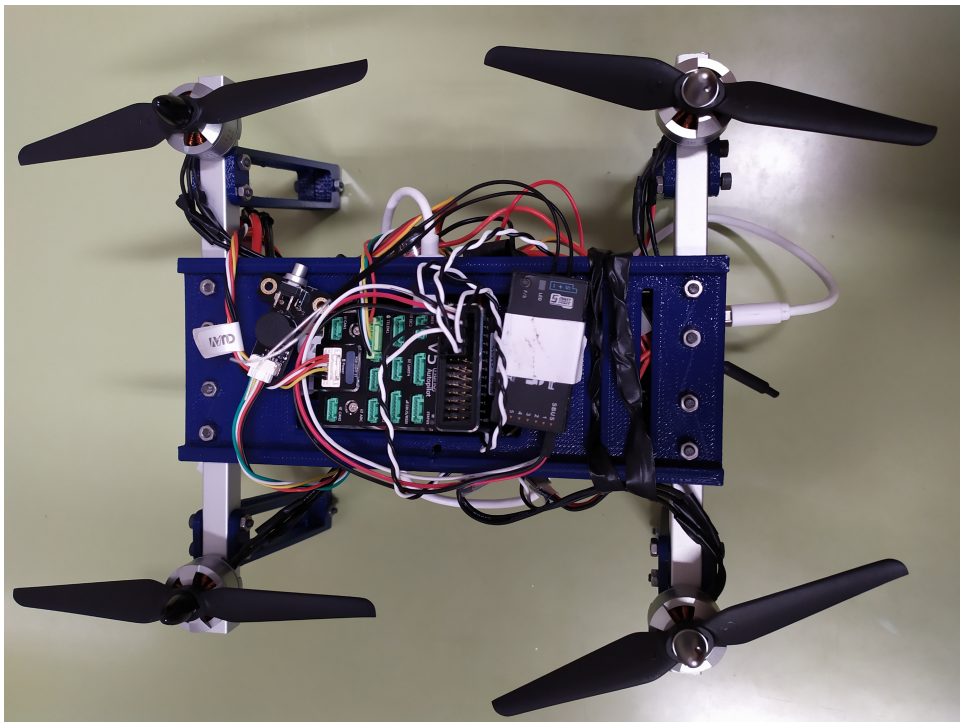


Figure 2.32: Version of the drone for experimental test

Chapter 3

Guidance algorithms

This chapter is dedicated to guidance algorithm. Firstly, an overview of guidance algorithms is made, then the two chosen algorithms are exposed. The first one is an improved APF that avoids GNRON and local minima problems and can be used in dynamic environments. The second one is an RRT*FN-DAPF, an improved version of RRT with several advantages with respect to RRT. It can find the optimal path, uses a fixed number of nodes in order to improve the computational effort, uses an APF to converge faster and can be used in dynamic environments.

3.1 Overview about guidance algorithms

The guidance system allows the drone to reach the goal avoiding the obstacles. These obstacles can be static or dynamic, known a-prori or not. the type of obstacle can lead into very different designs of the guidance algorithm.
The inputs of the guidance algorithm are the kinodynamic information of the drone and the kinetic information of the obstacles. Depending on the type of guidance algorithm it can need more or less of these constraints. These information are produced by the navigation system which is composed by sensors and observers such as the karman filter that estimate variables of the system that cannot be measured by the sensors. The outputs are the reference position, the reference velocity, the reference acceleration, and the reference yaw angle. These information are the inputs of the controller that calculates the forces needed by the plant to allow the drone to follow the references (figure 3.1 on the following page).

The schemes of a guidance algorithm can be various, but a classical configuration can be composed of a path planning, a trajectory generator, and a yaw angle generator (figure 3.2 on the next page).
As suggested by its name, the path planning generates the path that is a continuous curve

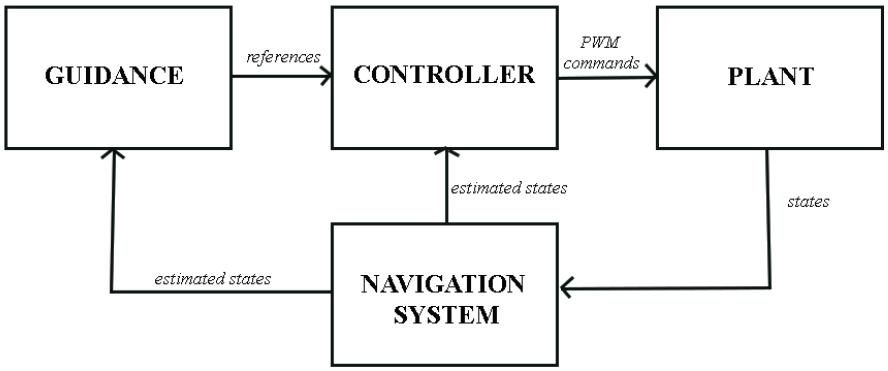


Figure 3.1: Architecture of the model

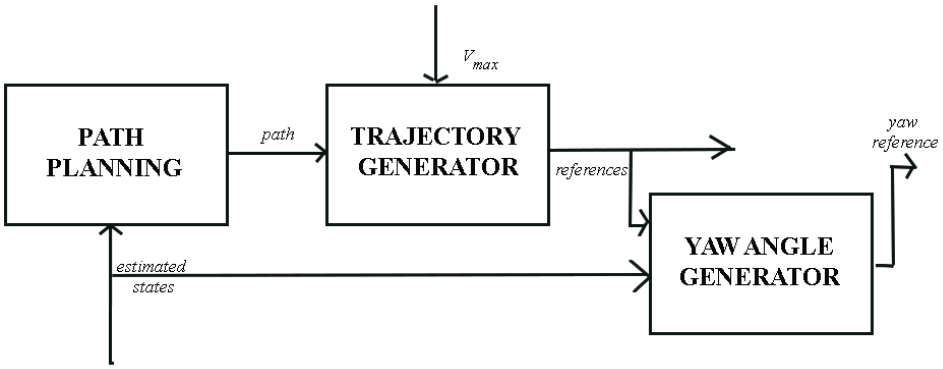


Figure 3.2: Architecture of the guidance system

in 3D physical space that starts from the initial point and ends in a point named goal, traced by the drone. Usually it is composed by a series of waypoints. In this phase no time information are considered. Path planning can be divided into five categories [**14**]:

Sampling based algorithms: This kind of algorithm divides the environment in a set of nodes and then map the environment searching randomly a feasible path. These algorithms need a low computational effort and are simple to implement, so a on-line implementation is possible. The reduced dependence on environment model allows to implement them in various environments. However, they depend on the guess made in every step which implicates that some of these algorithms find a path that is not always the optimal path. Some of the algorithms included in this category are:

- Rapidly Exploring Random Trees (RRT): this algorithm randomly samples a new node in each step, if there are no obstacles between the two nodes the new node is added. When the new node added corresponds to the goal the search is ended, and the path is determined. This algorithm cannot find the optimal path and does not

function in dynamic environment, but it has a low computational cost. Starting from this algorithm other improved versions were developped, for example RRT* which can find the optimal path because an optimal search algorithm is added.

- Probabilistic Road Map (PRM): This algorithm as the prevoius one randomly samples a new node in each step. But, differently from RRT, if the collision check is negative, it connects the new node to a determined number of nearest nodes and to nodes included in the region around the new node. Like RRT this algorithm has a low computational cost, but does not work in dynamic environment and needs a optimal search algorithm to find the optimal path. Starting from this algorithm a family of improved versions were developped.

- Artificial Potential Field (APF): in this algorithm a mathematical function is related to the goal and another to obstacles. The first function represents the attractive potential field and it is centered in the goal. This function has the minimum in the goal position that allows the goal to attract the drone. The second function represents the repulsive potential field and it is centered in the centre of the obstacles. Each obstacle produces its own repulsive potential field which is maximum in the obstacle position. It allows the obstacles to repulse the drone. Then the gradient of these potential fields is made and used to produce the reference velocity of the drone. This algorithm is computationally very inexpensive, but it risks falling into local minima and not reaching the goal if there is a nearby obstacle. An advantage of this algorithm is that it can find the optimal path.

Node based algorithms: This kind of algorithms explores a decomposed graph, so they need the environment to be completely defined. They calculate the cost of the path and find the optimal path exploring through the nodes. They are more computationally expensive than sampling based algorithms but not so much as to make the implementation on-line unfeasable. Because of these peculiarities, they are principally used in combination with a sampling based algorithm. Some of these algorithms are:

- Dijkstra's algorithm: It is named after its creator. this algorithm is composed by five step:

  – The minimum cost of starting point is initialized to 0 and the one of all the other nodes to infinite. The starting node is used as first current node.

  – For each node near the current node the cost is evaluated, if the cost evaluated is less than the minimum cost, the minimum cost value of the node taken in consideration is updated with this new value.

  – The node non visited with the smallest minimum cost is taken as new current node.

  - these three last step are repeated using the new node as current node until all the nodes are visited.
  - The path can be identified by taking the nodes with the smallest minimum cost.

  The cost can be any parameters, for example the distance or the time to reach the new node. This algorithm can found the optimal path. However, it cannot be used in dynamic environment and the computational cost is too high to implement this algorithm on-line. This algorithm is the base for other node based algorithms.

- A-Star (A*): This algorithm is an improved version of Dijkstra's algorithm. It reduces the number of nodes investigated by introducing a heuristic estimation of the cost. In the cost function a term that identifies the cost to the goal from this new node is added. This improvement allows to search only in the direction where the goal is. So, the convergence is faster, the computational effort is lower and it can be implemented on-line.

- D-Star (D*):It is a famous version improved of A*. This algorithm can be used with dynamic obstacles unlike A*. The cost function is improved so at each time a minimum heuristic function is updated when the drone encounters a new obstacle. The problem is D* uses unrealistic distance in its graph and the computational effort is higher.

- Cell decomposition (CD): this algorithm is different form the previous algorithms. It takes the whole environment and begins to decompose it in cells which are smaller regions. Then, the path can be defined by taking the cells that are free of collision and that connect the starting point with the goal. This family of algorithms can be divided in exact cell decomposition and adaptive cell decomposition. In the first one the number of cells is fixed. In the second one the decomposition continues until in the cell there are not obstacles or an arbitrary limit resolution is reached. Exact cell decomposition allows to find always the optimal path if it is possible but it could need an high computational effort. Adaptive cell decomposition has a lower computational cost but sometimes the path founded is not the best path possible.

Mathematical model based algorithms: These algorithms include linear algorithms and optimal control. This kind of algorithms models the environment in a way more complete than sampling based algorithm because they use all the kinodynamic constraints and the drone is no more point-like. For this reason, these algorithms can be identified also as trajectory planning. Employing many more constraints these algorithms can adapt themselves in any 3D cluttered environment. On the other hand, they have a high computational cost so they can not be implemented on-line.

Bioinspired algorithms: These algorithms imitate the biological behavior to solve problems. They can solve nondeterministic polynomial-time (NP) hard problems with a lot of environmental constraints and generate a near optimal path. Mathematical model based algorithms often fail to solve these problems. Bioinspired algorithms can be divided in two subcategories:

- Evolutionary algorithms (EA): They are stochastic search approaches. They start by selecting randomly feasible solutions as the first generation. Then they evaluate the fitness of the current generation taking in consideration all the kinodynamic constraints. According to this calculated fitness, a set of individuals is selected as parents for the next generation. Lastly mutation and crossover steps are made to generate new offsprings. This procedure is repeated until the goal is reached and the best fitness individuals are set as nodes of the optimal path. They include a lot of different algorithms:
  - The most popular evolutionary algorithm is the Genetic Algorithm (GA). In GA all the individuals can exchange information so the next generation can converge faster than other EA. On the other hand, if the population becomes too similar, GA risks to converge prematurely.
  - Another EA is the Artificial Bee Colony (ABC). There are three groups of bees: employed bees associated to specific food sources, onlooker bees looking in the hive the choice of food sources and the scout bees that search randomly for food sources. It is easy to implement and has a simple structure and few control parameters, but it has the problem about convergence because it risks converging prematurely with a low accuracy and a slow rate. Some improved versions were developed, for example differential evolution artificial bee colony (DE - ABC) overcomes ABC disadvantages using a DE algorithm that derives new populations through mutation, crossover, and selection operations.
- Neural network (NN): This algorithm generates a dynamic landscape by mimicking the operations of a human brain. The drone is attracted by unsearched areas. In each step the maximal neural activity among the neighbor neurons is chosen to find a new location. NN shares the disadvantages with others bioinspired algorithms, so the reliability and time consumption are not guaranteed. But it has the advantage to be stable under sudden changes in the network.

Multifusion based algorithms These algorithms are composed by two or more algorithms of previous categories. The objective is to overcome disadvantages of single algorithms with the advantages of other algorithms implemented. Multifusion algorithms can be divided in Embedded multifusion algorithms (EMA) and Ranked multifusion algorithms (RMA):

- In EMA, the algorithms work simultaneously. For example, using the APF as function cost for A* the algorithm becomes viable also for dynamic environment. Or using APF in combination with a RRT in the sampling of new nodes allows to focalize the search only in the goal direction reducing the computational effort and converging faster.

- In RMA, the algorithms work on separate levels. For example, using a node based algorithm like an A* after a sampling based algorithm like a PRM. The PRM forms the road map and the A* finds the optimal path. The computational effort of this combination is lower than that of A* because the PRM reduced the number of possible paths that A* has to scan. Another example is DE - ABC. It overcomes ABC disadvantages using a DE algorithm that derives new populations through mutation, crossover, and selection operations.

But to have a complete trajectory also information about velocity, acceleration and time are needed. In fact, the trajectory is a set of states that are associated with time. A trajectory generator is used to obtain these information. Moreover, it allows to have a smoother path than previous path. The trajectory generator takes the path and the maximum speed of the drone as input and returns as output the trajectory. The most popular trajectory generators are the minimum-acceleration trajectory generator and the minimum-snap trajectory generator.

- Minimum-acceleration trajectory generator returns a segmental third order polynomial trajectory. This method achieves the balance between velocity, safety from the obstacles and computational effort.

- Minimum-snap trajectory generator returns a segmental seventh order polynomial trajectory. This method has a bigger computational effort than the previous, but it generates a smoother acceleration profile.

Lastly the yaw angle generator allows to produce the desired profile of yaw angle. Usually, the request is that the drone has the nose pointing to the goal.

3.2 Artificial Potential Field guidance algorithm

The artificial potential field algorithm was designed by Khatib for the first time in 1986. Unlike classic guidance algorithms the APF is not divided in path planning and trajectory generator. In this method the drone is considered point-like, and the environment is modeled with a mathematical function that represents the potential field. The total potential field is the sum of the attractive potential field dependent on the goal and the repulsive

potential field dependent on the obstacles:

$$U_{tot}(x, y, z) = U_{att}(x, y, z) + U_{rep}(x, y, z)$$

The attractive potential field is:

$$U_{att}(x, y, z) = -\frac{1}{2}k_a((x_d - x_g)^2 + (y_d - y_g)^2 + (z_d - z_g)^2)$$

where k_a is the attractive constant, x_d, y_d and z_d are the coordinates of the drone and x_g, y_g and z_g are the coordinates of the goal.
The repulsive potential field is:

$$U_{rep}(x, y, z) = \sum U_{rep,i}(x, y, z)$$

where $U_{rep,i}$ is the repulsive potential field of the i-th obstacle:

$$U_{rep,i}(x, y, z) = \begin{cases} -\frac{1}{2}k_r(\frac{1}{\rho_O(x,y,z)} - \frac{1}{r_O})^2 & \text{if } \rho_O \leq r_O \\ 0 & \text{if } \rho_O > r_O \end{cases}$$

where k_r is the repulsive constant, r_O is the radius of the sfere of influence of the obstacle and ρ_O is the closest distance between the drone and the obstacle:

$$\rho_O(x, y, z) = \sqrt{(x_d - x_O)^2 + (y_d - y_O)^2 + (z_d - z_O)^2)}$$

with x_O, y_O and z_O as the coordinates of the obstacle.
Then, making the gradient of the potential field, normalizing it and multiplying it with the maximum velocity possible of the drone the desired velocity can be obtained:

$$\mathbf{v} = v_{max} * \nabla U_{tot}(x, y, z)$$

This is the speed that the drone needs to reach the goal while avoiding the obstacles.
But with this APF the GNRON problem and the local minima problem are not avoided and it is not applicable in a dynamic environment so it must be modified.

GNRON problem

The Goal Non Reachable with Obstacle Nearby is a problem that does not allow the drone to reach to goal if an obstacle is near. The reason is that the goal is in the radius of influence of the obstacle. So, the repulsive potential field is higher or equal to the attractive potential field and that does not allow the drone to enter this zone.
To avoid this problem the repulsive potential field is modified as follows introducing the dependency on the goal position:

$$U_{rep,i,gnron}(x, y, z) = \begin{cases} -\frac{1}{2}k_r(\frac{1}{\rho_O(x,y,z)} - \frac{1}{r_O})\rho_g^2(x, y, z) & \text{if } \rho_O \leq r_O \\ 0 & \text{if } \rho_O > r_O \end{cases}$$

where ρ_g is the distance between the drone and the goal:

$$\rho_g = \sqrt{(x_d - x_g)^2 + (y_d - y_g)^2 + (z_d - z_g)^2}$$

When $\rho_g < 1$, the repulsive potential field decreases while the drone is approching so the attractive potential field prevails and the drone can reach the goal.

Local minima problem

A local minimum is a point of the environment where the repulsive potential field is equal to the attractive potential field so the velocity desired of the drone becomes null and the drone gets stuck in this point.

In this thesis, the drone can't fall in a local minimum point because of the noise of the sensors implemented in the models but it can still fall in a local minimum zone. So, a solution for this problem is implemented.

To avoid this problem a virtual potential is placed in the previous location of the drone until he manages to escape:

$$U_{vir}(x, y, z) = \begin{cases} \frac{k_{vir}}{\rho_{vir}(x,y,z)} & \text{if the drone is in a local minimum zone} \\ 0 & \text{if drone is not in local minimum zone} \end{cases}$$

where k_{vir} is the virtual constant and ρ_{vir} is the distance between the drone and its previous location:

$$\rho_{vir} = \sqrt{(x_d - x_{d,prec})^2 + (y_d - y_{d,prec})^2 + (z_d - z_{d,prec})^2}$$

The previous position and the actual position are similar so U_{vir} is big enough to get the drone out of the local minimum zone.

Dynamic environment

When the obstacles are moving Khatib's APF is not valid anymore because is not safe enough. So, it must be modified to be used in this kind of environment.

A term depending on the obstacle velocity is added to the repulsive potential field:

$$U_{rep,i,d}(x, y, z) = \begin{cases} -\frac{1}{2}k_r(\frac{1}{\rho_O(x,y,z)} - \frac{1}{r_O})^2 - k_m\frac{v_{do}}{\rho_O} & \text{if } \rho_O \leq r_O \wedge v_{do} \geq 0 \\ 0 & \text{if } \rho_O > r_O \vee v_{do} < 0 \end{cases}$$

where k_m is a costant and v_{do} is the relative velocity component from the drone to the dynamic obstacle:

$$v_{do} = (v_d - v_O)^T e_{ao}$$

with v_d velocity of the drone, v_O velocity of the obstacle and e_{ao} is the pointing versor from the drone to the moving obstacle. So if $v_{do} \geq 0$ the drone moves toward the obstacle,

if $v_{do} < 0$ the drone moves away from the obstacle.

Now, this algorithm was implemented in the Matlab/Simulink model will be illustred. The strucuture used is in figure 3.3.

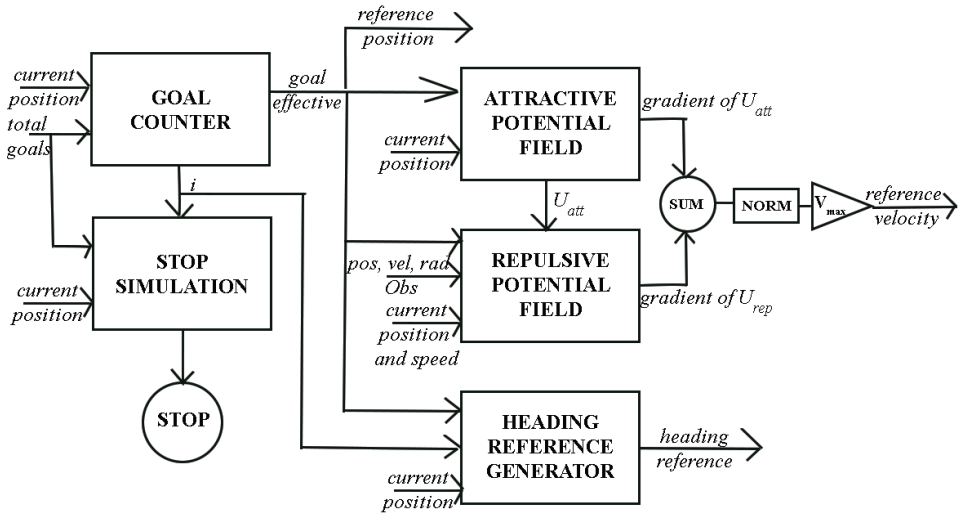


Figure 3.3: Structure of the APF guidance implemented

Goal counter

Since the drone flight is divided in take-off phase, flight phase and landing phase, a simulation input is an array with the coordinates of the waypoints where the drone passes from a phase to another.
The inputs of this function are:

- the total goals array
- the current position
- an index i that comes from a output of this function

This function returns the effective goal for the current phase, the previous goal and the index i that indicate the position of the current goal in the total goals array. This index, initialized as 1, goes through a delay block and returns as input of the function (figure 3.4). Its pseudocode is shown in algorithm 1. The effective goal is the reference position for the controller.

Attractive potential field

In this function, the attractive potential field is evaluated for the drone current position

Algorithm 1: [effective goal, precendent goal, i] = Goal counter (input)

initialization tolerance position;
if *i<length(total goals array)* **then**
  if *total goals array(i)-tolerance position < current position < total goals array(i)+tolerance position* **then**
    $i = i + 1$;
effective goal = total goals array(i);
if *i>1* **then**
  previous goal = total goals array($i - 1$);

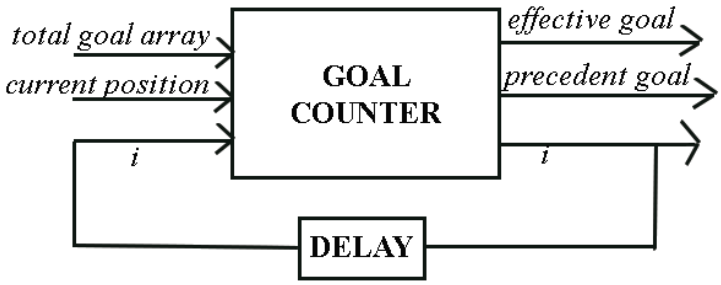


Figure 3.4: Goal counter block

and in the adjacent positions. Then the numerical gradient of the potential field is made using this 3D matrix of attractive potential field.
The inputs of the function are:

- the current position
- the effective goal
- the attractive constant and a constant step used to calculate the adjacent positions.

The outputs of this function are the attractive potential field that is needed in the repulsive potential field block and the numerical gradient of the potential field (figure 3.5 on the following page). Its pseudocode is shown in algorithm 2.

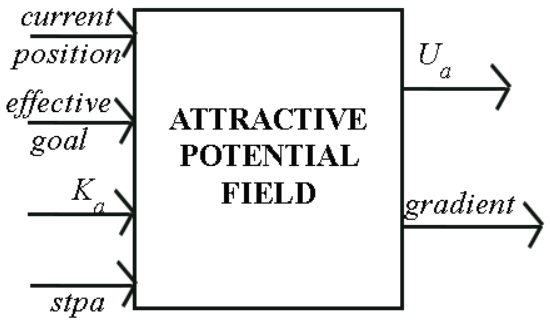


Figure 3.5: Attractive potential field block

Repulsive potential field

In this function the repulsive potential field is evaluated for the drone current position and in the adjacent positions. Then the numerical gradient of the potential field is made using this 3D matrix of repulsive potential field. In this block all the enhancements to improve the APF are implemented.
The inputs of this block are:

Algorithm 2: [Attractive potential field, gradient] =Attractive potential field (input)

```
calculation of positions 3D matrix;
calculation of attractive potential field of positions 3D matrix;
calculation of numerical gradient of attractive potential field;
```

Algorithm 3: [gradient] =Repulsive potential field (input)

```
calculation of positions 3D matrix;
for each obstacle do
    if GNRON then
        calculation of static GNRON repulsive potential field term of positions 3D
         matrix;
    else
        calculation of static classic repulsive potential field term of positions 3D
         matrix;
    calculation of dynamic repulsive potential field term of positions 3D matrix;
    Sum of repulsive potential field terms;
end
if local minimum problem then
    calculation of virtual potential field;
Sum of repulsive potential field and virtual potential field;
calculation of numerical gradient of total repulsive potential field;
```

- the goal effective and the previous goal
- the costant step analogous to that of the previous block and all the repulsive costant
- the attractive potential field used to determine if the drone has fallen into a local minimum
- the position, the radius and the velocity of the obstalces
- the current position, the previous position and the velocity of the drone

The output of this function is the numerical gradient of the potential field (figure 3.6 on the next page). Its pseudocode is shown in algorithm 3. The reference velocity for the controller is obtained making the sum of these two gradients, normalizing it and multiplying it by the maximum allowed velocity of the drone.

Heading reference generator

In this function the heading reference is calculated. The drone must fly with the nose pointing to the next waypoint so a profile of the heading angle must be generated. However,

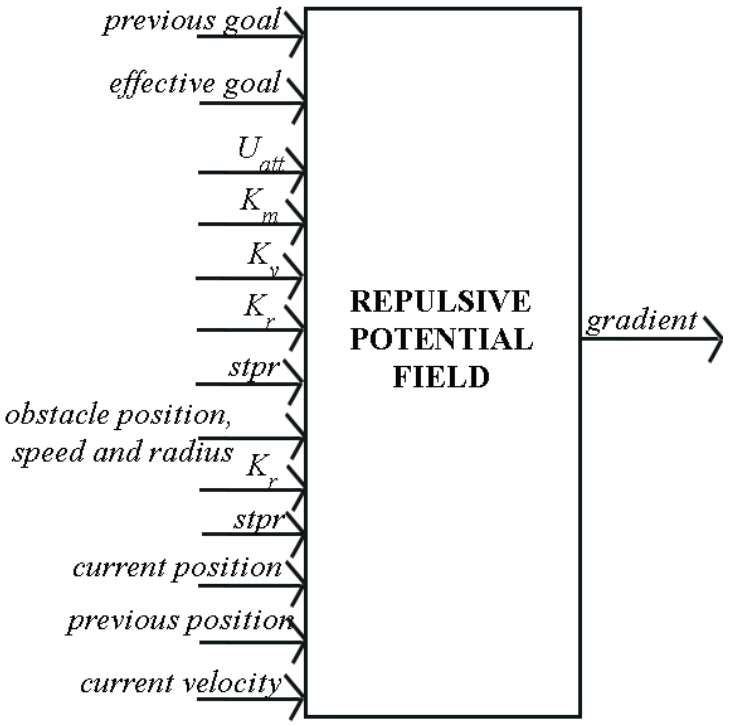


Figure 3.6: Repulsive potential field block

the waypoints which are along the vertical axis of the previous one must not be used in the generation of the heading reference. Take-off and landing are examples of this situation. So this function has a control to disitinguish these maneuvers and in case it uses the successive valid waypoint. In the landing maneuver the heading angle is kept constant.
The inputs of this funciton are:

- the position of the drone
- the total goals array, the effective goal
- the index i obtained from **Goal counter**
- the previous heading angle that comes from an output of this function.

The output of this function is the heading angle calculated, this value also goes trough a delay block and returns as input of the function to be used when the drone is in the landing phase (figure 3.7). Its pseudocode is shown in algorithm 4.

Stop simulation
This function stops the simulation when the landing is completed. This function is needed because the time of the simulaiotn is not fixed. The inputs of this function are:

Algorithm 4: [heading] = Heading reference generator (input)

```
if not landing then
    if vertical goal then
        take successive goal;
    else
        take effective goal;
    heading=tan^-1 (goal taken - drone position);
else
    heading=previous heading;
```

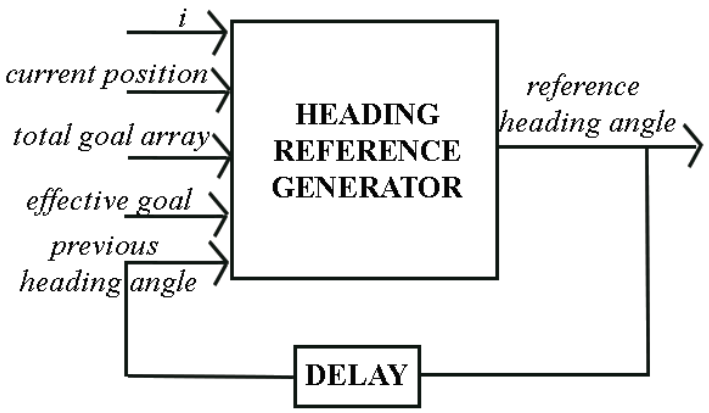


Figure 3.7: Heading reference generator block

- the drone position

- the total goals array

- the index i obtained from **Goal counter**

The output of this function is a boolean variable named y (figure 3.8) connected to a stop block. The stop block stops the simulation if it receives 1. Its pseudocode is shown in algorithm 5. The first if is necessary because, in its absence, the simulation will stop at the beginning when the drone has not yet taken off.

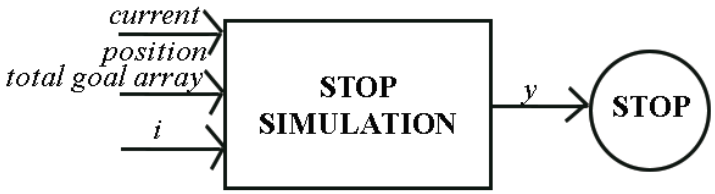


Figure 3.8: Stop simulation block

3.3 RRT*FNDAPF guidance algorithm

This guidance algorithm has the classic structure of guidance algorithm. The principal components are the path plannig, the trajectory generator and the heading angle generator. The heading angle generator is implemented in the same way that is implemented in the APF guidance algorithm. There are also the goals counter function and the stop simluation

Algorithm 5: $[y]$ = Stop simulation (input)

initialization tolerance for position;
if *i is equal to the length of the total goals array* **then**
    if *altitude final goals-tolerance < altitude drone < altitude final goals+tolerance* **then**
        $y = 1$;

function like in the previous guidance algorithm (figure 3.9 on the following page).

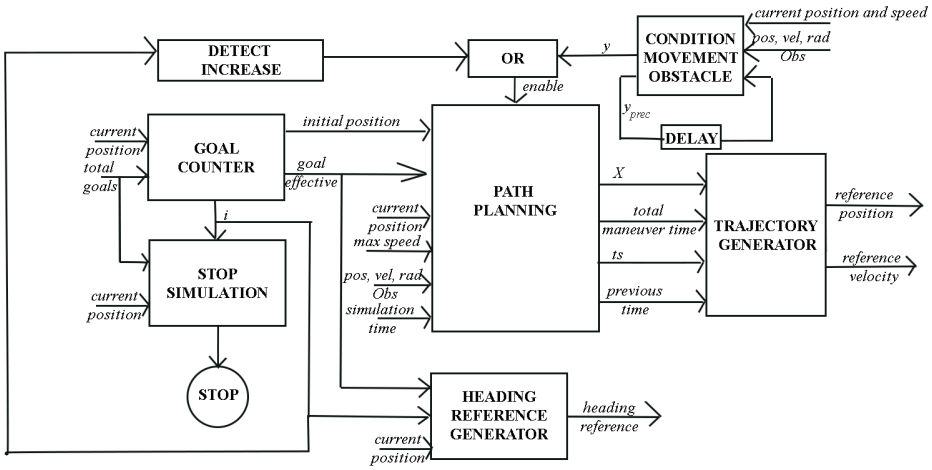


Figure 3.9: Architecture of the RRT*FNDAPF guidance implemented

Goal counter

This block is mostly implemented as in the previous guidance algorithm. It has one more output which is the value of the initial position for the path planning (figure 3.10). For each maneuver, this path planning find a new path. Thus this value must be updated. Its pseudocode is shown in algorithm 6.

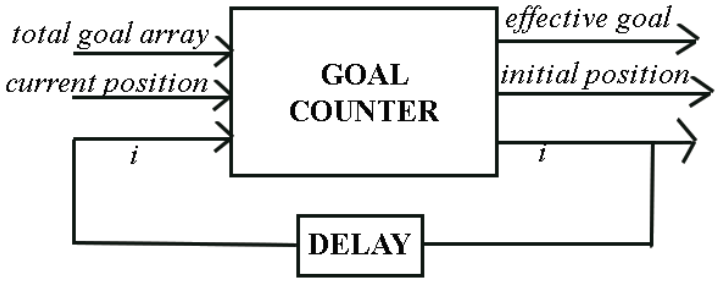


Figure 3.10: Goal counter block

Path planning

The path planning used is the RRT*FNDAPF which gives the name to the guidance algorithm. It is an improved version of the RRT algorithm and can be considered an embedded multifusion path planning algorithm. RRT algorithm randomly samples a new node in each step. If the minimum distance between this new node sampled and the nodes previously saved is minor than a tolerance the node sampled is taken into consideration.

Algorithm 6: [effective goal, precendent goal, i] = Goal counter (input)

initialization tolerance position;
if *i<length(total goals array)* **then**
  if *total goals array(i)-tolerance position < current position < total goals array(i)+tolerance position* **then**
    $i = i + 1$;
effective goal = total goals array(i);
if *i>1* **then**
  initial position = total goals array($i - 1$);

Otherwise, a nerear new node obtained with a steering function based on the euclidean metric is selected. If there are not obstacles beteween the node taken into consideration and the nearest of the node previously saved, the new node is added. The steering function is usefull in complicated zones as for example a narrow passage.
When the new node added corresponds to the goal the search is ended, and the path is determined as the path with the minimum cost between those found. As cost function the distance between the nodes is used. The disadvantages of this algorithm are that it can not find the optimal path, that it does not function in dynamic environment and that it randomly samples the new nodes so it could find a lot of nodes that are not useful.
The RRT*FNDAPF can avoid these disadvantages:

- The RRT* is an improved version that solves the problem of the optimal path with three new functions. The first one, when the new node is added, searches all existing nodes within a radius. The second one introduces a cost function that is the distance between the two nodes. The third one controls if one of these distances is smaller than the minimun distance previously calculated, and if yes, the node becomes the new parent of the new node.

- The RRT*FN allows to reduce the computational effort because it uses a fixed number of saved nodes. When the nodes list is full, if the goal is still not reached the algorithm finds the first childless node of the list and overwrites the new node on it. To understand if a node is childless, it compares the coordinates of the nodes with the list of coordinates of parent nodes. When the new node is added, the coordinates of the parent node are also saved.

- The RRT*FND is an improved version that can be used in dynamic environments. Two approaches can be used. The first one consists on rerunning all the path planning when the environment is updated considering the obstacles as static but this method is not computationally convenient. The second one consists on rerunning the path planning only when an obstacle passes trough the part of the path not yet used by

the drone. In addition, only nodes crossed by the obstacle are erased and the path is reconnected. This second approach is less computationally expensive and is the one used in this thesis.

- The RRT*FNDAPF allows to accelerate the convergence of previous algorithms. It uses an APF as search algorithm so new nodes are not randomly sampled anymore. A random component is left to avoid to fall in local minima. The APF function is added in the steering function. After calculating the new node from the one ramdomly sampled a component related to the goal is added and a component related to the obstacles is subtracted. This last component also helps the non collision function.

The inputs of this function are:

- the dimensions of the environment
- the position, the radius and the velocity of the obstalces
- the tolerance for the steering funtion, the maximum number of iteration and the number of nodes
- the initial position and the current position of the drone.

The output is the path.

In simulink path planning block is composed by an enabled subsystem and the condition that allows the subsystem to run (figure 3.11 on the following page). If the condition is not valid, the outputs of the subsystem are the last outputs calculated previously. In the enabled subsystem are present the path planning algorithm and two functions called *generate_ts* and *generate_X* which are part of the trajectory generator algorithm and will be exposed in the following section. *Generate_ts* generates the reference time in which the drone must arrive in the waypoints. *Generate_X* generates the coefficients of the polynomial equations of the references. By using this subsytem, the path, ts and X can be calculated only when a new maneuver begins or when an obstacle crosses the path. The path planning algorithm is made starting from a RRT* without collision check [**37**]. The pseudocode of this algorithm is shown in algorithm 7 on page 60, the function *steer* is shown in algorithm 8 on page 61 and the function *nocollision* is shown in algorithm 9 on page 61.
The condition, which enables the subsytem when a new maneuver begins, is the block *detectincrease*. It takes the index of the row of total goals array which corresponds to the effective goal and, if it is increased with respect to its value in the previous step, the output of the block will be the boolean value true.

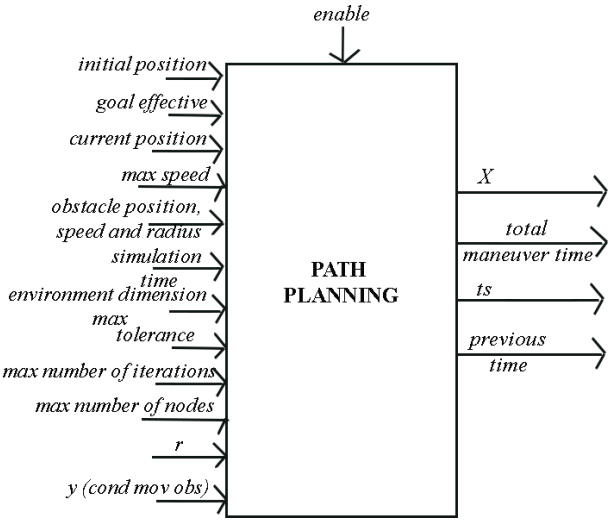


Figure 3.11: Path planning block

The condition, which enalbes the subsytem when a moving obstacle is detected, is a function self made.
The inputs of this function are:

- the position and the speed of the drone
- the position, the speed and the radius of the obstacles
- a variable y_{prec} that comes from a output of this function

The outputs are the boolean variable y and the variable y_{prec}. y_{prec} becomes 1 since y is true until the drone has passed the obstacle. y_{prec}, initialized as 0, goes through a delay block and returns as input of the function (figure 3.12). Its pseudocode is shown in algorithm 10 on page 62. If the output of one of these two conditions is true, the subsystem with the path planning is activated.

Trajectory generator

Path planning returns the reference position of the drone. But it is not sufficient because the controller needs more kinematic information. In addition to the path, the reference velocity is needed for the controllers implemented in the model. So, the trajectory generator combines a time and a velocity to each waypoint. The trajectory generator allows also to have a smoother path. In this thesis, a minimum acceleration trajectory generator is used. It returns a segmental third order polynomial trajectory and allows to reduce the computational effort with respect to the minimum snap trajectory that returns a segmental seventh order polynomial trajectory. This last one has the advantage of generating a

Algorithm 7: [path] = RRT*FNDAPF (input)

```
initialization of the variables;
if obstacle moving detected then
    q_start=current drone position;
for each iteration do
    q_rand = random node sampled;
    q_near = closest node from existing list;
    q_new = steer(q_rand, q_near, q_goal, obstacle) (algorithm 8 on the following page);
    if nocollision(q_new, q_near) (algorithm 9 on the next page) then
        parent of q_new = q_near;
        cost of q_new = distance between q_new and q_near + cost of q_near;
        q_nearest = nodes within a radius r;
        if nocollision(q_new, q_nearest) (algorithm 9 on the following page) then
            q_min= node with minimum cost between q_new and q_nearest;
        if q_min is a nodes of the existing list then
            parent of q_new = q_min;
        for each node saved do
            if node is not a parent then
                node is childless;
        end
        if maximum number of nodes is not reached then
            q_new is added to nodes list;
        else
            q_new is overwrited on the first node childless;
    if goal reached then
        break;
end
D = distance between goal and nodes;
parent of q_goal = node with minimum D;
add q_goal to path;
while node chosen has a parent do
    node chosen is added to path;
    parent of previous node is chosen;
end
order of path is inverted;
```

smoother acceleration profile but, since the controllers used do not need the acceleration as reference, it will be useless in this model.
The trajectory generator can be divided in three functions:

- The first one generates the times associated to each waypoint. It takes the path and the maximum velocity of the drone as inputs. Then it calculates the total path length, the length of the segments and the total time in which the drone has to travel

Algorithm 8: [q_{new}] = steer (q_{rand}, q_{near}, q_{goal}, obstacle)

if *distance between* q_{rand} *and* q_{near} < *tolerance* **then**
  q_{new}=q_{rand};
else
  q_{new}=q_{near}+tolerance*(q_{rand} - q_{near}) / (distance between q_{rand} and q_{near});
q_{new}=q_{new} + K_a*(q_{goal} - q_{new}) / (distance between q_{goal} and q_{new});
for *each obstacle* **do**
  if *distance between* q_{new} *and obstacles* < *radius of influence* **then**
    q_{new}=q_{new} - K_r*(pos_{obs} - q_{new}) / (distance between q_{goal} and q_{new});
end

Algorithm 9: [nc] = nocollision (q_{in}, q_{fin}, obstacles)

for *each obstacle* **do**
  C=linspace(q_{in}, q_{fin});
  if *a point of C is in the obstacle* **then**
    $nc = 0$;
  else
    $nc = 1$;

the path. With these three data it can calculate the istants in which the drone must cross the waypoints.

- The second one generates the coefficients of the polynomial for each waypoint solving 4*m linear equation, where m is the number of subpaths. It takes as inputs the three outputs of the previous function and uses 4*m costraints. The first 2*(m-1) are related to the position and require the end of a segment to be the start of the following one. The successive m-1 are related to the velocity and the last m-1 are related to the acceleration. So, four constraints remain and they are the initial and the final positions and their velocities that must be null. So, the coefficients can be obtained.

- The last takes as inputs the total time, the simulation time, the waypoints time and the coefficients obtained with the previous function. It returns as outputs the position, the velocity and the acceleration in each step of the simulation which are used by the controller as reference. It finds the biggest waypoint time smaller than the time simulation and evaluates the polynomial with the correct coefficients.

These functions are taken from [cartella github quadrotor]. The pseudocodes are shown in algorithm 11, algorithm 12 on the following page and algorithm 4.1 on page 65.
The first two functions are implemented within the block of the path planning. This is

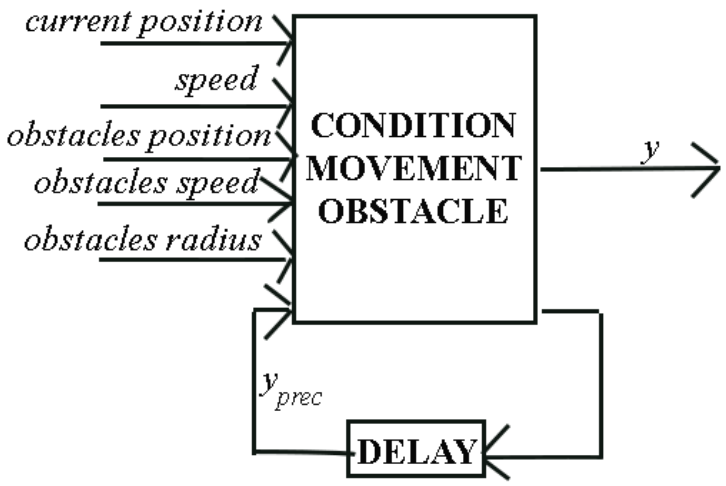


Figure 3.12: Condition obstacle movement block

Algorithm 10: $[y, y_{prec}]$ = condition obs mov (input)

initialization of y as false;
for *each obstacle* **do**
  if *obstacle speed* $\neq 0$ **then**
    v_{do}=relative velocity component from the drone to the dynamic obstacle;
  if $v_{do} < 0$ **then**
    $y_{prec} = 0$;
  ρ_O =distance between drone and obstacle;
  if ρ_O<*(radius obstacle+tollerance)* $\wedge$ $v_{do} > 0 \wedge y_{prec} = 0$ **then**
    y=true;
    $y_{prec} = 1$;
end

because the coefficients X as the path must be calculated once until the maneuver is over. In algorithm 11, to allow to use these functions for the differents maneuvers which start at different moments of the simulation, the simulation time is added to the vector ts. Then the last function is implemented in its own block because it must return the different references in each step (figure 3.13 on the next page). Also in algorithm 4.1 on page 65, for the same reason of the previous function, t_{prec}, which is equal to the previous simulation time, is added to total time in the comparison with t. Because the enable block stops to run when is disable and returns the last outputs calculated, t_prec is equal to the simulation time in which the path is calculated.

Algorithm 11: $[ts, \text{total time}, t_{prec}]$ = generate ts (path, maximum velocity, simulation time)

calculation the total path lenght;
total time = total path length/maximum velocity;
calculation length of single segments;
$ts(1) = 0$;
$ts(2 : end)$ = cumulative sum of lenght of single segments;
normalization of ts with $ts(end)$;
$ts = ts * total\ time + simulation\ time$;
t_{prec}=simulation time;

Algorithm 12: [X] = generate X (path, ts)

```
[m+1, n] = size(path);
initialization of the variables;
for i = 1 : n do
    constraint counter: idx = 1;
    for k = 1:(m-1) do
        A(idx, 4*(k-1)+1 : 4*k, i) = [ts(k+1)^3, ts(k+1)^2, ts(k+1), 1];
        Y(idx, i) = path(k+1, i);
        idx = idx + 1;
        A(idx, 4*(k)+1 : 4*(k+1), i) = [ts(k+1)^3, ts(k+1)^2, ts(k+1), 1];
        Y(idx, i) = path(k+1, i);
        idx = idx + 1;
    Analog procedure for constraints related to velocity and acceleration and for
     last four constraints;
    A(:, :, i) = A(:, :, i) + regularization to prevent singularity;
    X(:, i) = A(:, :, i)\Y(:, i);
```

Algorithm 13: [pos, vel, acc] = reference in each step (X, t, ts, total time, t_{prec})

```
if t ≥ total time+t_prec then
    pos=goal;
    vel=[0; 0; 0];
    acc= [0; 0; 0];
else
    k = find(ts ≤ t);
    take the last element of k;
    pos = [t^7, t^6, t^5, t^4, t^3, t^2, t, 1] * X(8*(k-1)+1 : 8*k, :);
    vel = [7*t^6, 6*t^5, 5*t^4, 4*t^3, 3*t^2, 2*t, 1, 0] * X(8*(k-1)+1 : 8*k, :);
    acc = [42*t^5, 30*t^4, 20*t^3, 12*t^2, 6*t, 2, 0, 0] * X(8*(k-1)+1 : 8*k, :);
```

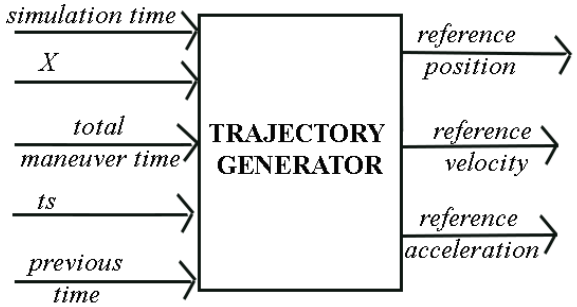


Figure 3.13: Trajectory generator block

Chapter 4

Simulation Results

In this chapter, the first environment model used is described and the simulation results are exposed and compared. Then the same is done for the second. Finally, the results of each environment are compared. The first environment model, totally implemented on Simulink, is taken from [**6**]. The characteristic parameters of the drone have been updated because the model was designed for a different drone which was bigger. Some improvements in the controller block have been made and will be explained in this chapter. The second environment model is taken from [**28**]. It is partially implemented on Simulink and partially on Unity. The overall model scheme is the same for each environment (figure 4.1, the blocks are explained in the following sections.

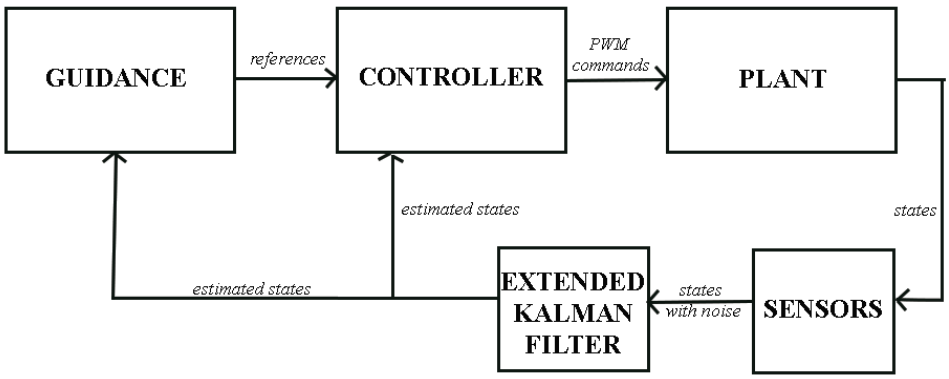


Figure 4.1: Simulation model

4.1 Simulations on Simulink

The Simulink model is composed by five principal blocks: the trajectory planner, the controller, the plant, the sensors and the extended Kalman filter used as estimator. All the blocks are quickly described:

Trajectory planner

The trajectory planners were described in the previous chapter, the constant parameters

algorithm version	parameter	value
Both	r_O	0.9 $[m]$
	r_{Ov}	0.3 $[m]$
	$toll$	0.25 $[m]$
	$goal$	$[0\ 0\ -2.5; 4\ 4.5\ -2.5; 4\ 4.5\ 0]$ $[m]$
APF	K_a	1
	K_r	5
	K_v	5
	K_m	10
	v_{max}	$[0.50.51]$ $[m/s]$
RRT*FNDAPF	x_{max}	5 $[m]$
	y_{max}	5 $[m]$
	z_{max}	5 $[m]$
	EPS	0.01
	$maxiter$	1000
	$maxnodes$	200
	r	0.2
	v_{max}	0.5 $[m/s]$

Table 4.1: constant parameters of trajectory planner

in 4.1 are used in the simulations, r_O and r_{Ov} are the radius of influence and the effective radius of the obstalces, $toll$ is the tolerance for the transition between each goal and EPS is the tolerance for the steering function.

Controller

The structure of the controllers is a cascade controller with an inner loop and an outer loop. The inner loop is related to the fast dynamics and controls the attitude and the altitude. The outer loop is related to the slow dynamics and controls the position on the NE plane. (figure 4.2 on the following page) Two different controllers are used, a PID controller and a Sliding Mode Controller (SMC). This allows to compare the results of the different simulations and to see the differences between them.

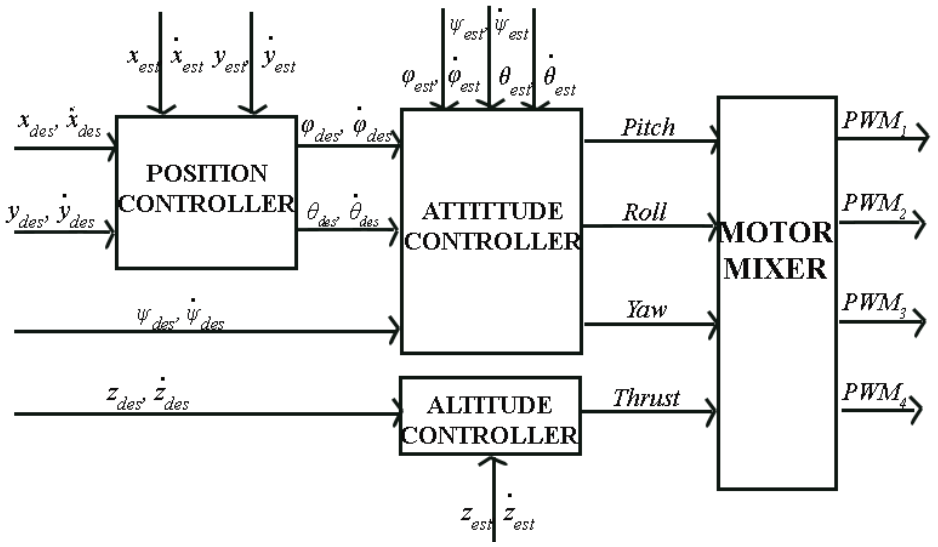


Figure 4.2: Controller architecture

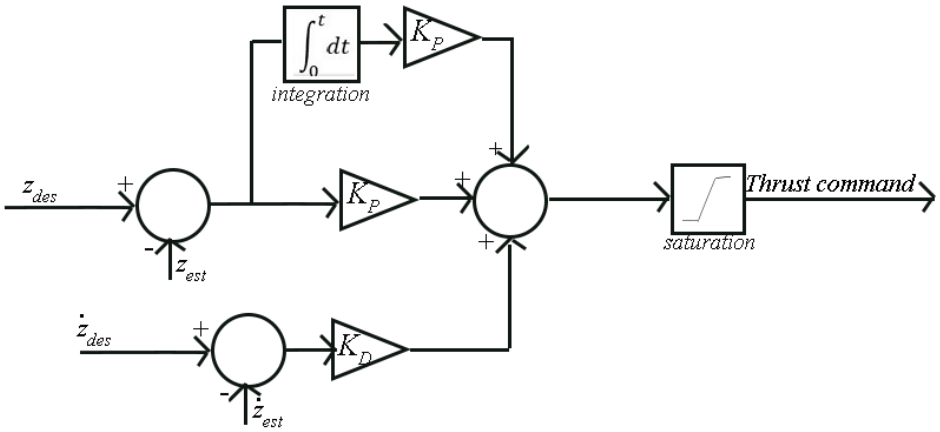


Figure 4.3: Altitude channel of the inner loop PID block

- **PID**: in this controller PIDs are implemented in both inner and outer loop. In the inner loop, classic Single Input-Single Output (SISO) PID controllers are used. Regarding the attitude, the errors on θ, ϕ and ψ are the inputs of each proportional channel. The integral of these errors is the input of each integral channel and p, q and r are the inputs of each derivative channel. The outputs are saturated in order to avoid torque command values too large (figure 4.3). Regarding the altitude, the structure of the PID is the same with the error on h and the error on $\dot{h}$ as inputs and the thrust command as output (figure 4.4 on the following page). To allow a more intuitive tuning, PIDs were designed in order to have the thrust command in

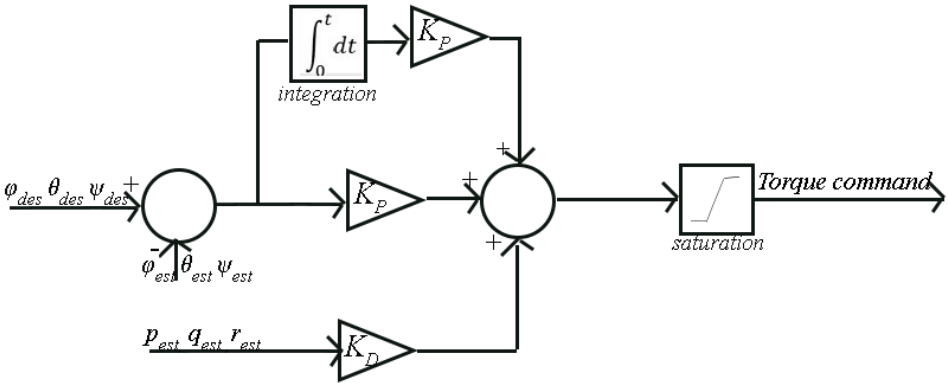


Figure 4.4: Attitude channel of the inner loop PID block

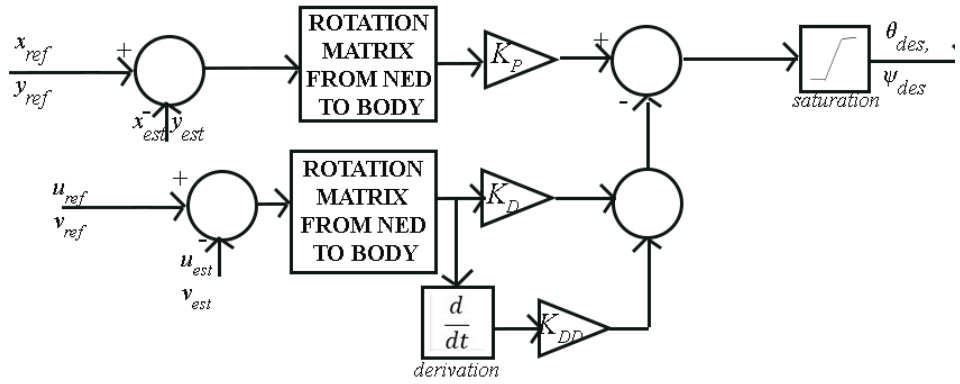


Figure 4.5: Outer loop PDD block

Newton and the torque commands in Newton-meters. In the outer loop, SISO PDDs controller are used. The PDD is a PD controller with a derivative controller on the velocity error that reduces the oscillations. The inputs are the x and y errors for the proportional channel and the u and v errors for the derivative channel. The outputs θ_{des} and ϕ_{des} are saturated because of the drone limits (figure 4.5).

- **SMC**: In this controller, the SMC is used only in the inner loop. The outer loop is still implemented with the PDD. $\dot{\theta}_{des}$ and $\dot{\phi}_{des}$ are added as new outputs of the PDDs because SMC requires also these information. Regarding the altitude, the sliding surface is:

$$s = \dot{h}_{des} - \dot{h}_{est} - \lambda_{alt} * (h_{des} - h_{est})$$

and the control law is:

$$F_z = \frac{m}{\cos(\theta) * \cos(\phi)}(g - \lambda_{alt} * (h_{des} - h_{est})) - K * sign(s)$$

where m is the mass of the drone, g is the gravitational acceleration and λ_{alt} and K are positive parameters to be tuned. Regarding the attitude, the sliding surface is

$$\mathbf{s} = \begin{bmatrix} \dot{\phi}_{des} - \dot{\phi}_{est} \\ \dot{\theta}_{des} - \dot{\theta}_{est} \\ \dot{\psi}_{des} - \dot{\psi}_{est} \end{bmatrix} + \begin{bmatrix} \lambda_{roll} & 0 & 0 \\ 0 & \lambda_{pitch} & 0 \\ 0 & 0 & \lambda_{yaw} \end{bmatrix} * \tilde{\mathbf{q}} \tag{4.1}$$

and the control law is:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = - \begin{bmatrix} (J_y - J_z) * \dot{\psi}_{est} * \dot{\theta}_{est} \\ (J_z - J_x) * \dot{\psi}_{est} * \dot{\phi}_{est} \\ (J_x - J_y) * \dot{\theta}_{est} * \dot{\phi}_{est} \end{bmatrix} + \mathbf{\Lambda} \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} * \dot{\mathbf{q}} - \mathbf{K} * sign(s) \tag{4.2}$$

with

$$\mathbf{K} = \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{bmatrix} \qquad \mathbf{\Lambda} = \begin{bmatrix} \lambda_{roll} & 0 & 0 \\ 0 & \lambda_{pitch} & 0 \\ 0 & 0 & \lambda_{yaw} \end{bmatrix}$$

where $\mathbf{K}$ and $\mathbf{\Lambda}$ are parameters matrices to be tuned and $\dot{\mathbf{q}}$ is the quaternion time-derivative

The inner loop commands are used as inputs for the motor mixer, that allows the controller to communicate with the ESCs. The Motor mixer generates the Power-Width Modulation (PWM) Duty Cycle (DC) with the following equations:

$$\upsilon_1 = (\frac{-R + P - Y}{2} * T + T) * 1000 + idle \tag{4.3}$$

$$\upsilon_2 = (\frac{R - P - Y}{2} * T + T) * 1000 + idle \tag{4.4}$$

$$\upsilon_3 = (\frac{R + P + Y}{2} * T + T) * 1000 + idle \tag{4.5}$$

$$\upsilon_4 = (\frac{-R - P + Y}{2} * T + T) * 1000 + idle \tag{4.6}$$

$$\tag{4.7}$$

where υ_i is the DC of the i-th motor and ranges between 1000 and 2000 and $idle$ is the DC set when armed ($idle = 1000$). R, P and Y are the required roll, pitch and yaw and T is the required thrust.

It is preferred that the drone flies with the nose pointing to the goal, so with a yaw angle different from zero. To make it possible, a rotation matrix from the NED reference frame to the body reference frame is needed:

$$\mathbf{R}^{body}_{NED}(1,:) = \begin{bmatrix} \cos(\theta) * \cos(\psi) \\ \sin(\phi) * \sin(\theta) * \cos(\psi) - \cos(\phi) * \sin(\psi) \\ \cos(\phi) * \sin(\theta) * \cos(\psi) + \sin(\phi) * \sin(\psi) \end{bmatrix} \tag{4.8}$$

$$\mathbf{R}_{NED}^{body}(2,:) = \begin{bmatrix} \cos(\theta) * \sin(\psi) \\ \sin(\phi) * \sin(\theta) * \sin(\psi) + \cos(\phi) * \cos(\psi) \\ \cos(\phi) * \sin(\theta) * \sin(\psi) + \cos(\phi) * \cos(\psi) \end{bmatrix} \tag{4.9}$$

$$\mathbf{R}_{NED}^{body}(3,:) = \begin{bmatrix} -\sin(\theta) \\ \sin(\phi) * \cos(\theta) \\ \cos(\phi) * \cos(\theta) \end{bmatrix} \tag{4.10}$$

So the error on the position and the error on the velocity in the body reference frame are:

$$\begin{bmatrix} x_{b,des} - x_{b,est} \\ y_{b,des} - y_{b,est} \\ z_{b,des} - z_{b,est} \end{bmatrix} = \mathbf{R}_{NED}^{body} * \begin{bmatrix} x_{NED,des} - x_{NED,est} \\ y_{NED,des} - y_{NED,est} \\ z_{NED,des} - z_{NED,est} \end{bmatrix} \tag{4.11}$$

$$\begin{bmatrix} u_{b,des} - u_{b,est} \\ v_{b,des} - v_{b,est} \\ w_{b,des} - w_{b,est} \end{bmatrix} = \mathbf{R}_{NED}^{body} * \begin{bmatrix} u_{NED,des} - u_{NED,est} \\ v_{NED,des} - v_{NED,est} \\ w_{NED,des} - w_{NED,est} \end{bmatrix} \tag{4.12}$$

The parameters in the left table of table 4.2 are used to tune the PID controller. The parameters in the right table of table 4.2 are used to tune the SMC.

Plant

In this block, the actuators and the mathematical model of the drone are implemented (figure 4.6 on page 71) The mathematical model is the one described in (2.17). The actuators block is designed thanks to the thrust tests described in chapter 2.3.1. A brushless motor and its propellers were mounted on a RCBenchmark Series 1520 that is a thrust test stand. Then, the motor is connected through an ESC to the battery and with a USB cable to the PC. The PWM signal is varied and thrust data are collected with a script of the software of RCBenchmark. After collecting the data, a second order polynomial equation is extrapolated. This allows to obtain the relationships between the PWM signal and the force and between the PWM signal and the moment generated by the spinning of the propeller. The PWM signal is produced by the motor mixer and provided to the motor through an ESC. The thrust curve is:

$$T(x) = (2.382 * 10^{-6})x^2 - (3.628 * 10^{-3})x + 1.177 \tag{4.13}$$

and the moment curve is:

$$M(x) = (1.045 * 10^{-8})x^2 - (5.332 * 10^{-3})x - 5.277 * 10^{-3} \tag{4.14}$$

Channel	Parameter	Value
	Inner Loop	
Pitch	K_P	0.032
	K_I	0.0004
	K_D	0.032
	$Saturation$	$\pm$ 0.56 $[Nm]$
Roll	K_P	0.032
	K_I	0.0004
	K_D	0.032
	$Saturation$	$\pm$ 0.56 $[Nm]$
Yaw	K_P	0.15
	K_I	0.0001
	K_D	0.15
	$Saturation$	$\pm$ 0.1 $[Nm]$
Thrust	K_P	2
	K_I	1.5
	K_D	50
	$Saturation$	$\pm$ 14 $[N]$
	Outer Loop	
Position (north)	K_P	0.003
	K_D	0.3
	K_{DD}	0.3
	$Saturation$	$\pm \frac{\pi}{6}$ $[rad]$
Position (east)	K_P	0.003
	K_D	0.3
	K_{DD}	0.3
	$Saturation$	$\pm \frac{\pi}{6}$ $[rad]$

Channel	Parameter	Value
	Inner Loop	
Pitch	λ_{pitch}	0.1
	k_{pitch}	-0.04
	$Saturation$	$\pm$ 0.56 $[Nm]$
Roll	λ_{roll}	0.1
	k_{roll}	-0.04
	$Saturation$	$\pm$ 0.56 $[Nm]$
Yaw	λ_{yaw}	0.1
	k_{yaw}	-0.05
	$Saturation$	$\pm$ 0.1 $[Nm]$
Thrust	λ_{thrust}	100
	k_{yaw}	2
	$Saturation$	$\pm$ 14 $[N]$
	Outer Loop	
Position (north)	K_P	0.01
	K_D	0.25
	K_{DD}	0.03
	$Saturation$	$\pm \frac{\pi}{6}$ $[rad]$
Position (east)	K_P	0.01
	K_D	0.25
	K_{DD}	0.03
	$Saturation$	$\pm \frac{\pi}{6}$ $[rad]$

Table 4.2: Constant parameters of the PID controller and of the SMC controller

So the input of the plant is:

$$\mathbf{u} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -(T(\upsilon_1) + T(\upsilon_2) + T(\upsilon_3) + T(\upsilon_4)) \\ l * (T(\upsilon_3) + T(\upsilon_2) - T(\upsilon_1) - T(\upsilon_4)) \\ l * (T(\upsilon_1) + T(\upsilon_3) - T(\upsilon_2) - T(\upsilon_4)) \\ M(\upsilon_3) + M(\upsilon_4) - M(\upsilon_1) - M(\upsilon_2) \end{bmatrix} \tag{4.15}$$

where l is the distance between the center of the propeller and the center of the body reference frame and υ_i is the throttle of the i-th motor.

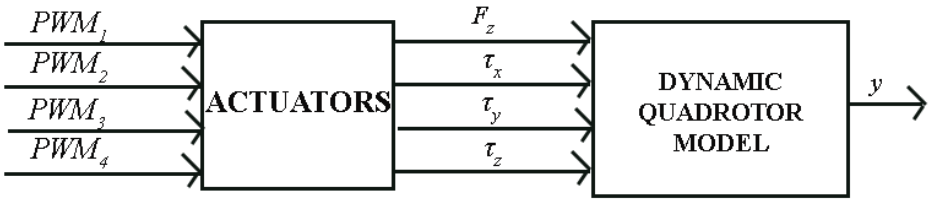


Figure 4.6: Plant block

Sensors

In this block the sensors are modeled. It takes the output of the plant as input and, after using a scalar factor and a misalignment one on the input, it adds a bias and a Gaussian noise. So the noisy output is:

$$\mathbf{z} = \begin{bmatrix} \mathbf{M} * \begin{bmatrix} rv - qw - g\sin(\theta) \\ -ru + pw + g\sin(\phi)\cos(\theta) \\ qu - pv + g\cos(\phi)cos(\theta) + \frac{F_z}{m} \end{bmatrix} * \mathbf{S} + \mathbf{\Delta} + \mathbf{v}_1 \\ \mathbf{M} * \begin{bmatrix} p \\ q \\ r \end{bmatrix} * \mathbf{S} + \mathbf{\Delta} + \mathbf{v}_2 \\ k_B p_0 e^{-\frac{gh}{RT_0}} + \mathbf{\Delta} + \mathbf{v}_3 \\ \mathbf{M} * \begin{bmatrix} p_N \\ p_E \\ h \end{bmatrix} * \mathbf{S} + \mathbf{\Delta} + \mathbf{v}_4 \\ \mathbf{M} * \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} * \mathbf{S} + \mathbf{\Delta} + \mathbf{v}_5 \end{bmatrix} \tag{4.16}$$

where $\mathbf{\Delta}$ is the sensor bias and $\mathbf{v}$ is the sensor noise. $\mathbf{M}$ and $\mathbf{S}$ are the misalignment and scale factor matrices:

$$\mathbf{M} = \mathbf{M}^T = \begin{bmatrix} 1 & M_{xy} & M_{xz} \\ M_{yx} & 1 & M_{yz} \\ M_{zx} & M_{zy} & 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix}$$

Extended Kalman Filter

The Extended Kalman Filter (EKF) is a near optimal and nonlinear observer. It is used

to filter the noise of the sensors and to estimate the variables that cannot be releved with the sensors. The EKF equations are:

- Initialization:

$$\hat{x}_{(0|0)} \sim N(\mu_0, \sigma_0)$$
$$P_{(1|0)} = \mathbb{I}_n$$

- Prediction:

$$\hat{x}_{(k|k-1)} = f^{DT}_{(\hat{x}_{(k-1|k-1)}, u_{(k-1)})}$$
$$P_{(k|k-1)} = F_{(k-1)} P_{(k-1|k-1)} F_{k-1}^T + V_1$$

- Correction/Filtering:

$$K_{(k)} = P_{(k|k-1)} H_k^T [H_k P_{(k|k-1)} H_k^T + V_2]^{-1}$$
$$e_{(k)} = z_{(k)} - \hat{y}_{(k|k-1)}$$
$$\hat{x}_{(k|k)} = \hat{x}_{(k|k-1)} + K_k e_k$$
$$P_{(k|k)} = [\mathbb{I}_n - K_{(k)} H_{(k)}] P_{(k|k-1)}$$

where :

$$F_{(k-1)} = \frac{\partial \mathbf{f}^{DT}}{\partial x}\Big|_{\hat{x}_{(k-1|k-1)}} \qquad H_{(k)} = \frac{\partial \mathbf{h}}{\partial x}\Big|_{\hat{x}_{(k|k)}}$$

and V_1 and V_2 are the covariance matrices of the process and sensor noise:

$$V_1 = \begin{bmatrix} 10^{-6} & & & \\ & 10^{-6} & & \\ & & \ddots & \\ & & & 10^{-6} \end{bmatrix}$$

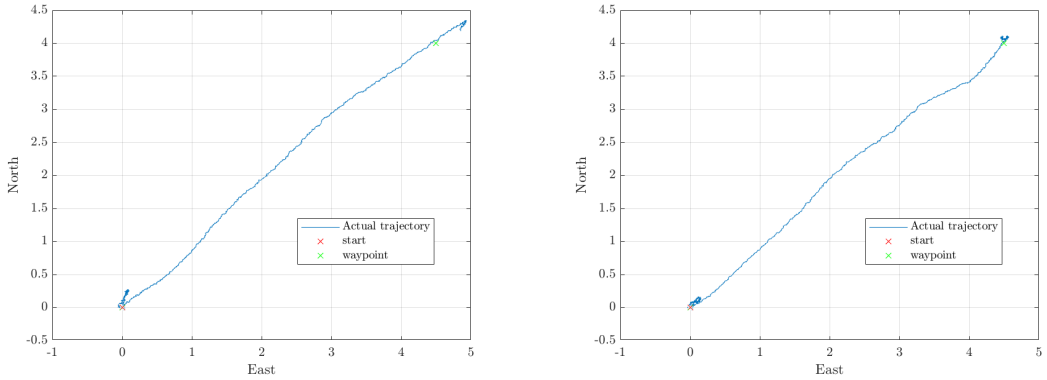
$$V_2 = \begin{bmatrix} diag(\mathbf{v}_1) & & & & \\ & diag(\mathbf{v}_2) & & & \\ & & diag(\mathbf{v}_3) & & \\ & & & diag(\mathbf{v}_4) & \\ & & & & diag(\mathbf{v}_5) \end{bmatrix}$$

For each trajectory planner five simulations are made with each controller. The first one is performed without obstacles to verify if the drone can reach the goal. The second

one is in a generic environment with static obstacles. The third one is in an environment with the GNRON problem. The fourth one is in a environment with a local minimum. The fifth one is in a environment with dynamic obstacles. The reference frame is the NED one. Simulations of APF are explained first, then the one of RRT*FNDAPF. Finally, a comparison between the results of both algorithm is made.

4.1.1 Simulations of APF without obstacles

In this simulation there are no obstacles. Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The results are reported with both controllers to make the comparisons.

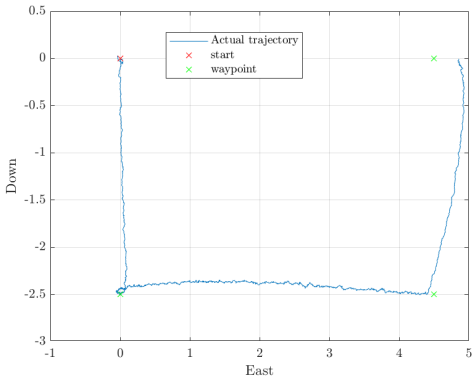


(a) *Simulation with the PID* (b) *Simulation with the SMC*

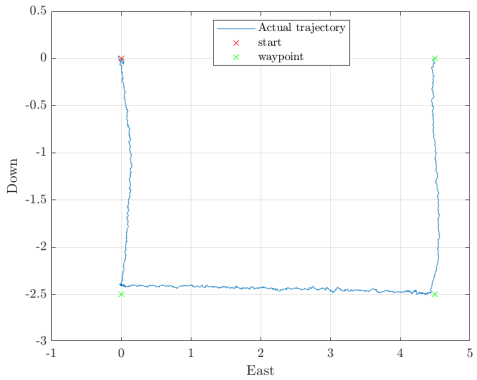
Figure 4.7: Case without obstacles with APF: trajectory in the NE plane

Figure 4.7a and figure 4.7b are the trajectories traced by the drone in the East-North plane. Figure 4.8a on the next page and figure 4.8b on the following page are the ones in the East-Down plane. Figure 4.9a on the next page and figure 4.9b on the following page are the ones in the North-Down plane. These results are resumed in the 3D NED environment in figure 4.10a on page 75 and figure 4.10b on page 75. These plots confirm that the APF allows the drone to reach the goal with a smooth trajectory. The trajectories are similar but there are some differences. Results of simulations made with SMC are better than the other ones. In the take off and landing phases the drone can better keep the position along the North and East axes, thus the landing precision is improved.

Figure 4.11a on page 75 and figure 4.11b on page 75 are the velocities of the drone along the Down axis. Figure 4.13a on page 76 and figure 4.13b on page 76 are the velocities of the drone along the East axis. Figure 4.12a on page 76 and figure 4.12b on page 76 are

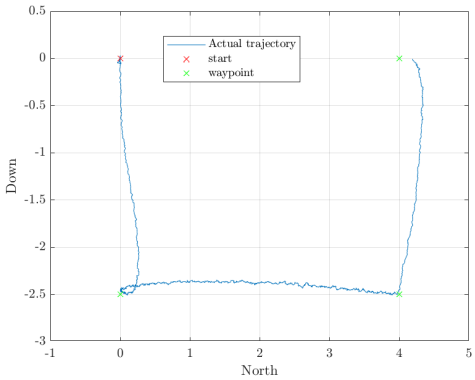


(a) *Simulation with the PID*

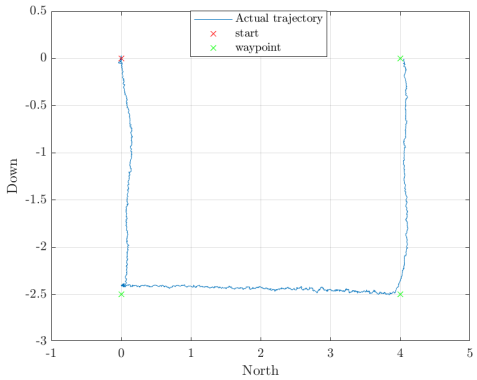


(b) *Simulation with the SMC*

Figure 4.8: Case without obstacles with APF: trajectory in the DE plane



(a) *Simulation with the PID*

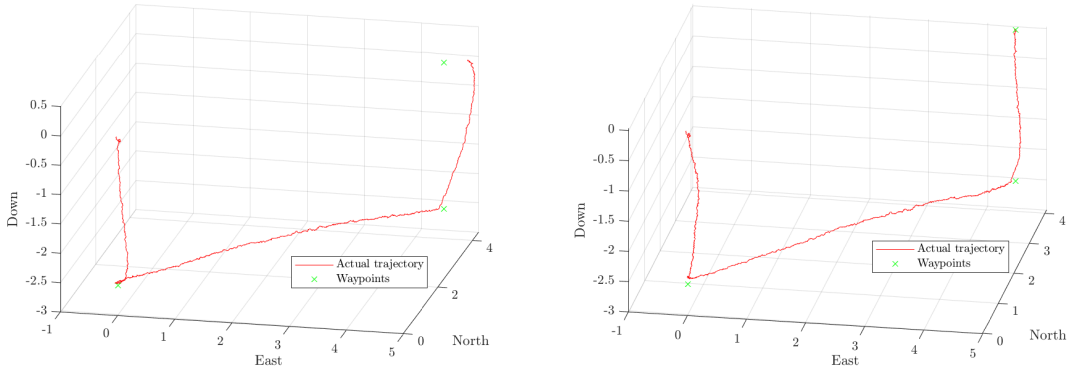


(b) *Simulation with the SMC*

Figure 4.9: Case without obstacles with APF: trajectory in the DN plane

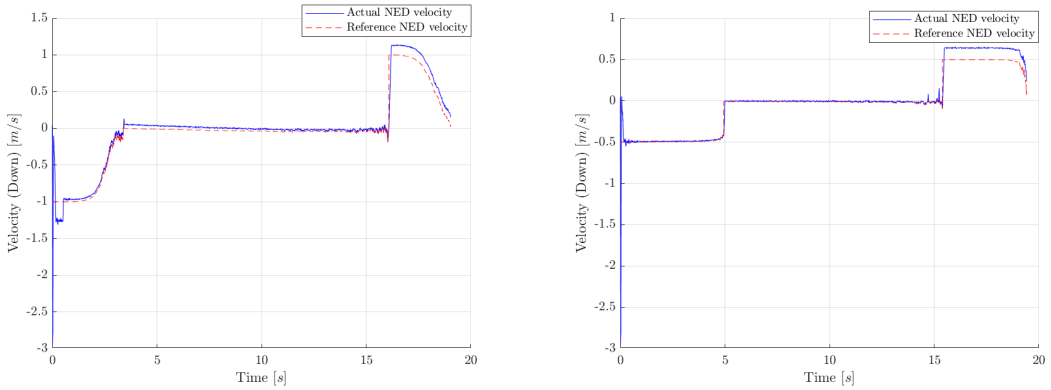
the velocities of the drone along the North axis. Regarding the velocity along the Down axis, the drone correctly follows the reference for both controllers. Along the North and East axes the trend is a little bit worse, the effective velocity has a delay with respect to the reference. The reason is that the drone is under-actuated as previously explained. However, the velocities references are quite regular so the effective velocities have the time to compensate the error.

The attitude can be described with the Euler angles (figure 4.14a on page 77 and figure 4.14b on page 77). The trend is a little bit different between the two controllers. The yaw angle is almost constant because the drone has the nose that points to the goal and the path between them is free. The pitch and the roll angle have a similar trend in both



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.10: Case without obstacles with APF: trajectory in NED

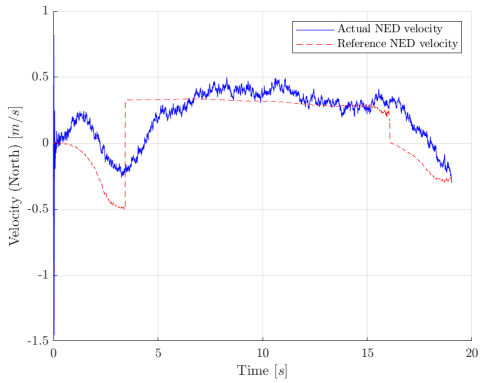


(a) *Simulation with the PID* (b) *Simulation with the SMC*

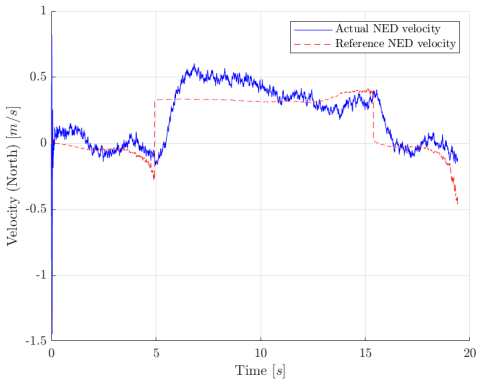
Figure 4.11: Case without obstacles with APF: velocity along D axis

controllers. However, the SMC ones have major changes because it is a more aggressive controller. This allows the drone to reach the goal before.

Figure 4.15a on page 77 and figure 4.15b on page 77 are the thrusts produced by the drone. The two trends are similar as expected from the similar trend of the velocity and the position along the Down axis. However, the one obtained in the simulation with the PID controller is a little bit more dense. Figure 4.16a on page 78 and figure 4.16b on page 78 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is that the SMC is a more aggressive controller than the PID.

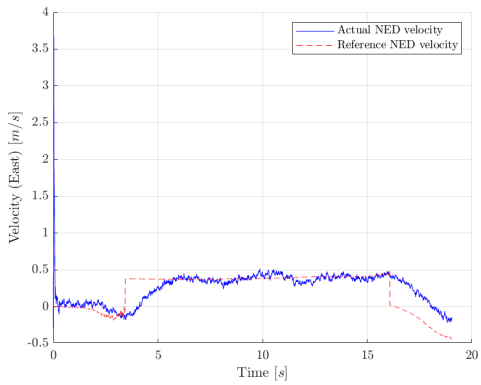


(a) *Simulation with the PID*

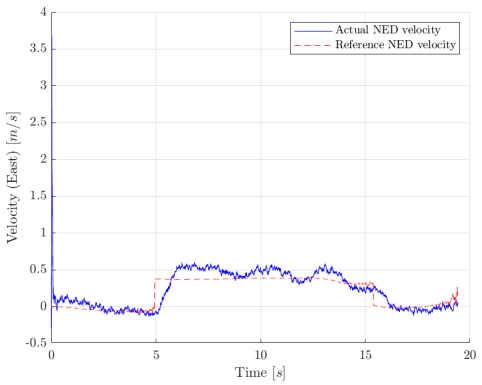


(b) *Simulation with the SMC*

Figure 4.12: Case without obstacles with APF: velocity along N axis



(a) *Simulation with the PID*



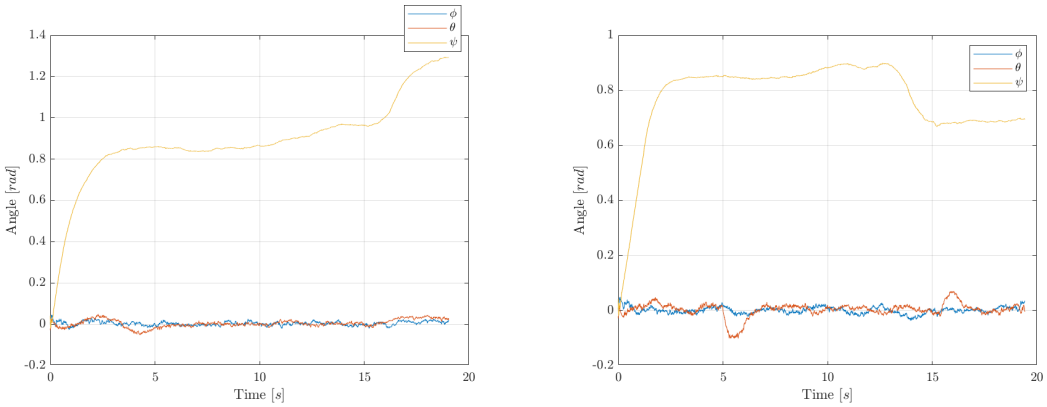
(b) *Simulation with the SMC*

Figure 4.13: Case without obstacles with APF: velocity along E axis

4.1.2 Simulations of APF with static obstacles

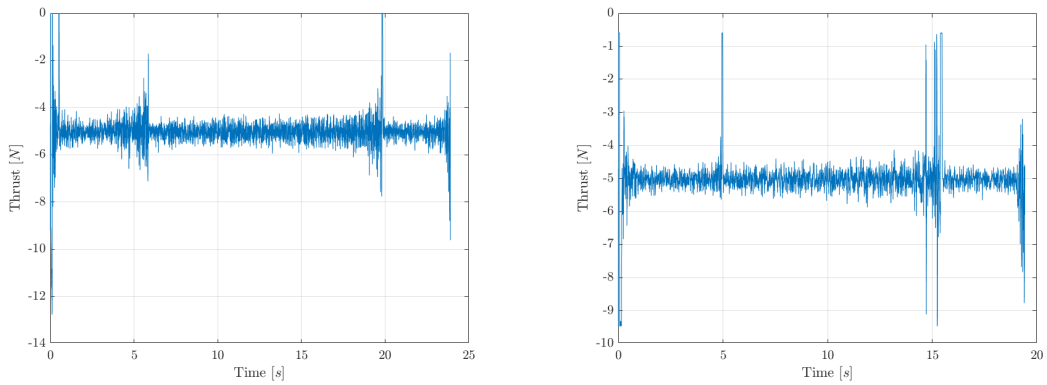
In this simulation there are three static obstacles placed in (1 1 -2.5), (2 2 2-5) and (3 4 -2). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The results are reported with both controllers to make the comparisons.

Figure 4.17a on page 78 and figure 4.17b on page 78 are the trajectories traced by the drone in the East-North plane. Figure 4.18a on page 79 and figure 4.18b on page 79 are the ones in the East-Down plane. Figure 4.19a on page 79 and figure 4.19b on page 79 are the ones in the North-Down plane. These results are resumed in the 3D NED environment



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.14: Case without obstacles with APF: Euler angles

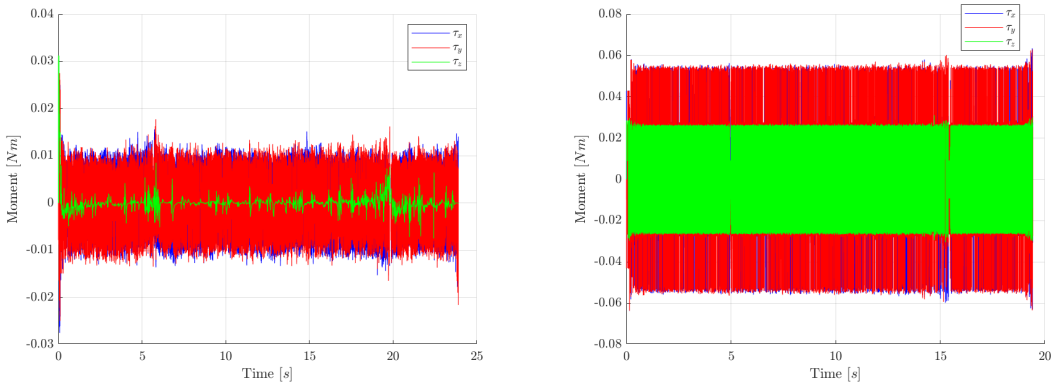


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.15: Case without obstacles with APF: thrust produced

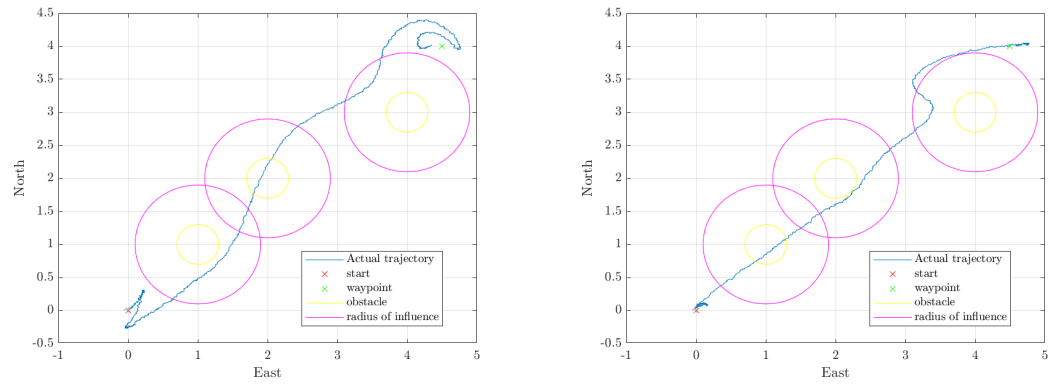
in figure 4.20a on page 80 and figure 4.20b on page 80. These plots confirm that the APF allows the drone to reach the goal avoiding the static obstacles with a smooth trajectory. The trajectories are similar but with the SMC controller it is more accurate: in the take off and landing phases the drone can better keep the position along the North and East axes , thus the landing precision is improved.

Figure 4.21a on page 80 and figure 4.21b on page 80 are the velocities of the drone along the Down axis. Figure 4.23a on page 81 and figure 4.23b on page 81 are the velocities of the drone along the East axis. Figure 4.22a on page 81 and figure 4.22b on page 81 are the velocities of the drone along the North axis. Regarding the velocity along the Down axis, the drone correctly follows the reference for both controllers. Along the axis North



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.16: Case without obstacles with APF: moments produced

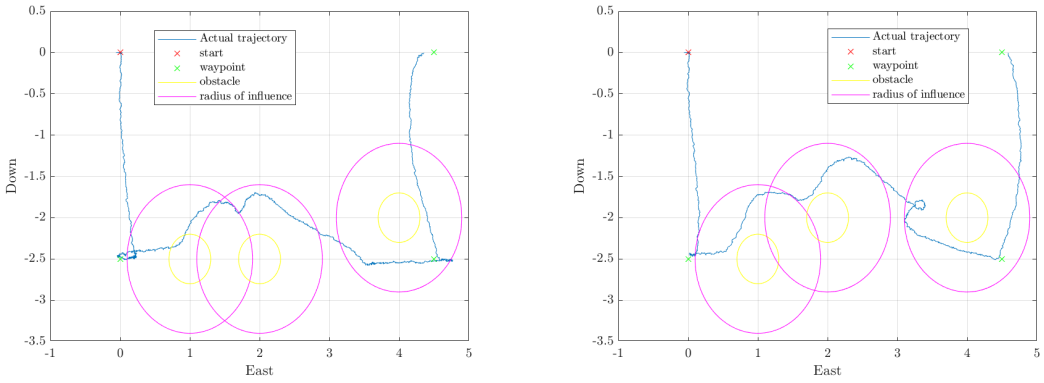


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.17: Case with static obstacles with APF: trajectory in the NE plane

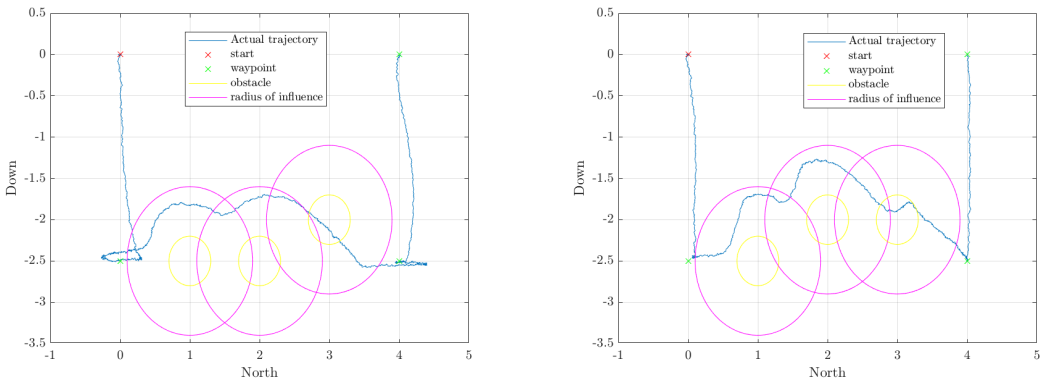
and East the trend is worse, the effective velocity has a delay with respect to the reference. The reason is that the drone is under-actuated as previously explained. Because of the presence of the obstacles, the velocities vary more so the negative effect of the delay is amplified. With the SMC controller this effect is reduced. This implies that the results with the SMC are better than these with the PID.

The attitude can be described with the Euler angles (figure 4.24a on page 82 and figure 4.24b on page 82). The trend is a little bit different between the two controllers. The yaw angle is determined thanks to the position of the drone with respect to the goal, so the difference in the final part of the curve are the conseguence of the different path near the waypoint (4 4.5 -2,5). As in the simulation without obstalces, the pitch and the roll



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.18: Case with static obstacles with APF: trajectory in the DE plane

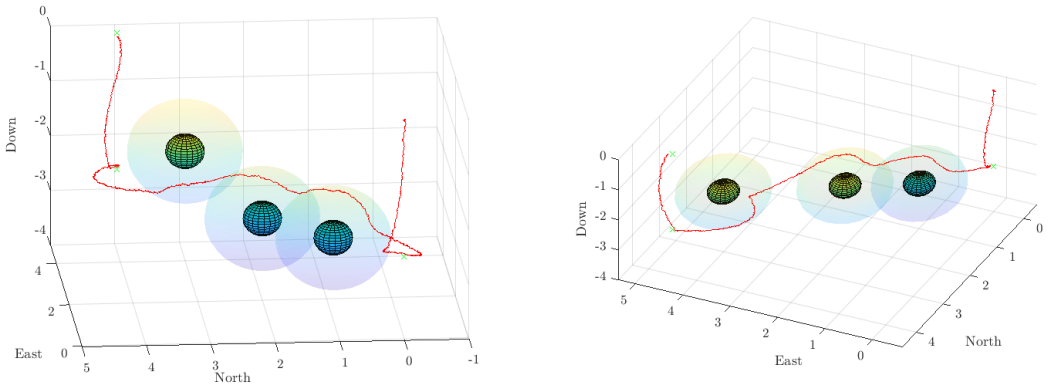


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.19: Case with static obstacles with APF: trajectory in the DN plane

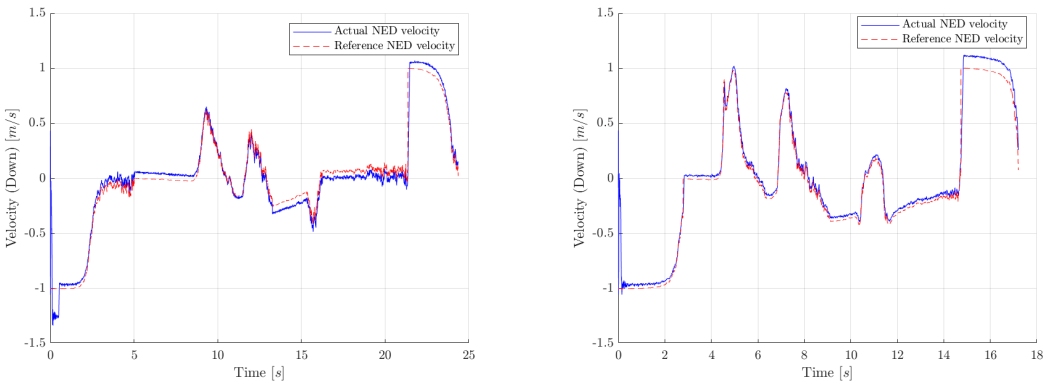
angle have a similar trend in both controllers. However, the SMC ones have major changes because it is a more aggressive controller. This allows the drone to reach the goal before.

Figure 4.25a on page 82 and figure 4.25b on page 82 are the thrusts produced by the drone. The two trends are similar as expected from the similar trend of the velocity and the position along the Down axis. Figure 4.26a on page 83 and figure 4.26b on page 83 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is that the SMC is a controller more aggressive than the PID. The moments aroud the z axis, obtained in the simulation with the PID controller, have in the last part a peak that explains why the yaw angle falls at the same time.



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.20: Case with static obstacles with APF: trajectory in NED



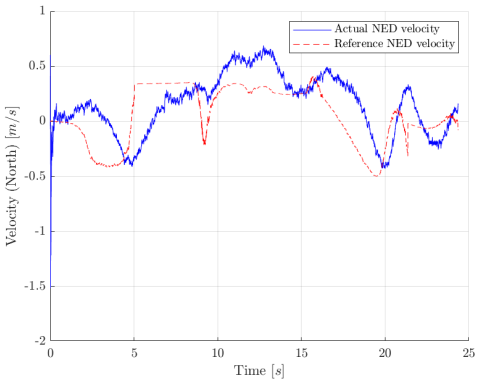
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.21: Case with static obstacles with APF: velocity along D axis

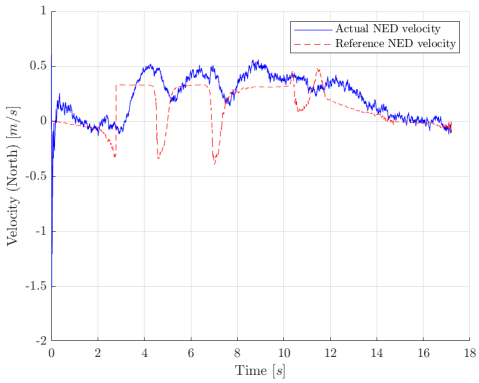
4.1.3 Simulations of APF with GNRON problem

In this simulation a static obstacle is placed in (4.2 4.9 -2.5). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The obstacle is too near to the second waypoint which is in the radius of influence of the obstacle, so the drone cannot reach the waypoint with the classic APF. The results are reported with both controllers to make the comparisons.

Figure 4.28a on page 84 and figure 4.28b on page 84 are the trajectories traced by the

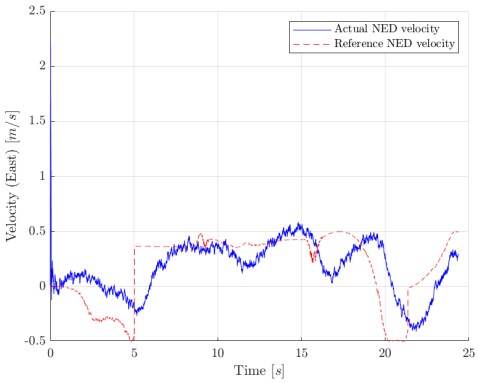


(a) *Simulation with the PID*

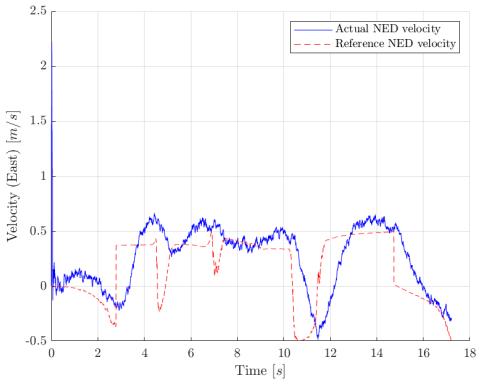


(b) *Simulation with the SMC*

Figure 4.22: Case with static obstacles with APF: velocity along N axis



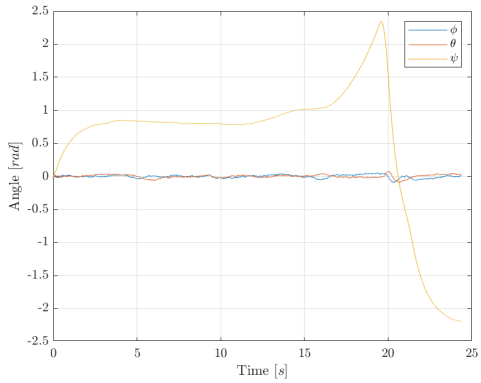
(a) *Simulation with the PID*



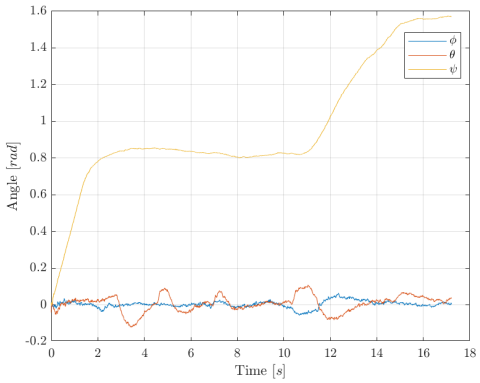
(b) *Simulation with the SMC*

Figure 4.23: Case with static obstacles with APF: velocity along E axis

drone in the East-North plane. Figure 4.27a on page 83 and figure 4.27b on page 83 are the trajectories obtained with the Khatib's version of the APF. Figure 4.30a on page 85 and figure 4.30b on page 85 are the trajectories traced by the drone in the Down-East plane. Figure 4.27a on page 83 and figure 4.27b on page 83 are the trajectories obtained with the Khatib's version of the APF. Figure 4.32a on page 86 and figure 4.32b on page 86 are the trajectories traced by the drone in the Down-North plane. Figure 4.27a on page 83 and figure 4.27b on page 83 are the trajectories obtained with the Khatib's version of the APF. In the simulations where the classic APF is used, the drone cannot reach the goal. However, in the simulations with the improved APF the drone reaches the goal. These results are resumed in the 3D NED environment in figure 4.33a on page 86 and figure 4.33b on page 86. The trajectories are similar but with the PID controller the take off and the

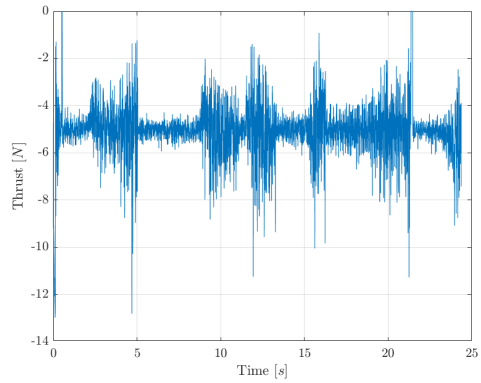


(a) *Simulation with the PID*

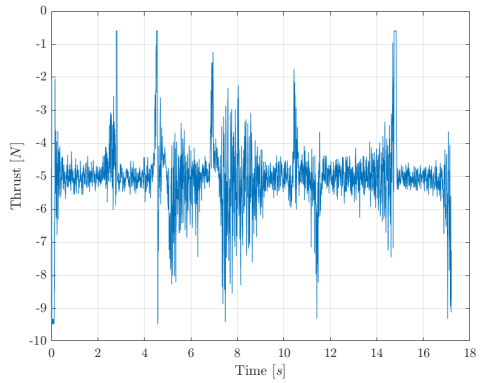


(b) *Simulation with the SMC*

Figure 4.24: Case with static obstacles with APF: Euler angles



(a) *Simulation with the PID*

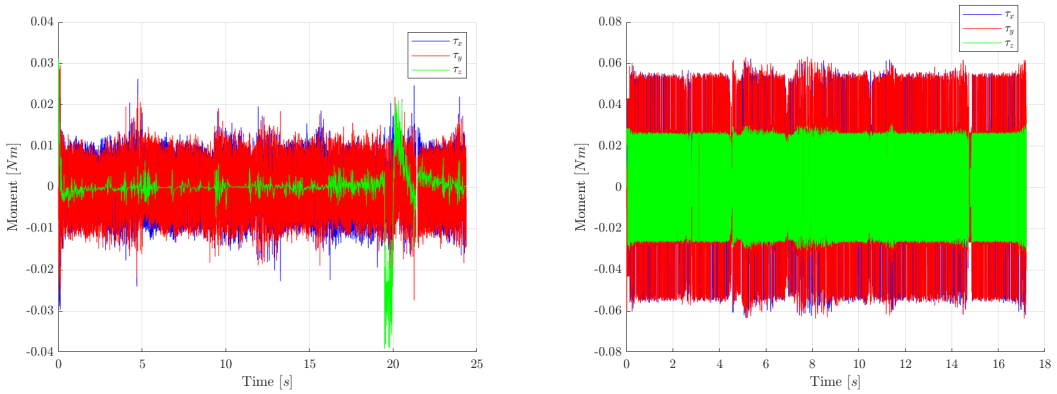


(b) *Simulation with the SMC*

Figure 4.25: Case with static obstacles with APF: thrust produced

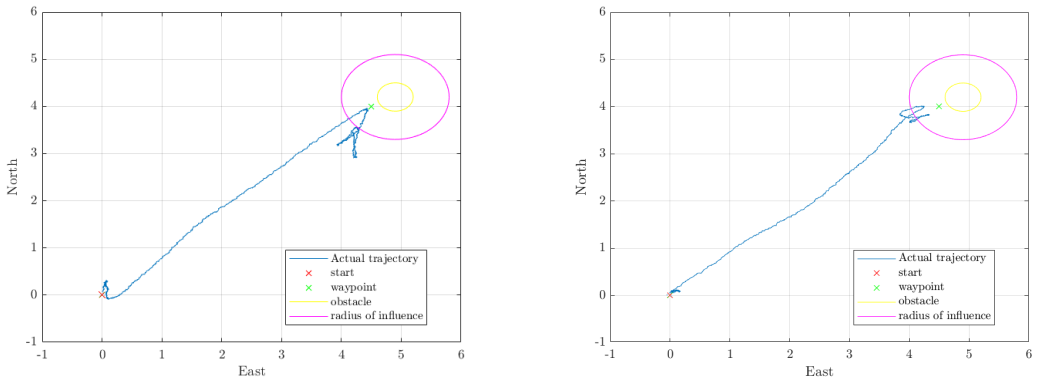
landing are less accurate.

Figure 4.34a on page 87 and figure 4.34b on page 87 are the velocities of the drone along the Down axis. Figure 4.36a on page 88 and figure 4.36b on page 88 are the velocities of the drone along the East axis. Figure 4.35a on page 87 and figure 4.35b on page 87 are the velocities of the drone along the North axis. Regarding the velocity along the Down axis, the drone correctly follows the reference for both controllers. Along the North and East axes the trend is worse, the effective velocity has a delay with respect to the reference. The reason is that the drone is under-actuated as explained in the previous chapters. Near the obstacle, the reference velocities along North and East axes have a negative peak that could disturb the trajectory of the drone. However, it is a variation too fast for the reactivity of



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.26: Case without obstacles with APF: moments produced



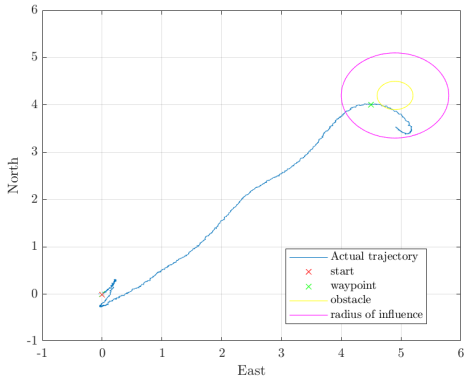
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.27: Case with GNRON problem with Khatib's APF: trajectory in the NE plane

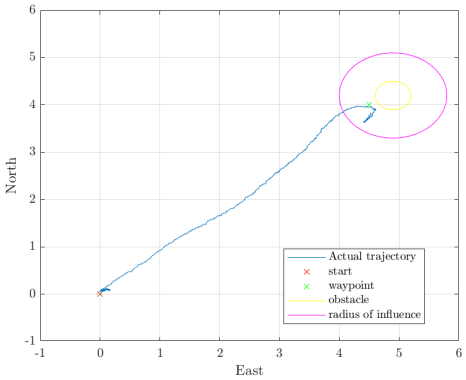
the effective velocities.

The attitude can be described with the Euler angles (figure 4.37a on page 88 and figure 4.37b on page 88). The trend is a little bit different between the two controllers. The yaw angle is almost constant because of the absence of obstacles in the central part of the path. As in the previous simulations, the pitch and the roll angle have a similar trend in both controllers. However, the SMC ones have major changes because it is a more aggressive controller. This allows the drone to reach the goal before.

Figure 4.38a on page 89 and figure 4.38b on page 89 are the thrusts produced by the drone. The two trends are similar as expected from the similar trend of the velocity and

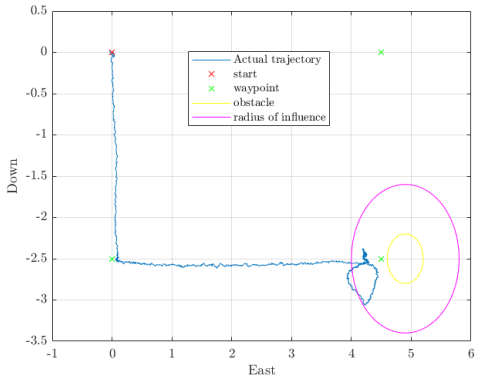


(a) *Simulation with the PID*

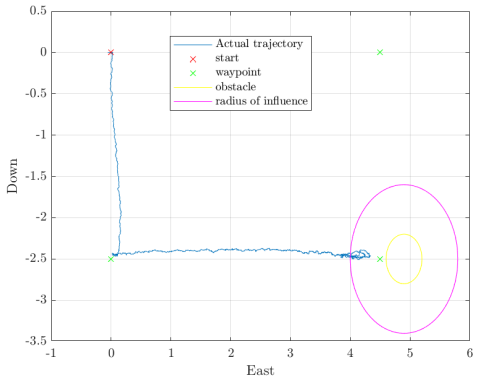


(b) *Simulation with the SMC*

Figure 4.28: Case with GNRON problem with improved APF: trajectory in the NE plane



(a) *Simulation with the PID*



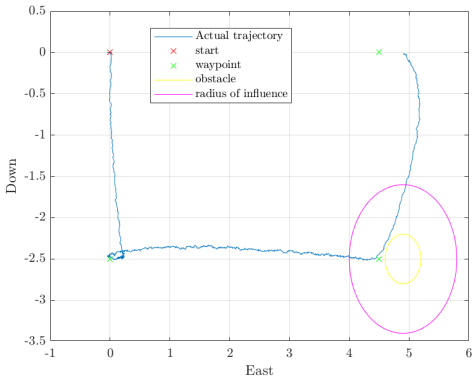
(b) *Simulation with the SMC*

Figure 4.29: Case with GNRON problem with Khatib's APF: trajectory in the DE plane

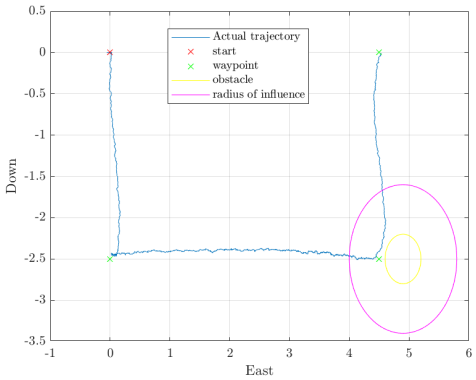
the position along the Down axis. Figure 4.39a on page 89 and figure 4.39b on page 89 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is the SMC is a controller more aggressive than the PID. The peaks in these plots are the consequence of the change of waypoint.

4.1.4 Simulation of APF with a local minimum

In this simulation four obstacles are placed in (1.50 2.25 -2.5), (2.25 1.50 -2.5), (1.875 1.875 -3.1) and (1.875 1.875 -1.9). Starting from the center of the reference frame, the drone has

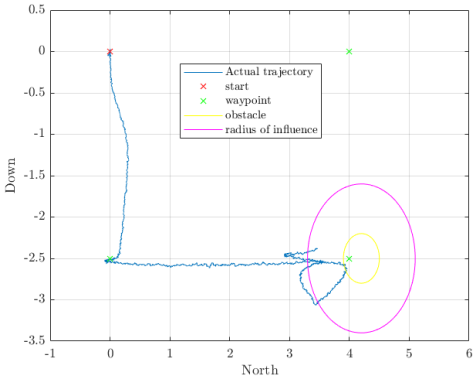


(a) *Simulation with the PID*

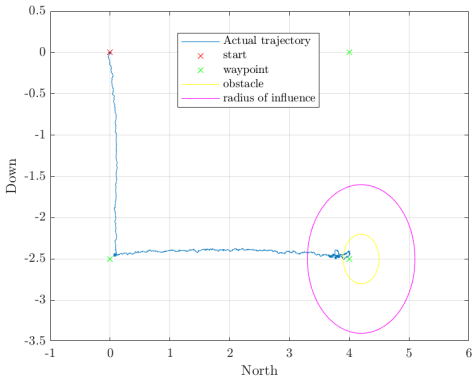


(b) *Simulation with the SMC*

Figure 4.30: Case with GNRON problem with improved APF: trajectory in the DE plane



(a) *Simulation with the PID*

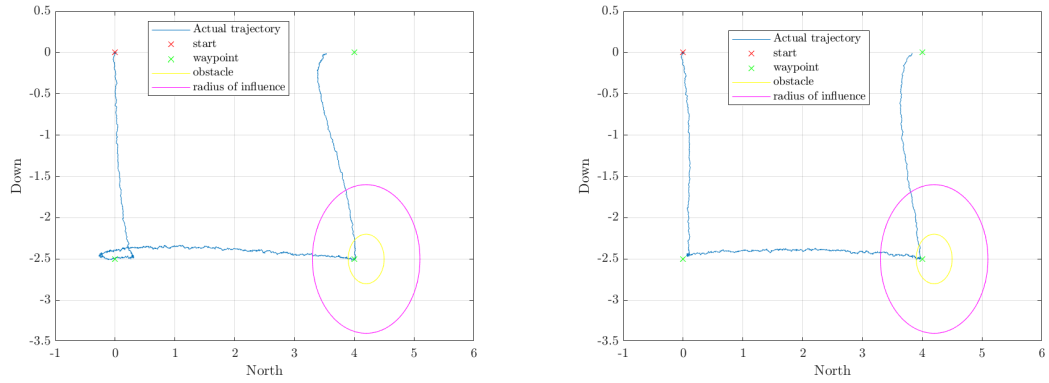


(b) *Simulation with the SMC*

Figure 4.31: Case with GNRON problem with Khatib's APF: trajectory in the DN plane

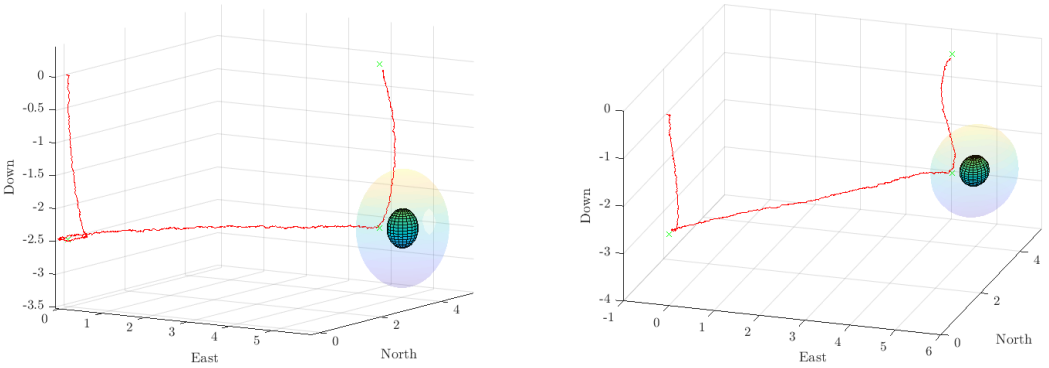
to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). So between these obstacles there is a local minimum zone where the drone with the classic APF falls without being able to escape. The results are reported with both controllers to make the comparisons.

Figure 4.41a on page 90 and figure 4.41b on page 90 are the trajectories traced by the drone in the East-North plane. Figure 4.40a on page 90 and figure 4.40b on page 90 are the trajectories obtained with the Khatib's version of the APF. Figure 4.43a on page 91 and figure 4.43b on page 91 are the trajectories traced by the drone in the Down-East plane. Figure 4.40a on page 90 and figure 4.40b on page 90 are the trajectories obtained with the Khatib's version of the APF. Figure 4.45a on page 92 and figure 4.45b on page 92 are the



(a) *Simulation with the PID* (b) *Simulation with the SMC*

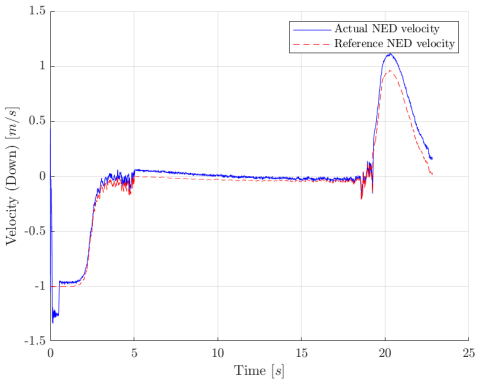
Figure 4.32: Case with GNRON problem with improved APF: trajectory in the DN plane



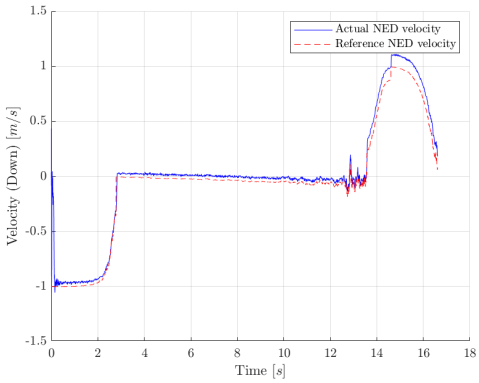
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.33: Case with GNRON problem with improved APF: trajectory in NED

trajectories traced by the drone in the Down-North plane. Figure 4.40a on page 90 and figure 4.40b on page 90 are the trajectories obtained with the Khatib's version of the APF. In the simulations where the classic APF is used, the drone falls in the local minimum zone. However, in the simulations with the improved APF the drone evades and reaches the goal. So, the improvement of the APF is valid. These results are resumed in the 3D NED environment in figure 4.46a on page 93 and figure 4.46b on page 93. The two trajectories are very different from each other. With the PID controller, the drone passes between the obstacles. In the other hand, in the simulation with the SMC controller the drone countours the obstacles. The reason is in how the virtual potential field was implemented. Because the simulation step is $0.004s$, the virtual repulsive force is really big and the difference between SMC and PID is more pronounced.

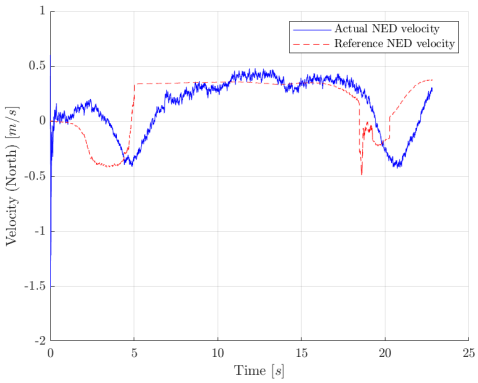


(a) *Simulation with the PID*

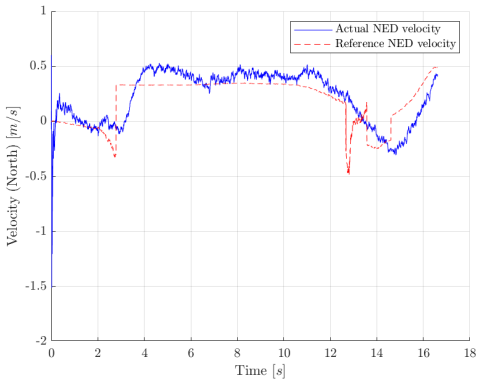


(b) *Simulation with the SMC*

Figure 4.34: Case with GNRON problem with improved APF: velocity along D axis



(a) *Simulation with the PID*

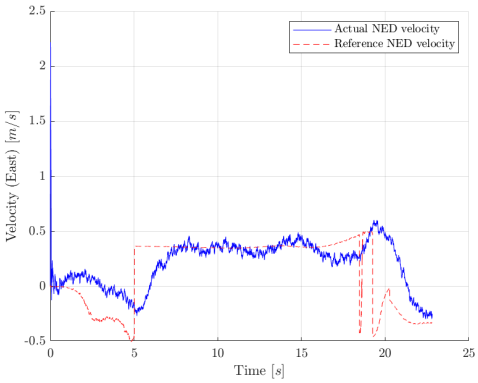


(b) *Simulation with the SMC*

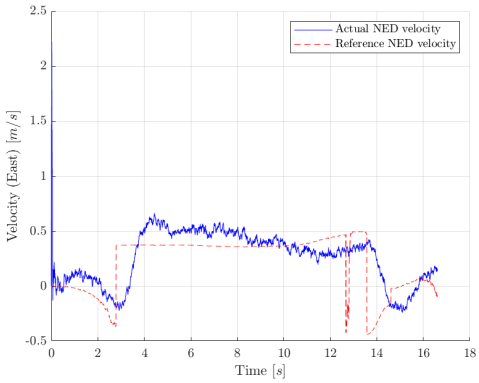
Figure 4.35: Case with GNRON problem with improved APF: velocity along N axis

Figure 4.47a on page 93 and figure 4.47b on page 93 are the velocities of the drone along the Down axis. Figure 4.49a on page 94 and figure 4.49b on page 94 are the velocities of the drone along the East axis. Figure 4.48a on page 94 and figure 4.48b on page 94 are the velocities of the drone along the North axis. The big virtual repulsive force affects also the velocities. In fact, when the drone is in the local minimum, the reference velocities have abrupt changes. Because SMC is more aggressive than PID, it reaches higher velocities. That allow the drone to exit and countour the zone.

The attitude can be described with the Euler angles (figure 4.50a on page 95 and figure 4.50b on page 95). The differences in the path affect the trend of the yaw angle. Unlike the one of the simulation with SMC, the one of the simulation with PID controller is similar

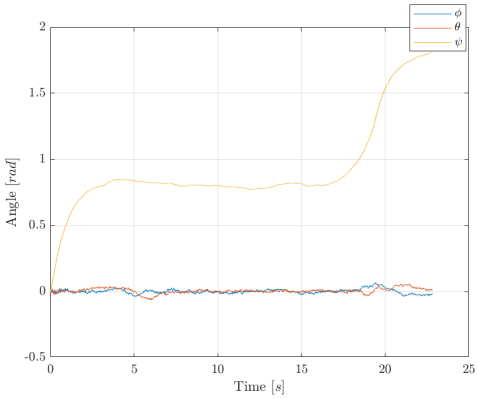


(a) *Simulation with the PID*

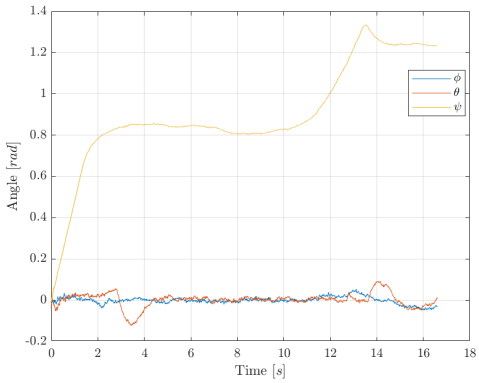


(b) *Simulation with the SMC*

Figure 4.36: Case with GNRON problem with improved APF: velocity along E axis



(a) *Simulation with the PID*

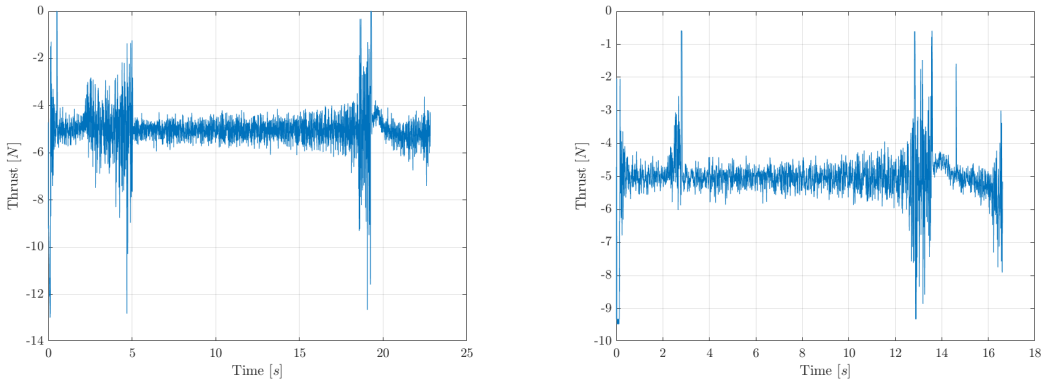


(b) *Simulation with the SMC*

Figure 4.37: Case with GNRON problem with improved APF: Euler angles

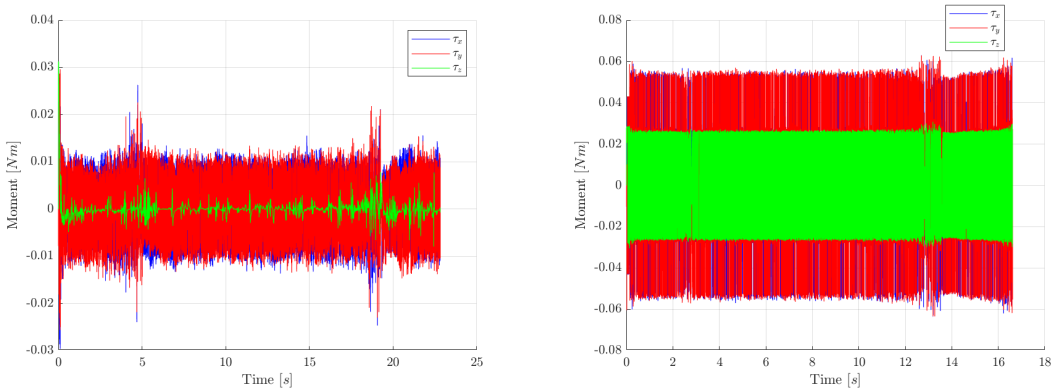
to the trends of the previous simulation. The pitch and the roll angle also have different trends because they are the reason why position on the North-East plane are different. In addition, the SMC ones have major changes because it is a more aggressive controller.

Figure 4.51a on page 95 and figure 4.51b on page 95 are the thrust produced by the drone. Figure 4.52a on page 96 and figure 4.52b on page 96 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is that the SMC is a controller more aggressive than the PID. As expected from the previous plots, the thrust is the maximum producible in the local minimum zone for both the simulations, so the corresponding throttle is at 100%. However, the moments have two different behaviors. For the simulation with PID



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.38: Case with GNRON problem with improved APF: thrust GNRON produced



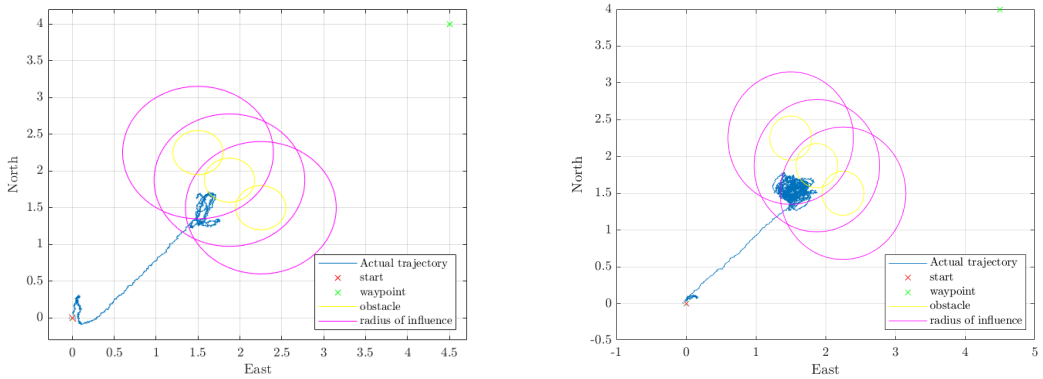
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.39: Case with GNRON problem with improved APF: moments produced

the moments are maximum in the local minimum zone but for the other simulation they are lower than in the rest of the simulation.

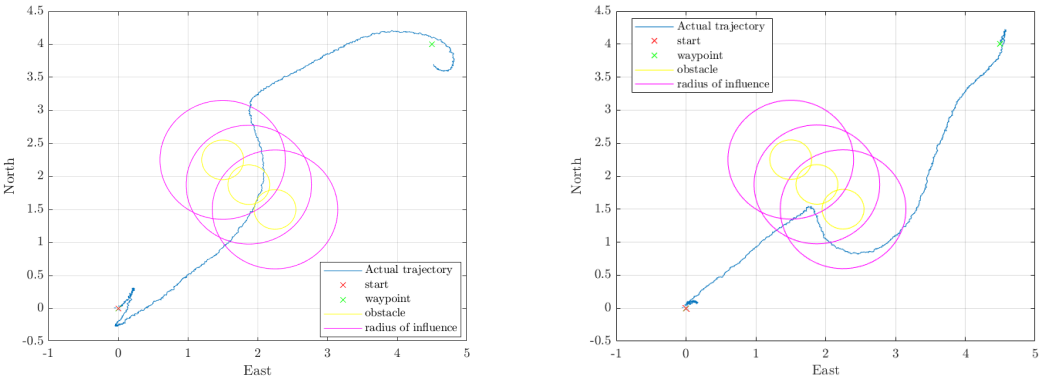
4.1.5 Simulations of APF with dynamic obstacles

Two dynamic obstacles that start from (9 9 -2.5) and (1 1 -5) and have a velocity of (0.5 0.5 0) and (0 0 0.5) are present in this simulation. Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The drone will then meet the obstacle between the waypoints (0 0 -2.5) and (4 4.5 -2.5). The results are reported with both controllers to make the comparisons



(a) *Simulation with the PID* (b) *Simulation with the SMC*

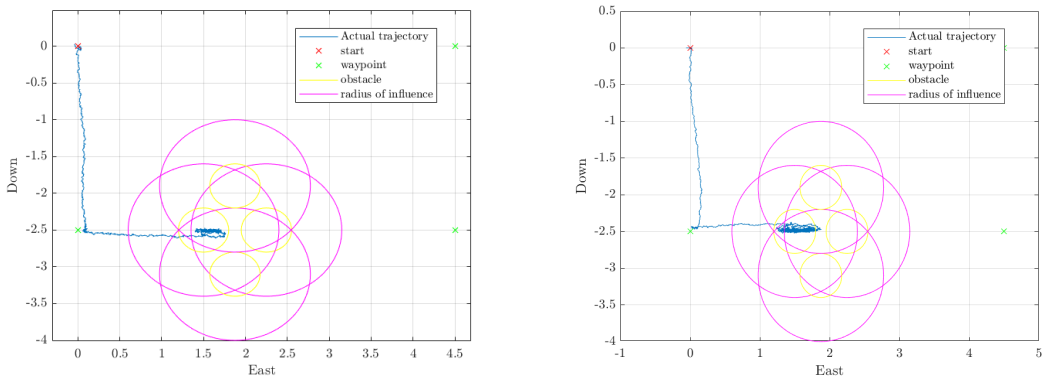
Figure 4.40: Case with local minimum problem with Khatib's APF: trajectory in the NE plane



(a) *Simulation with the PID* (b) *Simulation with the SMC*

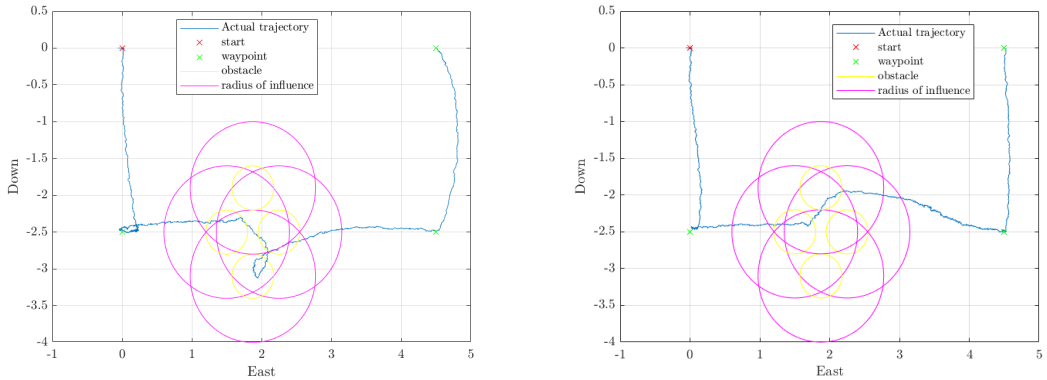
Figure 4.41: Case with local minimum problem with improved APF: trajectory in the NE plane

4.53a on page 96 and 4.53b on page 96 are the trajectories traced by the drone in the East-North plane. 4.54a on page 97 and 4.54b on page 97 are the ones in the East-Down plane. 4.55a on page 97 and 4.55b on page 97 are the ones in the North-Down plane. these results are resumed in the 3D NED environment in 4.56a on page 98 and 4.56b on page 98. The arrows indicate the direction of the motion of the obstacles. SMC controller is more accurate and aggressive with respect to the PID controller. Thus, the way the drone avoids the first obstacle is different in the two simulations. In the simulation with SMC the drone avoids it by reducing the altitude. However, in the simulation with PID the



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.42: Case with local minimum problem with Khatib's APF: trajectory in the DE plane

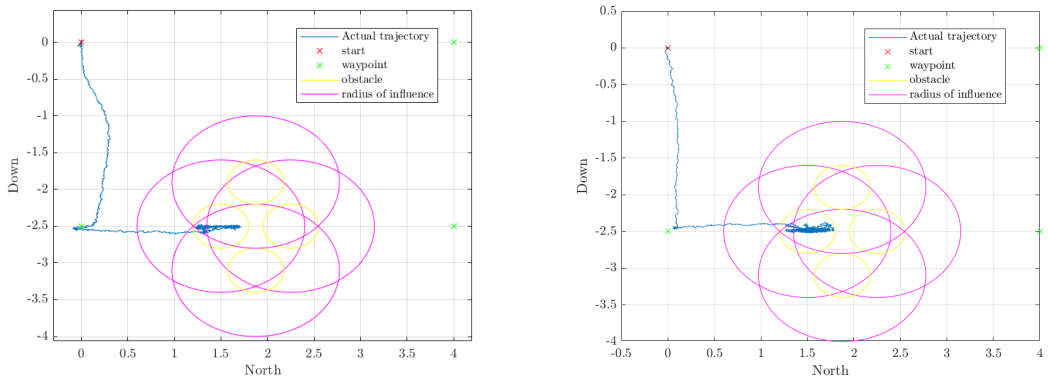


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.43: Case with local minimum problem with improved APF: trajectory in the DE plane

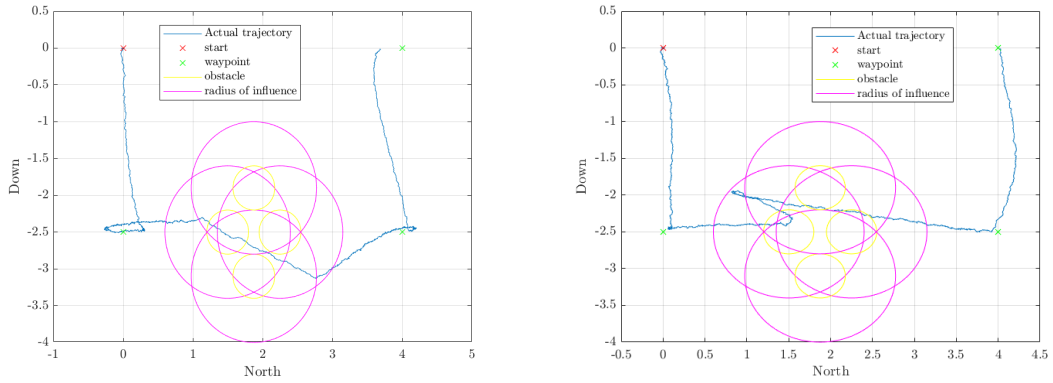
drone encounters the obstacle with a misalignment that implicates it avoids the obstacle by countourning it. These plots confirm that the improvement in the APF allows the drone to reach the goal avoiding the dynamic obstacles with a smooth trajectory.

Figure 4.57a on page 98 and figure 4.57b on page 98 are the velocities of the drone along the Down axis. Figure 4.59a on page 99 and figure 4.59b on page 99 are the velocities of the drone along the East axis. Figure 4.58a on page 99 and figure 4.58b on page 99 are the velocities of the drone along the North axis. Regarding the velocity along the Down axis, the drone correctly follows the reference for both controllers. Along the North and East



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.44: Case with local minimum problem with Khatib's APF: trajectory in the DN plane

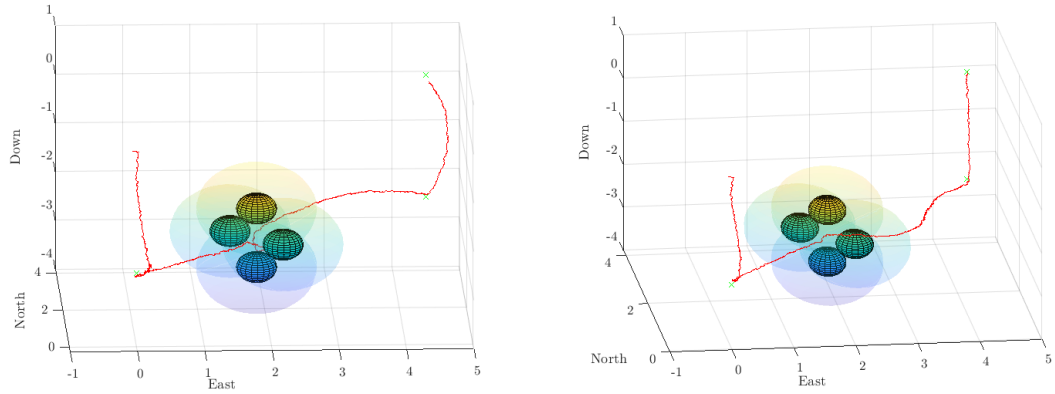


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.45: Case with local minimum problem with improved APF: trajectory in the DN plane

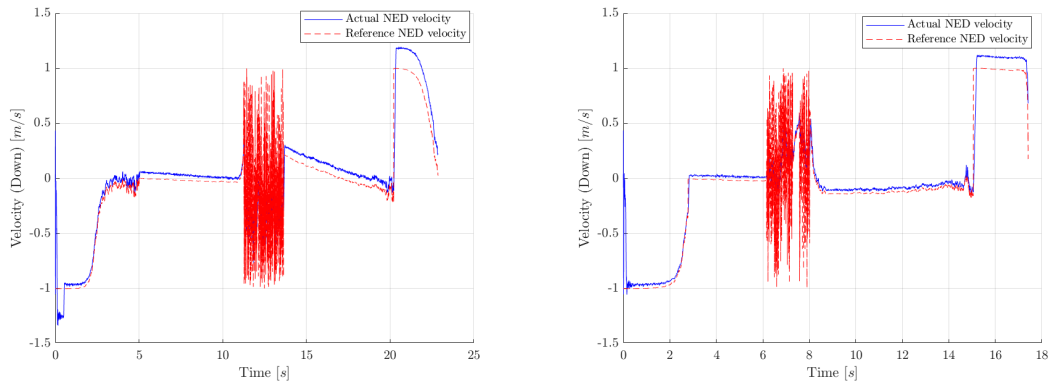
axes the trend is worse, the effective velocity has a delay with respect to the reference. The reason is that the drone is under-actuated as explained in the previous chapters. The different trends between the two simulations is due to the different ways that the drone encounters the first obstacle. The second obstacle causes a variation in the references speed along North and East axes. However, this variation is too short for the delay. Results with the SMC are better than these with the PID.

The attitude can be described with the Euler angles (figure 4.60a on page 100 and figure 4.60b on page 100). The trend is a little bit different between the two controllers but



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.46: Case with local minimum problem with improved APF: trajectory in NED

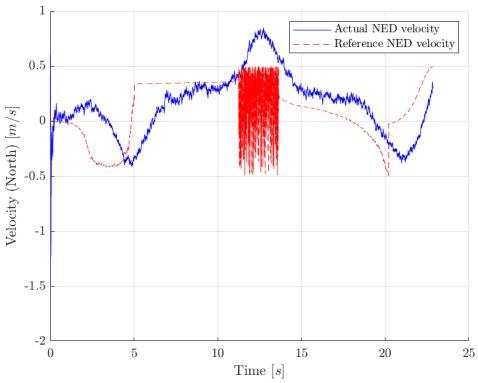


(a) *Simulation with the PID* (b) *Simulation with the SMC*

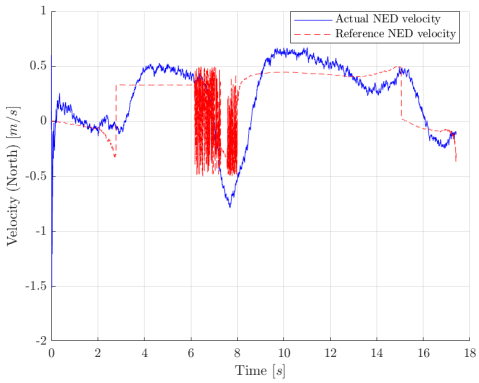
Figure 4.47: Case with local minimum problem with improved APF: velocity along D axis

they are similar to those of the previous simulations. The pitch and the roll angle have a similar trend in both controllers. However, the SMC ones have major changes because it is a more aggressive controller. This allows the drone to reach the goal before.

Figure 4.61a on page 100 and figure 4.61b on page 100 are the thrusts produced by the drone. The two trends are similar as expected from the similar trend of the velocity and the position along the Down axis. Figure 4.62a on page 101 and figure 4.62b on page 101 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is the SMC is a controller more aggressive than the PID.

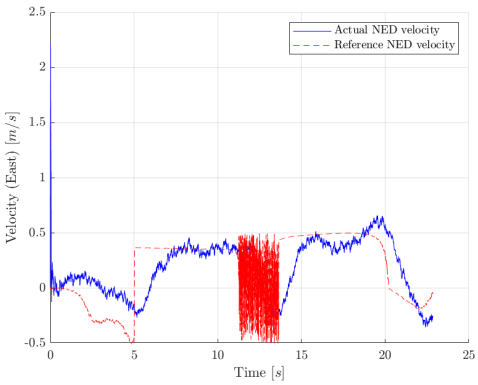


(a) *Simulation with the PID*

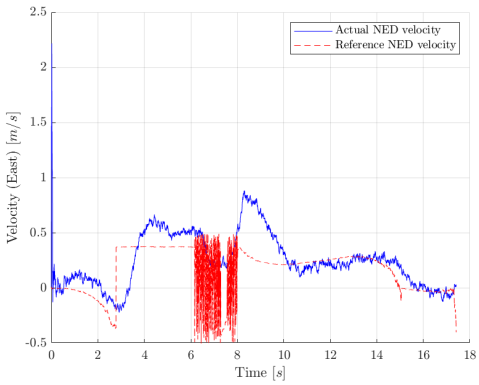


(b) *Simulation with the SMC*

Figure 4.48: Case with local minimum problem with improved APF: velocity along N axis



(a) *Simulation with the PID*



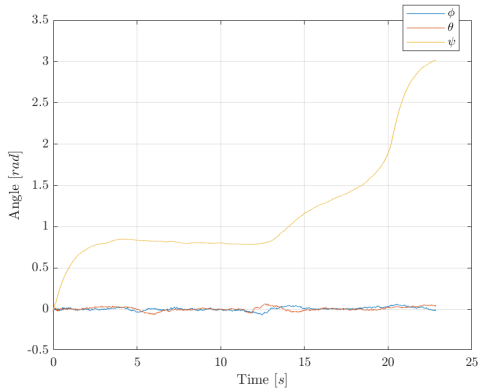
(b) *Simulation with the SMC*

Figure 4.49: Case with local minimum problem with improved APF: velocity along E axis

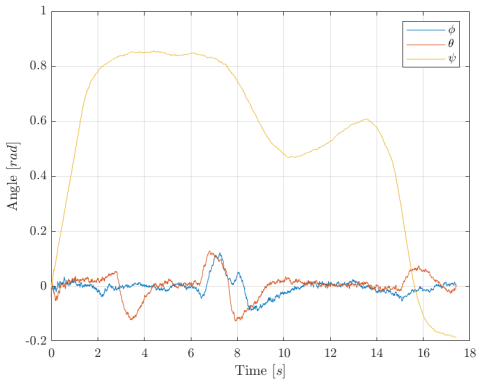
4.1.6 Simulations of RRT*FNDAPF without obstacle

No obstacles are present in this simulation. Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The results are reported with both controllers to make the comparisons

Figure 4.63a on page 101 and figure 4.63b on page 101 are the trajectories traced by the drone in the East-North plane. Figure 4.64a on page 102 and figure 4.64b on page 102 are the ones in the East-Down plane. Figure 4.65a on page 102 and figure 4.65b on page 102 are the ones in the North-Down plane. These results are resumed in the 3D NED environment in figure 4.66a on page 103 and figure 4.66b on page 103. With this path

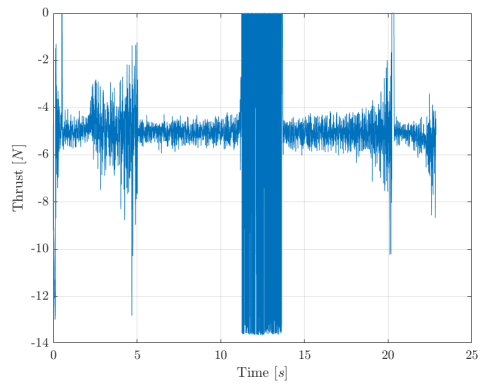


(a) *Simulation with the PID*

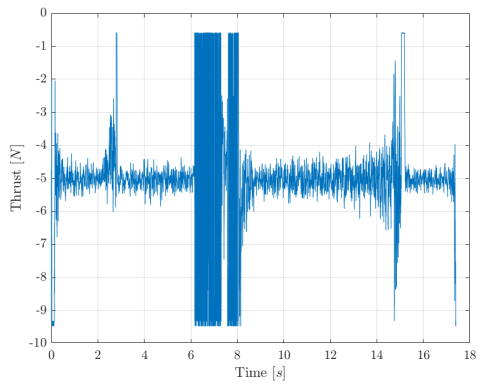


(b) *Simulation with the SMC*

Figure 4.50: Case with local minimum problem with improved APF: Euler angles



(a) *Simulation with the PID*

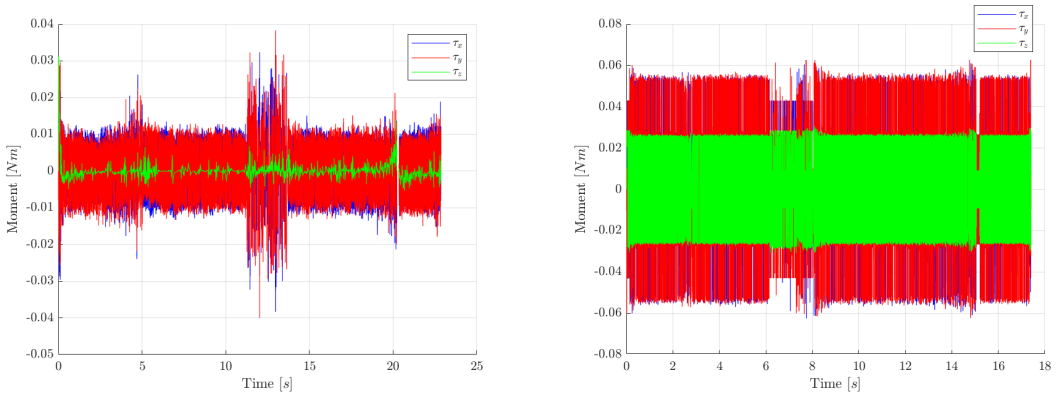


(b) *Simulation with the SMC*

Figure 4.51: Case with local minimum problem with improved APF: thrust lm produced

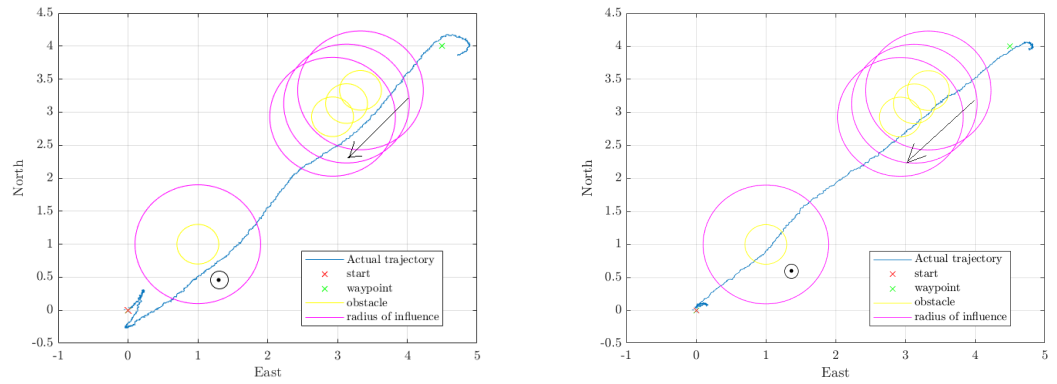
planning the results are not really accurate. In the take off and in the landing the position along the North and East axes are not constant. However, in this case the fault is not only of the controllers. In fact, the random component in the search of the nodes for the path provoques a misalignment of the position reference with respect to an ideal trajectory between the waypoints. The landing is also inaccurate. However, the simulation performed with the SMC is better than the one performed with the PID.

Figure 4.77a on page 108 and figure 4.77b on page 108 are the velocities of the drone along the Down axis. Figure 4.79a on page 109 and figure 4.79b on page 109 are the velocities of the drone along the East axis. Figure 4.78a on page 109 and figure 4.78b on page 109 are the velocities of the drone along the North axis. These velocities are oscillating because the references are calculated with a polynomial equation. Regarding the velocity along the



(a) *Simulation with the PID* (b) *Simulation with the SMC*

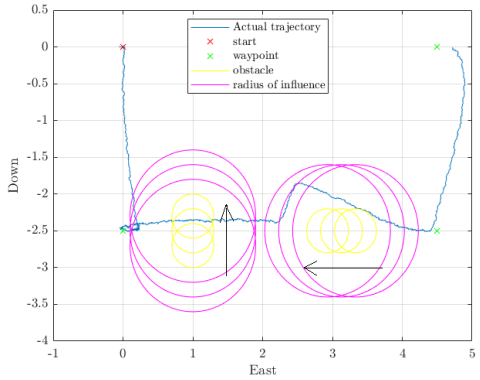
Figure 4.52: Case with local minimum problem with improved APF: moments produced



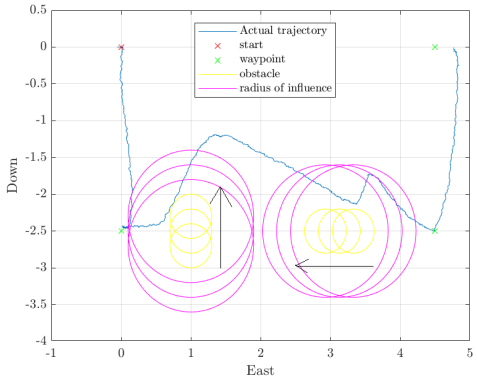
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.53: Case with dynamic obstacles with improved APF: trajectory in the NE plane

Down axis, the drone correctly follows the reference for both controllers. Along the North and East axes the trend is a little bit worse, the effective velocity has a delay with respect to the reference. The reason is that the drone is under-actuated as previously explained. However, the oscillations are small so the effective velocities can follow the references. Velocities calculated in the simulation with the PID have a time interval where they are null. In that time interval, the time calculated to perform the maneuver ended but the drone is not in the waypoint because of the position along the North and East axes. Thus, the reference speeds are imposed as zero until the drone enters in the sfere of tollerance of the waypoint. The Simulation with SMC does not have this problem so the drone reaches the goal before.

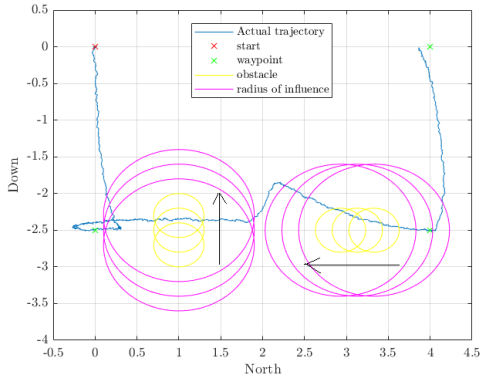


(a) *Simulation with the PID*

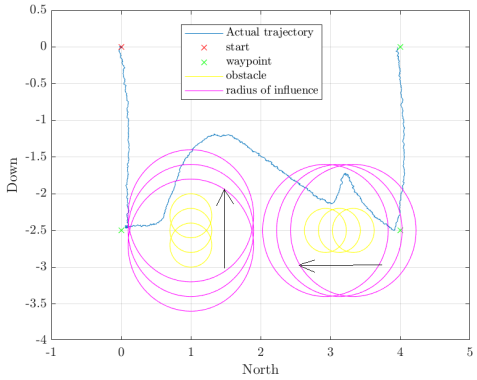


(b) *Simulation with the SMC*

Figure 4.54: Case with dynamic obstacles with improved APF: trajectory in the DE plane



(a) *Simulation with the PID*

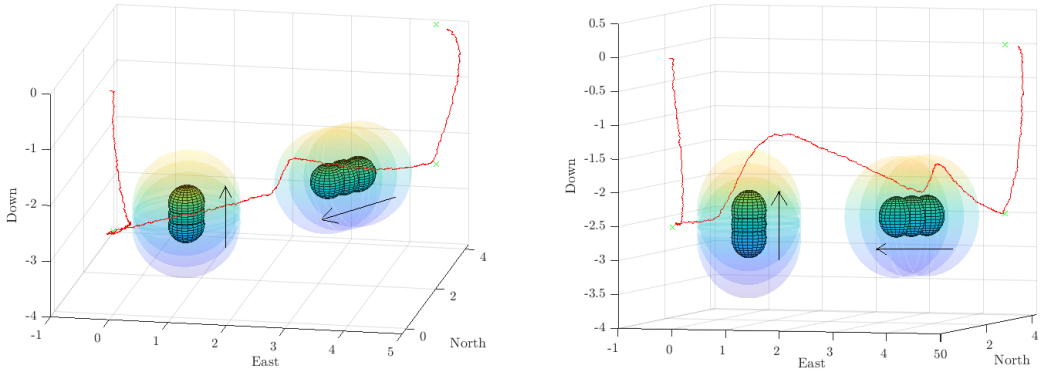


(b) *Simulation with the SMC*

Figure 4.55: Case with dynamic obstacles with improved APF: trajectory in the DN plane

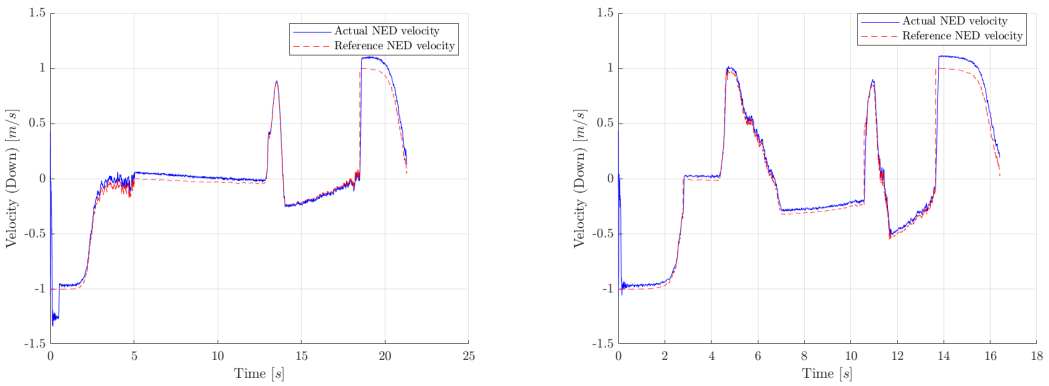
The attitude can be described with the Euler angles (figure 4.70a on page 105 and figure 4.70b on page 105). The trends of the curves are similar between the two controllers. The yaw angle is almost constant because the drone has the nose that points to the goal and the path between them is free. It has small variations because of the variations in the trajectory. The pitch and the roll angle have a similar trend in both controllers.

Figure 4.71a on page 105 and figure 4.71b on page 105 are the thrusts produced by the drone. The one obtained in the simulation with the SMC controller is more oscillating than the other. The reason is that the SMC can follow in a more accurated manner the reference velocity along the Down axis, so the oscillation affects also the thrust. Figure 4.72a on



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.56: Case with dynamic obstacles with improved APF: trajectory in NED



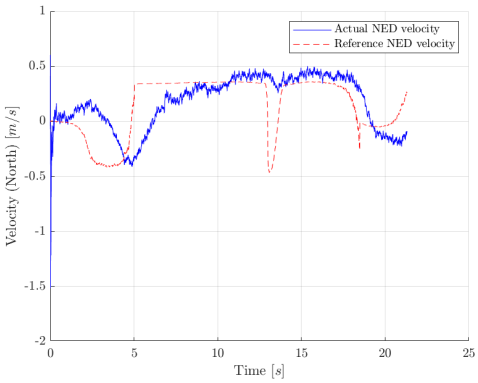
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.57: Case with dynamic obstacles with improved APF: velocity along D axis

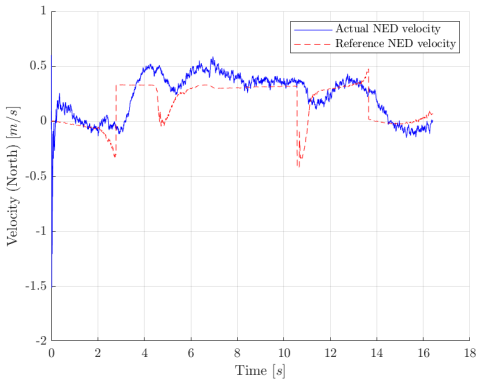
page 106 and figure 4.72b on page 106 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than these obtained from PID. The reason is that the SMC is a more aggressive controller than the PID.

4.1.7 Simulations of RRT*FNDAPF with static obstacles

In this simulation there are three static obstacles placed in (1 1 -2.5), (2 2 -2.5), (3 4 -2). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The results are reported with both controllers to make the comparisons.

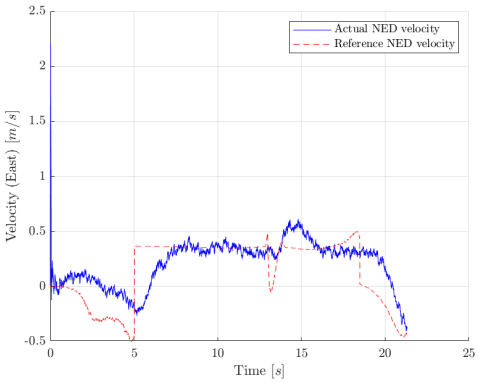


(a) *Simulation with the PID*

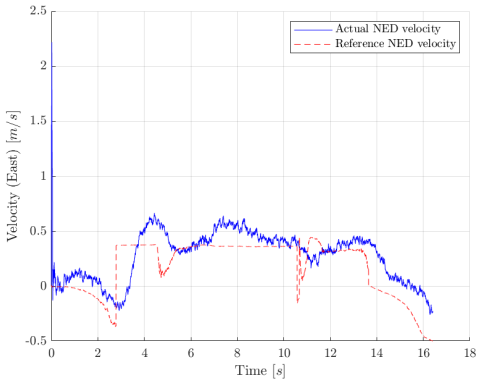


(b) *Simulation with the SMC*

Figure 4.58: Case with dynamic obstacles with improved APF: velocity along N axis



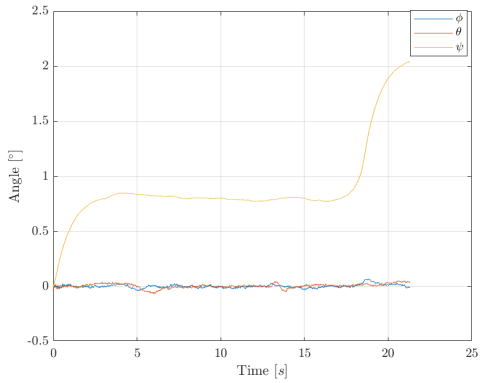
(a) *Simulation with the PID*



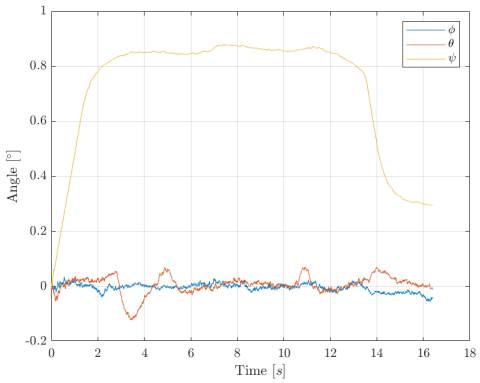
(b) *Simulation with the SMC*

Figure 4.59: Case with dynamic obstacles with improved APF: velocity along E axis

Figure 4.73a on page 106 and figure 4.73b on page 106 are the trajectories traced by the drone in the East-North plane. Figure 4.74a on page 107 and figure 4.74b on page 107 are the ones in the East-Down plane. Figure 4.75a on page 107 and figure 4.75b on page 107 are the ones in the North-Down plane. These results are resumed in the 3D NED environment in figure 4.76a on page 108 and figure 4.76b on page 108. The path obtained in the simulation with the SMC is smoother than the other. In addition, the take off is more accurate. However, the landing of the simulation with PID is more accurate than the one of the simulation with SMC. These plots confirm that the drone, with this algoirthm, can reach the goal avoiding the static obstacles.

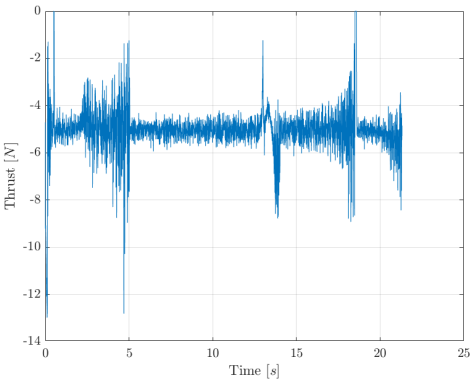


(a) *Simulation with the PID*

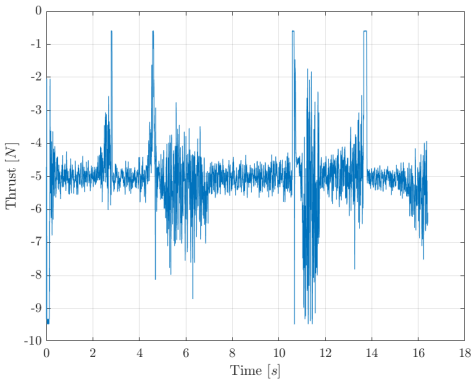


(b) *Simulation with the SMC*

Figure 4.60: Case with dynamic obstacles with improved APF: Euler angles



(a) *Simulation with the PID*

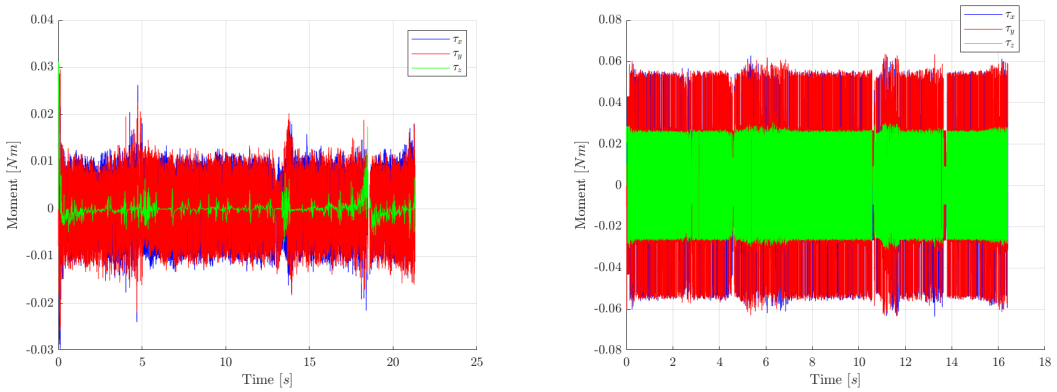


(b) *Simulation with the SMC*

Figure 4.61: Case with dynamic obstacles with improved APF: thrust lm produced

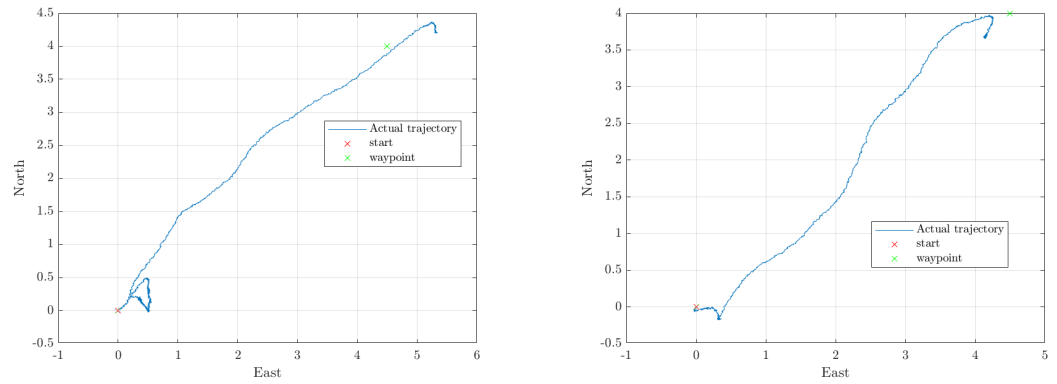
Figure 4.77a on page 108 and figure 4.77b on page 108 are the velocities of the drone along the Down axis. Figure 4.79a on page 109 and Figure 4.79b on page 109 are the velocities of the drone along the East axis. Figure 4.78a on page 109 and figure 4.78b on page 109 are the velocities of the drone along the North axis. Considerations about the velocities are similar to those of the previous case. The control on the Down axis is better than the one on the other axes because the drone is under-actuated. The simulation with PID is longer because of the time interval where the velocities are null as the previous case. Even if the velocities along the North and the East axes have the delay, they follow the reference well. SMC controller can control better than PID.

The attitude can be described with the Euler angles (figure 4.80a on page 110 and



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.62: Case with dynamic obstacles problem with improved APF: moments produced

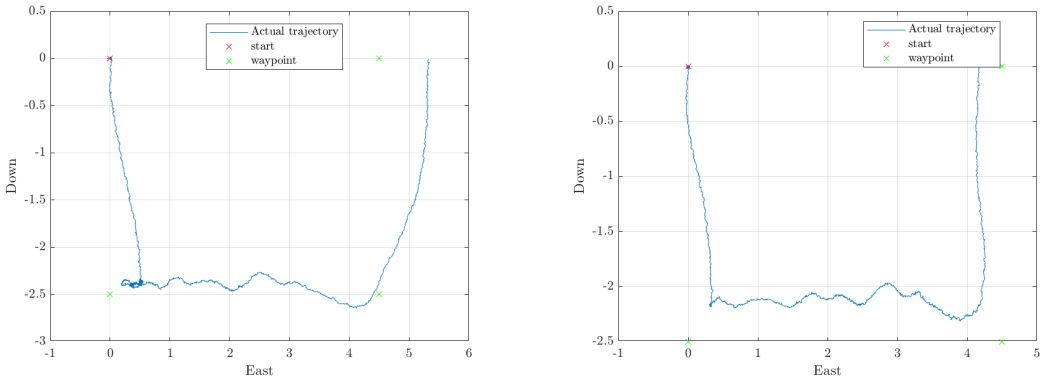


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.63: Case without obstacles with RRT*FNDAPF: trajectory in the NE plane

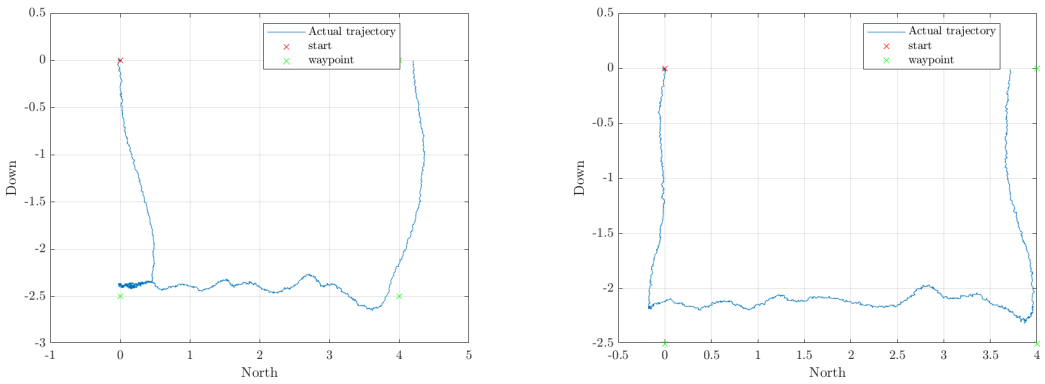
figure 4.80b on page 110). The trend is a little bit different between the two controllers. The yaw angle is determined thanks to the position of the drone with respect to the goal, so the differences in the final part of the curve are the consequence of the different paths near the waypoint (4 4.5 -2,5). As in the simulation without obstalces, the pitch and the roll angle have a similar trend in both controllers. However, the SMC ones have major changes because it is a more aggressive controller.

Figure 4.81a on page 110 and figure 4.81b on page 110 are the thrusts produced by the drone. The two trends are similar as expected from the similar trend of the velocity and the position along the Down axis. Figure 4.82a on page 111 and figure 4.82b on page 111 are the moments produced by the drone. The moments produced with the commands



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.64: Case without obstacles with RRT*FNDAPF: trajectory in the DE plane



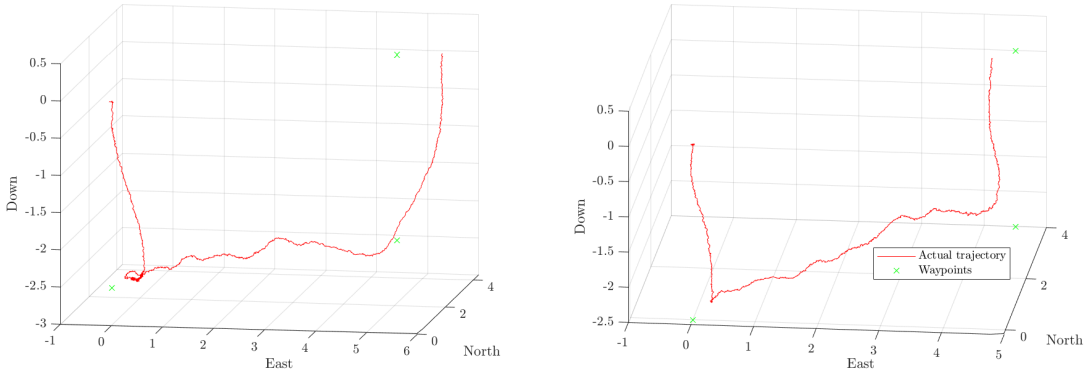
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.65: Case without obstacles with RRT*FNDAPF: trajectory in the DN plane

obtained from the SMC are larger than those obtained from PID. The reason is the SMC is a more aggressive controller than the PID. As the previous case the thrust obtained in the simulation with the SMC is less dense and has more oscillations than the other.

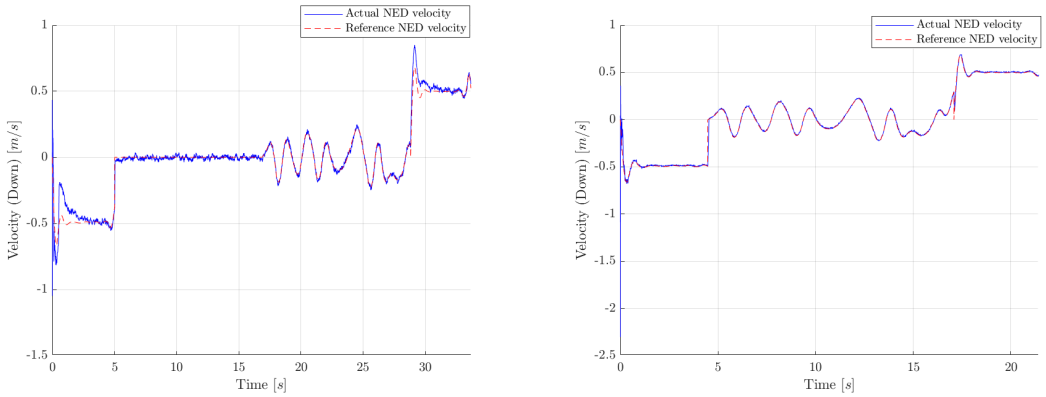
4.1.8 Simulations of RRT*FNDAPF with GNRON problem

In this simulation there is a static obstacle placed in (4.2 4.9 -2.5). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The obstacle is too near to the second waypoint which is in the radius of influence of the obstacle, so the drone cannot reach the waypoint with the



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.66: Case without obstacles with RRT*FNDAPF: trajectory in NED

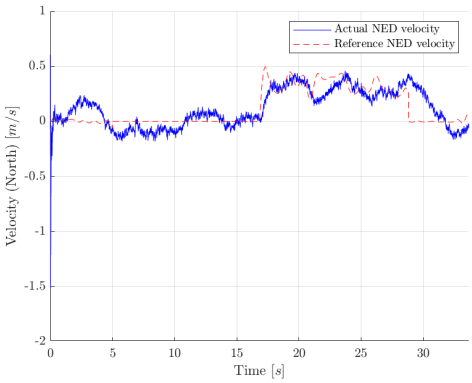


(a) *Simulation with the PID* (b) *Simulation with the SMC*

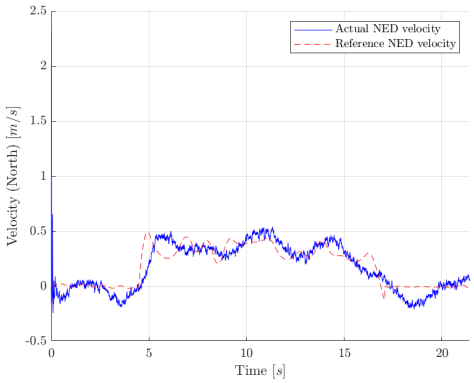
Figure 4.67: Case without obstacles with RRT*FNDAPF: velocity along D axis

classic APF. Thus, these simulations are made to verify if this guidance algorithm, that uses an APF as search algorithm, suffers the same problem. The results are reported with both controllers to make the comparisons.

Figure 4.83a on page 111 and figure 4.83b on page 111 are the trajectories traced by the drone in the East-North plane. Figure 4.84a on page 112 and figure 4.84b on page 112 are the trajectories traced by the drone in the Down-East plane. Figure 4.85a on page 112 and figure 4.85b on page 112 are the trajectories traced by the drone in the plane Down-North. Even if this algorithm has an APF as search algorithm, it does not have the GNRON problem. In fact, in these simulations the drone reaches the goal. These results are resumed in the 3D NED environment in figure 4.86a on page 113 and figure 4.86b on page 113. The

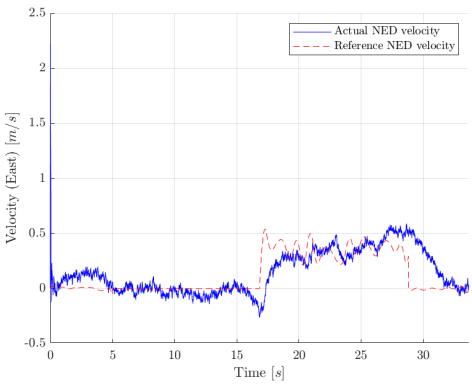


(a) *Simulation with the PID*

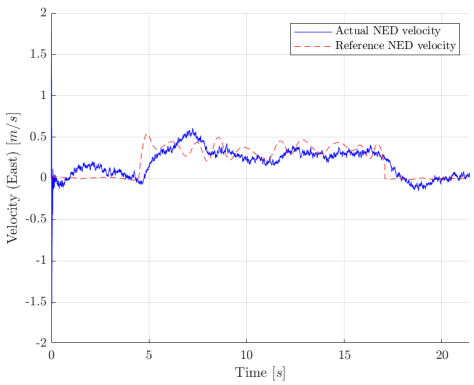


(b) *Simulation with the SMC*

Figure 4.68: Case without obstacles with RRT*FNDAPF: velocity along N axis



(a) *Simulation with the PID*

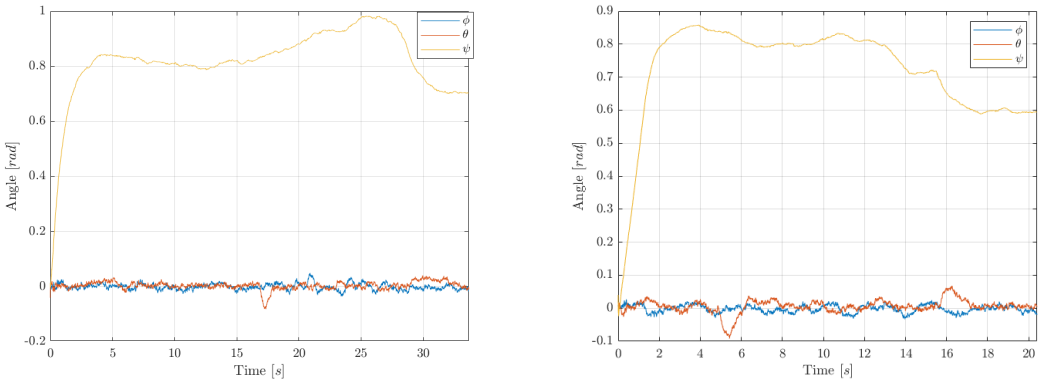


(b) *Simulation with the SMC*

Figure 4.69: Case without obstacles with RRT*FNDAPF: velocity along E axis

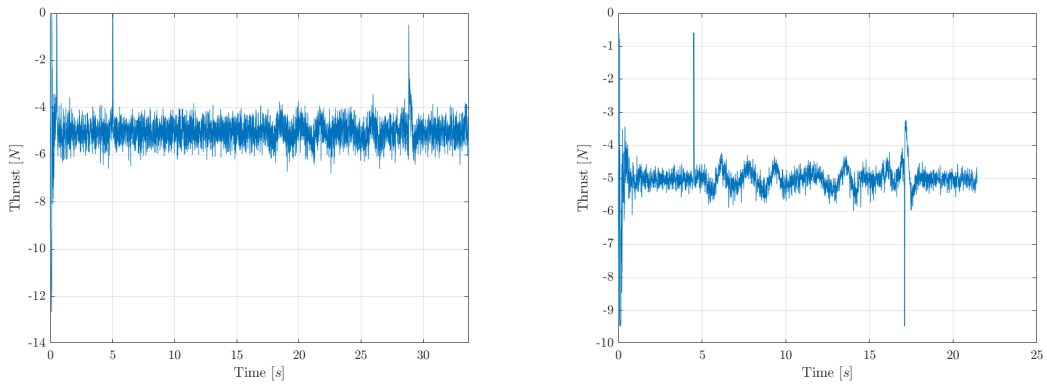
trajectory obtained with the simulation with the SMC is better than the other. In fact, take off and landing are more accurate in this simulation.

Figure 4.87a on page 113 and figure 4.87b on page 113 are the velocities of the drone along the Down axis. Figure 4.89a on page 114 and figure 4.89b on page 114 are the velocities of the drone along the East axis. Figure 4.88a on page 114 and figure 4.88b on page 114 are the velocities of the drone along the North axis. The trend of the curve of the speed is similar to the one of the case without obstacle. The curves oscillate because of the polynomial relationship and the one of the simulation with the PID has a time interval where they are null. As in the previous cases, the delay along the North and East axes is not a problem.



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.70: Case without obstacles with RRT*FNDAPF: Euler angles

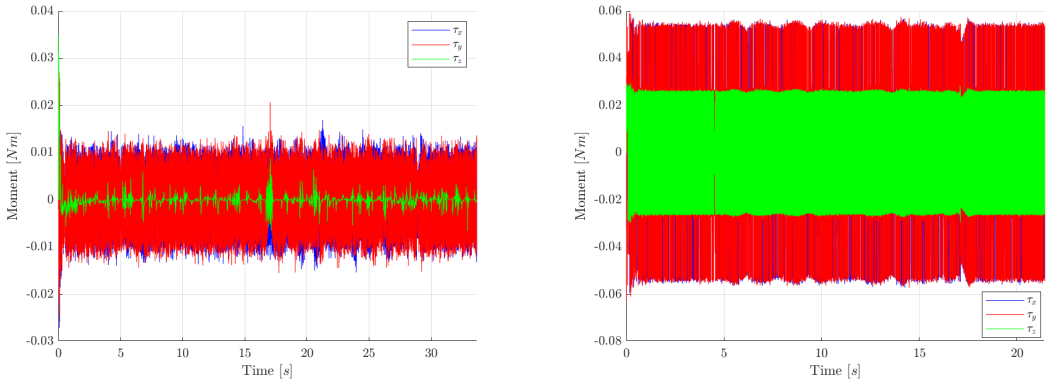


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.71: Case without obstacles with RRT*FNDAPF: thrust produced

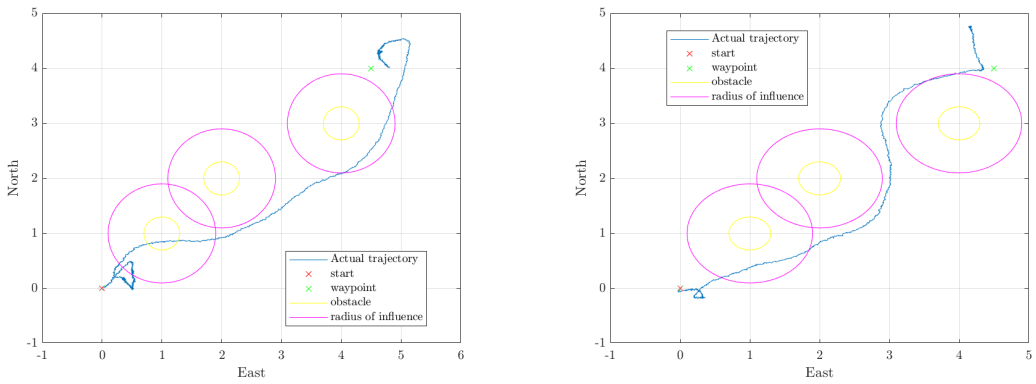
The attitude can be described with the Euler angles (figure 4.90a on page 115 and figure 4.90b on page 115). The trend is similar in the two simulations. The yaw angle is almost constant because of the absence of obstacles in the central part of the path. As in the previous simulations, the pitch and the roll angle have a similar trend in both controllers.

Figure 4.91a on page 115 and figure 4.91b on page 115 are the thrusts produced by the drone. As in the previous case the thrust is more dense in the simulation with the PID and the one of the simulation with SMC presents some oscillations in the central part. Figure 4.92a on page 116 and figure 4.92b on page 116 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.72: Case without obstacles with RRT*FNDAPF: moments produced



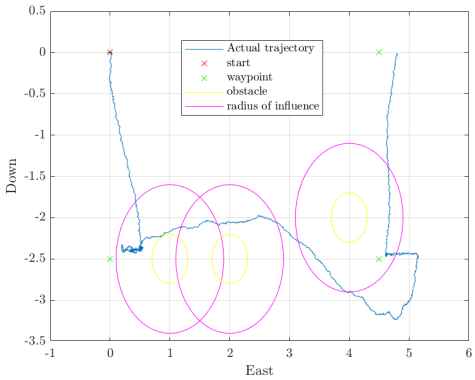
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.73: Case with static obstacles with RRT*FNDAPF: trajectory in the NE plane

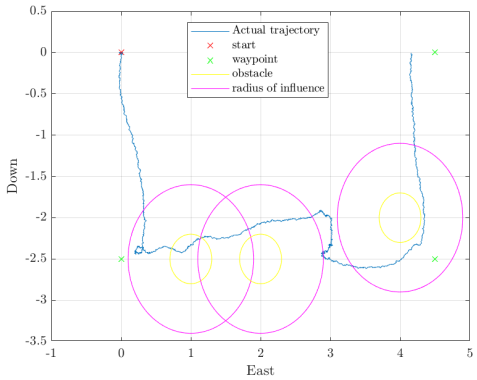
these obtained from PID. The reason is the SMC is a more aggressive controller than the PID.

4.1.9 Simulation of RRT*FNDAPF with a local minimum

In this simulation there are four obstacles place in (1.50 2.25 -2.5), (2.25 1.50 -2.5), (1.875 1.875 -3.1) and (1.875 1.875 -1.9). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). There is a local minimum zone between these obstacles where the drone with the classic APF falls without being able to escape. Because this algortihm uses an APF as search

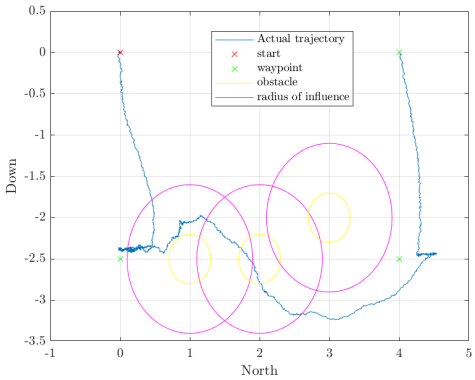


(a) *Simulation with the PID*

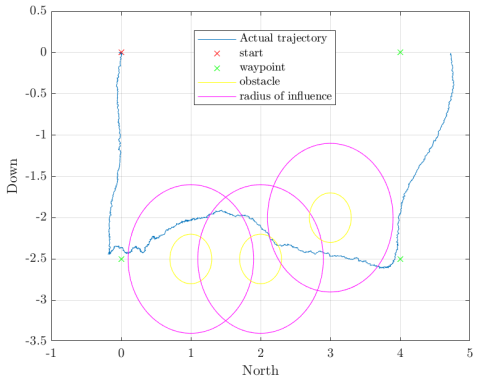


(b) *Simulation with the SMC*

Figure 4.74: Case with static obstacles with RRT*FNDAPF: trajectory in the DE plane



(a) *Simulation with the PID*

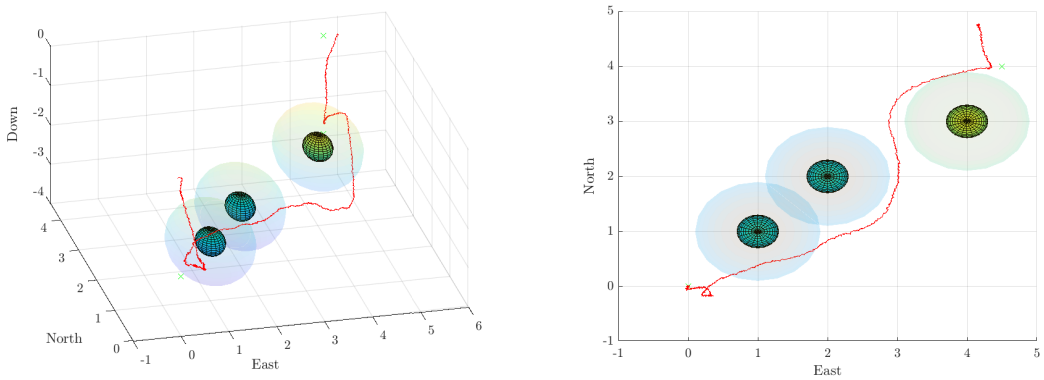


(b) *Simulation with the SMC*

Figure 4.75: Case with static obstacles with RRT*FNDAPF: trajectory in the DN plane

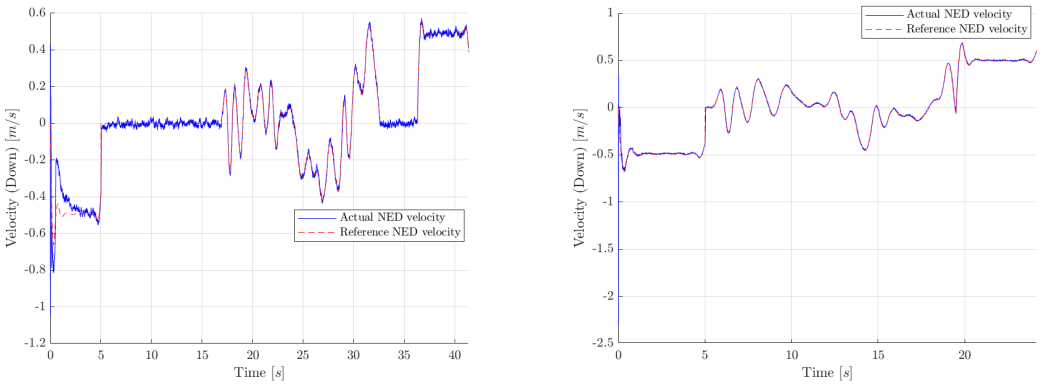
algorithm, these simulations are made to verify if it suffers the same problem. The results are reported with both controllers to make the comparisons.

Figure 4.93a on page 116 and figure 4.93b on page 116 are the trajectories traced by the drone in the East-North plane. Figure 4.94a on page 117 and figure 4.94b on page 117 are the trajectories traced by the drone in the Down-East plane. 4.95a on page 117 and figure 4.95b on page 117 are the trajectories traced by the drone in the Down-North plane. In the simulations where the classic APF is used, the drone falls in the local minimum zone. However, with this algorithm that does not happen because this algorithm maps all the environment before the departure. By doing so it can provide the local minimum zone and avoid to fall in it. these results are resumed in the 3D NED environment in



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.76: Case with static obstacles with RRT*FNDAPF: trajectory in NED

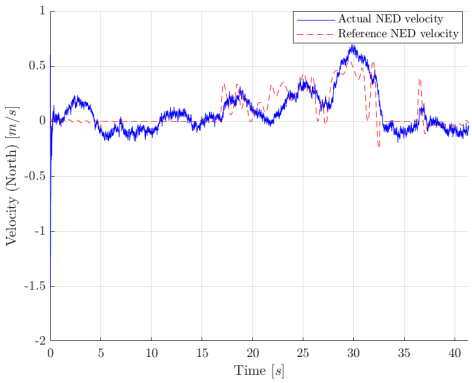


(a) *Simulation with the PID* (b) *Simulation with the SMC*

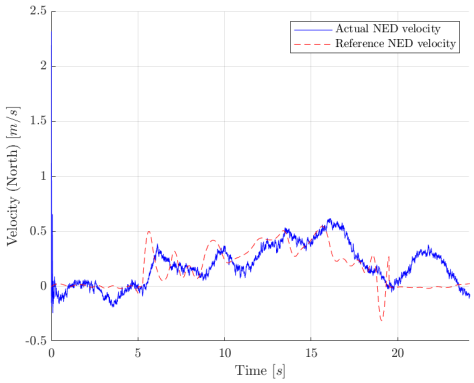
Figure 4.77: Case with static obstacles with RRT*FNDAPF: velocity along D axis

figure 4.96a on page 118 and figure 4.96b on page 118. In both the simulation the drone countour the obstacles. The trajectory of the simulation with the SMC is smoother than the other because SMC is a more performing controller than PID.

Figure 4.97a on page 118 and figure 4.97b on page 118 are the velocities of the drone along the Down axis. Figure 4.99a on page 119 and figure 4.99b on page 119 are the velocities of the drone along the East axis. Figure 4.98a on page 119 and figure 4.98b on page 119 are the velocities of the drone along the North axis. The velocities have the same trend of the previous simulations with this algorithm: several oscillations and the ones of the simulation with the PID have the time interval where they are null. The oscillations of the velocity along the North and East axes are mitigated by the delay.

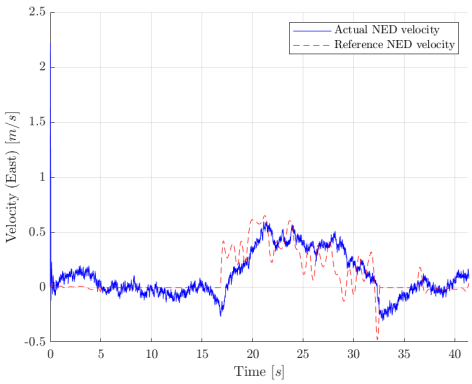


(a) *Simulation with the PID*

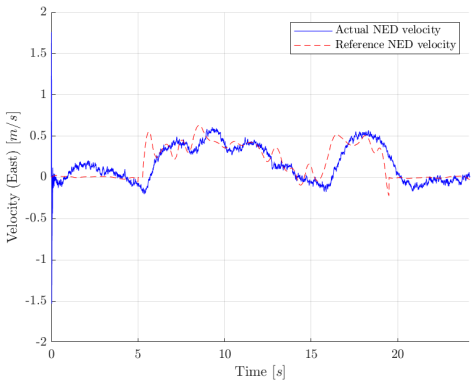


(b) *Simulation with the SMC*

Figure 4.78: Case with static obstacles with RRT*FNDAPF: velocity along N axis



(a) *Simulation with the PID*

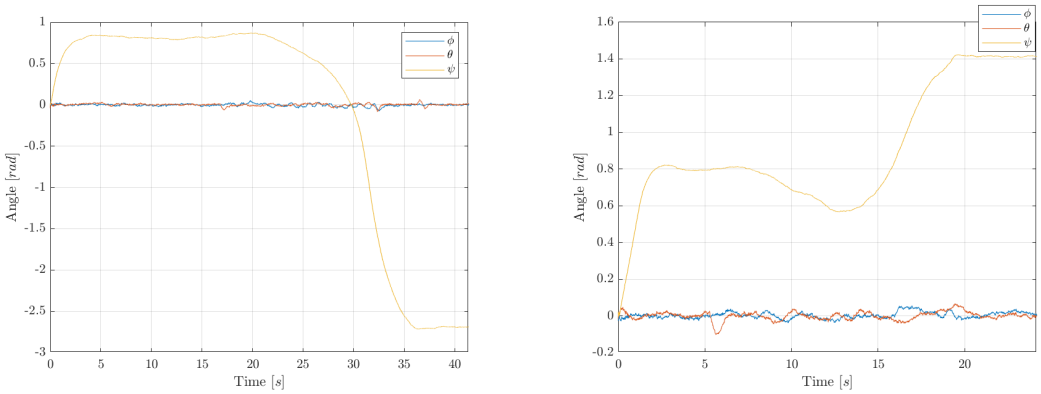


(b) *Simulation with the SMC*

Figure 4.79: Case with static obstacles with RRT*FNDAPF: velocity along E axis

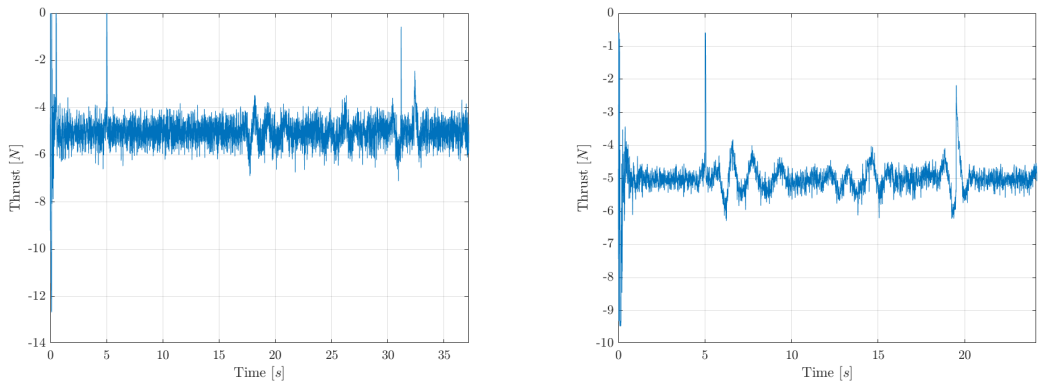
The attitude can be described with the Euler angles (figure 4.100a on page 120 and figure 4.100b on page 120). The differences in the path affect the trend of the yaw angle. The pitch and the roll angle also have different trends because they are the reason why position on the North-East plane are different. In addition, the SMC ones have major changes because it is a more aggressive controller.

Figure 4.101a on page 120 and figure 4.101b on page 120 are the thrusts produced by the drone. Figure 4.102a on page 121 and figure 4.102b on page 121 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than those obtained from PID. The reason is that the SMC is a more



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.80: Case with static obstacles with RRT*FNDAPF: Euler angles



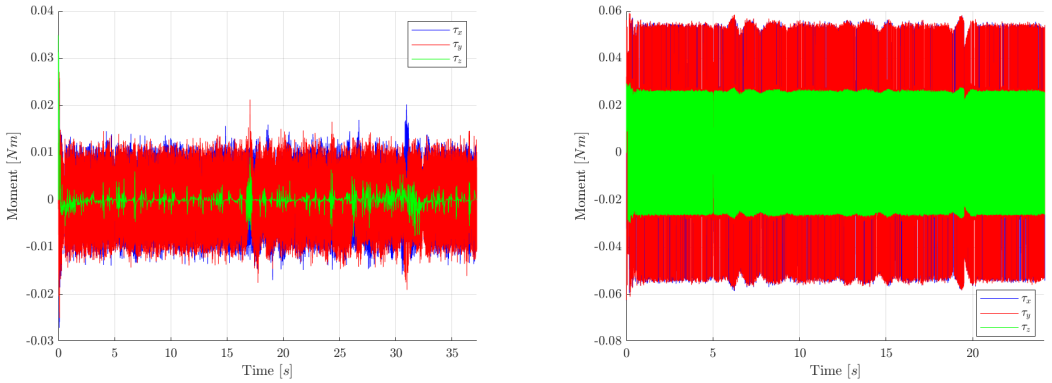
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.81: Case with static obstacles with RRT*FNDAPF: thrust produced

aggressive controller than the PID. The curves of the thurst have the same trend of the previous simulations with this guidance algorithm. Interestingly, these trends are very different from those of the previous guidance algorithm, the reason will be explained in the last section of this chapter.

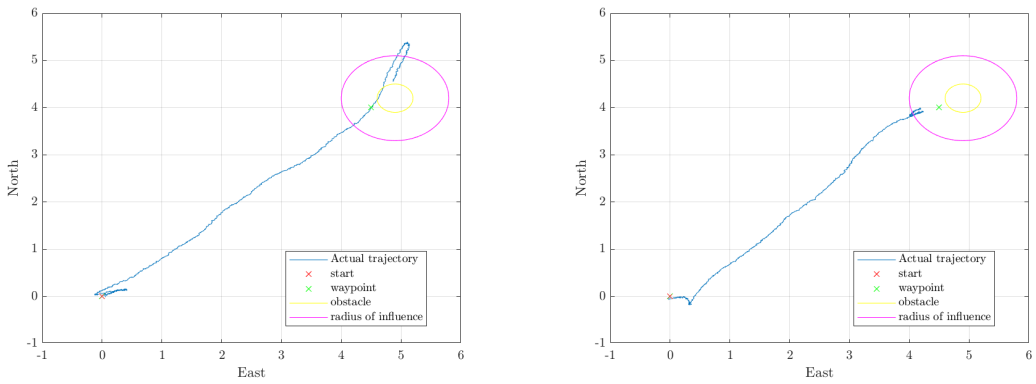
4.1.10 Simulations of RRT*FNDAPF with dynamic obstacles

In this simulation there are two dynamic obstacles that start from (9 9 -2.5) and (1 1 -5) and have a velocity of (0.5 0.5 0) and (0 0 0.5). Starting from the center of the reference frame, the drone has to take off to (0 0 -2.5), then to fly to (4 4.5 -2.5) and lastly to land in (4 4.5 0). The drone will meet the obstacle between the waypoints (0 0 -2.5) and (4 4.5



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.82: Case without obstacles with RRT*FNDAPF: moments produced

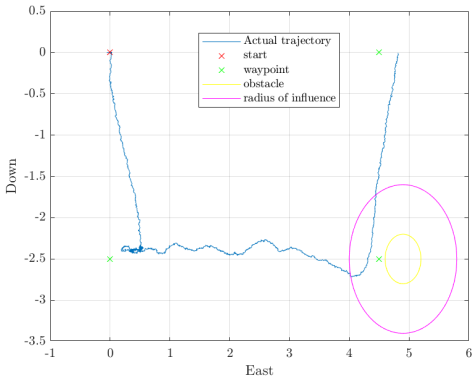


(a) *Simulation with the PID* (b) *Simulation with the SMC*

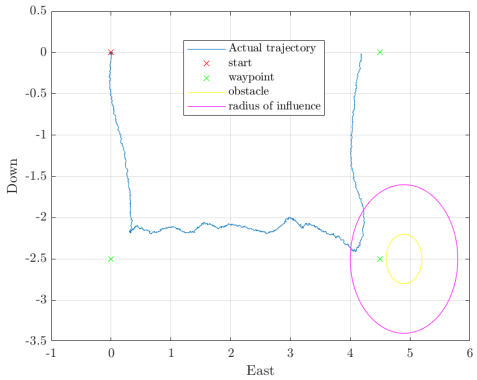
Figure 4.83: Case with GNRON problem with RRT*FNDAPF: trajectory in the NE plane

-2.5). The results are reported with both controllers to make the comparisons.

Figure 4.103a on page 121 and figure 4.103b on page 121 are the trajectories traced by the drone in the East-North plane. Figure 4.104a on page 122 and figure 4.104b on page 122 are the ones in the East-Down plane. Figure 4.105a on page 122 and figure 4.105b on page 122 are the ones in the North-Down plane. these results are resumed in the 3D NED environment in figure 4.106a on page 123 and figure 4.106b on page 123. The arrows indicate the direction of the motion of the obstacles. SMC controller is more accurate and aggressive with respect to the PID controller. Thus, the way the drone avoids the obstacles is different in the two simulations. In the simulation with SMC the drone avoids the first obstacle by reducing the altitude and the second by passing sideaway. However, in the

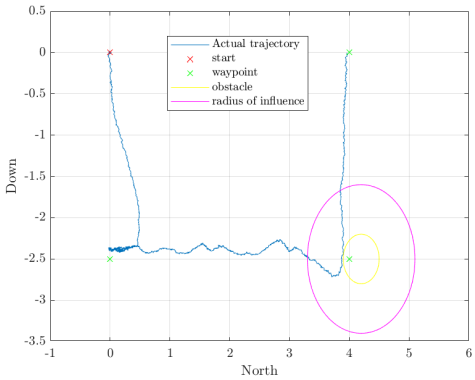


(a) *Simulation with the PID*

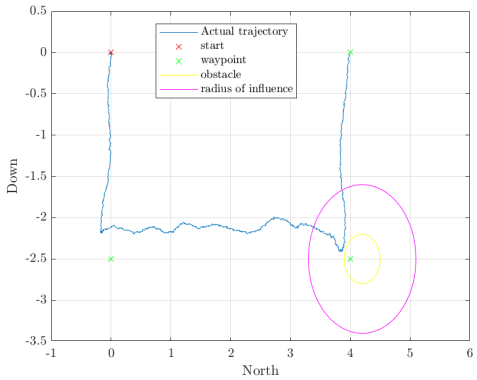


(b) *Simulation with the SMC*

Figure 4.84: Case with GNRON problem with RRT*FNDAPF: trajectory in the DE plane



(a) *Simulation with the PID*

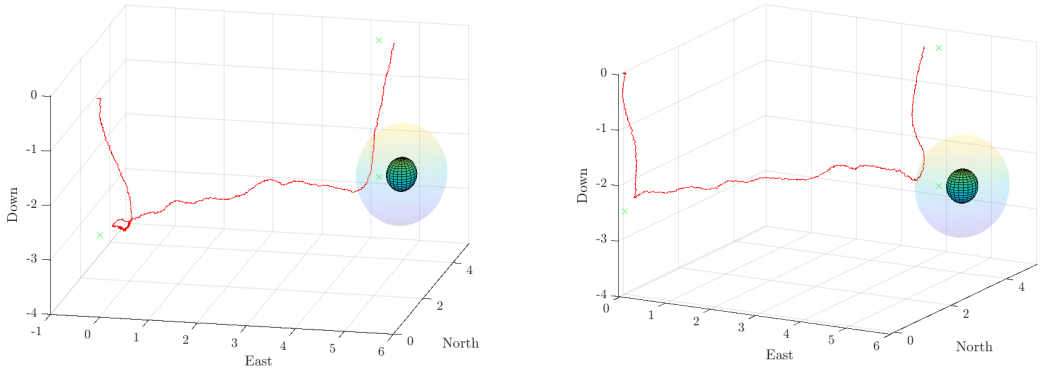


(b) *Simulation with the SMC*

Figure 4.85: Case with GNRON problem with RRT*FNDAPF: trajectory in the DN plane

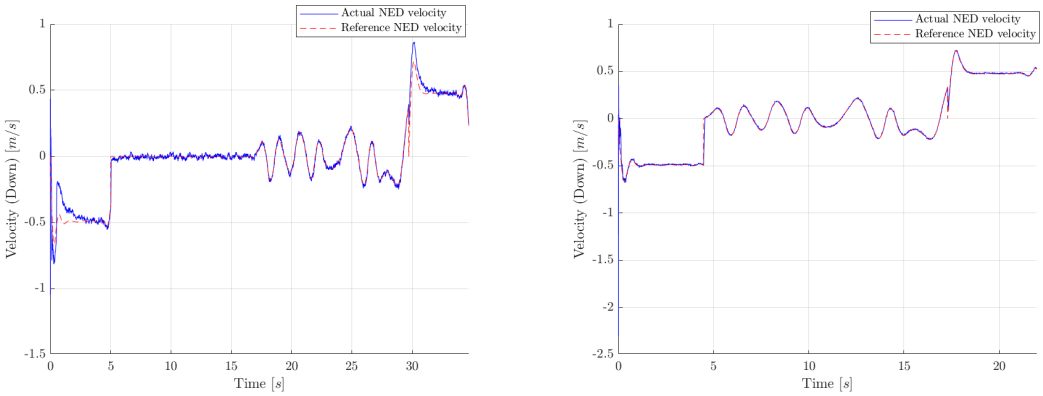
simulation with PID the drone does not encounter the first obstacle because of the interval time where it has to correct the trajectory of the take off. The second obstacle is avoided by reducing the altitude. These plots confirm that the improvement in the algorithm allows the drone to reach the goal avoiding the dynamic obstacles.

Figure 4.107a on page 123 and figure 4.107b on page 123 are the velocities of the drone along the Down axis. Figure 4.109a on page 124 and figure 4.109b on page 124 are the velocities of the drone along the East axis. Figure 4.108a on page 124 and figure 4.108b on page 124 are the velocities of the drone along the North axis. The trends are the same of the previous simulations. The effective velocity along the Down axis of the simulation with the SMC has some peaks probably due to the tuning of the controller. However, this



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.86: Case with GNRON problem with RRT*FNDAPF: trajectory in NED



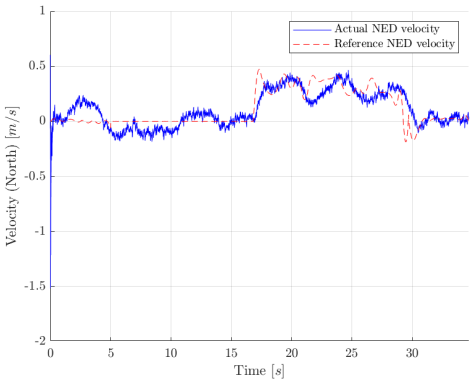
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.87: Case with GNRON problem with RRT*FNDAPF: velocity along D axis

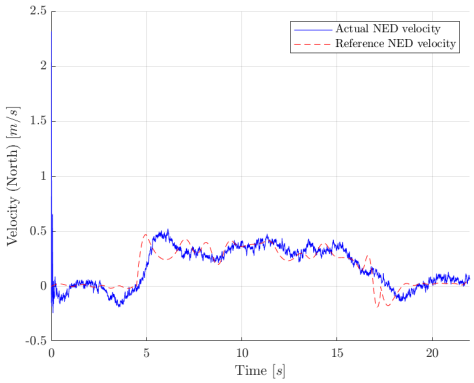
tuning is needed to reach the waypoints. There are some peaks in the reference velocities along the North and East axes which are too short for the delay.

The attitude can be described with the Euler angles (figure 4.110a on page 125 and figure 4.110b on page 125). The trends is similar to those of the previous simulations. The SMC ones have major changes because it is a more aggressive controller.

Figure 4.111a on page 125 and figure 4.111b on page 125 are the thrusts produced by the drone. The curves of the thurst of the simulation with the SMC have a different trend with respect to the previous simulation with this guidance algorithm. The reason is the tuning of the controller as for the speed along the Down axis. Figure 4.112a on page 126

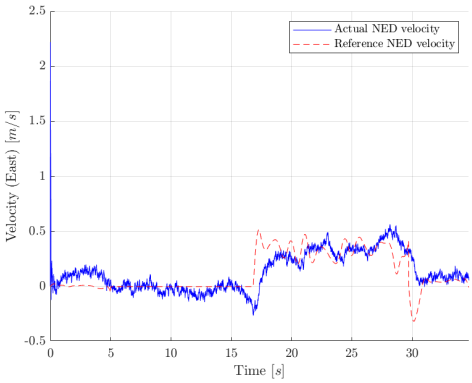


(a) *Simulation with the PID*

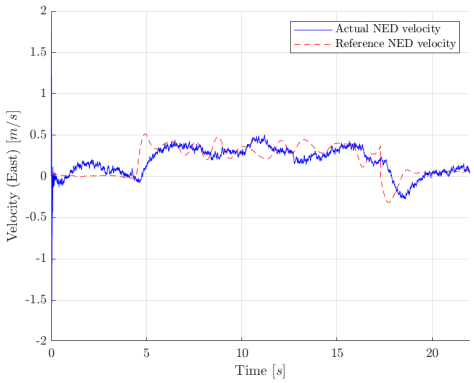


(b) *Simulation with the SMC*

Figure 4.88: Case with GNRON problem with RRT*FNDAPF: velocity along N axis



(a) *Simulation with the PID*



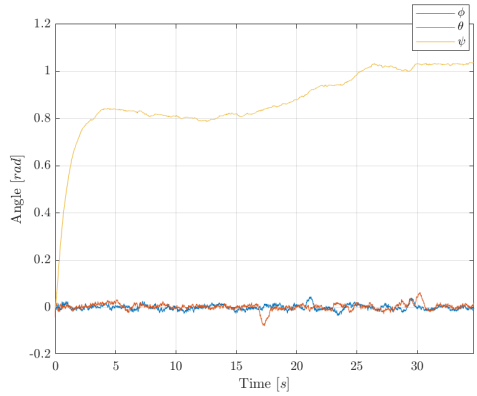
(b) *Simulation with the SMC*

Figure 4.89: Case with GNRON problem with RRT*FNDAPF: velocity along E axis

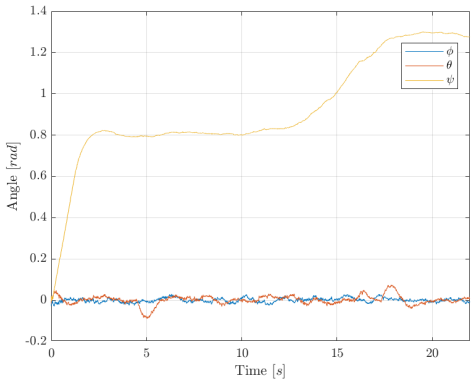
and figure 4.112b on page 126 are the moments produced by the drone. The moments produced with the commands obtained from the SMC are larger than those obtained from PID. The reason is that the SMC is a more aggressive controller than the PID.

4.1.11 Comparison of the algorithms

These two algorithms are quite different from each other. The APF returns more accurate paths because it does not have the random component of the RRT*FNDAPF and it is less computationally expensive. Its speeds are smoother and it is not bound by the time of the maneuver. It is also easier to implement. However, RRT*FNDAPF can map the

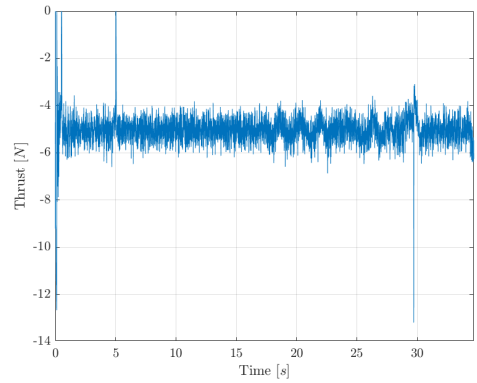


(a) *Simulation with the PID*

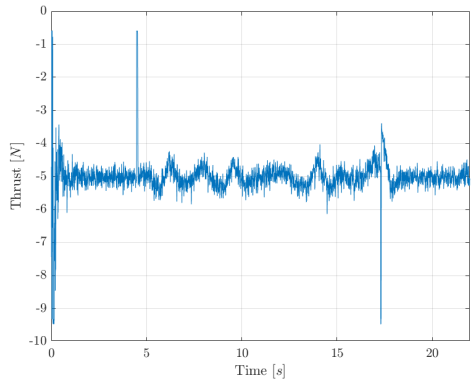


(b) *Simulation with the SMC*

Figure 4.90: Case with GNRON problem with RRT*FNDAPF: Euler angles



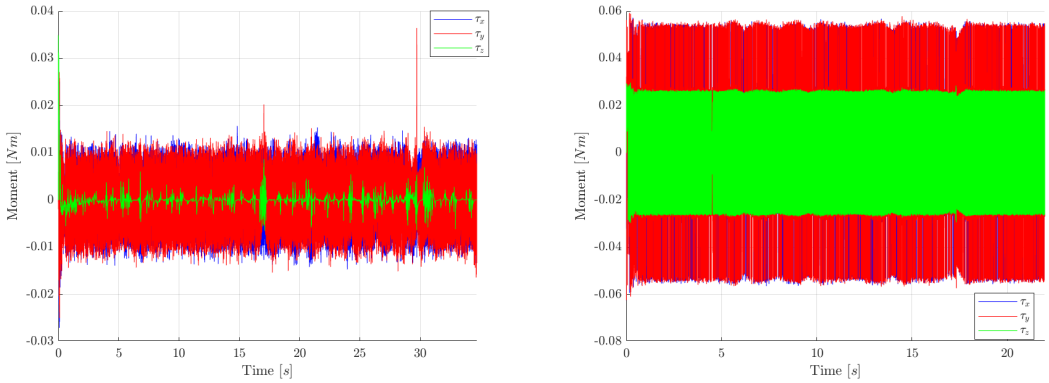
(a) *Simulation with the PID*



(b) *Simulation with the SMC*

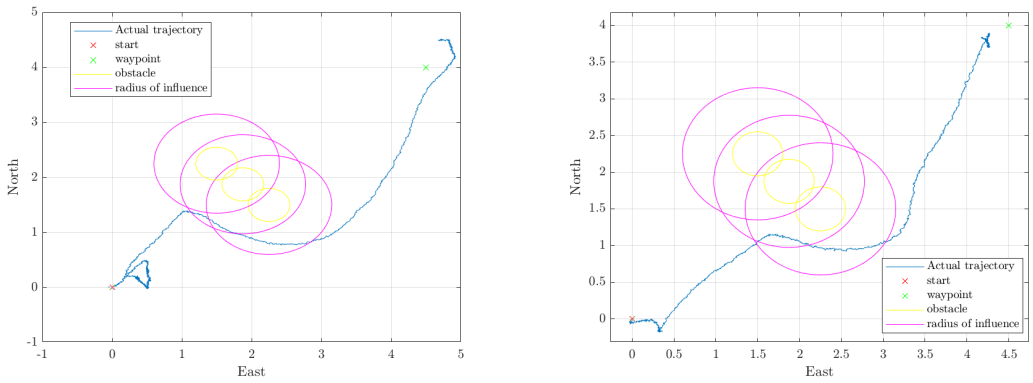
Figure 4.91: Case with GNRON problem with RRT*FNDAPF: thrust GNRON produced

environement before the flight that allows, for example, to avoid local minima zones. In addition, It produces thrusts with a better trend: it is almost constant and the throttle is less than the those produced using an APF as guidance algorithm. This is because the drone has decided before the starting of the flight which path to follow, so the reference is better. In the APF the reference for the position is constant and it is the goal. So, depending on the application for the drone each one can be the more useful than other one.



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.92: Case with GNRON problem with RRT*FNDAPF: moments produced

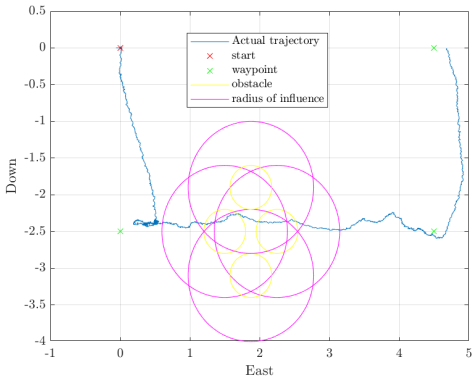


(a) *Simulation with the PID* (b) *Simulation with the SMC*

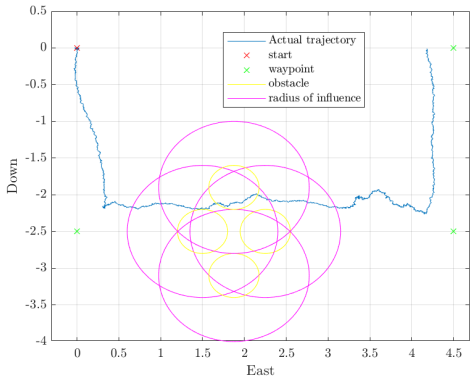
Figure 4.93: Case with local minimum problem with RRT*FNDAPF: trajectory in the NE plane

4.2 Simulations on Simulink and Unity

For these simulations the schema of the model is a little bit different. The blocks of the plant is implemented on Unity. Unity has a physics engine that allows to have more realistic results with respect to the ones obtained with the plant implemented on Simulink. The sensor and the EKF blocks are not used. It simulates an ideal situation where all the states are known and the noise is not present. The signal of the actuators passes through ROS to go in Unity. Then the odometry passes through ROS to go to Simulink. It is useful to use ROS because when in future works the guidance and the controller will be deployed on the autopilot they will be converted in ROS nodes using the code generation feature of

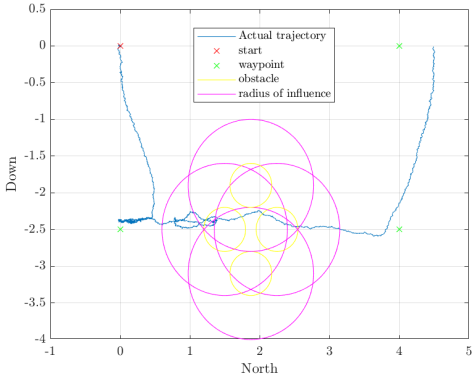


(a) *Simulation with the PID*

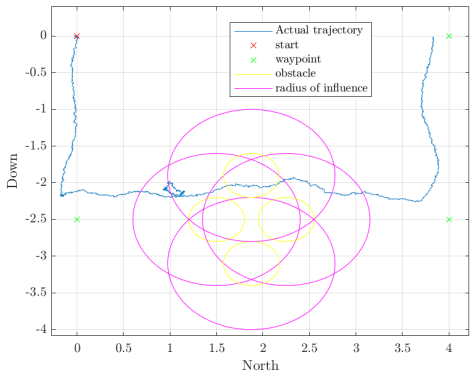


(b) *Simulation with the SMC*

Figure 4.94: Case with local minimum problem with RRT*FNDAPF: trajectory in the DE plane



(a) *Simulation with the PID*

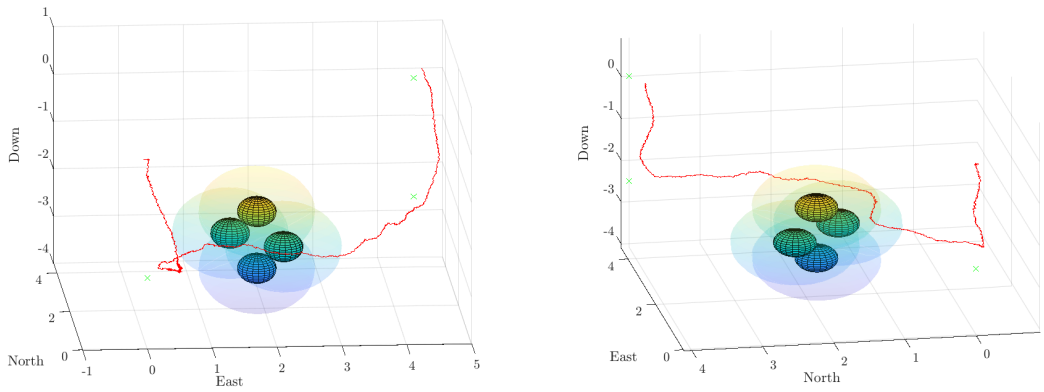


(b) *Simulation with the SMC*

Figure 4.95: Case with local minimum problem with RRT*FNDAPF: trajectory in the DN plane

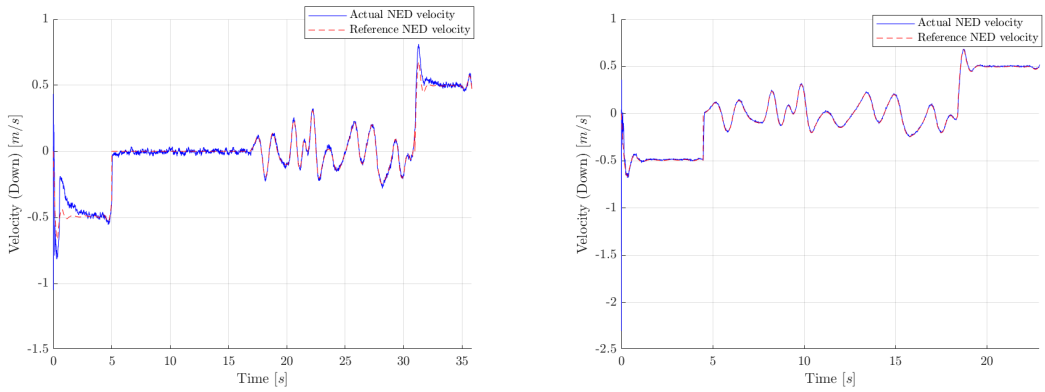
Simulink.
There are some differences with respect to the guidance and the controller blocks of the previous model:

- a Simulation Pace block is added, which allows to control the speed of the simulation. If the simulation in Simulink runs too fast with respect to Unity, this block forces the simulation on Simulink to decrease its speed.



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.96: Case with local minimum problem with RRT*FNDAPF: trajectory in NED



(a) *Simulation with the PID* (b) *Simulation with the SMC*

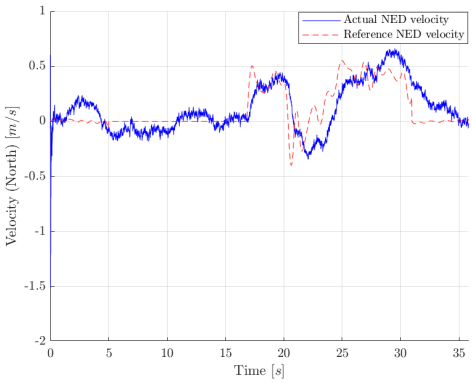
Figure 4.97: Case with local minimum problem with RRT*FNDAPF: velocity along D axis

- The actuator block is thus modified:

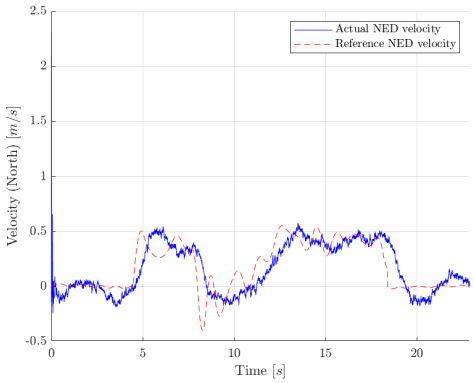
$$F_1(pwm_1) = (2.382 * 10^{-6})pwm_1^2 - (3.628 * 10^{-3})pwm_1 + 1.177$$
$$F_2(pwm_2) = (2.382 * 10^{-6})pwm_2^2 - (3.628 * 10^{-3})pwm_2 + 1.177$$
$$F_1(pwm_3) = (2.382 * 10^{-6})pwm_3^2 - (3.628 * 10^{-3})pwm_3 + 1.177$$
$$F_1(pwm_4) = (2.382 * 10^{-6})pwm_4^2 - (3.628 * 10^{-3})pwm_4 + 1.177$$

  and for the yaw angle control the yaw command is directly used.

For these simulations the reference frame used is the SEZ. The same simulations of the previous environment are made and the video of the results are in the repository. The

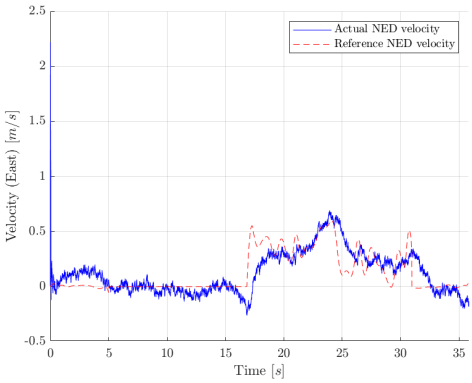


(a) *Simulation with the PID*

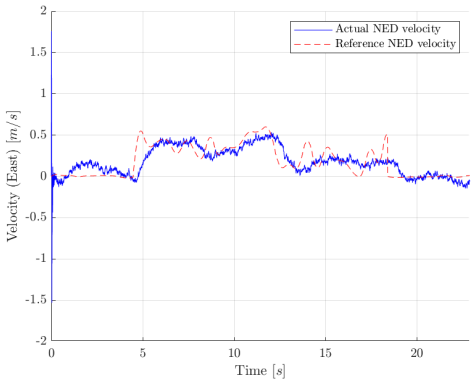


(b) *Simulation with the SMC*

Figure 4.98: Case with local minimum problem with RRT*FNDAPF: velocity along N axis



(a) *Simulation with the PID*

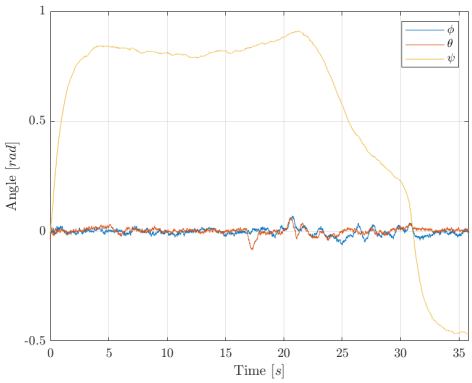


(b) *Simulation with the SMC*

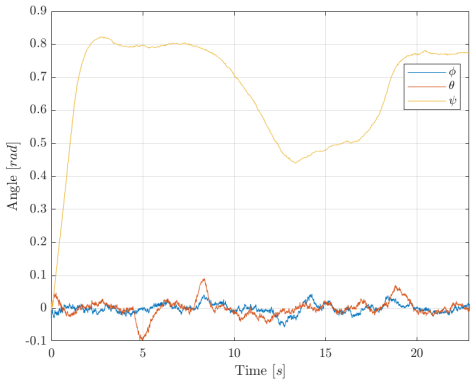
Figure 4.99: Case with local minimum problem with RRT*FNDAPF: velocity along E axis

results are worse than those obtained from simulink but the drone still reaches the goal avoiding the obstacles so they are positive. In these simulations, a PID controller is used because the SMC has too much chattering and it was unusable. In the future work, a Super Twisting Sliding Mode Controller (STSMC) could be implemented. It is a sliding mode of the second order so the chattering is reduced with this controller. The PID controller is a reason why these simulations are less accurate. In fact, also for the simulations on simulink the ones with the PID were less accurate than the ones with the SMC.
Regarding the simulations with the RRT*FNDAPF two problems surrounded:

- The first one is the delay on the position with respect to the reference. This one is present also in the simulations with the APF but for the APF the time of the

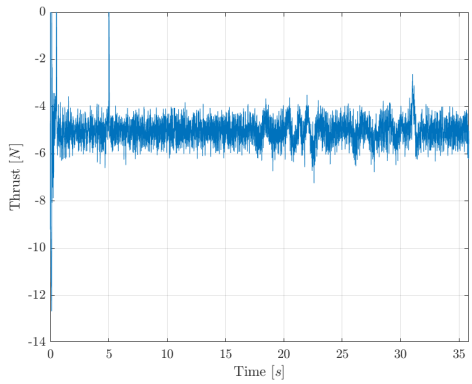


(a) *Simulation with the PID*

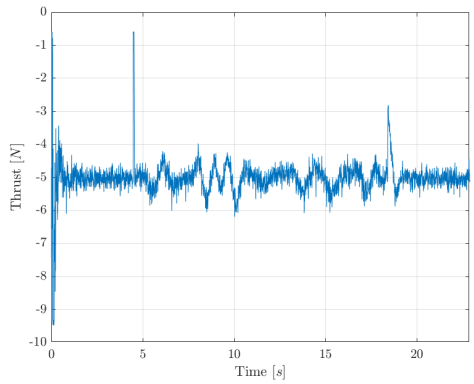


(b) *Simulation with the SMC*

Figure 4.100: Case with local minimum problem with RRT*FNDAPF: Euler angles



(a) *Simulation with the PID*

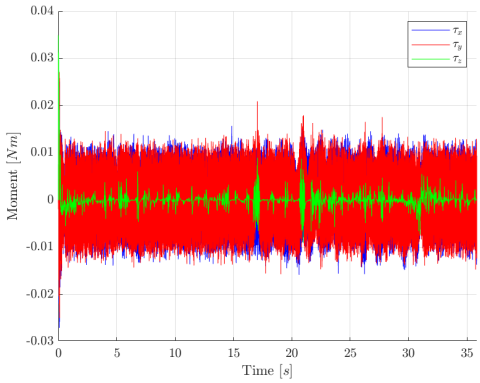


(b) *Simulation with the SMC*

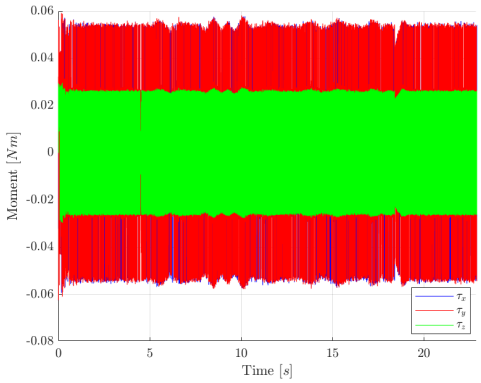
Figure 4.101: Case with local minimum problem with RRT*FNDAPF: thrust produced

simulation is not important. In simulations with the RRT*FNDAPF, when the time designed by the trajectory planner is over, the speed reference becomes null so the drone does not reach the goal. To avoid this problem, a control is added at the trajectory generator. If the time of the maneuver is over and the drone is not in a tollerance radius from the goal, a supplementary reference speed is added. This new reference speed decreases while the drone is approaching the goal.

- The second one is the asincronization between the dynamic of the quadrotor and the path planning. When the drone reaches a waypoint and points to a new goal, the path planning begins to calculate the new path. In this moment the simulink model is blocked for some seconds depending on the size of the environment of the drone. If

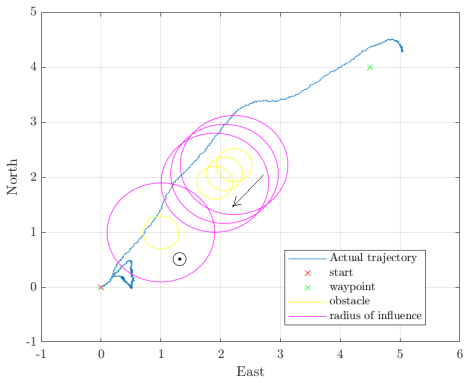


(a) *Simulation with the PID*

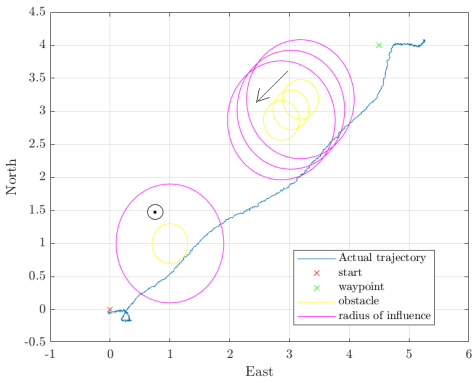


(b) *Simulation with the SMC*

Figure 4.102: Case with local minimum problem with RRT*FNDAPF: moments produced



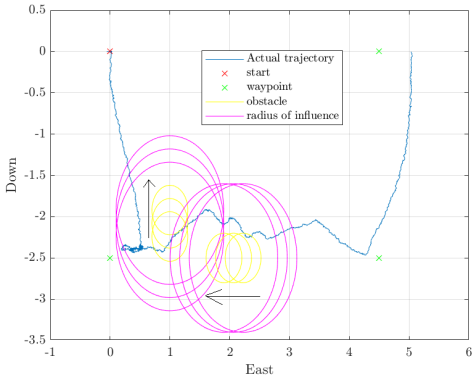
(a) *Simulation with the PID*



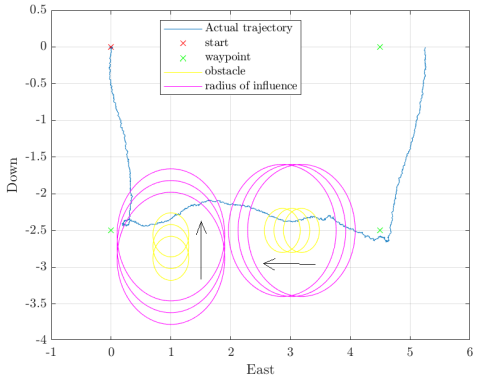
(b) *Simulation with the SMC*

Figure 4.103: Case with dynamic obstacles with RRT*FNDAPF: trajectory in the NE plane

the drone has some residual speed, it will move without control thanks to this speed. To mitigate this previous problem is used: in the final portion of the path, the drone, thanks to the delay, will reduce its speed because the reference is null. However, this trick increases the time of the simulation.

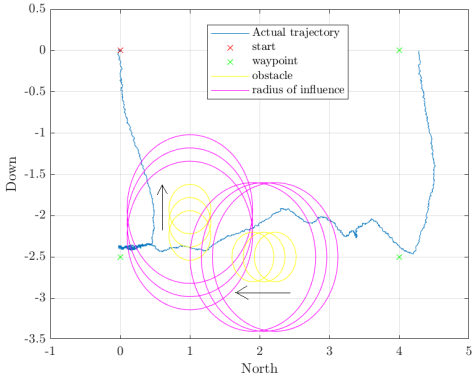


(a) *Simulation with the PID*

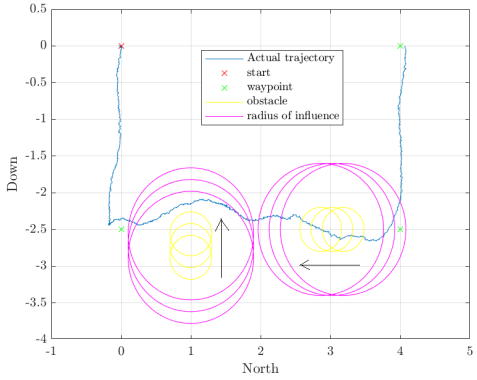


(b) *Simulation with the SMC*

Figure 4.104: Case with dynamic obstacles with RRT*FNDAPF: trajectory in the DE plane

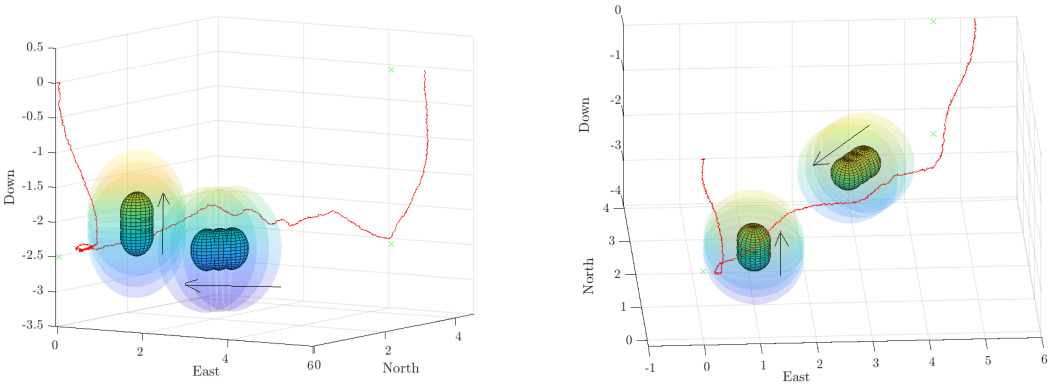


(a) *Simulation with the PID*



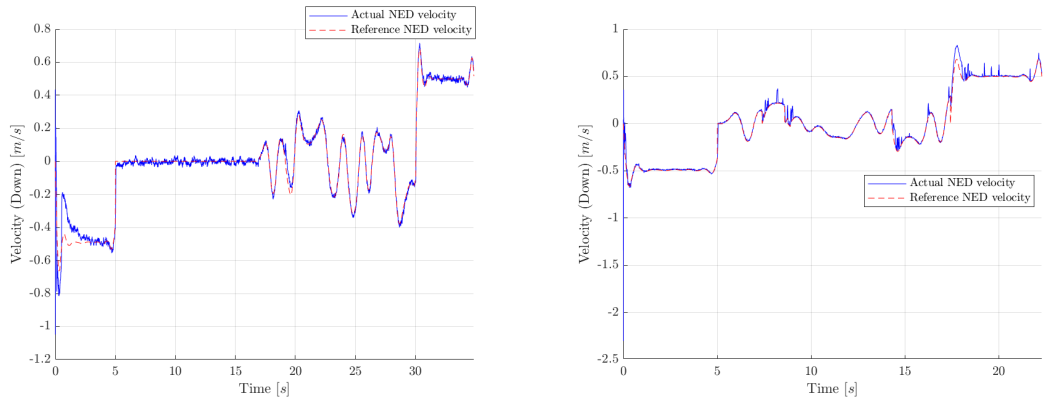
(b) *Simulation with the SMC*

Figure 4.105: Case with dynamic obstacles with RRT*FNDAPF: trajectory in the DN plane



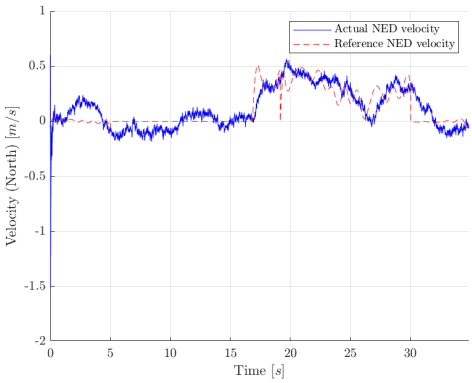
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.106: Case with dynamic obstacles with RRT*FNDAPF: trajectory in NED

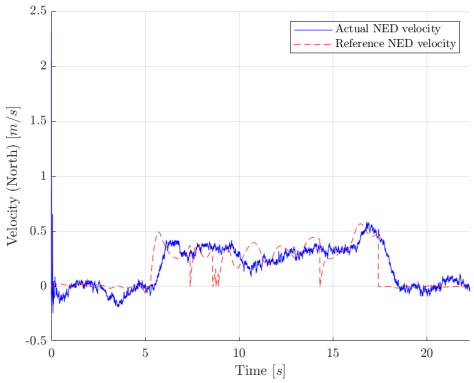


(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.107: Case with dynamic obstacles with RRT*FNDAPF: velocity along D axis

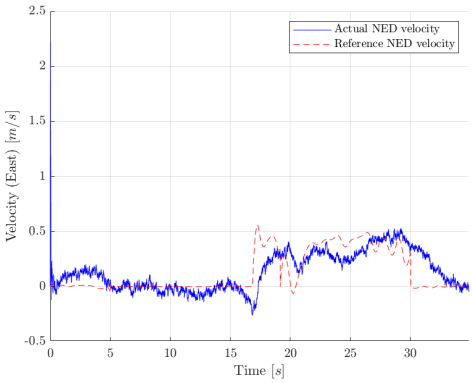


(a) *Simulation with the PID*

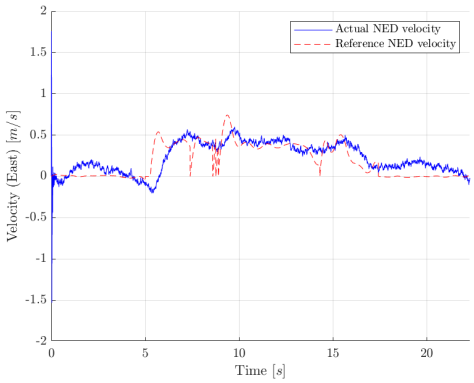


(b) *Simulation with the SMC*

Figure 4.108: Case with dynamic obstacles with RRT*FNDAPF: velocity along N axis

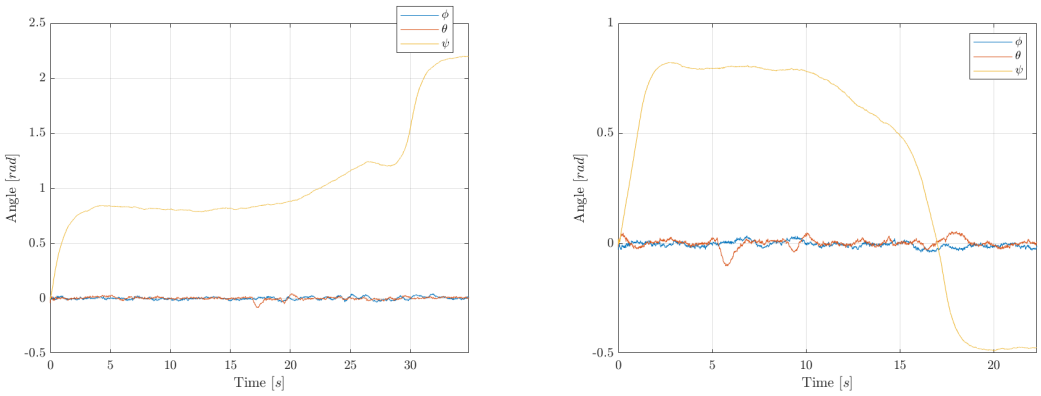


(a) *Simulation with the PID*



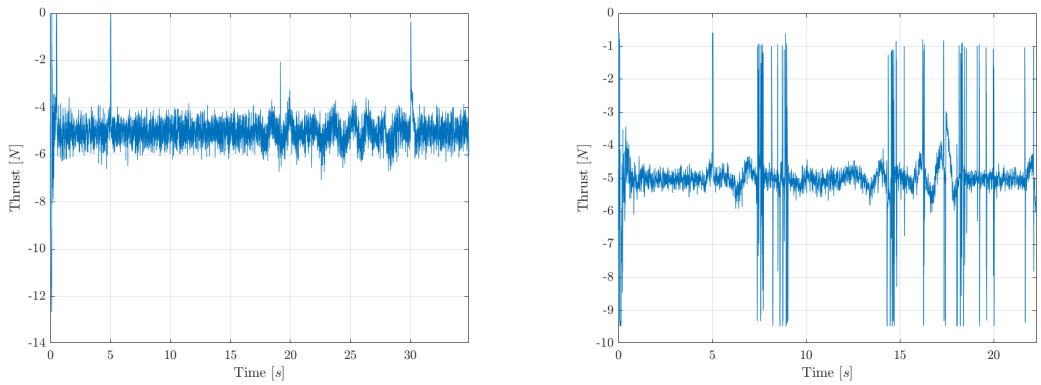
(b) *Simulation with the SMC*

Figure 4.109: Case with dynamic obstacles with RRT*FNDAPF: velocity along E axis



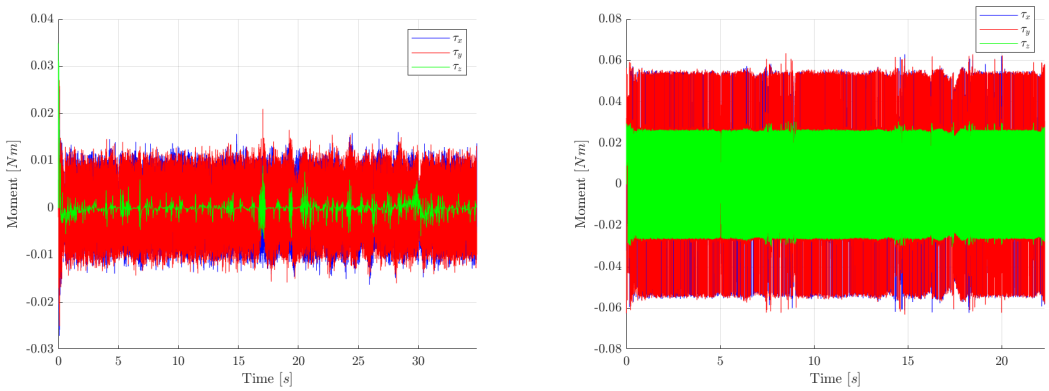
(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.110: Case with dynamic obstacles with RRT*FNDAPF: Euler angles



(a) *Simulation with the PID* (b) *Simulation with the SMC*

Figure 4.111: Case with dynamic obstacles with RRT*FNDAPF: thrust lm produced



(a) *Simulation with the PID*

(b) *Simulation with the SMC*

Figure 4.112: Case with dynamic obstacles problem with RRT*FNDAPF: moments produced

Chapter 5

Conclusions

This thesis can be divided in two principal sections. In the first one, the design, the modeling and the assembling of the drone are presented. It is a quadrotor, with a weight of about 500 g and an overall dimension of about 29 x 29 cm^2. Its endurance is of about 15 min that allows to have a great autonomy. The components chosen to be mounted are presented in 5.1.

The frame is composed by two 3D printed plates in PLA and the arms are made with

component	**model**
Motors	Tarrot MT1806
Propellers	T-Motor 5035x2
ESC	HobbyWIng XROTOR 10A micro
Battery	FullPower Li-Po 3S 2200 mAh 35C
microcontroller	STM32 nucleo L432KC
On-board computer	Orange Pi Zero 2

Table 5.1: components chosen for the drone

an aluminum tube with a thickness of 1 mm. The 3D printing process is usefull to build the frame of small drones because the production time is low and PLA is inexpensive and easy to print. Moreover, the 3D printing allows to have lightweight plates regulating the infill. Structural analyses on the plates were made to verify that the frame could resist the loads. Thrust tests were made to decide the better combination of motors, propellers and battery and to obtain the analytical relationship between the PWM signal and the thrust produced. This relationship is used in the models for the simulations.

In the second section, two guidance algorithms are presented. The first one is an improved APF. Unlike Khatib's APF, this improved version does not suffer from the GNRON and Local Minima problems and can be used in dynamic environments. The second one is an RRT*FNDAPF that is an improved version of RRT. It uses a heuristic function to obtain

the optimal path, a fixed number of nodes to reduce the computational effort and can be used in dynamic environments. In addition, it uses an APF as search algorithm to increase the convergence speed. These algorithms are implemented in Matlab/Simulink. They are implemented in a way to allow their code generation. The code generation is needed to allow the deployment of the guidance algorithm in the autopilot in future. Then, simulations in Simulink are made to validate the algorithms and compare them. Five different environments are used for the simulation: the first one is without obstacles, the second one is with some generic static obstacles, the third one has a GNRON problem, the fourth one has a Local Minimum and the fifth one has some dynamic obstacles. The simulations are made with two different controllers, a PID and a SMC, to compare the performances. Then, simulations are repeated in a model environment that use Unity as plant. Unity has a physics engine that allows to have more realistic results with respect to the ones obtained with the Simulink model. In these last simulations, PIDs are used as controller because the SMC has too much chattering to be usable.
The results on Matlab/Simulink show that the SMC is better than the PID because it is more aggressive and more precise. The APF produces paths more precise with respect to the RRT*FNDAPF because it does not have the random component in the search algorithm and it is less computational expensive. Its speeds are smoother and it is not bound by the time of the maneuver. However, RRT*FNDAPF tracks the path before the maneuver so it can avoid to fall into local minimum zones and the path produced in the complex environment is safer than the one produced with the APF. The thrust produced is better in the case of RRT*FNDAPF. The reason is that the reference position produced by this algorithm is a reference path instead of APF that uses as reference position the goal and produces only the reference speed. Simulations with Unity are less precise than the previous but the drone can still reach the goal avoiding the obstacles. Thus, these results are overall positive.
The drone is compact and small but the weight is still a bit high. In future works, the weight could be reduced using a more resistant material for the frame, that allows to reduce the thickness of the plates and reduce the weight. Also the vision-based sensors should be implemented and mounted on the drone. Regarding the simulations, a Super Twisting Sliding Mode Controller could be implemented in simulations with Unity. STSMC is a sliding mode controller of the second order, so it reduces the chattering and allows to use a sliding mode controller instead of PID. Thus, more precise results could be obtained. Then the algorithms could be deployed in the autopilot with the code generation feature of Matlab/Simulink. These guidance algorithms are sampling based algorithms. In future works a bionspired algorithm could be implemented to obtain better paths.
In future, the drone should communicate with a Unmanned Ground Vehicle (UGV) to guide it and to land on it. Another possible indoor application could be the exploration of

a crooked and narrow environment like for example a cave because of its limited dimensions and high endurance.

Bibliography

[1] Olzhas Adiyatov and Huseyin Atakan Varol. "A novel RRT-based algorithm for motion planning in Dynamic environments". In: *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2017, pp. 1416–1421. DOI: 10.1109/ICMA.2017.8016024.

[2] Kanaiya Agrawal and Punit Shrivastav. "Multi-rotors: A Revolution In Unmanned Aerial Vehicle". In: 2015.

[3] Nafis Ahmed, Chaitali J. Pawase, and KyungHi Chang. "Distributed 3-D Path Planning for Multi-UAVs with Full Area Surveillance Based on Particle Swarm Optimization". In: *Applied Sciences* 11.8 (2021). ISSN: 2076-3417. DOI: 10.3390/app11083417. URL: https://www.mdpi.com/2076-3417/11/8/3417.

[4] *Arduino Nano quadcopter*. https://www.instructables.com/Arduino-micro-Quadcopter/.

[5] Mustapha Bekhti et al. "Drone Package Delivery: A Heuristic approach for UAVs path planning and tracking". In: *EAI Endorsed Transactions on Internet of Things* 3 (Aug. 2017), p. 153048. DOI: 10.4108/eai.31-8-2017.153048.

[6] Davide Carminati. "Design and Testing of Indoor UAS Control Techniques". In: 2019.

[7] Jinbao Chen et al. "An Improved Probabilistic Roadmap Algorithm with Potential Field Function for Path Planning of Quadrotor". In: *2019 Chinese Control Conference (CCC)*. 2019, pp. 3248–3253. DOI: 10.23919/ChiCC.2019.8865585.

[8] *datasheet orange pi zero2*. http://www.orangepi.org/Orange%20Pi%20Zero2/.

[9] *datasheet pico flexx Camboard*. https://pmdtec.com/en/.

[10] *datasheet stm32 nucleo l432kc*. https://www.st.com/en/evaluation-tools/nucleo-l432kc.html#documentation.

[11] *datasheet Tarrot MT1806*. http://www.tarotrc.com/Product/Detail.aspx?Lang=en&Id=0a760a18-2ff4-4032-b0a9-29977dca5ed8.

[12] Elaf Dhulkefl and Akif Durdu. "Path Planning Algorithms for Unmanned Aerial Vehicles". In: *International Journal of Trend in Scientific Research and Development* Volume-3 (June 2019), pp. 359–362. DOI: 10.31142/ijtsrd23696.

[13] Yiqun Dong, Changhong Fu, and Erdal Kayacan. "RRT-based 3D path planning for formation landing of quadrotor UAVs". In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2016, pp. 1–6. DOI: 10.1109/ICARCV.2016.7838567.

[14] *Everything About LiPo Battery for Racing Drones*. https://oscarliang.com/lipo-battery-guide/.

[15] Xiaojing Fan et al. "Improved Artificial Potential Field Method Applied for AUV Path Planning". In: *Mathematical Problems in Engineering* 2020 (Apr. 2020), p. 6523158. ISSN: 1024-123X. DOI: 10.1155/2020/6523158. URL: https://doi.org/10.1155/2020/6523158.

[16] *FPV Drone ESC Buyer's Guide*. https://oscarliang.com/choose-esc-racing-drones/.

[17] M. A. Gutierrez-Martinez et al. "Collision-free path planning based on a genetic algorithm for quadrotor UAVs". In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020, pp. 948–957. DOI: 10.1109/ICUAS48674.2020.9213956.

[18] Cheng Haohao, Qi Xiaohui, and Yang Sen. "Obstacle Avoidance and Path Planning for Quadrotor based on Differential Evolution - Artificial Bee Colony Algorithm". In: *Journal of Physics: Conference Series* 1087 (Sept. 2018), p. 022030. DOI: 10.1088/1742-6596/1087/2/022030. URL: https://doi.org/10.1088/1742-6596/1087/2/022030.

[19] Samira Hayat et al. "Multi-objective drone path planning for search and rescue with quality-of-service requirements". In: *Autonomous Robots* 44.7 (Sept. 2020), pp. 1183–1198. ISSN: 1573-7527. DOI: 10.1007/s10514-020-09926-9. URL: https://doi.org/10.1007/s10514-020-09926-9.

[20] *How to Choose FPV Drone Motors*. https://oscarliang.com/quadcopter-motor-propeller/#quadcopter-weight-motor-thrust.

[21] *How to Choose Propeller for Mini Quad*. https://oscarliang.com/choose-propellers-mini-quad/.

[22] Iswanto Iswanto et al. "Artificial Potential Field Algorithm Implementation for Quadrotor Path Planning". In: *International Journal of Advanced Computer Science and Applications* 10.8 (2019). DOI: 10.14569/IJACSA.2019.0100876. URL: http://dx.doi.org/10.14569/IJACSA.2019.0100876.

[23] Herath Mpc Jayaweera and Samer Hanoun. "A Dynamic Artificial Potential Field (D-APF) UAV Path Planning Technique for Following Ground Moving Targets". In: *IEEE Access* 8 (2020), pp. 192760–192776. DOI: 10.1109/ACCESS.2020.3032929.

[24] Zhenyue Jia et al. "Online cooperative path planning for multi-quadrotors in an unknown dynamic environment". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 0.0 (0), p. 09544100211016615. DOI: 10.1177/09544100211016615. eprint: https://doi.org/10.1177/09544100211016615. URL: https://doi.org/10.1177/09544100211016615.

[25] Cheonghwa Lee, Seolha Kim, and Baeksuk Chu. "A Survey: Flight Mechanism and Mechanical Structure of the UAV". In: *International Journal of Precision Engineering and Manufacturing* 22.4 (2021), pp. 719–743. ISSN: 2005-4602. DOI: 10.1007/s12541-021-00489-y. URL: https://doi.org/10.1007/s12541-021-00489-y.

[26] *matlab-unity*. https://github.com/IrisDuMutel/matlab_unity.

[27] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[28] *mydroneunity*. https://github.com/IrisDuMutel/mydroneunity.

[29] *Open Category civil drone*. https://www.easa.europa.eu/domains/civil-drones-rpas/open-category-civil-drones.

[30] Prashant Pandey, Anupam Shukla, and Ritu Tiwari. "Three-dimensional path planning for unmanned aerial vehicles using glowworm swarm optimization algorithm". In: *International Journal of System Assurance Engineering and Management* 9.4 (2018), pp. 836–852. ISSN: 0976-4348. DOI: 10.1007/s13198-017-0663-z. URL: https://doi.org/10.1007/s13198-017-0663-z.

[31] Min Park and Min Cheol Lee. "A new technique to escape local minimum in artificial potential field based path planning". In: *Journal of Mechanical Science and Technology* 17 (Dec. 2003), pp. 1876–1885. DOI: 10.1007/BF02982426.

[32] B.K. Patle et al. "A review: On path planning strategies for navigation of mobile robot". In: *Defence Technology* 15.4 (2019), pp. 582–606. ISSN: 2214-9147. DOI: https://doi.org/10.1016/j.dt.2019.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2214914718305130.

[33] The Hung Pham, Dalil Ichalal, and Said Mammar. "Complete coverage path planning for pests-ridden in precision agriculture using UAV". In: *17th IEEE International Conference on Networking, Sensing and Control (ICNSC 2020)*. Proc. of the 17th IEEE International Conference on Networking, Sensing and Control (ICNSC 2020). Nanjing (on line), China: IEEE, Oct. 2020, pp. 1–6. DOI: 10.1109/ICNSC48988.2020.9238122. URL: https://hal.archives-ouvertes.fr/hal-03028878.

[34] *PX4 CUAV v5 nano*. https://docs.px4.io/master/en/flight_controller/cuav_v5_nano.html.

[35] *RCBenchmark*. https://www.rcbenchmark.com/.

[36] Salvatore Romano. "Implementation of an Ultralight Autopilot Drone for Service Robotics". In: 2018.

[37] *rrt toolbox*. https://github.com/olzhas/rrt_toolbox.

[38] Franklin Samaniego et al. "Recursive Rewarding Modified Adaptive Cell Decomposition (RR-MACD): A Dynamic Path Planning Algorithm for UAVs". In: *Electronics* 8 (Mar. 2019), p. 306. DOI: 10.3390/electronics8030306.

[39] Jose Luis Sanchez-Lopez et al. "A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments". In: *Journal of Intelligent & Robotic Systems* 93.1 (2019), pp. 33–53. ISSN: 1573-0409. DOI: 10.1007/s10846-018-0809-5. URL: https://doi.org/10.1007/s10846-018-0809-5.

[40] Simone Silvestro. "Optimization of an Ultralight Autonomous Drone for Service Robotics". In: 2019.

[41] *Simplify 3D*. https://www.simplify3d.com/.

[42] *Trajectory generator*. https://github.com/RajatBhageria/Quadrotor-Robotics.

[43] Yi Gao Rui Li Yingjing Shi Li Xiao. "Design of path planning and tracking control of quadrotor". In: *Journal of Industrial & Management Optimization* 0 (2021), pp. –.

[44] Liang Yang et al. "Survey of Robot 3D Path Planning Algorithms". In: *Journal of Control Science and Engineering* 2016 (2016), p. 7426913. ISSN: 1687-5249. DOI: 10.1155/2016/7426913. URL: https://doi.org/10.1155/2016/7426913.

[45] Ren Ziwu, Li Chunguang, and Sun Lining. "Minimum-Acceleration Trajectory Optimization for Humanoid Manipulator Based on Differential Evolution". In: *International Journal of Advanced Robotic Systems* 13 (Apr. 2016), p. 1. DOI: 10.5772/63070.