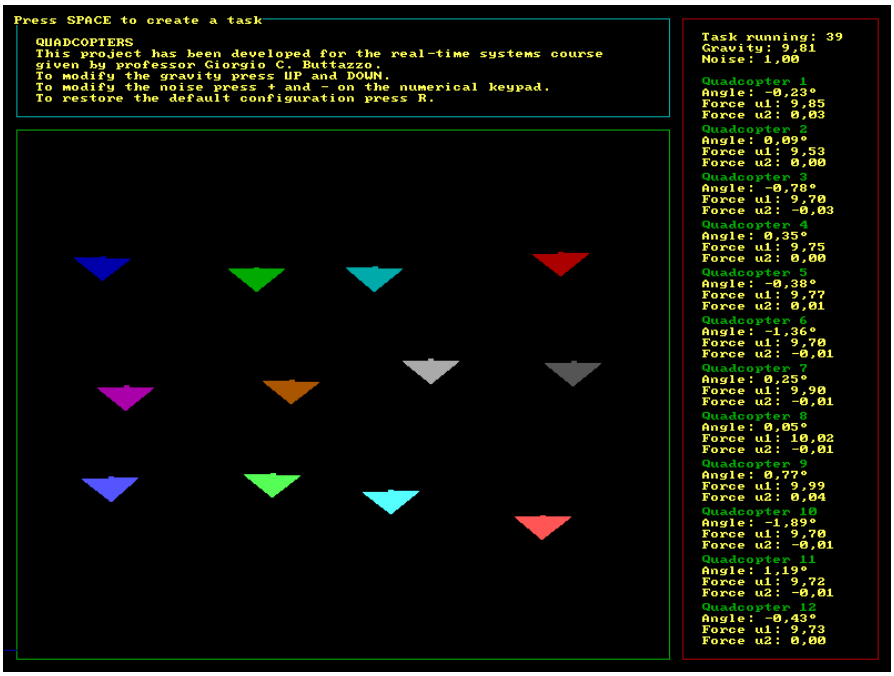
MASTER'S DEGREE IN EMBEDDED COMPUTING SYSTEMS

Real-time systems

Quadrotors



Delivery date:
23/02/2018

Student:
Arcangelo Alberico
albericoarcangelo@hotmail.it
a.alberico@studenti.unipi.it

Academic Year 2017-2018

Index

Index of the figures

1. Introduction

A quadrotor is a helicopter lifted and propelled by four rotors. The rotors of a quadcopter are usually fixed-pitch, so the aircraft is controlled by changing the relative speed of the four rotors and thus increasing or decreasing their thrust and torque. Some quadrotors have variable pitch blades, but even in this case the pitch can be changed only as a group property of the blades of each rotor and not depending on the position.

Small sized quadrotors are often used as UAVs (unmanned aerial vehicles) in research and amateur projects, because of the simple symmetric structure and relatively easy control law. The main advantages of quadrotors with respect to conventional helicopters are:

- quadrotors do not require mechanical linkages to change the pitch of the blades and they do not even use mechanically driven control surfaces like other aircraft types (flaps, rudders...), so they are much easier to design and build, especially in small size;
- the four rotors are smaller than the single rotor of a conventional helicopter of comparable size, allowing them to possess less kinetic energy and thus cause less damage in the case of an accidental crash;
- the symmetric structure and the fact that the movements of a quadrotor depend only on the rotation rates of the propellers make them quite manoeuvrable with relatively simple control systems.

The Kalman filter is a mathematical tool well suited for an algorithmic implementation that estimates the state of a dynamic system influenced by random noise given a set of measurements which are also corrupted by random noise. If the system and measurement equations are linear functions of the state variables and the noises are both normally distributed, the filter can be proven to be an optimal estimator. In practice, the linearity conditions are often not satisfied, but some rielaborations of the filter algorithm have been used for more than forty years in many nonlinear applications with good results. Moreover, it is well suited for computer implementation, because it uses a finite representation of the estimation problem and a discrete step algorithm.

The filter is used to obtain an improved estimation of the attitude and position of the aircraft by integrating the data from the sensors with the predictions of the aerodynamic model.

The purpose of this project is to simulate a set of a 2D quadrotors that follow a desired noisy GPS position estimated using Kalman filter. The project has been done for the Real-Time Systems course given by professor Giorgio Buttazzo for the Embedded Computing Systems Master's Degree. In order to develop the aforesaid, it has been used the C programming language and the Allegro library for the graphical simulation.

2. Quadcopter dynamics

For deriving quadcopter dynamics, it is a good idea to introduce the two frames in which we will operate. The inertial frame defined by the ground and the body frame defined by the orientation of the quadcopter. Since the simulation is in 2D, the inertial frame that has been considered is the Y−Z plane with the *y-axis* pointing right, the *z-axis* pointing up. For the body frame, it has been considered centred on the quadcopter.

2.1 Kinematics

Before delving into the physics of quadcopter motion, let us formalize the kinematics in the body and inertial frames. We define the position of the quadcopter in the inertial frame as $\mathbf{r} = (\mathrm{y}, \mathrm{z})^{\mathrm{T}}$ and the roll angle in the body frame as ϕ. We can relate the body and the inertial frame by a rotation matrix R which goes from the body frame to the inertial frame.

$$R = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \tag{1}$$

In this way a point in the body frame P^{B} will become in the inertial frame $\mathrm{P}^{\mathrm{I}} = \mathrm{P_0}^{\mathrm{B}} + \mathrm{R}{*}\mathrm{P}^{\mathrm{B}}$ where $\mathrm{P_0}^{\mathrm{B}}$ is the coordinate of the origin of the body frame in the inertial frame.

2.2 Dynamics

In order to properly model the dynamics of the system, we need an understanding of the physical properties that govern it. We will begin with a

description of the motors being used for our quadcopter, and then use energy considerations to derive the forces and thrusts that these motors produce on the entire quadcopter. All motors on the quadcopter are identical, so we can analyse a single one without loss of generality.

2.2.1 Motors

The motors used are DC motors, which in control systems are very common actuators because they directly provide rotary motion.

In the model, I assumed the voltage v applied to the motor's armature as the input of the system, while the angular velocity ω as the output. In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. To simplify, I will assume that the magnetic field is constant and, therefore, that the motor torque is proportional to only the armature current i by a constant factor K_T. The back electromotive force e_b is proportional to the angular velocity by a constant factor K_b. Neglecting the inductance L due to its small value and if there is no disturbance torque (T_{load} = 0), the simplified system in the Laplace domain could be written like this:

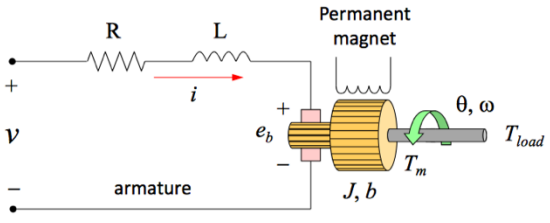


Figure 1 DC motor scheme

$$\frac{\Omega(s)}{V(s)} = \frac{K_T}{R(Js+b)+K_TK_B} \tag{2}$$

where R is the electric resistance, b is the damping coefficient and J is the moment of inertia.

2.2.2 Quadcopter

For a quadrotor modelled in the Y−Z plane, its orientation is defined by a roll angle, φ. It is assumed that its pitch and yaw angles are 0.

In the planer model of the quadrotor, I only consider the thrust force on two of the rotors. The quadrotor has two inputs: the thrust force (u_1) and the moment (u_2). u_1 is the sum of the thrusts at each rotor

$$u_1 = F_1 + F_2$$

while u_2 is proportional to the difference between the thrusts of two rotors

$$u_2 = L(F_1 - F_2)$$

here, L is the arm length of the quadrotor.

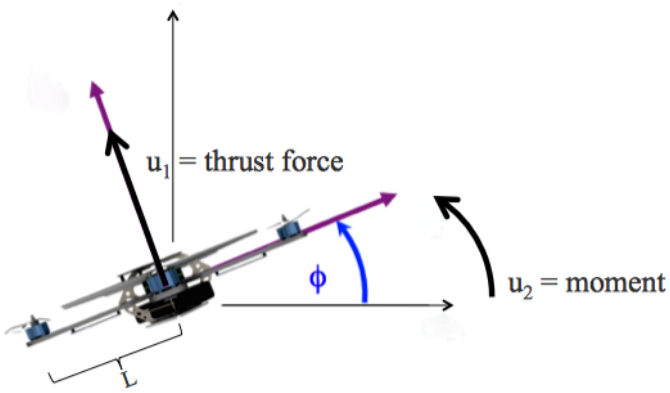


Figure 2 Planar quadrotor model

Let $\mathbf{r} = [\mathrm{y}, \mathrm{z}]^{\mathrm{T}}$ denote the position vector of planar quadrotor in the inertial frame. The forces on the system are gravity and thrust. Hence, by Newton's equations,

$$m\ddot{r} = m\begin{bmatrix}\ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ -mg\end{bmatrix} + \begin{bmatrix}-u_1 \sin\phi\\ u_1 \cos\phi\end{bmatrix} \tag{3}.$$

The angular acceleration is determined by Euler's equation of rotation

$$I_{xx}\ddot{\phi} = L(F_1 - F_2) = u_2$$

As a result, the system model can be written as,

$$\begin{bmatrix}\ddot{y}\\ \ddot{z}\\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix}0\\ -g\\ 0\end{bmatrix} + \begin{bmatrix}-\frac{1}{m}\sin\phi & 0\\ \frac{1}{m}\cos\phi & 0\\ 0 & \frac{1}{I_{xx}}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} \tag{4}.$$

3. Control

3.1 Linearization

The dynamic model of the quadrotor (Eq. 4) is nonlinear. To use a linear controller for this nonlinear system, we first linearize the equation of motions around an equilibrium configuration.

In the case of the quadrotor, the equilibrium configuration is the hover configuration at any arbitrary position y_0, z_0, with zero roll angle. The corresponding thrust force needed to hover at this configuration is exactly $m \cdot g$ while the moment must be zero. Explicitly, the values of the related variables at the hover configuration are

$$y_0,\ z_0,\ \phi_0 = 0,\ u_{1,0} = mg,\ u_{2,0} = 0$$

To linearize the dynamics, we replace all non-linear function of the state and control variables with their first order Taylor approximations at the equilibrium location. In this case, the non-linear functions are $\sin(\phi)$ and $\cos(\phi)$. Near $\phi = 0$, $\sin(\phi) \approx \phi$ and $\cos(\phi) \approx 1$, thus our linearized model is

$$\begin{aligned} \ddot{y} &= -g\phi \\ \ddot{z} &= -g + \frac{u_1}{m} \\ \ddot{\phi} &= \frac{u_2}{I_{xx}} \end{aligned} \tag{5}.$$

3.2 Controllers

In order to control our quadcopter, we have to focus on his position (y, z) and his attitude (ϕ). So, we should design a set of controllers placed in series which will give our desired behaviour. The structure I used to design the controllers is shown in the Figure 3.

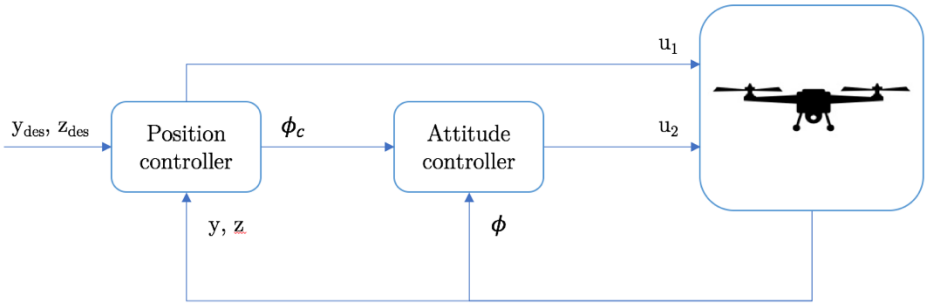


Figure 3 Control structure

The position controller, given a desired position, generates a force u_1 which, contrasting the gravitational force, allows the quadcopter to fly and a desired angle ϕ which is the input of the attitude controller that generates the force u_2 that allows the quadcopter to move horizontally.

Each controller has been designed in discrete time using the MATLAB suite, in particular the Control System Toolbox and then they have been converted into code.

4. Kalman filter

Originally developed for use in spacecraft navigation, the Kalman filter turns out to be useful for many applications. It is mainly used to estimate system states that can only be observed indirectly or inaccurately by the system itself. The Kalman filter not only works well in practice, but it is theoretically attractive because it can be shown that of all possible filters, it is the one that minimizes the variance of the estimation error. Kalman filters are often implemented in embedded control systems because in order to control a process, you first need an accurate estimate of the process variables.

4.1 Model of the filter

The Kalman filters are based on linear dynamical systems discretized in the time domain. They are modelled on a Markov chain built on linear operators perturbed by errors that may include Gaussian noise.

The approach used by Kalman filters is the prediction-correction. That is, they make a prediction, then correct it to provide the final estimate of the systems state. In essence, a Kalman filter

- estimate the state from the previous one;
- estimate the error;
- measure the system;
- update the state calculation;
- update the error calculation.

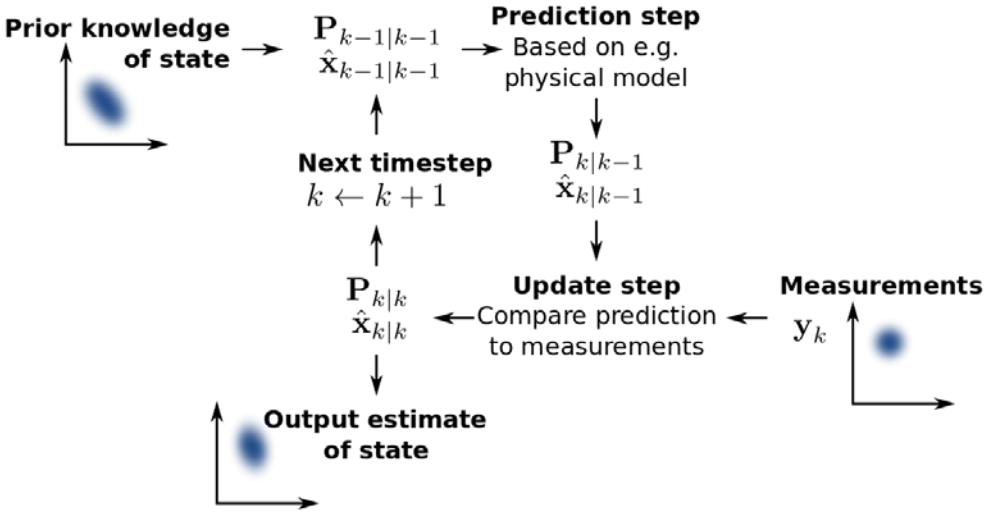


Figure 4 Kalman filter

5. Design choices

Being a real-time simulation, I had to design a multitasking application in which every task had to do something specific within a certain deadline for the correct working of the whole system. So, I had to decide how many tasks were involved and their parameters, what they had to do and with which algorithm they had to be scheduled.

5.1 User interface

Allegro is a cross-platform library mainly aimed at video game and multimedia programming. This library has been used to design the user interface.

I decided to use an 800x600 window with three main areas: *menu area* which contains basic information about the project and the instructions to interact with the simulation, *status area* which contains the status of each quadcopter and *box area* which contains the simulation. A quadcopter is simply an overturned isosceles triangle with a centre of mass in the middle point of the base. When the application starts it is possible to change different parameters (explained in the menu area) using the keyboard and starting the flight of a quadcopter pressing SPACE, then a setpoint is chosen randomly in a predefined quadcopter area for avoiding collisions but it could be also moved wherever in the box area by dragging it with the mouse.

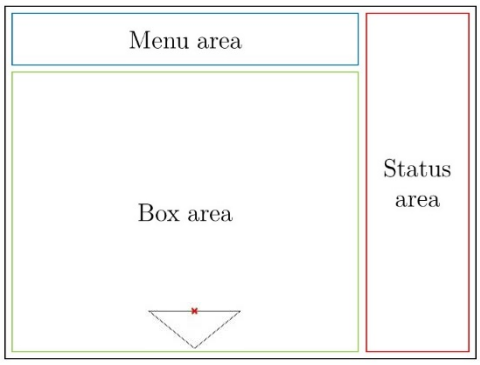


Figure 5 User interface

5.2 Tasks

For what concerning the number of tasks, this depends on the number of quadcopter that we want to simulate. Basically, there is a *user task* which

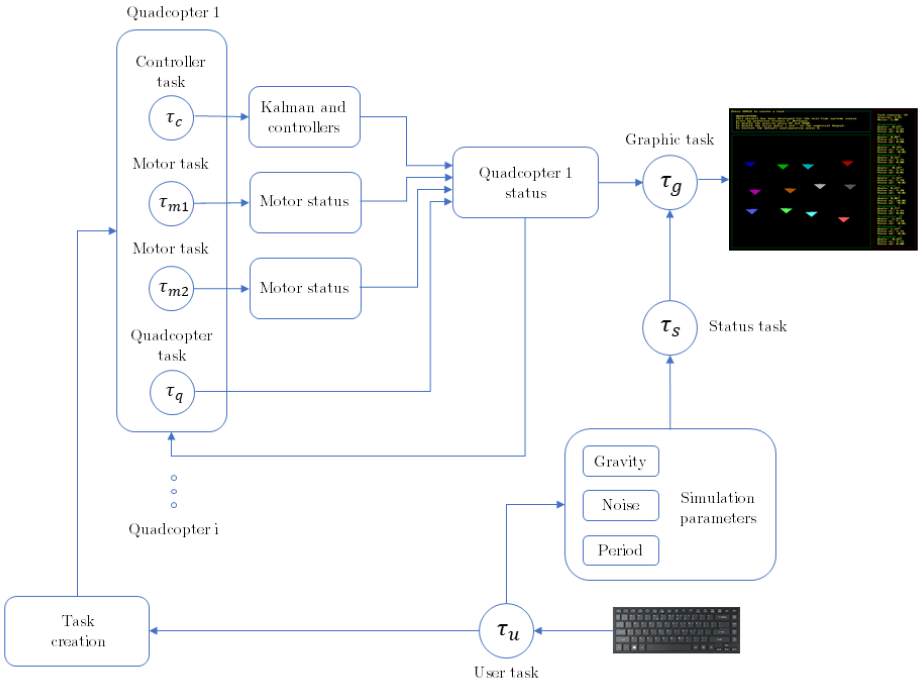


Figure 6 Task diagram

is in charge of generating the other tasks and correctly ending the application. Furthermore, it is also in charge of modify the parameters of the simulation through the keyboard. Then, there is the *status task* which simply refresh the values of the simulation parameters in the status area and a *graphic task* which refresh the whole box area and quadcopters' parameters. For each quadcopter, there is a *controller task* which estimates the position with a Kalman filter and controls the forces to be applied, a *quadcopter task* which calculate the angular velocity of the rotors in order to generate the appropriate forces and update the quadcopter status through the plants, and there are two *motor tasks* which controls the motors.

5.2.1 Parameters choice

For choosing the parameters of each task I had to run them singularly in a separate application to calculate the worst-case execution time, then the periods and the deadlines of the quadcopter's tasks were forced by the plants and controllers sampling time, while the ones of the graphic task have been chosen according to the quadcopter status update so that no position update is missed. The status task, updating only graphically the simulation parameters, has less importance than the others and it has been given a longer period and deadline in order to not jeopardize the execution.

Tasks	WCET	Period	Deadline
Quadcopter	1	30	30
Controller	1	30	30
Motor	1	15	15
Graphic	10	30	30
Status	1	100	100

Figure 7 Tasks' parameters table (in milliseconds)

5.3 Shared data structures

Due to the task structure chosen, the tasks have to communicate to exchange data: the controller has to take the actual position from the quadcopter and generate the appropriate forces that have to be stored in the quadcopter to compute the desired angular velocity of the rotors. These

values are also shared with the motors to generate the actual rotation of the propellers which value, in its turn, is shared with the quadcopter to update the state calculating the actual forces and applying them to the plants. Then, there is the graphic task that takes the position and updates the box area.

For this purpose, we need some data structures in order to share these values, in particular I choose to have for each *quadcopter task* a *struct status*, for each *controller task* a *struct controller* and for each *motor task* a *struct motor*. Each of these structures is properly protected with mutexes, which guarantee a correct access to shared variables and data consistency.

5.4 Scheduling algorithm

As a scheduling algorithm, I decided to use EDF (Earliest Deadline First) for its performances, because it schedules tasks with priorities inversely proportional to their absolute deadlines. On a single processor machine, a task set in order to be schedulable under EDF must have a utilization factor not greater than 1; in formulas, defining C_i as the worst-case execution time of a task and T_i as the period of a task (in this case period and deadline are equals), the condition to be satisfied to schedule a task set under EDF is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Using, instead, a multiprocessor architecture, the relative EDF algorithm is *gEDF* (global EDF) which can achieve performances depending on the number of processors (M) and the maximum utilization factor (U_{max}):

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq M - (M-1) \cdot U_{max} \tag{6}$$

In our case U_{max} is given by the graphic task which value is $\frac{1}{3} = 0.\bar{3}$ and M is equal to 4. In case we consider 12 quadcopters, we will have:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{1}{30} \cdot 12 + \frac{1}{30} \cdot 12 + \frac{1}{15} \cdot 24 + \frac{1}{3} + \frac{1}{100} \leq 4 - (4-1) \cdot 0.\bar{3}$$

$$2.74\bar{3} \leq 3$$

So, we can conclude that the task set is schedulable under *gEDF*.

5.5 Experimental results

In the end, when the whole application worked, I did some experimental simulations to test the quadcopters in different working conditions.

Having designed the controllers ad hoc for the quadcopters, if we modify the sampling time, we will notice that the quadcopters will become unstable.

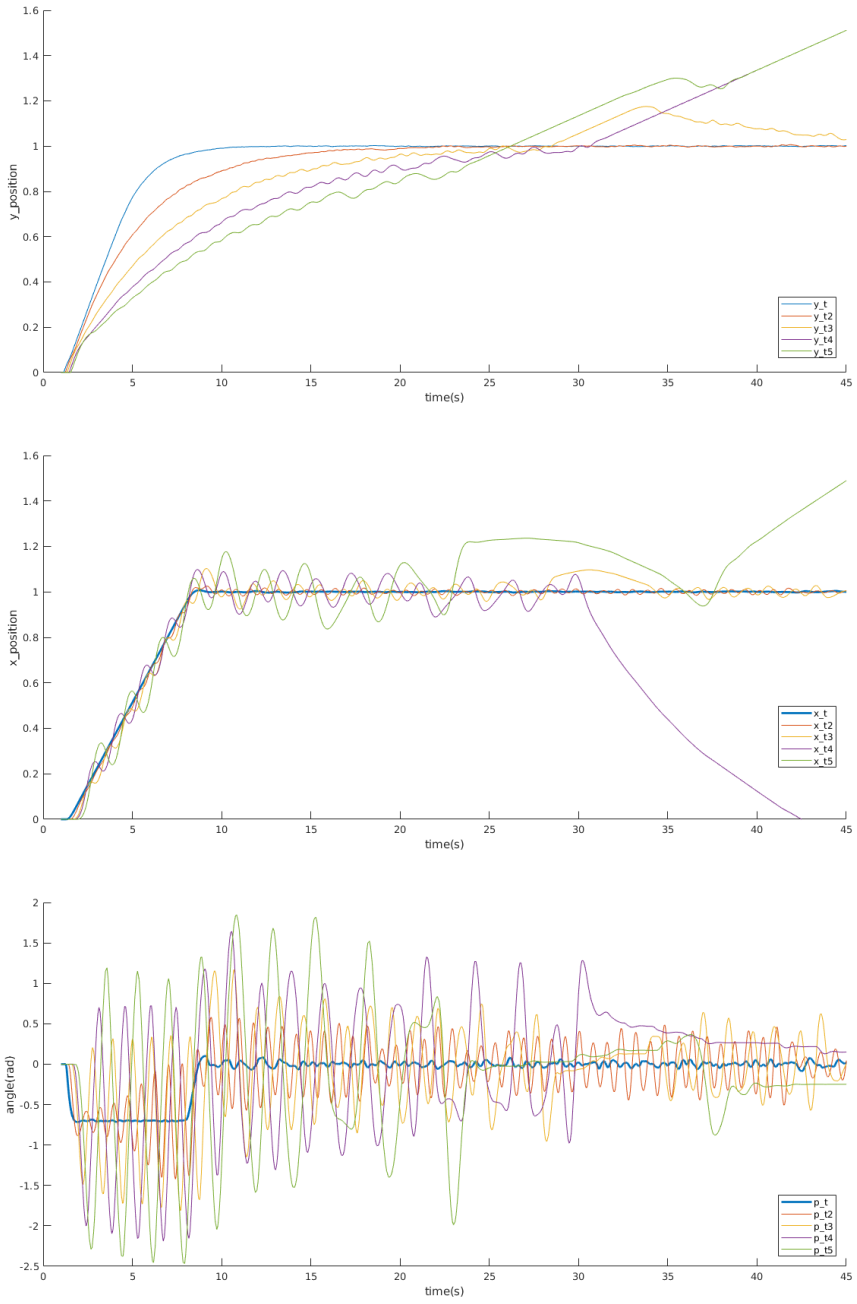


Figure 8 Simulation with different sampling time

In fact, the experiment in Figure 8 has been done modifying the sampling time of the controllers and the motors and remain fixed the one of the quadcopter's plants, in particular the values used for the simulation are $\frac{1}{2}T_s$, $\frac{1}{3}T_s$, $\frac{1}{4}T_s$, $\frac{1}{5}T_s$, where T_s is the original sampling time.

Another experiment (Figure 9) has been done under different noise conditions in which the value of the variable R (observation noise covariance) assumed different values, in particular 0, 1, 10, 20 and 30.

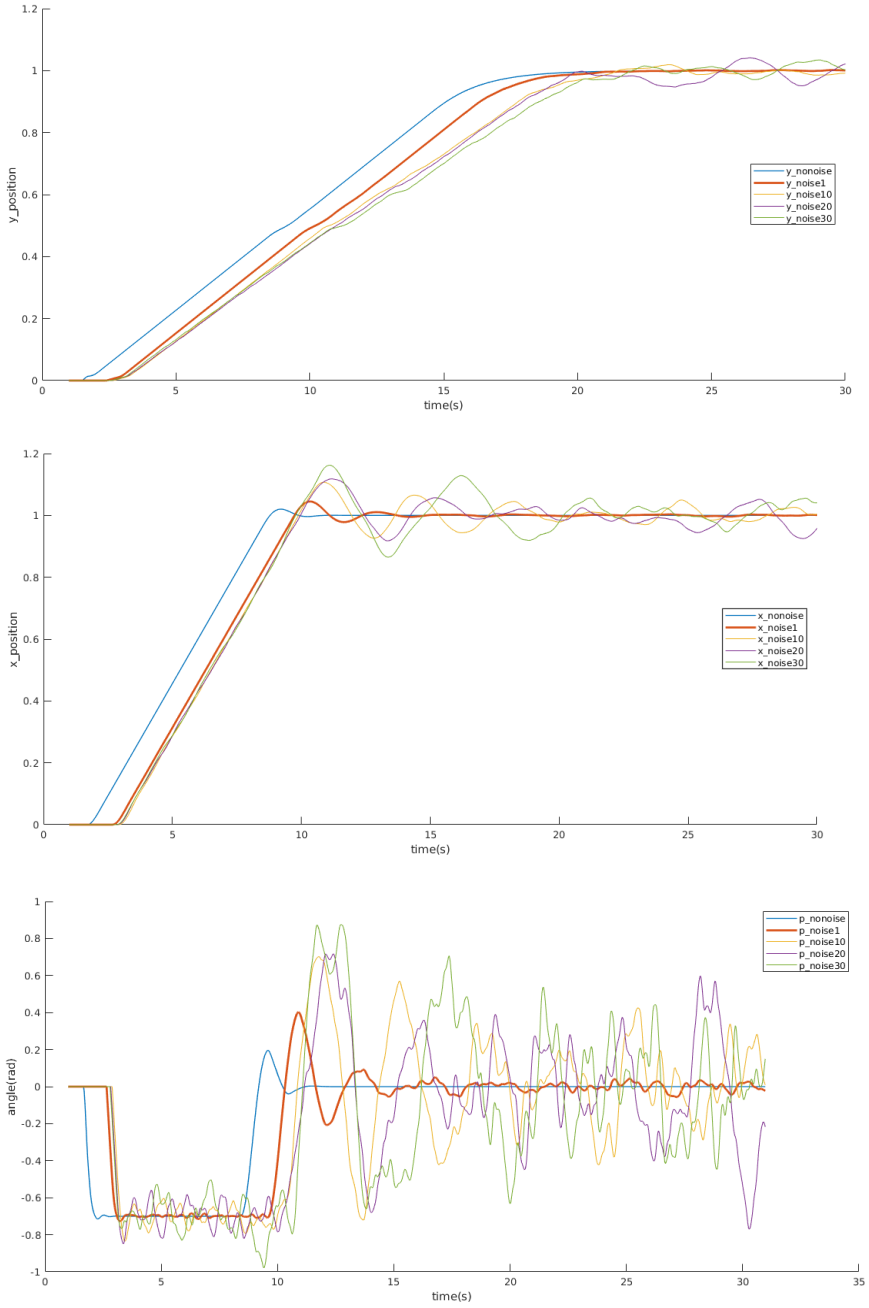


Figure 9 Simulation with different noise

This shows that even in high noise conditions, the Kalman filter struggle to maintain the desired position.

In fact, a last experiment (Figure 10) has been done showing the correction of the Kalman filter in presence of high noise for the altitude of the quadcopter.

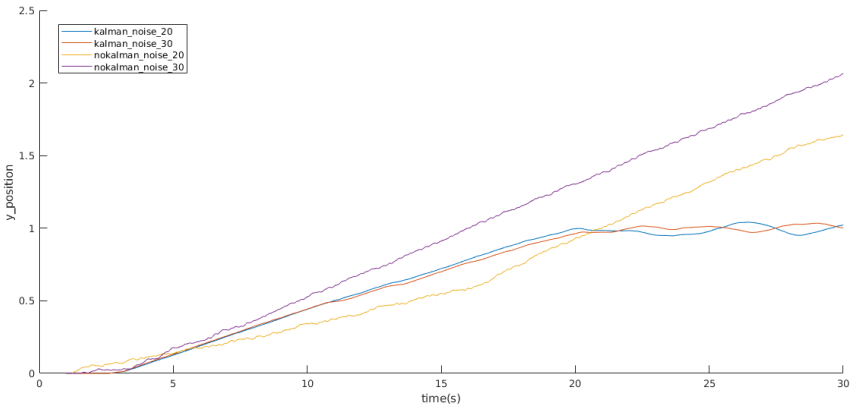


Figure 10 Action of the Kalman filter in high noise condition

Here we can see that with no Kalman filter the position increases with no bound due to the measurements drift. Adding a Kalman filter, the noisy measure is adjusted by considering the physical model and the quadcopter could follow, even if oscillating, the desired position.