Master’s Final Thesis

Master’s degree in Automatic Control and Robotics (MUAR)

Multi Shot Recording using Consensus-Based Cooperative Control for a Multiquadrotor System

August 24, 2019

Author: Marc Pujol Rubio

Advisor: Toru Namerikawa

Date: Spring 2019

慶應義塾
Keio University

ETSEIB
Escola Tècnica Superior
d’Enginyeria Industrial de Barcelona

To my family, that has always been supporting me regardless of the distance.

To Namerikawa-sensei, who brought me the opportunity of joining his amazing team while understanding my curiosity to discover Japan.

And to Kotani-san for his advices, always ready to help.

Summary

In this research work, we consider the control issues related to the recording of a moving object in a cinematographic environment using multiple quadrotors or Unmanned Aerial Vehicles (UAV). First of all, we propose a model of a quadrotor with nonlinear dynamics which physically represents the behaviour of this machine, to later linearize it in order to obtain a suitable control law. After that, we propose a Consensus-Based Cooperative Control so the quadrotors can fly in formation around the moving object and record it from the desired angles. This control technique uses a network structure that represents how the UAVs are connected to each other sharing their information. With that, each quadrotor can be aware of the state of the others in order to properly control the formation of the team. Then, various collision avoidance algorithms are proposed in order to make the flights secure, so the quadrotors do not crash. Moreover, a collision avoidance technique is proposed for the camera control so the UAVs do not get inside the Field of View (FoV) of the other recording drones. That way, the shots taken are suitable for cinematographic purposes. All the control algorithms are validated with simulation and, finally, a real machine experiment using commercial quadrotors is proposed.

Contents

Keio University

List of Figures

1 Introduction

The Movie Industry has recently been using Unmanned Aerial Vehicles (UAVs) in order to produce stunning camera shots. This domain does not only receive great interest from industry, but from end-user consumers. Professional camera teams can now use robots, that were designed to be suitable for a consumer level, in order to create impressive visuals that not much time ago required a helicopter or other expensive camera gear. However, manually controlling this robots which is, for example, flying a quadrotor remains to be a very difficult task which requires a lot of skill and experience. That is the reason why we want to propose a method that automatically controls the quadrotors for shooting purposes, mainly for the recording of a moving object like would happen in an action scene.
Several algorithms have been proposed to approach this problem, for example in [6] a feedback control is used in order to develop predefined trajectories that are commonly used in the movie industry, combining this with a path planning technique for the live capture of cinema scenes. In [4] this approach is taken further and the trajectories are automatically adapted in real-time while solving collision constraints with static and dynamic obstacles. Still, both of these works only use one filming UAV, which means that only one camera shot is taken, which is not suitable for expensive action scenes that can only be performed once, and more than one camera angles are needed.
Even though [5] tries to solve this problem by jointly planning trajectories for more than one UAV, and using Model Predictive Control (MPC) to make sure the quadrotors do not get inside other's Field of View (FoV), this technique is environment dependant and it is used to record shots indoors, like in a studio set.
In this research work, we consider the control issues related to the recording of a moving object in a cinematographic environment using multiple quadrotors in order to obtain multiple shots of a single scene. The objective of this work is to model the dynamics of a quadrotor to use them in order to prove, by simulation, that a real drone can be controlled in the way we desire and, later, show this results implementing the control in a real machine. To do so we consider the model of a quadrotor stated in [3] as well as its linearized model named as *Model for control design*. For the proposed control, we are going to follow the work in [1] in order to develop the network structure that defines the connections between UAVs, as well as for selecting the control gains that make the closed-loop system polynomial have its roots in the negative plane. This is done in [1] by using the generalized Routh stability criterion. After that, collision avoidance control, based on potential fields, is applied to the system in order to avoid collisions between UAVs and fix obstacles. The same technique is used to make sure the quadrotors do not get inside other's FoV, so the shots are suitable for the movie industry. All these algorithms are validated through simulation using tools like *Matlab* and *Simulink*. Regarding hardware, the *Mambo* drone produced by *Parrot Drones* is selected as a suitable quadrotor in order to conduct the real machine experiments.

2 Multirotors

A multirotor is a technological device capable of flying due to three or more propellers and whose take off and landing can be done vertically. Their main advantage respect to the fixed wing is the capability of hovering, staying above a fixed place, which have made them a very useful tool, having from civil applications like surveillance systems to entertainment applications like photography. These flying devices work with electric motors, avoiding the pollution produced by combustion motors, and are powered by high efficiency batteries.
Multirotors can be classified depending on the number of propellers they use. The more they have the easiest will be to control their flight as they will be more stable. However, the most commonly used multirotor is, due to its simplicity, energy consumption and size, the quadcopter.

(a) Hexacopter Drone.

(b) Octocopter Drone.

Figure 1: Two type of multirotors.

2.1 Quadrotor

Quadrotors or Quadcopters are multirotors that use two pair of identical propellers in order to fly. Two of them spin clockwise and the other two counterclockwise. By changing the speed of these propellers it is possible to control the movement of the drone. For example, we can specify the desired Total Thrust so the quadcopter can move vertically, or we can generate a certain moment around one of the drone's axis so its pitch and roll makes it move on the horizontal plane.
A quadcopter is a non-holonomic under-actuated system, as it can move freely in 6 degrees of freedom but only has 4 control inputs, and pitch and roll can not be set to a certain configuration without affecting the position of the drone. This means that position (3 degrees of freedom) and yaw (1 degree of freedom) are the only controllable states.

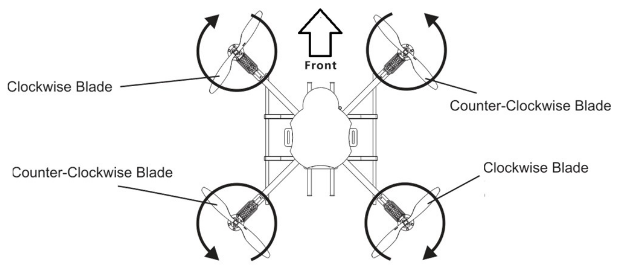


Figure 2: Quadcopter Propellers Turning Direction.

3 Sensors

Multicopters use a great amount of sensors in order to function, however we are only going to cover the ones needed to know the state of the machine and be able to control it. In order to do so, most drones are equipped with an Inertial Measurement Unit (IMU) which is an electronic device formed by the combination of 3-axis accelerometers, gyroscopes and magnetometers.

3.1 Accelerometer

Accelerometers are used to determine the position and orientation of the drone during the flight. They are capable of reading the acceleration along their axis. By combining 3 accelerometers, we can know the direction of movement or the orientation of the drone as accelerometers can sense gravity.

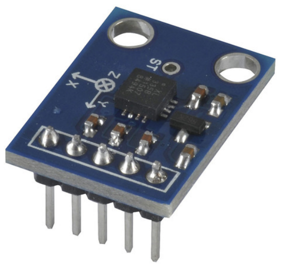


Figure 3: Three axis Accelerometer board.

3.2 Gyroscope

A gyroscope is spinning device, a wheel mounted on a pivoted support which allows it to rotate in a single axis. When it is turning, the orientation of this axis is not affected by the rotation of the support, called gimbal. By combining three of these in orthogonal orientations, the gyroscope axis can remain in the initial orientation independent of the orientation of its support. Then by attaching the gimbal to the drone we can determine its body frame's orientation and angular velocity.

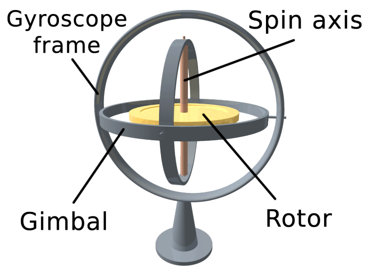


Figure 4: Three axis Gyroscope.

3.3 Magnetometer

A magnetometer is an electronic device capable of measuring the magnetic field around it, so it can basically sense where the north is. By combining three magnetometers we can determine the orientation in space of the drone's body frame.

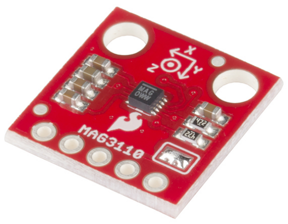


Figure 5: Three axis Magnetometer board.

3.4 Pressure Sensor

Pressure sensors measure the air pressure around them. This is useful for multirotors as air pressure is an indicator of altitude. Differences in air pressure through time can also help us determine the drone's vertical velocity.

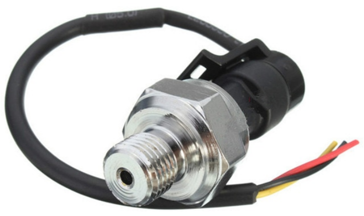


Figure 6: Pressure Sensor.

3.5 Ultrasonic Sensor

These sensors are used to measure distance in a simple way. The sensor head emits an Ultrasonic wave that will travel to the target and be reflected back. By measuring the time it takes for the wave to come back and given the sonic speed, we can calculate the distance to target.

Figure 7: Ultrasonic Sensor.

3.6 Motion Capture

One of the most important and difficult tasks when controlling a drone so it can be a UAV (Unmanned Aerial Vehicle) is positioning it respect to a known frame. When outdoors, aerial vehicles usually use GPS as positioning, but this technology doesn't work indoors. In this case, and in most of the experimental workplaces, Motion Capture (Mocap) is used. Mocap uses optical data to know the 3D position of a special marker. By attaching one marker to a drone we can know its position respect to the camera that is detecting the marker, being this camera a fixed frame. Moreover, if we attach more than one marker to the drone in known positions, we can also know its orientation in space. This is an accurate yet expensive technique for positioning and orientation obtaining for indoors experiments.

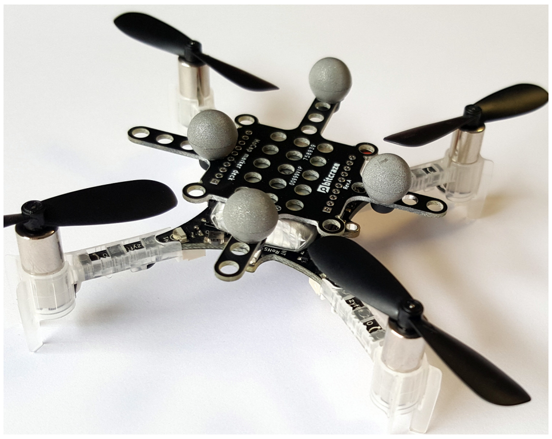


Figure 8: Drone with Mocap Markers attached.

4 Modeling of a Quadrotor

As it has been previously explained, a quadrotor is a device capable of moving freely in space, which means that it is a system with 6 degrees of freedom (3 for position and 3 for orientation). As any mechanical system, its dynamics can be explained with second order derivatives of these variables. In conclusion, a quadrotor system can be mathematically modeled as a nonlinear system of 12 states (3 for position, 3 for orientation, 3 for linear velocity and 3 for angular velocity). Moreover, as its name explains, it has 4 propellers so the system has 4 inputs. From now on, we are going to use the model of a quadrotor in [3] which we are now explaining. The following tables show the symbols that are going to be used to represent the variables and constants of the quadrotor system.

Symbol	Definition
x, y [m]	Horizontal position in world coordinate system.
z [m]	Altitude in world coordinate system.
u, v, w [m/s]	Linear Velocity (x, y, z) in body frame coordinate system.
ϕ, θ, φ [rad]	Attitude angles (roll, pitch and yaw).
p, q, r [rad/s]	Angular Velocities (roll, pitch and yaw).
$M_\phi, M_\theta, M_\varphi$ [Nm]	Moment Commands (roll, pitch and yaw).
T_{total} [N]	Total Thrust Command.
$T_i \in \{1, 2, 3, 4\}$ [N]	Thrust Command for each propeller.

Table 1: Definition of variables for the Quadrotor model.

Symbol	Definition
$I_x, I_y, I_z [kgm^2]$	Inertia Moment (roll, pitch and yaw).
d [m]	Distance between quadrotor geometric center and center of the propeller.
r_φ [m]	Yawing moment coefficient.
m [kg]	Quadrotor mass.
g [m/s]	Gravity constant.

Table 2: Definition of constant parameters for the Quadrotor model.

The equations that explain the dynamics of the system's position and linear velocities are the following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} cos\theta cos\varphi & sin\phi sin\theta cos\varphi - cos\phi sin\varphi & cos\phi sin\theta cos\varphi + sin\phi sin\varphi \\ cos\theta sin\varphi & sin\phi sin\theta sin\varphi + cos\phi cos\varphi & cos\phi sin\theta sin\varphi - sin\phi cos\varphi \\ sin\theta & -sin\phi cos\theta & -cos\phi cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + g \begin{bmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{bmatrix} + \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ -\sum_{i=1}^{4} T_i \end{bmatrix} \tag{2}$$

The equations that explain the dynamics of the system's orientation and angular velocities are the following:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} 1 & tan\theta sin\phi & tan\theta cos\phi \\ 0 & cos\phi & -sin\phi \\ 0 & -sin\phi sec\theta & cos\phi sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{(I_y - I_z)qr}{I_x} \\ \frac{(I_z - I_x)rp}{I_y} \\ \frac{(I_x - I_y)pq}{I_z} \end{bmatrix} + \begin{bmatrix} \frac{M_\phi}{I_x} \\ \frac{M_\theta}{I_y} \\ \frac{M_\varphi}{I_z} \end{bmatrix} \tag{4}$$

where,

$$\begin{cases} M_\phi = d(-T_1 - T_2 + T_3 + T_4) \\ M_\theta = d(T_1 - T_2 - T_3 + T_4) \\ M_\varphi = r_\varphi(T_1 - T_2 + T_3 - T_4) \end{cases} \tag{5}$$

Equations (1-5) represent the dynamics of the quadrotor system with nonlinear functions. For simplicity and to be able to develop the control algorithms that will stabilize the system we are going to linearize it.

4.1 Linearization

The linearization is going to be done around the following equilibrium point:

$$[x_0, y_0, z_0, u_0, v_0, w_0, \theta_0, \phi_0, \varphi_0, p_0, q_0, r_0]^T = 0 \tag{6}$$

$$[M_{\theta 0}, M_{\phi 0}, M_{\varphi 0}, T_{total0}]^T = [0, 0, 0, mg]^T \tag{7}$$

Using the first order Taylor expansion as an approximation we obtain the following linearized system:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = g \begin{bmatrix} -\theta \\ \phi \\ 0 \end{bmatrix} + \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ T_T \end{bmatrix} \tag{9}$$

Where $T_T = T_{total} - T_{total0}$.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{M_\phi}{I_x} \\ \frac{M_\theta}{I_y} \\ \frac{M_\varphi}{I_z} \end{bmatrix} \tag{11}$$

We can observe from the previous equations that the whole system can be separated into four "subsystems" that can be independently controlled.

4.1.1 Horizontal Subsystem (X-Y Plane)

For simplicity we are going to embed the constants inside the variables in the following way:

$$\widetilde{\theta} = -g\theta, \quad \widetilde{q} = -gq, \quad \widetilde{M_\theta} = -\frac{g}{I_y} M_\theta \tag{12}$$

$$\widetilde{\phi} = g\phi, \quad \widetilde{p} = gp, \quad \widetilde{M_\phi} = \frac{g}{I_x} M_\phi \tag{13}$$

Finally, the linearized system:

$$\begin{bmatrix} \dot{x} \\ \dot{u} \\ \dot{\widetilde{\theta}} \\ \dot{\widetilde{q}} \\ \dot{y} \\ \dot{v} \\ \dot{\widetilde{\phi}} \\ \dot{\widetilde{p}} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ \widetilde{\theta} \\ \widetilde{q} \\ y \\ v \\ \widetilde{\phi} \\ \widetilde{p} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \widetilde{M_\theta} \\ \widetilde{M_\phi} \end{bmatrix} \tag{14}$$

4.1.2 Vertical Subsystem (Z Coordinate)

For simplicity we are going to embed the constants inside the variables in the following way:

$$\widetilde{w} = -w, \quad \widetilde{T}_{total} = \frac{T_T}{m} \tag{15}$$

Finally, the linearized system:

$$\begin{bmatrix} \dot{z} \\ \dot{\widetilde{w}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ \widetilde{w} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \widetilde{T}_{total} \tag{16}$$

4.1.3 Yaw Subsystem

For simplicity we are going to embed the constants inside the variables in the following way:

$$\widetilde{M_\varphi} = \frac{M_\varphi}{I_z} \tag{17}$$

Finally, the linearized system:

$$\begin{bmatrix} \dot{\varphi} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \varphi \\ r \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \widetilde{M_\varphi} \tag{18}$$

4.2 Camera representation

Even though this is not part of the quarotor dynamics, for simulation purposes it is important to define how we are going to represent the camera Field of View and it's movement. The camera is going to be attached at the center of every drone recording in the direction of its X-Axis body frame. To be able to represent the camera Field of View we will need the following camera specifications:

- Depth of View D, which represents the maximum distance the camera can clearly record.
- Camera Angle Θ, which explains the angle in which the camera lens is able to record.

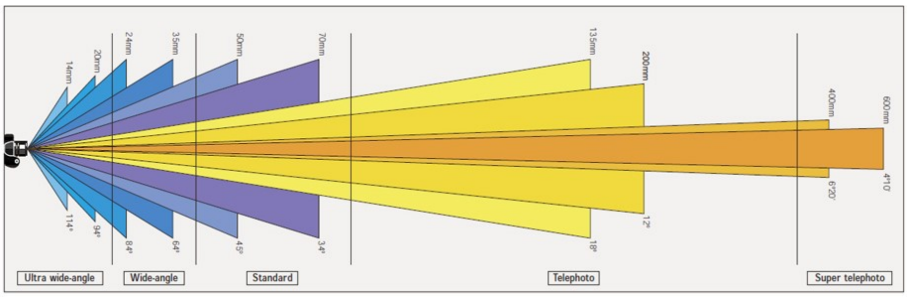


Figure 9: Camera's Depth of View and Angles.

For simulation purposes we are going to use $\Theta = 25$ and $D = 2m$.
With this, we are going to represent the camera field of view as a triangle with points at $[0, 0]$, $[D, Dtan\Theta]$ and $[D, -Dtan\theta]$ in the drone's body frame. The field of view is going to be rotated around the drone's Z-Axis body frame about its Yaw Angle.

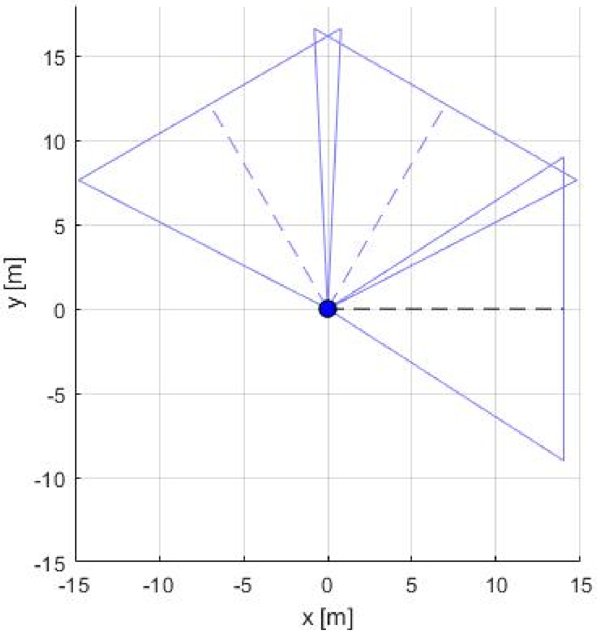


Figure 10: Representation of three camera Fields of View.

5 Modeling of a Multi-Unmanned Aerial Vehicle System

At this stage, when the drone dynamics are already modeled, we need to model the multi-UAV system as a group of drones, which will be called agents, and a leader that can exchange information creating a network. To do so, we are going to use graph theory. The graph $G = (\nu, \varepsilon)$ contains nodes $\nu = \left\{ v_1, v_2, ...v_N \right\}$ and edges $\varepsilon = \left\{ e_{i,j} \right\}$. Each node represent an agent and the edges are the connections between them. Being $A \in R^{NxN}$ the adjacency matrix, $D \in R^{NxN}$ the degree matrix and $L \in R^{NxN}$ the Laplacian matrix, we can assign the values of the adjacency matrix as:

$$a_{ij} = \begin{cases} 1 \quad if \quad e_{ij} \in \varepsilon \\ 0 \qquad\quad else \end{cases} \tag{19}$$

And we can define the degree matrix as:

$$D = diag(deg(v_1), ...deg(v_N)) \tag{20}$$

being $deg(v_i)$ the number of edges leaving the node v_i. With this, we can define the laplacian matrix as:

$$L = D - A \tag{21}$$

The laplacian matrix has the following interesting property: If it has a single eigenvalue at zero and all nonzero eigenvalues have a positive real part, the network contains a directed spanning tree. This means that one of the agents, which we will call the leader, is directly or indirectly connected to all the other agents.

6 Proposed Control

6.1 Control Objective

The objective of the control algorithm is to make N quadrotors fly maintaining a desired formation around a leader (agent N+1). The leader is a moving object, which means that each quadrotor has to converge to a time varying desired position that depends on the leader position and the chosen formation around it. To achieve that, we need to make sure that the graph that represents the network that connects the N+1 agents contains a directed spanning tree with the root at the leader.

6.2 Consensus Algorithm

The proposed control in order to achieve the explained objective is Consensus Algorithm. One of the advantages of this approach is that the drones do not need to be directly connected to the leader. The position of each drone is going to converge to the desired one if they are connected to their neighbors and the network contains a spanning tree with the root at the leader. As we have seen in Section 4, each drone's system can be separated into four subsystems which can be independently controlled. Given that fact, we are going to propose a different control law for each subsystem.

6.2.1 X-Y Plane

The control law we are going to feed to the quadrotor *i* in order to maintain formation in the horizontal plane is:

$$\widetilde{M}_i = -\sum_{j=1}^{N+1} a_{ij} \left[\sum_{k=0}^{3} \beta_k \left(\widehat{r}_i^{(k)} - \widehat{r}_j^{(k)} \right) \right] \qquad i \in \left\{ 1, 2, ..., N \right\} \tag{22}$$

$$\widehat{r}_j^{(k)} = r_j^{(k)} - d_{r_j}^{(k)} \qquad j \in \left\{ 1, 2, ..., N+1 \right\} \qquad k \in \left\{ 0, 1, 2, 3 \right\} \tag{23}$$

where,

- The subscript N+1 denotes the leader.
- β_k are the positive control gains, which are going to be chosen adequately taking into account the conditions stated in [1].
- $d_{r_j}^{(k)}$ is the desired position of the agent *j* relative to the leader frame.
- a_{ij} are the terms of the adjacency matrix. Note that this will make each agent only obtain information from the other directly connected quadrotors.

It is also important to remark that $\widetilde{M} = \left[\widetilde{M}_\theta, \widetilde{M}_\phi \right]$ as:

$$r^{(0)} = [x, y]^T, \quad r^{(1)} = [u, v]^T, \quad r^{(2)} = \left[\widetilde{\theta}, \widetilde{\phi} \right]^T, \quad r^{(3)} = [\widetilde{q}, \widetilde{p}]^T \tag{24}$$

6.2.2 Z Coordinate

The control law for the vertical movement looks very similar to the previous one:

$$\widetilde{T}_{total_i} = -\sum_{j=1}^{N+1} a_{ij} \left[\sum_{k=0}^{1} \gamma_k \left(\widehat{h}_i^{(k)} - \widehat{h}_j^{(k)} \right) \right] \quad i \in \left\{ 1, 2, ..., N \right\} \tag{25}$$

$$\widehat{h}_j^{(k)} = h_j^{(k)} - d_{h_j}^{(k)} \quad j \in \left\{ 1, 2, ..., N+1 \right\} \quad k \in \left\{ 0, 1 \right\} \tag{26}$$

where γ_k are the positive control gains, which are going to be chosen adequately taking into account the conditions stated in [1]. Note that in this case there are only two control gains instead of four, as the vertical subsystem is a second order system. It is also important to remark that:

$$h^{(0)} = z, \quad h^{(1)} = \widetilde{w} \tag{27}$$

6.2.3 Yaw Angle

The yaw angle subsystem is also a second order system, which makes its control law exactly the same as the one proposed for the vertical subsystem:

$$\widetilde{M}_{\varphi} = -\sum_{j=1}^{N+1} a_{ij} \left[\sum_{k=0}^{1} \alpha_k \left(\widehat{f}_i^{(k)} - \widehat{f}_j^{(k)} \right) \right] \quad i \in \left\{ 1, 2, ..., N \right\} \tag{28}$$

$$\widehat{f}_j^{(k)} = f_j^{(k)} - d_{f_j}^{(k)} \quad j \in \left\{ 1, 2, ..., N+1 \right\} \quad k \in \left\{ 0, 1 \right\} \tag{29}$$

where,

$$f^{(0)} = \varphi, \quad f^{(1)} = r \tag{30}$$

6.3 Yaw Setpoint Update

Usually the desired formation around the leader is going to be fixed, however, given that we want that all the agents, which are equipped with cameras, to be recording the leader, the desired yaw angle for each of them is going to change depending on its relative position to the leader. To be able to do so, we are going to need to update the desired relative yaw we are feeding the controller at each time step. Being W and L the world and leader frames, respectively, the desired yaw is calculated with the following formulas:

$$\varphi_{d_i}^{W} = arctan\left(\frac{y_L - y_i}{x_L - x_i} \right) \tag{31}$$

$$\varphi_{i_d}^{L} = d_{f_i}^{(0)} = \varphi_{d_i}^{W} - \varphi_{L}^{W} \tag{32}$$

6.4 Consensus Algorithm: Simulation Results

In order to prove that the proposed control fulfills its objectives we are going to simulate the system with Matlab/Simulink. The Simulink model used can be seen in figure (Fig.11).

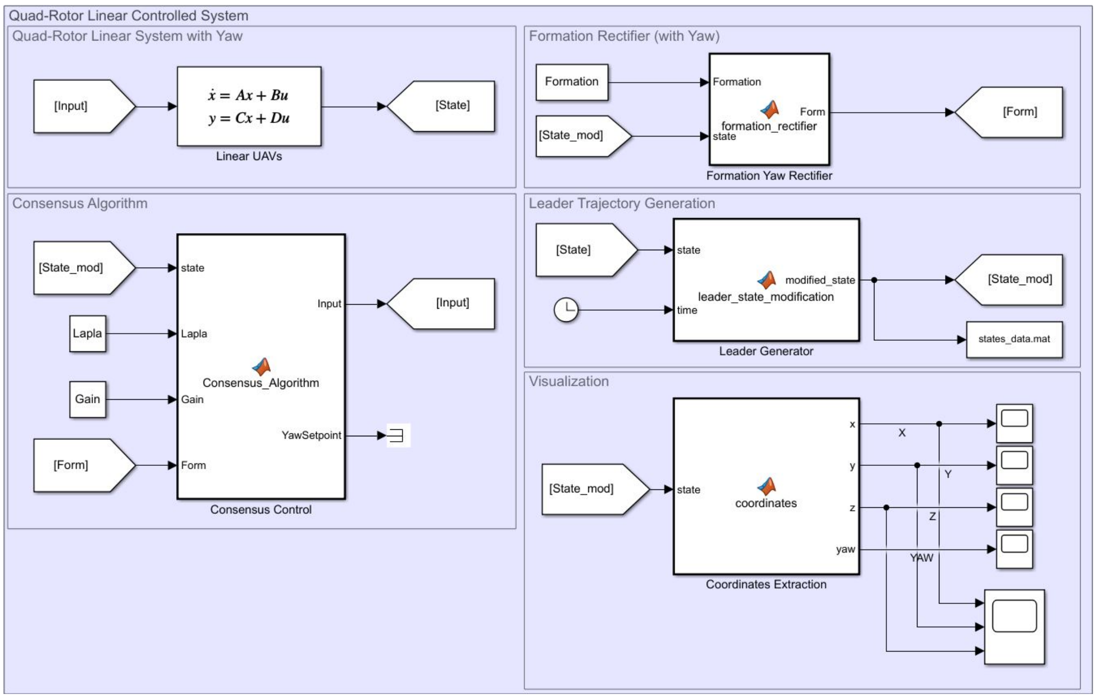


Figure 11: Consensus Algorithm Simulink System.

The mission of each block is:

- **Quadrotor Linear System**: Simulates N+1 quadrotors with 12 states each using the equations explained in Section 4, being $12(N+1)$ the total number of states of the system.
- **Formation Rectifier**: Computes the necessary rotation, given the leader's yaw, so the formation is correctly in the leader's frame.
- **Leader Trajectory Generation**: Generates a trajectory for the leader. This is only for simulation purposes, in a real world situation, the leader would be a moving object.
- **Consensus Algorithm**: Calculates the control inputs for the N agents using Consensus Algorithm.

6.4.1 Fix Leader

For the first simulation of the system we are going to assume:

- **Fix Leader**, the leader will remain in place [0,0,-5].

- **Formation = 0**, all the agents have to converge to the leader's position.
- **N+1 = 4**, a total of 3 follower agents and 1 leader.
- **Laplacian:**

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

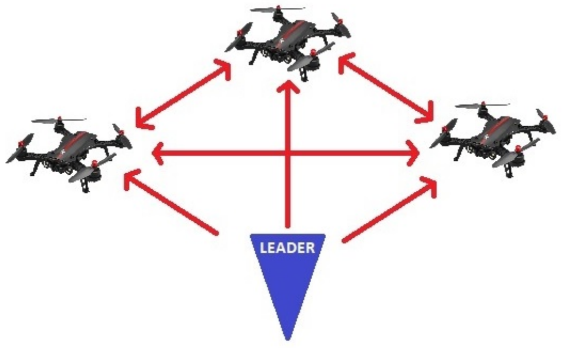


Figure 12: Drone connections for the first simulation.

- **Control Gains:**

Gains	k = 0	k = 1	k = 2	k = 3
β_k	6	15	15	5.5
γ_k	1	1.8	-	-
α_k	10	2.5	-	-

Table 3: Control Gains for the first simulation.

As we can see, all the follower drones are connected to each other and also receiving information from the leader. The desired formation has been set to 0 so all of them will have to converge into the leader's position, which is going to remain still. The following figure (Fig. 13), shows the results of the simulation:

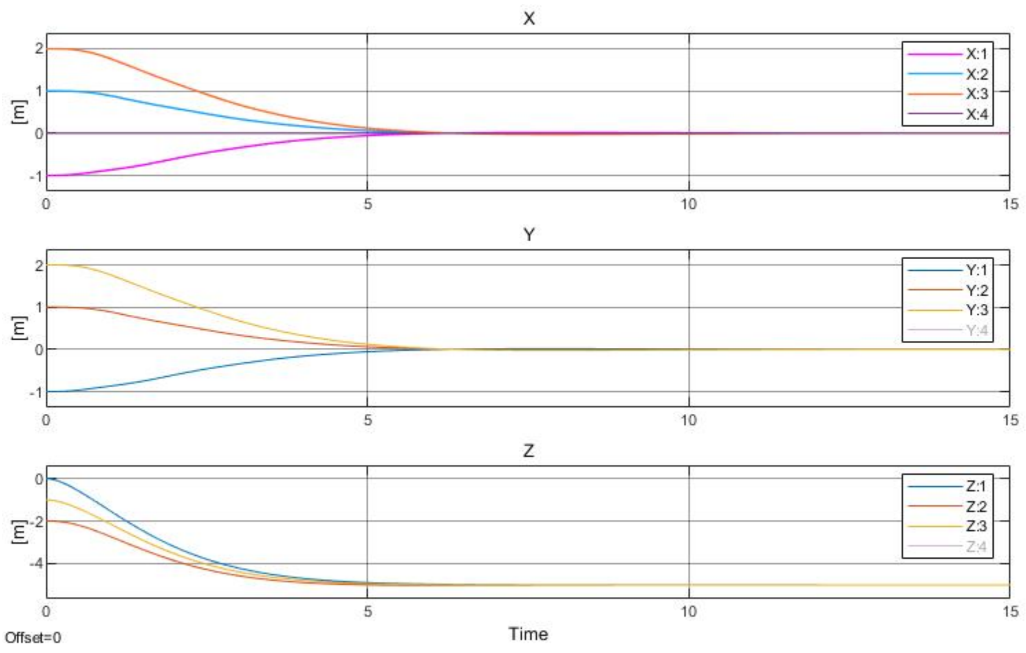


Figure 13: Simulation results with a fixed leader (N=4).

6.4.2 Moving Leader

Being the first simulation's results satisfactory, we can say that the control method is working. However, we have to make sure that the UAVs are able to maintain the formation when the leader is moving and spinning. To do so we are going to conduct a simulation with the following characteristics:

- **Moving Leader**, the leader will be moving following an arbitrary trajectory.
- **Formation:**

N	d_x	d_y	d_z
1	-0.5	0.0	0.0
2	0.0	-0.5	0.0
3	0.0	0.5	0.0

Table 4: Desired positions relative to the leader's frame.

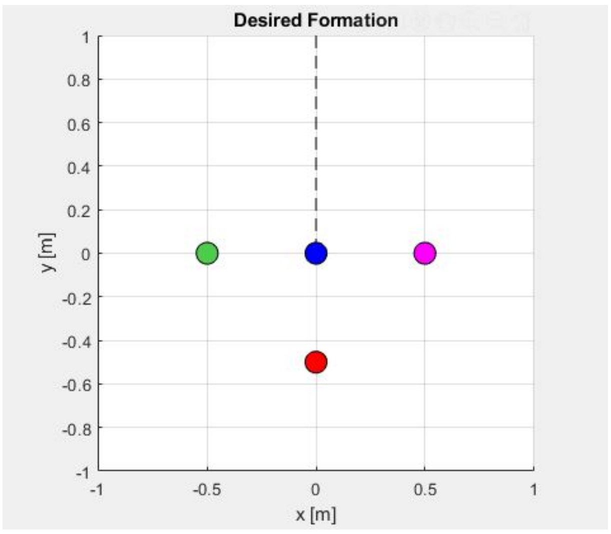


Figure 14: Desired formation for the moving leader (blue dot) simulation.

- **N+1 = 4**, a total of 3 follower agents and 1 leader.
- **Laplacian:**

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Control Gains**, we are going to use the same ones that were established in Table 3.

The results of this simulation can be seen in the following figure (Fig. 15) and in the provided videos ($moving\_leader\_2D.avi$ and $moving\_leader\_3D.avi$).

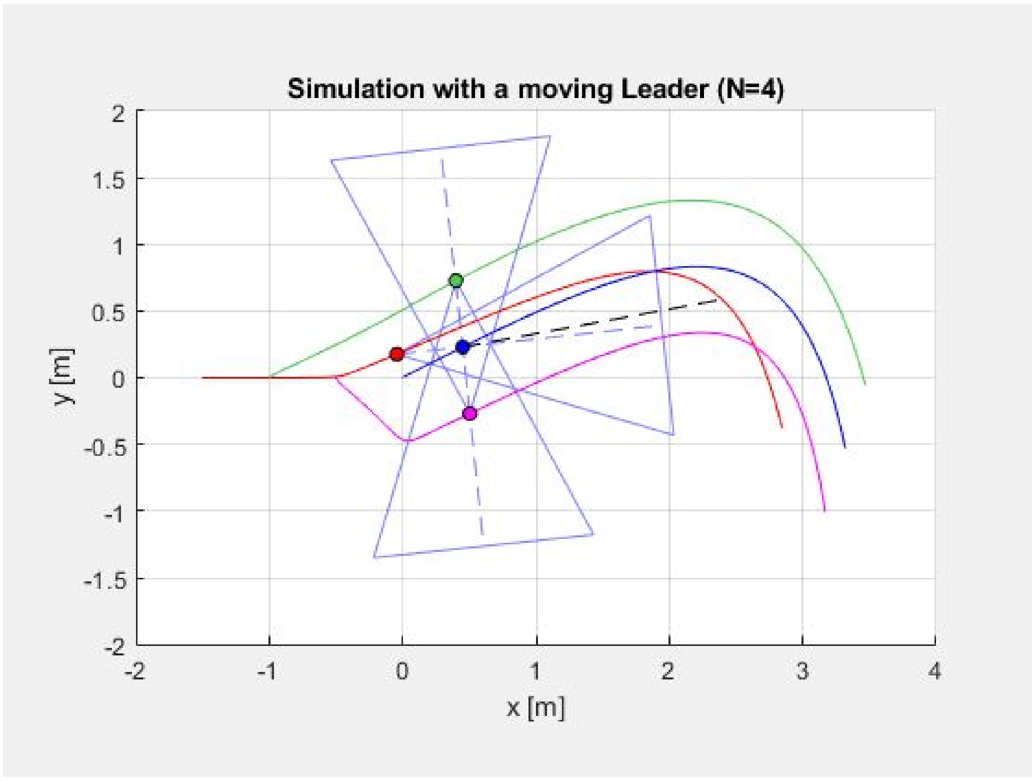


Figure 15: Moving formation, being the blue dot the leader.

6.5 Collision Avoidance: Potential Field

For safety reasons it is important to make sure that none of the drones will collide with each other or with an external object. The approach we are going to follow in order to make that sure is adding a collision avoidance term to the control inputs calculated with consensus algorithm. Specifically we are going to use the potential field technique.

6.5.1 Obstacle Avoidance: Fix obstacle to UAV

The following figure (Fig. 16) shows the required variables to calculate the collision avoidance term that is going to be applied to each agent.

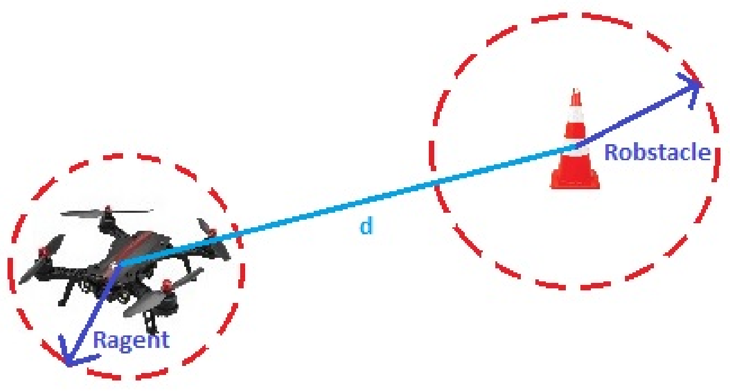


Figure 16: Obstacle collision avoidance variables.

Each quadrotor and fixed obstacles have a safety zone of radius R_{agent} and R_{obs} respectively. We are going to build a potential field such that the fixed obstacle can not be inside the agents' safety zone. We represent with d the distance vector between the agent and the fix obstacle.

$$U_{fix} = -\frac{K_f}{2}\left(\frac{1}{d - R_{agent}} - \frac{1}{R_{obs}}\right)^2, \quad d \in R^2 \tag{33}$$

From which we get the collision avoidance terms:

$$[f_x^{obs}, f_y^{obs}] = \begin{cases} -K_f\left(\frac{1}{|d| - R_{agent}} - \frac{1}{R_{obs}}\right)\frac{d}{|d|^3}, & if\ d \leq R_{agent} + R_{obs} \\ 0, & else \end{cases} \tag{34}$$

$$\widetilde{M_\theta} = \widetilde{M_\theta} + f_x^{obs}, \quad \widetilde{M_\phi} = \widetilde{M_\phi} + f_y^{obs} \tag{35}$$

To prove the effectiveness of the implemented method we are going to simulate the system with a fixed obstacle in the way. The simulation parameters are going to be the same that were used in Section 6.4.2, with a change in the formation:

N	d_x	d_y	d_z
1	-7.0	4.04	0.0
2	-7.0	-4.04	0.0
3	0.0	-8.00	0.0

The results of this simulation can be seen in the following figure (Fig. 17) and in the provided videos ($fix\_obstacle\_2D.avi$ and $fix\_obstacle\_3D.avi$).

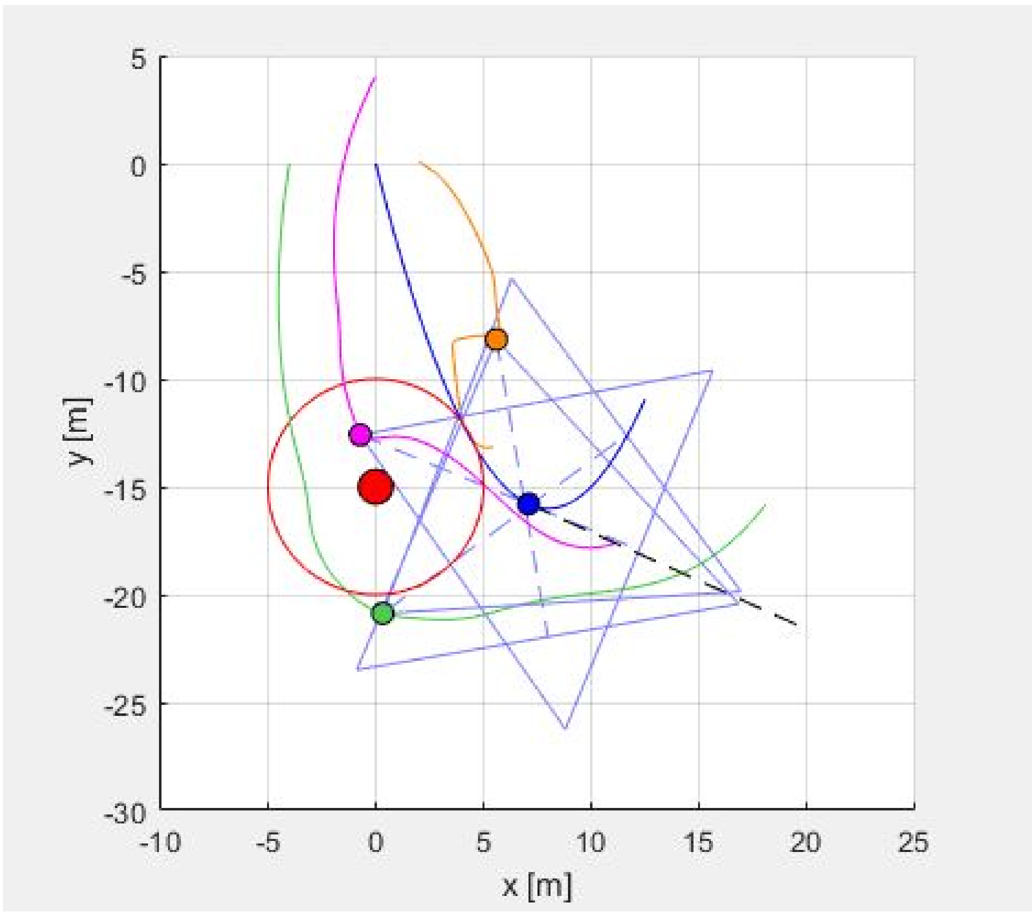


Figure 17: Obstacle avoiding trajectories, being the blue dot the leader.

6.5.2 Obstacle Avoidance: UAV to UAV

The approach used to make sure that the quadrotors do not collide among them is very similar to the one used in the previous section. In this case, we define d_{ij} as the distance vector between agent i and j. We are going to build a potential field such that each agent can not get inside others agents' safety region.

$$U_{ij} = -\frac{K_a}{2}\left(\frac{1}{d_{ij} - R_{agent_i}} - \frac{1}{R_{agent_j}}\right)^2, \quad d_{ij} \in R^2 \tag{36}$$

From which we get the collision avoidance terms:

$$[f_x^j, f_y^j] = \begin{cases} -K_a\left(\frac{1}{|d_{ij}| - R_{agent_j}} - \frac{1}{R_{agent_j}}\right)\frac{d_{ij}}{|d_{ij}|^3}, & if\ d_{ij} \leq R_{agent_i} + R_{agent_j} \\ 0, & else \end{cases} \tag{37}$$

$$\widetilde{M}_\theta = \widetilde{M}_\theta + \sum_{j=1, i\neq j}^{j=N+1} f_x^j, \quad \widetilde{M}_\phi = \widetilde{M}_\phi + \sum_{j=1, i\neq j}^{j=N+1} f_y^j \tag{38}$$

6.5.3 Field of View Avoidance

The main purpose of the Field of View Avoidance is to make sure that none of the agents gets in one of the other agent's field of view so the camera shots are useful in a cinematographic environment. To do so, we are going to build another potential field which is going to affect the yaw control input. The variables needed to formulate it can be seen in the following figure (Fig. 18).

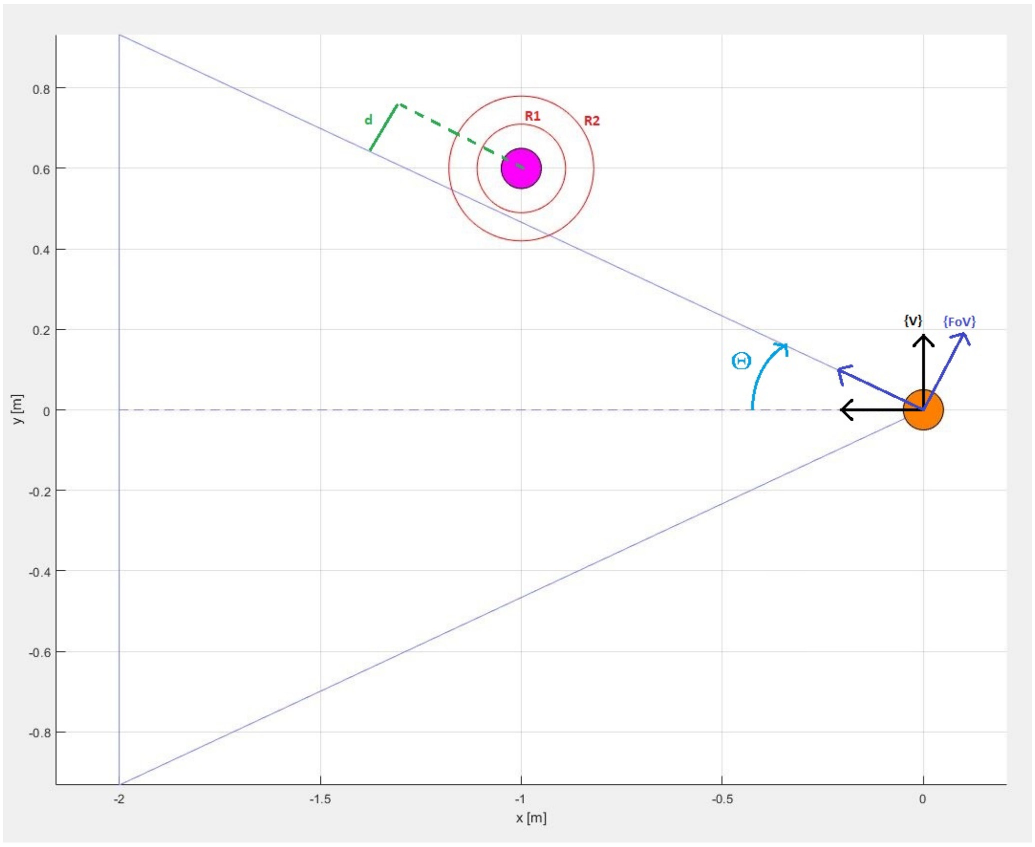


Figure 18: Field of View collision avoidance variables.

As it can be seen, we will need the perpendicular distance between the field of view and the other agents d, the camera angle Θ, the viewer agent frame V and the field of view frame FoV. Moreover, we are going to define two safety regions for each drone R_1 and R_2. The potential field is going to start acting when the field of view crosses the first region and has to make sure that it doesn't cross the second one. This is achieved with the following field:

$$U_\varphi = -\frac{K_\varphi}{2}\left(\frac{1}{d_j - R_1} - \frac{1}{R_2 - R_1}\right)^2, \quad d_j \in R \tag{39}$$

Being $P_v = [P^W_{vx}, P^W_{vy}]$ and $P_j = [P^W_{jx}, P^W_{jy}]$ the positions of the viewer agent and agent j in the world frame respectively and φ_v the yaw angle of the viewer angle. In order to obtain d, which is equivalent to the value of the y-coordinate of the agent j position in the FoV frame P_j^{FoV}, we are going to do:

$$P_j^v = rot(\varphi_v)(P_j^W - P_v^W) \tag{40}$$

$$P_j^{FoV} = rot(\Theta sign(P_{jx}^v))P_j^v, \quad d_j = P_{jy}^{FoV} \tag{41}$$

From which we get the collision avoidance term:

$$f_\varphi^j = \begin{cases} -K_\varphi \left(\frac{1}{d_j - R_1} - \frac{1}{R_2 - R_1}\right)\frac{1}{d^2}, & if\ d_j \leq R_2 \\ 0, & else \end{cases} \tag{42}$$

$$\widetilde{M}_\varphi = \widetilde{M}_\varphi + \sum_{j=1, i\neq j}^{j=N} f_\varphi^j \tag{43}$$

To prove the effectiveness of the implemented method we are going to simulate the system turning on the field of view collision avoidance. The simulation parameters are going to be the same that were used in Section 6.4.2, with a change in the formation:

N	d_x	d_y	d_z
1	-0.75	0.545	0.0
2	-1.00	-0.575	0.0
3	0.00	-0.500	0.0

The results of this simulation can be seen in the following figure (Fig. 19) and in the provided videos ($fov\_avoidance\_2D.avi$).

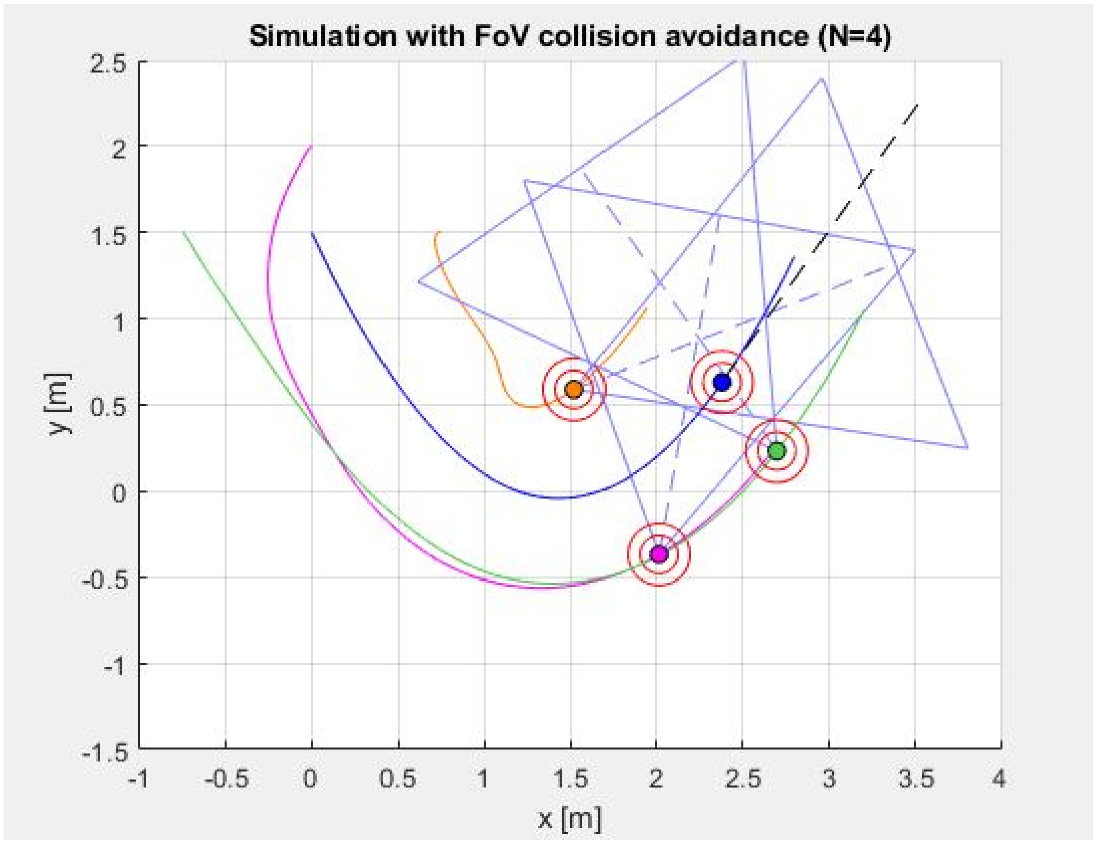


Figure 19: Yaw control avoiding other agents, being the blue dot the leader.

It can be observed how the agents' yaw are not directly pointing to the leader in order to avoid having one of the other agents inside their field of view.

6.6 Simulation Results: Full System

At this point we can already have the consensus algorithm working together with all the collision avoidance techniques. To prove the effectiveness of the all the implemented method we are going to simulate the system turning on all the collision avoidance and adding two fixed obstacles. The simulation parameters are going to be the same that were used in the previous subsection. The results of this simulation can be seen in the following figure (Fig. 20) and in the provided videos (*all_avoidance_2D.avi* and *all_avoidance_3D.avi*).

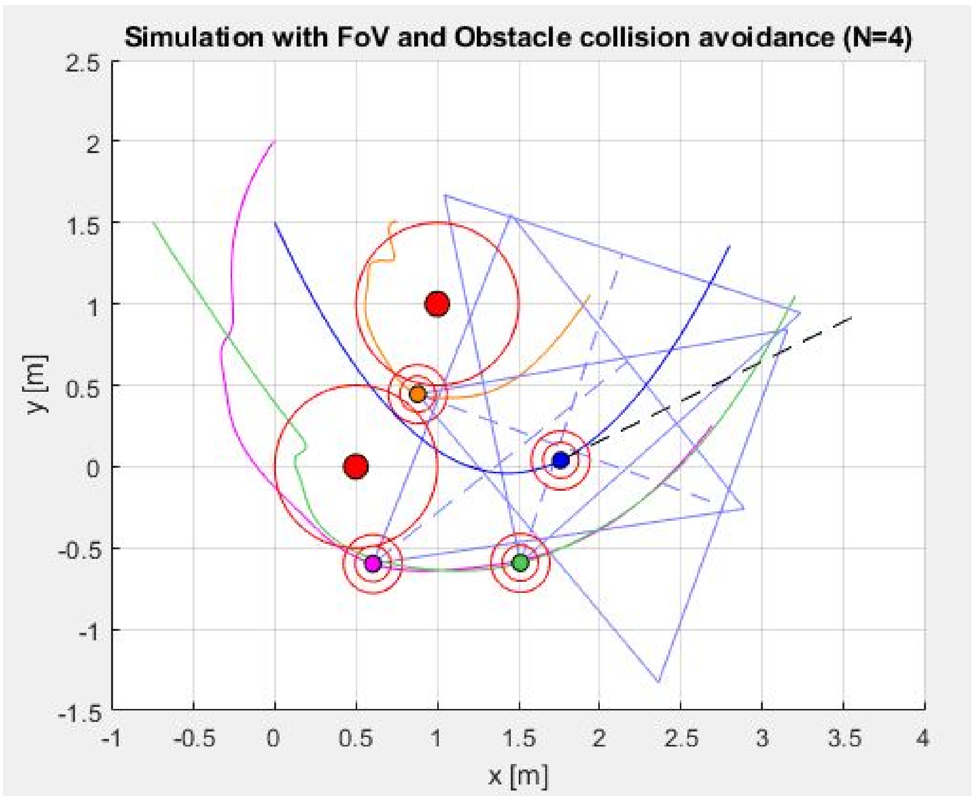


Figure 20: All control techniques together simulation, being the blue dot the leader.

7 Mambo Drone

We have been able to show how all the implemented control techniques are working for the linearized system. However, it is important to prove that these would also work on a nonlinear system, with dynamics as close as possible to the real one, in order to move to the next step, which is implementing it into a real machine. To achieve that we are going to use the *Mambo Drone* created by *Parrot Drones* company. This drone does not only have a lot of useful sensors for controlling it, but is also provided with plenty of support in Matlab/Simulink, which is a tool we have been using previously. *Parrot Drones* developed a Simulink Package that simulates the *Mambo Drone* being controlled by a PID Controller. We are going to use this model and modify it accordingly to have a simulation of the drone controlled by the techniques we have previously designed.

Figure 21: Mambo Parrot Drone.

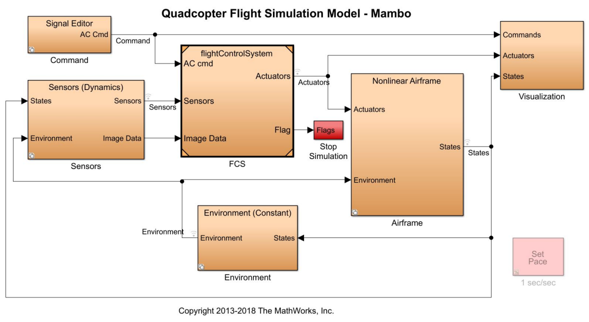


Figure 22: Vanilla Simulink Model (by *Parrot Drones*).

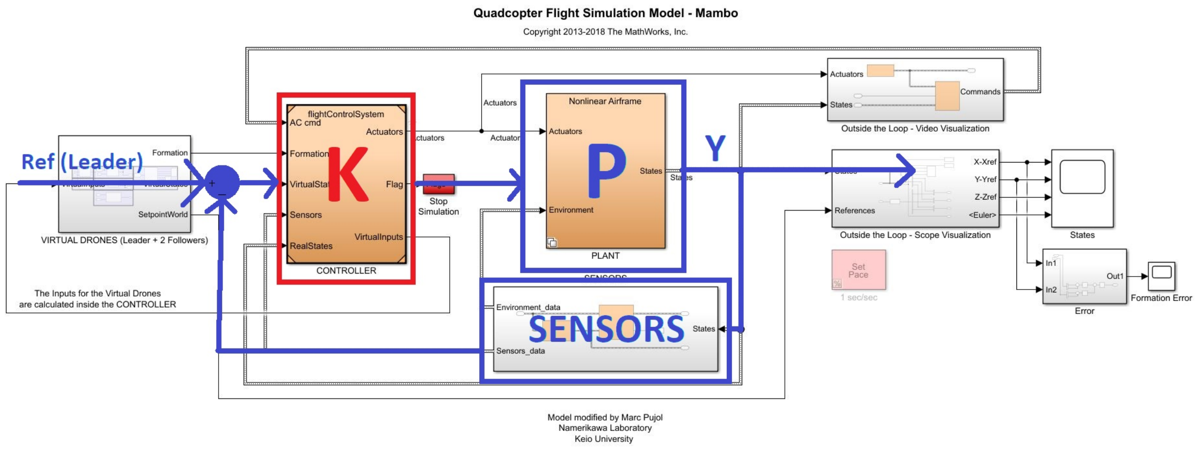


Figure 23: Modified Simulink Mambo Model (with Consensus Algorithm Controller).

7.1 Sensors

7.1.1 Hardware

Each Mambo drone is provided with various sensors to make sure that it can be stabilized and controlled. Specifically it has:

- **Inertial Measurement Unit (IMU)**, which is formed by a 3-axis accelerometer and a 3-axis gyroscope. With these two devices we can know the velocities and the orientation of the quadcopter.
- **Ultrasound Sensor and Pressure Sensor**: these two are used to assure a good stabilization in the vertical axis. The ultrasound sensor is facing the floor so it can know the distance to it.
- **Camera**: the camera is also facing the floor and is used to have a better hovering stability. The difference between camera frames is used to estimate the movement of the drone in the horizontal plane.

7.1.2 Software

The sensors signals are treated before being used by the controllers in order to get the correct information. The sensor preprocessing for the Accelerometer and Gyroscope consists in:

- **Bias Substraction**.
- **Frame Change** from the IMU Frame to the quadrotor Body Frame.
- **Low Pass Filter** to eliminate high frequency noise.

For the Ultrasounds Sensor, the only preprocessing needed is a **Bias Substraction**.
After the sensor preprocessing is completed, some filters are applied to the signals in order to obtain the states of the quadrotor that can not be measured and to enhance the precision of the measurements.

- A **Kalman Filter** is used to obtain the position and the linear velocities of the drone. This filter uses the measurements from the Gyroscope and Accelerometer together with the linearized Mambo system.
- A **Complementary Filter** is used to obtain a more precise measurement of the orientation using the data provided by the Gyroscope and the Accelerometer.

7.2 Control Inputs Calculation

As it can be seen, with the information gathered by the sensors and the sensor fusion, we are able to obtain all the states of the system. The data coming from the sensors (gyroscope and accelerometer) combined with the Complementary Filter and Kalman Filter are going to provide us with the linear and angular velocities (u, v, w, q, p, r) as well as the position and orientation $(x, y, z, \phi, \theta, \varphi)$. However, given that the consensus algorithm is using the linearized system to calculate the control inputs, we can not feed the controller directly with the output of the sensors, but the modified states, which are the ones stated in 12, 13 and 15. By using these variables we are going to obtain a consensus algorithm control input suitable for the linearized system, but in this case we are using the nonlinear Mambo model, which will need the nonlinear inputs that can be calculated with the following formulas:

$$M_\theta = -\frac{I_y}{g}\widetilde{M}_\theta, \quad M_\phi = \frac{I_x}{g}\widetilde{M}_\phi, \quad M_\varphi = I_z\widetilde{M}_\varphi, \quad T_{total} = m(g - \widetilde{T}_{total}) \tag{44}$$

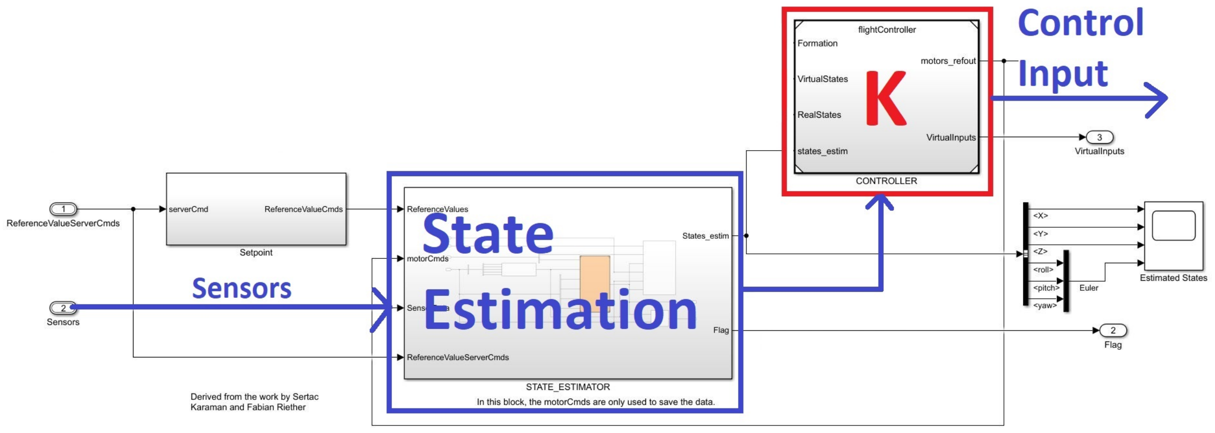


Figure 24: Controller Block of the Modified Simulink Mambo Model.

8 Mambo Simulation

8.1 Static Leader Simulation (Position State Estimation)

For the first simulation of the system we are going to assume:

- **Fix Leader**, the leader will remain in place [0,0,-5].
- **Formation = 0**, all the agents have to converge to the leader's position.
- **N+1 = 4**, a total of 3 follower agents and 1 leader. However, in this case, one of the followers is going to be a Mambo Drone, which means that we have:
  - 1 Leader (moving object).
  - 2 "Virtual" Followers, which have linearized dynamics.
  - 1 Mambo Parrot (with non linear dynamics).
- **Laplacian:**

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The obtained results can be observed in the following figure (Fig. 25), which shows a steady state error in position. This problem has its origin in the state estimation of the position, which accumulates error over time when the drone is moving.

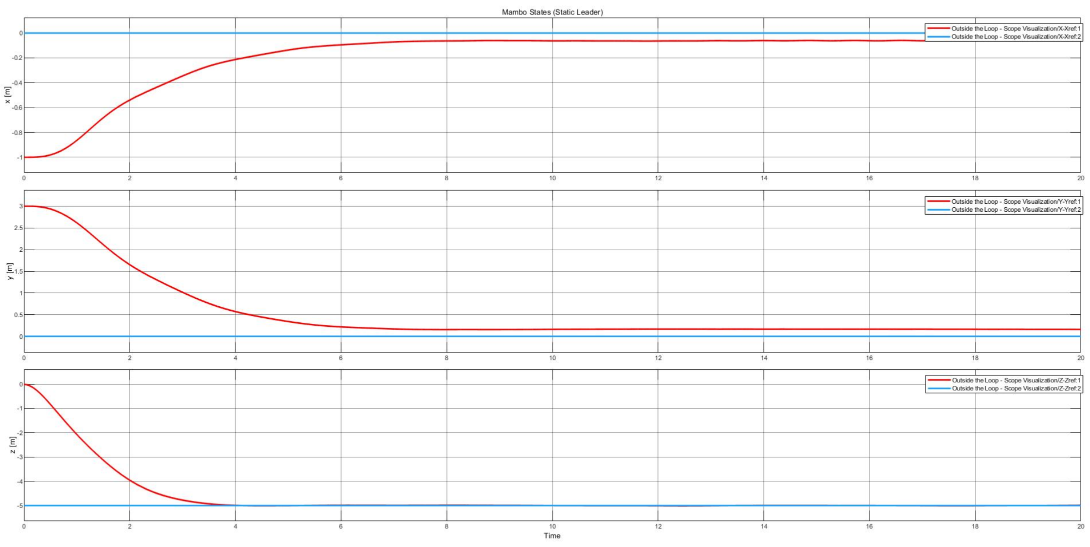


Figure 25: Mambo Drone states simulation with a fixed leader.

8.2 Static Leader Simulation (Motion Capture Position)

In order to solve the problem observed in the last simulation we need to assume that we have a Motion Capture system which gives the controller an accurate and precise reading of the position and attitude $(x, y, z, \phi, \theta, \varphi)$.

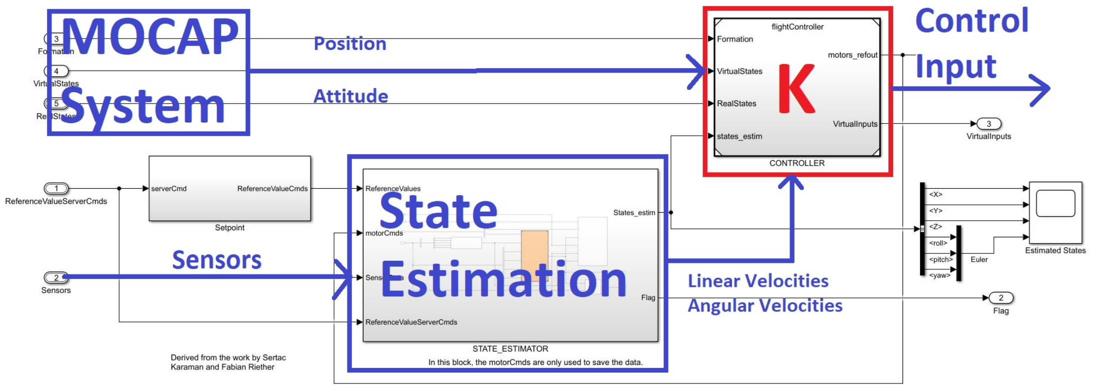


Figure 26: Controller Block of the Modified Simulink Mambo Model with MOCAP System.

For this second simulation we are going to use the same parameters used in 8.1. It can be observed in the next figure (Fig. 27) how there is no longer a steady state error as the Motion Capture system can provide accurate readings of the position.

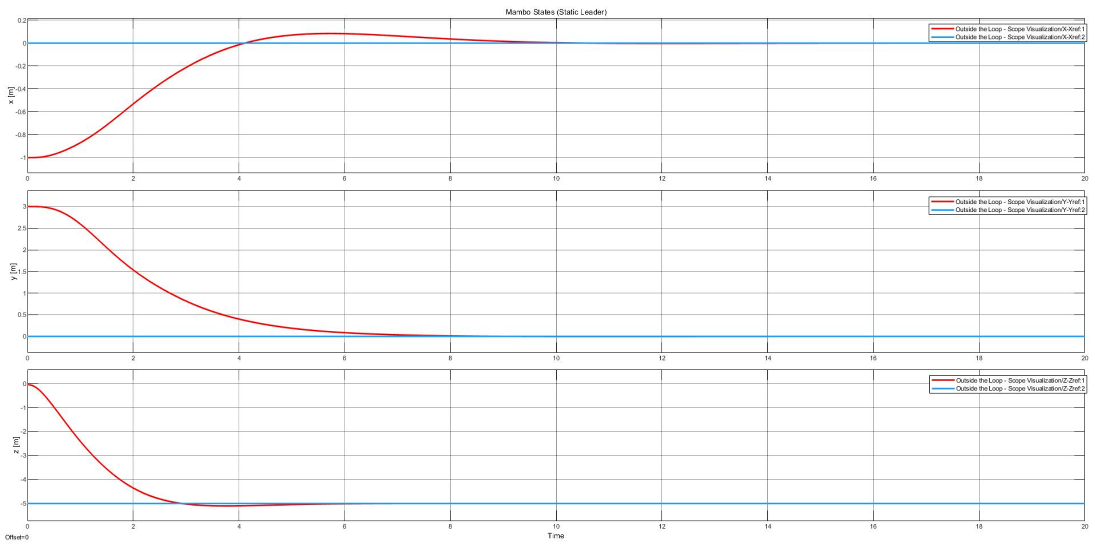


Figure 27: Mambo Drone states simulation with a fixed leader and MOCAP System.

8.3 Moving Leader Simulation (Motion Capture Position)

Lastly, we are going to simulate the system when the leader is a moving object. To do so we are going to assume the same simulation parameters we choose in 8.1 but with a different formation which looks like:

N	d_x	d_y	d_z
1	-0.75	-0.545	0.0
2	0.00	0.500	0.0
3	-1.00	0.575	0.0

The results of this simulation can be seen in the following figure (Fig. 29) and in the provided videos ($Mambo\_2D.avi$ and $Mambo\_3D.avi$).

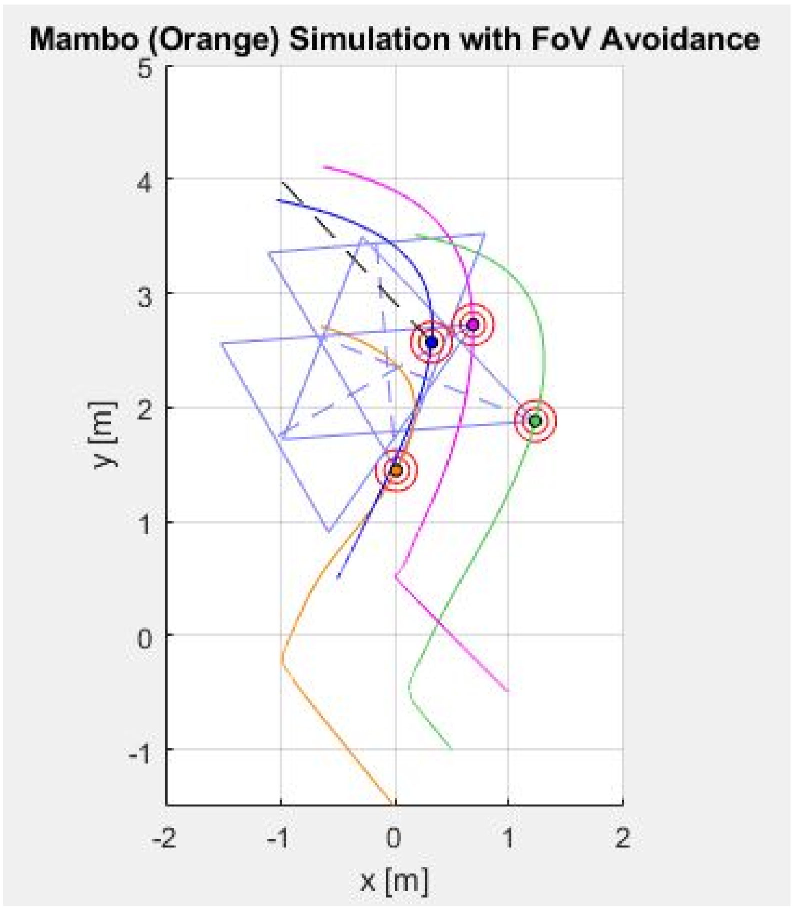


Figure 28: Multi quadrotor simulation with a Mambo Drone (Orange).

9 Mambo Real Machine Implementation (Future Work)

For future work we would like to prove the work that has been done by implementing it in a real world's machine. To do so, we have to be able to gather all the information from the drone's sensors and the Motion Capture system in a single controller. With this data, we can calculate the necessary control inputs using the proposed algorithms and then they need to be sent to the quadcopters. These are the tools we are provided:

- **Parrot Minidrones Support from Simulink**. By connecting the Mambo Drone to the computer via Bluetooth, we can deploy a Simulink model to it. With that, we can compile a controller on the Mambo board, or any other model with the desired behaviour.
  This package also has **Communications Protocol** support, which means we can easily set up UDP or TCP/IP communications.

- **OptiTrack Hardware and Motive Software**. With this Motion Capture System we can recognize the real-time position and attitude of the drones. A Visual Studio solution is used to turn on the drones and switch from manual to automatic control.

The solution we propose is the following:

- Deploy a Simulink Model to the Parrot Mambo board which creates an UDP communication that:
  - Sends the gyroscope and accelerometer sensor data.
  - Receives the Motor Thrust commands.
- Use the OptiTrack system to obtain the real-time positioning data from the drone in Visual Studio. In the same solution, set up an UDP communication to send the position and attitude readings.
- Use a simulink model that:
  - Sets up the UDP clients to receive data from the Mambo sensors and Visual Studio.
  - Using the received data calculates the necessary control inputs using the proposed algorithms.
  - Sets up the UDP communication to send the inputs (Motor Thrust Commands) to the Mambo Drone.

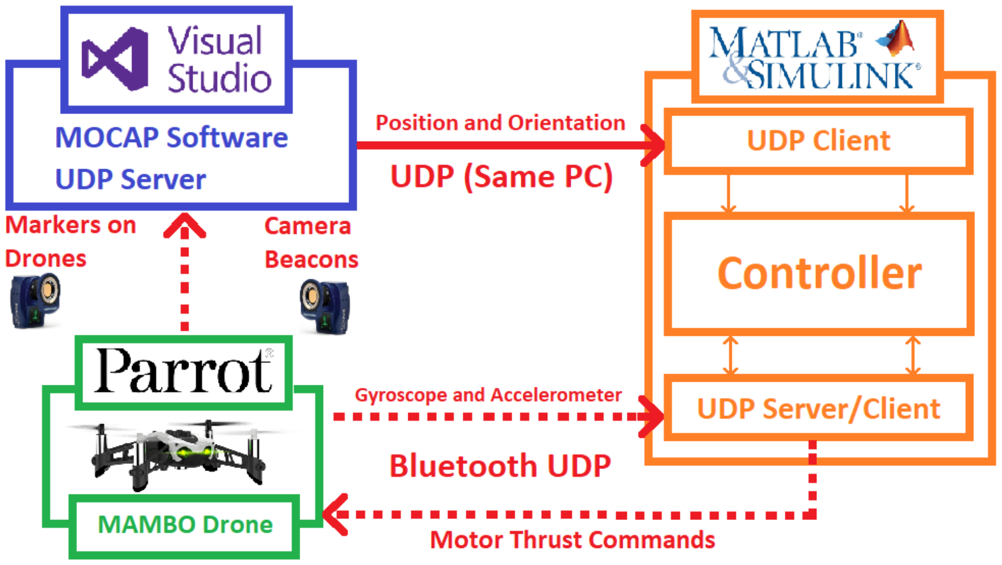


Figure 29: Mambo Real Machine Implementation Example.

10 Conclusions

We have demonstrated how to implement a Consensus-Based Cooperative Control for a group of quadrotors in order to make them maintain a formation around a moving object. Moreover, we have extended this algorithm to control the yaw angles of the UAVs so they are continuously recording the moving object. It has also been proven by simulation that both position and yaw control can be further enhanced by collision avoidance techniques without losing much of the performance provided by the consensus algorithm. It is important to note for this research work that having the quadrotors connected through a network enables the possibility of Field of View collision avoidance, which is very important for the shot recording feature. It has also been proven that all the techniques are suitable for a real machine such as the *Mambo drone* by using the proposed control in the drone model, provided by the machine developers, in a simulation environment. However, due to lack of time and hardware problems, the real machine experiments have yet not been conducted. For future work we propose implementing the system stated in 9, which is an intuitive solution for real drones experiments. Moreover, future work can be done in order to make this system more suitable for the end-user by, for example, enhancing the behaviour of the camera like they do in [7], where a tracking system is proposed for the gimbal camera attached to the drone in order to behave regarding on high-end commands.

References

[1] Toru Namerikawa, Yasuhiro Kuriki and Ahmed Khalifa, *Consensus-Based Cooperative Formation Control for Multiquadcopter System With Unidirectional Network Connections.*
Journal of Dynamic Systems, Measurement, and Control APRIL 2018, Vol. 140.

[2] Marc Ille and Toru Namerikawa, *Collision Avoidance between Multi-UAV-Systems considering Formation Control using MPC.*
2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM).

[3] Beard Randal, *Quadrotor Dynamics and Control Rev 0.1.*
2018 *All Faculty Publications. 1325.*

[4] Tobias Nägeli, Javier Alonso-Mora, Alexander Domahidi, Daniela Rus, and Otmar Hilliges, *Real-Time Motion Planning for Aerial Videography With Dynamic Obstacle Avoidance and Viewpoint Optimization.*
IEEE Robotics and Automation Letters, Vol. 2, No. 3, July 2017.

[5] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora and Otmar Hilliges, *Real-time Planning for Automated Multi-View Drone Cinematography.*
ACM Transactionson Graphics, Vol. 36, No. 4, Article 132, July 2017.

[6] Julien Fleureau, Quentin Galvane, Francois-Louis Tariolle and Philippe Guillotel, *Generic Drone Control Platform for Autonomous Capture of Cinema Scenes.*
DroNet16 June 26, 2016.

[7] Chin E. Lin and Sheng-Kai Yang, *Camera Gimbal Tracking from UAV Flight Control.*
2014 *International Automatic Control Conference (CACS 2014).*

[8] Hyo-Sung Ahn, Byung-Hun Lee, Kwang-Kyo Oh, Kyungmin Jeong, *orientation Alignment-based Formation Control with Reference Node.*
2015 *Proceedings of the 34th Chinese Control Conference.*

[9] Chong Huang, Fei Gao, Jie Pan et al., *ACT: An Autonomous Drone Cinematography System for Action Scenes.*
2018 *IEEE International Conference on Robotics and Automation (ICRA).*

[10] Ioannis Mademlis, Nikos Nikolaidis, Anastasios Tefas et al., *Autonomous Unmanned Aerial Vehicles Filming ind Dynamic Unstructured Outdoor Environments.*
2019 *IEEE Signal Processing Magazine.*

Annex I - Mambo Simulink KnowHow

In this document we would like to transfer all the knowledge about the *Simulink Support Package for Parrot Minidrones* that has been gathered during the confection of the Master's Thesis *Multi Shot Recording using Consensus-Based Cooperative Control for a Multiquadrotor System*.
The Simulink Package is provided by the company *Parrot Drones* which manufactures different kind of drones. We found this package very useful as it contains an accurate model of one of their drones, the *Mambo* Drone, which makes the simulations as close to reality as possible. Moreover, this model is suitable for any future work as, in case that we want to change the control law, the only piece of that needs to be modified is the controller block, leaving the system's model untouched.
This document is going to go through all the necessary points to be able to understand the characteristics of the Simulink Package, as well as the parts of the code that are important when facing a control problem. Moreover, we are going to provide some modified versions of the Vanilla Model which will help in the development of different control techniques, for example, consensus algorithm.

– **The Mambo Model: Sensors and Dynamics**

– **The Mambo Project**

– **Mambo Vanilla Control: PID**

– **First Modified Model: PID**

– **Second Modified Model: Consensus Algorithm**

– **Third Modified Model: Consensus Algorithm with FoV Avoidance**

The Mambo Model: Sensors and Dynamics

Model Dynamics

The model that simulates the dynamics of the *Mambo* drone is formed by 12 states:

– Position and Orientation: x, y, z and θ, ϕ, φ.

– Linear and Angular velocities: u, v, w and p, q, r.

The model also has 14 inputs, but only 4 of them are control inputs. The other 10 are **Environment Paremeters** that we can consider constant if working in the laboratory:

– Air Temperature: 283 $[K]$.

– Air Pressure: 1 $[atm]$.

– Air Desnity: 1.184 $[\frac{kg}{m^3}]$.

- Speed of Sound: 340 $[\frac{m}{s}]$
- Gravity: [0, 0, 9.81] $[\frac{m}{s^2}]$
- Magnetic Field: [0, 0, 0] $[T]$

The 4 **Control Inputs** are the four **Motor Commands** M_1, M_2, M_3, M_4.
However, the output signals of the controller are the **Moment Commands**: $T_{total}, M_\varphi, M_\theta, M_\phi$.
To get the desired inputs out of the outputs of the controller we need to do some transformations. First of all, the manufacturer gives use the following parameters related to the motors and the airframe geometry:

- $d = 0.0441\ [m]$
- $r_\varphi = 0.0024\ [m]$
- $K_M = 1530.7\ [\frac{\frac{rad}{s}}{N}]$

With these, we can first go from the Moment Commands to Thrust Commands. We need to define the Thrust to Moment Matrix (**Ts2Q**):

$$\begin{bmatrix} T_{total} \\ M_\varphi \\ M_\theta \\ M_\phi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ r_\varphi & -r_\varphi & r_\varphi & -r_\varphi \\ -d & -d & d & d \\ -d & d & d & -d \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

With that we can get:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = inv(Ts2Q) \begin{bmatrix} T_{total} \\ M_\varphi \\ M_\theta \\ M_\phi \end{bmatrix}$$

Finally, to obtain the Motor Commands:

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = -K_M \begin{bmatrix} T_1 \\ -T_2 \\ T_3 \\ -T_4 \end{bmatrix}$$

This Motor Commands absolute value are saturated to $\in [10, 500]$.

Sensors

In order to obtain all the data related to the state of the drone, it is provided with several sensors:

- **Inertial Measurement Unit (IMU)**, which is formed by a 3-axis accelerometer and a 3-axis gyroscope. With these two devices we can know the velocities and the orientation of the quadcopter.

- **Ultrasound Sensor and Pressure Sensor**: these two are used to assure a good stabilization in the vertical axis. The ultrasound sensor is facing the floor so it can know the distance to it.

- **Camera**: the camera is also facing the floor and is used to have a better hovering stability. The difference between camera frames is used to estimate the movement of the drone in the horizontal plane.

The sensors signals are treated before being used by the controllers in order to get the correct information. The sensor preprocessing for the Accelerometer and Gyroscope consists in:

- **Bias Substraction**.

- **Frame Change** from the IMU Frame to the quadrotor Body Frame.

- **Low Pass Filter** to eliminate high frequency noise.

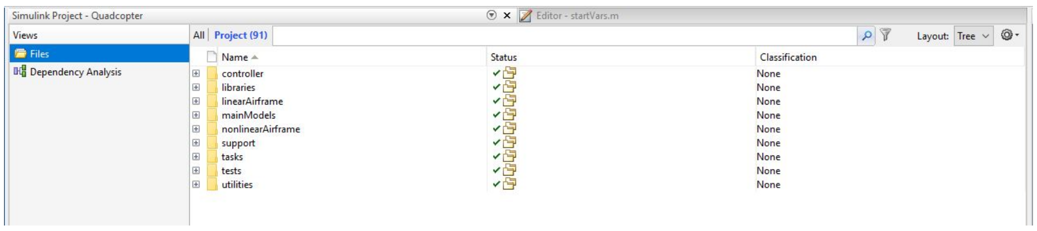
For the Ultrasounds Sensor, the only preprocessing needed is a **Bias Substraction**.
After the sensor preprocessing is completed, some filters are applied to the signals in order to obtain the states of the quadrotor that can not be measured and to enhance the precision of the measurements.

- A **Kalman Filter** is used to obtain the position and the linear velocities of the drone. This filter uses the measurements from the Gyroscope and Accelerometer together with the linearized Mambo model.

- A **Complementary Filter** is used to obtain a more precise measurement of the orientation using the data provided by the Gyroscope and the Accelerometer.

The Mambo Project

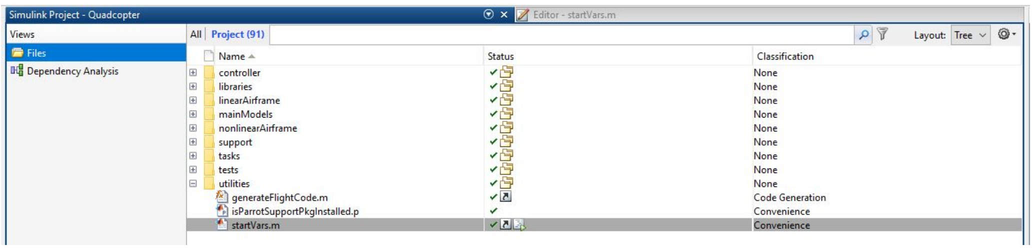
In this section we are going to explain the most important parts of the Mambo project in order to understand how it works so we can modify it later as we please. In order to create and open the Mambo project in *Matlab* we need to introduce **asbQuadcopterStart** in the Command Window. This will open a *Simulink* Model, as well as the project. In this section we are only going to refer to the project.



Name	Status	Classification
controller	✓	None
libraries	✓	None
linearAirframe	✓	None
mainModels	✓	None
nonlinearAirframe	✓	None
support	✓	None
tasks	✓	None
tests	✓	None
utilities	✓	None

Project window in Matlab.

When we open the project, it automatically runs the initializer script which is **startVars.m** located inside **utilities**.



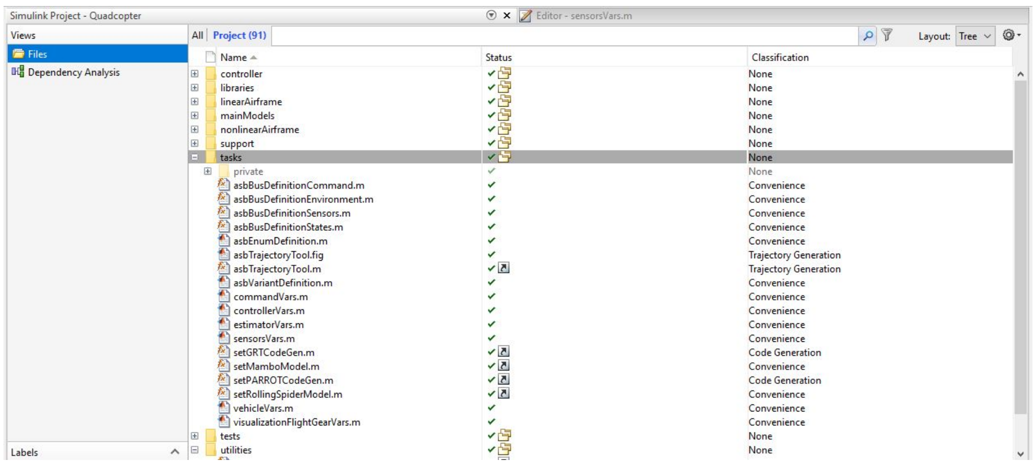
Name	Status	Classification
controller	✓	None
libraries	✓	None
linearAirframe	✓	None
mainModels	✓	None
nonlinearAirframe	✓	None
support	✓	None
tasks	✓	None
tests	✓	None
utilities	✓	None
generateFlightCode.m	✓	Code Generation
isParrotSupportPkgInstalled.p	✓	Convenience
startVars.m	✓	Convenience

StartVars.m in the project window.

This script is in charge of things like:

- Variants Conditions initialization, which sets whether we want to work with the linear or nonlinear system, with noise in the sensors, or how we want to visualize the results.
- Runs the bus initializer scripts.
- Defines the sampling time.
- Sets the initial conditions for the drone states.
- Runs the other features' initializers (sensors, controller, estimator...).

StartVars.m runs other necessary scripts for the model to work. These are located inside the **tasks** folder:



Name	Status	Classification
controller	✓	None
libraries	✓	None
linearAirframe	✓	None
mainModels	✓	None
nonlinearAirframe	✓	None
support	✓	None
tasks	✓	None
private	✓	None
asbBusDefinitionCommand.m	✓	Convenience
asbBusDefinitionEnvironment.m	✓	Convenience
asbBusDefinitionSensors.m	✓	Convenience
asbBusDefinitionStates.m	✓	Convenience
asbEnumDefinition.m	✓	Convenience
asbTrajectoryTool.fig	✓	Trajectory Generation
asbTrajectoryTool.m	✓	Trajectory Generation
asbVariantDefinition.m	✓	Convenience
commandVars.m	✓	Convenience
controllerVars.m	✓	Convenience
estimatorVars.m	✓	Convenience
sensorsVars.m	✓	Convenience
setGRTCodeGen.m	✓	Code Generation
setMamboModel.m	✓	Convenience
setPARROTCodeGen.m	✓	Code Generation
setRollingSpiderModel.m	✓	Convenience
vehicleVars.m	✓	Convenience
visualizationFlightGearVars.m	✓	Convenience
tests	✓	None
utilities	✓	None

Tasks folder in the project window.

The ones that have importance for us are:

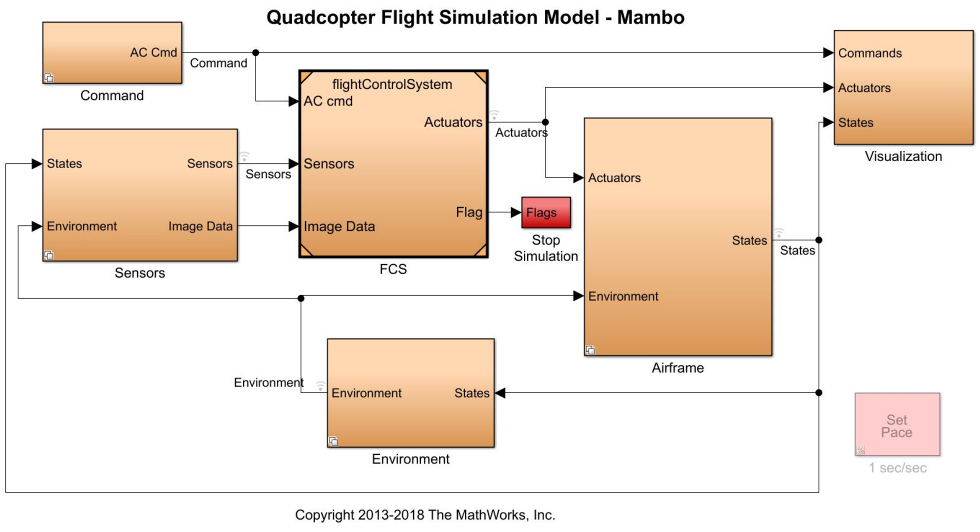
- **asbBusDefinition___.m**: These functions define the sizes of the buses and the names of each signal. If we want to modify the buses in the model we need to refer to these functions and change them.

- **___Vars.m**: These scripts initialize the parameters of the controller, the vehicle, sensors... In case we want to conduct a physical study of the drone, refer to **VehicleVars.m** where all the variables of the real airframe are stored.

The rest of the folders contain the libraries that form the model.

Mambo Vanilla Control: PID

The Simulink Model provided can be seen in the following figure:



Vanilla Simulink Model for the Mambo Drone.

The three main blocks that form this model are:

– **Sensors**: Which simulate the readings of the sensors, feeding them to the fligt controller.

– **fightControlSystem**: Which contains the state estimation that uses the sensors readings and the flight controller. In this case, it is a PID Controller.

– **Airframe**: Which contains the drone dynamics to simulate its behaviour.

If we look inside the **fightControlSystem** block we will observe that the control has the following characteristics:

– **Alitude Control**. It uses the Z position coming from the state estimation. Then calculates the error as $Error = Z_{est} - Z_{ref}$ with the reference point given. The control technique is a PD in this case.

– **Horizontal Control**. It is based on two different closed-loops:

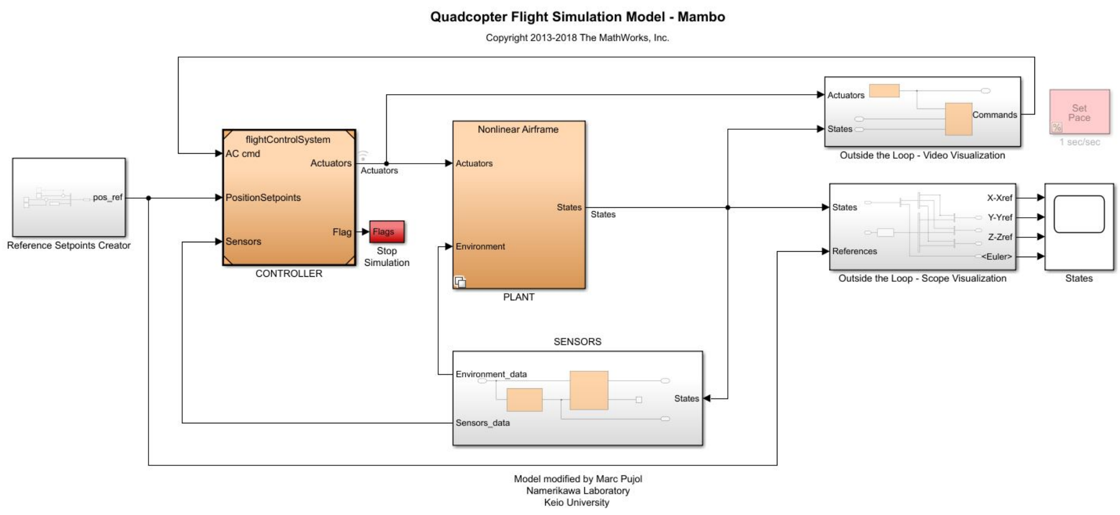
  * Outer Loop. It uses the X and Y position coming from the state estimation. Then calculates the error as $Error_{OL} = X, Y_{est} - X, Y_{ref}$ with the reference point given. The control technique is a PD in this case. Moreover it has a saturation of $\in [-3, 3]$.

  * Inner Loop. The result from the output Loop is fed into the inner loop uses θ and ϕ coming from the state the state estimation. Then calculates the error as $Error_{CL} = \theta, \phi_{est} - Error_{OL}$. The control technique is a PID in this case.

In order to make this model more intuitive we are going to make some changes which are going to help us further modify the system in the future, if needed.

First Modified Model: PID

This model can be found in the folder *V1.0 Modified Mambo PID*.



First Modified Simulink Model for the Mambo Drone.

The modifications that have been done are the following:

– **Remove Image data** in the controller. This block is only used for precision landing, thing that we don't need.
asbQuadcopter/flightControlSystem.

– **Remove Orientation Control** in the controller and reorganize the blocks. We are only going to use the position control.
asbQuadcopter/flightControlSystem/FlightController.

– **Modify PID Gains**. The PID provided by the manufacturers was not correctly tuned.

  ∗ $K_{P\phi} = 0.013$ and $K_{D\phi} = 0.002$. The original values were causing oscilations.
  asbQuadcopter/flightControlSystem/FlightController/XY-to-reference-orientation.

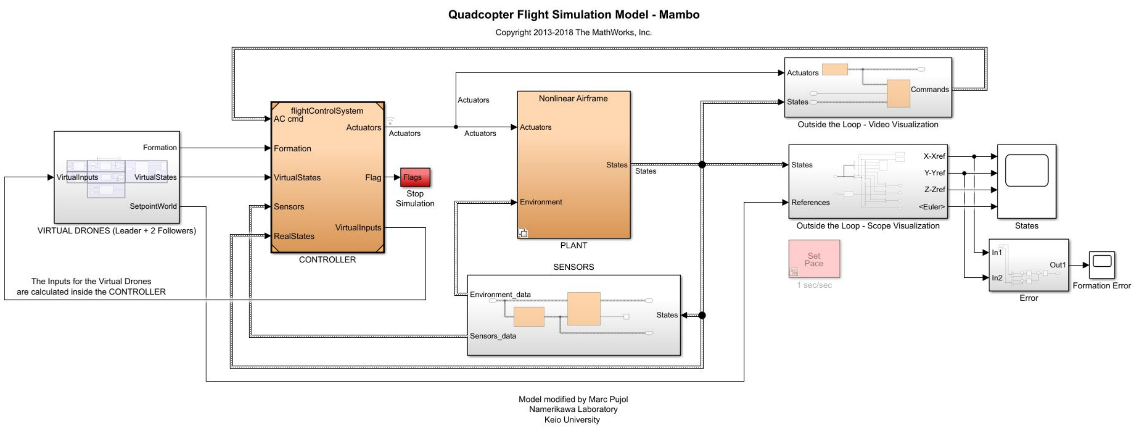
  ∗ $K_{Px,y} = 0.12$. The original value created a huge overshot.
  asbQuadcopter/flightControlSystem/FlightController/Attitude.

– **Reference Setpoints Modification**. Changed the setpoints to step signals, that can be tuned before running the simulation.
asbQuadcopter/flightControlSystem/SetpointCreation.

– **Signal Editor Modification**. Changed so it sets automatically the control mode to position. This block is no longer in charge of the reference trajectory points creation.
asbQuadcopter/Outside the Loop - Video Visualization/Command/Signal Editor.

Second Modified Model: Consensus Algorithm

This model can be found in the folder *V2.1 Modified Mambo Consensus*.



Second Modified Simulink Model for the Mambo Drone.

The modifications that have been done are the following:

– **Added Virtual Drones**. As now we want to work with Consensus Algorithm we need to simulate more drones. Inside this new block three new drones are created using the linearized model (2 Followers and 1 Leader). The states are then fed into the Mambo controller which will calculate the control inputs for all the drones. *asbQuadcopter/VIRTUAL DRONES (Leader + 2 Followers)*.

– **Consensus Algorithm Control**. The control input calculation has been changed to the desired consensus algorithm control. *asbQuadcopter/flightControlSystem/FlightController*.

– **startConsensus.m** added to the Project. This script defines the linearized system of the drones, the desired formation and the initial conditions of the team. It is automatically run when the Project is opened (via **startVars.m**). *utilities/startConsensus.m*.

– **Assume Motion Capture System**. As we have proven that the state estimation for the position is not precise enough, we assume we are using a MOCAP System. To do so, we feed the real states of the Mambo directly to the controller. Further simulations should be done adding noise to this measurement. *asbQuadcopter/flightControlSystem/FlightController*.

Third Modified Model: Consensus Algorithm with FoV Avoidance

This model can be found in the folder *V4.0 Modified Mambo Consensus with FoV*. The modifications that have been done are the following:

– **Added FoV Collision Avoidance**. This new block is in charge of the Field of View Collision Avoidance. Its output is added to the consensus algorithm outputs. *asbQuadcopter/flightControlSystem/FlightController*.