République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A.MIRA-BEJAIA

Faculté de Technologie
Département de Génie Electrique
Laboratoire de Technologies Industrielles et de l'Information

THÈSE

EN VUE DE L'OBTENTION DU DIPLOME DE DOCTORAT

Domaine : Sciences et Technologies
Filière : Electronique
Spécialité : Automatique

Présentée par
Mr. BELAID Amine

Thème

3D Trajectory planning for Autonomous Aerial Systems

Soutenue le : 02/03/2022 **Devant le Jury composé de :**

Nom et Prénom	Grade	Université	Qualité
Mr. LEHOUCHE Hocine	MCA	Univ. de Bejaia	Président
Mr. MENDIL Boubekeur	Professeur	Univ. de Bejaia	Rapporteur
Mr. HASSAM Abdelouahab	Professeur	Univ. de Setif1	Examinateur
Mr. MEKHMOUKH Abdenour	MCA	Univ. de Bejaia	Examinateur

Année Universitaire : 2021/2022

Acknowledgements

First and foremost, I would like to praise and thank God, the almighty, who has granted countless blessing, knowledge, and opportunity to me, so that I have been finally able to accomplish the thesis.

I express my deep gratitude to my parents for their encouragement, support and for the sacrifices they endured on me.

I also would like to express my greatest thanks to my supervisor Pr. Mendil Boubekeur, for his support, help and encouragement during this thesis.

I also thank the members of the jury for the honor they do us by participating in the judgment of this work.

Special thanks to my wife for her continuous support and encouragement during the preparation of this thesis.

My greatest thanks to my brothers and sisters and all my family for their continuous support.

I would like to thank all my friends and those who helped me from near or far to accomplish this work

Contents

List of Figures

List of Tables

Introduction

Unmanned Aerial Vehicles (UAVs) have received considerable attention from research because of their wide range of applications. They have been successfully applied to many challenging civilian and defense applications such as surveillance, search and rescue, border and coast patrol, fire monitoring, etc. They are particularly useful for dull and dirty operations without exposing humans to dangerous situations. With rapid advances in electronics, sensors and communications, it is relatively easy to manufacture inexpensive and easy to operate UAVs, and hence they are expected to be widely deployed. The success of a UAV mission heavily depends on the employed algorithms. To increase the safety and reliability of UAVs, there is a need to develop more accurate algorithms which enhance their mission success rate [1].

A fundamental requirement of an autonomous vehicle is to travel from one location to another. And in order to travel between different locations, the vehicle has to track a planned trajectory quite accurately. Trajectory planning techniques are important for any mission because the success of the mission heavily depends on how the trajectory being tracked by the UAV, especially in cluttered environments [2].

Sampling-based motion planning are very known algorithms in robotics by their capability to solve high degree of freedom problems, such as PRM (Probabilistic Road Map) and RRT (Rapidly Exploring Random Trees). These methods explores the configuration space randomly and generates a set of optimal/sub-optimal waypoints. Then, the trajectory is generated from these waypoints for the UAV, taking into consideration the kinodynamic constraints. This trajectory can be an optimal one subject to an objective function [8].

This thesis focuses on 3D motion planning for UAVs, more specifically for Quadrotor type. Where it contributes in the both stages: the path planning and the trajectory planning. In the path planning we propose an algorithm that reduces the collision checking time in environment with high number of obstacles for sampling-based path planning. The algorithm disassociate the dependence of collision checking time with the number of obstacles, where it performs a preprocessing that stores information about the obstacles, then using them in the collision checking queries. Also, we propose a new variant of Rapidly Exploring Random Trees Star (RRT*) named NP-RRT* to deal mainly with narrow passages in the workspace and minimize the memory consumption. The algorithm uses a developed steer function to explore efficiently the configuration space. Also, a proposed path optimization technique is used to speed up the convergence rate to optimal solution by generating configurations near the found path.

In the trajectory planning we implemented the Particle Swarm Optimization (PSO) to minimize the energy by minimizing the snap and distance trajectory for Quadrotor. Where the snap is the forth derivative of the path, and the motor commands and attitude accelerations of the vehicle are proportional to the snap of the path.

The thesis is organized as follow:

- Chapter 1: gives a short history about UAVs and their structures, classifications and applications. Then it presents the concept of the guidance loop and trajectory planning.
- Chapter 2: exposes the dynamic modeling and the PD control of Quadrotor, then it presents simulation results.
- Chapter 3: presents the preliminaries and the basic concepts of trajectory planning. Also, it presents our proposed method in reducing the collision checking time for sampling-based motion planning.
- Chapter 4: presents the details of our new variant of RRT* named NP-RRT* algorithm and gives statistical comparisons.
- Chapter 5: presents the implementation of PSO in minimizing the snap and distance trajectory for Quadrotors.

Chapter 1

Unmanned Aerial Vehicles

1.1 Introduction

The aerial robotics is a vast and interdisciplinary field. A drone can be described as a flying machine without a human pilot on board. The flight of a drone may operate with various degrees of autonomy, either under remote control by a human operator or autonomously by onboard computers. Multiple terms are used for drones, generally referring to the same concept, such as UAV (Unmanned Aerial Vehicle) and UAS (Unmanned Aircraft System). Research and development related to autonomous drones is based on the ability to control their flight in hostile and complex environments, in order to perform repetitive or dangerous missions [2]. In this chapter we will present a brief short history about UAVs and their structures and classifications, also their applications and guidance loop.

1.2 Short history about UAVs

The first known autonomous flying machine has been credited to Archytas from the city of Tarantas or Tarentum in South Italy, known as Archytas the Tarantine. Archytas has been referred to as Leonardo da Vinci of the Ancient World. In 1483, he designed an aircraft capable of hovering, called aerial screw or air gyroscope (Fig. 1.1). The first widely recognized manned flight took place in 1783 using a hot air balloon designed by the Montgolfier brothers and commemorated (Fig. 1.2). In 1754, Russian Mikhail Lomonosov developed a complex model with two contra-rotating coaxial rotors driven by a clockwork

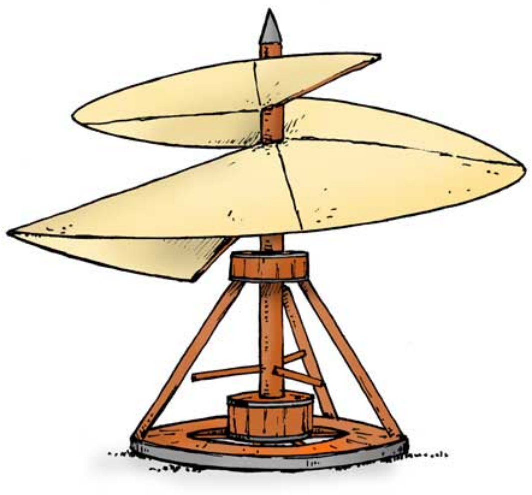


Figure 1.1: Aerial screw of Leonardo da Vinci 1483

mechanism (Fig. 1.2), the aircraft flew freely and reached a good altitude, this concept contributed to the development of the modern helicopter. The first real drone was made by the Americans Lawrence and Sperry in 1916. They developed a gyroscope to stabilize the drone in order to automate the piloting. This is known as the start of the concept of attitude control, which came to be used for autopilot. Then, in 1924, Etienne Oehmichen developed a helicopter that had four rotors and eight propellers (Fig. 1.3), all driven by a single engine. The aircraft exhibited a considerable degree of stability and increased control accuracy. This helicopters achieved 360m of hovering distance [1].

Since that time, the development of new UAVs gained an interesting attention and continued throughout World War II and the Cold War. The UAVs have been used mainly for military sector until around 1980s, their missions include reconnaissance, surveillance and attack missions. After that, their use has become wide spread, they can be classified according to their application for military or civil use.

1.3 UAVs Classification

During the last decades, significant efforts have been made in order to increase the flight endurance and the payload of UAVs, resulting in various UAV configurations with different sizes, endurance levels, and capabilities. Different UAV classification schemes have been proposed to help differentiate existing systems based on their operational charac-

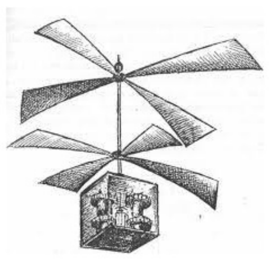


Figure 1.2: The left picture is the hot air balloon designed by the Montgolfier brothers 1783, and the right picture is the model of Mikhail Lomonosov 1754

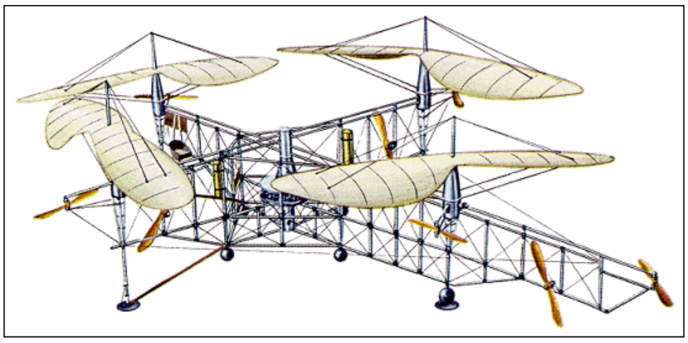


Figure 1.3: Etienne Oehmichen helicopter 1924

teristics and their capabilities. The UAVs can be classified based on their aerodynamics configuration into four categories [1]:

- **Fixed-wing UAVs:** which refer to unmanned airplanes (with wings) that require a runway to take-off and land, or catapult launching. These generally have long endurance and can fly at high cruising speeds.

- **Rotary-wing UAVs:** also called rotorcraft UAVs or vertical take-off and landing (VTOL) UAVs, which have the advantages of hovering capability and high maneuverability. These capabilities are useful for many robotic missions, especially in civilian applications. A rotorcraft UAV may have different configurations, with main and tail rotors (conventional helicopter), coaxial rotors, tandem rotors, multi-rotors.

- **Blimps:** such as balloons and airships, which are lighter than air and have long endurance, fly at low speeds, and generally are large sized.
- **Flapping-wing UAVs:** which have flexible and/or morphing small wings inspired by birds and flying insects.

Another common classification based on endurance and altitude to differentiate between UAVs [2]:

- **High Altitude Long Endurance (HALE) UAVs:** They are characterized by a wingspan close to that of a conventional aircraft, they can fly at an operational altitude of up to 20,000 meters with a range of several thousand kilometers and a range of about thirty hours. These planes have a large payload (Fig. 1.4).
- **Medium Altitude Long Endurance (MALE) UAVs:** They have an endurance of around 40 hours and can fly between 5,000 and 15,000 meters in altitude (Fig. 1.5).
- **Tactical UAVs:** They have a range of mission up to more than 100 km, a range of around 10 h, and can fly at an operational altitude of 200 to 5,000 meters (Fig. 1.6).
- **Mini Aerial Vehicles (MAV):** They have an endurance of a few hours and dimensions of the order of a meter, they can fly up to 300 m of altitude, operating at distances of up to about 30 kilometers carrying a very light payload.
- **Micro/Nano UAVs:** They refer to devices smaller than fifteen centimeters in size, and less than 5 g of mass. They are often equipped with propellers driven by electric motors, the autonomy is about 20 minutes for a range of mission 10 km (Fig. 1.7).

In 2005 the autonomous control levels (ACL) were proposed to classify the UAVs based on autonomy. Ten levels were proposed in [4] that were based on requirements like situational awareness, analysis, coordination, decision making, and operational capability. A list of the ACL is presented in Table 1.1.

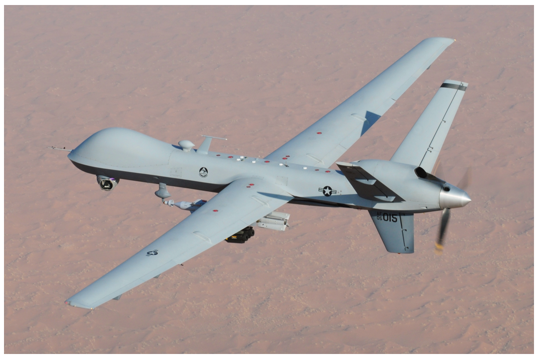


Figure 1.4: The General Atomics MQ-9 Reaper (Predator B) is a High Altitude Long Endurance UAV

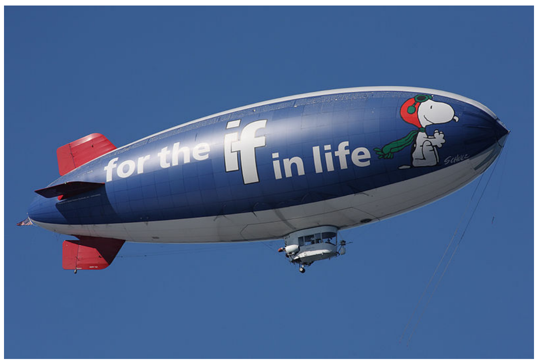


Figure 1.5: Metlife snoopy two blimp is a Medium Altitude Long Endurance

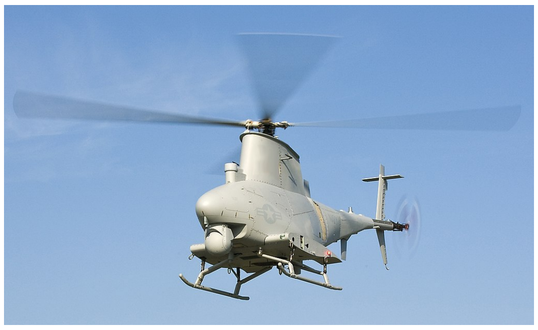


Figure 1.6: The Northrop Grumman MQ-8 Fire Scout is a Tactical UAV

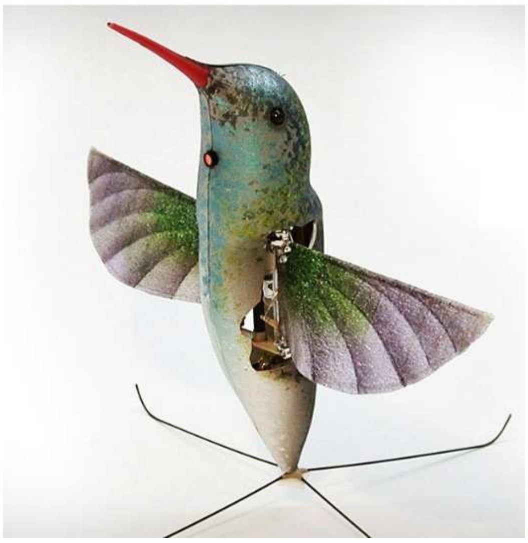


Figure 1.7: The Hummingbird flapping wing is a Nano UAV

Table 1.1: Autonomous control levels classification [4]

ACL	Level descriptor
0	Remotely piloted vehicle
1	Execute preplanned mission
2	Changeable mission
3	Robust response to real-time faults/events
4	Fault/event adaptive vehicle
5	Real-time multi-vehicle coordination
6	Real-time multi-vehicle cooperation
7	Battlespace knowledge
8	Battlespace cognizance
9	Battlespace swarm cognizance
10	Fully autonomous

1.4 UAVs Applications

Drone technology is now inexpensive and accessible, is continuously evolving and being put to several novel uses around the world. Initially known for their military use, drones are now being used in many civil domains to accomplish various tasks. Increasing work efficiency and productivity, decreasing workload and production costs, improving accuracy, refining service and customer relations, and resolving security issues on a vast scale are a few of the top uses.

Here we cite some of the main application fields for UAVs [1].

1.4.1 Agriculture

The agricultural uses of mini-drones are very diverse. Equipped with optical or hyper-spectral sensors, these machines make it possible to collect a large number of data, the processing of which will then facilitate the analysis of the evolution of crops and the identification of weeds or pests. Thus, regular monitoring can be guaranteed at a modest cost.

1.4.2 Inspection

The inspection of buildings, bridges or elements of an industrial architecture as well as oil pipelines or high voltage lines are all tasks to be carried out regularly to detect as quickly as possible a degradation or potential failure.

1.4.3 Civil security

Drones provide invaluable assistance in the immediate response to a natural disaster or accident. Again, they allow for the rapid deployment of sensors to gain information about the affected area. In addition to these advantages, they offer different views in collecting images compared to them from a satellite. This facilitates the post-processing of images.

1.4.4 Search and rescue

Presence of thermal sensors gives drones night vision and makes them a powerful tool for surveillance. Drones are able to discover the location of lost persons and unfortunate victims, especially in harsh conditions or challenging terrains. Besides locating victims, a drone can drop supplies to unreachable locations in war torn or disaster stricken countries. For example, a drone can be utilized to lower a walkie-talkie, GPS locator, medicines, food supplies, clothes, and water to stranded victims before rescue crews can move them to someplace else.

1.4.5 Geographic mapping

Available to amateurs and professionals, drones can acquire very high-resolution data and download imagery in difficult to reach locations like coastlines, mountaintops, and islands. They are also used to create 3D maps and contribute to crowd sourced mapping applications.

1.5 UAVs guidance and trajectory planning

1.5.1 The Guidance loop

In order to enable autonomous flight, it is necessary to use algorithms allowing the navigation, guidance and piloting tasks. Fig. 1.8 represents the loop in which a navigation, guidance and piloting algorithm is defined in a decoupled manner. The navigation task consists of estimating the information relating to the state of the vehicle (position, speed, orientation) and makes it possible to determine its location relative to a known frame of reference. The guidance system uses this information to follow a reference trajectory and provides for this purpose acceleration and attitude angle instructions, which will then be applied by the piloting loop. Subsequently, the control system provides a translation of the specifications emanating from the guidance in terms of the efforts to be made through the various actuators fitted to the vehicle. Several research works have been developed to build the guidance and control algorithms [3].

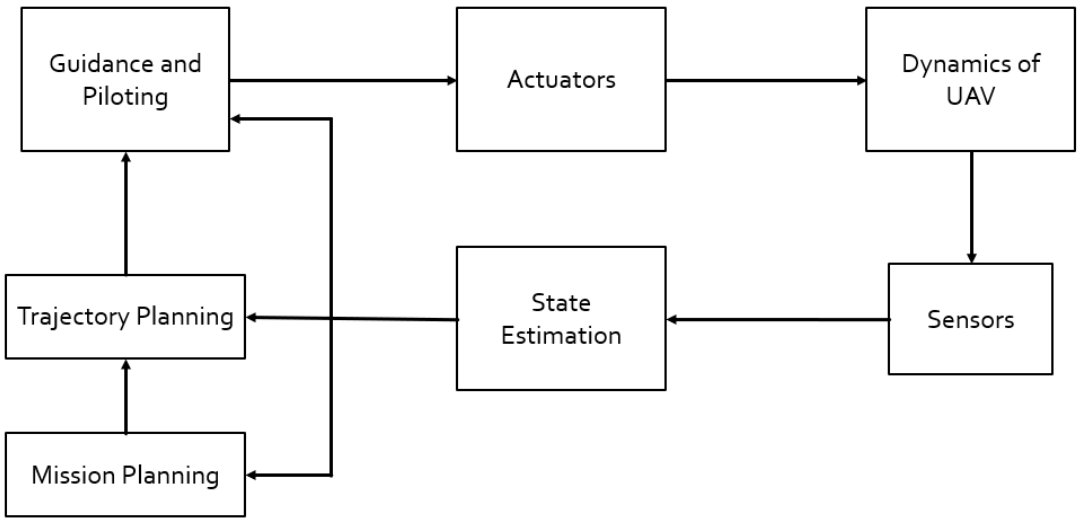


Figure 1.8: Guidance and piloting loop of UAVs

1.5.2 Trajectory planning for UAVs

The trajectory planning process is very important to accomplish the missions assigned to an UAV. This process has received considerable research attention, because following accurately the planned trajectory maximizes the likelihood of mission succession of a flying vehicle. The objective of the UAV trajectory planning is to find the optimum flight path that maximizes survivability while satisfying appropriate flight path constraints [5].

The trajectory planning goes through two stages: firstly, a path planning algorithm generate a set of waypoints from starting point to end point, then a trajectory planning generates from these waypoints three signals for each input (position, velocity and acceleration). Then the UAV tracks these signals using its controller.

1.6 Conclusion

This chapter have been presented a brief history of UAVs and the appearance of the first rotary wing drone. Also, it has presented the different classifications of UAVs, and their applications. The last part outline the guidance and piloting loop, and the concept of trajectory planning. This thesis focuses on the Quadrotor type, the next chapter will expose the modeling and control of the Quadrotor.

Chapter 2

Quadrotor modeling and control

2.1 Introduction

The area of small UAVs has seen a lot of exciting developments during the last fifteen years. Because of their small size, they can maneuver in indoor or outdoor environments easily and collect information using onboard sensors. Several domains used small UAVs to accomplish tasks like photography, whether monitoring, surveillance activities, agriculture missions, and many other tasks. The Quadrotor is the rotorcraft type mostly used. Its propulsive force is provided by four rotors. This makes it a flexible and adaptable platform for aerial robotics [2]. Examples of developed research platforms are: OS4, STARMAC, Pixhawk, Hummingbird, and Parrot ARDrone [6].

2.2 General description of Quadrotor

A Quadrotor is a multi-rotor helicopter that is lifted by four rotors. It is classified as rotorcraft, because it can vertically take off and land (VTOL). The four rotors are usually placed at the ends of a cross, and the electronic control board is usually placed in the center of the cross. To prevent the device from turning on itself around z axis, two propellers must turn in one direction, and the other two in the other direction. The operation of the Quadrotor is quite special. By varying the power of the motors, it is possible to make it go up / down, tilt it left / right (roll) or forward / backward (pitch) or even rotate it on itself (yaw). The Quadrotor has six degrees of freedom, three rotational movements and three translational movements, these six degrees must be controlled using only four triggers, therefore, it is an underactuated system (the number of inputs less than the number of outputs).

2.3 The movements of Quadrotor

In conventional helicopters, when the main rotor spins, it produces reactive torque that would cause the helicopter body to spin in the opposite direction. This problem is solved by adding a tail rotor which produces thrust in a lateral direction. However, this rotor has no contribution to the thrust. In other hand, in the case of a Quadrotor, two motors spin in clockwise direction and the two others spin in counter clockwise direction (Fig. 2.1). This effectively neutralizes unwanted reactive torque and allows the vehicle to hover without turning out of control [?].

The Quadrotor has 3 translational movements and 3 rotational movements generated by four input signals which are the thrust of the motors [2]

- **Translational movements:**

  The vertical movement along the z_f axis of the fixed frame, i.e., altitude movement, is obtained by increasing or decreasing the speed of the four motors with the same amount simultaneously. If the lift force is greater than the weight of the Quadrotor, the movement is upward, and if it is less, the movement is downward. The movement along x_f or y_f axis is generated by tilting the Quadrotor with a pitch angle or a roll angle from the horizontal.

- **Rotational movements:**

  The roll movement is obtained by applying a torque around the x_b axis, i.e., applying a difference in thrust between the rotor 2 and the rotor 4. The pitch movement is obtained by applying a torque around the y_b axis, i.e., applying a difference in thrust between the rotor 1 and the rotor 3. The yawing movement is obtained by applying a torque around the z_b axis, i.e., applying a difference in thrust between the rotors 1,3 and the rotors 2,4.

2.4 Quadrotor modeling

Modeling flying robots is a difficult task since the dynamics of the system is strongly nonlinear. In order to simplify the study, the dynamic model below is developed under these hypotheses [12]

– The structure of the Quadrotor is assumed to be rigid and symmetrical, which implies that the inertia matrix will be assumed to be diagonal.

– The propellers are supposed to be rigid in order to be able to neglect the effect of their deformation during rotation.

– The lift and drag forces are proportional to the squares of the rotational speed of the rotors.

– All friction forces and torques are neglected.

Using the Newton-Euler formulation, the dynamics of the Quadrotor can be written by following equations as described in [2]. The equation of the acceleration of the center of mass is:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathcal{R}_F^B \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \tag{2.1}$$

Where:

- $\mathbf{r} = [x, y, z]^t$ is the position vector coordinates of the center of mass of the Quadrotor in the fixed frame $(\mathcal{F})$, and $\mathcal{R}_F^B$ is the rotation matrix according to $Z - X - Y$ Euler angle convention between the body fixed frame $(\mathcal{B})$ and the world fixed frame $(\mathcal{F})$. Rotations are made around the instant moving axes:

$$\mathcal{R}_F^B = Rot_Z(\psi) Rot_X(\phi) Rot_Y(\theta) \tag{2.2}$$

$$\mathcal{R}_F^B = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix}$$

Where c and s denote the trigonometric functions *cos* and *sin* respectively

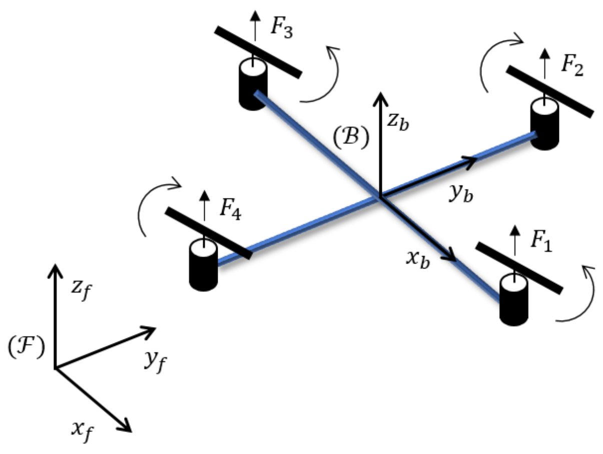


Figure 2.1: Reference frames and spin directions of the motors

- u_1 is the input signal which controls the displacement along the z_b axis expressed as a function of the thrust forces generated by the four propellers :

$$u_1 = F_1 + F_2 + F_3 + F_4 \tag{2.3}$$

  Where $F_i = b\omega_i^2$, $i = 1..4$, b is the thrust coefficient and ω_i is the motor angular speed.

The equation of rotational movement is:

$$I\dot{\mathbf{h}} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \mathbf{h} \times I\mathbf{h} \tag{2.4}$$

Where

- the sign $(\times)$ denote the cross product.
- $\mathbf{h} = [p, q, w]^t$ is the vector of the angular velocities of the Quadrotor in the body fixed frame $(\mathcal{B})$, These values are related to the derivatives of the angles in the fixed frame $(\mathcal{F})$ $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$ according to following equation:

$$\begin{bmatrix} p \\ q \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + Rot_Y(\theta)^{-1} \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + (Rot_X(\phi)Rot_Y(\theta))^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

$$\begin{bmatrix} p \\ q \\ w \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.5}$$

- I is the inertia matrix supposed to be symmetric

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.6}$$

- u_2,u_3 are the inputs signals that controls roll and pitch angles respectively

$$\begin{aligned} u_2 &= L(F_2 - F_4) \\ u_3 &= L(F_3 - F_1) \end{aligned} \tag{2.7}$$

  Where L is the distance between the center of mass of the Quadrotor and the axis of a motor.

- u_4 is the control input for the yawing movement written in function of drag forces generated by the motors

$$u_4 = M_1 - M_2 + M_3 - M_4 \tag{2.8}$$

  Where $M_i = d\omega_i^2$, $i = 1..4$, d is drag coefficient

By replacing the vector of angular velocities **h** with its components we get:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ w \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ w \end{bmatrix} \tag{2.9}$$

By developing the last equation we get the following equation:

$$\begin{aligned} I_{xx}\dot{p} &= u_2 - qw(I_{zz} - I_{yy}) \\ I_{yy}\dot{q} &= u_3 - pw(I_{xx} - I_{zz}) \\ I_{zz}\dot{w} &= u_4 \end{aligned} \tag{2.10}$$

Note that for the third line of the last equation, we got only u_4 this is because $I_{xx} = I_{yy}$ since the Quadrotor is considered to be symmetric.

2.5 PD Controller of Quadrotor

The proportional, integral, derivative (PID) controller has been widely used in industry and academy. This success is a result of many good features of this controller such as simplicity, robustness and wide applicability.

The Quadrotor is controlled by nested feedback loops as shown in Fig. 2.2. The inner attitude control loop uses onboard sensor to control the roll, pitch, and yaw, while the outer position control loop uses estimates of position and velocity of the center of mass to track the desired trajectory.

A trajectory for the Quadrotor is composed of four coordinates $[\mathbf{r_T}(\mathbf{t}), \psi_T(t)]$. This is used by the controller to compute the desired inputs. In this controller we assume that the Quadrotor is operating around hover conditions, $\theta \approx 0$, $\phi \approx 0$, $\psi = \psi_0$, $\dot{\theta} \approx 0$, $\dot{\phi} \approx 0$, $\dot{\psi} \approx 0$.

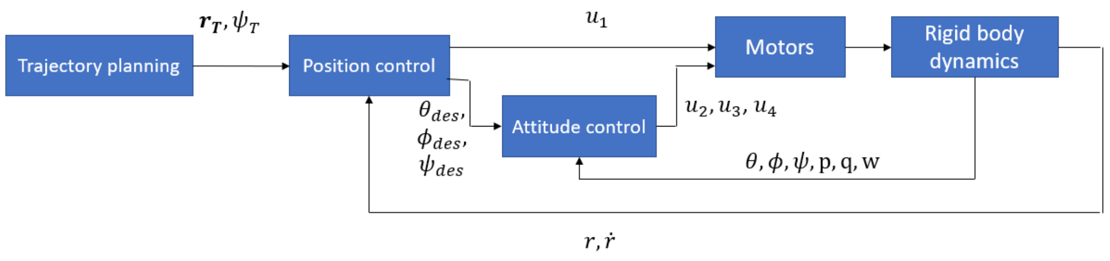


Figure 2.2: The nested control loop for attitude and position control

2.5.1 Attitude control

The attitude controller allows to track trajectories in SO(3) that are close to the nominal hover state where the roll and pitch angles are small. From equation 2.10, if we assume that the products of inertia are small, and under the assumption of nominal hover state $\dot{\phi} \approx p$, $\dot{\theta} \approx q$, and $\dot{\psi} \approx w$, we can use simple proportional derivative control laws as follows:

$$\begin{aligned}
u_2 &= k_{p\phi}(\phi_{des} - \phi) + k_{d\phi}(p_{des} - p) \\
u_3 &= k_{p\theta}(\theta_{des} - \theta) + k_{d\theta}(q_{des} - q) \\
u_4 &= k_{p\psi}(\psi_{des} - \psi) + k_{d\psi}(w_{des} - w) \qquad (2.11)
\end{aligned}$$

2.5.2 Position control

The position of the Quadrotor in x_f,y_f plan is controlled by pitch and roll angle, and the position along z_f is controlled by u_1. The command accelerations $\ddot{r}_{des}$ are calculated from the PD feedback of the position error as follow:

$$\mathbf{\ddot{r}_{des}} = \mathbf{\ddot{r}_T} + \mathbf{K_d}(\mathbf{\dot{r}_T} - \mathbf{\dot{r}}) + \mathbf{K_p}(\mathbf{r_T} - \mathbf{r}) \qquad (2.12)$$

where $\mathbf{K_p}$ and $\mathbf{K_d}$ are 3x3 diagonal matrices contains the proportional and derivative gains respectively.

The relationship between the desired accelerations and roll and pitch angles can be obtained from the equation 2.1 after the linearization:

$$\begin{aligned}
\ddot{x}_{des} &= g(\theta_{des} cos(\psi_T) + \phi_{des} sin(\psi_T)) \\
\ddot{y}_{des} &= g(\theta_{des} sin(\psi_T) + \phi_{des} cos(\psi_T)) \\
\ddot{z}_{des} &= \frac{u_1}{m} - g \qquad (2.13)
\end{aligned}$$

From these relationships the desired roll and pitch angles are computed for the attitude controller as follow:

$$\begin{aligned}
\phi_{des} &= \frac{1}{g}(\ddot{x}_{des} sin(\psi_T) - \ddot{y}_{des} cos(\psi_T)) \\
\theta_{des} &= \frac{1}{g}(\ddot{x}_{des} cos(\psi_T) + \ddot{y}_{des} sin(\psi_T)) \qquad (2.14)
\end{aligned}$$

The complete altitude and attitude control inputs can be written as follow:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} m(\ddot{z}_{des} + g) \\ k_{p\phi}(\phi_{des} - \phi) + k_{d\phi}(p_{des} - p) \\ k_{p\theta}(\theta_{des} - \theta) + k_{d\theta}(q_{des} - q) \\ k_{p\psi}(\psi_{des} - \psi) + k_{d\psi}(w_{des} - w) \end{bmatrix} \tag{2.15}$$

2.5.3 PD tuning

Many various tuning methods have been proposed in the literature for gaining better and more acceptable control system response based on desired control objectives.

In our case, we are using just the PD terms of the controller. The proportional and derivative gains from the equation 2.12 are diagonal matrices. In order to tune the PD gains, we have separated the system. Firstly, we tried to fix the gains for the altitude control on z axis. Then, we moved to attitude control, we consider the Quadrotor operating in the plan z,y so the rotation can only be done around x axis. We tried to fix the gains in this case for the attitude for ϕ angle. Since the simulation's conditions are considered to be ideal, so the same gains found for ϕ angle are used for θ angle. Lastly, we have tuned the gains for the attitude control for ψ angle [13].

After tuning all the gains, these following gains are used for simulation

$$\mathbf{K_p} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3.4 \end{bmatrix}, \mathbf{K_d} = \begin{bmatrix} 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 3.02 \end{bmatrix}$$

$$k_{p\phi} = k_{p\theta} = k_{p\psi} = 3.5$$

$$k_{d\phi} = k_{d\theta} = k_{d\psi} = 3$$

2.6 Quadrotor simulation

In order to test the presented PD controller, we have built a Simulink model for the Quadrotor's dynamics in MATLAB software (version 2014a) as shown in Fig. 2.3. The

Table 2.1: Quadrotor's parameters used for simulation [7]

Designation	Parameter	Value
mass of Quadrotor	m	$0.95kg$
acceleration due to gravity	g	$9.81m.s^{-1}$
drag factor	d	$7.5 \times 10^{-7}N.m.s^2$
thrust factor	b	$3.13 \times 10^{-5}N.s^2$
inertia moments in x and y axis	$I_{xx} = I_{yy}$	$5 \times 10^{-3}kg.m^2$
inertia moment in z axis	I_{zz}	$10^{-2}kg.m^2$
distance between rotor's center to CoG	L	$0.22mm$

simulation goes through two stages, first we test the model with the controller giving to the system step inputs, then, secondly we test the tracking of 3D reference trajectory by the system. For the simulations we assume that the Quadrotor is operating around the equilibrium stat, $\theta \approx 0$, $\phi \approx 0$, $\psi = \psi_T$. The parameters of the Quadrotor used in this simulation are presented in Table 2.1. All the simulations are performed on a computer with Intel Core i3 2.30 GHz, and 4 GB of RAM.

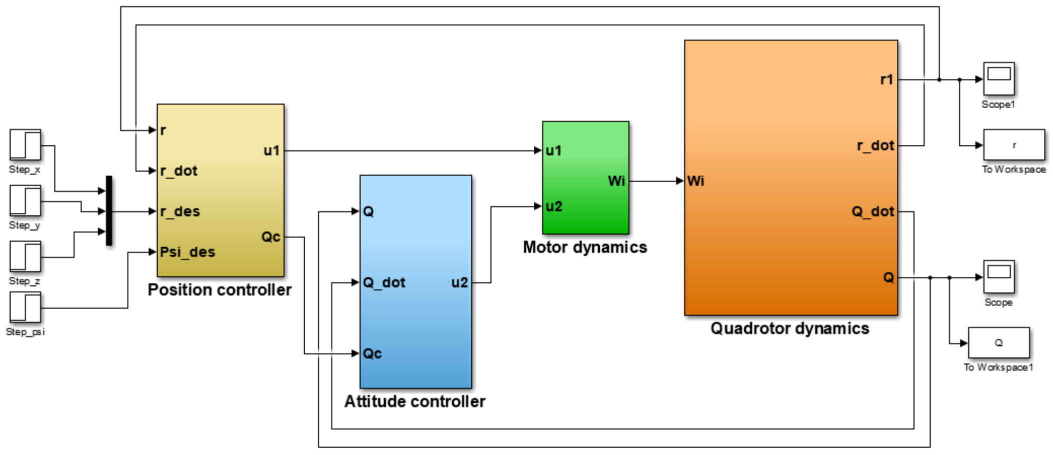
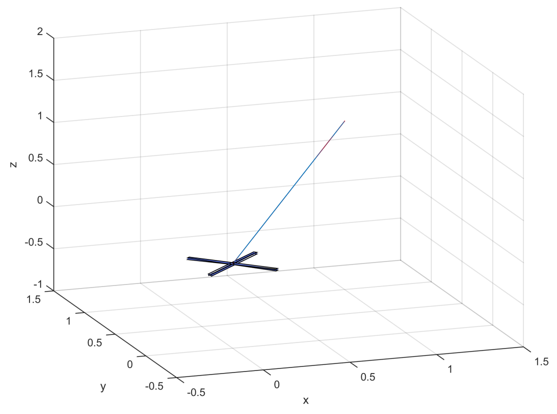


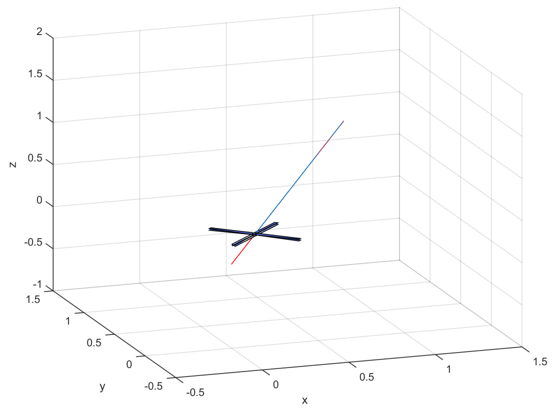
Figure 2.3: The Simulink bloc diagram model of the Quadrotor in MATLAB

For the first stage, Fig. 2.5 shows the step responses of the Quadrotor. Where the value of the step for each input are $[x, y, z, \psi] = [1, 1, 1, 0.02]$. The responses for the position and the angles coordinates are quite good with little overshot for z response. However, the rising time is long, where it take 5.5s to reach 1.

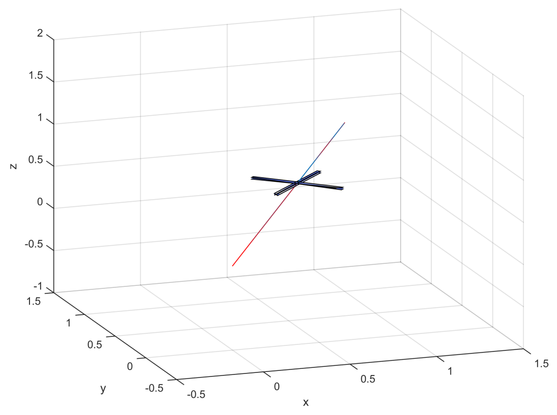
For the second stage, we generated a 7th order polynomial trajectory from point $[x, y, z, \psi] = [0, 0, 0, 0]$ to $[x, y, z, \psi] = [1, 1, 1, 0.1]$. The polynomial is the same for x,y and z coordinates. Fig. 2.4 shows the Quadrotor tracks this 3D trajectory. The responses of the system are plotted in Fig. 2.6 . From these responses it is clear that the PD controller succeeded to track the generated trajectory with small errors.



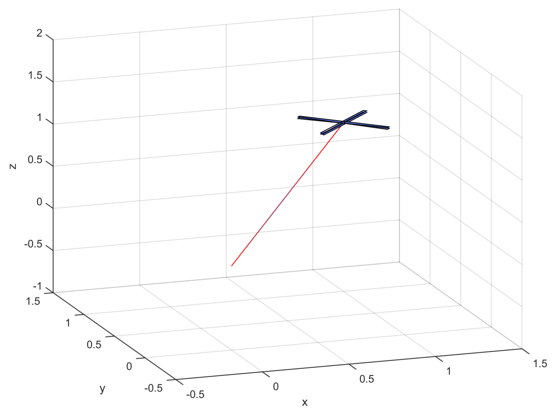
(a)



(b)

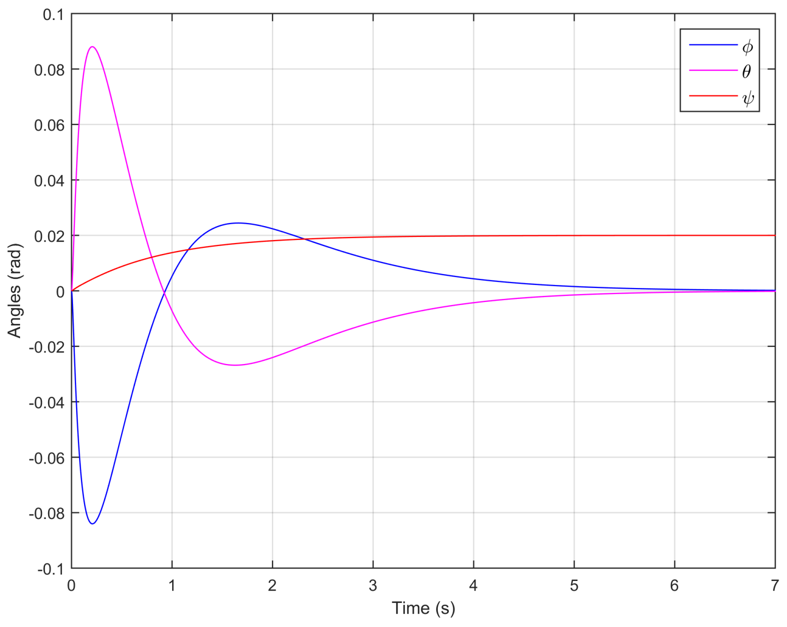


(c)



(d)

Figure 2.4: 3D simulation of Quadrotor tracking the trajectory from point $(x, y, z, \psi) = (0, 0, 0, 0)$ to $(x, y, z, \psi) = (1, 1, 1, 0.1)$



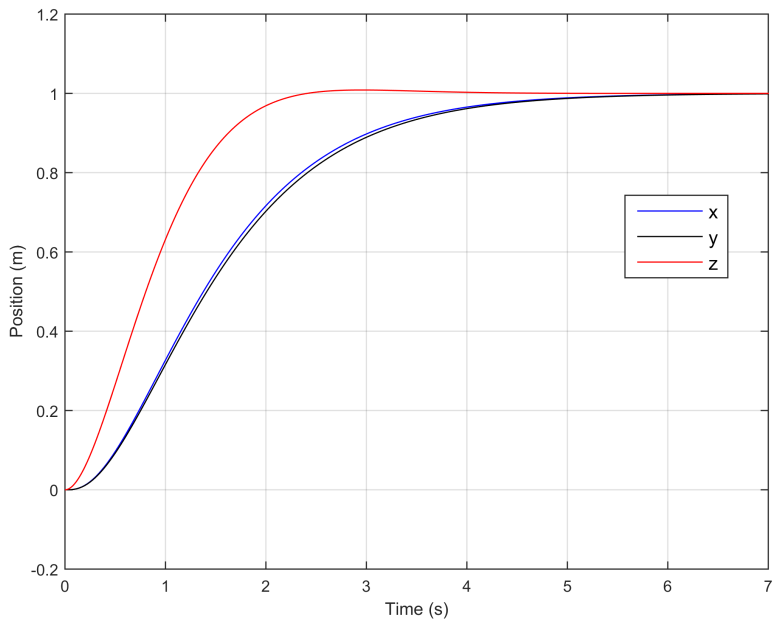
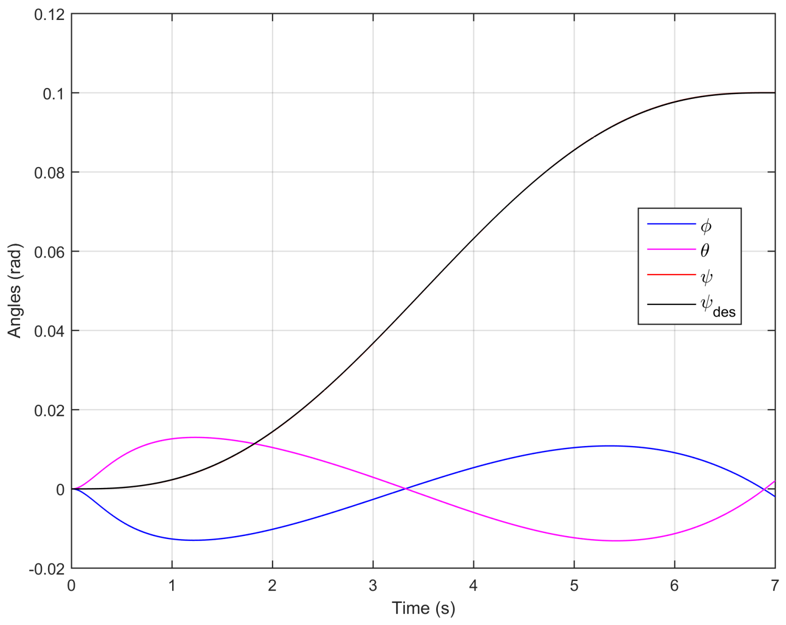


Figure 2.5: Step responses for Quadrotor dynamics: on the top the attitude responses ϕ, θ, ψ; on the bottom the position responses x, y, z



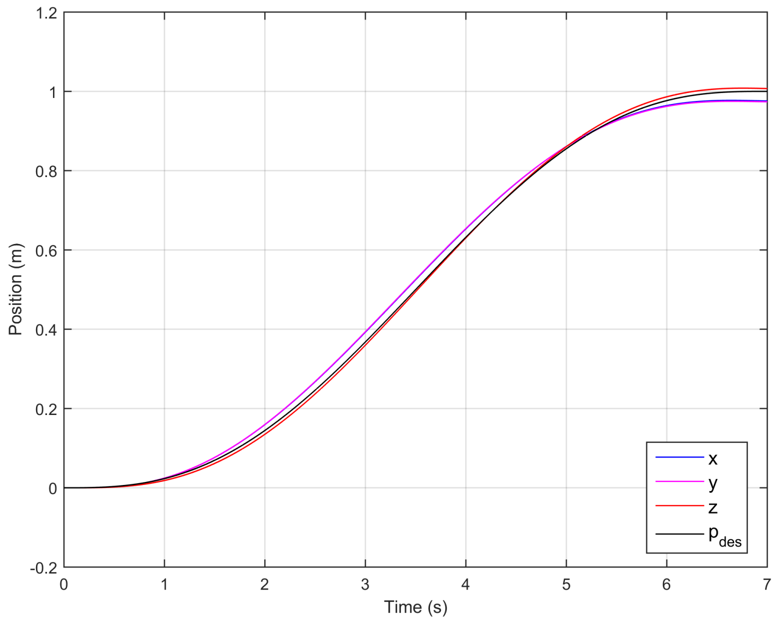


Figure 2.6: The output signals of the Quadrotor tracking a 3D trajectory from $(x, y, z, \psi) = (0, 0, 0, 0)$ to $(x, y, z, \psi) = (1, 1, 1, 0.1)$: on the top the attitude responses; on the bottom the position responses

2.7 Conclusion

In this chapter, we have presented the modeling and control of Quadrotor aircraft. Where the PD controller was used to track desired trajectories. Simulation of Quadrotor dynamics in MATLAB was presented with output responses. The next chapter will focus on motion planning for Quadrotors.

Chapter 3

Trajectory planning for Quadrotors

3.1 Introduction

Accomplishing tasks by UAVs in real environments, needs an important module in their piloting loop, which is the trajectory planning module. In order to move the UAV from starting point to end point, the trajectory planning module ensures finding a feasible, collision-free, and shortest safe path, then generates consign signals inputs for the controller of the UAV. This trajectory is most likely to be the optimal solution subject to an objective function and possibly some constraints.

Since the UAVs are operating in 3D environments, 2D planning algorithms are not the qualified methods. Path planning in 3D environment shows great prospect, but unlike 2D path planning, the complexity increases exponentially with kinematic constraints. In order to plan a collision-free path through a cluttered environment, modeling the environment while taking the kinematic constraints into consideration is the challenge that needs to be solved. From the optimization point of view, finding a 3D optimal path planning problem is NP-hard, thus, common solutions are not existing [11].

This chapter presents the basic terminologies and the mathematical notations for motion planning. Also, it presents the two sampling-based algorithms that are used in this thesis. Finally, a new collision checking strategy that we have proposed is presented with simulation results.

3.2 Definitions and mathematical model

To plan the movement of a robot we must be able to specify its position and orientation in the space. More precisely, we must be able to specify the position of each point of the robot, and ensure that it does not collide with an obstacle. The following concepts and notations will help us to define the geometric model of the environment, the robot, the obstacles and the path.

3.2.1 The workspace

The workspace $\mathcal{W}$ is the set of all the positions of a robot $\mathcal{M}$. $\mathcal{W}$ can contain a set of n obstacles, which can be expressed by the notation $\mathcal{WO}$. The robot $\mathcal{M}$ is physically present in $\mathcal{W}$. A position in $\mathcal{W}$ is reachable if $\mathcal{M}$ does not collide with an obstacle. if $\mathcal{W}$ is a 2D space, $\mathcal{M}$ acts in the plane, and if $\mathcal{W}$ is a 3D space, $\mathcal{M}$ acts in the space [8]. The set of reachable positions $\mathcal{W}_{free}$ by $\mathcal{M}$ is defined by the equation 3.1.

$$\mathcal{W}_{free} = \mathcal{W} \setminus \bigcup_{1}^{n} \mathcal{WO}_i \tag{3.1}$$

where the sign $\setminus$ is a subtraction operator.

3.2.2 The configuration space

To standardize the description of the workspace, $\mathcal{W}$ is transformed into Configuration space $\mathcal{Q}$. This transformation into configuration space makes it possible to replace the search for a path of $\mathcal{M}$ in $\mathcal{W}$ by the search for the path of a point in $\mathcal{Q}$. The robot $\mathcal{M}$ has n degrees of freedom, a configuration is defined by a vector $q = (q_1, q_2, \ldots, q_n)$ composed of the set of n values of the degrees of freedoms of $\mathcal{M}$. $\mathcal{Q}$ is defined by the set of all possible vectors q [8]. The free space $\mathcal{Q}_{free}$ in the configuration space is defied by the equation 3.2.

$$\mathcal{Q}_{free} = \mathcal{Q} \setminus \bigcup_{1}^{n} \mathcal{Q}_{obs} \tag{3.2}$$

Where $\mathcal{Q}_{obs}$ is the obstacle projection in $\mathcal{Q}$.

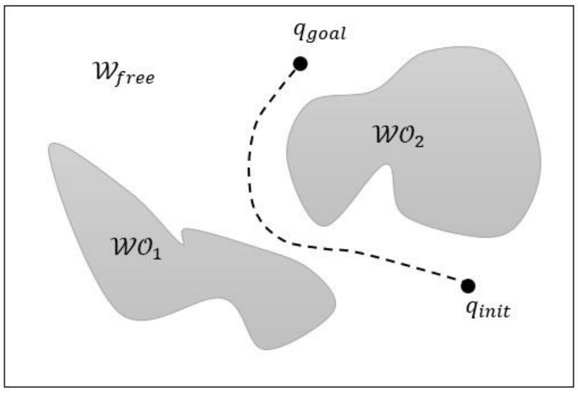


Figure 3.1: Example of a planned path from q_{init} to q_{goal} in a workspace with two obstacles.

3.2.3 The problem of path planning

To make the robot move from one point to another, we have to find a succession of admissible configurations allowing it to achieve its objective. The path planning problem corresponds to the search for a sequence of configurations belonging to $\mathcal{Q}_{free}$ from the start q_{init} to the goal q_{goal}. The solution in the configuration space is a continuous function $c(s) \in C^0$ [8], such that:

$$c : [0,1] \rightarrow \mathcal{Q} \text{ where } c(0) = q_{init}, c(1) = q_{goal} \text{ and } c(s) \in \mathcal{Q}_{free} \forall s \in [0,1] \qquad (3.3)$$

3.2.4 Optimal path planning problem

According to [9] the optimal path planning can be formulated as follows: Let $\mathcal{Q} \subset \mathbf{R}^d$ be the configuration space, where d is the space dimension. And, let $c : [0,1] \mapsto \mathcal{Q}$ be a sequence of configurations that forms a path, and Σ be the set of all nontrivial paths. The optimal path $c^* \subset \mathcal{Q}_{free}$ that connect q_{init} to q_{goal}, and minimize a given cost function $\sigma : c \mapsto \mathbf{R}^+$ is given by this equation

$$c^* = \arg\min_{c \in \Sigma} \{ \ \sigma(c) \mid c(0) = q_{init}, \ c(1) = q_{goal}, \ \forall s \in [0,1], \ c(s) \in \mathcal{Q}_{free} \} \qquad (3.4)$$

3.2.5 Path planning and trajectory planning

Path planning and trajectory planning problems are two distinct parts of robotics, but they are intimately related. Path planning provides a geometric description of the robot's movement, but it does not specify the dynamic aspects of the movement. Whereas trajectory planning is the parametrization of the path c by time t, then velocities and accelerations can be computed by taking derivatives with respect to time of $c(t)$. This means that c should be at least twice-differentiable, i.e., in the class C^2. Trajectory planning can be referred also by motion planning [10].

3.2.6 Motion planning terminology

A motion planning algorithm is considered to be complete if and only if it's able to find a path when exists, and can detect that no path exists in finite time. It is considered to be optimal when it returns the optimal path with respect to some criterion. Two forms of completeness and optimality are also commonly used: resolution completeness with optimality, and probabilistic completeness with optimality. Resolution completeness is related to the discretization of the solution space, and means that as the resolution of the discretization increases, an exact solution is achieved. Probabilistic completeness means that as computing time approaches infinity, the probability of finding an exact solution approaches one [10].

3.3 Sampling based motion planning

The main idea of sampling-based motion planning is to avoid the explicit construction of $\mathcal{Q}_{free}$, instead, it conducts a search that probes the configuration space with a sampling scheme. This probing is enabled by a collision detection module, which enables the development of planning algorithms that are independent of the particular geometric models. The collision detection module handles concern such as whether the models are semi-algebraic sets, 3D triangles, nonconvex polyhedron, and so on. This general philosophy has been very successful in recent years for solving problems from robotics, manufacturing, and biological applications that involve thousands and even millions of geometric primitives. Such problems would be practically impossible to solve using techniques that explicitly represent $\mathcal{Q}_{free}$ [11].

The crucial feature of sampling algorithms is randomness, where the samples of configurations are generated randomly in $\mathcal{Q}$. Then, a local planner is used to connect between samples by local paths. This information is stored in a graph, where its nodes represent the samples and its edges represent local paths. The shortest path in $\mathcal{Q}$ can be found using an algorithm that finds the shortest path in the graph between the start and the end nodes.

The most two known sampling-based algorithms in the literature are PRM [14] (Probabilistic Road Maps), and RRT [15] (Rapidly Exploring Random Trees). PRM is a multi-queries algorithm, this means that it builds the graph one time then it answers planning quires. RRT is a single query algorithm, this means that it rebuilds the graph for each planning query. These two methods have seen many developments. The optimal versions of these methods have been published in 2011, under the names PRM* and RRT* [9]. In this thesis we will focus on RRT* algorithm, where we propose a new efficient variant. The details will be explained in next chapter. Also, we will use the PRM algorithm for testing our proposed collision checking strategy. The details will be presented in the next section.

3.3.1 The basic PRM

The probabilistic roadmap [14] was the first proposed sampling based-algorithm. It proceeds in two phases: learning phase and query phase. In the learning phase, the sampled configurations from $\mathcal{Q}$ are stored in a graph data structure called a roadmap. The latter is constructed in a probabilistic way. The roadmap is a unidirectional graph $G = (V, E)$, where V is a set of free configurations sampled randomly and E is a set of edges corresponding to local paths. For every configuration $q \in V$, a set N_q of k closest neighbors to the configuration q is chosen from V according to a metric distance $dist$. Then, the local planner is called to connect q to each configuration $q' \in N_q$. If the local planner Δ succeeds to compute a feasible path between q and q', the edge (q, q') is added to the roadmap.

In the query phase, the initial and the end configurations q_{init}, q_{goal} are connected to the roadmap using the local planner then an algorithm of finding the shortest path in the graph, like A^* or Dijkstra, is used to find the path between the start and the end configuration. A post-processing step can be performed to improve the quality of resulting path by checking whether nonadjacent configurations q and q' along the path can be connected with the local planner.

Algorithm 1 Learning phase of PRM

```
1: **Input:**
2: $nn$ : number of nodes to put in the roadmap
3: $k$ : number of closest neighbors to examine for each configuration
4: **Output:**
5: A roadmap $G(V, E)$
6: **while** size of $V < nn$ **do**
7:     **while** $q$ in collision **do**
8:         $q \leftarrow$ random configuration from $\mathcal{Q}$
9:     **end while**
10:    $V \leftarrow V \cup q$
11: **end while**
12: **for all** $q \in V$ **do**
13:    $N_q \leftarrow$ the $k$ closest neighbors of q chosen from $V$ according to $dist$
14:    **for all** $q' \in N_q$ **do**
15:        **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq NIL$ **then**
16:            $E \leftarrow E \cup \{(q, q')\}$
17:        **end if**
18:    **end for**
19: **end for**
```

3.3.2 The basic RRT*

RRT* [9] build incrementally a tree $G(V, E)$ of feasible paths rooted at q_{init}. Such that V is the set of nodes that forms the tree, and E is the set of edges of G. In each iteration, a configuration q_{rand} is sampled randomly from Q, then q_{rand} is rejected if it lies in Q_{obs}. However, if q_{rand} is in Q_{free}, a nearest node q_{nrst} is searched from G, according to a metric. If the path between q_{nrst} and q_{rand} is in collision, then a steer function is used to generate a configuration q_{new}, such that the path between q_{nrst} and q_{new} is collision free. Now, within the radius r_{RRT^*} from q_{new}, the nearest neighbor which has minimum cost is connected as parent to q_{new}.

Algorithm 2 Query phase of PRM

```
1: **Input:**
2: $q_{init}$ : the initial configuration
3: $q_{goal}$ : the goal configuration
4: $k$ : number of closest neighbors to examine for each configuration
5: $G(V, E)$ : the roadmap constructed by the Algorithm 1
6: **Output:**
7: A path from $q_{init}$ to $q_{goal}$ or failure
8: $V \leftarrow V \cup \{q_{init}\} \cup \{q_{goal}\}$
9: for $q \leftarrow \{q_{init}, q_{goal}\}$ do
10:    $N_q \leftarrow$ the $k$ closest neighbors of $q$ chosen from $V$ according to $dist$
11:    $q' \leftarrow$ the closest neighbor of $q$ from $N_q$
12:    while $N_q$ is not empty do
13:       if $\Delta(q, q') \neq NIL$ then
14:          $E \leftarrow E \cup (q, q')$
15:       end if
16:       $N_q \leftarrow N_q - \{q'\}$
17:       $q' \leftarrow$ the closest neighbor of $q$ from $N_q$
18:    end while
19: end for
20: $path \leftarrow$ The shortest path in $G$ from $q_{init}$ to $q_{goal}$
21: if $path$ is empty then
22:    return failure
23: else
24:    return $path$
25: end if
```

$$r_{RRT^*} = \gamma \left(\frac{\log(n)}{n} \right)^{1/d} \tag{3.5}$$

where n is the number of nodes in the tree, γ is the constant of planning, dependent on the environment, and d is the configuration space dimension.

A rewiring procedure is used in each iteration to identify whether q_{new} can be a parent of one of the nearest neighbors within the radius r_{RRT^*}. This procedure improves the quality of paths. Indeed, the processes of selecting the minimum cost nearest neighbor and rewiring tree are the crucial features of RRT*.

For all the algorithms in this thesis we are using some procedures explained as follows:

`Sample`: creates a random configuration q_{rand} in Q_{free} space.

`Nearest`: returns the closest node q_{nrst} in the tree G to q_{rand}.

`Steer`: returns a configuration q_{new} on the straight line (q_{nrst}, q_{rand}) if the path between q_{nrst} and q_{rand} is in collision. Else, it returns q_{rand}.

`Near`: returns the closest nodes Q_{near} to q_{new} in the tree G according to r_{RRT^*}.

`CollisionFree`: this function checks the collision for a given configuration or for a path between two given configurations. It returns $True$ if there is no collision, else, it returns $False$.

`BestParent`: returns the best cost node q_{min} in Q_{near}.

`RewireTree`: this function checks if q_{new} can be parent of a node in Q_{near}.

3.4 New collision checking strategy

Given a configuration q in $\mathcal{Q}$, the collision checking module returns a boolean $true$ if the q is in collision, and returns $false$ if q is collision-free. In sampling-based motion planning, the collision checking module called many times and this operation takes the most planning time. The time complexity of the collision checking increases exponentially in function of the number of obstacles. Thus, this module is the critical part of sampling methods. A variety of collision detection algorithms exist, ranging from theoretical algorithms that have excellent computational complexity to heuristic.

Algorithm 3 Basic RRT*

```
1: $V \leftarrow \{q_{init}\}$ ; $E \leftarrow \emptyset$;
2: for $i = 1$ to $N$ do
3:     $q_{rand} \leftarrow$ Sample($Q$);
4:     $q_{nrst} \leftarrow$ Nearest($G = (V, E)$, $q_{rand}$);
5:     $q_{new} \leftarrow$ Steer($q_{nrst}$, $q_{rand}$);
6:     if CollisionFree($q_{nrst}$, $q_{new}$) then
7:         $Q_{near} \leftarrow$ Near($q_{new}$, $G = (V, E)$, $r_{RRT^*}$);
8:         $q_{min} \leftarrow$ BestParent($q_{new}$, $Q_{near}$);
9:         $V \leftarrow V \cup \{q_{new}\}$; $E \leftarrow E \cup \{(q_{new}, q_{min})\}$;
10:        $G \leftarrow$ RewireTree($q_{new}$, $Q_{near}$, $G = (V, E)$);
11:    end if
12: end for
13: return $G = (V, E)$;
```

Many algorithms have been proposed to reduce the number of calls to collision checking procedure, in order to minimize the planning time. The most of them use prior information about the sampled configurations [16] [17] or collision queries [18] [19]. In other words, they modify the sampling strategy to reduce the complexity of the approach.

Our contribution in this thesis is an approach of collision checking for sampling-based motion planning [20]. The approach addresses mainly the problems with high number of obstacles. The next sub-sections presents the details of the method.

3.4.1 Collision checking in cluttered environment

Collision testing takes the most of time in path planning, especially with high number of obstacles, as the number of obstacles increase as the complexity augment. Our main contribution is to disassociate the dependence of collision testing complexity with the number of obstacles in the workspace. To avoid the collision test with all the obstacles as the traditional method do, we add a pre-processing to localize the obstacles in the workspace by decomposing the workspace into rectangular cells then we give for each obstacle a rectangular region by taking the maximum and the minimum of vertices coordinates of the obstacle (Fig. 3.2). Then, we construct the matrix M representing the workspace. Each matrix element contains the number of obstacle regions overlapping the corresponding cell.

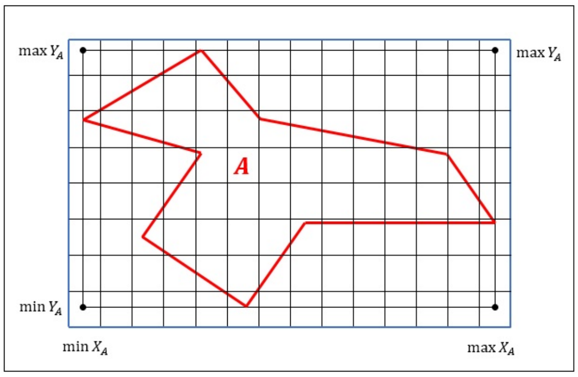


Figure 3.2: A non-convex obstacle A with his rectangular region found by taking the maximum and the minimum abscises according to X and Y axises. The blue rectangle is the obstacle region

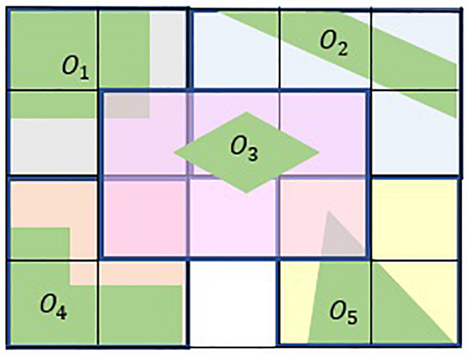
Once the construction of the matrix M is done, it can be used now in the collision checking procedure. After localizing the sampled configuration in the workspace, we do the collision checking with just the obstacles found in the cell of the matrix M that corresponding to the location of the sampled configuration.

3.4.2 Pre-processing (obstacles localization)

The idea of the pre-processing algorithm is to perform the most of collision checking tests in the beginning. In this paper, we consider a 2D environment, and the obstacles are non-convex polygons, so the matrix M that represent the environment is of dimension $(n \times m)$. First of all, the algorithm divides the environment into a grid as is shown in Fig. 3.3. The number of the grid cells along the X axis is n and the number of the grid cells along the Y axis is m. Then, the algorithm creates the $(n \times m)$ empty matrix M witch has the same dimension of the grid. The obstacles are numbered from 1 to k. The algorithm gives for each obstacle O_i a rectangular region (Fig. 3.3), where the vertices coordinates of this region in clockwise direction are $X_{O_i region} = [min(X_{O_i})\ min(X_{O_i})\ max(X_{O_i})\ max(X_{O_i})]$, $Y_{O_i region} = [min(Y_{O_i})\ max(Y_{O_i})\ max(Y_{O_i})\ min(Y_{O_i})]$. In the workspace, each obstacle region cover some cells of the grid, so for each obstacle region, the algorithm will write its number in all the elements of the matrix M that correspond to the cells of the grid covered by this obstacle region. Note that a cell of the grid can be covered by more than one obstacles region. An example of creating the matrix M for a 2 dimension environment is shown in Fig. 3.3 .

3.4.3 Collision checking procedure

We consider the robot is punctual, so in this procedure, when a configuration is sampled, the algorithm finds the cell of the grid that contains this configuration. Next, the collision checking will be performed with just the obstacles whose numbers are found in the element of the matrix M that correspond to this cell. By applying this strategy we avoid the collision checking with all obstacles in the workspace. This strategy can be used with polygonal robot, where we find all the cells that contain the robot then we perform the collision checking with the union of obstacles numbers found in the elements of the matrix M corresponding to those cells. This algorithm can be generalized to three dimensions environment.



$$M = \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 1 & 1,3 & 2,3 & 2,3 & 2 \\ 4 & 3,4 & 3 & 3,5 & 5 \\ 4 & 4 & NIL & 5 & 5 \end{bmatrix}$$

Figure 3.3: 2D environment divided into grid with its corresponding matrix M. The bold rectangles are the obstacles regions. One cell can be covered by more than one obstacle region. NIL means the cell is not cover by any obstacles region

Algorithm 4 The pre-processing procedure (obstacles localization)

```
1: **Input:**
2: $Obs$ : the vector of obstacles
3: **Output:**
4: $M$ : the matrix representing the environment, contains the information about the obstacles
5: for $A$ in $Obs$ do
6:   $I_{min}, I_{max} \leftarrow$ indices of cells that contain $min(X_A)$, $max(X_A)$ respectively according to X axis
7:   $J_{min}, J_{max} \leftarrow$ indices of cells that contain $min(Y_A)$, $max(Y_A)$ respectively according to $Y$ axis
8:   for $i \leftarrow I_{min}$ to $I_{max}$ do
9:     for $j \leftarrow J_{min}$ to $J_{max}$ do
10:      if $N(A)$ in $M(i,j)$ then
11:        Continue
12:      else
13:        $M(i,j) \leftarrow M(i,j) \cup N(A)$
14:      end if
15:    end for
16:  end for
17: end for
```

Algorithm 5 Collision checking procedure

```
1: Input:
2: $Obs$ : obstacles vertices coordinate
3: $conf$ : the configuration to test for collision
4: $M$ : the produced matrix by algorithm 4.
5: Output:
6: true: if the configuration is in collision ; false: otherwise
7: $i, j \leftarrow$ indices of cells that contain $conf$ according to $X$ axis and $Y$ axis
8: $Obs_t \leftarrow M(i, j)$
9: for $A$ in $Obs_t$ do
10:    if $conf$ in collision with $A$ then
11:        return true
12:    end if
13: end for
14: return false
```

3.4.4 Simulation results

To state the performance of our algorithm, we have integrated it with the probabilistic roadmap planner (PRM). Then, we compared its performance with a PRM that uses a traditional collision checking. A 2D environment and a punctual robot model are considered. The comparison in the simulation has been done only for the learning phase of the PRM. We have used two sampling strategies: the Gaussian and the uniform. The number of configurations to generate in each simulation is set to $n = 500$.

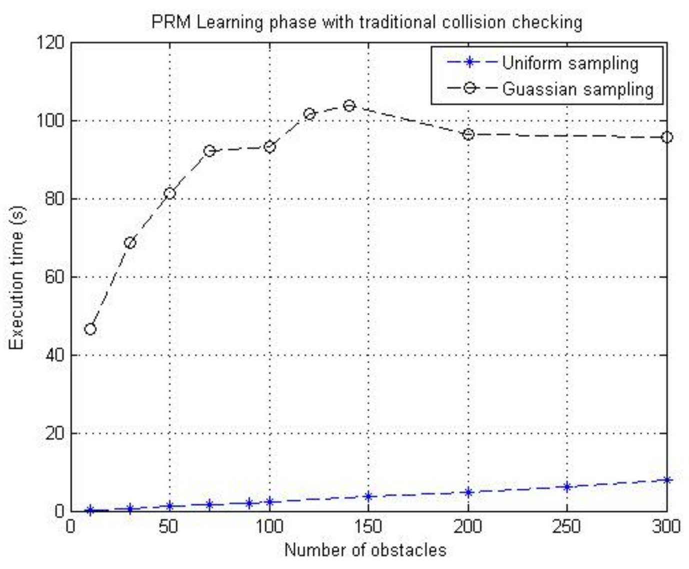


Figure 3.4: Execution time in function of number of obstacles for learning phase of PRM path planning with traditional collision checking, using the uniform sampling and the Gaussian sampling strategies.

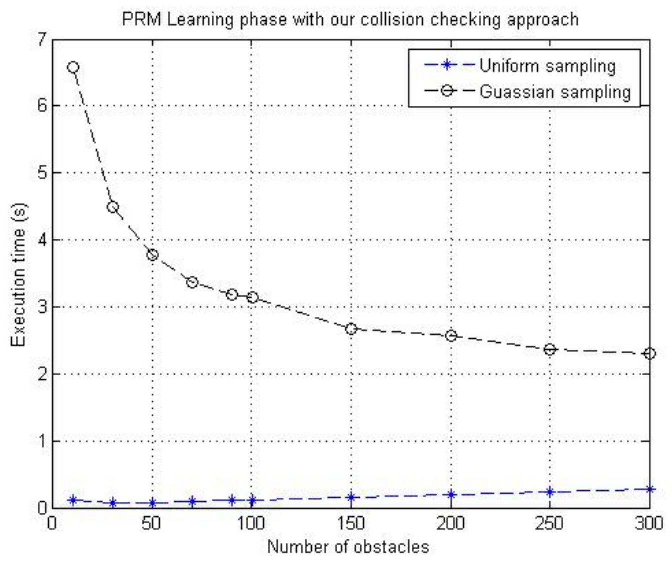


Figure 3.5: Execution time in function of number of obstacles for learning phase of PRM path planning with our collision checking approach, using the uniform sampling and the Gaussian sampling strategies.

Fig. 3.4 shows the variation of execution time of learning phase in function of number of obstacles for PRM with a traditional collision checking procedure. It is clear for traditional collision checking that the execution time increases linearly in function of number of obstacles, t = 7.96s with 300 obstacles for uniform strategy, and t = 97.3s with 300 obstacles for Gaussian strategy. While, with our algorithm (Fig. 3.5), the execution time stay roughly constant, around t = 0.28s for uniform strategy, and it decreases until t= 2.3s for the Gaussian strategy. In Fig. 3.4, the execution time for Gaussian strategy does not increases linearly because the saturation of the scene of obstacles, this gives high chances to find the second configuration. Otherwise it should grow linearly.

From this results we can conclude that our approach in collision checking is very efficient with high number of obstacles.

3.5 Conclusion

We have presented in this chapter the basic concepts and preliminaries of motion planning. Then, we have focused on sampling-based motion planning algorithms which are known by their efficiency in solving high degree of freedom problems. After that, we presented the PRM and RRT* algorithms witch are used in this thesis. Then, we have presented our proposed approach for reducing the collision checking time in cluttered environment. The next chapter will focus on the proposed RRT* variant called NP-RRT*.

Chapter 4

Narrow Passage RRT*

4.1 Introduction

Motion planning algorithms aims to find feasible paths from starting point to goal point, avoiding all the obstacles. In the context of optimality, the planned path should be the optimal path in the workspace. Motion planning has been widely used in robotics, video games, digital character animation, and in many other related domains [21]. Many path-planning algorithms have been presented over the last decades, including Potential Fields [22], Cell Decomposition [23], Graph-Based [24] [25], Roadmap, and sampling-based algorithms [14]. Each of them has advantages and disadvantages, in term of complexity and convergence to optimal path. Sampling-based planning algorithms are the most popular compared with the other approaches, because of their less complexity and their efficiency in solving high dimensional configuration space problems [26].

(RRT*) Rapidly Exploring Random Trees Star [9] introduced in 2011, is a sampling-based algorithm. It was a major breakthrough development in the optimal path planning. RRT* is probabilistic complete and asymptotically optimal. This means it can answer a query if adequate number of samples are provided, and it converges to an optimal solution when the iterations number goes to infinity [8]. However, this method has many limitations like the slow convergence rate, large memory requirements, dealing with narrow passages and the complexity of collision checking module.

In this chapter we will present our contribution in path planning, which is a modified RRT* called Narrow passage RRT* (NP-RRT*) [27]. It addresses mainly the problem of narrow passages, also, the convergence rate and the memory consumption issues.

4.2 Literature review

Many researches have been done to overcome the limitations of RRT*, modifying the sample biasing, using heuristic-based sample rejection, graph pruning, and proposing other strategies [28]. The Anytime RRT* [29] was proposed to overcome the large computational time, by executing the planner for a specified amount of time. Once an initial path is found, the rest of the time is allocated to improve the initial solution. The authors in [30] introduced a new version of RRT* called RRT* Fixed Nodes (RRT*FN), to deal with the large memory limitation. It allows limited number of nodes in tree. When this pre-defined number is reached, then a new node can only be inserted by deleting an old one. This variant has been improved in [31] [32]. Another variant of RRT*, called RRT*-Smart proposed by [33], addresses the issue of slow convergence. The first path is searched using the basic RRT* algorithm. Then, it is optimized using an intelligent sampling strategy with triangular inequality principle. In other variant of RRT* named Informed-RRT* [34], a direct sampling method introduced by defining the accepted region as a hyper-ellipsoid, in order to reduce the execution time and convergence rate. This variant has been improved in [35] [36] [37]. However, these methods relies upon the first solution found by RRT*. RRT*N [38] introduced a technique to reduce the processing time, where they concentrate the generated random configurations near a geometric vector between the start point and the goal point, according to uniform distribution. In 2019, a modified RRT* named Quick-RRT* [39] proposed a new strategy to generate better initial solution and converges to the optimal faster than RRT*. This method enlarge the set of possible parents in the Choose Parent and Rewiring procedures.

A combination of RRT*-connect and Informed-RRT* was proposed in [40] named Informed-RRT*-Connect. The latter takes the advantages of both methods to reduce the number of iterations and to generate low cost solution. Another combination named PQ-RRT* was introduced in [41], where they combined between Potential-RRT* [42] and Quick-RRT*, resulting to a method that finds the first solution very fast and converges to optimal solution quickly. Similar work in [43] propose a potentially guided bi-directional RRT* to speed up the convergence rate and to optimize the memory. The method builds incrementally two trees, the first rooted to start point and the second to the goal point.

4.3 Collision checking in RRT*

The complexity of collision checking module in RRT* as shown in [9], depends on number of obstacles. As a result, solving problems with high number of obstacles needs infinite execution time. Several studies have been addressed this problem by proposing strategies to reduce the number of calls to collision checking module [16] [18] [44]. Our approach presented in [20], shows that the complexity of collision checking do not depend on the number of obstacles. The idea is to perform a pre-procedure that saves information about the obstacles position in a matrix at the beginning of planning process. Then, this matrix is used to answer collision checking queries. The advantage of this approach is to perform the collision checking with just nearby obstacles to the configuration, instead of testing the collision with all the obstacles.

4.4 NP-RRT* Algorithm

In this section, the NP-RRT* algorithm alongside the modified steer function and the path optimization technique are described. The algorithm searches for initial path performing the same as RRT* except that NP-RRT* uses a modified steer function named `SteerNP` to generate samples near the obstacles as will be discussed in the next subsection. Once a first path is found, the path optimization technique starts executing at regular intervals of iterations by generating a random configuration near the path. This interval depends on the environment. Whenever a new low-cost path is found, the path optimization technique will generate a configuration near that path, performing a tradeoff between exploring the configuration space and path optimization. This process is outlined in Algorithm 6.

The steps 1,3 and from 14 to 18 execute the same as RRT*, the difference here between NP-RRT* and RRT* is there is an if condition in step 4 giving two ways to generate a sample that expand the tree, either optimizing the path (step 5 to 8) or exploring the configuration space (step 10 to 12). The optimization path branch uses `RandConfNearPath` function to generate randomly a sample q_{rand} near the new found path returned by `FindPath` function (step 22), then q_{new} is generated using the same steer function of RRT*. This last branch is executed only if the iteration is equal to the optimization iteration n_{op}. The variable n_{op} is returned once a first path is found (steps 19 to 21). Then, n_{op} is incremented by the pre-selected interval I in step 5. In the exploring branch, q_{rand} is generated in the same way as in RRT*, then q_{new} is returned from `SteerNP` function. After the generation of q_{new}, the same process of RRT* is executed (step 14 to 18) adding q_{new} to the tree and rewiring procedure.

Algorithm 6 NP-RRT* Algorithm

```
1: $V \leftarrow \{q_{init}\}$ ; $E \leftarrow \emptyset$;
2: $n_{op} \leftarrow 0$;
3: for $i = 1$ to $N$ do
4:     if $i = n_{op}$ then
5:         $n_{op} \leftarrow n_{op} + I$;
6:         $q_{rand} \leftarrow$ RandConfNearPath($Q$, $path$);
7:         $q_{nrst} \leftarrow$ Nearest($G = (V, E)$, $q_{rand}$);
8:         $q_{new} \leftarrow$ Steer($q_{nrst}$, $q_{rand}$);
9:     else
10:        $q_{rand} \leftarrow$ Sample($Q$);
11:        $q_{nrst} \leftarrow$ Nearest($G = (V, E)$, $q_{rand}$);
12:        $q_{new} \leftarrow$ SteerNP($q_{nrst}$, $q_{rand}$);
13:    end if
14:    if CollisionFree ($q_{nrst}$, $q_{new}$) then
15:        $Q_{near} \leftarrow$ Near($q_{new}$, $G = (V, E)$, $r_{RRT^*}$);
16:        $q_{min} \leftarrow$ BestParent($q_{new}$, $Q_{near}$);
17:        $V \leftarrow V \cup \{q_{new}\}$; $E \leftarrow E \cup \{(q_{new}, q_{min})\}$;
18:        $G \leftarrow$ RewireTree($q_{new}$, $Q_{near}$, $G = (V, E)$);
19:        if Initial path found then
20:            $n_{op} \leftarrow i + 1$;
21:        end if
22:        $path \leftarrow$ FindPath($G = (V, E)$, $q_{init}$, $q_{goal}$);
23:    end if
24: end for
25: return $G = (V, E)$;
```

4.4.1 Steer function

The steer function of RRT* as described in [9]: given two configuration q_{rand} (generated randomly in the configuration space) and q_{nrst} (the configuration from which the algorithm tries to expand the tree), the function `Steer` (q_{nrst}, q_{rand}) returns a configuration q_{new}

between q_{nrst} and q_{rand} such that q_{new} is closer to q_{rand} then q_{nrst}. This strategy of sampling has bad performance in the presence of narrow passages. Because it samples the space uniformly, there is low chance to sample narrow passages. For that, the `SteerNP` function samples the near obstacles region to cover these narrow passages. The idea of this modified steer function was inspired from the Gaussian sampling strategy in probabilistic roadmap method [45], where the configurations are generated near the obstacles according to Gaussian distribution.

Let Q_{nor} be the near obstacles region and Q_{np} denotes the narrow passages sub-space. Let w_o be the width of obstacle region around that obstacle. It is obvious that $Q_{nor} \cap Q_{np} \subset Q_{nor}$. Also, when w_o is big enough $Q_{np} \subset Q_{nor}$. From this, we can say that sampling Q_{nor} sub space means sampling Q_{np}.

The `SteerNP` function takes as inputs two configurations q_{nrst} and q_{rand}. Then, it moves incrementally an intermediate configuration named q_{int} from q_{nrst} towards q_{rand} along a straight line by a step Stp, until q_{int} is in collision or the distance between q_{nrst} and q_{int} is greater than a pre-defined distance d. After that, q_{int} moved back by a step Stp, and it returned as q_{new}. This process is outlined in Algorithm 7 and illustrated in Fig. 4.1. The main two features of `SteerNP` function is it returns a collision free edge (q_{nrst}, q_{new}) directly without calling the collision checking procedure, such as the step 6 in RRT* algorithm. The second feature, is that it can explore open spaces outside of obstacles regions, by setting the maximum distance d between q_{nrst} and q_{new}.

Algorithm 7 `SteerNP` function

```
1:  $q_{int} \leftarrow q_{nrst}$; $q_{new} \leftarrow q_{nrst}$;
2:  while CollisionFree($q_{nrst}$, $q_{int}$) do
3:      if dist($q_{nrst}$, $q_{int}$) $> d$ then
4:          $q_{new} \leftarrow q_{int}$;
5:          return $q_{new}$;
6:      end if
7:      $q_{int} \leftarrow q_{int} + Stp$;
8:  end while
9:  $q_{new} \leftarrow q_{int} - Stp$;
10: return $q_{new}$;
```

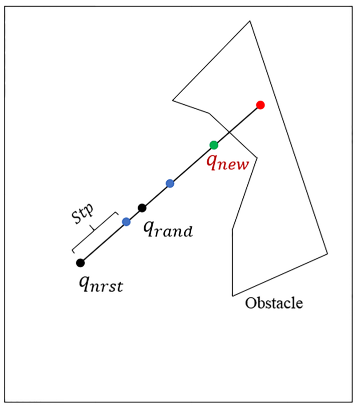


Figure 4.1: The blue points denote q_{int} which moves incrementally from q_{nrst} towards q_{rand} along straight line, until it reachs an obstacle. The red point is q_{int} which is in collision. The green point denotes the returned q_{new}.

4.4.2 Path optimization technique

During the RRT* planning process, the tree is expanding towards configurations generated uniformly. Even an initial path was found, the algorithm pursues the same behavior. As a result, the convergence rate of RRT* is slow. In NP-RRT* algorithm, the found path is exploited as information to generate useful nodes. Once an initial path is found, the path optimization technique branch (step 5 to 8 in algorithm 6) will be executing at regular intervals of iterations. The `randNodeNearPath` function takes the found path as parameter, then it selects randomly an edge from this path, and generates a configuration near that edge using Gaussian distribution. After that, q_{new} is generated simply using Steer function of RRT*.

4.5 Complexity analysis

4.5.1 Space complexity

The space complexity of an algorithm is the amount of memory space required to solve an instance of the computational problem as a function of the size of the input [46]. Since NP-RRT* requires n number of memory configurations to execute n number of iterations, the space complexity of NP-RRT* is the same as RRT* which is $O(n)$.

4.5.2 Time complexity

The time complexity is the computational complexity that describes the required amount of time taked to run an algorithm [46]. In the case of NP-RRT*, the additional `SteerNP` function and path optimization steps have neglected effect on the time complexity. But the used collision checking procedure, significantly reduced the execution time. The T-Notation analysis for both algorithms, comparing the execution time for the same number of iterations, gives a significantly less time for NP-RRT* compared to RRT*. Therefore, the time complexity of NP-RRT* is the same as RRT* which is $O(nlogn)$ with much better performance.

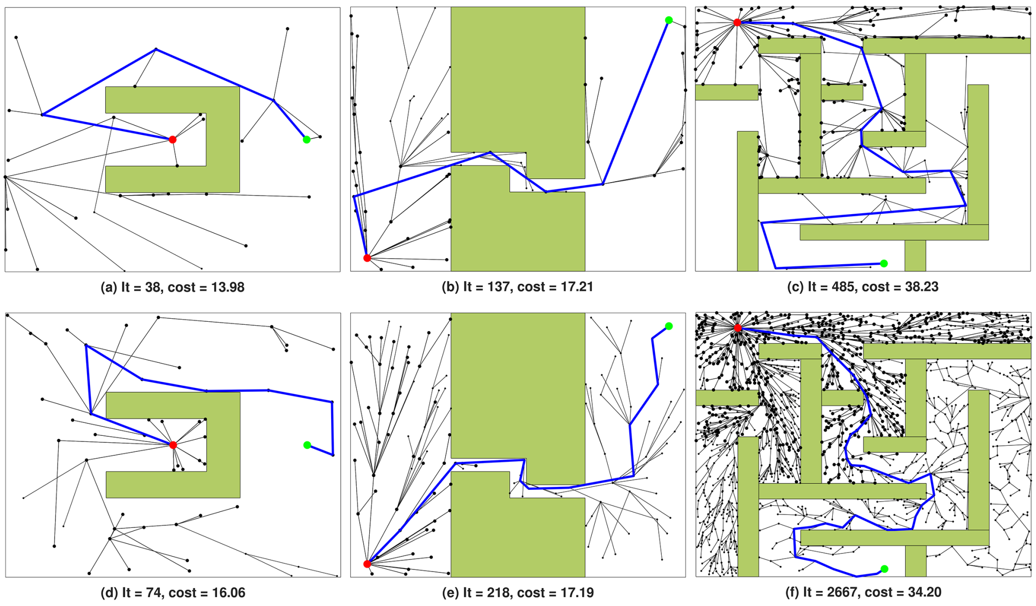
4.6 Simulations and results

In this section, statistical comparison results between NP-RRT* and RRT* are presented. This comparison goes through three stages. The first one shows the performance of both methods in finding the first path in three different environments.Then, the second stage presents the convergence rate of the two methods in simple and narrow passage environments. The last stage shows the effect of collision checking procedure for both algorithms in high number of obstacles environments.

For all the simulations, the cost of paths is chosen to be the Euclidean distance as metric in the configuration space. We set the gamma constant for both methods to $\gamma = 8$, and we fixed the interval of iteration to execute the path optimization technique for NP-RRT* to $I = 5$. In the figures, the red point denotes the starting node and the green point denotes the destination. The blue line is the path and the green solid boxes are the obstacles. All the simulations are performed on a computer with Intel Core i3 2.30GHz, and 4GB of RAM, using MATLAB version (2014a).

4.6.1 First path

In order to test the performance of our method in finding the first path, we have chosen three different environments: simple, narrow passage and maze as shown in Fig. 4.2. The results of 50 runs are presented in Table 4.1. In this stage, we are not using the presented collision checking approach, in order to show the effect of `SteerNP` function in finding the first path.


(a) It = 38, cost = 13.98 (b) It = 137, cost = 17.21 (c) It = 485, cost = 38.23

(d) It = 74, cost = 16.06 (e) It = 218, cost = 17.19 (f) It = 2667, cost = 34.20

Figure 4.2: The first path found by NP-RRT* and RRT* in three different environments: simple (on the left), narrow passage (in the middle) and maze (on the right). (a), (b), (c) for NP-RRT*, (d), (e), (f) for RRT*.

Table 4.1: Statistical comparison between NP-RRT* and RRT* in finding the first path in three different environments.

Env.	Simple				Narrow Passage				Maze			
Alg.	RRT*		NP-RRT*		RRT*		NP-RRT*		RRT*		NP-RRT*	
	Avrg.	Std. d	Avrg.	Std. d	Avrg.	Std. d	Avrg.	Std. d	Avrg.	Std. d	Avrg.	Std. d
Ex. t (s)	1.55	0.52	1.56	0.67	2.26	0.68	1.94	0.44	6.98	1.80	6.10	1.62
Nodes	66.80	33.82	50.02	26.27	123.05	55.85	75.50	21.18	707.05	260.07	356.57	111.35
Cost	15.73	1.89	15.98	1.62	18.39	1.50	18.50	1.25	39.61	3.14	40.77	2.79
It	99.62	41.89	73.60	30.15	303.25	167.98	168.30	59.88	1463.37	468.80	808.50	255.01
Env.: Environments; **Avrg.:** Average; **Std. d:** Standard deviation; **Ex. t:** Execution time; **It:** Iterations.												

From Table 4.1, it is clear that NP-RRT* has better performance in term of iteration and node number for the three environments. But in term of cost, the first path found by RRT* has less cost comparing to NP-RRT* for all the environments. In term of execution time, NP-RRT* takes less time in narrow passage and maze environment, whereas they take roughly the same time in the simple environment.

From this comparison, NP-RRT* shows good performance in finding the first path faster than RRT* in difficult and simple environments. Beside that, NP-RRT* consumes less memory in all the simulations.

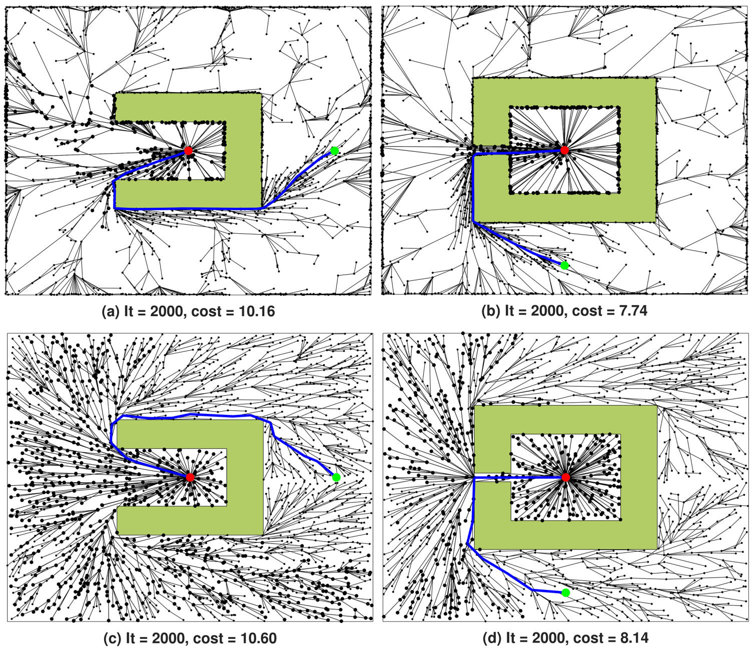
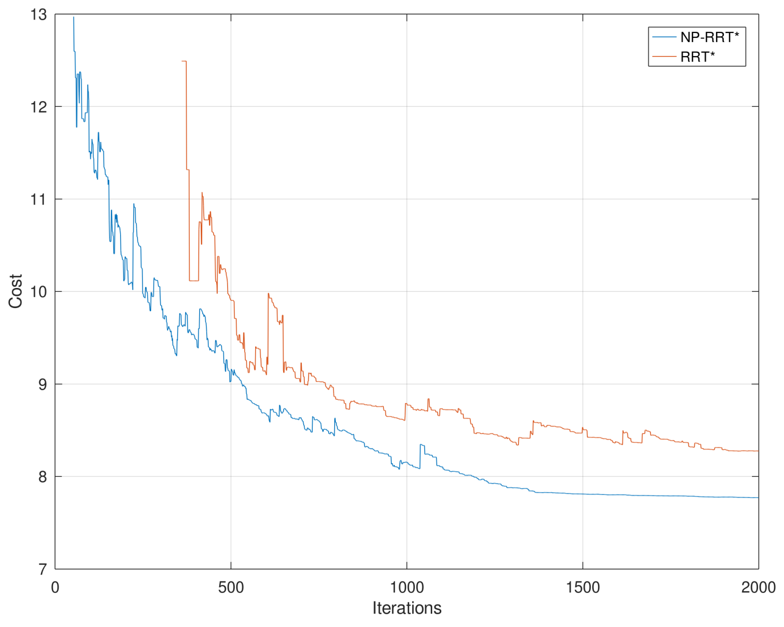

(a) It = 2000, cost = 10.16
(b) It = 2000, cost = 7.74
(c) It = 2000, cost = 10.60
(d) It = 2000, cost = 8.14

Figure 4.3: The path at 2000 iterations in simple environment on left side, and in narrow passage environment on the right side. (a), (b) for NP-RRT* and (c), (d) for RRT*.

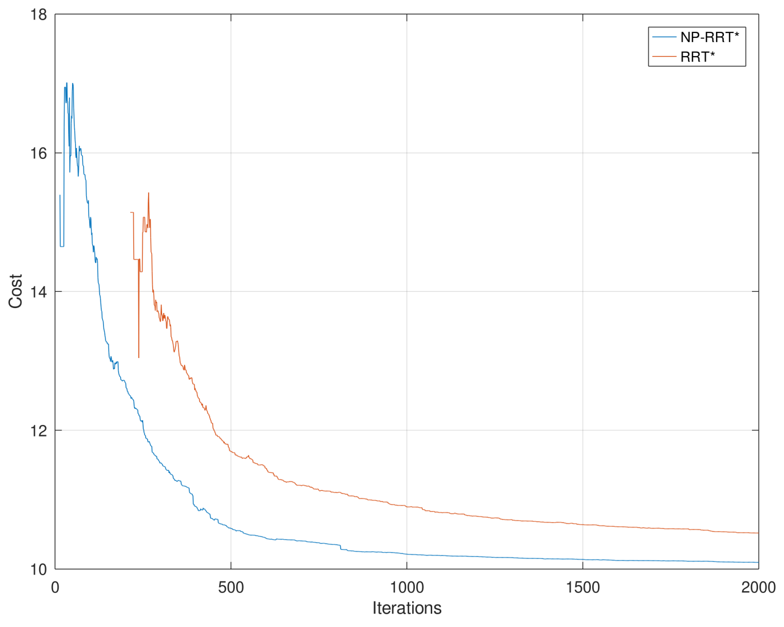
4.6.2 Convergence test

In this stage, we have selected two environments to compare the convergence rate of our method with RRT*. These two environments are shown in Fig. 4.3, where the first one is designed to be a simple environment, and the second one is designed to contain a narrow passage. The statistical results of 50 runs are presented in Fig. 4.4, where the costs are plotted versus number of iterations.

From Fig. 4.4, it can be seen that for the two environments, NP-RRT* starts from a big initial cost comparing to RRT*, then it converges very fast to an optimal or sub-optimal path in finite number of iterations, whereas RRT* still in process of reaching an optimal solution with slow rate of convergence. It is clear that the graph of NP-RRT* start before the graph of RRT* taking a smaller number of iterations for the two environments. This ensures the efficiency of our algorithm in finding quickly the first solution.



(a)



(b)

Figure 4.4: Convergence rate for NP-RRT* and RRT*. (a) in simple environment, (b) narrow passage environment.

Table 4.2: Comparison of execution time between NP-RRT* and RRT* in different cluttered environments.

Number of obstacles	Ex. t (s) for RRT*	Ex. t (s) for NP-RRT*
10	60.54	7.06
50	293.05	7.29
100	889.61	9.56
150	1001.20	8.53
300	1554.32	9.96
500	2012.54	9.98

4.6.3 Cluttered environments

In order to show the effect of the presented collision checking, we have generated randomly three cluttered environments with different numbers of obstacles. Fig. 4.5 shows these environments, where the first has 10 obstacles, the second has 50 obstacles and 300 for the third. The average results of both methods are presented in the Table 4.2, where the execution time are captured for the same number of iterations it=2000.

Table 4.2 demonstrates clearly that NP-RRT* has significantly less time comparing to RRT* for all the environments. Also, the pattern of execution time of RRT*, is growing as the number of obstacles grows, where it starts from 60.54s for 10 obstacles, to reach 2012.54s for 500 obstacles. Whereas, NP-RRT* stays roughly constant,where the execution time for 10 obstacles is 7.06s, and 9.98s for 500 obstacles.

From these results, one can observe the huge reduction in time complexity due to the presented collision checking for NP-RRT*. This shows the efficiency of our method in solving problems with high number of obstacles.

4.7 Conclusion

We have presented in this chapter our a new variant of RRT*, named NP-RRT*, to overcome the limitations of RRT* in solving problems with narrow passages and high number of obstacles, also to address the issue of convergence rate. A new steer function that generates configurations near the obstacles has been proposed, in order to sample difficult spaces. Path optimization technique was presented to speed up the convergence rate by generating nodes near the found path. Simulation results demonstrate the efficiency of our method in finding the first solution in different environments, comparing to basic RRT*, in respect to iteration number, execution time and node number. The NP-RRT* converges very fast to optimal or sub-optimal path, comparing to RRT*. Furthermore, the execution time of RRT* in cluttered environments has grown as the number of obstacles grow, whereas, the number of obstacles has no effect on our method.

In the next chapter we will present the trajectory generation and optimization for Quadrotors in 3D environment.

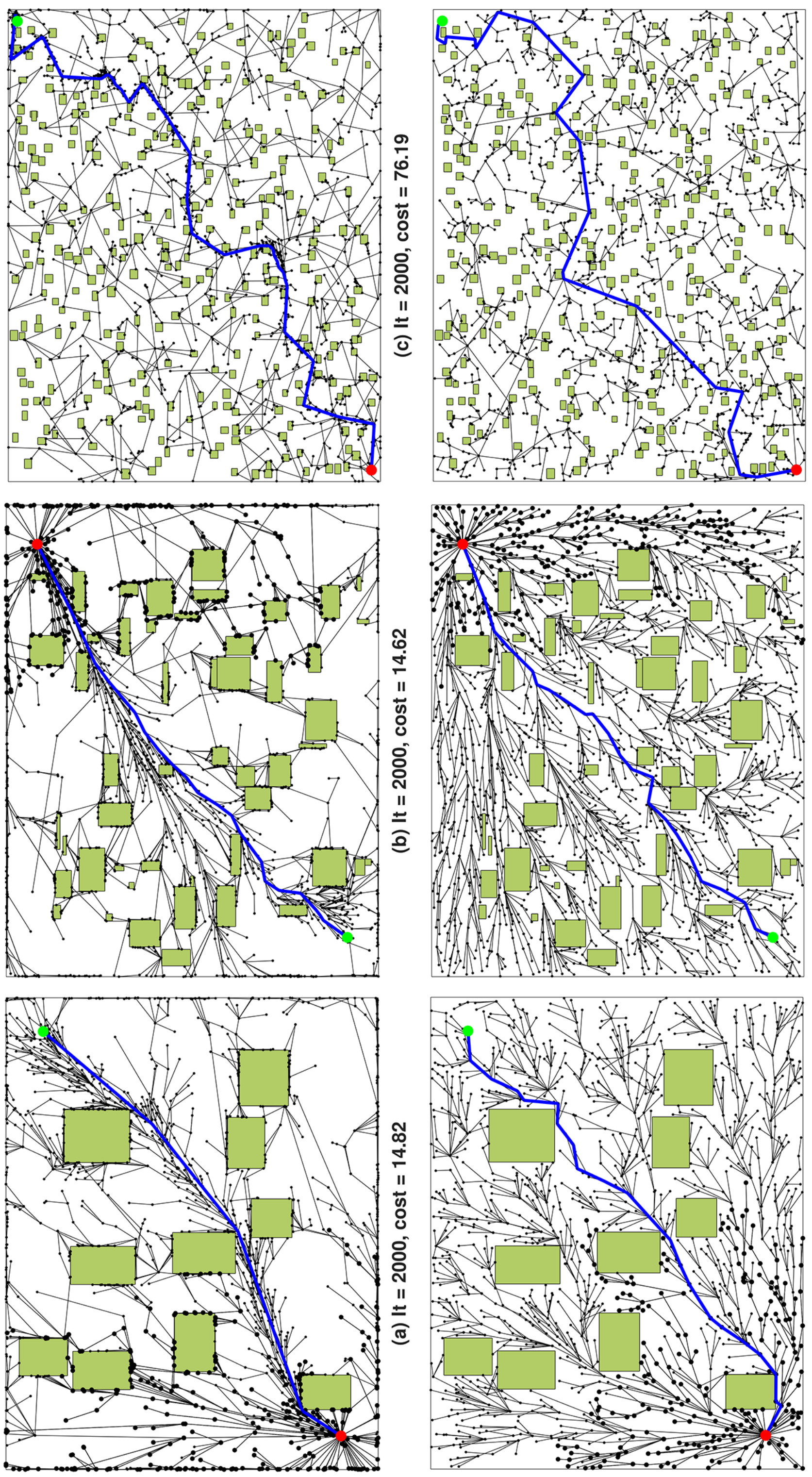

(a) It = 2000, cost = 14.82
(b) It = 2000, cost = 14.62
(c) It = 2000, cost = 76.19
(d) It = 2000, cost = 16.10
(e) It = 2000, cost = 15.18
(f) It = 2000, cost = 79.99

Figure 4.5: The path at 2000 iterations in three cluttered environments generated randomly with 10, 50, 300 obstacles. (a), (b), (c) for NP-RRT* and (d), (e), (f) for RRT*.

Chapter 5

Minimum snap trajectory for Quadrotors

5.1 Introduction

Accomplishing tasks by a Quadrotor needs motion planning, to find safe, feasible, and collision-free trajectories, possibly in the presence of some constraints. Two planning approaches are used. The decupled approach, first, looks for a collision-free path in the configuration space and, then, finds the trajectory in the state space. While the direct approach finds, directly, the trajectory in the state space [8].

The Quadrotors are non-linear systems with six degrees of freedom, controlled by four input signals. Optimizing trajectories for such systems has been the subject of many research works. Cowling et al. [48] and Bouktir et al. [47] presented decupled optimal trajectory planners based on polynomials and B-Splins, respectively. The path is planned in the configuration space, then the trajectory is parametrized in time considering dynamical and feasibility constrains. Lai et al. [49] handled the problem of time optimal control using nonlinear programming method coupled with Genetic Algorithm (GA). Mixed Integer Linear Programming (MILP) was implemented in [50] to minimize trajectories for UAVs. But it increases the number of variables, resulting on increasing number of evaluations of the recursive functions. This causes a heavy computational burden. Differential flatness representation for Quadrotor guarantees sufficiently continuous trajectories, in flat output space. Then, these trajectories can be translated into dynamically feasible trajectories, in the state space [51]. Based on this concept, Mellinger and Kumar [52] have presented an algorithm to minimize the snap, using quadratic programming. Richter et al. [53] extended this work to find the optimal waypoints in cluttered environment, using RRT*. Then the whole polynomial trajectory is optimized, based on unconstrained quadratic programming.

Since trajectory optimization for Quadrotors can be a multi-objective problem, the numerical solutions stated before became computationally impractical when trying to optimize trajectory with high number of waypoints, in addition to other constraints, like energy and distance optimization. In such complex situations, Evolutionary Algorithms may be useful, because they can find good compromise, by approximating the Pareto optimal set in a single run [54].

Many studies have used these algorithms, but most of them tried to solve path planning in the configuration space. In [55] Bezier curves are used with GA to produce trajectories that respect a minimum curvature for UAVs. Bao et al. [56] presented a Particle Swarm Optimization (PSO) based method to compute the shortest path between two points. Nikolos et al. [57] presented an offline/online path planner based on GA and B-Spline curves.

In this chapter will present a PSO algorithm to minimize the snap of polynomial trajectories for Quadrotors, using PD control. The main contribution is we add the minimum distance criterion beside the minimum snap and time to the cost function. This improves the quality of the trajectory.

5.2 Minimum snap and distance trajectory

The snap is the forth derivative of the path. Minimum snap trajectory for Quadrotors was first introduced by Mellinger and Kumar [52]. They showed that the Quadrotor dynamics can be written as differential flatness representation. Then, they showed that minimum snap polynomial trajectories are the natural choice for Quadrotors. Because the motor commands and attitude accelerations of the vehicle are proportional to the snap of the path. This work has been extended by Richter et al. [53]. They used the quadratic program to find the minimum snap polynomial trajectory composed of many segments in indoor cluttered environments. The waypoints are generated by RRT* algorithm.

In this work, the same problem formulation described in [53] is used, but we added the minimum distance term to the cost function and, instead of using the quadratic program, we used the PSO algorithm for the optimization task. It is, also, assumed that the waypoints are given in 3D environment without obstacles.

For one segment trajectory between two configurations, the Quadrotor has four independent polynomials x, y, z, ψ. The cost function for one polynomial $P(t)$, that minimize the snap and the distance, can be written as

$$J(T) = \int_0^T (P'(t)^2 + P^{(4)}(t)^2)dt \tag{5.1}$$

Where T is the duration of the trajectory.

From the calculus of variations, the polynomial $P(t)$ is seventh order, the example for the polynomial of x is:

$$x(t) = c_7t^7 + c_6t^6 + c_5t^5 + c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 \tag{5.2}$$

The solution of this integral can be written in matrix form as follows:

$$J(T) = \mathbf{p_i^T} Q_{s_i}(T)\mathbf{p_i} + \mathbf{p_i^T} Q_{d_i}(T)\mathbf{p_i} \tag{5.3}$$

where $\mathbf{p_i} = [c_7, c_6, \dots, c_0]^T$ is the vector of one polynomial coefficients and (Q_{s_i}, Q_{d_i}) are the Hessian matrices for the snap and distance.

For n polynomial segments, the cost function will be

$$J_t = \mathbf{p^T} Q_s(T)\mathbf{p} + \mathbf{p^T} Q_d(T)\mathbf{p} \tag{5.4}$$

Where $\mathbf{p} = [\mathbf{p_1}, \dots, \mathbf{p_n}]^T$ is the total vector of n polynomial coefficients, and (Q_s, Q_d) are the total Hessian matrices written as block diagonal fashion.

To ensure the continuity and the differentiability, each end segment i conditions must be the same as starting conditions of the next segment $i + 1$. Since the polynomials are seventh order, the coefficients of one polynomial are calculated by solving a linear system of eight order

$$A_i\mathbf{p_i} = \mathbf{d_i} \tag{5.5}$$

where $\mathbf{d_i}$ is the vector that contains the initial and the end conditions for the i^{th} polynomial, and A_i is a 8×8 matrix written in function of T. Therefore, for n polynomial segments, (eq. 5.5) becomes

$$A\begin{bmatrix}\mathbf{p_1}\\ \vdots \\ \mathbf{p_n}\end{bmatrix} = \begin{bmatrix}\mathbf{d_1}\\ \vdots \\ \mathbf{d_n}\end{bmatrix}$$

$$A\mathbf{p} = \mathbf{d} \tag{5.6}$$

By replacing the vector of coefficients $\mathbf{p}$ in the cost function by the vector of conditions $\mathbf{d}$, the decision variables will be the waypoint's conditions. The new cost function will be

$$J_t = \mathbf{d^T}A^{-1}Q_sA^{-1}\mathbf{d} + \mathbf{d^T}A^{-T}Q_dA^{-1}\mathbf{d} \tag{5.7}$$

In the previous cost function, the duration of each segment is supposed to be priori fixed. This assumption is used in [52], where they fixed the total time heuristically, then, they minimized the snap using the remaining degrees of freedom. However, the authors in [53] showed that, from the planning context point of view, the time cannot be priori fixed, instead, one can let it free to vary with the optimization process. This will improve the quality of the solution. Hence, in this work, the sum of segment times is added to the previous cost function, then we penalize each term by a factor, as it will be discussed in the next section. The final cost function has the following form:

$$\begin{aligned} J_t &= k_s\mathbf{d^T}A^{-1}Q_sA^{-1}\mathbf{d} \\ &+ k_d\mathbf{d^T}A^{-T}Q_dA^{-1}\mathbf{d} + k_t\sum\nolimits_{i=1}^{n} T_i \end{aligned} \tag{5.8}$$

5.3 PSO implementation

Particle swarm optimization is a population-based optimization technique proposed in 1995 by Eberhart and Kennedy [58]. The idea is inspired from the swarm behavior of birds. A basic variant of the PSO algorithm initializes a swarm's particles randomly in a prespecified range. In each iteration, these particles update their positions and velocities using the following equations:

$$\begin{aligned} v_{t+1} &= wv + c_1 r_1 (x_t^{Lbest} - x_t) + c_2 r_2 (x_t^{Gbest} - x_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned} \tag{5.9}$$

Where x is a particle, v is the velocity of a particle, w is the inertia coefficient, c_1, c_2 are the acceleration coefficients, x^{Lbest} is the local best position particle, x^{Gbest} is the global best position particle.

The movements of the particles, in the search-space, are guided by their local best position and the global best position. When improved position is discovered, the latter will guide the search of the swarm. This process is repeated until a maximum number of iterations is reached or a satisfactory solution is discovered.

5.3.1 Multi-objective PSO

Multi-objective PSO Many works have been proposed to extend the basic PSO to handle multi-objective problems [59]. The reason that we have chosen PSO among the other evolutionary algorithms is that it is a population based approach and it is simple and power full, it has been used widely for trajectory optimization problems. This work uses the version proposed in [60]. The latter looks first for the front of pareto-optimal solutions set, then it uses a gradient technique to find an optimal solution. The idea of this approach is to divide the swarm equally to m subswarms. Then, for each subswarm, a set of weights, penalizing the objective function, are used to compute the cost of particles. Each subswarm evolves into the direction of its own leader. The total swarm is guided by the particle yielding the best cost value. Fig. 5.1 summarizes the method.

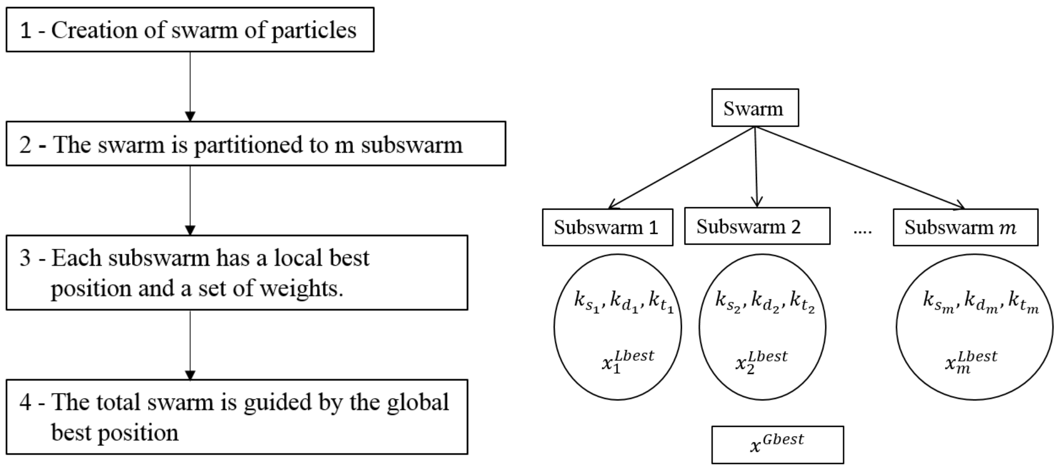

1 - Creation of swarm of particles
2 - The swarm is partitioned to m subswarm
3 - Each subswarm has a local best position and a set of weights.
4 - The total swarm is guided by the global best position
Swarm
Subswarm 1
Subswarm 2
....
Subswarm m
$k_{s_1}, k_{d_1}, k_{t_1}$
x_1^{Lbest}
$k_{s_2}, k_{d_2}, k_{t_2}$
x_2^{Lbest}
$k_{s_m}, k_{d_m}, k_{t_m}$
x_m^{Lbest}
x^{Gbest}

Figure 5.1: The multi-objective PSO proposed in [60]

The proposed objective function (eq. 5.8) has three terms: distance, snap, and time. These terms are weighted by k_d, k_s, k_t respectively, where the sum of this coefficients is one. These factors can be fixed heuristically, if the number of subswarms is small, or randomly, if it is high. In our implementation, we have fixed the number of subswarms to $m = 10$. Therefore, we have set the weight factors randomly for each subswarm. The other parameters of PSO are fixed as follows

- The inertia coefficient $w = 0.8$
- The acceleration coefficients $c_1 = 1$, $c_2 = 1$
- r_1, r_2 are random values between 0 and 1

5.3.2 Decision variables vector components

For one polynomial, the decision variable vector D has the form

$$D = [\mathbf{d}, T] \tag{5.10}$$

The vector $\mathbf{d}$ is composed of initial and final conditions for each waypoint, where the conditions are: position, velocity, acceleration, and jerk. For n waypoints, $\mathbf{d}$ has the dimension $8(n-1)$. Since the position of each waypoint is known priori and the conditions of the staring and the end points are also known, the dimension of $\mathbf{d}$ will be $3(n-2)$. The time of segments T has the dimension $(n-1)$, hence, the dimension of the total decision variable vector is $(4n-7)$. Since we are using a linear controller, the yaw must be close to the initial yaw. In our implementation, we let it fix. Therefore, the trajectory will have just three polynomials. The total decision vector will have the dimension $10n-19$, and its form will be

$$D = [\mathbf{d_x}, \mathbf{d_y}, \mathbf{d_z}, T] \tag{5.11}$$

5.4 Time and motors constraints

When generating a trajectory for a Quadrotor, we have to ensure that the control input signals require admissible motor thrusts. The formal solution to this problem, when using a differential model, is to map the motor constraints into the flat output space, before starting the optimization. An alternative technique is to compute the motors thrusts algebraically during each iteration of the optimization process to verify that the thrusts stays within the motors range capability [53]. Sine we are using a linearized controller, the roll and pitch angles are limited to be close to zero. Hence, the motor constraint is always verified. The constraint that we have to check iteratively will be the maximum and minimum allowed angles.

The second constraint that must be taken into account is the time of each segment. Since the trajectory depends on time, this can activate the angles constraint. Therefore, the segment times must be limited by a minimum allowed time. A simple expression is used to calculate the minimum time for each segment.

$$T_i^{min} = \beta d_i \tag{5.12}$$

Where d_i is the Euclidian distance between two waypoints, and β is a coefficient calculated by simulating the Quadrotor cruising along a straight-line trajectory between two waypoints with the maximum allowed angles.

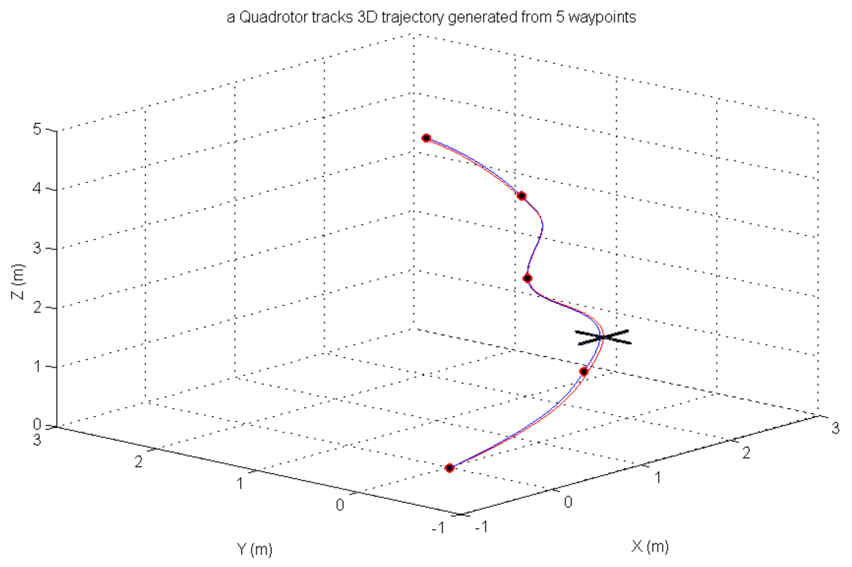


Figure 5.2: Tracking of an optimized trajectory generated from 5 waypoint using the modified cost function

5.5 Simulation results

In this section, we will present a software validation of the proposed scheme. All the simulations are performed on a computer with Intel Core i3 2.30GHz, and 4GB of RAM, using MATLAB version (2014a). The Quadrotor parameters are listed in the Table 2.1.

Since the decision vector is composed of derivative conditions in each waypoint, the angle limits can be transformed to derivative limits using the dynamical model. But the angle limits constraint is not sure to be verified along the polynomial segments. Hence, an iteration test is required, which add more computation. The used PD controller can support up to 10 for pitch and roll angles, in order to avoid the iteration testing for the angle limits constraint we set the limits to 5^o. After hundreds of simulations, 98% of the generated trajectories remain in the angle range supported by the controller. Fig. 5.2 shows the Quadrotor tracks a 3D trajectory generated from 5 waypoints. We note that the waypoints are generated manually in free of obstacles 3D space.

Table 5.2 shows the results corresponding to three samples of waypoints, $n = 5$, $n = 50$, and $n = 100$, where the execution time and the trajectory length are computed for two sets of the cost function terms: (time + snap) and (time + snap + distance). From the results presented in Table 5.2, it is clear that the used multi-objective PSO succeeded

Table 5.1: The execution time and the trajectory length in function of the waypoints number. Where CF is the cost function, S for the snap term, d for the distance term and t for the time term

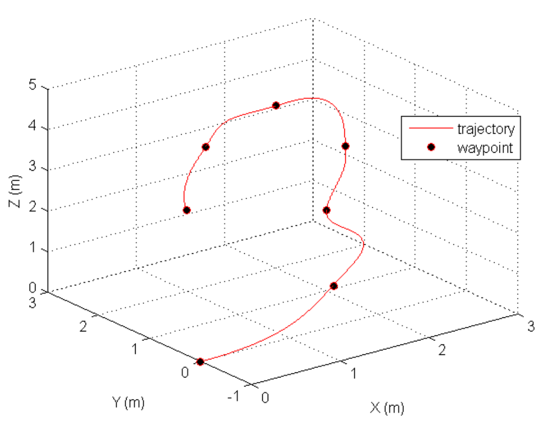
CF terms	S + t		S + d + t	
N	$T_{ex}\,(s)$	$L_{path}\,(m)$	$T_{ex}\,(s)$	$L_{path}\,(m)$
5	8.493	6.135	11.112	5.848
50	326.80	39.476	569.32	29.165
100	1804.28	70.352	3352.78	61.65

to solve the trajectory optimization problem for all the different samples of waypoints. In term of trajectory length, the use of distance term in the cost function gives shortest paths for all the sample of waypoints. However, the execution time in the absence of the distance term in the cost function is small comparing to when it is present.

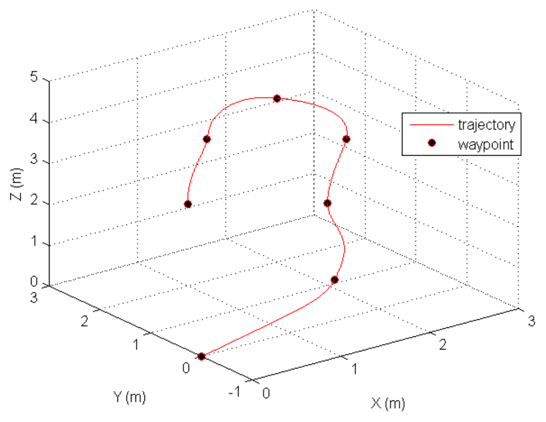
Fig. 5.3 shows two paths generated for the same 7 waypoints, without and with the distance term. From the results, it is clear that the use of the distance term in the cost function gives short trajectories. However, this increases the execution time of the optimization process proportionally to the number of waypoints.

5.6 Conclusion

This chapter presented an application of a multi objective PSO to minimize the snap of polynomial trajectories for Quadrotors. A linearized PD controller is used to track the trajectories. The distance term is added to the cost function to improve the quality of the trajectory. Simulation results show the efficiency of the algorithm in solving the trajectory optimization problem for different number of waypoints. The generated trajectories with the distance term in the cost function had the best quality, but it takes more execution time.

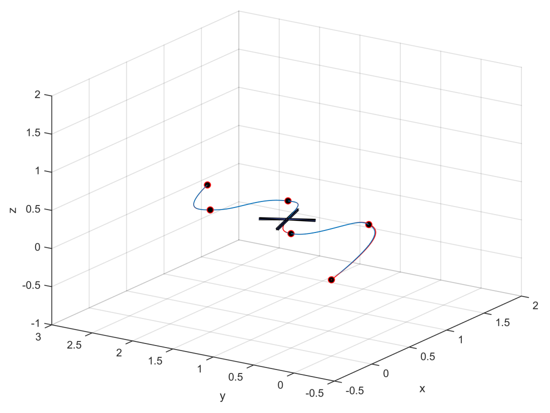


(a)

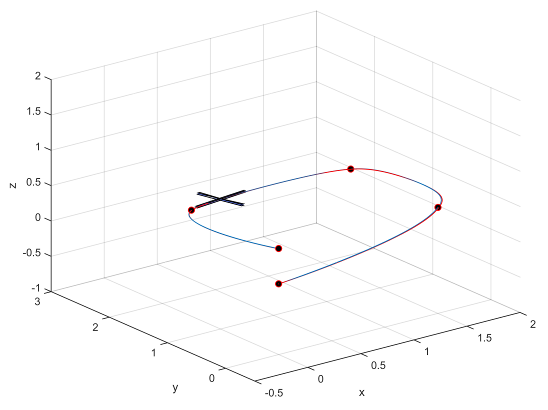


(b)

Figure 5.3: Trajectories generated from 7 waypoints: (a) minimum snap and time trajectory 9.02m, (b) minimum snap, time and distance trajectory 8.61m



(a)



(b)

Figure 5.4: Tracking of different shape of 3D trajectories: (a) wave form generated from 6 waypoints, (b) square form generated from 5 waypoints

Conclusion and future work

The topic of this thesis has been the development of new techniques in motion planning for UAVs, more specifically for Quadrotors. Path planning and trajectory generation are two important areas for the development of autonomous systems. Path planning tries to obtain a sequence of vehicle configurations (waypoints) between an initial configuration and a final configuration that avoids collision and minimize certain objective function. Trajectory generation aims at obtaining a time parametrized path that describe the motion along the path.

The contributions of this thesis have been presented in both stages: path planning and trajectory planning. In the path planning, a collision checking algorithm has been developed for sampling-based motion planning to address the problem of high number of obstacles. The method disassociates the dependence of time complexity with number of obstacles, By performing a pre-processing that stores information about the obstacles, then using them in the collision checking queries. We have also presented a new variant of RRT* algorithm named NPRRT* to address mainly the problem of narrow passages and convergence rate to optimal solution. Also, it addresses the memory consumption limit. We have introduced a steer function that explores more efficiently the difficult parts of the configuration space. A proposed path optimization technique has been used to speed up the convergence rate by generating configurations near the found path.

In the trajectory planning, the generation of 3D trajectory for Quadrotors has been implemented using PSO algorithm. The algorithm succeeded to minimize the snap, time and distance for polynomial trajectories. Where, a PD controller has been used to track the generated trajectory.

During this thesis a number of issues that require further investigation and consideration have been brought to attention and the most important are summarized here as suggestions for future works.

- In the presented collision checking method, the robot was considered to be punctual and the obstacles are polygons. As future work, the polyhedron shape for robot and obstacles will be considered for 3D environment.
- The future extensions for the NPRRT* may be the generalization of the method for higher dimensional space and improving the process of parameter selection. Also, the implementation of the method on real robots may require some improvements such as the smoothing of the generated path to adapt the kinodynamics of the robot.
- For the PSO implementation in the minimum snap and distance trajectory, future work would be using a non linear control in order to relax the near hovering conditions and studying the case of cluttered environments.

Bibliography

[1] Nonami, K., Kendoul, F., Suzuki, S., Wang, W. and Nakazawa, D., (2010). *Autonomous flying robots: unmanned aerial vehicles and micro aerial vehicles*, Springer Science, Business Media.

[2] Valavanis, K. P., Vachtsevanos, G. J. *Handbook of unmanned aerial vehicles*, Dordrecht, The Netherlands: Springer, pp. 2993–3009, 2015.

[3] Yanushevsky, R. (2011). *Guidance of unmanned aerial vehicles.* CRC press.

[4] B. Clough, (2002) 'Metrics schmetrics, How do you track a UAV's autonomy?', *In Proceedings of the AIAA 1st Technical Conference and Workshop on Unmanned Aerospace Vehicles*, Portsmouth.

[5] Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., Grixa, I.L., Ruess, F., Suppa, M. and Burschka, D., (2012) 'Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue'. *IEEE robotics, automation magazine*, Vol. 19, No. 3, pp.46-56.

[6] Lim, H., Park, J., Lee, D., Kim, H. J. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 33–45, 2012.

[7] Derafa, L., Madani, T., Benallegue, A., (2006) 'Dynamic modelling and experimental identification of four rotor helicopter parameters', IEEE-ICIT Mumbai, India, pp. 1834- 1839

[8] Choset, H., Hutchinson, S., Lynch, K., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005) *Principles of robot motion: theory, algorithms, and implementation, MIT press.*

[9] Karaman, S. and Frazzoli, E. (2011) 'Sampling-based algorithms for optimal motion planning', *The international journal of robotics research*, Vol. 30, No. 7, pp.846-894.

[10] Goerzen, C., Kong, Z. and Mettler, B., (2010). 'A survey of motion planning algorithms from the perspective of autonomous UAV guidance'.*Journal of Intelligent and Robotic Systems*, Vol. 57, No. 1, pp.65-100.

[11] LaValle, S.M. (2006) *Planning algorithms*, Cambridge university press.

[12] Khebbache, H. (2018). Tolérance aux défauts via la méthode backstepping des systèmes non linéaires: application système UAV de type quadrirotor (Doctoral dissertation).

[13] Mellinger, D. (2012). Trajectory generation and control for quadrotors. University of Pennsylvania.

[14] Kavraki, L.E., Svestka, P., Latombe, J.C. and Overmars, M.H., 1996. 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces'.*IEEE transactions on Robotics and Automation*, Vol. 12, No. 4, pp.566-580.

[15] LaValle, S.M., (1998). 'Rapidly-exploring random trees: A new tool for path planning'. *Computer Science Dept, Iowa State University.*

[16] Hauser, K., (2015). 'Lazy collision checking in asymptotically-optimal motion planning'. In *IEEE nternational conference on robotics and automation (ICRA)*, pp.2951-2957.

[17] Bialkowski, J., Karaman, S., Otte, M. and Frazzoli, E., (2013). 'Efficient collision checking in sampling-based motion planning'. In *Algorithmic Foundations of Robotics X*, pp.365-380, Springer, Berlin.

[18] Pan, J. and Manocha, D., (2016). 'Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing'. *The International Journal of Robotics Research*, Vol. 35, No. 12, pp.1477-1496.

[19] Daee, P., Taheri, K. and Moradi, H., (2014). 'A sampling algorithm for reducing the number of collision checking in probabilistic roadmaps'. In *22nd Iranian Conference on Electrical Engineering (ICEE)*, pp.1313-1316, IEEE.

[20] Belaid, A. and Mendil, B. (2019) 'Reducing the collision checking time in cluttered environment for sampling based motion planning', *Proceedings of the 4th International Conference on Electrical Engineering and Control Applications*, pp.978-981.

[21] Gonzalez, D., Perez, J., Milanes, V. and Nashashibi, F. (2016) 'A review of motion planning techniques for automated vehicles', *IEEE Transactions on Intelligent Transportation Systems*, Vol. 17, No. 4, pp.1135-1145.

[22] Khatib, O. (1986) 'Real-time obstacle avoidance for manipulators and mobile robots', *International Journal of Robotics Research*, Vol. 5, No. 1, pp.90-98.

[23] Lingelbach, F. (2004) 'Path planning using probabilistic cell decomposition', *In IEEE International Conference on Robotics and Automation*, pp.467-472.

[24] Hart, P.E., Nilsson, N.J. and Raphael, B. (1968) 'A formal basis for the heuristic determination of minimum cost paths, *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp.100-107.

[25] Dijkstra, E.W. (1959) 'A note on two problems in connexion with graphs', *Numerische mathematik*, Vol. 1, No. 1, pp.269-271.

[26] Paden, B., Cap, M., Yong, S.Z., Yershov, D. and Frazzoli, E. (2016) 'A survey of motion planning and control techniques for self-driving urban vehicles', *IEEE Transactions on intelligent vehicles*, Vol. 1, No. 1, pp.33-55.

[27] Belaid, A., Mendil, B. and Djenadi, A. (2021) 'Narrow Passage RRT*: A new variant of RRT*', *Int. J. Computational Vision and Robotics*, Vol. 12, No. 1, pp.85-100.

[28] Noreen, I., Khan, A. and Habib, Z. (2016) 'Optimal path planning using RRT* based approaches: a survey and future directions', *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 11, pp.97-107.

[29] Karaman, S., Walter, M.R., Perez, A., Frazzoli, E. and Teller, S. (2011) 'Anytime motion planning using the RRT', *In IEEE International Conference on Robotics and Automation*, pp.1478-1483.

[30] Adiyatov, O. and Varol, H.A. (2013) 'Rapidly-exploring random tree based memory efficient motion planning', *In IEEE International Conference on Mechatronics and Automation*, pp.354-359.

[31] Tong, B., Liu, Q. and Dai, C. (2019) 'A RRT* FN Based Path Replanning Algorithm', *In 2019 IEEE 4th Advanced Information Technology Electronic and Automation Control Conference*, pp.1435-1445.

[32] Jiang, J. and Wu, K. (2020) 'Multi-agent Path Planning based on MA-RRT* Fixed Nodes', *In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp.1875-1877.

[33] Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M. and Muhammad, M.S. (2013) 'RRT*-SMART: A rapid convergence implementation of RRT', *International Journal of Advanced Robotic Systems*, Vol. 10, No. 7, p.299.

[34] Gammell, J.D., Srinivasa, S.S. and Barfoot, T.D. (2014) 'September. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic', *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.2997-3004.

[35] Ryu, H. and Park, Y. (2019) 'Improved informed RRT* using gridmap skeletonization for mobile robot path planning', *International Journal of Precision Engineering and Manufacturing*, Vol. 20, No. 11, pp.2033-2039.

[36] Luo, S., Liu, S., Zhang, B. and Zhong, C. (2017) 'Path planning algorithm based on Gb informed RRT* with heuristic bias', *In IEEE 36th Chinese Control Conference*, pp.6891-6896.

[37] Meng, J., Pawar, V.M., Kay, S. and Li, A. (2018) 'UAV Path Planning System Based on 3D Informed RRT* for Dynamic Obstacle Avoidance', *In IEEE International Conference on Robotics and Biomimetics*, pp.1653-1658.

[38] Mohammed, H., Jaradat, M. and Romdhane, L. (2018) 'RRT* N: An improved rapidly-exploring random tree approach for reduced processing times', *In 11th International Symposium on Mechatronics and its Applications*, pp.1-6.

[39] Jeong, I.B., Lee, S.J. and Kim, J.H. (2019) 'Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate', *Expert Systems with Applications*, Vol. 123, No. 1, pp.82-90.

[40] Mashayekhi, R., Idris, M.Y.I., Anisi, M.H., Ahmedy, I. and Ali, I. (2020) 'Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method', *IEEE Access*, Vol. 8, No 1, pp.19842-19852.

[41] Li, Y., Wei, W., Gao, Y., Wang, D. and Fan, Z. (2020) 'PQ-RRT*: An improved path planning algorithm for mobile robots', *Expert Systems with Applications*, p.113425.

[42] Qureshi, A.H. and Ayaz, Y. (2016) 'Potential functions based sampling heuristic for optimal path planning', *Autonomous Robots*, Vol. 40, No. 6, pp.1079-1093.

[43] Xinyu, W., Xiaojuan, L., Yong, G., Jiadong, S. and Rui, W. (2019) 'Bidirectional potential guided RRT* for motion planning', *IEEE Access*, Vol. 7, No. 1, pp.95034-95045.

[44] Kim, D., Kwon, Y. and Yoon, S.E. (2018) 'Adaptive lazy collision checking for optimal sampling-based motion planning', In 15th International Conference on Ubiquitous Robots (UR), pp.320-327.

[45] Boor, V., Overmars, M.H. and Van Der Stappen, A.F. (1999) 'The Gaussian sampling strategy for probabilistic roadmap planners', *In Proceedings IEEE International Conference on Robotics and Automation*, pp.1018-1023.

[46] Canny, J. (1988) *The complexity of robot motion planning.* MIT press, USA.

[47] Bouktir, Y., Haddad, M. and Chettibi, T., (2008) 'Trajectory planning for a quadrotor helicopter'. In *16th mediterranean conference on control and automation*, pp.1258-1263.

[48] Cowling, I.D., Whidborne, J.F. and Cooke, A.K., (2006) 'Optimal trajectory planning and LQR control for a quadrotor UAV'. In *International Conference on Control.*

[49] Lai, L. C., Yang, C. C., Wu, C. J. (2006) 'Time-optimal control of a hovering quadrotor helicopter'. *Journal of Intelligent and Robotic Systems*, vol. 45, No. 2, pp.115-135.

[50] Culligan, K., Valenti, M., Kuwata, Y., How, J. P. (2007) 'Three-dimensional flight experiments using on-line mixed-integer linear programming trajectory optimization'. In *American Control Conference*, IEEE, pp.5322-5327.

[51] Tang, S., Kumar, V. (2018) 'Autonomous flight'. *Annual Review of Control, Robotics, and Autonomous Systems*, Vol. 1, pp.29-52.

[52] Mellinger, D., Kumar, V. (2011) 'Minimum snap trajectory generation and control for quadrotors'. In *IEEE International Conference on Robotics and Automation*, IEEE, pp.2520-2525.

[53] Richter, C., Bry, A. and Roy, N., (2016) 'Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments'. In *Robotics research*, pp.649-666. Springer.

[54] Zhou, A., Qu, B.Y., Li, H., Zhao, S.Z., Suganthan, P.N. and Zhang, Q., (2011) 'Multiobjective evolutionary algorithms: A survey of the state of the art'. *Swarm and Evolutionary Computation*, Vol. 1, No. 1, pp.32-49.

[55] Macharet, D.G., Neto, A.A. and Campos, M.F.M., (2010) 'Feasible UAV path planning using genetic algorithms and Bézier curves'. In *Brazilian Symposium on Artificial Intelligence*, pp.223-232, Springer, Berlin.

[56] Bao, Y., Fu, X. and Gao, X., (2010) 'Path planning for reconnaissance UAV based on particle swarm optimization'. In *Second International Conference on Computational Intelligence and Natural Computing*, Vol. 2, pp.28-32. IEEE.

[57] Nikolos, I.K., Valavanis, K.P., Tsourveloudis, N.C. and Kostaras, A.N., (2003) 'Evolutionary algorithm based offline/online path planner for UAV navigation'. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 33, No. 6, pp.898-912.

[58] Eberhart, R., and Kennedy, J., (1995) 'Particle swarm optimization'. In *Proceedings of the IEEE international conference on neural networks*, Vol. 4, pp.1942-1948.

[59] Coello, C. A. C., Lamont, G. B., and Van Veldhuizen, D. A., (2007). *Evolutionary algorithms for solving multi-objective problems*, Vol. 5, pp. 79-104. New York: Springer.

[60] Baumgartner, U., Magele, C., and Renhart, W., (2004) 'Pareto optimality and particle swarm optimization'. *IEEE Transactions on magnetics*, Vol. 40, No. 2, p.1172-1175.

Abstract:

This thesis addresses the development of trajectory planning algorithms for unmanned aerial vehicles (UAVs). It focuses on the motion planning for Quadrotor type and contributes in the both stages: the path planning and the trajectory planning. In path planning, a technique is proposed to reduce the collision checking time in cluttered environment for sampling-based path planning. The method disassociates the dependence of time complexity with number of obstacles, by performing a pre-processing that stores information about the obstacles, then using them in the collision checking queries. Also, we propose a new variant of Rapidly Exploring Random Trees Star (RRT*) named NP-RRT* to deal mainly with narrow passages in the workspace and to speed up the convergence rate to optimal solution, also, to minimize the memory consumption. In the trajectory planning, we implement the Particle Swarm Optimization (PSO) to minimize the snap and distance trajectory for Quadrotor. The minimum snap polynomial trajectories are the natural choice for Quadrotors, since the motor commands and attitude accelerations of the vehicle are proportional to the snap of the path.

Résumé :

Cette thèse porte sur le développement d'algorithmes de planification de trajectoires pour les véhicules aériens sans pilote (UAV). La thèse porte sur la planification de mouvements pour le type Quadrotor. La contribution concerne les deux étapes : la planification du chemin et la planification de trajectoires. Dans la planification du chemin, une technique est proposée pour réduire le temps de test de collision dans des environnements encombrés pour les méthodes de planification basée sur l'échantillonnage. Le procédé dissocie la dépendance de la complexité temporelle du nombre d'obstacles, en effectuant un prétraitement qui stocke des informations sur les obstacles, puis en les utilisant dans les requêtes de vérification de collision. De plus, nous proposons une nouvelle variante de l’algorithme RRT* (pour : Rapidly Exploring Random Trees Star) nommée NP-RRT* pour traiter principalement les passages étroits dans l'espace de travail et accélérer la convergence vers une solution optimale, ainsi que minimiser la consommation de mémoire. Dans la planification de trajectoires, nous implémentons la méthode (PSO) pour minimiser l’énergie et la distance pour les Quad-rotors. Les trajectoires polynomiales en minimisant la quatrième dérivée sont le choix naturel pour les Quad-rotors, puisque les commandes du moteur et les accélérations d'attitude du véhicule sont proportionnelles à la quatrième dérivée de la trajectoire.

ملخص :

تتناول هذه الأطروحة تطوير خوارزميات تخطيط المسار للمركبات الجوية غير المأهولة. تركز الأطروحة على تخطيط الحركة للنوع رباعي المحركات. حيث تساهم الأطروحة على مستويين: تخطيط المسار وتخطيط الحركة. في تخطيط المسار نقترح تقنية لتقليل وقت فحص التصادم في بيئة مزدحمة لطرق تخطيط المسار القائمة على أخذ العينات. حيث أن الطريقة تفصل علاقة عدد المعيقات مع وقت فحص التصادم وهذا بتخزين معلومات حول البيئة مسبقا، ثم يتم استعمال هذه المعلومات عند فحص التصادم. أيضًا، نقترح نسخة مطورة لطريقة شجرة الاستكشاف السريعة العشوائية النجمية. والمسماة طريقة شجرة الاستكشاف السريعة العشوائية النجمية للممرات الضيقة، أين تتعامل هذه الأخيرة بشكل أساسي مع الممرات الضيقة في مساحة العمل وتسريع عملية الحصول على الحل الأمثل ، وكذلك تقليل استهلاك الذاكرة. في تخطيط الحركة نطبق طريقة سرب الجسيمات للتحسين في تقليل الطاقة والمسافة لرباعي المحركات. حيث أن استعمال المسارات محسنة المشتق الرابع هو الاختيار الطبيعي لرباعيات المحركات لأن التسارع يتناسب طرديا مع هذه المسارات.