UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN CONTROL SYSTEMS ENGINEERING

Architecture and Control design of a legged quadrotor for landing on sloped surfaces

Relatore: Prof. Cenedese Angelo
Correlatrice: Dott. Michieletto Giulia
Correlatore: Dott. Antonello Riccardo

Laureando: Danieli Alessio

ANNO ACCADEMICO 2022-2023
Data di laurea 5/03/2024

Abstract

This study aims to optimize the landing operation of a quadcopter on a sloped surface, a condition that is usually unfeasible for standard quadcopters. Starting from a previous research, a redesign of the landing structure has been performed. The goal of the new design is to allow a safe landing on an inclined surface characterized by high slope. To ensure a lighter structure, so that the flying capabilities of the quadrotor are not significantly compromised and with the purpose of optimization, only one actuator has been used for moving the landing skids.

The initial phase of the study focused on the design of the structure, the mechanisms for its movement, and the positioning of the motor. An analysis to assess the optimality of the solution was conducted in MATLAB environment. This analysis involved mathematical functions relating the slope of the landing surface, the angular range of motion of the skids and the limits imposed to avoid the collision between the drone body and the landing surface.

The second phase concentrated on the development of a software for drone control using ROS2. This consisted in designing nodes to control the drone movement, the skids movement and perform the slope detection. The slope detection was performed with a Time of Flight (ToF) sensor positioned along the drone heading direction, facing downward. To verify the efficiency of the solution the algorithms have been tested in the Gazebo simulation environment.

The planned next phase involves the implementation of the solution on a real quadrotor and test it in a laboratory, comparing simulative and real-world results.

Dedication

I want to thank my family for supporting me during these years of study.
A special thank you to Martina for always being by my side during these difficult years.

Contents

List of Figures

Chapter 1

Introduction

In recent years, the use of multirotor unmanned aerial vehicles (UAVs) has been consistently growing in various fields, thanks to their numerous qualities. They can be either operated remotely by a pilot on the ground or fly autonomously using pre-programmed missions. Over the years several different types of categorizations have been proposed. UAVs can be categorized taking into considerations their dimensions and weight, their fly capabilities and autonomy or a risk based approach.

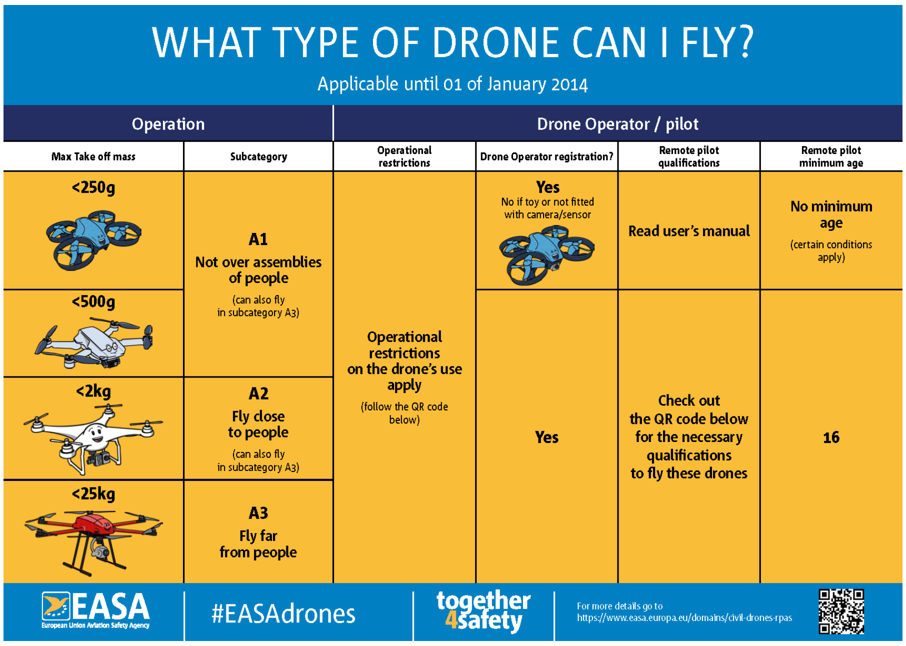
The risk based approach is used by the European Union Aviation Safety Agency [5], which categorizes UAVs based on their weight and usage. This regulation is applied in all the countries of the European Union and specifies three main categories: open , specific and certified.

- The open category is the one with lower risk, it outlines rules for drones with weight up to twenty-five kilograms, divided in 3 sub-categories based on the weight, usage destination, and considerations on the distance of the airspace from people or residential areas. A schematic representation of this specific class can be seen in figure 1.1.

- The specific category defines a higher level of risk with respect to the open category. Examples of activities falling under this category include UAV flights beyond visual line of sight, drones that drop materials or those with a mass greater than twenty-five kilograms. Operating drones in this category requires an authorization and a risk assessment.

- The last category caters for the operations with the highest level of risk, with safety requirements similar to those used for manned aviation. This last class will include drones carrying passengers such as a taxi, which is required to perform potentially hazardous tasks due to transporting people in residential environments. Beyond the categories of UAVs, this type of vehicles allows to easily complete every day tasks that in the past were impossible, more expensive or time consuming. They can easily reach positions that would be difficult to access otherwise.

Thanks to the technological progresses nowadays multicopters can have different sizes to accomplish various tasks. In some particular cases dimensional and weight reduction has allowed their usage, an example is the indoor application that was not possible with the first bigger models. Dimensional reduction also make multirotors usage more convenient and cheaper. An example could be an outdoor photo or video



WHAT TYPE OF DRONE CAN I FLY?

Applicable until 01 of January 2014

Operation		Drone Operator / pilot			
Max Take off mass	Subcategory	Operational restrictions	Drone Operator registration?	Remote pilot qualifications	Remote pilot minimum age
<250g	A1 Not over assemblies of people (can also fly in subcategory A3)	Operational restrictions on the drone's use apply (follow the QR code below)	Yes No if toy or not fitted with camera/sensor	Read user's manual	No minimum age (certain conditions apply)
<500g			Yes	Check out the QR code below for the necessary qualifications to fly these drones	16
<2kg	A2 Fly close to people (can also fly in subcategory A3)				
<25kg	A3 Fly far from people				

EASA European Union Aviation Safety Agency | #EASAdrones | together 4safety | For more details go to https://www.easa.europa.eu/domains/civil-drones-rpas

Figure 1.1: Subcategories in the open category defined by the European Union Aviation Safety Agency

shoot that nowadays can be made with drones not larger than a dozen of centimetres, which can be easily transported without requiring special equipment because of their reduced dimensions. They are also capable of moving very quickly from point to point and performing agile manoeuvres. Another advantage is the characteristic of being unmanned, making them useful for dangerous tasks. These are only a few of the qualities that make multicopters suitable for many different applications like photograph/videomaking, package delivery, reconnaissance, surveillance, crop inspection, etc. . For these reasons the number of studies on this field has also being consistently growing [1] in order to constantly increase performance and address common problems. Examples can be found in the usage of drones for Search and Rescue tasks [2]: in particular for these tasks the ability of the UAV to scan a wide area in a small amount of time is crucial. Another recently developed application is the inspection of wind power plants [3], in which UAV usage allows to avoid the deployment of specialised technicians to inspect the growing number of plants, thereby increasing human safety, allowing also time and money saving. UAVs perform great also in forest wildfires detection [15], providing reliable and fast identification on wide areas, increasing prevention and control of forest fire.

1.1 Motivations of the thesis

In different fields of application, the necessity of a landing on a non ideal flat smooth surface could arise. In recent years several researches have been conducted to address this issue, testing various solutions in different scenarios. An example is the use of the multirotor weight for perching on a pipe or a branch [12]: this solution avoids contact with the ground, eliminating the need to compensate for rough terrain. The use of a passive mechanism permits to save battery and weight, but it makes it impossible to land in a scenario where there are no structures similar to

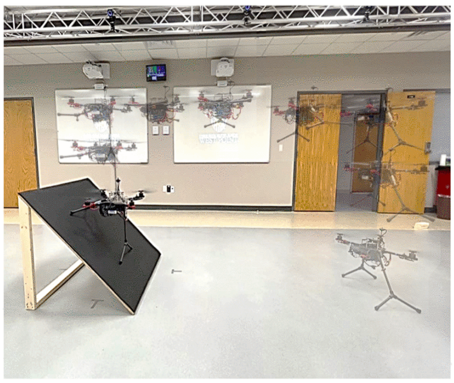


Figure 1.2: Solution proposed in [13]

pipes or branches where the multicopter can perch. Another approach for perching and taking-off from vertical walls is presented in [16], for micro air vehicle using dry adhesive gripper. The approach was tested and proved to be effective and efficient, requiring simple manoeuvres to perch. Also in this case drawbacks are the impossibility to safely land in absence of a smooth vertical surface to which the adhesive can stick on, situation that could arise in outdoor applications, and the challenges posed by applications for larger vehicles. An example in case of applications on rough terrain, a landing could be required as failsafe action, or for waiting for another task while saving battery charge. To address this issue, an additional structure can be added to the main body of the multirotor, in order to compensate for the uneven terrain. In order to not excessively compromise the flight performance of the multirotor, the structure has to be designed to be light and possibly symmetric for not moving significantly the center of mass of the entire structure, which would lead to a less stable body. In order to reduce the additional weight, the aim is to design a compact lightweight structure, with a minimum number of actuators.

1.2 Related work

Different articles studied the problem of landing on tilted surfaces, presenting different solutions. The solution presented in [13] uses an asymmetric leg structure with two linear actuators. The proposed structure is represented in figure 1.2 during a flight test. The actuators through a special mechanism allow to move the landing skid. The target angles for the legs are calculated using an objective function. The function fixes 15 degrees as target angle for both the legs, considering also the distance between the propeller and the ground. The proposed solution achieves a safe landing on surfaces up to 45 degrees.

A different structure is presented in [17] (figure 1.4). This solution is composed by two different parts, a passive skid structure and a two link arm. The choice of a passive skid was made in order to reduce electricity consumption, thus eliminating the need to carry additional batteries and reducing the overall weight. The landing procedure involves the contact of the passive mechanism with the ground, ensuring two contact points. Subsequently the robotic arm is used to provide the third contact point achieving stability while maintaining the robot body horizontal. The presented solution ensures a safe landing on surfaces tilted with respect to two different axes, achieving landing for angles between 20 and -60 degrees for the first angle and

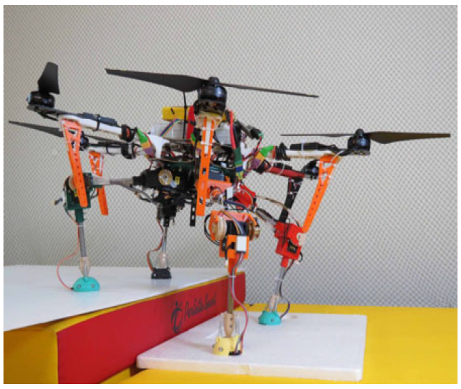


Figure 1.3: Solution proposed in [18]

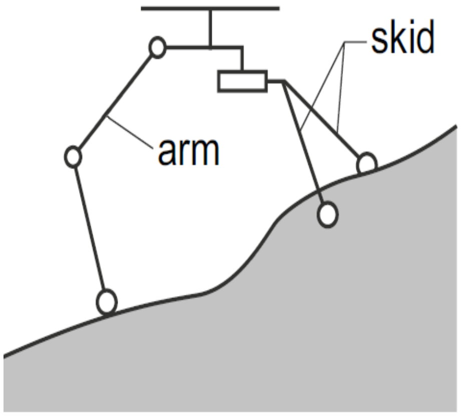


Figure 1.4: Example of asymmetric structure with multiple actuators proposed in [17]

between ±60 degrees for the second angle.

Another approach is the one presented in [18], in which four robotic legs with two degrees of freedom are used (figure 1.3). The legs are equipped with force and torques sensors to perceive the external disturbances. In addition the leg tips are equipped with movable pads in order to add compliance to the leg for uneven terrain. The solution achieved remarkable results in landing on slopes up to 11 degrees and on small steps.

In particular our project starts from a previous research [14], in which the usage of a movable skid allows the drone to perform a safe landing on tilted surfaces up to 25 degrees. It also incorporates a Time-of-Flight (TOF) sensor to perceive the inclination of the landing surface, and determine the slope direction, in order to automatically position the quadrotor accordingly.

1.3 Aims of the Project

This thesis project wants to bring an improvement to the previous work [14]. The aim is to achieve a safe landing on surfaces tilted up to 60 degrees, which is a result not presented in other papers in the literature. Another improvement is testing the designed mechanism on a real quadrotor after a successful simulation. To accomplish this, a new landing structure has been realized, with a different design that allows the skids to be positioned above the drone body line. A study has been conducted to create skids with a minimal length in order to reduce the weight. Furthermore in order to minimize the number of actuators, the structure is moved by only one servo motor that acts directly on one skid. A reduction gear enables the transmission of motion to the second skid. Such reduction gear has been designed to ensure the full range of motion required. The Time-of-Flight sensor is maintained for same purpose of [14], so that the multirotor can automatically align its yaw angle and position the skids accordingly to the performed detection of the landing surface.

Chapter 2

Leg study

The first objective for the landing gear design was to establish an optimality criterion, upon which the design itself would be based.

The first design choice was to have symmetric legs. The aim of this choice was to lower the impact on the inertia matrix of the system. To do so an important factor to consider is the length of the legs, which obviously should be as short as possible, ensuring that the multirotor flying capabilities are not significantly affected. Consequently the system will be as lightweight as possible, also reducing battery consumption.

The second parameter being addressed is the maximum angular displacement of the legs and their distance from the vertical symmetric plane of the system. These parameters impact the drone landing capabilities, and also its stability once landed. Clearly, a larger distance of the legs from the system's vertical symmetric plane will result in a more stable system once landed. However this would require longer legs to reach a safe landing, given the angle of the landing surface.

In figure 2.1 it can be seen a schematic view of the previously cited parameters. It can be noticed that there are two different pairs of legs in the figure. The first pair is coloured in blue, the second is coloured in red. The parameters required for the two different configurations are indicated with the respective colour. It can be easily seen that the red legs have a larger distance from the vertical symmetric plane of the quadrotor body. The legs with the subscript 2 have the same length both in the red and in the blue solution, but the one in red requires a higher angular displacement θ_2 for the same surface slope φ. For the leg with subscript 1 it is shown that the red one requires a greater length for the same angular displacement θ_1.

The aim of the study is to determine a trade-off between these quantities. The system has been equipped with only one servomotor to move both legs in order to have less weight. The next steps of the design will be the creation of the mechanism to transmit motion between the legs.

2.1 Landing Dynamics

The legs have been positioned on the sides of the multicopter along the y-axis, where the x-axis is the heading direction (figure 2.2). Otherwise, if positioned along the x-axis, they would interfere with the camera sensor positioned at the front of the body.

The first step of the design was find the relationship between the slope angle

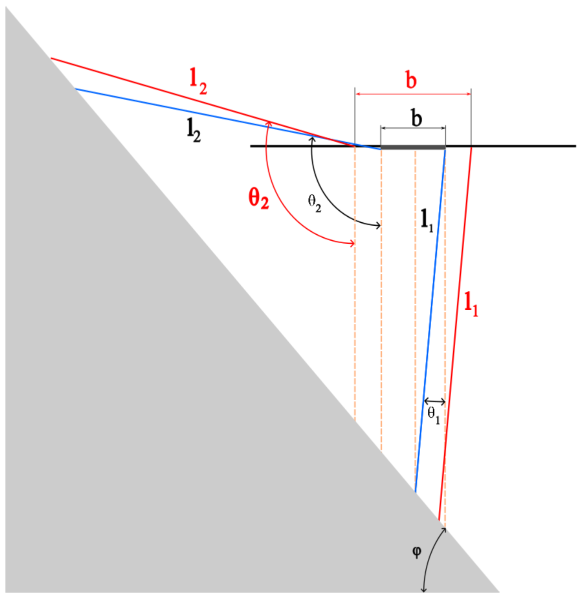


Figure 2.1: Schematic view of the main parameters object of the design

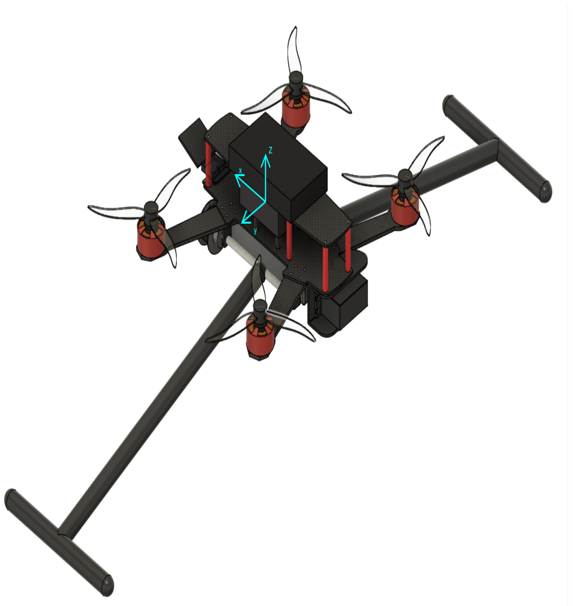


Figure 2.2: Drone model with legs attatched along the y-axis

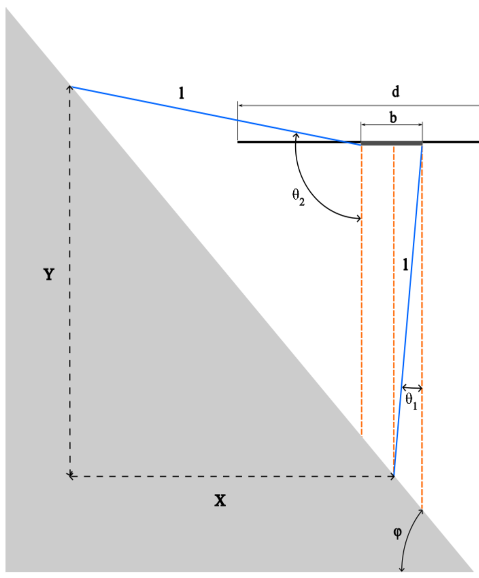


Figure 2.3: Side geometric scheme

of the landing surface and the 2 control parameters θ_1 and θ_2 (figure 2.3). Notice that the angular displacements of the legs are considered positive in the clockwise direction and negative in the counterclockwise direction. Notice also the numbering of the legs : leg with subscript 1 is toward the lower part of the slope while leg with subscript 2 is towards the higher part of the slope. The slope angle is indicated with the letter φ. The drone dimension, meaning the horizontal distance between the propellers tips is indicated with the letter d. The distance between the legs joints is called b. To find the relationship between the slope of the surface and the legs angular displacement the problem can be thought in two dimensions, considering the horizontal and vertical distance between the legs tips, respectively the quantities indicated with the letters x and y in figure 2.3.

The following relations hold:

$$\varphi = \arctan\left(\frac{y}{x}\right) \tag{2.1}$$

$$x = l\sin(\theta_2) - l\sin(\theta_1) + b \tag{2.2}$$

$$y = l\cos(\theta_1) - l\cos(\theta_2) \tag{2.3}$$

$$\begin{aligned}\varphi &= \arctan\left(\frac{l\cos(\theta_1) - l\cos(\theta_2)}{l\sin(\theta_2) - l\sin(\theta_1) + b}\right) \\ &= \arctan\left(\frac{\cos(\theta_1) - \cos(\theta_2)}{\sin(\theta_2) - \sin(\theta_1) + \frac{b}{l}}\right)\end{aligned} \tag{2.4}$$

$$l = \frac{b\tan(\varphi_{target})}{cos(\theta_1) - cos(\theta_2) + \tan(\varphi_{target})\sin(\theta_1) - \tan(\varphi_{target})\sin(\theta_2)} \tag{2.5}$$

In the equation (2.5) we set, as previously said, the φ_{target} to 60°. The angular displacements θ_1 and θ_2 for a landing on an horizontal plane are fixed to:

$$\theta_{1min} = -85^{\circ} \tag{2.6}$$

$$\theta_{2min} = 85^{\circ} \tag{2.7}$$

The choice of these two values has been dictated by the purpose of maintaining the barycentre of the system al low as possible reducing the risk of rollover. Clearly

this risk is increasing proportionally with the height of the barycentre and with the proximity of the centre of mass to the boundaries of the support surface. On an horizontal surface it can be easily verified that the height of the barycentre of the quadrotor lowers proportionally with the raise of the two leg displacements towards positive values for the leg number 2 and towards negative values for the leg number 1. These parameters has been chosen with a safety margin of 5° from the $\pm 90^{\circ}$ displacement, a condition the latter, in which the joints of the legs are touching the surface, which has to be avoided.
The chosen configuration achieves maximum stability of the system by lowering the barycentre and maximizing the support surface.

For the maximum angular displacement of leg 1, a value of 5° was initially chosen. A greater angular displacement of leg number one would permit reaching higher slopes, keeping the leg length l and θ_{2max} fixed. However this would result in a smaller support surface and a higher center of mass. To verify the achievement of stability given θ_{1max} it's necessary to know the length of the legs, so the proof of stability will be shown later. The choice of the parameter θ_{2max} was made through a numerical analysis in MATLAB, evaluating among different values of the parameter and determining the required leg length needed to achieve a safe landing on the target slope. The result of this analysis produced a function relating θ_{2max} and l , from which was possible to choose the best parameters.

Another constraint was introduced to verify that the drone was not colliding with the plane (2.8 and 2.9). In particular the function (2.8), once fixed the drone dimensions (b and d) directly relates the two parameters θ_{2max} and l. In fact from geometrical relations involving the angle displacement θ_2 in the left part of the inequality it is retrieved the distance between the propeller tips and the projection on the horizontal plane of the leg length. Function 2.9 retrieves the minimal length for the leg in order to avoid the collision.

$$l\sin(\theta_2) - \frac{d-b}{2} + \frac{l\cos(\theta_2)}{\tan(\varphi)} > 0 \tag{2.8}$$

$$l > \frac{d-b}{2}\frac{\tan(\varphi)}{\tan(\varphi)\sin(\theta_2) + \cos(\theta_2)} \tag{2.9}$$

2.2 Design Results

The analysis for choosing the best possible parameters for θ_{2max} and l was done in MATLAB taking into account both functions (2.5 and 2.9). The former equation defines the minimum leg length required to achieve landing on the desired sloped surface, the latter establishes a lower limit for the length of the leg in order to avoid collisions between the landing surface and the propeller tips.

As shown in figure 2.4 an optimum trade-off value is given by a leg length of 286 mm and a θ_{2max} of 125°. It can be noticed that these values don't represent the point of intersection between the two curves, because the points on the blue curve coincides with the collision. In fact, as mentioned before, 2.9 is representing a lower limit, and the point of intersection between the curves represents the case in which the quadrotor will land on the surface, maintaining an horizontal body line, touching whit the propeller the landing surface. Condition that obviously has to be avoided. Given these results the legs length was fixed to l = 290 mm, and $\theta_{2max} = 125^{\circ}$

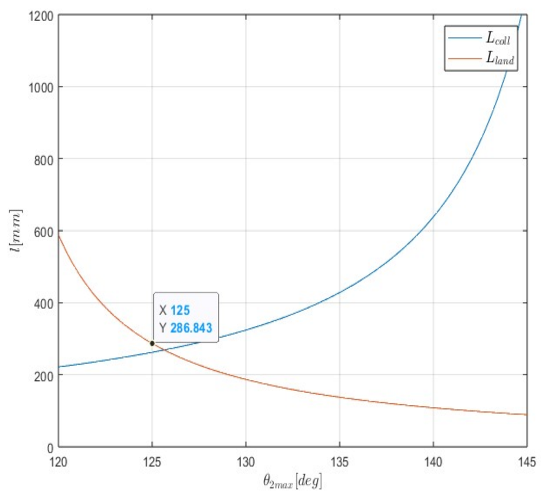


Figure 2.4: Landing and collision curves

Once defined the three parameters it is necessary to verify that with θ_{1max} stability is achieved. The condition for the barycenter's projection to fall inside the support surface is

$$\theta_{1max} < \arcsin(\frac{b}{2l}) = 4.945^{\circ} \tag{2.10}$$

This condition is not verified with the actual parameters, this will cause the multirotor to fall in the case of a landing on planes tilted by 60°. Consequentially it's necessary a slight change in the parameters in order to avoid such problem. Different options were considered:

- Bringing θ_{1max} to 0° will require a legs length greater than 30 cm. This solution is not preferable.
- Increasing only the length of the legs will not yield any benefits.
- Redefining $\theta_{1max} = 4^{\circ}$ and $\theta_{2max} = 128^{\circ}$ would permit reaching a safe landing up to 61° with the same legs length, while satisfying the stability condition (2.11). Despite this noticeable result, this would also bring the multirotor close to a collision. Therefore a different solution was chosen.

The adopted solution was bringing the parameter b (the distance between the legs) from $50mm$ to $60mm$. This solution meets all the requirements and a transmission was designed to place the legs joints slightly protruding from the drone body.

The new landing and collision curves can be seen in figure 2.5 with a final $\theta_{2max} = 127°$. With the new value of parameter b stability is achieved, in fact:

$$\theta_{1max} < \arcsin(\frac{b}{2l}) = 5.938^{\circ} \tag{2.11}$$

The last step is the design of the mechanism to transmit motion between the legs, this was realised with a belt transmission with a transmission ratio that satisfies the established angular excursions, given by:

$$\tau = \frac{\theta_{2max} - \theta_{2min}}{\theta_{1max} - \theta_{1min}} = \frac{7}{15} \tag{2.12}$$

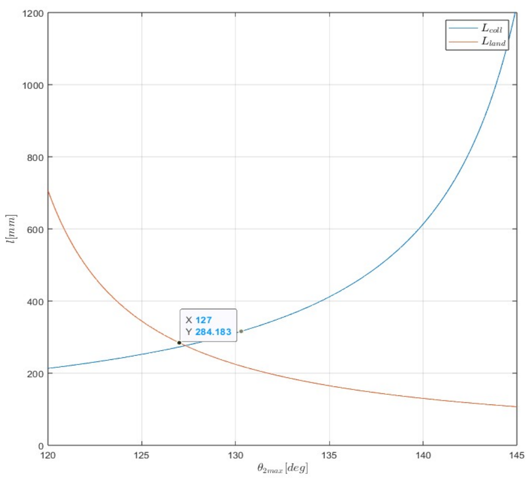


Figure 2.5: New landing and collision curves

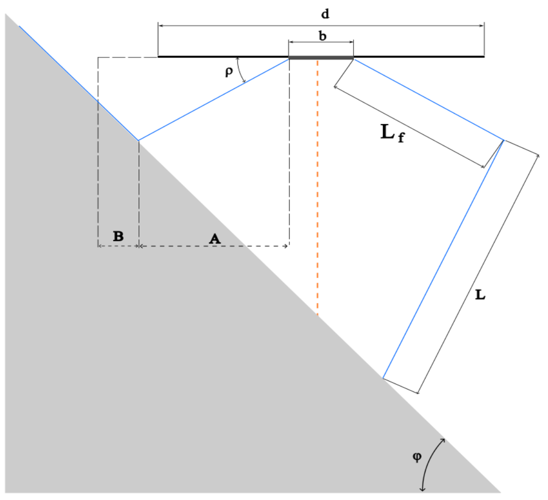


Figure 2.6: Side geometric scheme

2.3 Different solution

A different solution was studied. This second option consisted of a sort of "knee" in the legs, with a first section of the leg that is fixed to the body of the drone, and a second part connected to the first one with a revolute joint.

In order to compare this solution with the chosen one, the leg length limit was fixed to $290mm$, which had already been determined as the optimal solution. This configuration came from the idea of using one of the leg joints as a point of support for landing when the sloped surface has an inclination that was the maximum reachable. As in the previous case the slope angle is indicated with the letter φ. The horizontal distance between the propellers tips is indicated with the letter d. The distance between the legs fixed joints is called b. The fixed angle between the drone body and the first part of the legs is indicated with the letter ρ. With this configuration (figure 2.6) from geometrical relations, the non-collision condition is:

$$\tan(\varphi) < \frac{L_f sin(\rho)}{\frac{d-b}{2} - L_f cos(\rho)} \tag{2.13}$$

From this constraint it can be retrieved the minimum required length for L_f:

$$L_f > \frac{d-b}{2} \frac{\tan(\varphi)}{\tan(\varphi)\cos(\rho) + \sin(\rho)} \tag{2.14}$$

This represents a lower limit, ensuring the propeller is not colliding with the landing surface. Once verified the condition to avoid the collision with the landing surface, the second step is to verify that the configuration is capable to compensate for the required slope:

$$\tan(\varphi) = \frac{L\cos(\theta_1)}{2L_f\cos(\rho) - Lsin(\theta_1) + b} \tag{2.15}$$

Having imposed a limit to the length of the leg we have the following relation:

$$L = 290 - L_f \tag{2.16}$$

With the substitution of (2.16) in (2.15) we have:

$$L_f = \frac{290\tan(\varphi) + 290\cos(\theta_1) - b\tan(\varphi)}{2\tan(varphi)\cos(\rho) + \tan(\varphi)\sin(\theta_1) + \cos(\theta_1)} \tag{2.17}$$

Analysing the equation 2.17 it results that the value of θ_1 that minimises the length L_f is $\theta_1 = 0°$. As in the previous case the two presented constrains are evaluated in MATLAB in order to establish an optimal trade-off for the leg length. The mentioned constraints (2.14 and 2.17) don't define an intersection point for the minimum length. As it can be seen in figure 2.7, the constraint to avoid collision represents a higher lower limit with respect to the landing constraint. Therefore considering only the constraint 2.14, the best solution has been retrieved :

1. $L_f = 107mm$
2. $\rho = 30°$

From equation 2.15 the retrieved value for L is :

$$L_{=}425mm \tag{2.18}$$

As result of the analysis the solution with the "knee" requires legs divided in two section. The first section L_f with a length of $107mm$ and a second section L with length $425mm$. The sum of the two required lengths L_f and L it is almost doble the value required from the previous solution and it is clear as this solution is worst with respect to the first one presented. For this reason it has been discarded.

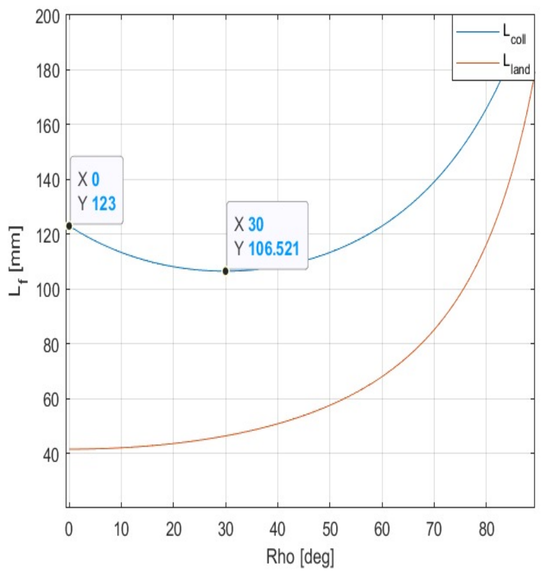


Figure 2.7: Landing and collision curves

2.4 Resulting landing system

The starting point of the design was the choice of a symmetric landing structure and the usage of a single actuator. With these starting constraints it has been developed a study to verify the feasibility of the solution.

With the results presented in this chapter it has been highlighted the optimality of the chosen solution, considering different parameters.

Some of them were fixed by the drone dimensions or initially fixed to a value in order to reduce the complexity of the analysis. In particular the distance between the propeller tips $d = 306mm$ is imposed by the structure of the purchased drone available in the laboratory, the value of the parameter b (distance between the legs joints) was initially fixed to $50mm$ that corresponds to the quadrotor main body dimension on y axis and the θ_{1max} value was fixed at 5°. This allowed to reduce the complexity of the equations, allowing the optimization of the legs length. The initial values were modified after the optimization shown in the previous section to the values reported in table 2.1.

	b [mm]	d [mm]	θ_{1max} [deg]	θ_{2max} [deg]	l [mm]
initial values	50	306	5	125	290
final values	60	306	5	127	290

Table 2.1: parameters values

The landing structure is composed of two symmetric legs of length 290 mm and two skids attached at the legs tips. The legs are moved with the usage of a single actuator and a transmission gear. A front view of the final design can be seen in figure 2.8, in the figure the transmission gears are highlighted in light blue, the time of flight (ToF) support is highlighted in violet.

The necessary transmission ration between the two legs is of $\frac{7}{15}$ that is sufficient to ensure the full range of motion needed for the two legs. Leg 1 has a minimum

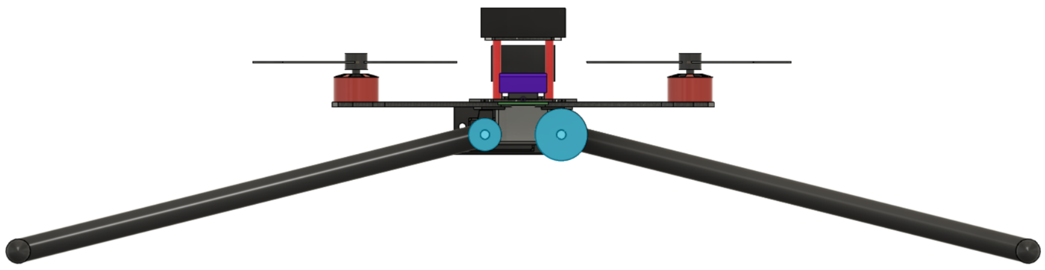


Figure 2.8: Front view of the proposed solution

angle of -85 degrees and a maximum angle of 5degrees. The range of motion for leg 2 is from 85 degrees up to 127 degrees. The servomotor it is placed in the back of the model, and the movements between the two highlighted gears can be transmitted with a belt (that for simplicity was not added to the model).

Chapter 3

Drone Model Dynamics and Control Architecture

3.1 Quadrotor Model

The quadrotor is composed of a main body with four arms, each of them equipped with a spinning propeller. To describe the quadrotor model, a reference frame (Body Frame F_B) is attached to its center of mass (CoM).

In the standard quadrotor the four arms are usually disposed with an angular displacement of 90° between them, so the Body frame will have the x and y axes pointing towards two propellers and the z axis pointing up. To describe the full pose (position and orientation) of the body with respect to the inertial world frame (F_W) a vector $p \in \mathbb{R}^3$ and a rotation matrix $R \in \mathbb{SO}(3)$ are used. Defining v and ω the linear and angular velocities of F_B with respect to F_W, with ω expressed in F_B we have the kinematic model :

$$\dot{p} = v \tag{3.1}$$

$$\dot{R} = R[\omega_B]_x \tag{3.2}$$

For the dynamic model, the starting point is to consider the forces (f) and torques (τ) generated by each propeller $(i = 1, 2, 3, 4)$. Defining ω_i the controllable spinning rate of the propeller i and $u_i = \omega_i|\omega_i|$ as the control input, for the thrust force we have:

$$f_i = c_{f_i} u_i \hat{z}_i \tag{3.3}$$

where c_{f_i} is a propeller coefficient and $\hat{z}_i$ is the propeller spinning axis, pointing upward with respect to the body frame. Notice that the generated force f will be positive if the propeller is spinning accordingly to its spinning direction (clockwise propeller spinning clockwise and counterclockwise propeller spinning counterclockwise) and negative otherwise.

The rotation of each propeller around its axis generates a drag torque (τ_i^d) equal to:

$$\tau_i^d = c_{\tau_i} u_i \hat{z}_i \tag{3.4}$$

where c_{τ_i} is a propeller coefficient. Another torque (force torque) is generated by the fact that the propellers are applying a force translated from the CoM of the quadrotor body. For each propeller the generated force torque is:

$$\tau_i^f = p_i \times f_i \tag{3.5}$$

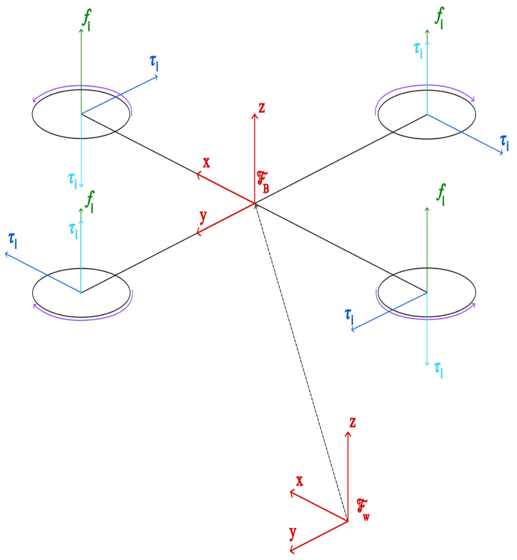


Figure 3.1: Forces and torques applied to the quadrotor

The overall forces and torques applied to the quadrotor's body can be seen in figure 3.1.
Defining the matrices F and M as follow:

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_f & c_f & c_f & c_f \end{bmatrix} \quad M = \begin{bmatrix} 0 & lc_f & 0 & -lc_f \\ -lc_f & 0 & lc_f & 0 \\ c_\tau & c_\tau & c_\tau & c_\tau \end{bmatrix} \tag{3.6}$$

The sum of forces and torques can be expressed in matrix form as:

$$\sum_1^4 f_i = Fu \quad \sum_1^4 \tau_i = Mu \tag{3.7}$$

where u is the input vector $u = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T$
With the above notation the quadrotor dynamics from Newton's Euler equation is:

$$m\ddot{p} = -mge_3 + RFu \tag{3.8}$$
$$J\dot{\omega} = -\omega \times J\omega + Mu \tag{3.9}$$

where $J \in \mathbb{R}^{3\times3}$ is the quadrotor inertia matrix.

3.2 Simplified model

In the rotational dynamics formula (3.9) the gyroscopic effect ($-\omega \times J\omega$), assuming a diagonal inertia matrix J is equal to:

$$-\omega \times J\omega = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix} \begin{bmatrix} J_x\omega_x \\ J_y\omega_y \\ J_z\omega_z \end{bmatrix} \tag{3.10}$$

$$= \begin{bmatrix} (J_y - J_z)\omega_z\omega_y \\ (J_z - J_x)\omega_x\omega_z \\ (J_x - J_y)\omega_x\omega_y \end{bmatrix} \tag{3.11}$$

This effect can be neglected in the case of similar values in the inertia matrix ($J_x \approx J_y \approx J_z$). With this approximation formula 3.9 simplifies in:

$$J\dot{\omega} = Mu \tag{3.12}$$

The relationship between the angular velocity ω and the derivatives of the orientation angles ϕ,θ and ψ can be retrieved starting from 3.2.

In fact $\dot{R}$ it is also equal to:

$$\dot{R} = \frac{\partial R}{\partial \phi}\dot{\phi} + \frac{\partial R}{\partial \theta}\dot{\theta} + \frac{\partial R}{\partial \psi}\dot{\psi} \tag{3.13}$$

Inserting 3.13 in 3.2 it can be derived the relation:

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\theta) & cos(\theta)sin(\phi) \\ 0 & -sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.14}$$

That can be simplified assuming the quadrotor is almost in hovering condition ($\theta \simeq 0$, $\phi \simeq 0$) , resulting in:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \simeq \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.15}$$

With the above simplifications, using 3.1 , 3.8, 3.15 and 3.12 the resulting model is:

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \ddot{p} \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} v \\ -g\, e_3 \\ \omega \\ 0_{3\times1} \end{bmatrix} + \begin{bmatrix} 0_{3\times1} \\ \frac{1}{m} R\, Fu \\ 0_{3\times1} \\ J^{-1}Mu \end{bmatrix} \tag{3.16}$$

From 3.6 and 3.7 the two new input variables for the control can be defined as the thrust force :

$$Fu = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{3.17}$$

and the torque:

$$Mu = \tau = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{3.18}$$

3.3 Control example

The structural properties of the standard quadrotor determines its under-actuation, allowing to control only four variables among the six degrees of freedom of the system. A possible choice is to take as control variables the position $\begin{bmatrix} x & y & z \end{bmatrix}^T$ and the yaw angle ψ. The evolution of ϕ and θ angles will the depend on the x and y states. For the quadrotor a decoupled control architecture can be applied. The controller can be divided in two main blocks: the position controller and the attitude controller. The former takes as input the desired position and produces as output an

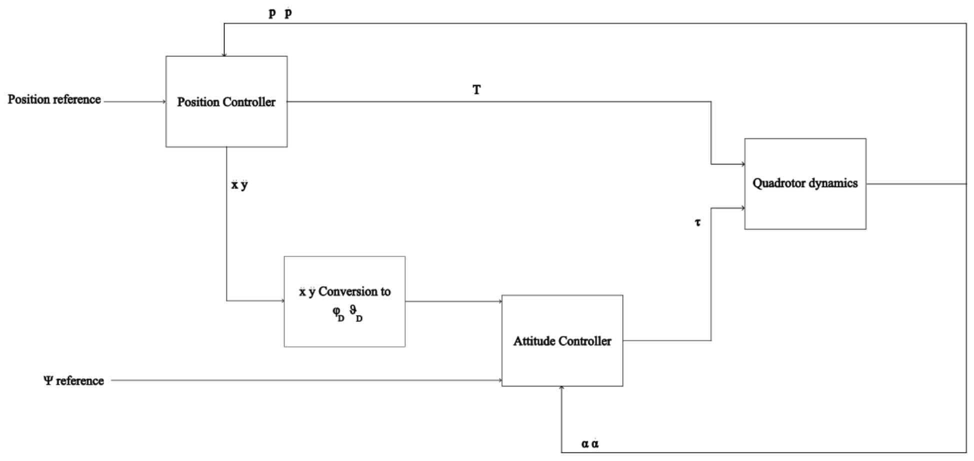


Figure 3.2: Quadrotor decoupled control scheme

acceleration in x and y and a trust force, the latter takes as input the desired angles of roll (ϕ) pitch (θ) and yaw (ψ) producing a torque τ. The desired accelerations in the horizontal plane $\ddot{x}$ $\ddot{y}$ can be related with angular velocities ϕ θ starting from 3.8 and 3.17

$$\ddot{p} = \begin{bmatrix} (sin(\psi)sin(\phi) + cos(\psi)sin(\theta)cos(\phi))\frac{T}{m} \\ (-cos(\psi)sin(\phi) + sin(\psi)sin(\theta)cos(\phi))\frac{T}{m} \\ -g + (cos(\theta)cos(\phi))\frac{T}{m} \end{bmatrix} \tag{3.19}$$

It can be performed a linearization around the desired trajectory, considering the hovering condition. The three angles can be approximated as:

- $sin(\phi_{trgt}) \approx \phi_{trgt} \quad , \quad cos(\phi_{trgt}) \approx 1$
- $sin(\theta_{trgt}) \approx \theta_{trgt} \quad , \quad cos(\theta_{trgt}) \approx 1$
- $\psi = \psi_{trgt}$

With these approximations the equation 3.19 becomes:

$$\ddot{p} = \begin{bmatrix} (sin(\psi_{trgt})\phi_{trgt} + \theta_{trgt}cos(\psi_{trgt}))\frac{T}{m} \\ (-cos(\psi_{trgt})\phi_{trgt} + sin(\psi_{trgt})\theta_{trgt})\frac{T}{m} \\ -g + \frac{T}{m} \end{bmatrix} \tag{3.20}$$

looking at the third row of equation 3.20 and recalling the condition for hovering: $\ddot{z} = 0$ it results:

$$\frac{T}{m} = g \Rightarrow T = gm \tag{3.21}$$

Using equation 3.21 in 3.20, looking at the first two rows, we have:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} (sin(\psi_{trgt})\phi_{trgt} + cos(\psi_{trgt})\theta_{trgt})g \\ (-cos(\psi_{trgt})\phi_{trgt} + sin(\psi_{trgt})\theta_{trgt})g \end{bmatrix} \tag{3.22}$$

$$= g \begin{bmatrix} sin(\psi_{trgt}) & cos(\psi_{trgt}) \\ -cos(\psi_{trgt}) & sin(\psi_{trgt}) \end{bmatrix} \begin{bmatrix} \phi_{trgt} \\ \theta_{trgt} \end{bmatrix} \tag{3.23}$$

Inverting the relation 3.23 the desired accelerations in the horizontal plane produced as output by the position controller can be converted in reference roll and pitch angles to feed as reference to the attitude controller, as it can be seen in figure 3.2.

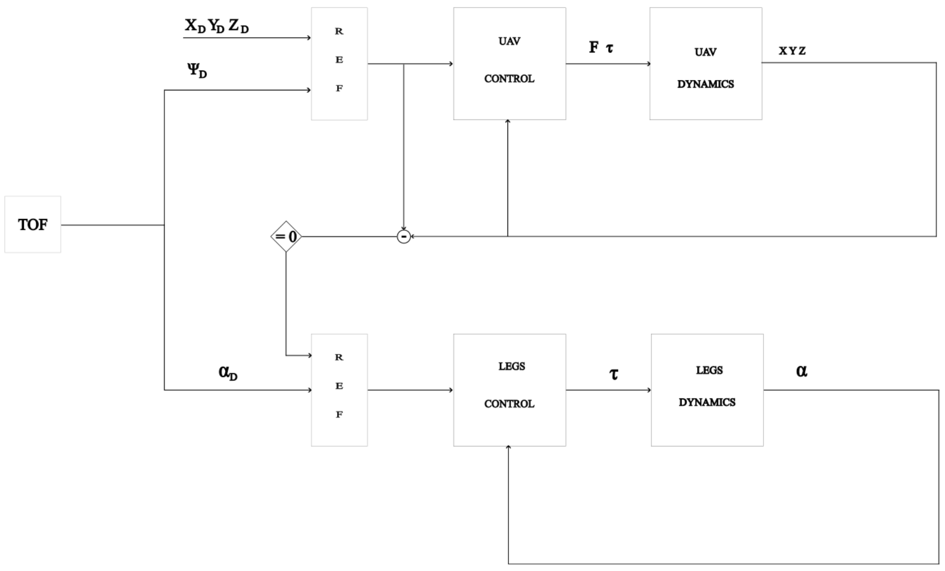


Figure 3.3: Control architecture

The desired accelerations are produced as output from the position controller that can be implemented as a PID controller with the addition of the feedforward action as follow:

$$\ddot{p}_{trgt} = \ddot{p}_{ref} + K_p e + K_d \dot{e} + K_i \int e \tag{3.24}$$

The attitude controller can be implemented as a PD controller while the desired trust can be modelled as:

$$T = \frac{m}{cos(\phi)cos(\theta)} \left[g + \ddot{p}_{trgt} + K_p e_z + K_d \dot{e}_z \right] \tag{3.25}$$

For the control of the drone during the flight it has been used the autopilot controller described in previous section, commanded by ROS2 [10] nodes running on a Raspberry Pi 4 board mounted on the drone body. In figure 3.3 it is possible to see the control scheme used, in which the position setpoints are given as input to the autopilot from a predefined sequence of trajectory setpoints. Once the last trajectory setpoint is reached, the drone starts the detection of the plane. The retrieved information is: the yaw direction of the slope, the slope value, and the drone altitude. The first manoeuvre is the alignment of the drone yaw angle in order to position the left side of the drone facing the landing surface. The following phase is a sequence of detections of the landing surface, a correction of the drone yaw angle and lowering of the altitude. This phase was made necessary by the errors of the real sensor readings that are 5% of the value for readings greater than $20cm$.

The lowering of the altitude is performed taking as reference the altitude of the leg 2 calculated from the estimated altitude of the drone and the estimated slope value. Once reached the altitude of $30cm$ of the leg 2, it starts the movement of the legs, through a PID controller described in section 3.5.2. Once the legs are correctly positioned the quadrotor starts the autonomous landing procedure.

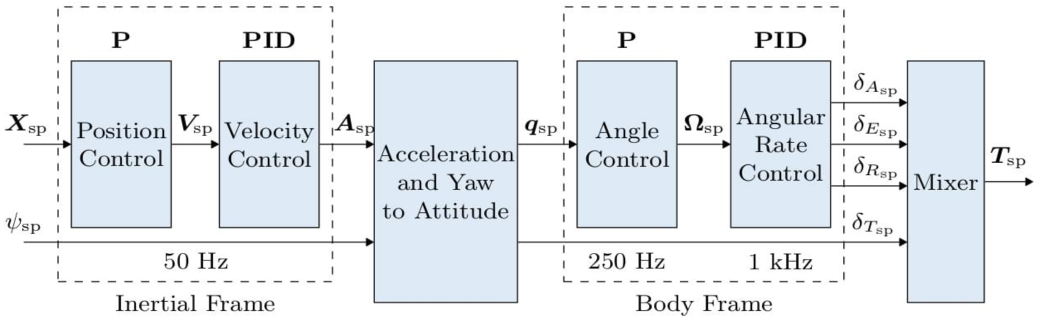


Figure 3.4: Px4 Autopitol control scheme

3.4 Px4 autopilot scheme

The autopilot uses a cascaded control architecture (figure 3.4).

The two main blocks of the architecture are both composed by the cascades of two controllers. The former block takes as input the desired setpoint specified in x y z coordinate and the desired yaw angle ψ_d and produces a desired acceleration. The latter takes the desired attitude for the multicopter and produces angular rates and thrust forces that are then converted in different inputs for the rotors. Both blocks use a cascade of a P and a PID control, with an anti wind up. Between these two blocks it is applied a conversion in order to transform the desired accelerations, specified in the inertial world frame, in desired attitude in the body fixed frame. The outer controllers can be bypassed depending on the offboard control mode. It fact it is possible to specify if the reference is given in position, velocity, acceleration, attitude, body rates, thrust and torques or directly rotors input. Once specified the reference input, the outer control loops are disabled. As example if the reference is a velocity command, the position controller is bypassed.

3.5 Leg control design

3.5.1 Leg inertia calculation

The first step to define the model dynamics is to calculate the inertia of the legs. The inertia of the single leg is calculated as sum of two different components, the inertia relative to the first section of the leg, connecting the joint and the skid, and the inertia of the skid. The mass of the component is denoted with the letter m, r indicates the ray of the skid, l the length of the component. The inertia of the first part of the leg is the well known inertia of a rod with axis of rotation passing through one of the end: $I = \frac{1}{3}ml^2$. The inertia of the skid can be calculated as the inertia of a cylinder with axis of rotation translated of a distance l from the axis of the cylinder, where the distance $l = 290mm$ is the length of the leg. The resulting formula for the inertia calculation is expressed in 3.26. The parameters used in formula 3.26 and the resulting inertia are reported in 3.27.

$$I = (\frac{1}{3}ml^2 + \frac{1}{2}mr^2 + ml^2) \tag{3.26}$$

$$m = 0.05kg \quad l = 0.290m \quad d = 0.195m \quad r = 0.007m \quad I = 0.0056kgm^2 \tag{3.27}$$

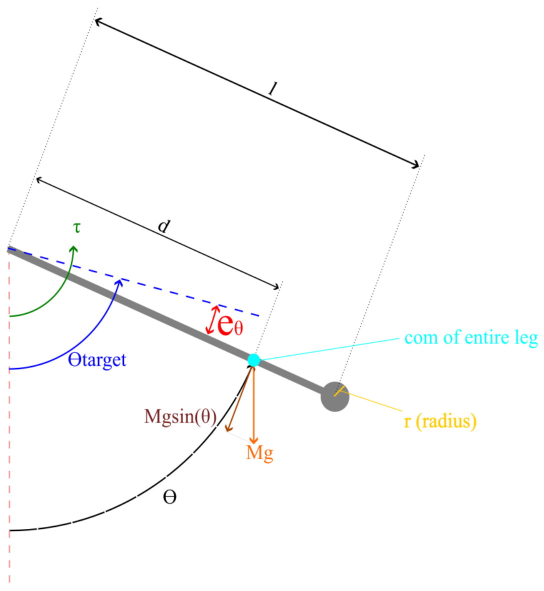


Figure 3.5: Scheme of the leg system

3.5.2 Leg dynamics and control

The target of the control is obviously the angle θ of the leg, defined as in the previous chapters, meaning angle 0 correspond to the leg in a vertical position in the negative direction of the z axis, angle are defined positive in the clockwise direction, and negative in the counterclockwise direction. With the inertia calculated in 3.26, it is possible to define the dynamics of the leg. In formula 3.28 it is denoted with the letter M the total mass of the leg, while with the letter d is indicated the distance between the joint rotation axis and the leg center of mass (CoM).

$$I\ddot{\theta} = -Mdgsin(\theta) - \dot{\theta}\mu + \tau \tag{3.28}$$

The scheme of the model is represented in figure 3.5, in which it is possible to see the forementioned quantities. From the system dynamics (equation 3.28) the input torque τ can be defined as the sum of two different components. The first input component is needed to compensate the effect of the gravity force $-Mdgsin(\theta)$, while the second component is formed by a PID action and it is used to drive the error to 0. The error is defined as the difference between the target position θ_{trgt} and the position of the leg θ

$$e_\theta = \theta_{trgt} - \theta \tag{3.29}$$

The resulting torque τ applied to the joint is then:

$$\tau = Mdgsin(\theta) + \tau' \tag{3.30}$$

where $\tau' is$:

$$\tau' = K_p e_\theta + K_d \dot{e}_\theta + K_i \int e_\theta \tag{3.31}$$

inserting 3.31 in 3.28 the resulting dynamics is:

$$I\ddot{\theta} = - Mdgsin(\theta) - \dot{\theta}\mu + Mdgsin(\theta) + \tau' \tag{3.32}$$
$$I\ddot{\theta} = - \dot{\theta}\mu + \tau' \tag{3.33}$$

From 3.33 it is possible to define the State space system representing the leg dynamics as :

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{\mu}{I} \end{bmatrix} x + \begin{bmatrix} 0 \\ -\frac{1}{I} \end{bmatrix} \tau' \tag{3.34}$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x \tag{3.35}$$

Where the state x is composed with the leg angle θ and its first derivative $x = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^T$. With the usage of an integral action for the control, the model has to be extended, with the addition of the integral state $x_i = \int_0^t \left[y(t) - r(t) \right]$ the new state vector becomes:

$$x = \begin{bmatrix} \int_0^t \left[\theta - \theta_{trgt} \right] \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{3.36}$$

The previously defined state space system 3.34 can be extended as:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{\mu}{I} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{I} \end{bmatrix} \tau' - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} r \tag{3.37}$$
$$y = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} x \tag{3.38}$$

With the new model 3.37, with the feedback gain matrix K defined as : $\begin{bmatrix} k_i & k_p & k_d \end{bmatrix}$ the input τ' becomes :$\tau' = -Kx$ The A matrix of the closed loop system is then:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{k_i}{I} & -\frac{k_p}{I} & -\frac{\mu - k_d}{I} \end{bmatrix} \tag{3.39}$$

The matrix eigenvalues can be placed at arbitrary position by selecting the gains k_i, k_p and k_d that correspond to the gains of the PID controller. The poles have to be placed in positions such that the closed loop system has the desired performances.

Performances in the transient are specified in the time domain as a desired settling time t_s and a maximum desired overshoot M_p^*. The steady state requirement is the following of the step reference with zero error. Notice that the presence of the integral action in the controller ensures the meeting of the steady state requirement. Approximating the behaviour of the closed loop system with the one of a second order system (dominant pole approximation), it is possible to approximatively relate time and frequency domain specification.

Starting from the second order system transfer function:

$$W(s) = \frac{1}{\frac{s^2}{\omega_n} + \frac{2\xi}{w_n} s + 1} \tag{3.40}$$

The closed loop bandwidth can be approximated with the frequency of the dominant poles. The time domain specifications can be related to the damping factor and natural frequency on the poles with the known formulas:

$$M_p = e^{\frac{-\pi\xi}{\sqrt{1-\xi^2}}} \Rightarrow \xi = \frac{\log\left(\frac{1}{M_p}\right)}{\sqrt{\pi^2 + \log\left(\frac{1}{M_p}\right)^2}} \tag{3.41}$$

$$t_{s5\%} = \frac{3}{\xi\omega_n} \Rightarrow \omega_n = \frac{3}{\xi_{ts5\%}} \tag{3.42}$$

$$\tag{3.43}$$

From 3.42 and 3.43 it is possible to retrieve admissible regions for the pole allocation in the complex plane: the admissible regions are represented in figure 3.6.

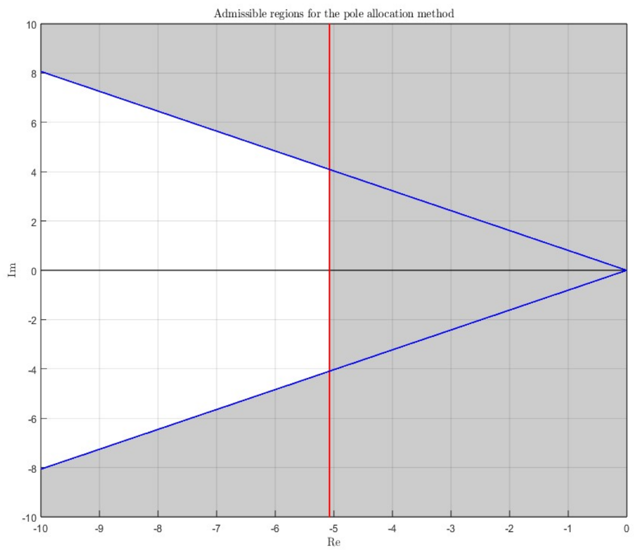


Figure 3.6: Admissible regions for the pole allocation method based on the performance requirements

Three different gains tested are:

	K_p	K_i	K_d
gains 1	4.4066	13.534	0.0287
gains 3	11.1499	30.2057	0.0599
gains 5	6	4	0.0287

Table 3.1: Gain values

The step responses of the different gain values can be seen in figure 3.7

The resulting poles position can be seen in figure 3.8. It is evident that the two gains 1 and 5 are positioning the poles outside the prescriber region. A confirm of the worse performance can be seen in the step response graph in figure 3.7. In particular gains 1 causes a larger overshoot, gains 5 cause a slow convergence to the target value. Gains 3 places the poles inside the desired area with an evident increase in the performance reducing overshoot and settling time.

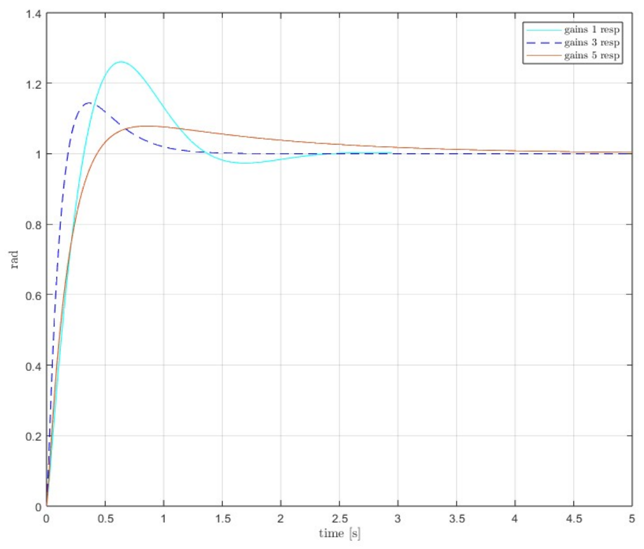


Figure 3.7: Step responses corresponding to 3 different gain values

In figure 3.9 it can be seen the leg step response measured in the gazebo simulation environment, the red dotted line highlights the value of 1.05 values that defines the settling time $t_{s5\%}$, the overshoot is also highlighted by a black dotted line. In the simulation were inserted saturations in the PID output to consider the limitations of the servomotor that will be used in a real application, and a saturation in the integral component of the controller to prevent large overshoot effects due to saturation. In figure 3.10 it is shown the measured torque at the joint, which respects the physical limitations of the real actuator.

3.6 Simulink schemes

In figure 3.11 it is shown the overall control scheme with the feedback action used to calculate the error, which is fed into the PID controller. the feedback action it is also used to calculate the component of the input torque used to compensate the gravity action as shown in 3.32.

The system representing the leg dynamics is implemented respecting the equation 3.28.

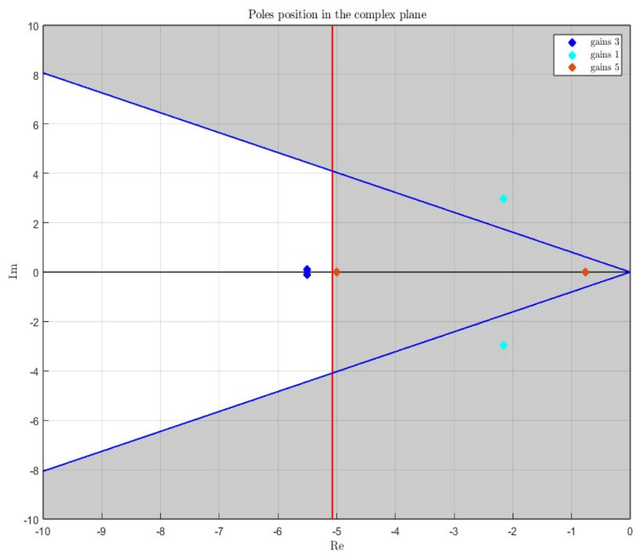


Figure 3.8: Poles location in the complex plane for different gain values

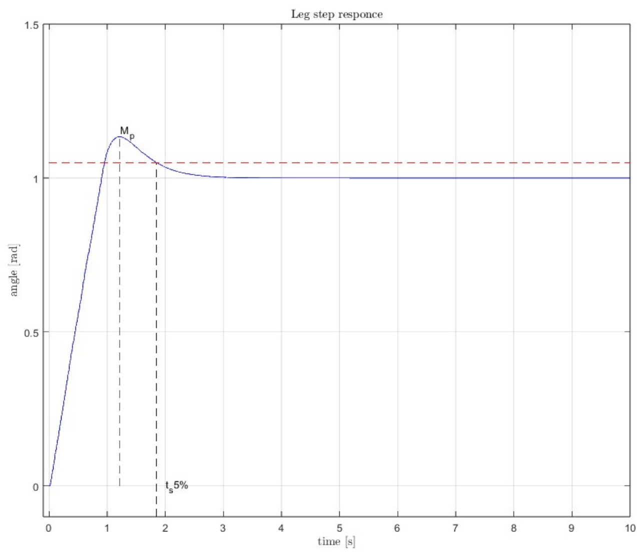


Figure 3.9: Leg step response

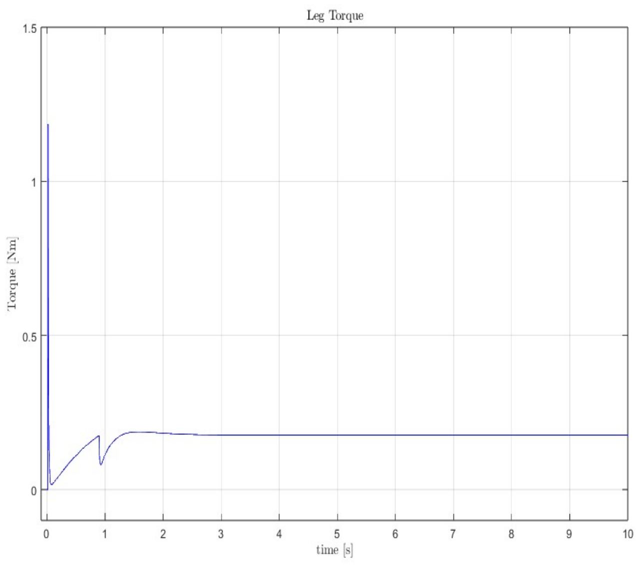


Figure 3.10: Leg step response measured torque

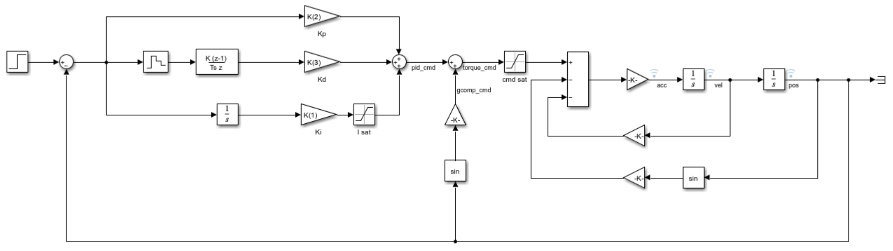


Figure 3.11: Simulink control scheme

Chapter 4

Software and Simulation setup

To control the quadrotor during the flight it has been used the PX4 Autopilot. This autopilot has been chosen because it is designed to be compatible with the Pixhawk controller board purchased with the drone. It's main characteristics are described in 4.1.

For the simulation environment it has been chosen Gazebo[4] Garden which is the only available version from Ubuntu Linux 22.04. ROS2 [10] framework has been used to manage the upper level control tasks due to the various advantages of this framework described in 4.2 and its compatibility with Gazebo [4] .

4.1 PX4-Autopilot

PX4[7] is an open source flight control software known for its great adaptability, and modularity. These characteristics make it able to control many different types of vehicles including aircraft, ground vehicles and underwater vehicles. It is used in a wide range of cases, from consumer to industrial applications. It's also compatible with different flight controllers, especially with the Pixhawk Series, for which it was initially designed. The flight controller can be integrated with companion computer for higher level command and control. Another great feature is the possibility to define flight modes, which provide different levels of automation.

4.2 ROS2

ROS[10] is an open-source set of tools and libraries that provides the building blocks to develop robot applications. It is based on the concept of nodes, which are independent programs. The nodes can communicate with each other exchanging information through 3 different ways depending on their needs. The 3 types of communications are:

- Messages: This is the simplest way to communicate between nodes, it is anonymous and asynchronous. The node that sends data is called publisher, and it publish information on a specific topic which acts as middleware between publisher and subscriber, where information are stored. The node that receives information is called subscriber. When a message is published, every other node that needs that information can subscribe to the specific topic and read the data.

- Services: Services are used when a node needs to exchange information with another specific node. They are used to perform tasks or computations that require a response. The service client asks to the service server to perform the specific task, waits for the result and then proceeds with the program.
- Actions: Actions are similar to services, but are specifically designed for processes that take a longer time to be executed. This type of communication provides, in addition to a request and a response, a feedback to inspect the progress of the process and the possibility to cancel the process.

This approach of dividing an application in subprograms (nodes), the efficient ways to communicate, the modularity, the reusability and adaptability make ROS2 [10] a perfect choice for every robotic application.

ROS [9] was developed in 2007, it has been widely tested and over the years different new versions have been released, bringing constant improvements. The ROS1 [9] development was primarily focused on a single robot, with no real time requirements. It was based on a great network connectivity with no critical aspects. Furthermore it's development was intended mostly for academic use. Year after year ROS1 [9] usage has been consistently growing, with application also in commercial products in different fields of application. The ROS2 [10] project has the goal to leverage the great characteristics of ROS1 [9] improving its lacking aspects. The wide use of ROS [9]framework in many different situations brought the necessity to improve some aspects. In particular regarding:

- Standard approach for teams of multiple robots, overcoming the single-master structure of ROS1 [9].
- Optimization for embedded platforms
- Support for real time systems
- Account for non ideal connectivity, taking into account delays, data loss and other network problems
- Improvements of ROS [9] as framework for market and real world products instead of research limited applications

Over the years several new technologies and protocols have been developed. In addition, many ready-to-use open source libraries have been created. These tools can be applied in the ROS [9] framework in order to improve it.

4.3 uXRCE-DDS middleware

A middleware is necessary to allow communication between ROS2 [10] and PX4 [7], it allows ROS2 [10] to interact with uORB messages as ROS2 [10] topics (figure 4.1).

In this way the companion computer running ROS2 nodes can easily access information related to the vehicle. In fact uORB [11] messages are internally used by the flight controller for inter-thread/inter-process communication. The uXRCE-DDS middleware consists of:

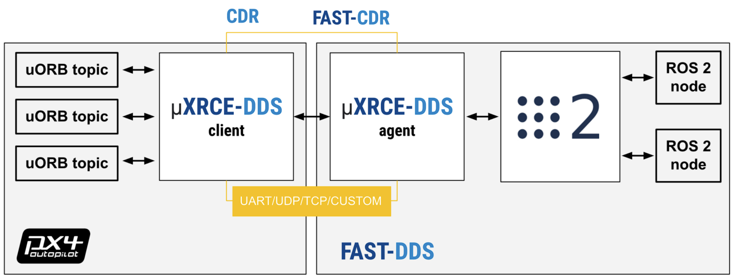


Figure 4.1: uxrce-dds middelware

- Client publishes to/from a predefined set of uORB [11] topics to the global DDS data space.
- Agent runs on the companion computer and acts as a proxy for the client in the DDS/ROS 2 network.

The client is generated at build time and included in PX4 [7] firmware by default, the agent is independent from the client code and it can be built or installed alone. ROS2 [10] needs to have the same message definitions in the built workspace in order to interact with the middleware. uORB [11] messages are a set of predefined messages useful for exchanging information about the controlled system. For example, the message 'VehicleLocalPosition' contains information about the vehicle's position, velocity and acceleration in the local reference frame.

In this project this middleware was used to read from the px4 autopilot information on the vehicle state and to send command such as trajectory setpoints, set up and maintain the offboard control mode as will be explained in following chapters.

4.4 Mavlink Messaging

Mavlink [6] is a lightweight message protocol for drones. It uses a 14 bytes overhead to provide fast and secure communication even with small bandwidth, noise or latency. It provides methods for packet drop detection, and corruption. It can be used both for onboard and offboard communications. It offers both multicast and unicast communication modes. The former is based on a publish/subscribe pattern, it is optimised for high frequency data streams in which there is not a specific receiver but many devices can subscribe to the specific topic and access the information. This approach allows to remove the target systems and id saving link bandwidth. The latter is designed for communications that requires a secure delivery such as configuring an onboard mission. In this project it has been used to communicate with the ground station (QGroundControl [8] see section 4.6).

4.5 Gazebo Garden

Gazebo [4] is an accurate physics simulator, and one of its features is offering multiple physics engines to better simulate different simulation contexts. The physics engine that has been utilized for the simulation is the default one for PX4 [7]which is the ODE engine. The simulator also offers the possibility to customize the simulation environment, to better represent specific situations. For this purpose Gazebo [4] offers many different types of ready-to-use models and worlds for simulations on a cloud-hosted sever. Another feature is the possibility to simulate many different types of sensor like IMUs, lidar or cameras with the possibility to reproduce sensor noise. To integrate with ROS2 [10], a ROS2 package 'ros_gz_bridge' is necessary for handling message conversion between the two environments.

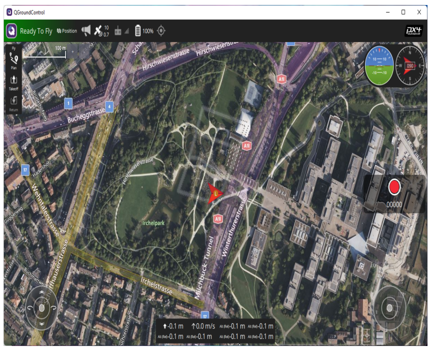
4.6 QGroundControl

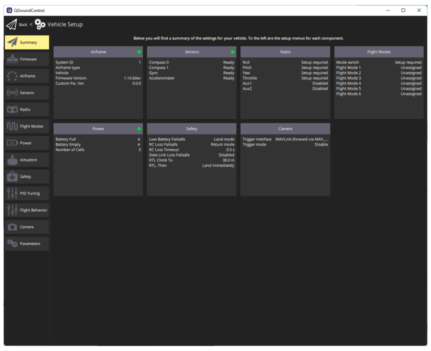
QGroundControl [8] is a ground control station designed to work with vehicles microcontrollers using the MAVLink [6] protocol. In figure 4.2a it can be seen the main window of the application with a top view of a predefined map with the position of the vehicle. It offers the possibility to completely set up the vehicle with different options:

- Firmware : Install PX4 [7] or ArduPilot firmware onto the microcontroller
- Airframe : Specify the airframe type for the vehicle choosing among different groups/types such as 'Generic Quadcopter' or 'Generic Hexarotor' and then selecting a specific configuration within the group that best matches the real vehicle.
- Radio : Complete configuration and calibration of the main transmitter.
- Sensors : Configure and calibrate the vehicle's compass, gyroscope, accelerometer and any other sensors.
- Flight Modes : Mapping flight modes to radio channels, and the switches on the radio control transmitter.
- Power : Configure battery parameters and advanced settings for propellers pwm control.
- Actuators : Configure propeller number, position and spinning direction. Configure and test general actuators.
- Safety : Configure failsafe actions such as landing in the case of low battery, or return to start position in the case of loss of radio-control signal.
- PID Tuning : Tuning the flight controllers.
- Camera : Configure physical outputs to trigger a camera.
- Parameters : Modify all parameters associated with the vehicle. Parameters are divided in groups that contain related specific parameters, group examples are : EKF2 and PWM Outputs.

In figure 4.2 it can be seen the QGroundControl [8] window that allows the configuration of the parameters cited before.

On top of that it offers instruments for a complete analysis of the recorded data, mission planning for autonomous flight and map visualization.


(a) QGroundControl application window



Below you will find a summary of the settings for your vehicle. To the left are the setup menus for each component.

Airframe

```
System ID                      1
Airframe type
Vehicle
Firmware Version       1.14.0dev
Custom Fw. Ver.            0.0.0
```

Sensors

```
Compass 0                  Ready
Compass 1                  Ready
Gyro                       Ready
Accelerometer              Ready
```

Radio

```
Roll              Setup required
Pitch             Setup required
Yaw               Setup required
Throttle          Setup required
Aux1                    Disabled
Aux2                    Disabled
```

Flight Modes

```
Mode switch       Setup required
Flight Mode 1         Unassigned
Flight Mode 2         Unassigned
Flight Mode 3         Unassigned
Flight Mode 4         Unassigned
Flight Mode 5         Unassigned
Flight Mode 6         Unassigned
```

Power

```
Battery Full                   4
Battery Empty                  4
Number of Cells                3
```

Safety

```
Low Battery Failsafe        Land mode
RC Loss Failsafe          Return mode
RC Loss Timeout                 0.5 s
Data Link Loss Failsafe      Disabled
RTL Climb To                   30.0 m
RTL, Then            Land immediately
```

Camera

```
Trigger interface   MAVLink (forward via MAV_...
Trigger mode                          Disable
```

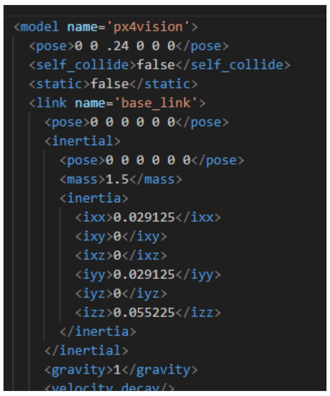
(b) Vehicle setup in QGroundcontrol

Figure 4.2: QGroundControl windows

Having completed the initial phase addressing the mechanism design, the next stage is numerical simulation.

There are several positive aspects related to the simulation phase. The first one is that it is money and time saving. A simulation can be easily repeated several times, always starting from the desired initial conditions, with no dependence on external factors. If the simulation environment is compromised or changed during the simulation, it doesn't require to be manually restored to the initial conditions. During the simulation time can be controlled, allowing adjustment on the speed at which it runs: faster, slower or even stopped. In general it is also possible to proceed step by step allowing an accurate inspection of the system dynamics.
In addition to that, there is no risk of damaging expensive equipment or hurting human beings, leading to a safety improvement. Furthermore it is possible to have complete control over the environment conditions, such as wind in an outdoor simulation, illumination and sensor noise. High-performance physics engines allows a precise representation of the real world, ensuring safety and satisfactory performance for the future real implemented system.

As previously said, it has been used the Gazebo [4] simulation environment, described in 4.5 to verify the effectiveness of the proposed solution. The SDFormat is a standard format used to describe objects in the Gazebo [4] environment. The mentioned format it has been used to define the quadrotor model used in the simulation.



```xml
<model name='px4vision'>
  <pose>0 0 .24 0 0 0</pose>
  <self_collide>false</self_collide>
  <static>false</static>
  <link name='base_link'>
    <pose>0 0 0 0 0 0</pose>
    <inertial>
      <pose>0 0 0 0 0 0</pose>
      <mass>1.5</mass>
      <inertia>
        <ixx>0.029125</ixx>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyy>0.029125</iyy>
        <iyz>0</iyz>
        <izz>0.055225</izz>
      </inertia>
    </inertial>
    <gravity>1</gravity>
    <velocity_decay/>
```

Figure 4.3: SDFormat example

4.7 SDFormat

SDFormat is an xml format that describes objects and environments, used for robotic simulation. It has been in use for several years, becoming stable and robust. It allows the description of every aspect of robots, objects, ambient light condition, terrain and physics. Specifically, to describe a Robot, the main elements needed are:

- The model tag: It is the main tag that contains all other elements that describe the robot.
- The link tag: Inside the model tag, it is used to describe every link composing the robot. The principal components for its description are the link pose, inertial properties like mass and position of its center of mass (CoM), collision and visual properties.
- The joint tag: Inside the model tag, used to describe interactions between links. The main description elements for a joint are the type of joint, such as revolute or prismatic, the parent and child links, the joint axis and pose.
- The sensor tag: It can be placed inside a link or inside a joint tag. The main components to describe it are the sensor type, the topic on which it has to transmit, the update rate, the pose and different other parameters that are dependent on the specific sensor type.

An example of robot description using the SDFormat can be seen in figure 4.3

4.8 Gazebo model

In Gazebo the SDFormat (described in section 4.7) is used to describe the simulation environment (world model) and the drone model. As previously said, Gazebo offers lots of different ready to use models in its cloud server, but among them there was no model suitable for this study. The multirotor model was created from scratch, starting from taking precise measurements of the drone parts. The 3d model was created using Autodesk Fusion360 (figure 4.4) and then exported in SDFormat to be used in the simulation environment.

In the SDF file used in Gazebo, the drone model is composed of three main links. The first one is the base link, within which all metal plates composing the main body

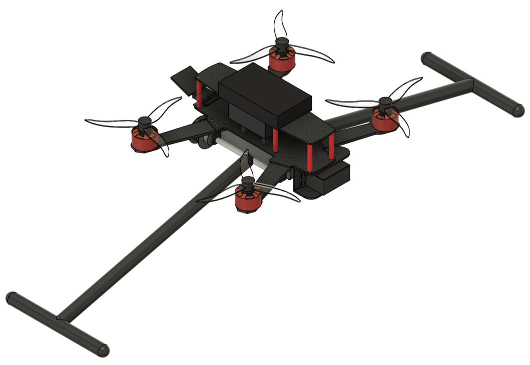
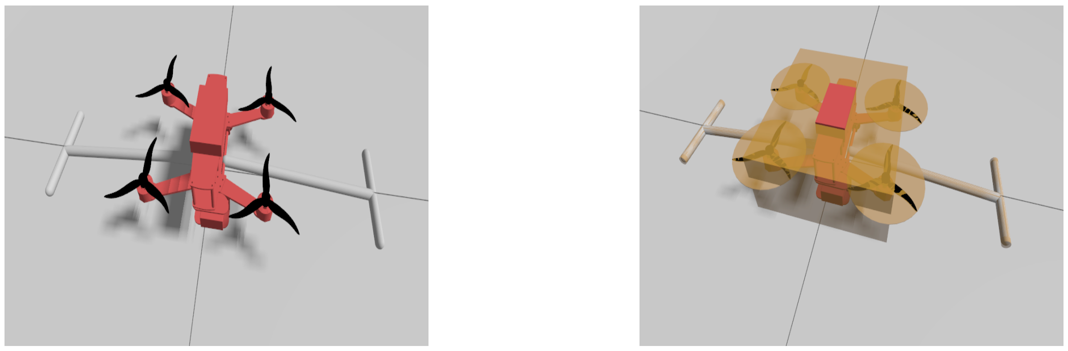


Figure 4.4: Fusion360 model


(a) Gazebo Model

(b) Collsion model shape

Figure 4.5

and every other part rigidly attached to it such as batteries, the four motors and the servomotor, were merged. The other two links are the two movable legs. In addition to that other four links are used to describe the four rotors. Figure 4.5a is showing the resulting model in the Gazebo world.

During the simulation the model collision shapes are used by the physics engine to simulate interactions between objects. For simplicity it has been used a simple cube shape for the main body, cylindrical shapes for the rotors and two cylindrical shapes for each leg (figure 4.5b). These simplifications are not compromising the results of the experiment. In fact the simplifications are increasing the real collision shapes of the model. For the rotors, a cylinder is representing the region of space in which the rotor will collide with objects while spinning. Similarly for the skids a cylinder shape is sufficient to model the contact with the ground.

The pre-built plugin for Gazebo, `JointPositionController` was modified as later explained to be used for legs control. The main parameter required for this plugin is clearly the name of the joint that has to be controlled. The plugin can be used sending commands through a specific topic, that can be customised, using position or velocity commands. The movement control is performed with a tunable PID controller; it also allows limits imposition for the produced command or PID components.

The feedback for the leg position was provided by a pre-built plugin, `JointStatePublisher` that publishes joint information such as position and velocity on a specific topic.

4.9 Time of Flight Sensor

The sensor that is used in this project is the VL53L5CX Time of Flight sensor produced by STMicroelectronics. It provides accurate ranging up to four meters and works with a frequency of 60 Hz. Using the sensor specifications it was modelled in Gazebo [4] as lidar, inserted inside the `base_link` link, meaning the main drone body, positioned in the front of it, pointing downwards. The sensor was configured to have 8 laser reading in the horizontal direction (with respect to the sensor frame, meaning along the z,y plane in the body frame) and eight vertical readings forming a grid of 64 measurements.
The errors in the laser readings were introduced artificially following the datasheet specifications. Tests conducted on the real sensor allowed to verify the better performance of the real sensor with respect to the specifications, thereby ensuring in this way a worst case analysis in the simulation environment.

4.10 Laserscan translation from Gazebo to Ros2

The translation was made through the ROS2 [10] package `ros_gz_bridge`, which offers different nodes to translate topics between Gazebo [4] and ROS2 [10]. The Translation can be made in both directions using the executable `parameter_bridge` for the desired topic.

The message definition in ROS2 [10] for the Laserscan is not fully compatible with the Gazebo [4] definition of the corresponding message. In fact The Gazebo [4] message definition allows the message to contain data from a sensor reading in 2 dimensions, while the message definition in ROS2 [10] allows only planar Laserscan. The quick fix was modify the source code of the `ros_gz_bridge` package.

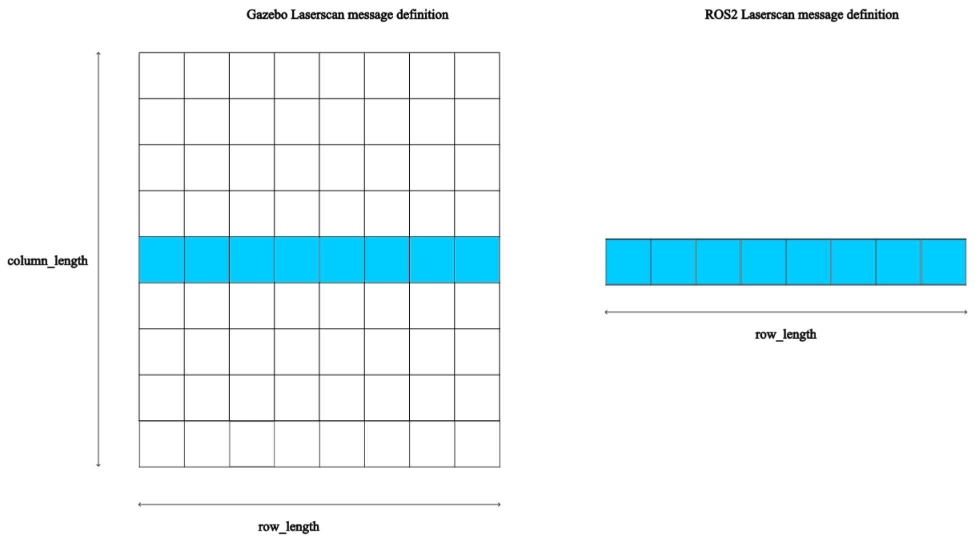
The original behaviour of the package was to select the middle one, among all the vertical Laserscan detections in the gazebo message, and copy that only one planar reading as ROS2 [10] Laserscan message. It was sufficient to modify the indexes inside the node coping function to ensure all data were translated to ROS2 [10] as a planar laserscan of length: `row_length` * `column_length` , where `row_length` is the number of points in a single planar laserscan detection, and `column_length` is the number of planar laserscan performed vertically.

A schematic representation of the original behaviour is shown in figure 4.6a while the modified version is showed in figure 4.6b.

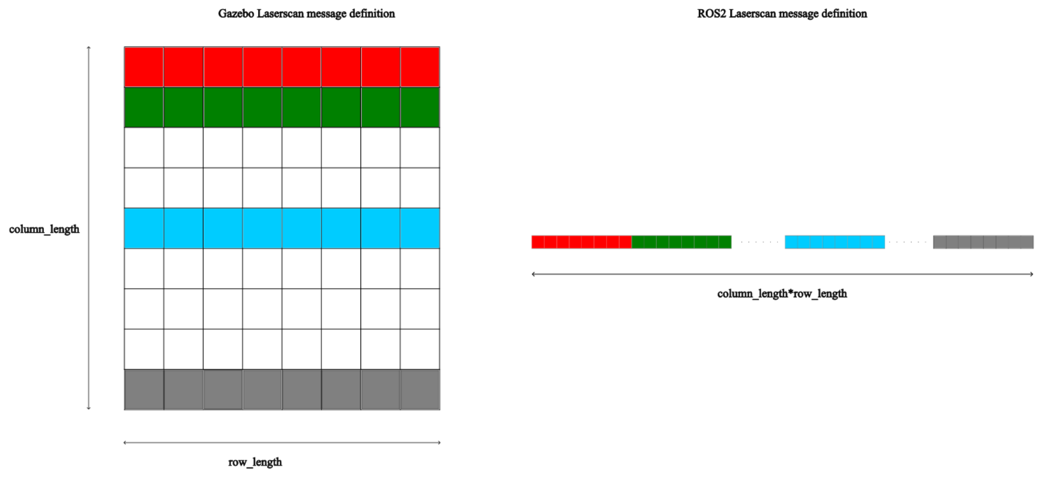
To reconstruct the 3d structure of the measurement in ROS2 [10] from the message definition is sufficient to know the number of readings in the two directions, in our case both equal to eight.

4.11 Plane detection

The detection of the slope value from the time of flight (ToF) sensor, was improved with respect to the previous project ([14]). It didn't require the exclusion of any inconsistent measurement due to the fact that the legs are not in the sensor field of view when it is performing the slope detection. The legs can be moved laterally up to 85 degrees positive or negative, depending on the leg, and are moved in position for the landing only after the sensor reading.


Gazebo Laserscan message definition
ROS2 Laserscan message definition
column_length
row_length
row_length

(a) ros_gz_bridge plugin original behaviour


Gazebo Laserscan message definition
ROS2 Laserscan message definition
column_length
row_length
column_length*row_length

(b) ros_gz_bridge plugin modified behaviour

Figure 4.6: ros_gz_bridge plugin original and modified behaviours

The time of flight (ToF) sensor collects a total of 64 measurement, returning information on the distance and angular displacement of the rays. This information is then converted in x,y and z coordinate in the multirotor frame. The vector of z values and matrix of x and y coordinate define a linear regression problem, in which for every point:

$$z = \alpha x + \beta y + \gamma \tag{4.1}$$

In order to have a more compact description of the problem the following elements can be defined:

$$w = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \quad X = \begin{bmatrix} x_1 & y_1 & 1 \\ & . & \\ & . & \\ x_i & y_i & 1 \\ & . & \\ & . & \\ x_{64} & y_{64} & 1 \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ . \\ . \\ z_i \\ . \\ . \\ z_{64} \end{bmatrix} \tag{4.2}$$

Where x_i, y_i and z_i are the i-th point coordinate. Resulting in the following relation

$$Xw = Z \tag{4.3}$$

if the matrix $\left(X^T X\right)$ is invertible the solution can be foud as:

$$w = \left(X^T X\right)^{-1} X^T Z \tag{4.4}$$

Otherwise it can be used the singular value decomposition of the matrix X:

$$X = USV^T \tag{4.5}$$

The solution of the problem is then:

$$w = VS^{-1}U^T Z \tag{4.6}$$

Once retrieved from the data the coefficients α β and γ looking at the standard plane equation:

$$ax + by + cz + d = 0 \tag{4.7}$$

$$\frac{a}{c}x + \frac{b}{c}y + \frac{d}{c} = -z \tag{4.8}$$

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \frac{a}{c} \\ \frac{b}{c} \\ \frac{d}{c} \end{bmatrix} = -z \tag{4.9}$$

Looking at the equations 4.9 and 4.3 it is clear that the plane orthogonal vector is:

$$n = \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{4.10}$$

A simple example is the detection of an horizontal plane, as it can be seen in figure 4.7. The quadrotor is placed at an altitude of $0.35m$, with the described procedure the retrieved w is :$w = \begin{bmatrix} 00 - 0.35 \end{bmatrix}$ Inserting the values of w in 4.9 it results:

$$0x + 0y – 0.35 = -z \tag{4.11}$$

Meaning the equation of a x-y plane passing through the point: $= \begin{bmatrix} 000.35 \end{bmatrix}^T$. The altitude of the drone is clearly the third component of the vector w and the coefficients of the vector n orthogonal to the plane can be found as showed in 4.10 resulting in:

$$n = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.12}$$

That is clearly a vector parallel to the z axis and orthogonal to the detected plane.

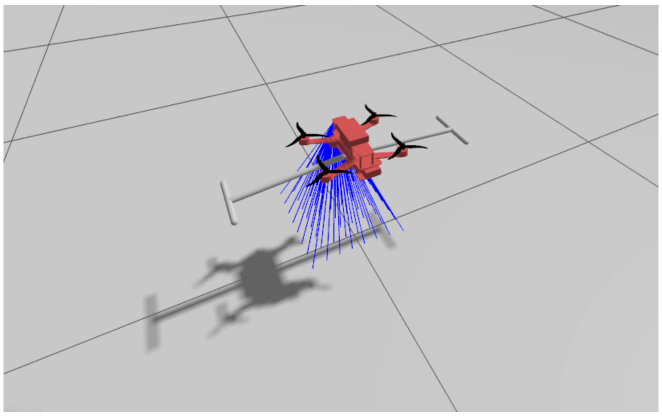


Figure 4.7: Detection example in Gazebo

The normal vector is represented with respect to the time of flight (ToF) sensor frame, attached to the drone body. In order to calculate the plane slope and direction in the world reference frame, the normal vector n is then converted through a change of coordinate. Then the vector yaw angle can be calculated, the operation can be thought in the xy plane, through the `atan2` operator. The target heading direction for the drone will be orthogonal to the plane normal (figure 4.8a). The slope value is the angle between the plane normal and the z axis (figure 4.8b).

4.12 Px4 Autopilot Offboard control

The autopilot can be controlled offboard imposing the correspondent flight mode. In this mode the vehicle can be controlled in position, velocity, acceleration, attitude rates or thrust and torque setpoints. The reference setpoints need to be provided from an external source like a companion computer using the Mavlink [6] protocol described in section 4.4. The autopilot requires a constant stream of data, either setpoints or `OffboardControlMode` messages, from the external controller as a ”proof of life”. If the frequency of the stream of data drops below $2Hz$ the autopilot will exit the offboard control mode performing a failsafe action. A simple representation of the described procedure is showed in figure 4.9.

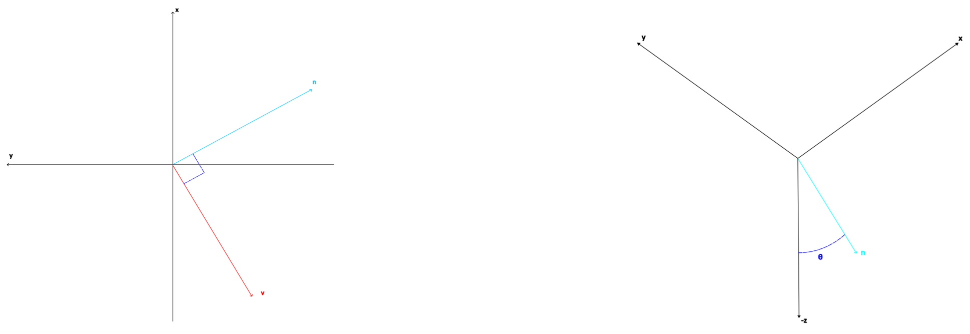

(a) Plane normal vector yaw angle (b) Plane slope value

Figure 4.8: Informations retrieved from the plane normal vector

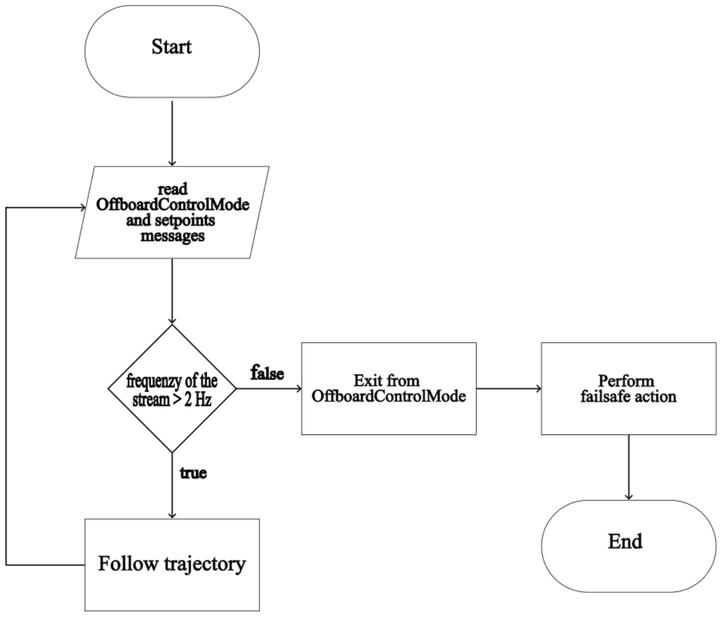

Start
read OffboardControlMode and setpoints messages
frequenzy of the stream > 2 Hz
false
true
Exit from OffboardControlMode
Perform failsafe action
Follow trajectory
End

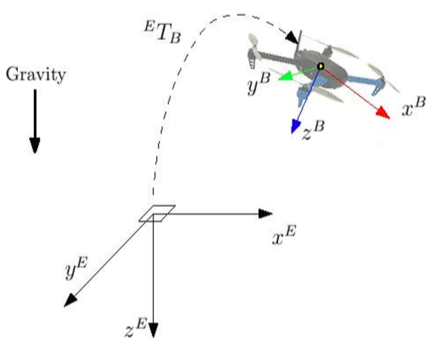
Figure 4.9: PX4 Offboard control mode control flow chart

In addition, before switching to this mode, in order to ensure the stability of the communication, the autopilot requires that the above-mentioned data stream is sent for at least a second

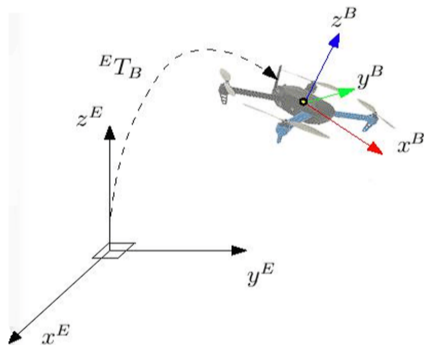
4.12.1 Ros2 and Px4 reference frames

The Px4 [7] Autopilot uses a FRD reference frame both for the inertial world frame and for the body frame. The frame name is specifying the directions of frame axis x y and z meaning F for forward, R for right and D for down. In the ROS2 [10] and Gazebo [4] environments different reference frames are used, both the body frame and the world frame are FLU frames, where the axis x y and z are pointing forward, left and up. A representation of the different reference frames is shown in figure 4.10. To compensate for this difference a static transformation in the data is needed in order for the offboard control to properly send position commands and receive feedback. The necessary transformation can be retrieved in terms of roll pitch and yaw angles. As shown in figure 4.11 it is sufficient a rotation of 180° about y axis and -90° about the z axis. The resulting transformation matrix is:

$$R = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.13}$$



(a) Frame in Px4 Autopilot



(b) Frame in Ros2

Figure 4.10: Difference in reference frames

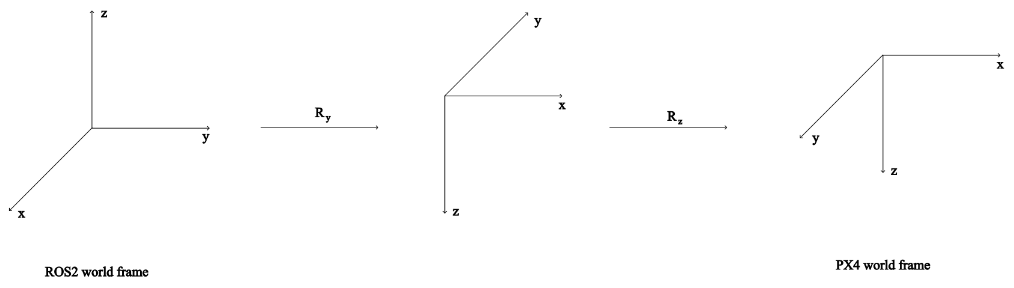


Figure 4.11: Frame transformation between Gazebo and ROS2 world frames

4.13 Ros2 nodes structure

In order to exploit the peculiar characteristics of the ROS2 framework, different nodes were designed to benefit from the property of modularity. The main nodes that have been developed are the `Offboard_control_node`, the `Leg_control_node` and the `Plane_detection_node`.

The schematic representation of the nodes structure and the information flow is showed in figure 4.12.

A flow chart with the different phases of the simulation can be seen in figure 4.13.

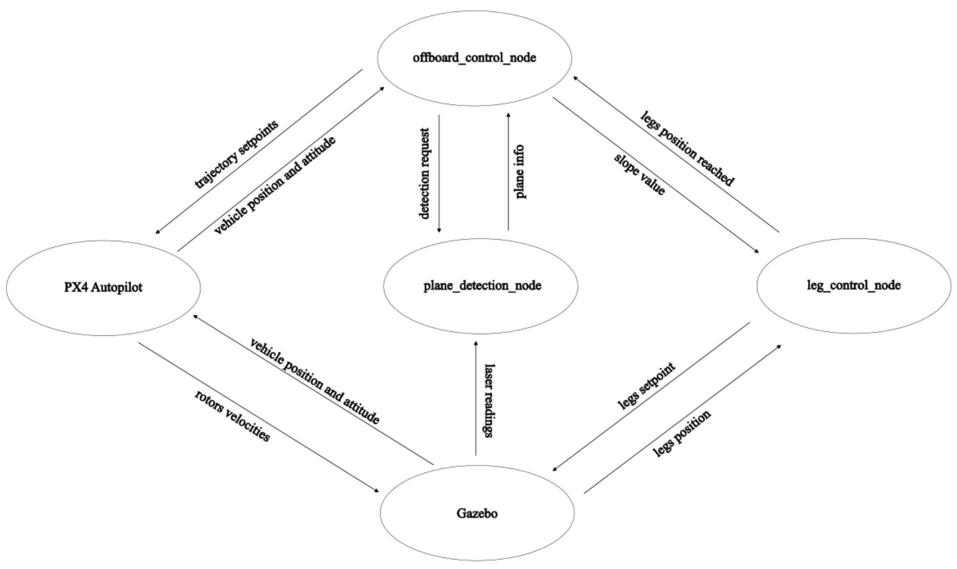


Figure 4.12: Schematic ROS2 nodes representation

4.13.1 Offboard _control Node

The `Offboard_control_node` is designed to be the main node for the platform control. It's first function is to publish the commands specifying the offboard control mode for the px4 autopilot at 10 Hz , a frequency that is sufficient as a "proof of life" as already explained. The second function is to publish the trajectory setpoints with the same frequency. Trajectory Setpoint are specified before starting the simulation, they are specified with respect to the ROS2 [10] fames and then converted to the px4 reference frame (see 4.12.1). Setpoints are used to navigate over the landing plane. Once in position, the node command the detection of the slope angle and slope orientation at the service node `plane_detection_node` using a custom service. Once received a response, the node, after a transformation to the ROS2 [10] world reference frame, proceeds in aligning the vehicle yaw accordingly. Before landing, legs are moved in position through a request to the service node `leg_control_node`.

4.13.2 Pre-loaded data

As previously seen in chapter 2, is possible to compute the desired angular displacements of the legs offline, through the formula 2.4. In particular, with the introduction of a function $disp$ representing the leg_1 angular displacement, using the fact that θ_1 and θ_2 have a ratio of $\frac{7}{15}$ as shown in 2.12, the angles θ_1 and θ_2 can be expressed as a functions of the initial minimum values and the $disp$ variable:

$$\theta_1 = \theta_{1min} + disp \tag{4.14}$$

$$\theta_2 = \theta_{2min} + disp\frac{7}{15} \tag{4.15}$$

The equation 2.4 becomes:

$$\varphi = \arctan\left(\frac{l\cos(\theta_{1min}+disp)-l\cos(\theta_{2min}+disp\frac{7}{15})}{l\sin(\theta_{2min}+disp\frac{7}{15})-l\sin(\theta_{1min}+disp)+b}\right) \tag{4.16}$$

The $disp$ variable has a range of vales that start from 0 corresponding to θ_{1min} and a horizontal plane, to 90° corresponding to the maximum slope value for the plane, meaning 60°

Figure 4.14 illustrates the relation given by the formula with respect to θ_1. The computed values can be loaded into the onboard controller in order to save computational time.

4.13.3 Leg Control Node

The Ros2 node to control the legs movement is implemented as a service node. It is composed by two publishers, used to send commands to the plugins for the joints control, and a subscriber used to have a feedback on legs position.
For simplicity during the first phase of the simulation, instead of implementing the gear mechanism transmitting motion between legs, allowing a control with a single plugin, it has been preferred to individually control each leg, maintaining the relation between the two sent commands of $\frac{7}{15}$.

A custom service was designed for the communication with this node. The request has a field containing the slope value of the landing surface, the response consists in a boolean value specifying if the action is completed successfully. Once a request is received, the slope value is searched in a pre-loaded data structure in order to find the correspondent angular displacement required for the servomotor. Then the command with the motor position is sent with a certain frequency to the joint position controller plugin. Once the legs are in position the response in sent back with a true value.

4.13.4 Leg Control Plugin

The plugin used for the control of the legs is a plugin from the Gazebo [4] system package modified to act as specified in previous sections. The plugin is divided in two main pieces of code, the code relative to the configuration of the plugin and the code for the main function of the plugin. The former is composed by instructions that are executed only once, when the plugin is loaded. Information to set the parameters is retrieved from the file written in SDFormat (section 4.7). The main parameters set in this phase are the name of the joint that has to be controlled, the type of message received as input by the plugin and the PID gains with the possibility to specify saturations for the integral action and for the total PID output (figure 4.15). If the PID gains are not specified explicitly in the SDF file, the plugin uses the default values.

The main code executed by the plugin to accomplish its function is contained in the PreUpdate function. This function is a callback that is invoked at every iteration before the physics engine runs, applying the PID control signal. The main modification from the standard plugin was the addition of the feedback non linear term to compensate the action of the gravity force (`gCompGain`), with the possibility to set the gain value in the model SDF file. The resulting controller output formula used by the plugin is shown in figure 4.16. The gains `gCompGain` is previously calculated using the system parameters as explained in section 3.5.2.

4.13.5 Issues during the simulation setup

The setup for the simulation was a highly time consuming activity. Starting from the creation from scratch of the model in Autodesk Fusion360 that required an accurate design of each single part composing the robot structure. The export of the model in a format usable by Gazebo was a difficult task due to the lack of a pre-built tool in the software to export SDF files. The file export was performed thanks to an unofficial plugin that the software allows to install. Then the exported SDF file was completed with the addition of the sensors plugins and the modification of the collision shape as described in section 4.8.

The set up of the communication between Gazebo Garden [4] andROS2 [10] was source of many different and also time consuming problems. This was caused by the fact that Gazebo Garden [4] is the newer version of Gazebo [4] and is the default version integrated with the PX4 [7]toolchain. This newer version of Gazebo [4], and the only supported version for Ubuntu 22.04 onwards. Unfortunately it has not a recommended version of ROS2 [10] associated with it. In order to allow the communication between the two software it has to be installed the non official package `ros_gz`. The install instructions were often out of date, due to the rapid evolution of the software and the presence of different branches in the online repositories, each containing different software versions.

The addition of the plugins in Gazebo [4] was also a difficult task due to the lack of documentation. Some minor problems were encountered in the offboard control of the drone during the landing phase, caused by a lack in the documentation of the PX4 autopilot.

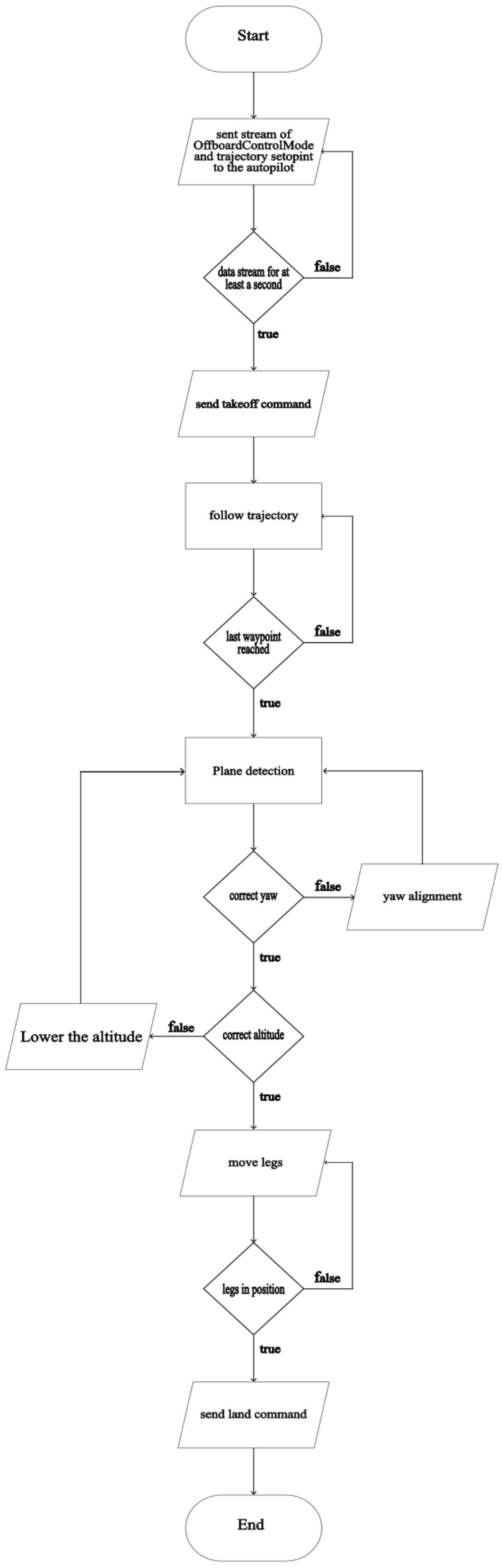


Figure 4.13: Flight flow chart

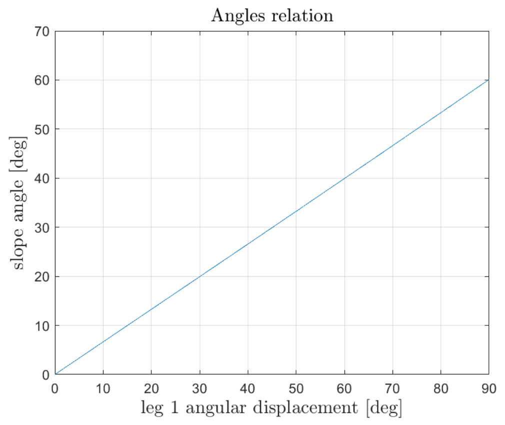
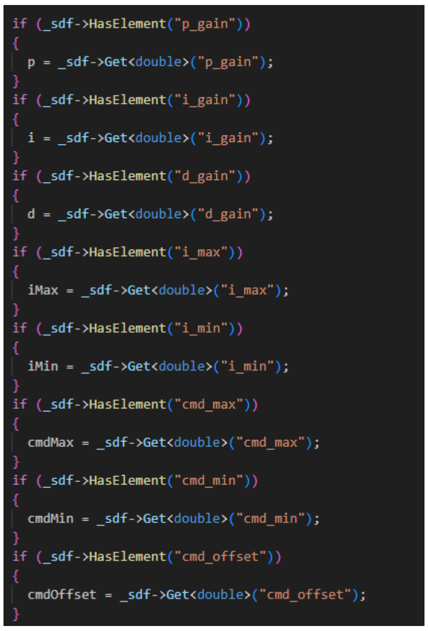
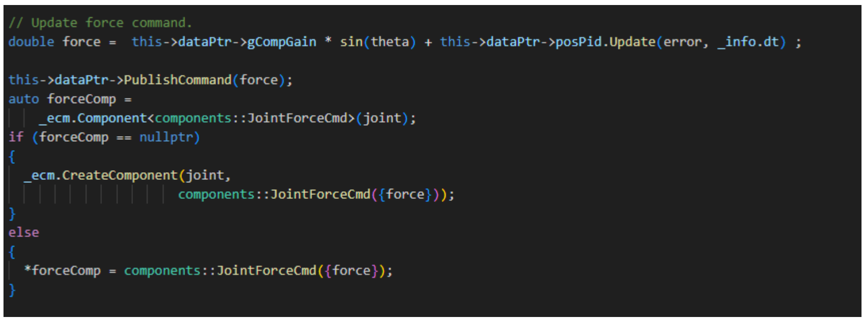


Figure 4.14: Linear relation between the slope of the landing surface and leg angular displacement



```
if (_sdf->HasElement("p_gain"))
{
  p = _sdf->Get<double>("p_gain");
}
if (_sdf->HasElement("i_gain"))
{
  i = _sdf->Get<double>("i_gain");
}
if (_sdf->HasElement("d_gain"))
{
  d = _sdf->Get<double>("d_gain");
}
if (_sdf->HasElement("i_max"))
{
  iMax = _sdf->Get<double>("i_max");
}
if (_sdf->HasElement("i_min"))
{
  iMin = _sdf->Get<double>("i_min");
}
if (_sdf->HasElement("cmd_max"))
{
  cmdMax = _sdf->Get<double>("cmd_max");
}
if (_sdf->HasElement("cmd_min"))
{
  cmdMin = _sdf->Get<double>("cmd_min");
}
if (_sdf->HasElement("cmd_offset"))
{
  cmdOffset = _sdf->Get<double>("cmd_offset");
}
```

Figure 4.15: Plugin parameters settings



```
// Update force command.
double force =  this->dataPtr->gCompGain * sin(theta) + this->dataPtr->posPid.Update(error, _info.dt) ;

this->dataPtr->PublishCommand(force);
auto forceComp =
    _ecm.Component<components::JointForceCmd>(joint);
if (forceComp == nullptr)
{
  _ecm.CreateComponent(joint,
                       components::JointForceCmd({force}));
}
else
{
  *forceComp = components::JointForceCmd({force});
}
```

Figure 4.16: PID output calculation with the addition of gravity compensation term

Chapter 5

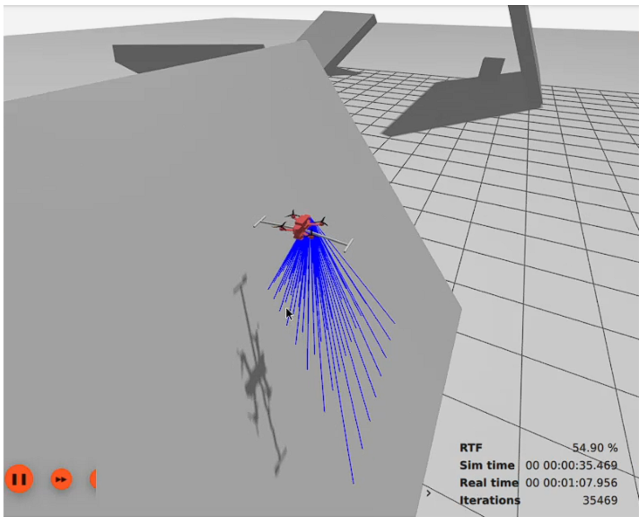
Flight Simulation Results and conclusions

The simulations of a complete flight were conducted in Gazebo [4] and consisted in several steps. The drone starts from an initial position laying on the flat ground plane waiting for commands. When the connection with the ROS2 [10] node is established, after the start command given by hand, the drone takes off. After following the preloaded trajectory setpoints, the drone is hovering above the landing surface. Above the target landing site a phase starts with several readings of the ToF sensor to retrieve information about the landing surface, with the following alignment of the yaw angle of the drone and a lowering of the altitude to get more accurate measurements. When the tip of the leg facing the slope is at an estimated altitude of $30cm$ it is sent a request to the ROS2 [10] service node preposed to manage the leg movements. When the legs are in position, the autonomous landing command is sent to the PX4 [7] autopilot. In figure 5.1 it is represented the UAV during the lowering altitude and sensor reading phase. The leg reference is sent as a ramp of increasing angle displacements instead of a single step with the aim of reducing even more the overshoot in the leg movements and avoid as much as possible oscillations in the position of the drone.

The legs positions and measured torques during the flight can be seen in figure 5.3. The graphs are reporting three different simulation results for different gains of the PID controller. It can be seen that for a choice of gains the overshoot is greater with respect the other two cases, this is because those gains where positioning the poles of the system slightly outside the allowed region, it is evident as a reposition of the poles inside the prescribed region has increased the performances, in particular has reduced the overshoot. In the graphs representing the angular position it is possible to observe different phases:

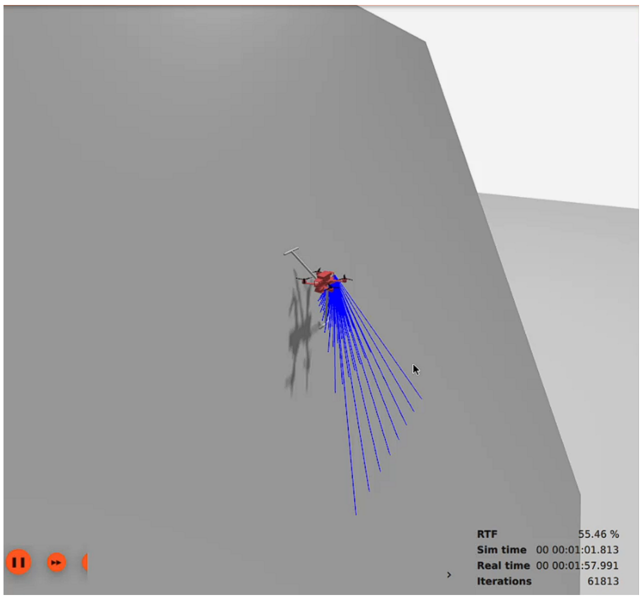
1. The legs are maintaining a constant position, in particular the default position for horizontal plane landing which is $-1.4834rad$ for the leg number 1 and $1.4834rad$ for leg 2. As previously said in this position the legs are not in the field of view of the ToF sensor when it is performing the detections of the surface. In the torques graphs it can be observed the time of take off of the drone, in which the measured torque changes from $\pm 1Nm$ to $\pm 0.2Nm$ that is the torque needed to compensate for the gravity force.

2. Leg movement; there is a quick change in their positions. Subsequently the



```
RTF           54.90 %
Sim time   00 00:00:35.469
Real time  00 00:01:07.956
Iterations          35469
```

Figure 5.1: The UAV hovering above the landing surface



```
RTF           55.46 %
Sim time   00 00:01:01.813
Real time  00 00:01:57.991
Iterations          61813
```

Figure 5.2: The UAV landed on the landing surface

legs maintains the prescribed position for the landing. In the torques graphs it is visible a peak corresponding to the legs movement.

3. Landing moment is highlighted by a relevant change in the measured torque. In particular before the evident change in the torque measured, it is possible to notice a small peak, that is the moment in which the legs are touching the landing surface.

After that moment the PID controller is increasing the commanded torque in order to bring back the legs at the prescribed position. In fact after the impact with the landing surface legs slightly change their position. For two gains values in the torques graphs it can be seen a slow decaying of the measured torque due to the fact that the legs are slowly reaching the target angle causing a decrement also in the commanded torque. The error after landing is different between the two legs, and it is depending also on the slope value of the landing surface. In fact for high slope value surfaces such as at $60°$ it has to be increased the friction coefficient in order to be able to perform a safe landing to avoid the slipping of the skids on the surface. In general, errors are higher for leg number one, which is positioned lower than leg number two and supports the majority of the drone weight. The greater error measured for the legs position after landing is of $0.0176rad$ for the leg number one which converted in degrees is approximatively equal to $1°$. Although the final error in the leg position during the simulation, it is important to notice that the leg displacement from the target position after landing would not be present in a real world applications. In fact the chosen actuator present in laboratory has a stall torque of $2Nm$ which is considerably higher with respect to the torque peak that can be observed in the graphs (fig. 5.3).This ensures that the real actuator will not move when legs touches the ground.

The attitude value were recorded through QGroundControl, the complete flight attitude graphs are shown in figure 5.4. Graphs represents the attitude for a complete flight and landing on a surface tilted of $60°$. It can be observed the changes in roll and pitch values during the flight phase. The greater displacement from zero degree is in the pitch angle, with a final value of $1°$ after landing. This is caused by an error occurred during the plane detection phase, in which the sensor has relevant errors in measuring laser distances, especially for longer rays that has a greater error. This causes a detection error that is increasing with the tilting angle of the landing surface, causing the legs to be positioned to land on slopes with an inclination different from the real one. Despite the minor error in the plane detection phase, the simulation results showed in the graphs highlights the effectiveness of the adopted solution, in fact a tilt of $1°$ is not compromising the landing and takeoff phases of the drone. As last observation it can be noticed the yaw angle that is changing from $-180°$ to $180°$ due to the fact that the drone was rotated of exactly $180°$ on the z axis, small errors in attitude detection were causing the atan2 operator used to retrieve the angle to quickly change between the two values.

5.1 Conclusions

In this study it has been re-designed the landing structure of a legged quadrotor starting from a previously developed project. The goal was to increase the landing capability of the UAV increasing the range of slopes for which is possible to safely

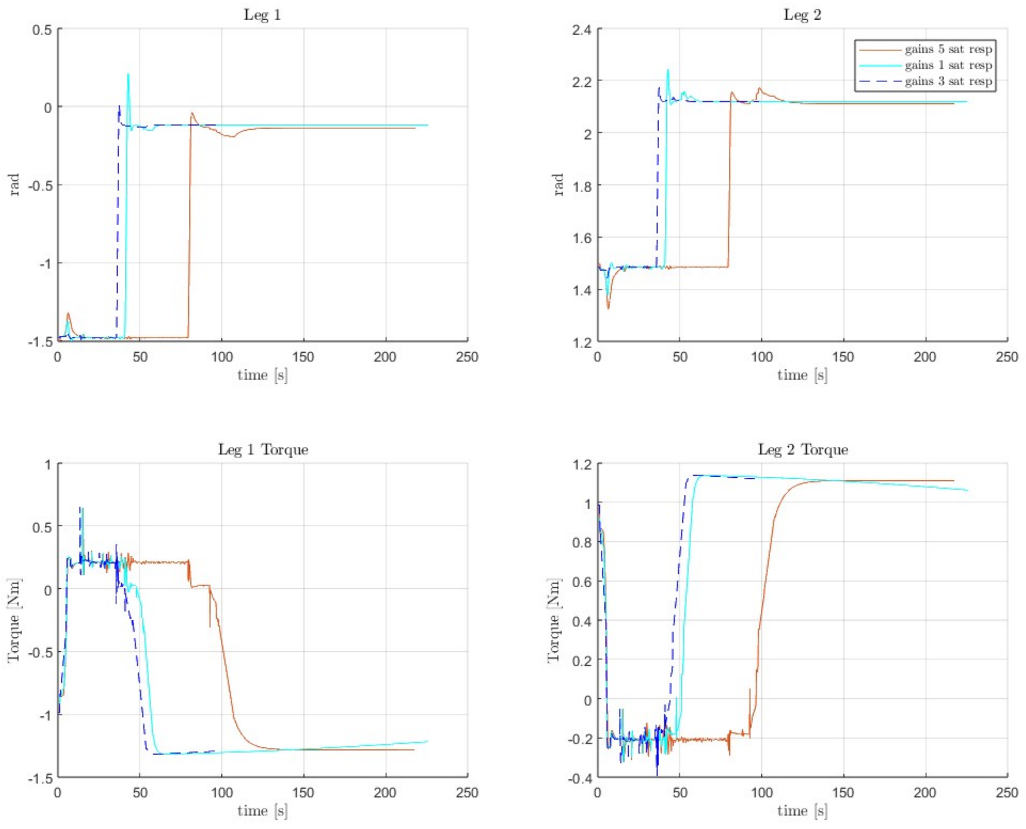
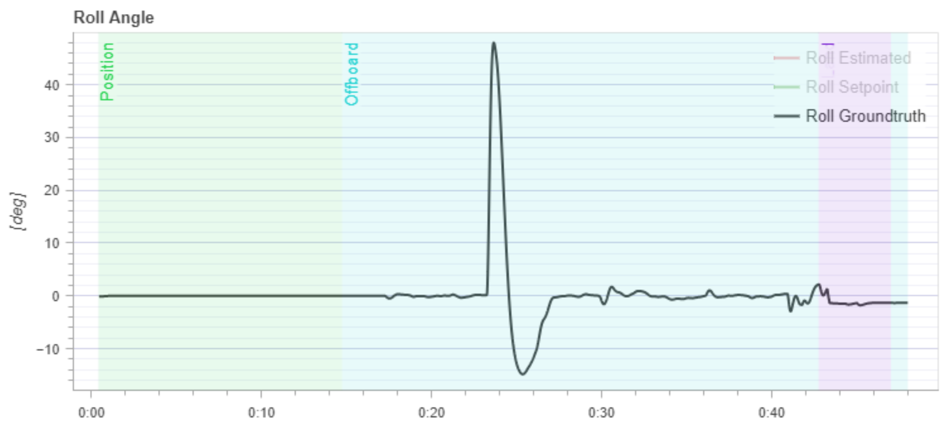
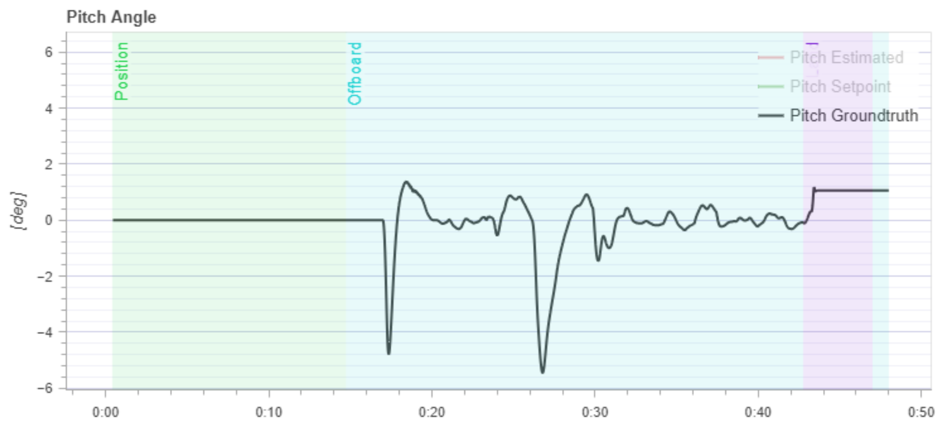


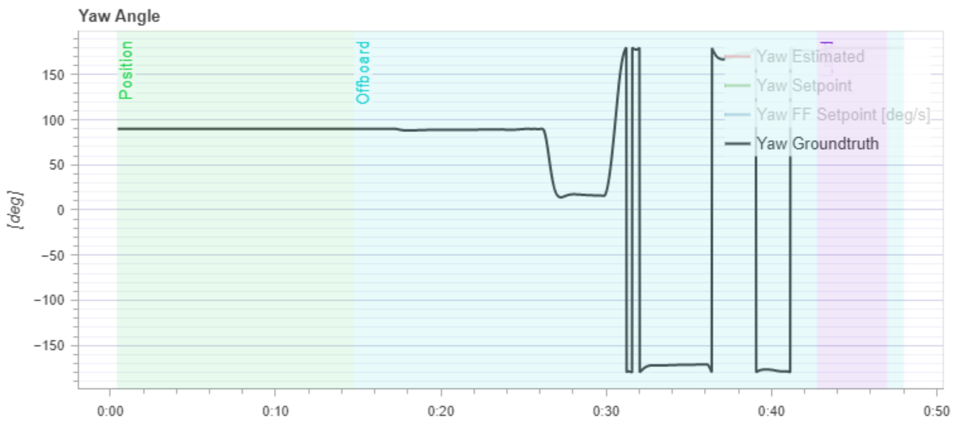
Figure 5.3: Legs position and measured torques



(a) Roll angle



(b) Pitch angle



(c) Yaw angle

Figure 5.4: qav attitude graphs

land up to 60°, the second goal was to ensure the optimality of the solution. To reduce weight during the flight and increase performances it has been used one actuator to move the landing structure. The quadrotor it has been equipped with a ToF sensor to sense the landing surface allowing an automatic orientation of the yaw angle and automatic leg positioning.

The first phase of the study involved the definition of an optimality criterion for the required dimension of the landing structure evaluating geometrical relations. Different proposed structures have been evaluated, with the subsequent choice of the structure satisfying the optimality criterion defined. The actuation of the landing structure has been modelled and it was performed a controller definition and tuning to optimize the performances. It was also studied the Plane detection algorithm in order to retrieve information from the ToF sensor.

The second phase involved the setup of a simulation to test the designed solution. It has been required the creation of the simulation world, UAV model and ROS2 nodes structure con manage the flight, the plane detection algorithm, and the legs movement.

The third phase involved the simulation of a complete flight and autonomous landing on a tilted surfaces, which information are not known in advance, except for the x y coordinates.

The proposed solution gave satisfactory results with the achievement of the objective of a safe landing on surfaces tilted up to 60°. The different landing surfaces tested have been correctly detected with a precise calculation of the slope value and yaw direction. As desired the movement of the legs didn't cause relevant oscillations in the UAV in flight position and the structure maintains the main body of the quadrotor horizontal with no contact with the landing surface. A possible future challenge is the implementation of the solution on the real quadrotor in the laboratory, with comparation of the results between the simulation and real world tests.

Bibliography

[1] ”Ahmed F. Mohanta J.C. Keshari A. et al. ”. “”Recent Advances in Unmanned Aerial Vehicles: A Review””. In: *”Arabian Journal for Science and Engineering ”* (2022), pp. 7963–7984.

[2] ”P. Pecho M. Hrúz I. Škvareková V. Ažaltovič ”. “”Optimization of Persons Localization Using a Thermal Imaging Scanner Attached to UAV””. In: *”2020 New Trends in Aviation Development (NTAD)”* (2020), pp. 192–196.

[3] ”P. Udvardy G. Tóth K. Pál T. Jancsó ”. “”Inspection of wind power plant turbines by using UAV””. In: *”2022 New Trends in Aviation Development (NTAD)”* (2022), pp. 247–250.

[4] ”Unknown”. “”About Gazebo””. In: *”https://gazebosim.org/about”* ().

[5] ”Unknown”. “”Civil drones””. In: *”https://www.easa.europa.eu/en/domains/civil-drones#group-easa-downloads”* ().

[6] ”Unknown”. “”Mavlink””. In: *’https://mavlink.io/en/’* ().

[7] ”Unknown”. “”PX4 Autopilot User Guide””. In: *”https://docs.px4.io/main/en/”* ().

[8] ”Unknown”. “”QGroundControl””. In: *’http://qgroundcontrol.com/’* ().

[9] ”Unknown”. “”ROS Documentation””. In: *’https://www.ros.org/’* ().

[10] ”Unknown”. “”ROS2 Humble Documentation””. In: *’https://docs.ros.org/en/humble/index.html’* ().

[11] ”Unknown”. “”uORB messaging””. In: *’https://docs.px4.io/main/en/middleware/uorb.html’* ().

[12] ”Xu Maozheng Takaki Takeshi Jiang Mingjun Ishii Idaku”. “”Development of Parallel-link-passive-gripper by Using a Multicopter’s Own Weight for Perching””. In: *”Proceedings of the SICE Annual Conference”* (2019), pp. 431–432.

[13] ”J. Kim M. C. Lesak D. Taylor D. J. Gonzalez C. M. Korpela”. “”Autonomous quadrotor landing on inclined surfaces using perceptionguided active asymmetric skids””. In: *”IEEE Robotics and Automation Letters”* 6.4 (2021), pp. 7877–7877.

[14] ”Massimiliano Mocellin”. “”Progettazione e Controllo di un Sistema Attuato di Atterraggio per un Quadrirotore””. 2022.

[15] ”X. Xie J. Wang Y. Zhang J. Xin L. Mu and X. Xue ”. “”A UAV Forest Fire Detection Method Based on Dual-Light Vision Images””. In: *”2023 CAA Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS)”* (2022), pp. 1–6.

[16] ”A. Kalantari K. Mahajan D. Ruffatto M. Spenko”. “”Autonomous perching and take-off on vertical walls for a quadrotor micro air vehicle””. In: *”IEEE International Conference on Robotics and Automation”* (2015), pp. 4669–4674.

[17] ”Maozheng Xu Taku Senoo Takeshi Takaki”. “”Condition analysis of a multicopter carried with passive skid for rough terrain landing””. In: *”ROBOMECH Journal”* 8.25 (2021).

[18] ”Yuri S. Sarkisov Grigoriy A. Yashin Evgeny V. Tsykunov and Dzmitry Tsetserukou”. “”DroneGear: A Novel Robotic Landing Gear With Embedded Optical Torque Sensors for Safe Multicopter Landing on an Uneven Surface””. In: *”IEEE ROBOTICS AND AUTOMATION LETTERS”* 3.3 (2018).