Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Théo Degeorges

Close Control of an Autonomous Drone for Physical Module Replacement on Mobile Platforms

Master's thesis in Electronic Systems Design
Supervisor: Dominik Osinski
Co-supervisor: Tor Arne Johansen
June 2021

NTNU
Norwegian University of
Science and Technology

ascend
aerial robotics team NTNU

Théo Degeorges

Close Control of an Autonomous Drone for Physical Module Replacement on Mobile Platforms

Master's thesis in Electronic Systems Design
Supervisor: Dominik Osinski
Co-supervisor: Tor Arne Johansen
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

Fully autonomous drones are getting more and more reliable and are willing to be used in an increasing amount of fields. The International Aerial Robotic Competition claims that repairing moving platforms in an autonomous way is an important one. This thesis will try to improve the control of an autonomous drone close to a moving boat's mast. This includes the creation of an attitude controller through LQR, an Extended Kalman Filter to estimate the position of the moving target and some strategy about the movements around the mast. A feed-forward and non-linear input to the LQR will be explored. The research includes real-life validation and collision testing. The results of the LQR will be compared with an autopilot built-in PID controller. This thesis shows that the LQR could not get significantly better than the PIDs and that a good EKF tuning in terms of error makes the drone jittering. For this kind of application, it is better to lose a bit of accuracy on the EKF side but make the flight smoother.

Contents

1 Introduction

Due to large improvements in drone accuracy and intelligence, fully autonomous drones are emerging, including those equipped with repairing capacities. This is highlighted by the scope of this year's International Aerial Robotic Competition (IARC)'s mission [10], requiring the creation of an autonomous drone that achieves a communication module replacement on a ship.

This thesis aims to improve autonomous drone capacities for repairing mobile platforms. More specifically, this project will go through the control of an autonomous drone in the proximity of a ship. The drone must consistently stay close to an oscillating mast. This problem is similar to trajectory tracking issues where a drone has to follow a time-parametrized reference. However, in this case, the reference is not known in advance and will be estimated in real-time by another project outside of the scope of this thesis. The research will be done in collaboration with Ascend NTNU, one of the student organizations that participates in IARC's competitions.

1.1 State of the Art Overview

The accuracy of UAVs had not been looked into previously within Ascend, but many papers did. [15] presents a real-time control in position for quadcopters in outdoor conditions. It proposes a PD control with gain scheduling with the aim of being used for trajectory control. It gets significant improvement over a simple PD in terms of rapidity, accuracy, overshoot and wind rejection. A different strategy is proposed in [3], which compares a PD control with an Extended Prediction Self-Adaptive Control (EPSAC) approach to MPC. The design of the EPSAC focuses on a fast response without overshoot, which looks promising. On the other side, [14] presents a simple quadcopter model for MPC and focus on keeping low development costs and using computationally light calculations. [4] also details a quadcopter Matlab model based on angular rate control. Finally, [19] presents the state of the art of load transportation and interaction between UAVs and their environment. And [11] develops a collision mitigation solution for contact inspection.

Control-wise, this thesis aims to investigate the use of a Linear Quadratic Regulator (LQR) controller. Some improvements of the LQR, such as feedforward and non-linear input, will also be looked into.

1.2 Description of Ascend-NTNU

Ascend NTNU is an organization of volunteering students from NTNU who designs and builds drones. Every year, Ascend aims to participate in the International Aerial Robotics Competition (IARC). It is *"The World's Premier*

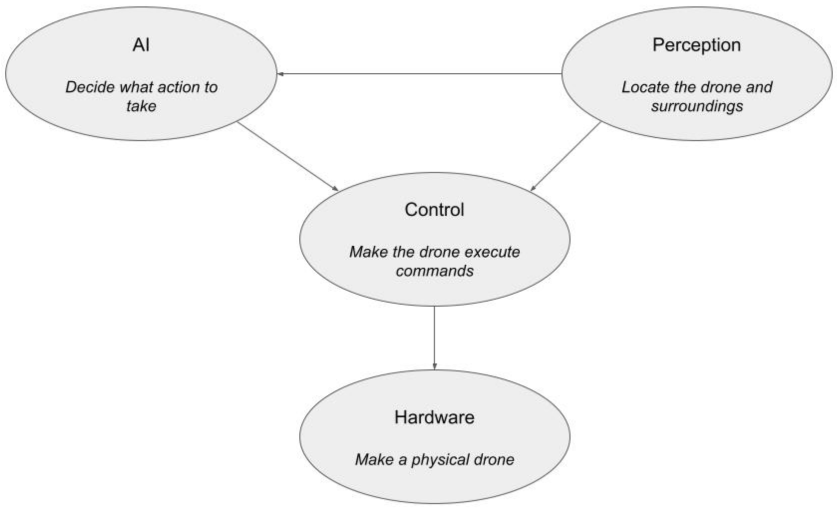

AI
Decide what action to take
Perception
Locate the drone and surroundings
Control
Make the drone execute commands
Hardware
Make a physical drone

Figure 1: Schema of the link between the different technical groups within ascend

and longest running Aerial Robotics Challenge" where competitors have to design and produce state-of-the-art aerial robots to face challenges. At the time they are proposed, the latter are considered as 'impossible' and it is possible that not any team successfully manages the mission. In that case, the same mission will be held the following year.

The organization is composed of 4 technical groups articulated as in fig. 1. Among them, the Hardware group deals with all the physical components of the drone. This includes mechanical and electrical parts and electronics. The Perception group creates the 'eyes' of the drone by adding sensors and gather and analyze their data. The AI group represents the brain of the drone. They take high-level decisions throughout the mission. This group is also responsible for making and maintaining a simulator where all groups can test their system independently. Finally, the Control group, which the author is a member of, executes orders from the AI group. It includes the control of the drone during several kinds of operations, such as taking off, traveling, or executing the module replacement operation. From the data figured out by the perception group, the AI group may want to make the drone move through a certain waypoint at a certain speed, which will be handled by control.

1.3 Description of the Mission of the Year

The mission was created in 2019 but has not been solved last year because of the current pandemic. A fully autonomous aerial robot (drone in our case) will receive a 2kg payload that represents a communication module (fig. 2).

At the start, the drone will take off and fly at an altitude of less than than 15 m to approximately 3 km to apprehend a vessel. The aerial robot will then remove the communication module from the mast of the ship (dropping it on site) and replace it with the communication module payload that it is carrying (fig. 3). Upon completion of the module swap, the aerial robot will return to its point of origin and land. The entire mission must be completed in 9 minutes. If several teams were to succeed in the mission, the fastest team wins.

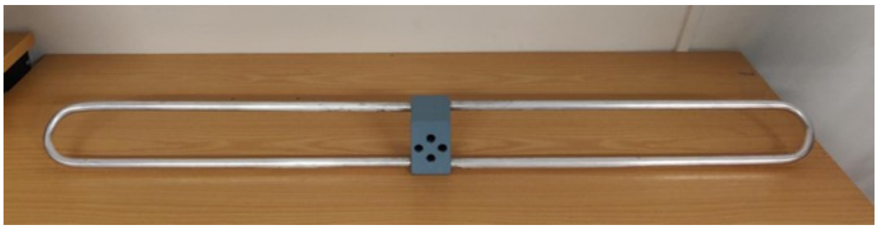


Figure 2: Picture of a communication module

More specifically, the created drone will need to perform

- Precision manipulation of heavy and large objects
- Fast outdoor operations over long distances
- Interaction with moving frames of reference
- Aerial robotic repair of mobile platforms
- Optical recognition
- Using ONLY onboard computing

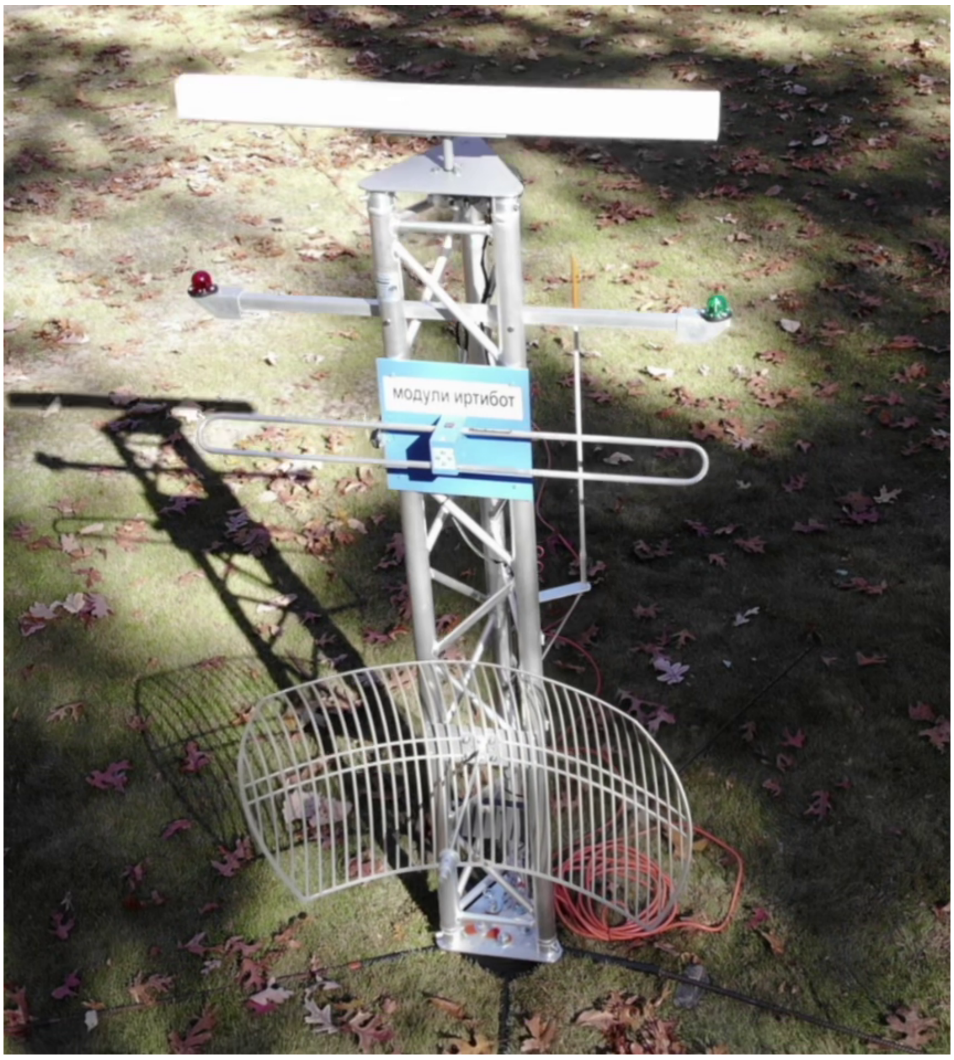


Figure 3: Picture given by the competition of the mast with its communication module

1.4 Description of the Drones

A 1.9 x 2.3m drone is being built. The drone must be wide because it must carry heavy loads. Also, for stability reasons, the module should not be in front of any of the propellers at any time. Therefore, the drone must have some room to fit the mast between 2 propellers and be able to place a module on it. A render of the drone can be seen fig. 4. In this figure, the mast will fit between the two propellers on the left, which is in front of the drone.

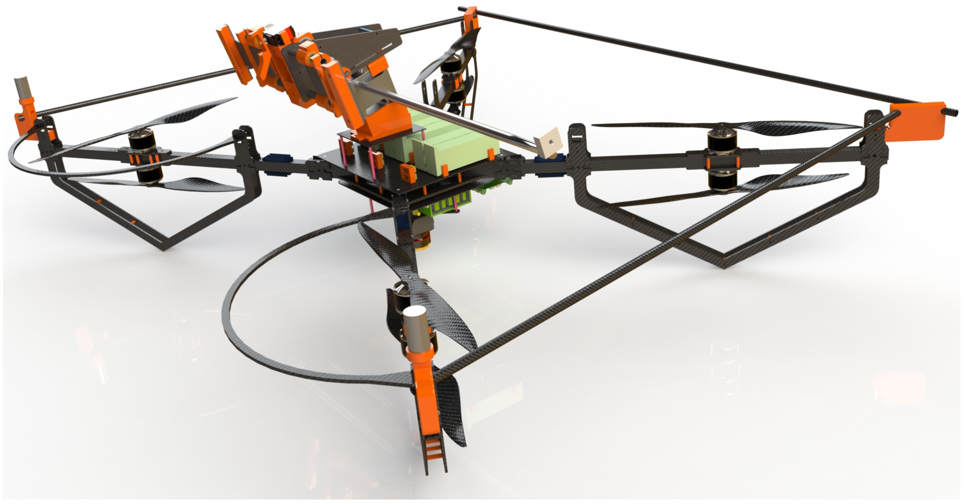


Figure 4: Render of the big drone that will be used during the competition

In addition to this big drone, Ascend has 2 smaller drones from the previous missions (fig. 5). Their small size makes them more convenient for testing and less risky.

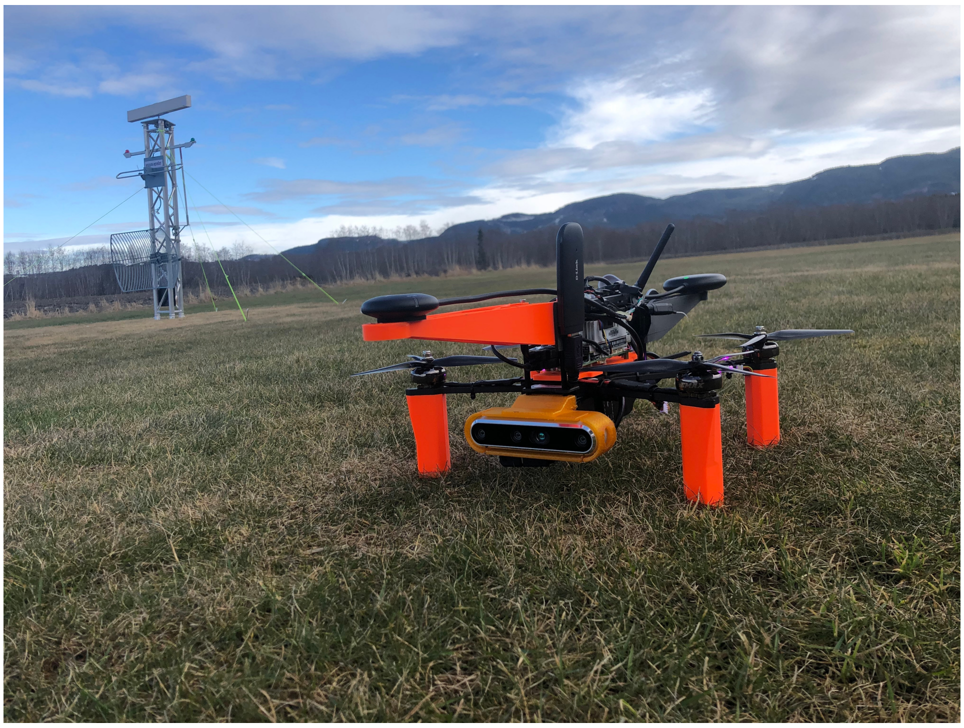


Figure 5: Picture of one of the test drones

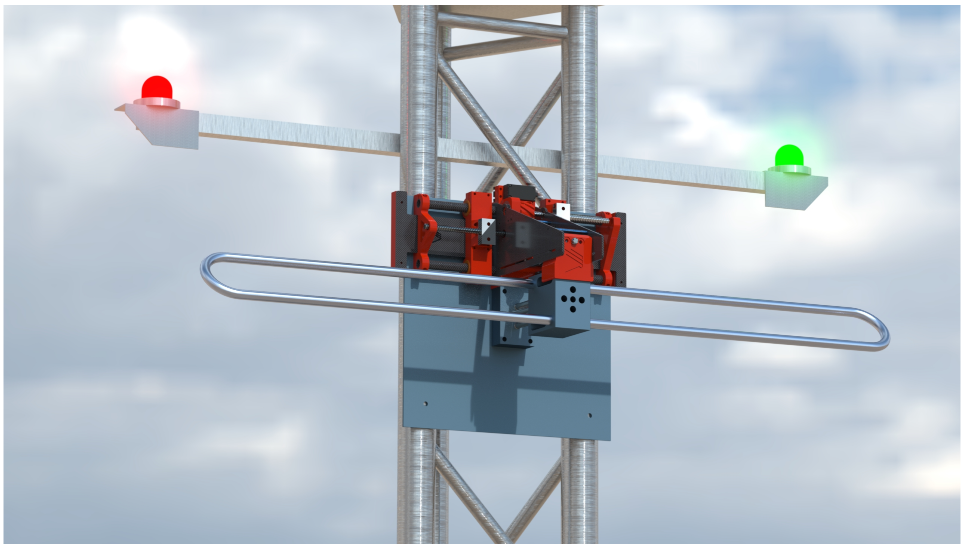


Figure 6: Render of the FaceHugger on the Mast

Every drone is equipped with a flight controller (pixhawk4 run with Ardupilot autopilot) that collects data from the different sensors of the drone and controls the propellers. The collected data is filtered through an EKF to estimate the drone state (6D position and velocity). Every drone is also equipped with a companion computer which is used as the brain of the drone in autonomous mode. This computer can access any information that the flight controller has and send it commands such as position or attitude set-points. It communicates using ROS through the Mavros interface as it is robust and widely used. This interface allows the use of the same software for both simulator and real-life testing. It makes the tests easier and safer.

1.5 Strategy for Module Replacement

The drone that will be used in the competition will hold an independent autonomous mechanism that carries the communication module. It is called the FaceHugger (see fig. 6). Its goal is to settle on the mast and autonomously take the existing communication module off and place the new one that it is carrying. The FaceHugger will be set by the drone on the blue plate where a communication module is mounted. The middle top of the blue plate will be called the interaction point. This is where the FaceHugger (more precisely, its hook) is aimed to be placed.

The use of this FaceHugger will largely reduce the complexity of the mission. First, the accuracy needed to set the FaceHugger is much less than the one needed to place the module without any other helping system. Secondly, the time during which the drone will need to be across the mast is also reduced by a lot. In addition, once the FaceHugger is set, the drone can leave the mast and go back to its base. without it, the drone would need to take the first module out and place the new module in a relatively slow manner. That should therefore reduce the overall mission time.

1.6 Scope of the Thesis

This thesis is the continuation of a one-semester project [5] on the same subject. It aims to help Ascend NTNU in its creation of a drone for IARC worldwide competition in the United State of America. The aim of this thesis is to handle the control of the drone for the whole module replacement process. It begins when the drone hovers still in front of the mast and ends when the FaceHugger is successfully placed on the mast and the drone safely exited the area.

This report explores a solution based on LQR control, an EKF, and a state machine. The Kalman filter will estimate the state of the mast (position, velocity and acceleration) that the regulator will use to control the drone toward the position defined by the state machine. The LQR will only control the drone on a horizontal plane. The control in altitude will be mentioned but kept outside of the scope of the thesis since it has been considered less challenging. Several improvements of the basic LQR controller will be explored, such as non-linear input and feed-forward. This thesis will also compare the homemade LQR against an autopilot built-in position and velocity PID controller. However, getting the best results possible from the latest controller will not be a priority.

The report will first elaborate on the scope of the thesis, setting the requirements and boundaries. It will then explain the theory used in the experiments that will follow. The latter includes both simulator and real-life testings. Then, the results will be discussed in order to establish a general conclusion.

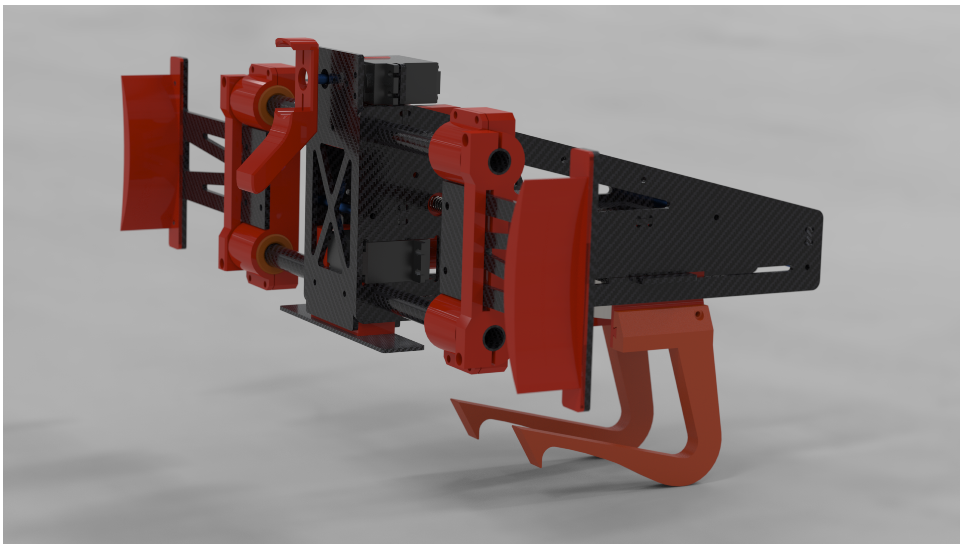


Figure 7: Render of the side and back of the FaceHugger

2 Requirement Analysis

The replacement of the communication module is a mission critical task. This has been simplified by the placement of the FaceHugger system. During this thesis, it will be assumed that the module replacement operation will be successful as long as the FaceHugger is hooked on the top of the blue plate.

2.1 Accuracy Requirement

The FaceHugger must be placed on the blue plate of the mast (fig. 3). An example is shown fig. 6. The position of the FaceHugger on the top of the blue plate should not impact it success as it can center itself after being set. Still, the room to place the FaceHugger is narrow. The diagonal beam between the two outer poles reduces the available room.

The FaceHugger is hanged on the mast from its front hook (see fig. 7). This hook ends 40mm behind the backplate (the part that will be against the blue plate of the mast). So the placement has to be 20mm accurate on the x axis (the axis going forward from the mast). This hook can be positioned on the whole bottom left triangle marked by the diagonal beam, the top of the blue plate and the left outer pole of the mast. This constrains the design to about 50mm accuracy on the y and z axes.

As the mast is moving at a maximum speed of 470mm/s (as estimated later in fig. 8), the drone must not be delayed more than about $20/470 = 0.043$sec (when the mast is at maximum velocity). This delay can either be positive or negative, the drone should not be ahead of the module by more than 0.043sec.

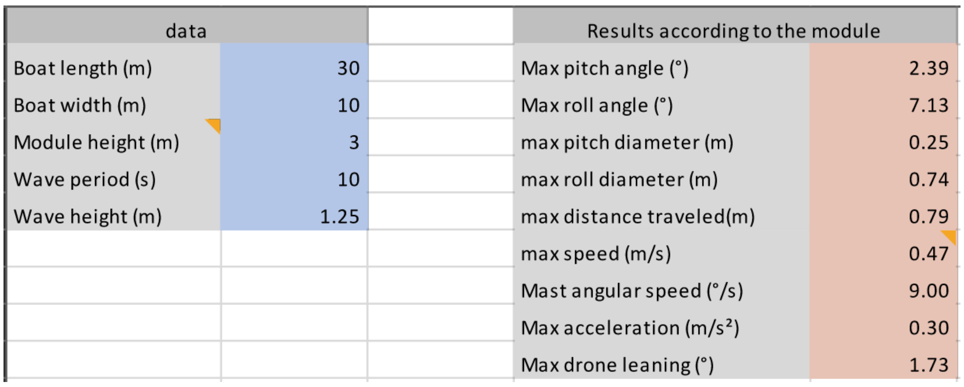
The placement being consistently successful, it becomes important to place the FaceHugger as quickly as possible. This is because speed is important in this competition. The winning team is the team that finishes the whole mission in the quickest manner.

2.2 Boundaries

The movement of the mast simulates natural wave motion (sea state ≤ 3), resulting in a mostly sinusoidal oscillation of the ship in both pitch and roll. Neither heave (vertical component of motion) nor forward motion will be implemented at the competition. Nonetheless, noise will be introduced so that the pitch and roll are not entirely predictable.

An estimate of the oscillation of the mast has been calculated on an interactive sheet fig. 8 to estimate the scale of the amplitude and the speed of the movements. It results on relatively slow movements: the maximum expected speed near $0.5 m.s^{-1}$ with a maximum acceleration of $0,3 m.s^{-2}$.

The precision of the position of the drone is not only limited by the accuracy of the GLSs. Ardupilot stores the drone position using latitude and longitude coordinates and the precision of these coordinates limits the accuracy with which the position is stored to 11mm at the equator and 9mm in Trondheim (Norway). A post on Ardupilots's forum [2] gives a more detailed explanation and proposes a solution to increase accuracy to sub-millimeters. If the default storage accuracy is good enough for most applications, the drone needs to be controlled with 20mm to 50mm accuracy. The way the position of the drone is stored may be limiting. Still, since the issue does not seem critical and the solution was not trivial to implement, it has not been used in the scope of the thesis.



data			Results according to the module	
Boat length (m)	30		Max pitch angle (°)	2.39
Boat width (m)	10		Max roll angle (°)	7.13
Module height (m)	3		max pitch diameter (m)	0.25
Wave period (s)	10		max roll diameter (m)	0.74
Wave height (m)	1.25		max distance traveled(m)	0.79
			max speed (m/s)	0.47
			Mast angular speed (°/s)	9.00
			Max acceleration (m/s²)	0.30
			Max drone leaning (°)	1.73

Figure 8: Estimate of the mast oscillation amplitudes

3 Theory

In this chapter, we will describe the theory around the different tools used to achieve the thesis results. We will first go through the different types of control available with Ardupilot autopilot. Then we will detail the use of a Linear Quadratic Regulator (LQR). Finally, we will explain the use of an Extended Kalman Filter (EKF).

3.1 Control with a Flight Controller

The flight controller hardware used in Ascend is the Pixhawk4 [16]. On this controller, the Ardupilot autopilot is used [1]. The main goal of the autopilot is to assist the user in the control of the drone. Even during autonomous flights, the autopilot is very useful. It has several layers of control [8] that allows the designer to only consider high-level tasks.

3.1.1 Basic Control of the Drone

During basic autonomous control, one gets the opportunity to set the drone position, velocity, or both at the same time. Ardupilot will handle the control has show fig. 9. Position error is translated into a velocity target through a squared root regulator. If a velocity set-point is set, it is added to the just found target. Then, a PID converts the velocity error into a desired leaning angle. The latter is controlled by a lower-level control with a PID on the leaning rates.

3.1.2 Attitude Control

Ardupilot also allows for a lower-level control of the drone. One can decide to control the attitude of the drone (fig. 10). It is then possible to set the orientation, the body rate, and the thrust of the drone. The controller is also used after the previous controller in case of a position or velocity control.

On the one hand, using this middle-level control allows a faster response of the drone. Indeed, it is much faster to set the drone to a specific orientation than a specific position or velocity. One can also get a better hold on the drone behavior as they design themselves what should the drone do to achieve their goals. On the other hand, it requires a more complex algorithm, it is less user-friendly and becomes more complex to debug.

3.2 Linear Quadratic Regulator

The work done during last semester's project suggested that a position and velocity control is not enough to achieve our needs (less than 5 cm accuracy with real data). The implementation of a more complex control strategy has then been looked into. The implementation of a Linear Quadratic Regulator

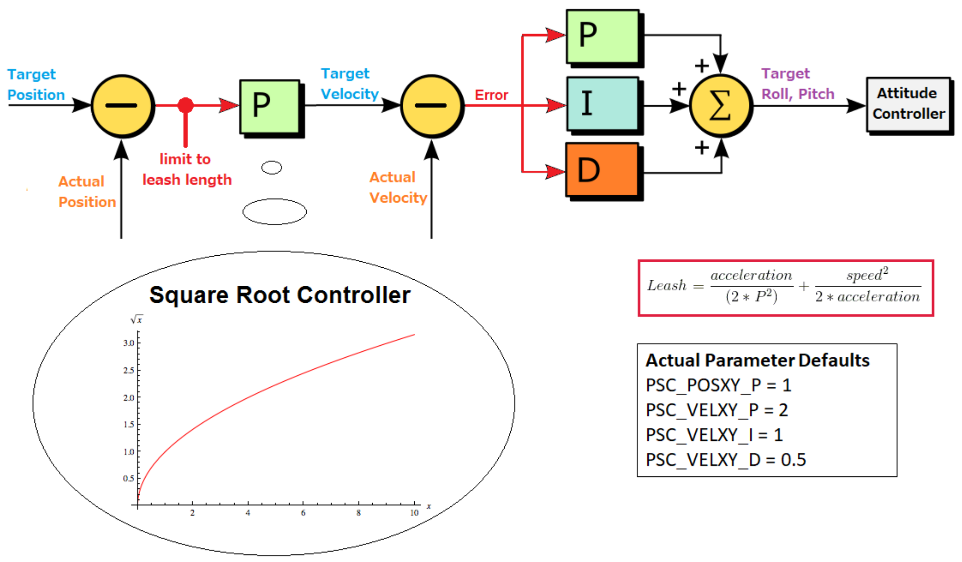


Figure 9: schema of the high-level control with Ardupilot (2020, ArduPilot Dev Team. [8])

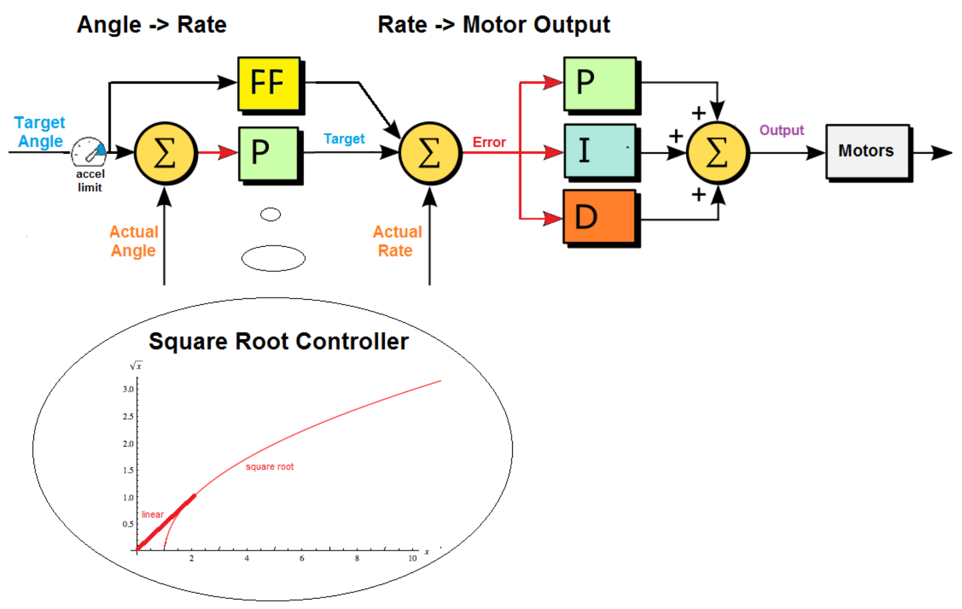


Figure 10: schema of the middle-level control with Ardupilot (2020, ArduPilot Dev Team. [8])

(LQR) is a natural step after a PID. The LQR is easy to design and gives high-performance control. Well tuned, this regulator can be both fast and accurate.

To be efficient, the LQR needs to control low latency inputs. The attitude of the drone is quick to set and directly accessible with Ardupilot. It is not too low level so that it is still fairly easy to control and to get a feeling of how it behaves. This regulator will only be used on a horizontal plane. Indeed, attitude control only allows controlling the altitude of the drone through the thrust. But the latter is not independent of the acceleration of the drone on the horizontal plane, which would make the control much more complex. Fortunately, altitude control is less critical than horizontal control as the oscillations are smaller slower. We will then let Ardupilot control the altitude of the drone internally, in a similar way as for position control.

Therefore, we will only consider the LQR for horizontal movements.

3.2.1 Model for LQR

Controlling the attitude of the drone is similar to controlling its acceleration. Assuming that the drone will keep a slowly varying altitude, the upward force from the propellers will almost be constant. The acceleration of the drone is then a function of the leaning angle of the drone. On each axis,

$$acc = g \times tan(\phi) \tag{1}$$

with g the gravity constant and ϕ the leaning angle of the drone.

The LQR will therefore send acceleration directives through leaning angle set-points to the drone. The input of the controller must then be the error in position and in velocity, or the relative position and velocity of the drone compared to the mast.

We will make the assumption that the control of each horizontal axis (x and y) is independent. They will be controlled the same way and therefore, the attitude input to the drone becomes

$$U = K_{LQR} * \begin{pmatrix} position \\ velocity \end{pmatrix} \tag{2}$$

A bloc diagram of the LQR control can be seen fig. 11. The scale of each element of LQR gain can be found using the emblematic Q and R matrices. They are convenient to efficiently find a good pre-tune. However, in the case of this thesis, these matrices will not be used for further tuning. We will prefer to adjust the K matrix gain directly.

On the LQR, the weights of the matrices R and Q respectively rates how much power can be set to the system versus the control error. A first approximation of these matrices can be done using Bryson's rule where

$$Q_{ii} = \frac{1}{max\_acceptable\_error\_of\_state\_i^2} \tag{3}$$

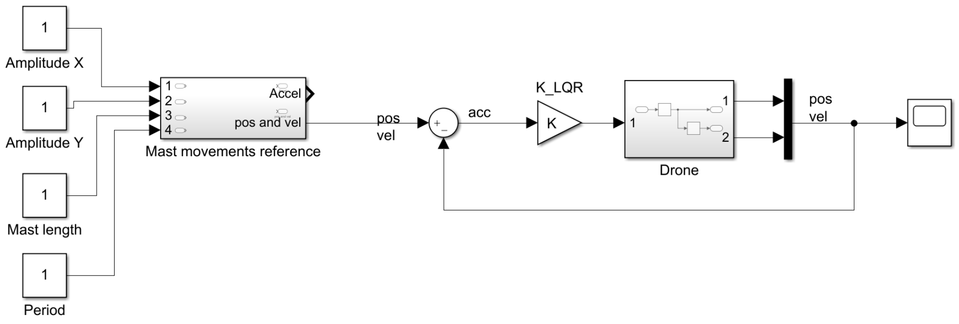


Figure 11: Bloc Diagram of an LQR

and

$$R_{jj} = \frac{1}{max\_acceptable\_value\_of\_input\_j^2} \tag{4}$$

We are using the following estimations.

- The maximum allowed error in position is 20mm.
- The maximum allowed error in velocity is 100 mm/s. (Considering 0.2sec latency)
- The maximum acceptable input for this application is 1m/s^2 (the estimated maximum acceleration of the mast is 0.5m/s^2)

Therefore, we get

$$Q = \begin{pmatrix} 2500 & 0 \\ 0 & 25 \end{pmatrix} \tag{5}$$

And

$$R = \begin{pmatrix} 1 \end{pmatrix} \tag{6}$$

From these matrices and considering a double integrator for the model of the drone, Matlab can easily calculate the corresponding gain matrix. Using the 'lqr' function has been, we obtain

$$K_{LQR} = \begin{pmatrix} 50.0 & 14.1421 \end{pmatrix} \tag{7}$$

Unfortunately, this rule has not been applied with the good parameters at the beginning of the thesis and it resulted in a completely different matrix:

$$K_{LQR} = \begin{pmatrix} 0.4189 & 1.1062 \end{pmatrix} \tag{8}$$

The actual theoretical value will still be discussed once the best experimental value will be found.

As mentioned earlier, the tuning of the LQR will be done from the K_{LQR} matrix. Its elements respectively represent the weight of an error in position and in velocity. Even though it is more common to use the Q and R matrices or the pole placement method to tune this regulator, tuning the gain matrix directly allows to bypass the simplifications made in the model and ensure that the best tuning possible is reachable. In this case, the K_{LQR} matrix is short and explicit, so it is also intuitive and convenient to tune it directly.

3.2.2 Feed-Forward

However fast is a control, it can never be instantaneous. It will always be some delay between a change of the input and the correction of the drone. In our case, the position target (the mast) is continuously oscillating, and so does the input. Therefore, we expect the drone to be slightly delayed compared to its reference. This delay may be shortened using two different techniques. First, one could try to predict the future and give future set-points to the drone to overcome this issue. This will be discussed section 3.3.3.

Adding a feed-forward to the control strategy can also improve the latency. Indeed, considering that we can estimate the acceleration of the mast, we can increase or decrease the inputs whether the mast is accelerating or decelerating. A block diagram including the feed-forward can be seen fig. 12. However, it may be difficult to estimate an accurate acceleration from position measurements. A noisy estimation may worsen the results.

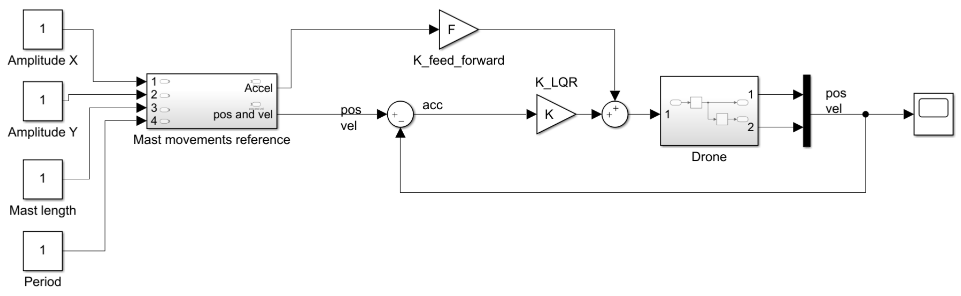


Figure 12: Bloc Diagram of an LQR with Feed-forward

3.2.3 Non-Linear Input Error

An accurate control requires relatively big gains. However, they can quickly become very unstable. This is partially due to the inertia of the system. As cited in the introduction, [15] propose a gain scheduling approach that allows a faster, more accurate and with less overshoot response. In this solution, the gain of the system increases when the error decreases. In a similar way, the Ardupilot autopilot presents a square root feedback control that is also equivalent to having a higher gain for small errors.

Following these examples, it seems relevant to try to take the square root of the error before applying the LQR gain. The block diagram becomes the one fig. 13. As the square root changes the input to the LQR, a new tuning will have to be done.

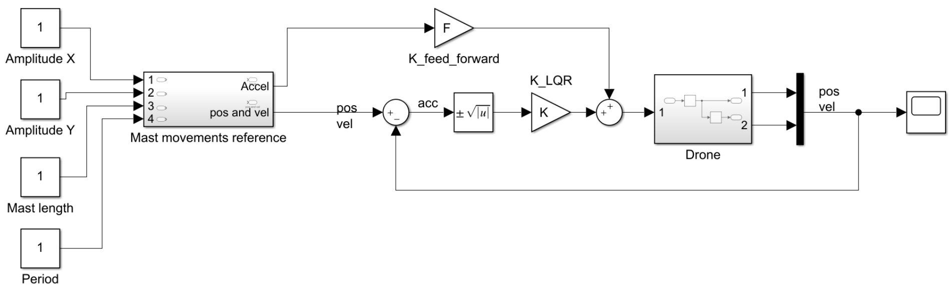


Figure 13: Bloc Diagram of an LQR with feed-forward and squared root block

3.3 Kalman Filter

The Kalman Filter is a widely used tool to estimate the state of a linear system from a series of measurements. The mission requires a very accurate estimate of the mast position, and therefore filtering of the measurements is necessary. The Kalman filter is the best estimator in the minimum mean-square-error sense. It allows merging the measurements from several sensors toward the estimation of the state of the system and its covariance. Finally, it is relatively easy to implement and not computationally heavy for today's computers.

The Kalman filter proceeds in two steps. First, the prediction step uses a model to estimate the future state of the system. Then the update step adjusts the estimation considering all the measurements that have been made. The Kalman filter assumes that all the measurements include a white noise of a known covariance. These two steps have to be repeated for each time step.

It is also possible to do the prediction step at a higher frequency than the update step. This method can make the prediction more accurate if the frequency of the measurement is not high enough compared to the system dynamics. It can also make the Kalman filter more robust over not constant measurement frequency.

3.3.1 Extended Kalman Filter

If the Kalman filter only works with a linear system, an extended version can handle a non-linear system. However, as shown in [13] this filter only converges locally and the global convergence can not be proven. In practice, the Extended Kalman filter (EKF) does not present convergence issues and it is very used for non-linear systems.

The EKF looks very similar to the linear Kalman filter, but with non-linear equations for the prediction step and locally linearized matrices for the update step. The EKF considers the following non-linear system

$$x_{k+1} = f(x_k, u_k) \tag{9a}$$
$$y_k = h(x_k, u_k) \tag{9b}$$

Where:
x: is the state of the system
u: is the input to the system
y: is the output of the system (being measurable)

At each prediction step:

$$\hat{x}_{k+1} = f(x_k, u_k) \tag{10a}$$
$$Pp_{k+1} = F_k * P_k * F_k^T + Q_k \tag{10b}$$

And for the update step:

$$K_{k+1} = Pp_{k+1} * H_{k+1}^T * (H_{k+1} * Pp_{k+1} * H_{k+1}^T + R_{k+1})^{-1} \tag{11a}$$
$$x_{k+1} = \hat{x}_{k+1} + K_{k+1} * (y_{k+1} - h(\hat{x}_{k+1}, u_{k+1})) \tag{11b}$$
$$P_{k+1} = (Pp_{k+1} - K_{k+1} * H_{k+1} * Pp_{k+1}) \tag{11c}$$

Where:
x and $\hat{x}$: are the estimated and predicted state of the system,
u and y: is the input and measurement vector of the system,
P and Pp: are the estimated and predicted covariance matrix,
K: is the Kalman gain,
H and F: are the Jacobian of the measurement and the prediction function
Q and R: are the covariance matrices of system uncertainty and the measurements noise (considered white).

3.3.2 Application of the Model for this Thesis

In this study, the EKF will help to estimate the state of the mast of the ship. From the mission's rules, and as explained in section 2.2, the oscillations of the mast will be mostly sinusoidal in both pitch and roll. The model will consider constant oscillation characteristics. Their slow variations will be handled by the filter as model uncertainty. Considering the mast in a polar coordinate system, the state vector is as follow:

$$X = \begin{pmatrix} p \\ r \\ \dot{p} \\ \dot{r} \\ \omega \\ L \end{pmatrix} \tag{12}$$

Where:
p and r: are the pitch and the roll angle of the mast,
$\dot{p}$ and $\dot{r}$: are the derivative of the pitch and the roll of the mast,
ω: is the frequency of the wave,
L: is the length of the mast.

The pitch and the roll of the mast are expected to have the following shape:

$$\begin{aligned} p &= A_p sin(\omega t + \phi_p) \\ r &= A_r sin(\omega t + \phi_r) \end{aligned} \tag{13}$$

therefore:

$$\begin{aligned} \ddot{p} &= -A_p \omega^2 sin(\omega t + \phi_p) \\ \ddot{r} &= -A_r \omega^2 sin(\omega t + \phi_r) \end{aligned} \tag{14}$$

Which is equivalent to

$$\begin{aligned} \ddot{p} &= -\omega^2 p \\ \ddot{r} &= -\omega^2 r \end{aligned} \tag{15}$$

Finally, neither the f nor the h function depends on the input set into the system. So, the following prediction function is obtained:

$$f : x = \begin{pmatrix} p \\ r \\ \dot{p} \\ \dot{r} \\ \omega \\ L \end{pmatrix} \longmapsto \begin{pmatrix} p + \dot{p} \times dt \\ r + \dot{r} \times dt \\ \dot{p} - \omega^2 \times p \times dt \\ \dot{r} - \omega^2 \times r \times dt \\ \omega \\ L \end{pmatrix} \tag{16}$$

Because the frequency of the waves ω is unknown, the system is not linear or linearizable. The Extended version of the Kalman filter must be used and the following Jacobian matrix has been calculated:

$$F = \begin{pmatrix} 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \\ -\omega^2 dt & 0 & 1 & 0 & -2\omega p \times dt & 0 \\ 0 & -\omega^2 dt & 0 & 1 & -2\omega r \times dt & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{17}$$

On the measurement side, the Perception group can give information on 4 states: the 3D position of the interaction point (the middle top of the blue plate, where the FaceHugger should be placed) and the pitch of the mast. It gives the following h function:

$$h : x \longmapsto y = \begin{pmatrix} x \\ y \\ z \\ pitch \end{pmatrix} = \begin{pmatrix} Lsin(p) \\ Lsin(r) \\ Lcos(p)cos(r) \\ p \end{pmatrix} \tag{18}$$

Once again, we see that the system could not be linearized. The following Jacobian matrix has been calculated:

$$H = \begin{pmatrix} Lcos(p) & 0 & 0 & 0 & 0 & sin(p) \\ 0 & Lcos(r) & 0 & 0 & 0 & sin(p) \\ Lcos(r)sin(p) & Lcos(p)sin(r) & 0 & 0 & 0 & cos(p)cos(r) \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{19}$$

The last things to establish are the Q and R matrices presented eq. (11). The Perception group estimates the covariance of their measurement to be diagonal:

$$R = \begin{pmatrix} 0.001667 & 0 & 0 & 0 \\ 0 & 0.001667 & 0 & 0 \\ 0 & 0 & 0.001667 & 0 \\ 0 & 0 & 0 & 0.0001667 \end{pmatrix} \tag{20}$$

However, the noise set into the simulator has the following R matrix that is 6 times bigger:

$$R = \begin{pmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{pmatrix} \tag{21}$$

As a result, if the early estimate of the covariance matrix by the Perception group is correct, it should be easier to get an accurate estimate of the mast state in real life than in the simulator.

There is no method to estimate the Q matrix before beginning tuning it. It will be considered diagonal in this study. Slowly varying wave frequency and amplitude of pitch and roll are expected. These variations can be considered as model error in the calculation of $\dot{p}$, $\dot{r}$ and ω. Therefore these three parameters should have a bigger coefficient.

$$Q = \begin{pmatrix} 0.005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.005 \end{pmatrix} \tag{22}$$

3.3.3 Predict the Future

The use of the prediction step several times in a row shall give the best estimate of the future state of the system as possible. This estimation also includes the state covariance matrix, which estimates how trustable is the estimation. The covariance increases exponentially. Therefore, long-term predictions are not viable. However short-term predictions can be reliable. As nothing can be controlled without a bit of latency, such estimations may be more valuable than the present estimation of the system.

4 Experimentation

In this chapter, we will put into application the different strategies developed in the Theory chapter to control the drone. We will first go through the methodology used by presenting the most important tools. After, we will show what performances the position and velocity control can give. Then we will detail the use of an LQR and its variants. After, we will be testing the EKF and creating an autonomous operation capable of autonomous module replacement in the simulator. Finally, we will check the simulation and improve the tuning based on real-life testing.

4.1 Description of the Most Important Tools

Good tests require good tools. Here will be described which are the most important ones used during this thesis.

4.1.1 ROS

First of all, ROS (Robot Operating System) is a widely used tool in the robotics field. It is a communication infrastructure that allows hardware abstraction. One can broadcast messages all over the system or call for specific services to and from nodes. One can use the exact same code to talk to the simulator or to the real drone. So ROS makes the development easier, quicker and safer.

As ROS quickly became the backbone of robotic systems, a bunch of packages has been built on top of it. For example, MAVROS is a library that translates messages understandable by an autopilot (such as Ardupilot or PX4) to ROS messages, and all the way around.

More details about ROS concepts can be found [17], but here is a quick overview. With ROS, one can create nodes that handle a process and communicate with other nodes. They do it by subscribing or publishing to topics, which could be seen as a radio channel. Each topic will talk about a specific thing and will only be able to send one kind of data. For example, it can be a topic about the robot position. Nodes can also execute or call for services. Services only include two nodes, the one calling the service and the one executing it. The caller can send arguments at the same time and will receive an acknowledgment. A typical service could be takeoff.

It is also possible to record bags. A bag is a file format that can store part or all messages going through the ROS network. The bags can then be played back which offers many good testing and debugging opportunities. For example, if one wants the drone to always follow the same path, one can record a bag of the reference set-points and play it again and again.

Last but not least, there is also a very convenient structure for running codes. One can create launch files that can start several nodes or scripts at the same time and include parameters. Important parameters such as whether we want to show debugging prints or tuning options are very convenient to have as they are all gathered at the same place and they can be tweaked without recompiling the code. They are also very easily accessible so that whoever not knowing the code can still use them.

4.1.2 Ascend Simulator

As mentioned in the Introduction, Ascend develops its own simulator. Other simulators already exist, but they have been judged not good enough for Ascend's application. It was especially limited when it came to physics and camera render. The Ascend simulator is based on Unity and allows everyone to test their work independently. For example, there is a topic publishing ground-truth and noisy data of the interaction point position of the blue plate if one wants to test the control strategy without running the perception nodes.

4.1.3 Matlab

Finally, one needs to be able to analyze the performance of each test. Matlab is a widely used tool for simulations and data analysis. During the previous semester project [5], A Matlab script has been developed to extract data from the tests and compare them. This script offers easy visual analysis opportunities by plotting matching data on the same graph. The script also executes automatic calculations to evaluate the performance of the drone. It estimates the latency and the average and maximal distance between two signals. The latency estimation is based on the maximum absolute cross-correlation of the two signals ([9]).

A description of how to use the Matlab scripts can be found [7]. It has been built to be very user-friendly. The only thing one has to do is to save the relevant data into files (a C++ library has been made for that) and read them with the script.

This script was especially convenient when doing some real-life testing outside. One could get graphs of the flight of the drone almost instantly after each test. This made testing more efficient as it allowed to check almost "on the fly" the performances of the drone and tune accordingly.

4.2 Control With Position and Velocity Set-Points

The LQR is being implemented to try to outperform Ardupilot's built-in position and velocity control. The default tuning parameters give somewhat bad results (average error of 0.40 and 0.114m on the x and y axes). But the accuracy can largely be improved by tuning the regulator for its application. Fig 14 has been obtained with the following gains:

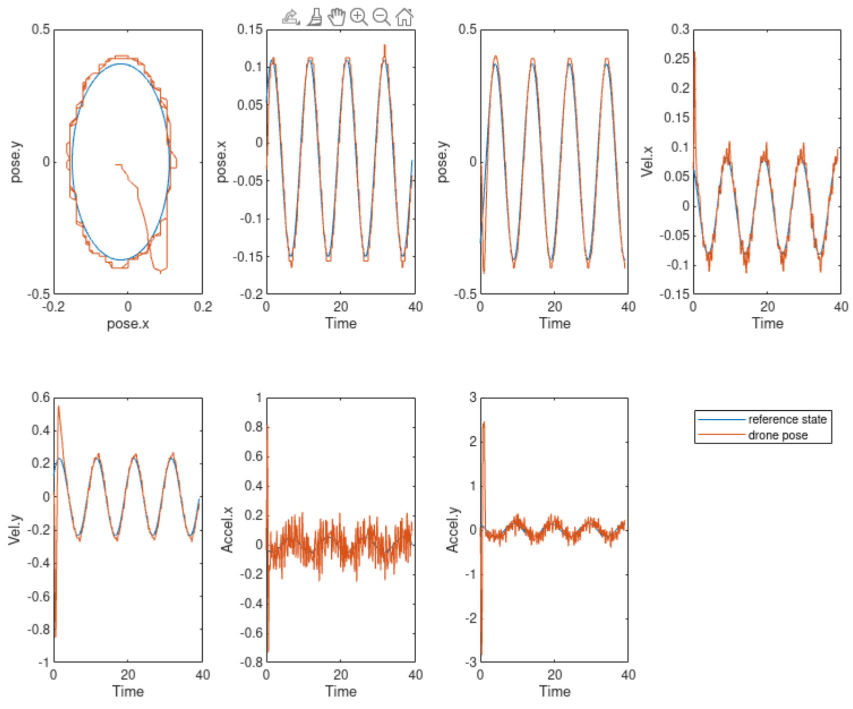


Figure 14: Graphs of the best simulation using position and velocity control

Gain	Default	Best
Position Proportional	1	3
Velocity Proportional	2	4.5
Velocity Derivative	0.5	1

Table 1: Table of the default versus the best gains for position and velocity control

The results showed table 2 are very promising. Notice that the position proportional gain is considered dangerous by Ardupilot. No improvement has been observed by tweaking the I gain of the velocity control.

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.000
Average absolute error (m)	0.006	0.011
Max absolute error (m)	0.020 m (at 18.5s)	0.035 (at 16.2s)

Table 2: Error analysis of the best position and velocity control simulation

Fig 15 and table 3 show how well the control reacts to wind. It will be

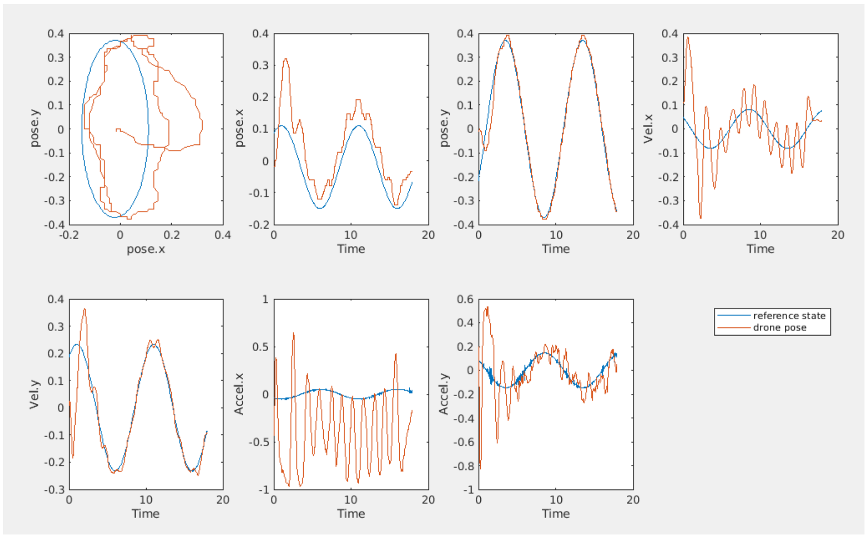


Figure 15: Graphs of the best simulation under windy conditions with position and velocity control

difficult to make it better with the LQR control.

Error projected along	the X axis	the Y axis
Average delay (s)	0.050	0.000
Average absolute error (m)	0.047	0.011
Max absolute error (m)	0.078 (at 16.7s)	0.039 (at 16.6s)

Table 3: Error analysis of the best position and velocity control simulation under windy conditions

4.3 Control With an LQR on Attitude

4.3.1 Implementation

In this section will be presented an implementation of an LQR controller for horizontal positioning with Ardupilot autopilot.

The LQR will use the position and velocity of the interaction point as an input and give an acceleration target. Ardupilot does not handle acceleration set-points by default. It is possible to tweak it, but it would reduce its reliability and take a lot of time. A safer and easier method is to control directly the leaning angle of the drone. The leaning angle of the drone is a bijective function of its acceleration:

$$\begin{aligned} pitch &= arctan(\frac{acc_p}{g}) \\ roll &= arctan(\frac{acc_r}{g}) \end{aligned} \tag{23}$$

Where:

g: is the gravity constant

Combining with the input-output equation from the LQR on both axis (eq. (2)), we get

$$\begin{aligned} pitch &= arctan(\frac{K_{LQR} * X_p}{g}) \\ roll &= arctan(\frac{K_{LQR} * X_r}{g}) \end{aligned} \tag{24}$$

where:

X_p and X_r: are the state vector along the pitch and the roll axes

During the following experiments, the LQR algorithm will be run at 20Hz. The tuning obtained only aims to point out a good idea of the performances of the controller. It should validate whether it is a good control solution in the context of the competition. In that case, more tuning can be done in real-life on the final drone.

4.3.2 Experiments

The LQR controller is being prototyped from a simple python script and is tested through Ascend's homemade simulator. The script creates a ROS node that guides the drone throughout its entire autonomous flight. First, the drone is asked to take off. Then trajectory following can start. The script subscribes to the state of the drone, runs the LQR algorithm, calculates the desired attitude, and publishes it. Internally, Ardupilot will receive the desired set-points and control the attitude of the drone toward this reference with its internal PIDs.

This script has shown being very convenient to spot mistakes and bugs in the implementation. The python language also showed the advantage of being a quick solution to test different tunings. It does not require recompiling between each try. Python has some drawbacks though. If there is an error in the code, it may take some time before occurring, which makes syntax debugging longer. Also, it is difficult to keep control of what we are exactly doing on a very high-level programming language, and some errors may be misinterpreted (such as spelling issues).

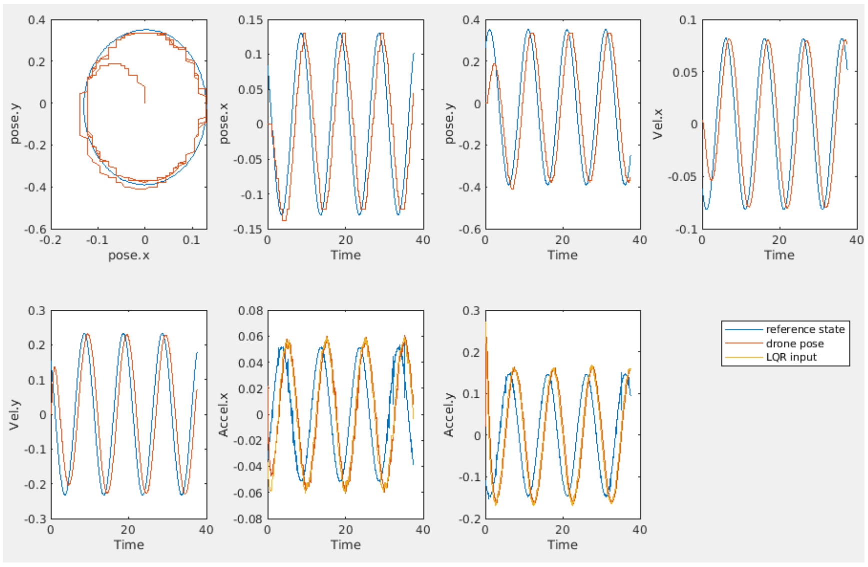


Figure 16: Graphs of the first LQR simulation results

The data from the simulations are saved and analyzed with the Matlab script explained earlier. The following simulations are based on perfect positioning data for both the drone and the reference, and no wind is being simulated. These choices have been made in order to create a base controller that is simple to implement, test and tune.

For the first test, the gain matrix obtained eq. (8) is being used. The result from the Matlab script are summed up fig. 16 and table 4.

Error projected along	the X axis	the Y axis
Average delay (s)	0.849	0.799
Average absolute error (m)	0.047	0.131
Max absolute error (m)	0.082	0.226

Table 4: Error analysis of the first LQR simulation

This first result shows pretty big errors and delays. The best results on the simulator have been obtained with $K_{LQR} = (7.5 \ \ 5)$ fig. 17 and table 5.

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.000
Average absolute error (m)	0.008	0.020
Max absolute error (m)	0.028 (at 13.2s)	0.048 (at 10.7s)

Table 5: Error analysis of the best LQR simulation

In this simulation, one can observe that the acceleration input oscillates

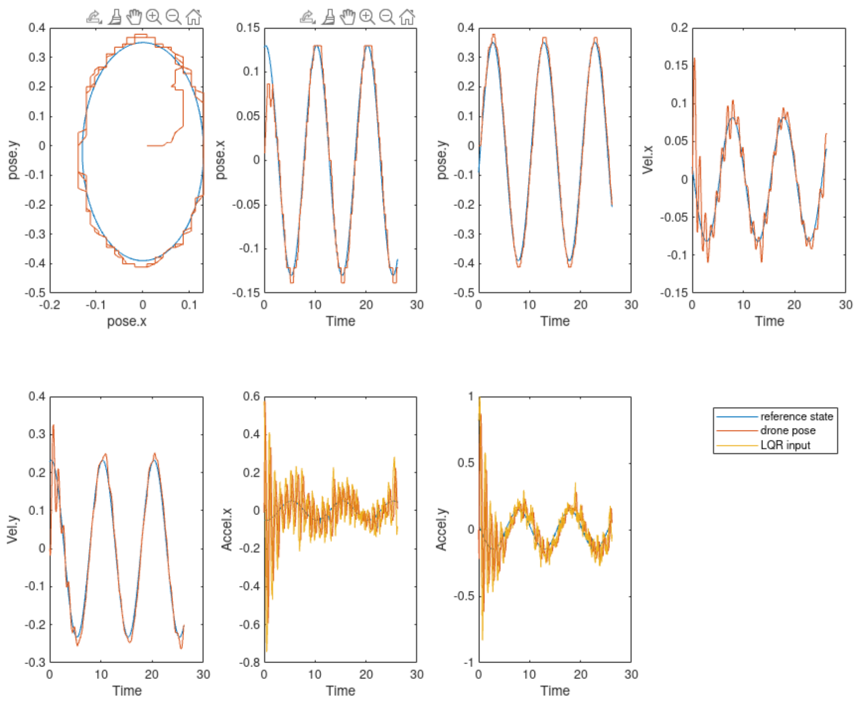


Figure 17: Graphs of the best simulation with LQR control

around mast acceleration, which is reflected in the velocity of the drone on the x axis. In the simulator, one could observe the drone oscillating. This is due to the high gains. It creates some instability in the drone and makes the FaceHugger oscillating even though the center of the drone hardly moves.

In these simulations, the drone is following identical sinusoidal cycles again and again. In order to better understand the results and how they can be improved, a new graph presenting the position error over each half sinusoid has been drawn. The superposition of the results over each half cycle is shown fig. 18. It can be observed that the error is pretty constant throughout the sinusoid. This shows that the controller is very consistent whatever is the velocity or acceleration of the reference. This also means that there is no weak point of the controller and that the drone can stay close to the mast whenever during the whole oscillations without adding risk.

4.3.3 Instability Test

As mention just before, the gains used on the simulations to get the best results are very high. This increases the risk of drone instability and it must be tested. Bigger gains are going to be simulated to try to create instability and analyze how the drone behaves.

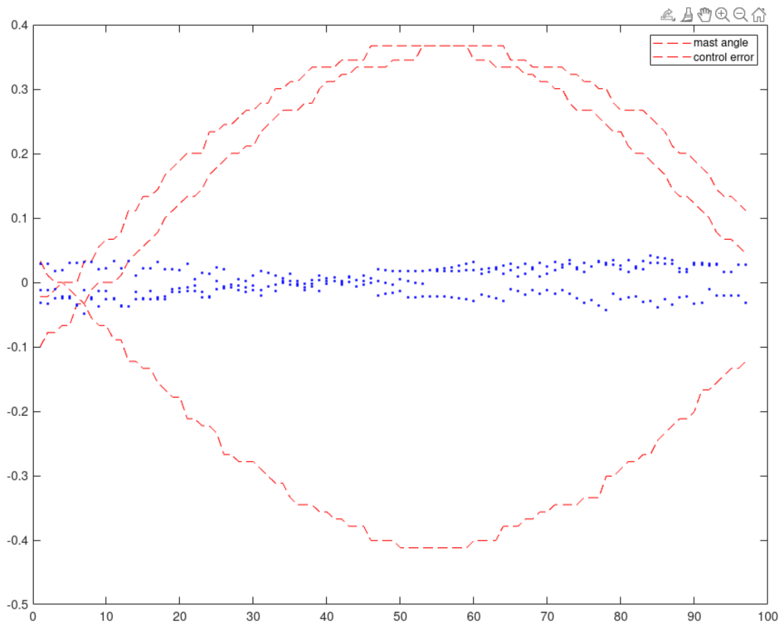


Figure 18: Position error of the best LQR control for each half sinusoid of the mast on the Y axis

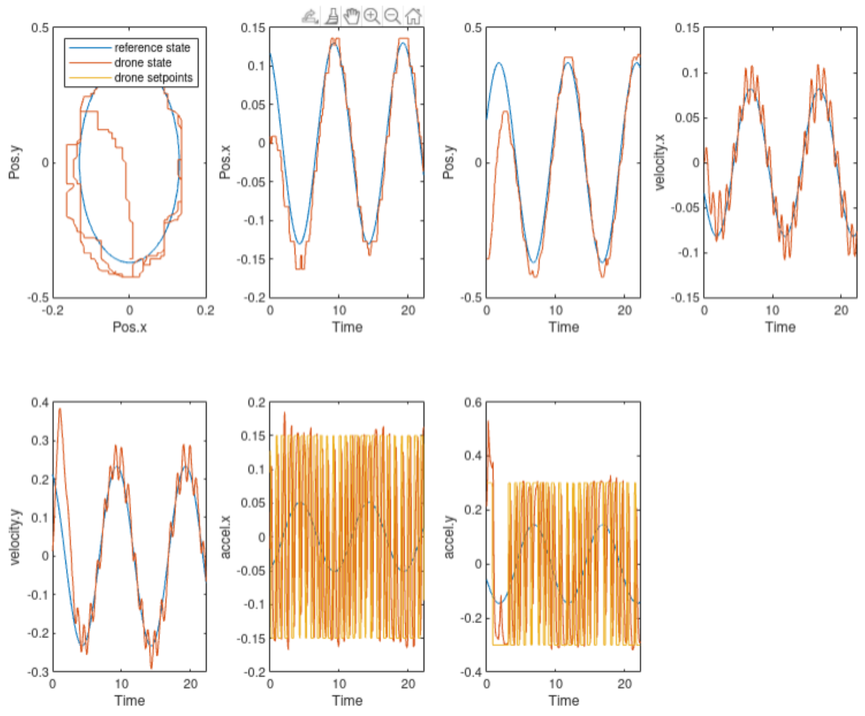


Figure 19: LQR control with very high gains

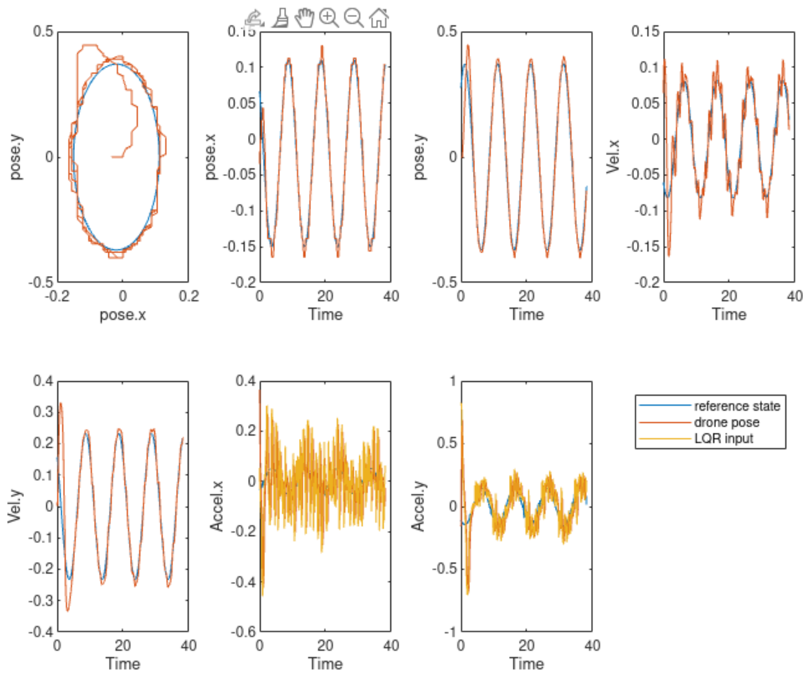


Figure 20: Best results with squared root input on LQR control

In fig. 19, one can see that the acceleration set-points line (light orange) looks like a PWM signal. This is due to the saturation that constrains the input. That allows the drone to remain pretty stable despite the use of too big gains. It shows that this saturation is very important to tune, especially when our best tuning has big gains. Its tuning is a trade-off between the capacity of the drone to follow fast moving references or face disturbances and ensuring stability and smooth behavior.

This simulation was done with gains 4 times bigger than in the previous simulation. Even higher gains show the same results and confirm this conclusion.

4.3.4 LQR Control from Non-Linear Error

As explained in the theory chapter section 3.2.3, this thesis wanted to explore the possibility of having a non-linear error input. The signed square root of the difference between the reference and drone state is taken before applying the LQR gain.

The best result has be found after lowering the gains to $K_{LQR} = (1.0 \;\; 0.75)$ from eq. (8) and can be seen fig. 20 and table 6

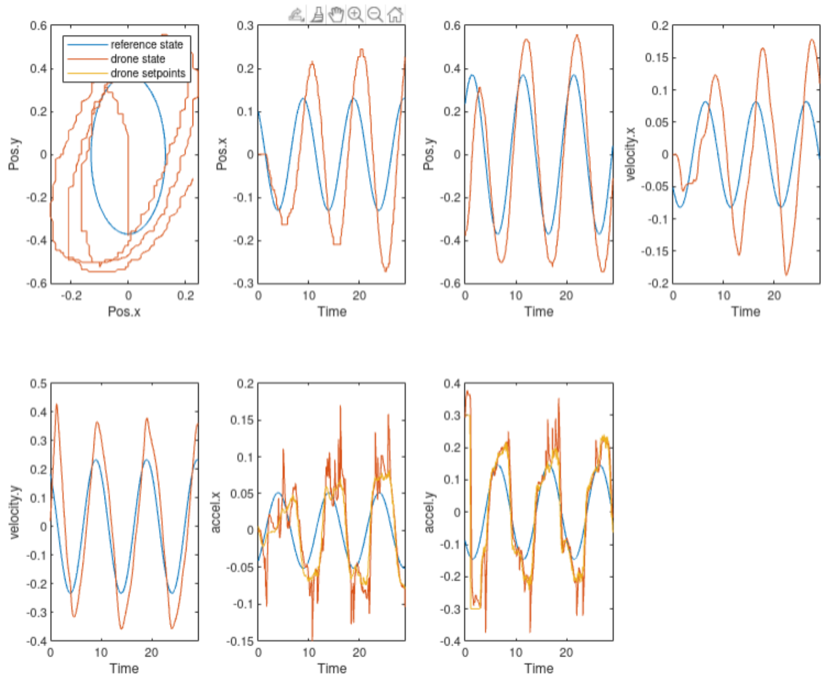


Figure 21: Graphs of LQR control with signed squared input

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.000
Average absolute error (m)	0.007	0.018
Max absolute error (m)	0.026 (at 19.2s)	0.044 (at 29.0s)

Table 6: Error analysis of LQR control simulation with squared root input

The results are not significantly better than the best results with the linear input LQR. Even if they tend to be a bit better and the drone may be slightly less jittery, it will be preferred to keep a linear input for the rest of the tests. Still, if more accuracy is needed, the use of the squared root will be looked at again.

Similar tests have been run taking the signed squared error as an input. As one can see fig. 21, the results have been worse.

4.3.5 LQR with Feed-forward

In the theory chapter, we expected to be able to reduce the phase shift between the drone movement and its reference by adding a feed-forward on the acceleration of the interaction point. Even though there is barely any phase shift, feed-forward showed to improve the results fig. 22. It is adding valuable information to the system and helps to prevent position overshoot at

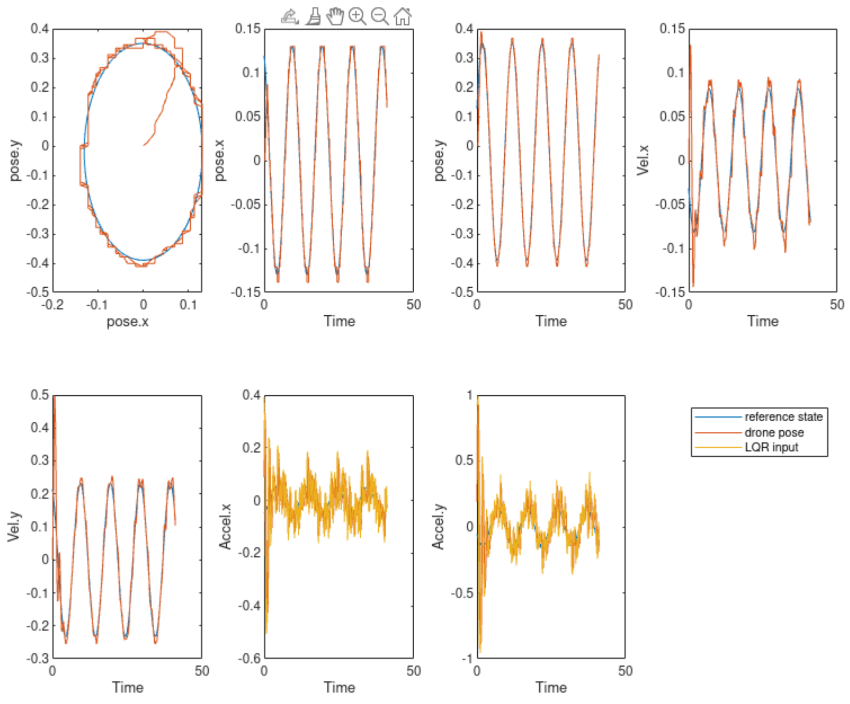


Figure 22: Best result with a feed-forward on the interaction point acceleration

the end of the sinusoids. Here, a feed-forward of 0.5. and $K_{LQR} = (7.5\ \ 4.0)$ have been used. This brings the LQR control at the same level as the built-in position and velocity control from Ardupilot (table 7). Notice that it has not been possible to get better results by using the signed squared root error with feed-forward.

Error projected along	the X axis	the Y axis
Average delay (s)	0.050	0.000
Average absolute error (m)	0.006	0.011
Max absolute error (m)	0.025 (at 32.2s)	0.039 (at 19.7s)

Table 7: Error analysis of the best simulation with feed-forward

4.3.6 Using Future Set-points

In the Theory chapter, section 3.3.3, it was proposed to try using an estimation of future set-points to improve any control strategy. However, both the LQR or the position and velocity control or in sync with the reference, or even a little bit ahead. Therefore, it is unnecessary and worse to use the

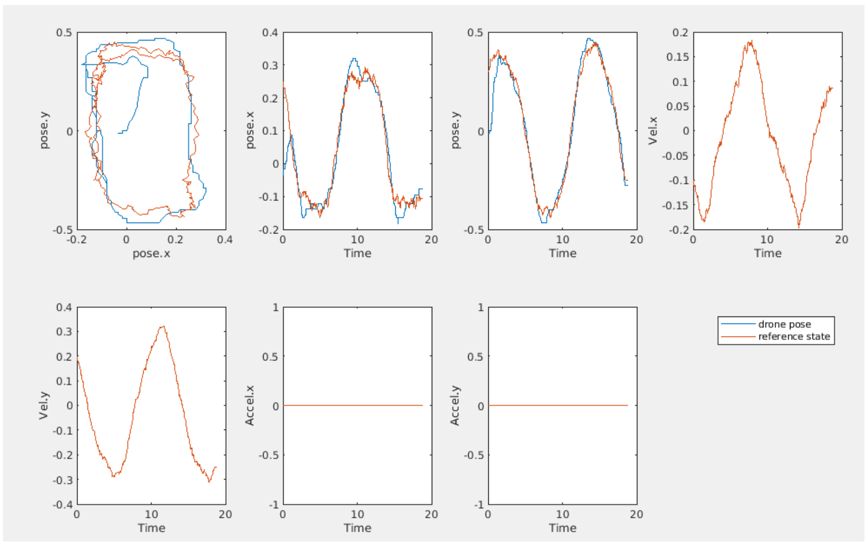


Figure 23: Control of the drone with LQR using estimated future set-points (0.05sec ahead)

estimated future set-points. Fig 23 shows one test with LQR using estimated set-points 0.05 seconds in the future. Here we can clearly see that the drone (in blue) is too much ahead of the mast. And the maximum error is obtained when the mast has its maximum velocity.

4.3.7 LQR Control Under Simulated Wind

Now that a good configuration for LQR has just been found, it is time to try it under windy conditions (fig. 24). The usual error analysis if shown table 8. The drone is jittering and the accuracy of the center of the drone is much worse. The oscillations are due to the necessary big gains to keep the error low and the randomness of the wind.

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.000
Average absolute error (m)	0.0030	0.038
Max absolute error (m)	0.087 (at 17.0s)	0.139 (at 10.4s)

Table 8: Error analysis of a simulation with the best LQR control configuration and simulated wind

4.4 Extended Kalman Filter

The Kalman Filter aims to find the best estimate of the interaction point state as possible from the Perception's measurements. To remind, the in-

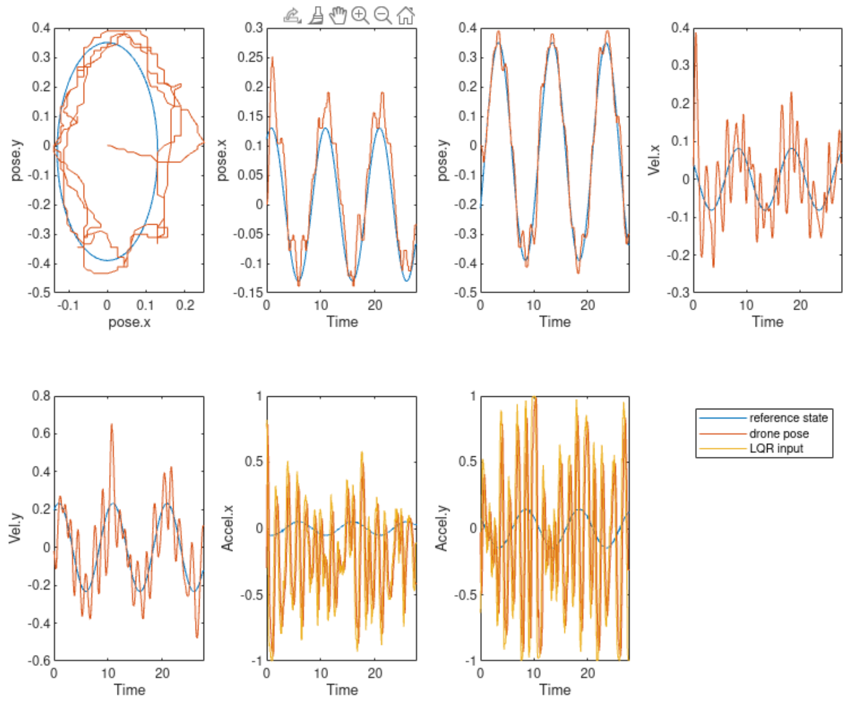


Figure 24: Control of the drone with simulated wind using LQR control

teraction point has been defined as the middle top of the blue plate, where the FaceHugger should be hooked. Matlab is a good tool to experiment and prototype Kalman Filters. One can plot results to get visual representations. On can also keep track of each state for debugging. In addition, there is no need to compile the code, which makes iterative testing quick and easy. This is especially useful to tune the filter.

4.4.1 Testing Procedure

The system developed in the Theory chapter is being built and validated based on artificial mast position and measurement signals. The rules describe that the movement of the mast will mostly be sinusoidal. So two ground-truth sinusoids of different amplitude and phase have been implemented for the pitch and roll axes. Matlab's random function has been used to transform the position of the mast into measurement by adding a white noise with adjustable standard deviation. Later, Matlab will take ROS bags with simulated perception measurements from the simulator. The Kalman filter could have been tested directly from the ROS bags, but it has been convenient to have full control of the measurements and be able to change the parameters easily.

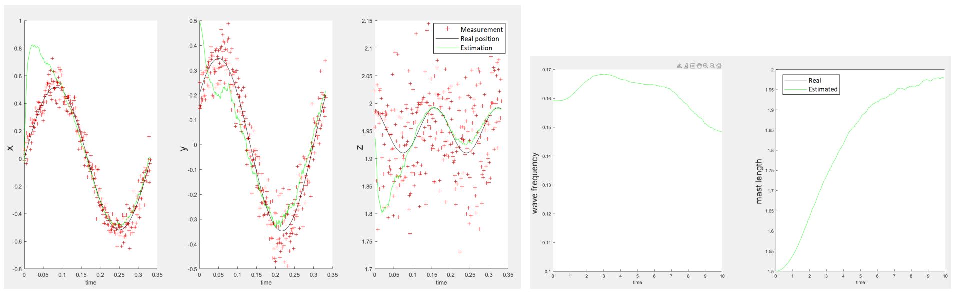


Figure 26: EKF visualization on Matlab from unknown initial state

4.4.2 Experimentation

The results after the first tune on Matlab can be found fig. 25. In this simulation, the initial state is perfectly known. In this context, the filter is very efficient. This is especially true for the z position which is adjusted by the x and y position of the interaction point.

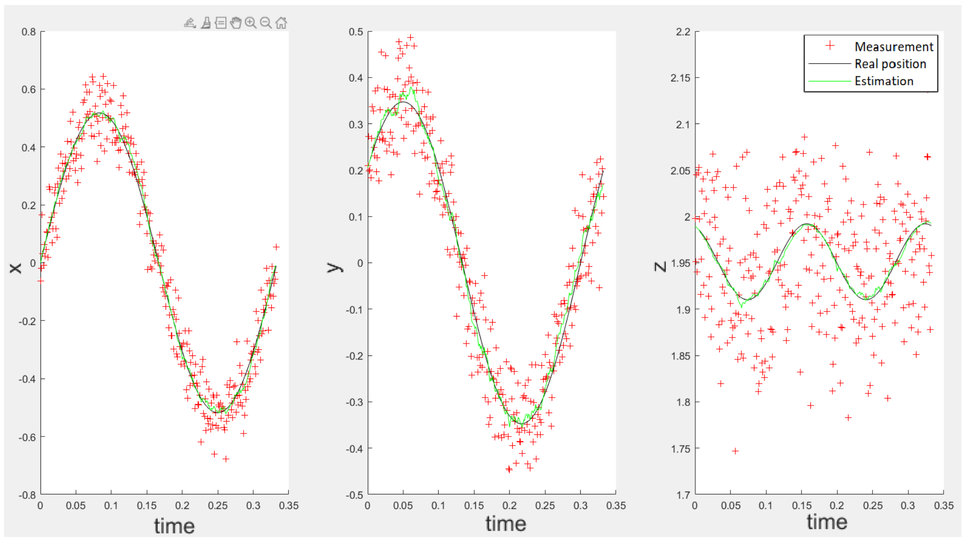


Figure 25: EKF visualisation on Matlab

Naturally, the results are not as good when the initial state is unknown (see fig. 26). In this second case, the filter begins with very bad estimations but reaches reasonable results in about 3 seconds. Longer simulations show that after about 10 seconds the filter reaches its steady-state and the output becomes almost as good as the first example. This underlines that the EKF tends to converge.

This simulation also shows how much the Kalman filter is sensitive to the initial state. Still, tuning can be adapted to obtain a more or less quick transition at the expense of the "smoothing" effect.

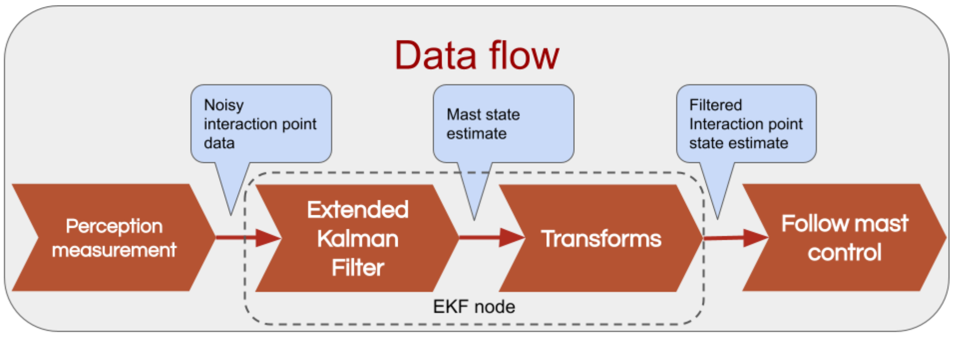


Figure 27: Schema of the data flow in between the different nodes involved in the control of the drone

4.4.3 Integration

The filter is implemented into Ascend's pipeline as a new node that takes the noisy data as input topics and outputs the estimated state of the mast (see fig. 27). The module replacement operation node can then use the smoothed data from the Kalman filter as the interaction point state reference.

It is possible to automatically translate a Matlab code toward C++. Unfortunately, the generated code is not really usable in practice. Its quality is bad so that it is very difficult to understand what is going on, which makes debugging almost impossible. In the spirit of creating code that is usable for several years in a row, it has been decided to write a proper code in C++. It is based on an EKF library originally developed for Arduino [12]. Arduino is a board with low computational power, so this library is optimized to run in a short time. This library did not work out of the box, and the fixed version can be found on GitHub [6].

The EKF estimates the state of the mast which includes pitch, roll, their derivative, the length of the mast and the frequency of the oscillations. From that, it is possible to estimate the state of the interaction point, which is the data needed by the control node. It is basically the h function (eq. (18)) for the position of the interaction point. The velocity and acceleration are derived from the spherical coordinates as follow:

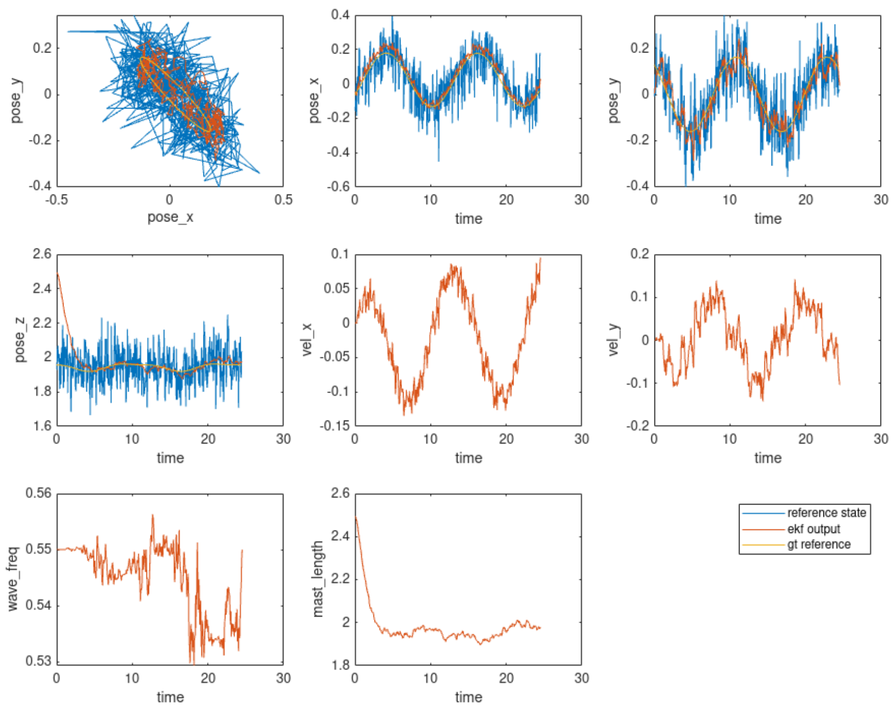


Figure 28: Interaction point state estimation from the EKF

$$
\begin{aligned}
\dot{x} &= L_{mast} \times \dot{p} \times cos(p) \\
\dot{y} &= L_{mast} \times \dot{r} \times cos(r) \\
\ddot{x} &= L_{mast} \times (\dot{p}^2 \times sin(p) + \ddot{p} \times cos(p)) \\
\ddot{y} &= L_{mast} \times (\dot{r}^2 \times sin(r) + \ddot{r} \times cos(r))
\end{aligned}
\tag{25}
$$

In the real-time simulation fig. 28, the initial state is set to 0 for the mast angles and their rate. Also, neither the mast length nor the frequency of the movement is initialized to the ground-truth value. The y axis is clearly noisier than the x axis. This is due to the fact that Perception cannot measure the roll of the mast. As a result, the z axis, which is mostly estimated from the x and y estimations is somewhat noisy as well. The velocity estimates are quite noisy as well. More tuning could be done to try to smooth it more, but one should first consider how does the control algorithm react to this signal. Notice that the velocity estimate is still much better than it would be from a simple derivation (as it was before implementing the Kalman Filter).

4.5 Combination of the Work into a Module Replacement Operation

It is time to come back to the main goal of the thesis, creating an autonomous operation capable of an autonomous module replacement. The logic of the operation will be studied to create a Final State Machine (FSM). Then, the EKF will be combined with the LQR control, and a fix to handle altitude control will be proposed. Finally, smooth movement relative to the mast will be studied.

4.5.1 A State Machine Based Operation

The LQR controller will be implemented into the pipeline used by Ascend to control the drone. But as the LQR and the position and velocity control need the same input, the same code can be adapted for both control strategies.

The pipeline is written in C++ and the regulator must be integrated into the operation that will swap the communication module of the mast. This is a final state machine (FSM) that ensures accurate and safe movements relative to the mast (see fig. 29). The FSM decides where should the drone be compared to the interaction point, and the controller makes it happen. This operation will be called by the AI node as shown fig. 1 when the drone will have found the mast position and orientation.

Code simplicity and ease of tuning have been some of the main concerns while implementing the new controller. The controller must be usable by anybody in the control group of ascend, and hopefully for the following years.

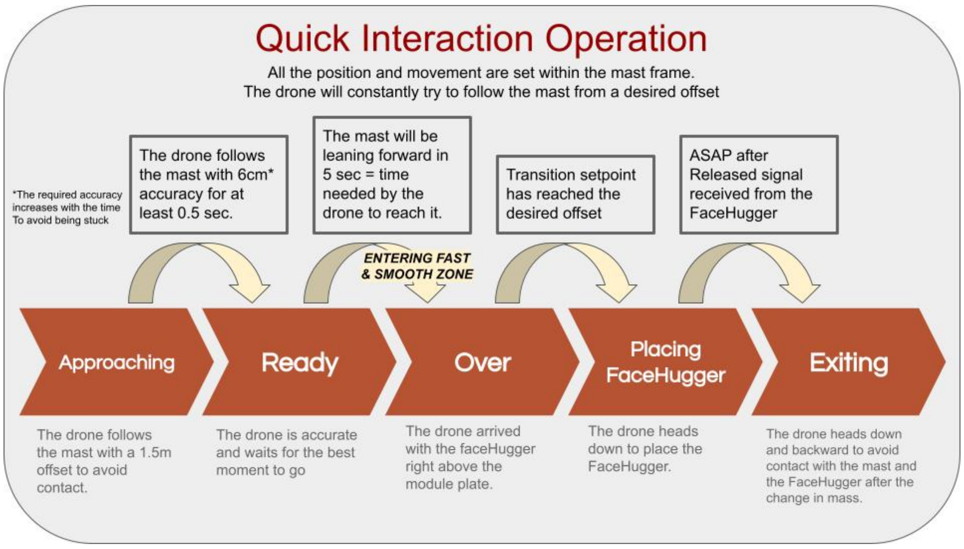

Quick Interaction Operation
All the position and movement are set within the mast frame.
The drone will constantly try to follow the mast from a desired offset
*The required accuracy increases with the time To avoid being stuck
The drone follows the mast with 6cm* accuracy for at least 0.5 sec.
The mast will be leaning forward in 5 sec = time needed by the drone to reach it.
ENTERING FAST & SMOOTH ZONE
Transition setpoint has reached the desired offset
ASAP after Released signal received from the FaceHugger
Approaching
Ready
Over
Placing FaceHugger
Exiting
The drone follows the mast with a 1.5m offset to avoid contact.
The drone is accurate and waits for the best moment to go
The drone arrived with the faceHugger right above the module plate.
The drone heads down to place the FaceHugger.
The drone heads down and backward to avoid contact with the mast and the FaceHugger after the change in mass.

Figure 29: Schema of the module replacement operation's state machine

Collisions between the drone and the mast should be avoided as much as possible. They may lead to drone instability, loss of accuracy in the positioning and could even be dangerous. Therefore, the following of the mast will begin with a 1.5m offset. This avoids potential collisions with the mast while the drone is not accurate.

Once the desired accuracy is reached, the drone goes to the "Ready" state where it waits for the best moment to move toward the mast. The drone should reach the mast when the latter leans toward the drone as the placement of the FaceHugger is easier and more likely to work well. The way the drone will move toward the mast is always the same and therefore, the drone can estimate how much time it will take. Also, by estimating the oscillation period of the mast, the drone can calculate when it should go.

On-time, the drone can move to the "Over" state. There, the desired position of the drone is such that the hook of the FaceHugger is just over the interaction point of the blue plate. As soon as the drone reaches its target, it switches to the "Placing FaceHugger" state. The drone just moves downward in the mast frame to set the FaceHugger. Whether the latter has been placed or not, the drone will quickly exit the mast to avoid being close to the mast for too much time after the potential interaction.

The FaceHugger is equipped with sensors allowing it to know if it has been hanged correctly. It will be detached only if the placement is correct. Else, the drone will redo the operation starting from the "Approaching" state. Once the placement has been successful, the operation is marked as finished and the mission can continue.

The FSM has been designed so that the drone prioritized a short interaction time over a very accurate movement. This is the reason why the drone does not wait for the drone to be as accurate as possible during the "Over" state. Indeed, it has been considered that the small potential gain of accuracy by hovering for a long time close to the mast did not worth the increased risk of collision. This also allows choosing when the FaceHugger will be released compared to the leaning angle of the mast.

4.5.2 LQR from EKF Set-points

If the EKF and the control nodes work well independently, it does not mean that they will work well together. The different tunings may need to be tweaked accordingly.

Fig 30 shows how does the drone behave when flying with the EKF estimate as an input of the LQR (the yellow line marked as "reference state"). We can see from the velocity graphs that the drone was jittering a bit. The noise in the reference is getting amplified by the LQR high gains.

One may not want to reduce the gain of the LQR not to limit its accuracy. The model of the mast should be good enough so that it can be trusted more by the Kalman filter. The R noise matrix will then be increased resulting in

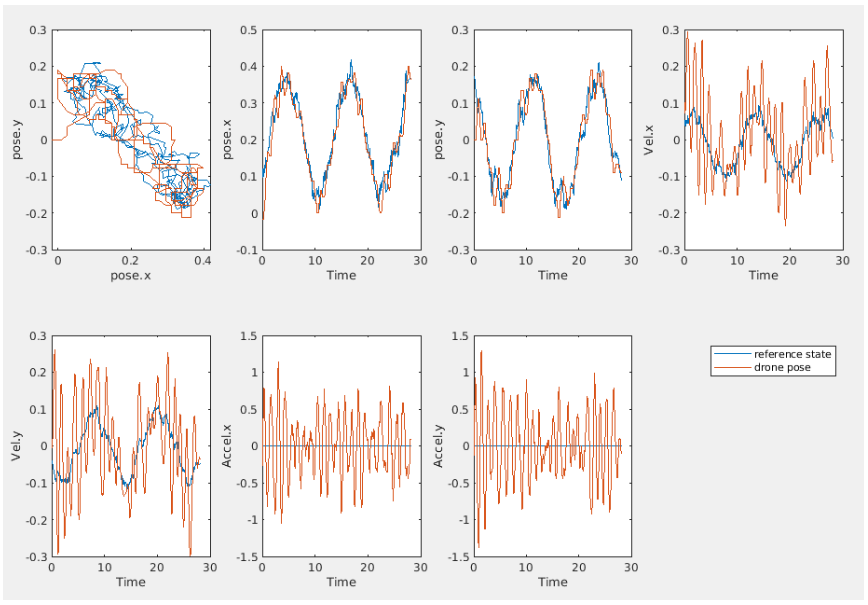


Figure 30: Graphs of the drone flying from a noisy signal filtered with an EKF

a smoother estimation as shown fig. 31. The improvement is not significant, and the error becomes quite big: about 13cm on average. This error mostly comes from the EKF being inaccurate; the error between the LQR reference and the drone position averages 2cms. A good trade-off between filtering and accuracy will need to be found once the real perception measurement will be obtained. Also, the gain of the LQR may have to be tweaked to adapt to this new shape of input.

4.5.3 Altitude Control

Another issue mentioned in the theory chapter is the control of the altitude of the drone. This can not easily be done through attitude control. However, it is possible to set the attitude controller such that it tries to always keep the same altitude. This is going to be tweaked by using a normal position and velocity set-point which includes the altitude at which we want the drone. This set-point is sent to Ardupilot just before the usual attitude set-point. In this way, Ardupilot internally updates the altitude reference without causing any disturbance with the external regulator. The results showed fig. 32 have been run with an altitude P gain of 6. This is higher than what Ardupilot advises, but for small movements like the mast oscillations, it is completely fine. However, a big overshoot can be observed for step altitude inputs. It is especially scary to see the drone falling down when the altitude set-points is suddenly decreased.

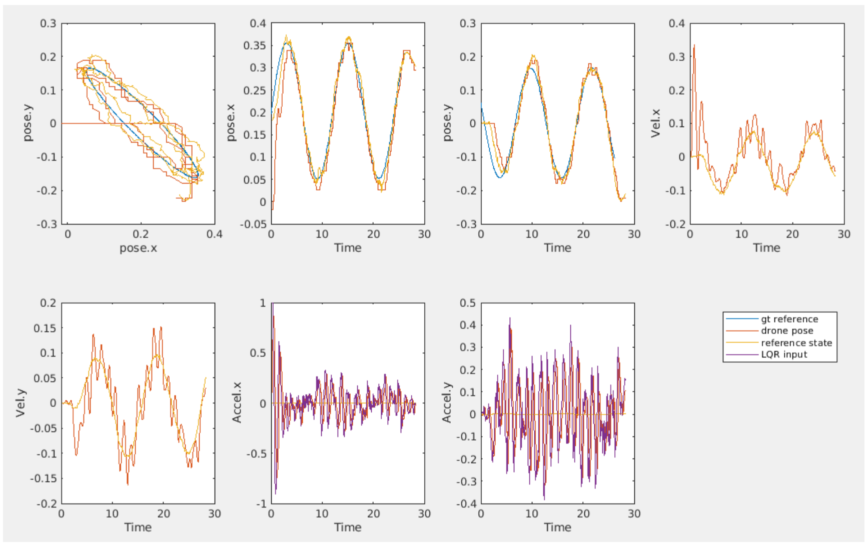


Figure 31: Graphs of the drone flying from a noisy signal filtered with an EKF with smooth tuning

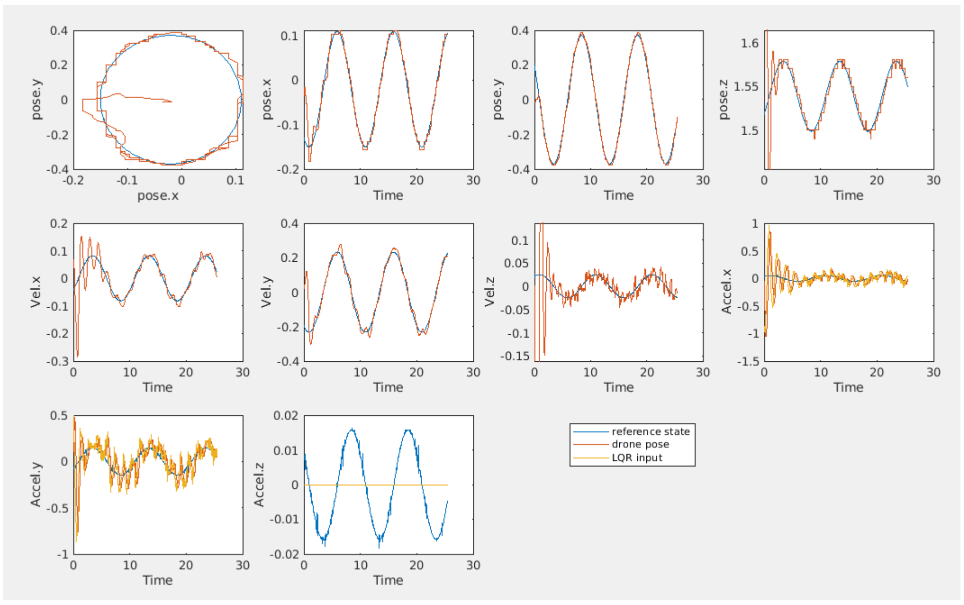


Figure 32: Graphs of the drone flying with both LQR and altitude control

Here the error is at a maximum of 10mm. This is theoretically good enough, but the simulation considers perfect data without disturbances. This has not been considered as the most challenging aspect of the operation and will therefore be outside of the scope of this thesis.

4.5.4 Smooth Movement Relative to the Mast

When the FSM will go from one state to another, while approaching the mast, for example, the desired offset will suddenly change. The drone will try to reach the new set-points as fast as possible and lose the steady accurate state it just obtained. The solution proposed by this thesis is based on the four following variable:

- The state of the drone (position, velocity and acceleration)
- The state of the interaction point on the mast (position, velocity and acceleration)
- The desired offset distance to the mast (position vector in the mast frame)
- The current offset state (position, velocity and acceleration)

The last variable is the key to create a smooth transition when the offset suddenly changes. It is the offset that the drone will always try to follow, and which aims to become the desired offset. At all times, the drone position setpoint is the sum of the interaction point position and the current offset. When the desired offset changes, the current offset will follow the first one in a smooth manner by

- Keeping a constant small acceleration toward the desired offset.
- Constraining its velocity (relative to the mast).
- Slowly "braking" or decelerating to reach the desired offset at a null relative velocity.

The resulting setpoint will keep a smooth path and the drone can follow it while staying in a steady state.

The current offset state includes velocity and acceleration at which it moves in the mast frame. They will be added to the interaction point state for the input of the LQR. If this would not be done, the drone would see this movement as a perturbation and its accuracy would decrease.

The result can be found fig. 33. In this example, it approximately takes 12 seconds to move by 50cm. The maximum velocity was 0.05m/s and maximum acceleration 0.03 m/s^2. As the transition state is known, the drone can follow a smooth transition without any significant loss of accuracy.

This can be verified by simulating a faster transition as in fig. 34. Here, the maximum speed is 0.50m/s and the maximum acceleration is 0.15m/s^2. In this case, the transition distance has been multiplied by three but only took less than 5 seconds. The drone can still follow the reference pretty accurately.

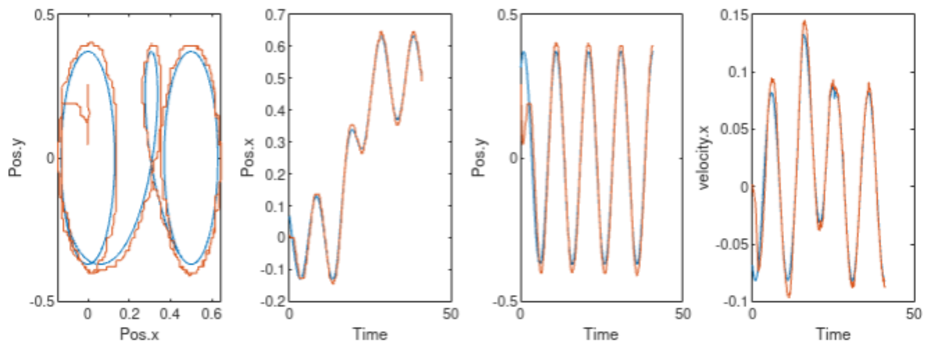


Figure 33: Graphs of the drone doing a smooth transition (In blue are the smooth set-points and in orange the drone position)

One can also observe from the velocity graph that the maximum transition speed has just been reached. Since it is not expected to travel more than 1.5m with this smooth transition, we can conclude that the maximum transition speed is not a limiting factor.

However, pushing the acceleration of the transition even farther is not suitable (see fig. 35). As mentioned during the instability test of the LQR, the drone is constrained to a max acceleration of 0.15m/s^2 on the x axis. Therefore, the drone could not handle such an acceleration and got delayed. If moving fast was a requirement, it would be possible to increase the maximum acceleration of the drone. But that would reduce the LQR performance, and this is the reason why it is not done here.

4.6 Real Life Testing

The simulations are getting validated in real life. The test drone is being used with two RTK GPSs (Real Time Kinematic GPS) mounted on it. They give a very accurate position and yaw estimation. For safety reasons, flights have never been handled alone. At least a pilot, who was handling a remote control (RC) (in case of unexpected behaviors), and an operator, who was monitoring the ground station and launching autonomous programs into the drone, were needed.

The testing procedure was similar at every test. First, one must test that the drone can fly properly in a non-autonomous mode using GPS data. This was done by flying in "Loiter" mode. Then, safety features such as the motor kill-switch are checked. Then, one can run the code to guide the drone autonomously. For safety reasons again, the drone was not allowed to arm itself. It must be done by the RC. When the drone is armed, it can start to fly by itself.

The control of the drone is being tested the same way in real life as

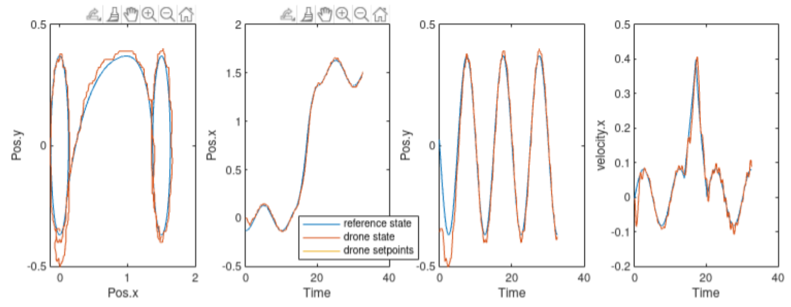
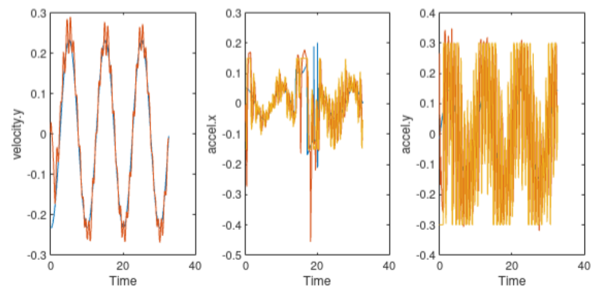



Figure 34: Graphs of the drone doing a fast smooth transition

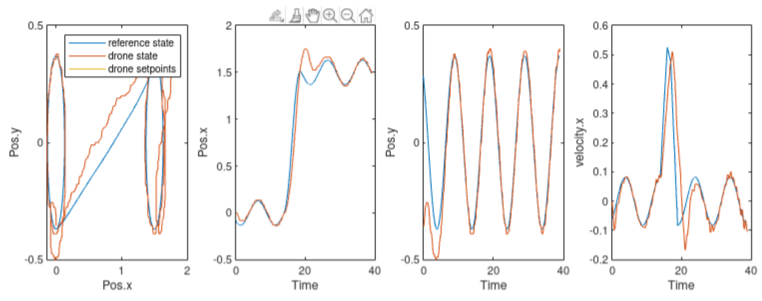
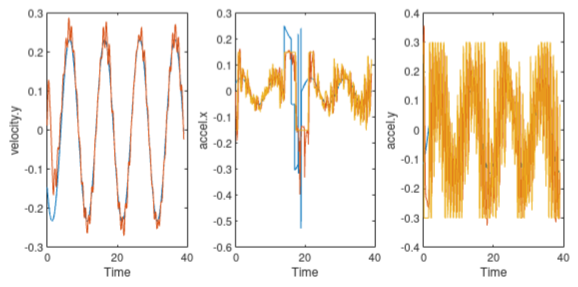



Figure 35: Graphs of the drone doing a too fast transition

it was in the simulator. With the exception of the tuning parameters, the exact same code is being used. But as the simulator is not being launched, the reference position is being sent in real-time through a ROS bag of the simulator. Of course, the drone is not following a physical mast for those tests, it would be too dangerous at this stage.

In the same manner as the simulations, the drone writes control data into files that can be analyzed by a homemade Matlab script. However, the drone does not have Matlab nor a screen to analyze the data on the fly. Therefore, a shell script has been created to transfer the files into the monitoring laptop, and save it into a specific repository. Matlab on the laptop could see the files and analyze them straight away. As a result, it took less than 30sec to get the usual graphs after that the drone landed. That made testing very efficient, especially for recursive tuning. The people who had to join for safety reasons appreciated it.

4.6.1 Real Life Testing Difficulties

The first difficulty is to pass all the safety checks and be able to arm the drone. The error codes are not very explicit and that makes them difficult to debug. In Ascend, there is a pretty good experience with the potential errors that can occur and how to try to fix them. However, this knowledge was not written into documentations that anybody can easily access. Therefore, all errors that occurred during the testing process have been written into a single page that is accessible by any Ascend member. Access to this page can be asked to the author. The aim is to gather all the most likely errors in the long run and make flying much less cumbersome. In this thesis, most of the errors that occurred were link with the two RTK GPS that are added to the drone.

Once the drone was flying, it has been difficult to get the measured drone acceleration coherent with the LQR input. This was due to both software and hardware issues. Code-wise, some frame rotations were poorly handled, but they did not have any impact while the drone was flying at null yaw. Hardware-wise the Pixhawk was mounted on a plate that was too softly mounted on the drone. As a result, it was oscillating a lot when the drone was flying, and so was the drone acceleration estimation. So the control of the drone was still working as it was not using this estimation, but the results did not make sense as the estimated acceleration was not following the acceleration set-points set by the LQR. This last point was emphasized by the most important issue; the internal attitude control of the drone was poorly tuned which made the drone slow to reach the desired attitude set-points. The LQR relies on a properly tuned drone at a lower level. The tests held with the bad internal tunes showed how important it is. As soon as these issues have been fixed, the real-life testing got very similar to the simulator ones.

Accounting for wind requires changes in some of the settings set by simulator testing. As explained section 4.3.3, a maximum leaning angle had been set in order to limit the input that the LQR may give to the drone to avoid highly unstable behavior. It had been optimized in the simulator in a context without wind resulting in a 4° constrain. This is equivalent to an acceleration of 0.69m/s^2. However, testing the drone in windy conditions shows that it may also limit the efficiency of the control. The drone may have to put a lot of effort to face strong winds and avoid drifting away.

Some tests under winds of 7 to 12 m/s led to a maximum acceleration set-points of about 1 m/s^2. As a comparison, the wind simulated in the simulator required the drone to add an acceleration offset of about 0.5m/s^2. It is not expected to have very strong wind during the competition, so the maximum acceleration has been set to 1.2m/s^2.

The maximum acceleration of the drone used to be constrained through an Ardupilot parameter. However, this also limited the drone's max angle during recovery loiter or land mode. This made the recovery dangerous and almost impossible during windy conditions, especially when the pilot was not expecting it. Therefore, The maximum angle parameter is set back to normal (15°), and the maximum acceleration is being constrained directly in the code through the maximum LQR input. The difference is important to keep in mind as the acceleration input is not exactly the same as the drone acceleration.

The control in position and velocity also created some issues. Some pseudo-random offset came between the drone position and the drone position target. Since this offset could reach more than 1m under no wind, it was not due to a lack of accuracy of the control. It seems like it is due to a miss-match in the frames used. Indeed, the log from the flight controller did not plot the same position target as the one we sent. If the issue is suspected to be either in the reading of the drone position or in the sending of position and velocity set-points, the exact source of the issue has not been found yet. However, it has still been possible to extract interesting data from flight using position and velocity control. The data is being taken directly from the log files using the website https://plot.dron.ee/. Unfortunately, this website does not give statistics of the flight as the Matlab script does.

4.6.2 GPS Accuracy Test

The accuracy of the localization system is being tested fig. 36. This aims to see how viable it is and how inaccurate it may introduce in future tests.

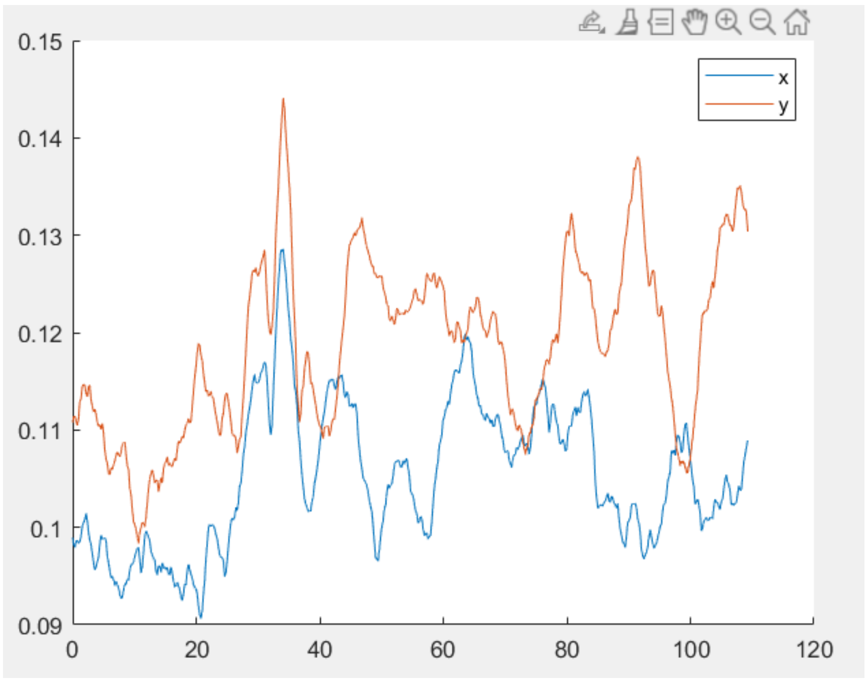


Figure 36: RTK GPS measurement while the drone is standing still on the ground

We can observe a maximum amplitude over the 110sec test of approximately 40mm for the x and y axes. It results in a standard deviation of 7.4 and 8.9mm respectively.

4.6.3 Hovering Test

In order to break down the experiments, some tests have been run asking the drone to keep the same position with position and velocity or LQR control. These tests have been run under a wind similar to the one on the simulations, but probably a bit more during the LQR control.

With position and velocity control, as explained in section 4.6.1 the drone was not receiving set-points in the same frame as the drone position frame received by the companion computer. As a result, it was an offset in the data printed by Matlab. Therefore, the data from the Pixhawk log file will be shown instead fig. 37. This best result has been obtained with a position P gain of 3, and velocity P, I, and D gains of respectively 4.5, 1 and 1.

With LQR control fig. 38, the best result has been obtained for Kp = 7.5 and Kv = 5. A sum up of the error in position of both tests can be found table 9.

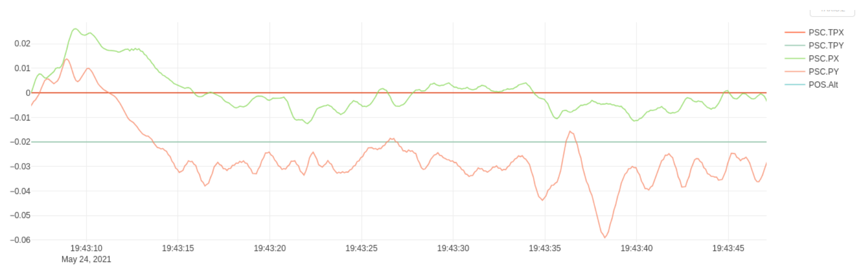


Figure 37: Graphs of the best hovering performance using position and velocity control

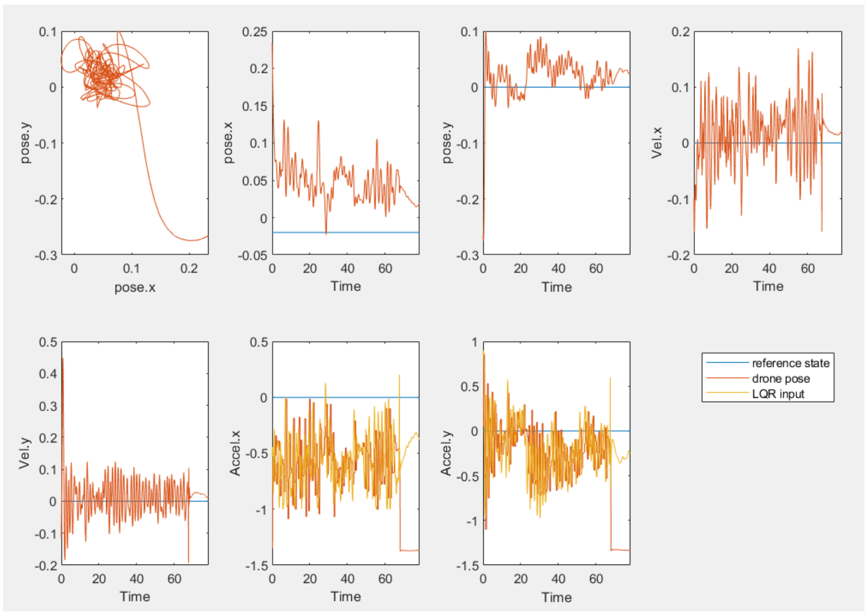


Figure 38: Graphs of the best hovering performance using LQR control

Control strategy	Position and Velocity		LQR	
Error projected along axis	X	Y	X	Y
Average absolute error (m)	*0.05*	*0.010*	0.0011	0.009
Max absolute error (m)	0.027	0.040	0.048	0.035

Table 9: Error analysis of a hovering test for both position and velocity and LQR control

4.6.4 Position and Velocity Control

Tests using only position and velocity control have been run to set a reference for future tests with LQR control. The best gains found are the same as for

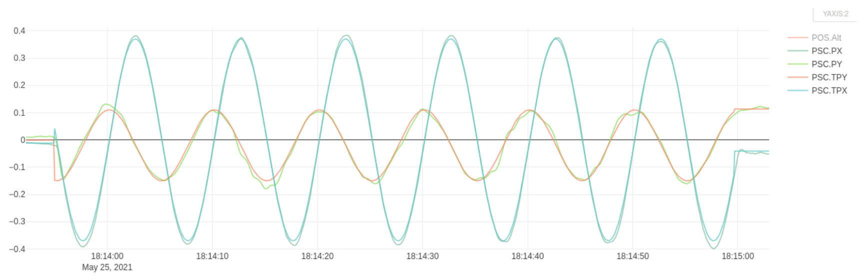


Figure 39: Logs of real life test using position and velocity control with best tuning

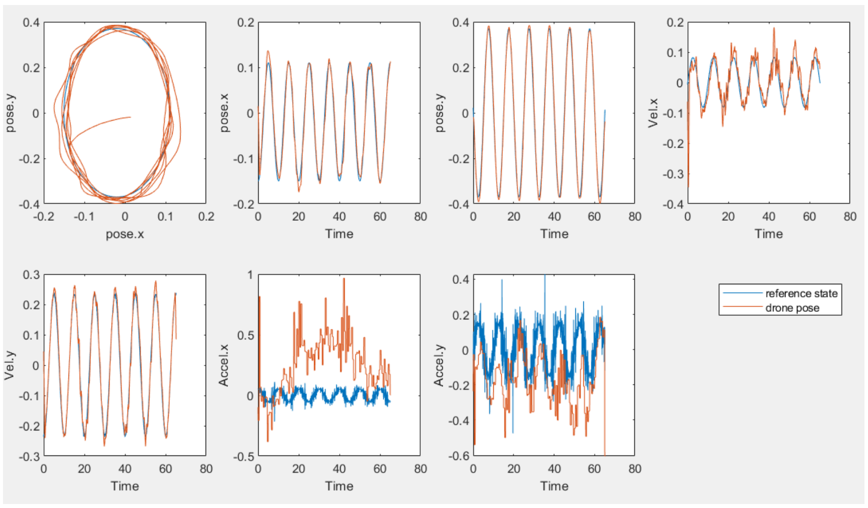


Figure 40: Centered data for real life test using position and velocity control with best tuning

the hovering test. The flight is represented fig. 39 using the logs from the Pixhawk. The graph obtained on Matlab has been re-centered fig. 40 to align with what could be seen from the logs. The flight has been done using the best gains found from Ardupilot. It was a bit windy but less than in the windy simulations.

Error projected along	the X axis	the Y axis
Average delay (s)	0.050	0.050
Average absolute error (m)	0.0011	0.017
Max absolute error (m)	0.043 (at 47.7s)	0.049 (at 24.7s)

Table 10: Error analysis of the flight with the best gains for position and velocity control

The performances of the drone (table 10) are at least as good as in the

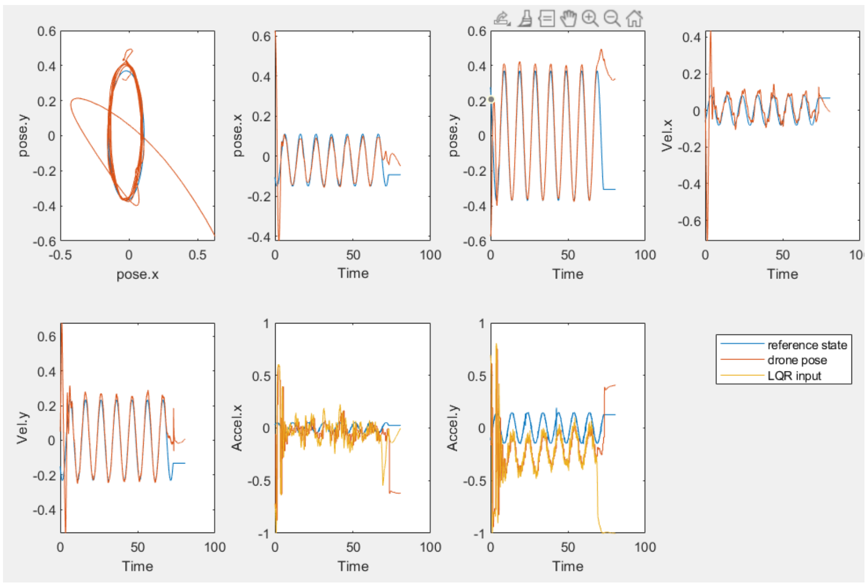


Figure 41: Real life test using LQR control

simulator for a similar amount of wind. Even though the gains were pretty high, the drone was no oscillating that much.

4.6.5 LQR Control

The test (fig. 41) has been done after that all the testing difficulties (section 4.6.1) have been fixed. In this test, $K_{LQR} = (7.5 \;\; 4.0)$. The accuracy of the drone is summed up table 11. During this test, it was just a bit of wind on the y axis and ground-truth set-points have been used.

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.000
Average absolute error (m)	0.0011	0.024
Max absolute error (m)	0.035 (at 25.3s)	0.060 (at 30.6s)

Table 11: Error analysis of a real life LQR control

During this test, it was almost no wind and ground-truth set-points have been used. Though, the acceleration graph on the y axis shows a bit offset from the acceleration of the reference. That means that the drone had to fight against a little bit of wind. Unfortunately, it could not be tested with more wind.

According to the simulations, these results can be further improved by adding a feed-forward on the mast acceleration. The simulation fig. 42 has been done with $K_{LQR} = (7.5 \;\; 5.5)$ and and feed-forward of 0.5. As the big

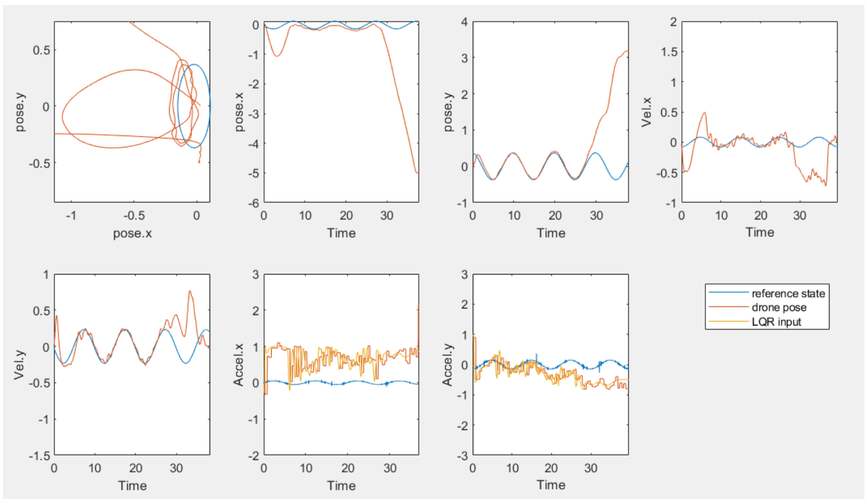


Figure 42: Real life test of LQR control with feed-forward

drift in position and the offset in the acceleration graphs may suggest, a lot of wind was present during this test. Still, during the first 25sec, the flight has been possible and the statistics for this period of time are summed up table 12.

Error projected along	the X axis	the Y axis
Average delay (s)	unknown	unknown
Average absolute error (m)	0.087	0.021
Max absolute error (m)	0.149 (at 17.6s)	0.065 (at 20.9s)

Table 12: Error analysis of the best real life control with LQR and feed-forward

4.6.6 Control With EKF

Once the control works well on fake ground truth data, one can try to fly the drone from the Extended Kalman Filter estimation. As analyzed in the simulator, The EKF resulting noise is being amplified by high LQR gains. Therefore big R matrix coefficients are used from the first real-life tests.

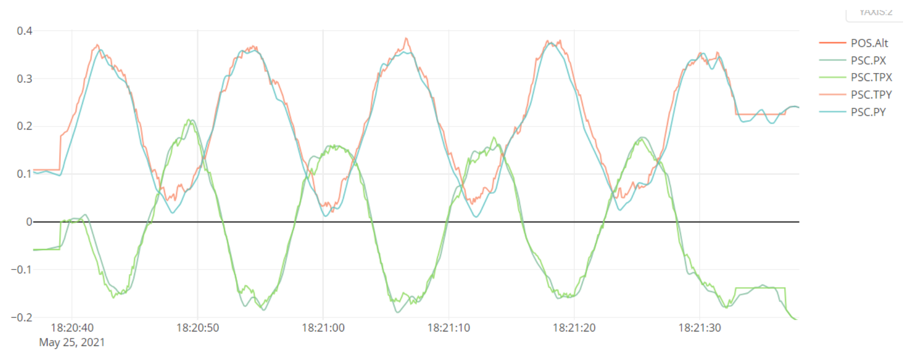


Figure 43: Real life test of a flight based on EKF data with position and velocity control

Fig 43 shows how does the drone reacts when the reference state is coming from the EKF. This flight has been done using position and velocity control. It has unfortunately not been possible to retrieve the data with Matlab, and therefore to get the error estimate for this flight. The average error seems to be about 15 and 30mm with a maximum of about 40 and 50mm respectively on the x and y axis. These numbers just come from visual analysis and won't be used to demonstrate anything as they are not reliable.

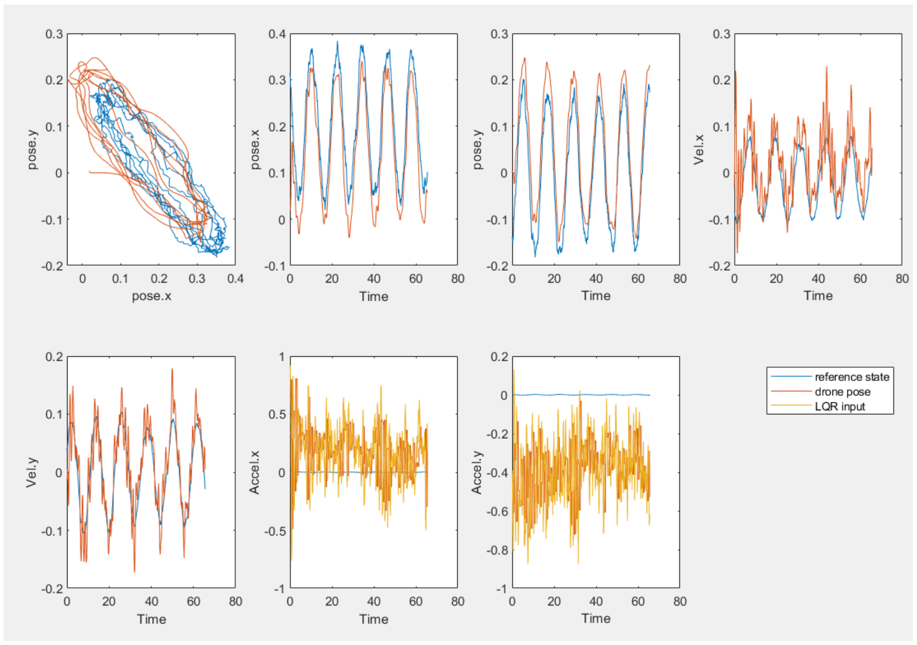


Figure 44: Real life test of LQR control with feed-forward based on EKF data

On the LQR side, fig. 44 shows the result of a flight done under a wind similar to the one in the simulator. This has been done with $K_{LQR} = (7.5 \quad 5.5)$ and a feed-forward of 0.5. This tuning is probably the best one for the LQR. The accuracy of the flight is summed up table 13. The error is quite big, but as the acceleration graph suggests, the wind was somewhat intense.

Error projected along	the X axis	the Y axis
Average delay (s)	0.000	0.100
Average absolute error (m)	0.0050	0.048
Max absolute error (m)	0.107 (at 26.9s)	0.099 (at 8.9s)

Table 13: Error analysis of LQR control based on EKF data

4.6.7 Collision Testing

Finally, the last kind of tests that are being done within this thesis concerns collisions. A stick has been used to simulate contact or obstacles. The drone handled everything in a very stable way, including 2 meter kicks fig. 45. Disturbances have also been tested upward on the z axis and no particular instability has been noticed.

It is very difficult to test mast-like collision without following a real mast. During outdoor testing, it is hard to know in real-time if the drone follows its trajectory in an accurate way, and even more in what direction is the error. Therefore, it is not possible to simulate the action of the mast exactly when the drone deviates too much. However, even collisions pushing the drone away from its trajectory did not make the drone unstable. Therefore, potential collisions with the mast should not be problematic.

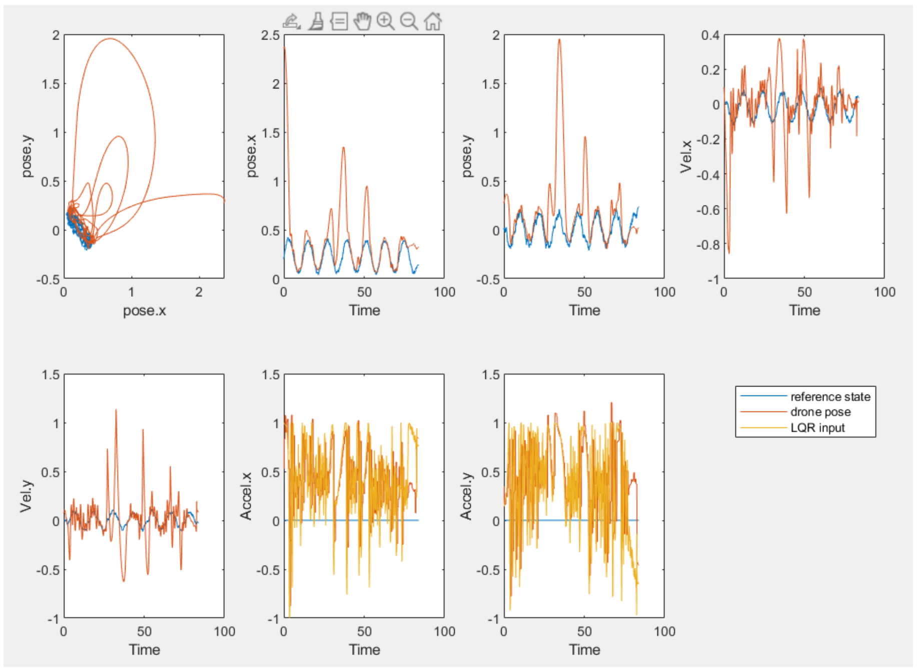


Figure 45: Graphs of the response of the drone after a 2m kick (at 40sec) when flying using LQR control.

5 Results and Discussion

A lot of experiments have been done with both the Ardupilot position and velocity control and the homemade LQR. It is now time to analyze more deeply the results, compare the control strategies and draw a line for further work on the subject.

5.1 Theory Versus Experimentation

The gains found theoretically are quite far from the best results found. To remind, the theory found $K_{LQR} = (50 \;\; 14.1421)$ and in practice, we found $K_{LQR} = (7.5 \;\; 5)$. This would have been found by choosing a maximum acceptable error of $0.19m$ and $0.45m.s^{-1}$ and keeping the maximum input value to 1m/s^2. Even if Bryson's rule is just a rule of thumb and that it should not be taken as a ground-truth rule, there is still a quite big ratio between the best theory and the experimental results. As explained in the theory, it has been considered that the acceleration can be set instantly. In reality, it takes about 0.2sec to set it. As the drone is too slow to react to the input the LQR sends, the gains cannot be very high without having big overshoots. If the delay was shorter, we could expect the experimental results to be closer to the theory.

On the Kalman filter side, the theoretical covariance matrix is about 100 times smaller than the one used to get the best results. This results in a bigger error in the estimation of the mast's state but ensures a smooth reference position for the drone. Like with the control of the drone, it is not only the estimation error that matters here but also how smooth is the estimation. Indeed, if the estimation is not smooth, the drone won't be able to follow it. That would also result in a lot of movement of the FaceHugger hook, which would then be almost impossible to place.

In general, the theory does not take into account the offset between the drone center and the hook of the FaceHugger. An angle of 1° of the drone results in a movement of 8.5mm of the FaceHugger hook compared to the drone center. Therefore, having a drone that is 1° smoother is worth losing 8.5mm of accuracy. This is all the more true that the drone seems to handle contact well. One does not have to worry too much about it. Also, an oscillating drone consumes more power, and the autonomy of the drone and the lifetime of the battery can be reduced. Batteries suffer from not stable and higher discharge current.

The movements relative to the mast worked very well as the change in the trajectory was known by the drone. As long as no constrain was reached, the drone did not significantly lose accuracy. This is because they did not add any additional disturbance in the system and the movements relative to the mast were rather smooth and slow. Faster movements could have been a difficulty, even without acceleration constrain, because of the delay between the acceleration set-points set by the drone and the time the drone actually gets this acceleration.

5.2 Simulations and Real Life Tests Comparison

First of all, the results in the simulator and in real life have been pretty similar. The control accuracy averages 10mm in real life and in the simulator, for a ground truth path with no or low wind. During the research for this thesis, several versions of the simulator have been developed. Depending on the version, the best LQR tuning has changed (all the results showed in this thesis have been taken from the same simulator's version). These versions often included changes in the drone, which required tuning it again. The trajectory following control tuning is based on the internal attitude control of the drone. So if the tuning was not ideal, the LQR would behave differently and require a different tuning as well. This internal tuning has been obtained by using the Ardupilot's auto-tune on both the simulator and in real life. But the tuning is not always ideal on the first try.

It should also be noted that the drone used in the simulator is the big drone that will be used in the competition and not the drone used for the outdoor testings. Due to its size and its configuration (X8), the dynamic is quite different from the test drone that has been used in real life. Still, once everything is properly tuned, the LQR worked best with the same gain on the simulator as outside. So one can expect similar results with the big drone in real life as well. This similarity comes from, among others, the use of the software in the loop (SITL) version of Ardupilot.

Another important difference between the simulator and the real-life application is the localization of the drone. In the simulator, the drone is receiving a ground-truth position. However, in real life, it comes from the GPSs which have about 7mm standard deviation. As the control reached this kind of accuracy, the precision of the GPS becomes important. Still, it should be noted that in real life, the perception group will measure the distance directly from the drone to the interaction point. Even if the real position of the interaction point is useful for the Kalman filter, the impact of the GPSs accuracy should be reduced.

When it comes to the add of perturbations, the simulator has one more time showed results similar to the real-life tests. The drone had the same reaction under wind for both weak and strong tuning, in the simulator or in real life. However, if the simulator can simulate wind-like perturbations, it can not simulate wind as an airflow. For example, it is not possible to set average wind speed, but an average force. This has the drawback of not simulating changes in the lift generated by the drone because of the wind. This drawback did probably not significantly impact this thesis as the control of the altitude was not considered. But at the opposite of what had been assumed in this thesis, the control of the altitude of the drone is not completely independent of the control of the attitude. Another drawback is that the user can not compare the simulated data with real-life testing when the speed is known.

Another important perturbation to consider is collisions between the drone and the external environment. The drone handled it very smoothly in both environments. On the simulator, when the drone hit the mast during mast following, it stays perfectly horizontal. Also, when the drone flying through an obstacle, it stayed stable and did not over-lean. It was not possible in real life to test collisions between the drone and the mast the same way it will be experienced in real life, but good results are being expected since collisions pushing the drone away from its reference trajectory were very well handled.

The ability of handling collisions is very valuable for repairing moving platforms. Given these results, this thesis advises Ascend-NTNU to add a V-shaped guide in front of the drone. It would surround the mast as the drone get closer to it and give the accuracy needed on the right-left axis. Also constantly pushing the drone toward the drone, the accuracy needed forward would also be fixed. As just shown, the drone would still be very stable and able to keep on following the reference. The main issue may then be to follow the correct altitude.

Finally, simulations first seemed to show that the saturation of the attitude input was very important. However, further tuning showed that it is finally not necessary. The saturation was only limiting during very windy tests. This saturation should therefore be removed.

All in all, the simulator can be considered trustworthy for the application of the thesis. Nonetheless, One should keep in mind that it is not perfect and that the simulations outside the scope of the thesis should be validated by real-life experiments.

5.3 Limitations of the Results

Simulated flight results have been confirmed by real-life tests, but it is not the case of the data used as an input of the system. Noisy data has only been simulated by adding white noise to the ground-truth data. As the detection of the reference was not finished at the time this thesis has been written, it was not possible to test the system with real data. The main expected difference on the shape of the noise. Since the Kalman filter model considers white noise, the results may worsen.

Also, one should remember that these results have been obtained with a position that was stored with 9mm accuracy. That should not impact the results too much but only make them slightly worse than it would be with a better position storage accuracy. As the results are good enough with this accuracy, the thesis will not push toward fixing the issue. This fix does not worth the added code complexity and potential bugs. This limiting storage accuracy is also the reason why all the Matlab graphs have squared lines.

From all the experiments, it seems that the estimation of the delay is only 0.050 seconds accurate. Considering that the maximum delay that the drone should have when the mast reaches its maximum velocity is 0.043sec, this accuracy is not enough.

Also, when the center position of the drone fits the reference, the position of the FaceHugger's hook is not necessarily where it was wanted. This is due to the drone oscillations and the offset of the FaceHugger hook from the drone center. The drone needs to adjust its leaning angle both to follow the mast trajectory and as a way to correct the inaccuracy in position and velocity. Under no wind, the maximum acceleration of the drone is about 0.3m/s^2 which gives a leaning angle of $arctan(0.3/9.81) = 1.8$. As explained section 5.1, an angle of $1°$ of the drone leads to a shift of 8.5m of the FaceHugger. So at the end of a movement, when the acceleration of the mast is maximum, the FaceHugger hook will have shifted by 15.3mm. This calculation did not take into account the corrections of the drone, which increases the leaning angle amplitude. This added error is not neglectable. Some actions to take it into account will be proposed section 5.5.

In this thesis, the Kalman filter R matrix has been considered diagonal as the noise of each measurement was independent. However, that may not be the case in real life with real sensors. So the Kalman filter may be improvable by estimating a non-diagonal covariance matrix. Also, the tuning of the EKF should be optimized to reduce the error in position for the hook of the FaceHugger and not the drone. A better trade-off between EKF accuracy and smoothness may be found. Notice that the smoother the filter the longer it takes to converge. The tuning proposed by this thesis already takes between 20 and 30 sec to reach its steady-state. One may also want to test to adjust the tuning during the flight. For example, it could be beneficial to have bigger gains on the Q matrix for the mast length and the wave frequency while the steady-state is not reach. It can put it back to normal afterward to increase the accuracy of the filter. This can be even more important that we don't know the mast length nor the wave frequency at the beginning.

Real-life testing added a lot of unknown and randomness, mostly due to the wind. This thesis did not have the opportunity to do many tests with the same configuration. The understanding of the effects of the wind and how to tune the drone under windy weather could be improved by making the exact same test many times, and by looking at the statistics of the results.

Finally, all the real-life tests have been done with a smaller drone than the drone that will be used during the competition. However, the controller is only on a good tuning of the drone. As long as the internal attitude is well-tuned the results should be similar for whichever drone. This can be emphasized by comparing the simulator results (with a big drone) and the real-life results (with a smaller drone).

So this work assumes that the drone is properly tuned. But there are no good metrics to know if it is the case or not. Of course, the faster the attitude is set, the better the results will be. But what overshoot should be acceptable? From which accuracy can the attitude be considered set? These are questions that a reader willing to reproduce the experiments may wonder. The thesis suggests using the auto-tune feature of Ardupilot or if it is not available, the auto-tune procedure can be followed manually. It would be very interesting to try to get better results by tweaking the attitude controller. One may figure that it is worth allowing more overshoot to decrease the response time. Since the attitude control is just an internal controller, overshoot is not as critical as for the external controllers.

5.4 Methodology

This thesis is a good place to talk about methodology, what was good and what can be improved. The testing procedure has been fairly efficient both with the simulator or in real life. Each test result was added into a big documentation. So it was easy to go back and see previous results. Even though it takes some time to write, it saved some time overall as tests are long to do. The train of thought has been detailed so that other people could follow the progress and learn from this. But it has also been helpful for the author as writing helps to think. Of course, such a document has been very helpful to write this thesis.

On the other side, the analysis done after each test could have been improved. Some issues have been detected quite late, but they could have been observed before. That required redoing a bunch of tests and wasted time. Therefore, the author advises pushing the reflection and the analysis deeply after the first tests of each series of tests.

The learning outcome of the tests including the EKF could have been increased by recording a ROS bag of the reference found by the Kalman filter for each EKF tuning. That way, all the flights with the same EKF tuning would be easier to compare. During this thesis, it has been difficult to compare two different control tuning as the shape of the reference was significantly changing in between the tests.

Overall, Matlab has been a great analysis tool, but it has not been pushed to its maximum efficiency. It is possible to connect Matlab to ROS and to get data in real-time. Matlab Simulink can listen to topics and continuously plot the data. By saving the data obtained, further analysis of error, delay and more can be done afterward. This is potentially also a good way to visualize ROS bags. Though, it has not been judged worthwhile to spend more time optimizing the setup used to analyze the flights. The used analyzing setup had been considered good enough.

No special methodology has been used for tuning. Still, it has been realized afterward that a good way to tune the LQR would be to choose a very low position gain and increase the velocity gain until some oscillations around the reference start to occur. Then, the gain should be reduced by about 25%. After that, the same process can be done with the position gain, keeping the final value found for the velocity gain. Then, the feed-forward gain can be adjusted until the best result is found. Notice that it may be possible to get a smaller error in average by choosing a bit bigger gain. However, this would increase the oscillations of the drone. This is not desired as this will make the FaceHugger hook move compared to the center of the drone. In the scope of the thesis, it is actually the position of the hook of the FaceHugger that matters and not the position of the drone.

The LQR presents the advantage of having only 3 parameters to tune against 4 for the position and velocity control (P gain for position and PID gains for velocity). But Ardupilot includes a range within which the gains should be which ease their tuning. So the tuning difficulty is similar for both control strategies.

5.5 Proposition of Improvements

The biggest difficulty of outside testings was to counter wind perturbations. Without wind, the control of the drone is accurate enough for the goal of the thesis. On one hand, including a wind estimation may improve the robustness of the control. This is not an easy task since the propellers create a lot of air disturbances, which makes sensors like Pitot tube unreliable. However, many researchers are developing methods to estimate it sensorless. This can for example be done using machine learning [18]. On the other hand, adding an integral effect to the LQR can also help to fight consistent winds. But wind is seldom consistent and it may only hardly limit the size of the bounced induced by the wind at the beginning of the gusts. In case of wind, not only the steady-state matters. It may still be worth testing it.

In addition, one may also consider controlling the attitude rates instead of the attitude itself. This would reduce the control latency from 0.2 to 0.1sec. That would however increase the amount of data needed for the controller and therefore significantly increase its complexity, increase the chance of bugs and make debugging and tuning less user friendly. If it is not necessary, this thesis advises not to use this solution.

It is also possible to constrain the velocity and acceleration of the drone within tubes that have the mast's velocity and acceleration as center lines. That would avoid overrated corrections and should therefore limit overshoots and drone oscillations.

There are potentially some improvements that can be done on the control in position and velocity as well. This control includes a filtering of the inputs and an integral saturation that have not been tuned.

The measurement that the controller is based on may not be accurate enough to reach the requirements set to place the FaceHugger. The accuracy of the filtered measurement is estimated by the EKF through a covariance matrix. It can be used to abort the mission if it gets too bad. This feature is interesting for safety aspect, but may not be critical since the drone handles collisions very well. Aborting may also be done in case it is noticed that the drone gets out of control (because of collisions or others). This can be for example detected by unusual big movements or a leaning angle that is far from the reference.

In addition, it may be possible to improve the estimation of the state of the mast through the Kalman filter. One can expect a constant or slowly varying phase shift between the pitch and the roll. Then, the same shift is to be expected between their derivative. Adding these two additional relations would constrain the system and potentially make the output of the Kalman filter more accurate. However, as constraints are added, the robustness of the filter is reduced. If for some reason, the assumption is not verified, the accuracy of the filter will worsen. And since the competition won't deal with a real ship on the sea, but with a simulated mast, one can not be sure if the assumption is verified or not. Therefore, it should be tested how the filter reacts if the assumption is not verified, and maybe keep a relatively high covariance gain relative to this constant phase shift.

Without adding new constrain, the accuracy of the filtering may be improved by using a non-diagonal R matrix for the Kalman filter. This covariance matrix will have to be estimated from a data set of the measurements.

Finally, as explained section 5.3, due to the drone needing to lean to correct its position and velocity, the position of the FaceHugger hook is shifting compared to the drone center during the mast following. Therefore, one wants the drone to be farther away from the mast when the latter is leaning frontward, and closer when it is leaning backward. In general, the reference path of the drone should be updated so that it takes into account the expected leaning angle of the drone. A proposition of how to implement it is described in appendix appendix A.

6 Conclusion

This thesis tested in a simulator and in real life two different control strategies: the built-in Ardupilot's position and velocity control, and a homemade LQR control. The experiments led in the simulator and in real-life showed that the two controllers gave similar results. Both the feed-forward and the non-linear input improved the accuracy of the controller, but the latter was not significantly better. The position and velocity control has the ability to control both the altitude and the horizontal movements of the drone at the same time, and presents a very low code complexity (outside of the Autopilot). Therefore, this thesis advises the use of this type of control for accurate trajectory following for the International Aerial Robotics Competition Mission 9. These results can also be generalized to follow slow time-parametrized reference.

This thesis also showed that the control of a drone can be very accurate by very simple control strategies when it has good data to be based on. Therefore, the main issue for accurate control of drones is to get accurate data about the drones and the target position.

A Appendix - Improvement of the Reference Path

Here is a description of how to take the expected acceleration of the drone into account in the reference path to improve the FaceHugger Hook position accuracy. To remind, due to the drone needing to lean to correct its position and velocity, the position of the FaceHugger hook is shifting compared to the drone center during the mast following. Increasing the size of the path to take into account the change of position of the FaceHugger's hook will increase the acceleration needed by the drone. So it is not the acceleration of the mast that should be taken into account but the acceleration of the reference path. Still, in practice, the difference is very little, so the acceleration of the mast is a good estimation and is much easier to find. Mathematically, we want

$$d_{mast\_drone} = d_{mast\_hook} + atan(\frac{acc_{path}}{g}) \tag{26}$$

If the wind is estimated, it could also be added to the equation. The acceleration needed by the drone to face it is

$$a = \frac{F_{wind}}{M_{drone}} \tag{27}$$

Therefore, eq. (26) becomes

$$d_{mast\_drone} = d_{mast\_hook} + atan(\frac{acc_{mast} + \frac{F_{wind}}{M_{drone}}}{g}) \tag{28}$$

Still, if the estimation of the wind is very noisy, the reference trajectory may present a lot of oscillations. They could lead to an increase of the drone oscillations and therefore decrease the accuracy more than it would improve it.

References

[1] *Ardupilot ArduCopter Documentation.* https://ardupilot.org/copter/index.html. Accessed:2021-19-03.

[2] *Chasing better accuracy.* https://discuss.ardupilot.org/t/chasing-better-accuracy/58966. Accessed:2021-02-10. December 2020.

[3] A. Hernandez; H. Murcia; C. Copot and R. De Keyser. *Model Predictive Path-Following Control of an AR.Drone Quadrotor.* Oct. 2014.

[4] P. Corke. *Robotics, Vision and Control: Fundamental Algorithms In MATLAB®[2 ed.]* Springer, 2017.

[5] T. Degeorges. *Autonomous Physical Module Replacement.* december 2020.

[6] T. Degeorges. *EKF Node.* https://github.com/AscendNTNU/ekf. Accessed:2021-04-19. 2021.

[7] T. Degeorges. *How to plot and analyse data from files.* https://confluence.ascendntnu.no/x/YobEAg. Accessed: 2020-12-19.

[8] *description of the control implementation with Ardupilot.* https://ardupilot.org/dev/docs/code-overview-copter-poscontrol-and-navigation.html. Accessed:2020-12-10.

[9] *finddelay matlab function.* https://se.mathworks.com/help/signal/ref/finddelay.html. Accessed: 2020-12-06.

[10] *International Aerial Robotics Competition Mission 9 rules.* http://www.aerialroboticscompetition.org/assets/downloads/mission9rules_2.01.pdf. Accessed: 2021-03-03.

[11] L.M. González deSantos; J. Martínez-Sánchez; H. González-Jorge; F. Navarro-Medina and P. Arias. *UAV payload with collision mitigation for contact inspection.* 2020.

[12] S. Levy. *Tiny EKF for Arduino.* https://github.com/simondlevy/TinyEKF. Accessed:2021-04-19. 2015.

[13] H. Rafaralahy M. Boutayeb and M. Darouach. *Convergence Analysis of the Extended Kalman Filter Used as an Observer for Nonlinear Deterministic Discrete-Time Systems.* 1997.

[14] J. Dentler; S. Kannan; M. A. Olivares Mendez and H. Voos1. *A real-time model predictive position control with collision avoidance for commercial low-cost quadrotors.* 2015.

[15] N. P. Nguyen and S. K. Hong. *Position Control of a Hummingbird Quadcopter Augmented by Gain Scheduling.* 2018.

[16] *Pixhawk 4 Official Documentation.* https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html. Accessed:2020-04-03.

[17] A. M. Romero. *ROS concept overview*. `http://wiki.ros.org/ROS/Concepts`. Accessed:2021-05-10. 2014.

[18] Balaji Jayaraman Sam Allison He Bai. *Wind estimation using quadcopter motion: A machine learning approach*. Aerospace Science and Technology, Volume98, march 2020.

[19] K. P. Valavanis and G. J. Vachtsevanos. *Unmanned Aerial Systems Physically Interacting with the Environment: Load Transportation, Deployment, and Aerial Manipulation*. Springer, Dordrecht, 2015.

NTNU
Norwegian University of
Science and Technology

ascend
aerial robotics team NTNU