ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

信州大学
SHINSHU UNIVERSITY

Laboratoire de Systèmes Robotiques, EPFL
And
Faculty of Textile Science and Technology, Shinshu University

Projet de Master 2017

Section de Microtechnique

Formation Control of Multiple Small Quadrotors by Using Model Predictive Control

Loïc Dubois

Professors:
Hannes Bleuler
Satoshi Suzuki

Assistant:
Romain Baud

November 4, 2016

Project Summary

Student: **Loïc Dubois**

Title: **Formation Control of Multiple Small Quadrotors by Using Model Predictive Control**

Formation control is a high-level process coordinating the motion of several units positioned along a defined shape. By implementing formation control in small unmanned aerial vehicles (UAV), new applications, unthinkable with a single UAV, arise. For example wider coverage, cooperative tasks. Furthermore, the robustness of the system is improved.

The final goal of the project is to achieve formation control of 4 drones.
The first step is to adapt the numerical simulation of one small helicopter controlled using Model Predictive Control (MPC) for several CrazyFlie quadrotors also controlled using MPC. The second step is, using the Robot Operating System (ROS) environment and the OptiTrack motion capture system, to implement formation control using a centralized control architecture. Finally, if time allows it, the formation control should be implemented in a decentralized manner.

The output of this project is a C++ control architecture, scalable and adaptable to any flying platform, conceding minor changes.

Associate Prof. Suzuki Satoshi

E-mail s-s-2208@shinshu-u.ac.jp
Telephon +81-268-21-5605
Adress 3-15-1 Tokida
Ueda, Nagano, 386-8567
Japan

SUMMARY - To be included - Template: c.f.: Project instructions, STI-MT website

Contents

Symbols and Abbreviations

Symbols

v	A scalar
$\dot{v}$	First-order time derivative of scalar v
$\ddot{v}$	Second-order time derivative scalar v
c_α	Cosinus of angle α
s_α	Sinus of angle α
$\boldsymbol{v}$	A vector in world frame
$\boldsymbol{v}_B$	A vector in body frame
$\|\boldsymbol{v}\|$	L^2 norm of vector $\boldsymbol{v}$
$\boldsymbol{v}_{i:j}$	Sub vector from composant i to j of vector $\boldsymbol{v}$
V	A matrix
$V^T, \boldsymbol{v}^T$	Transpose of a matrix V or a vector $\boldsymbol{v}$
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
$\boldsymbol{x}$	State vector
$\boldsymbol{u}$	Input vector
$\boldsymbol{x}_s$, $\boldsymbol{u}_s$	Working point subscript
$\boldsymbol{x}_r$, $\boldsymbol{u}_r$	Reference subscript
A	A matrix
B	B matrix
C	C matrix
D	D matrix

Abbreviations

MPC	Model Predictive Control
ROS	Robot Operating System
GMRES	Generalized Minimum Residual method
C/GMRES	Continuation-GMRES method
FPS	Frame Per Second
PWM	Pulse Width Modulation

List of Tables

List of Figures

1 Introduction

Build on [1]

2 Setup

WHOLE SETUP PRESENTATION The setup for this project is the same as the one used in [1].

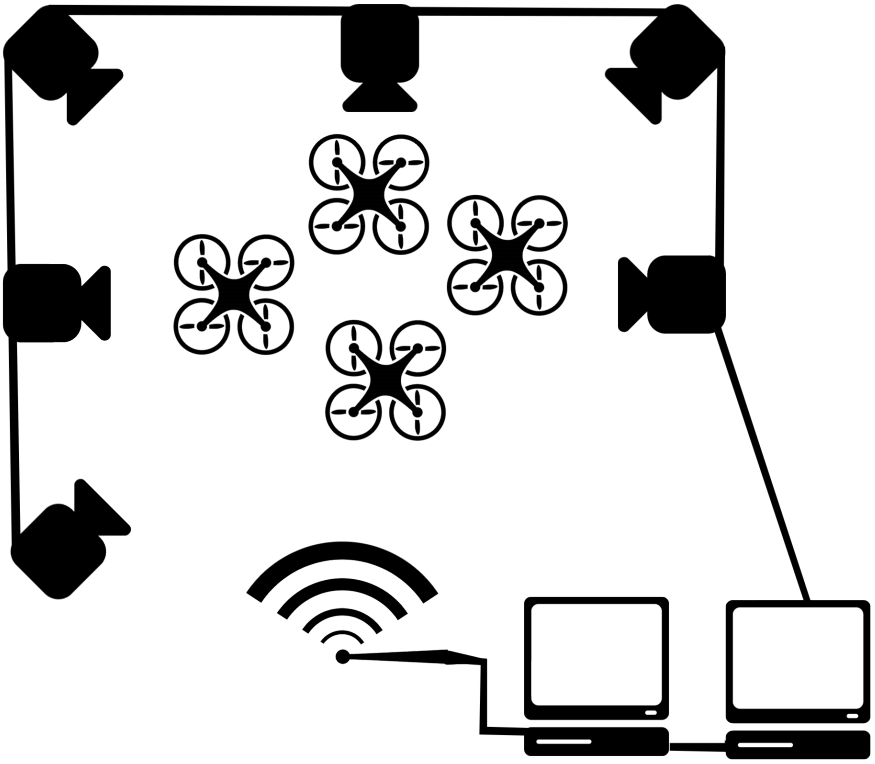


Figure 1: Overview of the complete setup

2.1 Crazyflie 2.0 Quadrotor

The Crazyflie 2.0 (or, from now on, simply CrazyFly) is the second model of small quadrotors (or quadcopter or, simply, quad) developed by Swedish company, BitCraze [2]. Its length of 92 millimeters from rotor to rotor and its weight of 27 grams make it a safe indoor flying machine. Its on-board battery allows flights up to 7 minutes. Flights that can be controlled thanks to an on-board long-range radio receiver and the appropriate emitter, the CrazyRadio PA or thanks to a Bluetooth LE (low-energy) connexion and a smartphone with the dedicated application. The quadrotor is also equipped with different sensors, including a IMU with a 3-axis high-performance MEMs gyros and accelerometers as well as a 3-axis magnetometer and a barometer.

The Crazyflie 2.0 runs on an open source firmware, available on BitCraze website[2], which makes an ideal tool for experimentation on control, estimation, navigation or algorithm validation. The code can be adapted, modified or improved without restrictions. Moreover we can log firmware variables up to 100 Hz and export them to a CSV file, which can be decrypted by Matlab.

Picture from [2]

Figure 2: The CrazyFlie quadrotor from BitCraze

Firmware

A description of the core control C modules is given in section 5.2. Note that the current firmware allows to log state and/or sensor data at a maximal rate of 100 Hz.

2.2 Robot Operating System

A ECRIRE

2.3 OptiTrack Motion Capture System

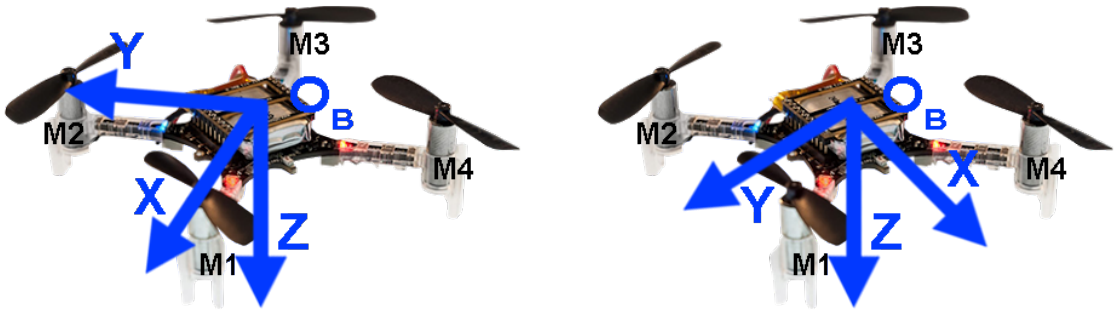
The motion capture system consists, on one side, of eight Prime 13 cameras from OptiTrack and, on the other side, Motive:Tracker, a motion analysis software also from OptiTrack, capable of performing all steps from calibration to the motion capture itself. This system allow a frame rate of 240 FPS and capture the drone position and attitude, or orientation. Specs Camera: buffer size and length.

3 Quadrotor dynamics

A quadrotor can fly in two different configurations, they are commonly called + (plus) and × (cross). In a + configuration. The X- and the Y-axes are aligned with two perpendicular arms, the control is easier because only two propellers require to have their speed changed to move along a principal axis. On the other side, a × configuration require all the propellers to change their speed. However the variation of speed per motor is important. In the following section, all dynamics equations are derived for a + configuration. To fly in ×, minor changes have to be performed.

3.1 State representation

To derive the equation of dynamics of a quadrotor, two reference frames are required, a world one (with index W or without index in the following equations) and the body one (always with index B). The body frame has its origin O_B fixed on the center of mass of the quadcopter, the X-axis pointing towards the motor M1 and the Y-axis pointing towards the motor M2, in case of + configuration. Or, in × configuration, with the X-axis pointing between the motors M1 and M4 and the Y-axis pointing between the motors M1 and M2. Therefore, in both cases, the Z-axis is pointing downwards when the quadcopter lies on the floor (c.f.: figure 3).
And the world frame origin is fixed in the center of the experiment room and its X- and Y- axis are parallel to floor. Moreover, they are oriented such that the Z-axis is pointing downwards as well.



(a) Body frame in + configuration (b) Body frame in × configuration

Figure 3: Body frames in the two main configurations

Dynamics can be explained using a 12-variables state description of the Crazyflie with the following state vector: the position of O_B with respect to the world-frame, the speed of O_B with respect to the world-frame and the attitude angles (Euler angles) and the angular speed in the body frame:

$$\boldsymbol{x} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \phi\ \theta\ \psi\ \omega_{xB}\ \omega_{yB}\ \omega_{zB}]^T$$

When the Crazyflie is at the origin of the world frame, without flying and the axes of both frames are aligned, the state is:

$$\boldsymbol{x} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$$

This is a simple model considering translations and rotations along three axes. Some more advanced models can also include also the accelerometers biases, the gyroscopes drift, or noise on the estimates, making easily a 20-or-more-states description of a drone. But this is behind the scope of this project.

Attitude Representation

The attitude can be defined as the relative orientation of the body-frame with reference to the world frame and, nowadays, several representations of the attitude an object in space [3]. Among these representations, Euler angles, rotation matrices and unit quaternions are the most used. To derive the dynamics equations, we require the first two and the last one is implemented in the firmware for attitude estimation.

As stated above, the state of the drone includes its attitude in terms of Euler angles. The angles are the values of three ordered rotations around three axes in space and the sequence of the three axes around which, each rotation is accomplished, defines the sub-representation. Officially, there is 12 valid sub-representations but two are considered as the main ones: the $(3, 1, 3)$ sequence and the $(1, 2, 3)$ sequence.

The first one, also known as the *x-convention*, is often used in the study of spinning objects. The angles are called, in order, *spin*, *nutation* and *precession*.

The second one, commonly found in aerospace engineering and computer graphics, is also often called *Cardan angles*, *Tait-Bryan angles* or *nautical angles*. In this sub-representation, angles are called often as follow *bank*, *attitude*, *heading* or *roll*, *pitch*, *yaw*.

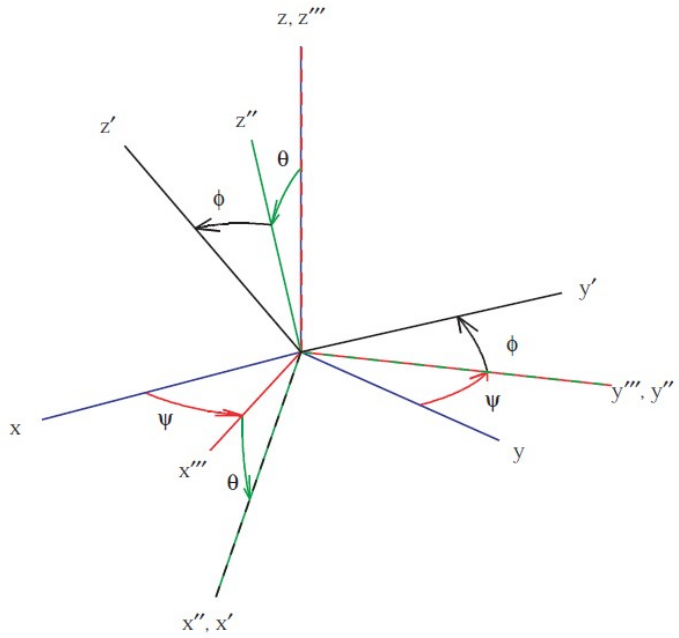
From now on, referring to Euler angles refers to the latter sub-representation. Moreover, the roll-pitch-yaw convention for angles name is preferred.

Euler angles are defined as a sequence of three rotation to map the body-frame on the world-frame or inversely. For this purpose, first a rotation around the X-axis gives the roll angle ϕ , then a rotation around the Y-axis gives the pitch angle θ and finally a rotation around the the Z-axis gives the yaw angle ψ. The order (X, Y, Z) is what gives this sub-representation its name $(1, 2, 3)$. On the figure 4, the roll angle corresponds to the rotation between the x"y"z" and x'y'z' frames, the pitch x"'y"'z"' and x"y"z" and the yaw xyz and x"'y"'z"'.

$$\boldsymbol{q} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

Although, Euler angles are easy to visualize, they suffer from singularities. In the case of a pitch angle $\theta = 90°$, variation in yaw angle are indistinguishable from variation in roll angle. However, these are extreme cases that are never met in the scope of this project.

The second convention for representing attitude is a rotation matrix. A rotation matrix maps a vector from the body frame to the world frame (or reversely) in a length-preserving manner. The



Drawing from [3]

Figure 4: Roll ϕ, Pitch θ and Yaw ψ Angles

reverse operation can be easily computed, as the all the matrix rotations are part of the special orthogonal group, their inverse is also their transpose.

$$\boldsymbol{v_B} = R_{BW}\boldsymbol{v_W}$$

$$\boldsymbol{v_W} = R_{WB}\boldsymbol{v_B}$$

$$R_{BW} = R_{WB}^{-1} = R_{BW}^{T}$$

Each rotation in a 3-dimensional space can be decomposed in three rotations around the three main axes:

$$R_{Ox}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} R_{Oy}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} R_{Oz}(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consecutive rotations can be easily combined by left-multiplying the matrices. Although its simplicity, the representation uses requires 9-states variable. Therefore, Euler angles are kept for representation and a transition to rotation matrix is performed for computation purposes.

The mapping between Euler angles and rotation matrix, consisting of the multiplication of the three individual rotation matrices in the right order, and is computed as follow (c_α corresponds to $\cos\alpha$ and s_α to $\sin\alpha$):

$$R_{BW}(\phi,\theta,\psi) = R_{Ox}(\phi)R_{Oy}(\theta)R_{Oz}(\psi) = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix}$$

Finally the last convention is the unit quaternion. The unit quaternion is a 4-dimensional generalization of complex numbers whose norm $\|\boldsymbol{q}\| = 1$.

$$\boldsymbol{q} = q_0 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ q_{1:3} \end{bmatrix}$$

They profit from most of the tools of complex algebra such:

$$\bar{\boldsymbol{q}} = \begin{bmatrix} q_0 \\ -\boldsymbol{q}_{1:3} \end{bmatrix}, \quad \|\boldsymbol{q}\| = \sqrt{q_0^2 + q_2^2 + q_3^2 + q_4^2}, \quad \boldsymbol{q}^{-1} = \frac{\bar{\boldsymbol{q}}}{\|\boldsymbol{q}\|}$$

However, multiplication is non-commutative:

$$\boldsymbol{q} \cdot \boldsymbol{p} = \begin{bmatrix} q_0 p_0 - \boldsymbol{q}_{1:3}^T \boldsymbol{p}_{1:3} \\ q_0 \boldsymbol{p}_{1:3} + p_0 \boldsymbol{q}_{1:3} - \boldsymbol{q}_{1:3} \times \boldsymbol{p}_{1:3} \end{bmatrix}$$

Equivalently, a vector in the world frame can be mapped in the body frame:

$$\begin{bmatrix} 0 \\ \boldsymbol{v_B} \end{bmatrix} = \boldsymbol{q} \cdot \begin{bmatrix} 0 \\ \boldsymbol{v_W} \end{bmatrix} \cdot \boldsymbol{q}^{-1}$$

The attitude estimation on the CrazyFlie is computed using quaternions and then mapped into Euler angles for control. A detailed explanation can be found in appendix A.

The weakness of unit quaternions is, especially in optimization problem like MPC, enforcing the quadratic constraints on the norm. Although several methods exist to solve this problem, none of them is satisfying enough.

3.2 Newton-Euler equations

In a quadcopter, each propeller generates a force F_i and a torque M_i proportional to the square of its spinning speed (c.f.: figure 5). The vector $\boldsymbol{u}$ is the vector of the squared spinning rates.

$$\boldsymbol{u} = [\omega_1^2 \ \omega_2^2 \ \omega_3^2 \ \omega_4^2]^T$$

$$F_i = K_f u_i \quad M_i = K_f u_i, \ i = 1, 2, 3, 4$$

In a "*plus*" configuration, the thrust force (or simply thrust) F_z and the torques M_x, M_y, M_z applied to the center of mass of the quadrotor, depends on $\boldsymbol{u}$:

$$\begin{bmatrix} F_z \\ M_x \\ M_y \\ M_z \end{bmatrix}_B = \begin{bmatrix} K_f & K_f & K_f & K_f \\ 0 & -K_f \cdot L & 0 & K_f \cdot L \\ K_f \cdot L & 0 & -K_f \cdot L & 0 \\ -K_t & K_t & -K_t & K_t \end{bmatrix} \mathbf{u} = K\mathbf{u}$$

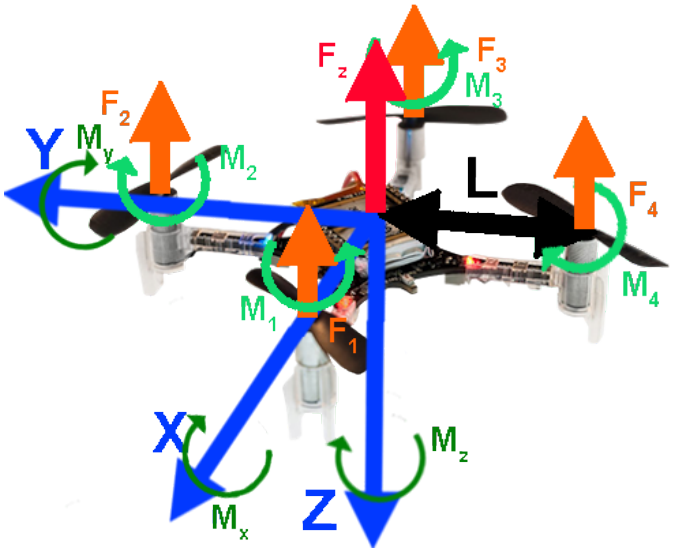


Figure 5: Force and torques applied on the CrazyFlie

Where L is the distance from the axis of rotation of a motor and O_b.

The dynamics of a quadrotor can now be derived using Newton-Euler equations:

$$m\ddot{\boldsymbol{r}} = \sum_i \boldsymbol{F_i} = \boldsymbol{G} + \boldsymbol{T} + \boldsymbol{D} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R_{BW}^T(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ -F_z \end{bmatrix}_B + \begin{bmatrix} D_x \\ Dy \\ D_z \end{bmatrix}$$

$$I\dot{\boldsymbol{\omega}}_{\boldsymbol{B}} = \sum \boldsymbol{M_B} - \boldsymbol{\omega_B} \times (I\boldsymbol{\omega_B}) = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}_B - \boldsymbol{\omega_B} \times (I\boldsymbol{\omega_B})$$

Where $\boldsymbol{r}$ is the position of O_B, expressed in the world frame, $\boldsymbol{G}$ is the gravity force, $\boldsymbol{T}$ is the thrust force generated by the propellers, $\boldsymbol{D}$ is the drag force, I is the inertia matrix of the quadcopter and $\boldsymbol{\omega_B}$ is the angular speed in the body frame.

To fully define the quadrotor, the attitude rates to body rate matrix $W(\phi, \theta, \psi)$ is still required:

$$W(\phi, \theta, \psi) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix}$$

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W(\phi, \theta, \psi)\boldsymbol{\omega_B}$$

3.3 Full model

$$\boldsymbol{x} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \phi\ \theta\ \psi\ \omega_{xB}\ \omega_{yB}\ \omega_{zB}]^T$$
$$\boldsymbol{u} = [\omega_1^2\ \omega_2^2\ \omega_3^2\ \omega_4^2]^T$$

$$
\begin{cases}
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix} \\
\begin{bmatrix} \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_5 \end{bmatrix} = \frac{1}{m}\left(\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R_{BW}^{T}(x_7, x_8, x_9) \begin{bmatrix} 0 \\ 0 \\ F_z(\boldsymbol{u}) \end{bmatrix}_B + \left(\begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \right) \right. \\
\begin{bmatrix} \dot{x}_7 \\ \dot{x}_8 \\ \dot{x}_9 \end{bmatrix} = W(x_7, x_8, x_9) \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} \\
\begin{bmatrix} \dot{x}_{10} \\ \dot{x}_{11} \\ \dot{x}_{12} \end{bmatrix} = I^{-1}\left(\begin{bmatrix} M_x(\boldsymbol{u}) \\ M_y(\boldsymbol{u}) \\ M_z(\boldsymbol{u}) \end{bmatrix}_B - \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} \times \left(I \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} \right) \right)
\end{cases}
$$

4 High-Level Control - Formation Control

Collective movement is widely observed phenomenon in the wild and civilized worlds. In animal societies, the phenomenon, called flocking, herding, schooling or swarming, depending on the species, happens for all size of groups, for all size of individuals, homo- or heterogeneous groups and sometimes can be circumstance-driven, like migrations. A flock, generally used to describe any of the denomination introduced before, is characterized by directed movement and quick reactions to obstacle at the global scale, even though there is no leader. Among the benefits of group motion, dilution of risk, improved exploration, foraging and sensing capabilities, energy saving are the most interesting. Of course costs arise also from group formation like the diminution of resource per capita but these concern mostly living entities [12, 13].

Flocks of robots are a perfect example of swarm or distributed intelligence, in which global structure appears solely through numerous local interactions [14, 15]. Indeed, computing trajectories for every robots, taking into account other robots, obstacles and some other constraints can soon become intractable when the number of robots increase. For collective motion in robotics, however, another kind of collective movement exists, motion in formation. Compared to flocking, where the relative location and the neighborhood of an individual is in permanent evolution, motion in formation assigns a specific position to each member and its neighborhood becomes static. Despite these constraints, this collective motion can also be implemented using a distributed control architecture, knowing that each individual needs to be assign a specific position in the formation. While flocking benefits more from robustness, flexibility and scalability, formations are more efficient, require less individuals for an equivalent coverage and are more intuitively commanded. Of course, the ideal group configuration depends on the application, the environment and the type of robot. Concerning this work, it will focus on motion in formation.
In a more global point of view, swarm robotics, compared to a single robot achieving the identical task, require simpler units and increase reliability [16]. The former implies simpler hardware, software, as well as sensing, acting and communicating capabilities. But, often in distributed intelligence, the cost of robustness is efficiency, in terms of time, total energy required or eventually another metrics. Or, restated, swarm robotics needs to carefully balance exploration et exploitation behaviors.

4.1 Taxonomy

When studying formations, the taxonomy defined in [17] about position determination serves as reference. In each possibility, each robot compute its place in the formation with respect to a given physical or mathematical landmark.

- *Unit-centered formation.* The unit center of a formation corresponds to its center of mass. While, this position determination configuration seems to achieve the best results, it can often be inadequate given the application. Moreover in case of fully distributed control, it requires a global knowledge of the position of every member of the formation, which can be implemented through communication channels or sufficiently performing sensing capabilities.
- *Leader-referenced formation.* The reference is one of the member of the group, the leader. This configuration achieves also good results and is ideal when the leader is human-controlled. However, it is constantly necessary to know, or being able to measure, the (relative) position of

the leader, which can require either a long range-communication, or sensing module, depending on the application and the localization method.

- *Neighbor-referenced formation.* In this case, there is no common reference. Each member of the formation determines its position from the position of a unique other member. Although it is more demanding in terms of emitters than a leader referenced formation, the range of communication can be considerably reduced. In case of relative localization without communication, this configuration has the lowest requirements in term of range. Note also that mixed leader-neighbor referenced formations can exist [18]. This is suitable when one leader is human-controlled and the individuals have low-range communication capabilities.

4.2 Implementation

Formation control can be implemented in various ways but we will focus on two, already widely implemented, from behavior-based control [17, 18, 19] to graph-based control [20, 21]. Both of these methods can easily be implemented in a distributed system for any of the configuration presented previously using relative localization sensors and IDs. But before selecting one of these methods, constraints of the actual control architecture have to be considered required. Indeed the MPC position controller can takes specific input vectors. Therefore, if a formation control block is devised, it can take as many input as wanted but the output must necessarily be a 6-state vector of three positions and three velocities for each drone.

Behavioral controllers, used in the form of motor-schema or, also called, potential field methods [17, 19] is an elegant way for designing adaptive controllers. Several zone can be defined, in which the potential can be chosen freely in order to induce the desired reaction. Typically the potential provides a speed vector from which position can be integrated. This approach can be implemented for any type of formation. An example of three-zones potential could be the following:

$$\dot{r}_i = \begin{cases} 0 & if\ r_{f,i} - r_i\ \in [0; d_1] \\ \alpha(r_{f,i} - r_i - d_1) & if\ r_{f,i} - r_i\ \in [d_1; d_2] \\ \beta(r_{f,i} - r_i - d_2)^2 + \alpha(r_{f,i} - r_i - d_1) & if\ r_{f,i} - r_i\ \in [d_2; \inf] \end{cases} for\ i = 1, 2, 3$$

Where $r_{f,i}$ is the reference position in formation, r_i the current position in one dimension, d_1, d_2, α and β are parameters

Graph-based control seems well adapted to provide a reference vector for leader- or neighbor-referenced formations. This kind of controllers are derived from so-called consensus problems, which try to make a set of agent with different initial states converge to a single final state, identical for all agents. [22]. The Rendez-vous problem [20] is one of these problems perfectly adapted to formation control. Required notions of graph theory are:

- $\mathcal{G} =< \mathcal{N}, \mathcal{E} >$, an (un)directed graph.
- $\mathcal{N} = \{R_1, \cdots, R_n\}$, the finite nodes set, with n, the number of robots.
- $\mathcal{E} \in \mathcal{N}^2, \mathcal{E} = \{(Ri, Rj) | R_i, R_j \in \mathcal{N} and\ i \neq j\} = \{e_1, \cdots, e_m\}$, the (un)directed finite edges set.

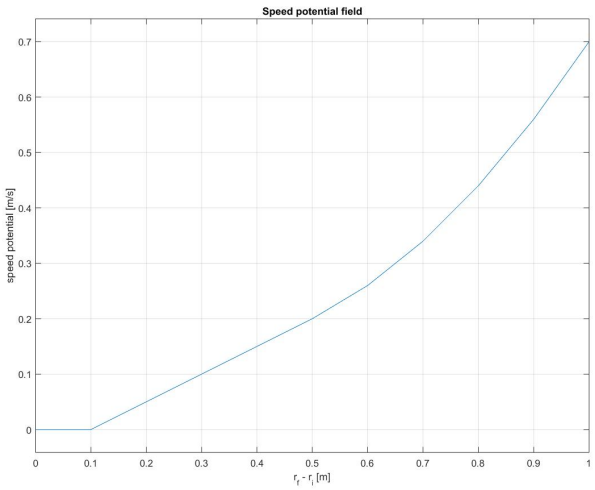


Figure 6: An example of attractive potential field

- $A \in \mathbb{R}^{n \times n}$, the adjacency matrix is a symmetric matrix, the image of the connections between the nodes:

$$a_{ij} = \begin{cases} 1 & if\ (R_i, R_j)\ or\ (R_j, R_i) \in \mathcal{E} \\ 0 & else \end{cases}$$

- $D \in \mathbb{R}^{n \times n}$, the degree matrix is a symmetric matrix, whose diagonal elements represent the cardinality of each node (i.e.: the sum of edges originating or arriving in this node):

$$d_{ii} = \sum_{j=1}^{N} a_{ij}$$

- $B \in \mathbb{R}^{n \times m}$, the incidence matrix is a matrix, describe which node is connected using which edge. In undirected graphs, edge are given a random direction.

$$b_{ij} = \begin{cases} 1 & if\ e_j = (R_i, \bullet) \\ -1 & if\ e_j = (\bullet, R_i) \\ 0 & else \end{cases}$$

- $W \in \mathbb{R}^{m \times m}$, a diagonal weight matrix whose element $w_{i,i}$ corresponds to the weight of edge $e_i \in \mathbb{E}$.

- L and $L_w \in \mathbb{R}^{n \times n}$, the Laplacian and the weighted Laplacian matrices:

$$L = D - A = BB^T \qquad L_w = BWB^T$$

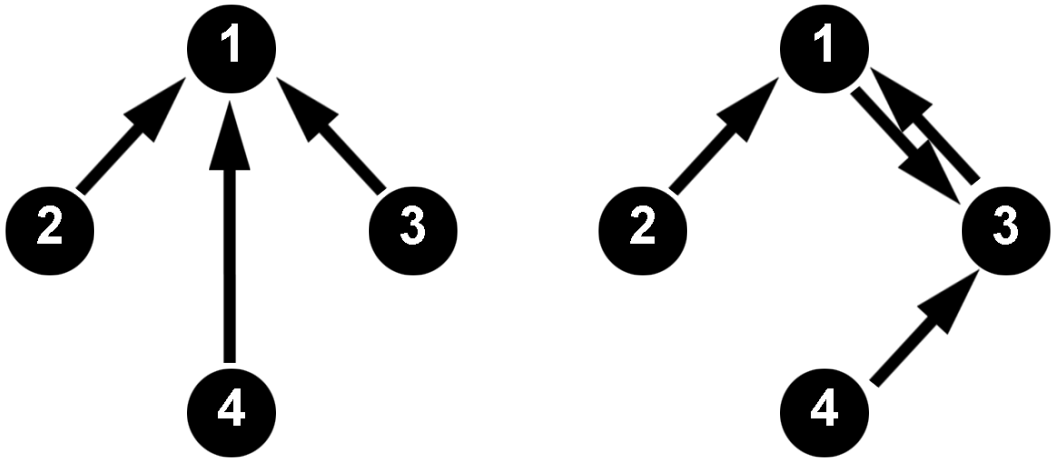
The Laplacian and the weighted Laplacian matrices benefits from interesting properties: They positive-semidefinite. This implies that all eigenvalues are nonnegative. Moreover, at least one eigenvalue isequal to zero. Proofs can be found in [20].

The feedback law, for state i of all nodes in a N-state system is:

$$\dot{\boldsymbol{x}}_{\boldsymbol{i}}(t) = -L\boldsymbol{x}_{\boldsymbol{i}}(t)\ for\ i = 1, \cdots, N$$

$$\boldsymbol{x}_{\boldsymbol{i}}(t) \in \mathbb{R}^n$$

Below, in figure 7, we show the directed graph for the leader-referenced formation and one possible graph for a neighbor-referenced formation. we will quickly compute the matrices A, D, B and L. We assume $W = I_n$, however in the neighbor-referenced case, it could be a good interesting to increase the weight of the edges (R_1, R_3) and (R_3, R_1), as they are central to the formation directed graph.



(a) The leader-referenced graph (b) A neighbor referenced graph

Figure 7: The directed graphs for a leader-referenced formation and a neighbor-referenced

$$A_{leader} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad A_{neighbor} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$D_{leader} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad D_{neighbor} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_{leader} = \begin{bmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad B_{neighbor} = \begin{bmatrix} 1 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$L_{leader} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad L_{neighbor} = \begin{bmatrix} 3 & -1 & -2 & 0 \\ -1 & 1 & 0 & 0 \\ -2 & 0 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

The discrete equivalent of the Laplacian feedback control requires another matrix [23], $L_k \in \mathbb{R}^{n \times n}$, the stochastic weight of links matrix. In a stochastic matrix, the sum of every row is equal to one. Therefore, 1 is an eigenvalue and $\mathbf{1}_n$ is the corresponding eigenvector.

$$l_{ij} = \begin{cases} \omega_{ij} > 0 & if \ = (R_i, R_j) \in \mathbb{E} \ or \ i = j \\ 0 & else \end{cases}$$

The feedback law become, for state i of all nodes in a N-state system:

$$\boldsymbol{x_i}(k+1) = L_k \boldsymbol{x_i}(k) \ for \ i = 1, \cdots, N$$

$$\boldsymbol{x_i}(t) \in \mathbb{R}^n$$

Again for the graphs in figure 7, assuming a equal weight for all edges, we provide the discrete equivalent of the Laplacian matrix:

$$L_{k,leader} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix} \quad L_{k,neighbor} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Another control method can be devised, taking the control architecture into account. It consists of directly acting on the MPC controller constraints or objective function[CITATIONS]. By implementing formation control in such a way, no additional block needs to be integrated in the current architecture. However, a more in-depth work in the controller is required as, an ideal formulation of cost and a weight need to be found. Moreover this approach reduce the modularity of the control system in its global aspect.

4.3 Measuring stability

[18]

- distance on formation line.
- distance perpendicular to formation line.
- angle deviation.

4.4 Measuring robustness

4.5 Evaluation

- time to formation (static ref) tToForm_refStatic.
- time to formation (dynamic ref) tToForm_refDynamic.
- % time in formation (normalized by (tTot-tToForm).
- average deviation from formation position (after tToForm).
- max deviation from formation position (after tToForm).

[17] [18]

5 Low-Level Control

5.1 Control Architecture

5.2 Inner Controller

The inner controller, responsible for attitude control can be found in the stabilizer.c module. Its main loop can be divided in four steps, the corresponding file are indicated in brackets:

- State acquisition (*sensors_stock.c* and *estimator_complementary.c* or *estimator_kalman.c*).
- Reference acquisition (*commander.c*).
- Control step (*controller_pid.c* and then *attitude_pid_controller.c* and *pid.c*).
- Motor distribution (*power_distribution_stock.c*).

State and reference acquisition

Independently of the chosen estimation method, 13-states Kalman filter or attitude update through Mahony or Magdwick quaternion method, the first step will provide data for attitude angles and rates. However, Kalman estimator allows the stabilizer loop to run only at 500 Hz, while the other method allows 1000 Hz. Sensor acquisition runs at 500 Hz for IMU and 100 Hz for barometer. In the second estimaiton method, attitude is updated at 250 Hz and position (if wanted) at 100 Hz. For Kalman, update are performed at 100 Hz for all state, barometer is processed at 25 Hz.

During the acquisition of the references, the inner control architecture is also defined or updated. Depending on the control mode, the reference are translated differently and the chain of controller is activated accordingly. There are six configuration variables, each can be set in absolute, velocity or disabled mode. These variables are X, Y, Z, the 3D positions and roll, pitch, yaw, the three attitude angles. For this project, the configuration implies that the reference are always going to be roll angle, pitch angle, yaw rate and thrust (c.f.: table 1). Despite the presence of a magnetometer in the IMU, it is not yet integrated in the firmware, therefore an absolute yaw positioning is not possible.

Table 1: Reference and inner control architecture configuration

Configuration variables	**State**
Position X	Disabled
Position Y	Disabled
Position Z	Disabled
Attitude Roll	Absolute
Attitude Pitch	Absolute
Attitude Yaw	Velocity

Controllers

The first module, *controller_pid.c*, defines the control architecture and the rate of the two sub-control loops: PID at 100 Hz for position control or cascade controllers at 500 Hz for attitude control. In the latter (*attitude_pid_controller.c*), the yaw rate reference is first integrated to obtain a yaw reference angle, then, the first layer of control is given the attitude angles references and estimations. In the second layer, rates are controlled. The output of the attitude controllers serve as rate reference and the gyros as attitude rate as measurement (c.f.: figure 8). This assumption can be considered as true for small angles or first-order approximations. For exact attitude rate computation, refer to the section ref:sec:dynamicsEquations. In the cascade controller, both layers are composed of standard PID controllers using an integral limit as shown in figure 9. Note also that the estimation method has an influence on the gains. In table 2, gains are provided for a CrazyFlie 2.0 using the stock estimator.

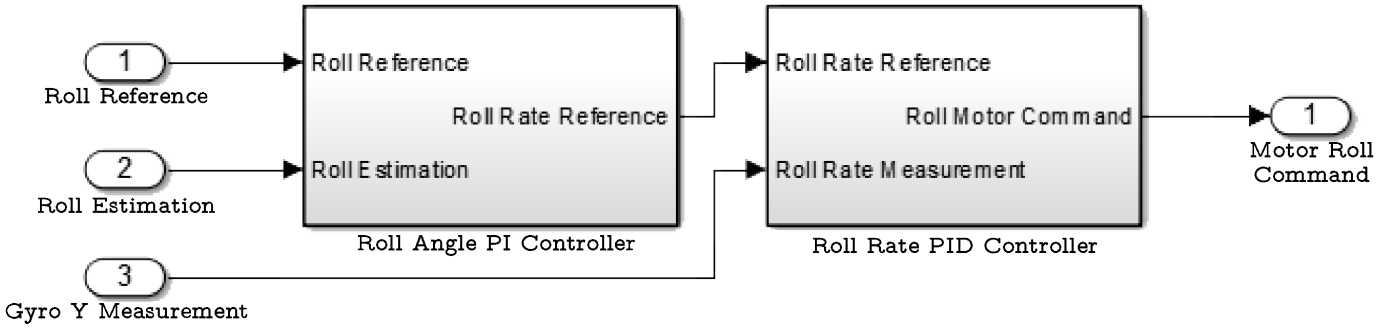


Figure 8: Roll cascade controller architecture

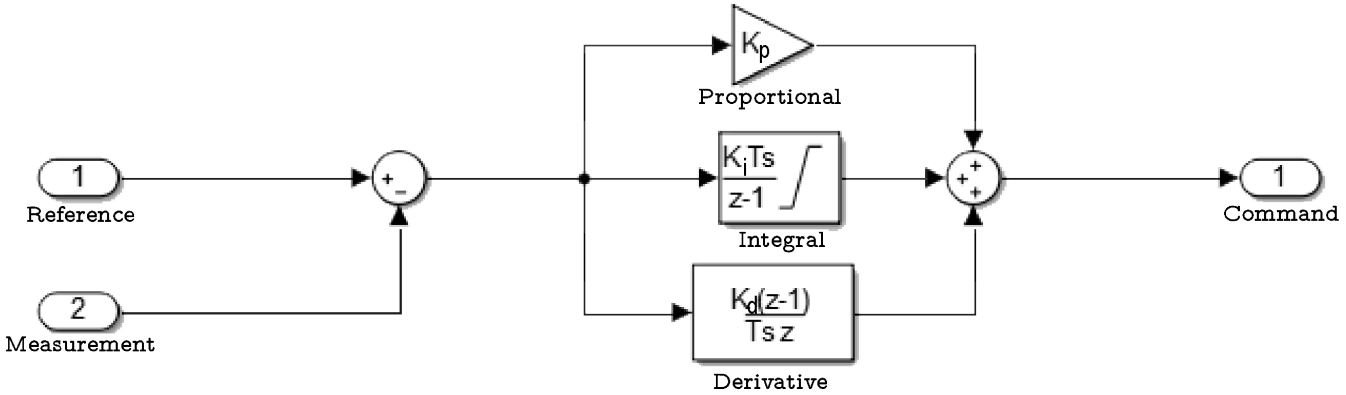


Figure 9: PID controller architecture

Table 2: Gains and integral limit of the inner PID controllers

Controller	K_p	K_i	K_d	Integral Limit
Roll	6	3	0	20
Pitch	6	3	0	20
Yaw	6	1	0.35	360
Roll rate	250	500	2.5	33.3
Pitch rate	250	500	2.5	33.3
Yaw rate rate	70	16.7	0	166.7

Motor distribution

The output of the 3 attitude cascade controllers and the thrust input are combined, with respect to the flying configuration, as shown in table 3, saturated and converted to a 8-bit value before being sent as PWM commands to the motors.

Table 3: Command summation for a + flying configuration

Motor 1	u_{thrust}	+	u_{pitch}	+	u_{yaw}
Motor 2	u_{thrust}	-	u_{roll}	-	u_{yaw}
Motor 3	u_{thrust}	-	u_{pitch}	+	u_{yaw}
Motor 4	u_{thrust}	+	u_{roll}	-	u_{yaw}

5.3 Force to Angle and Thrust Commands

5.4 Outer Controller

The outter posiiton controller is part of the receiding horizon controllers, or model prective controller, family. This type of controllers are well suited for constrained situations, strongly nonlinear systems or, as the name implies, for some future knowledge [4]. In a standard closed-loop implementation, the controller plans a series of N discrete actions for the open-loop problem, corresponding to the following N time steps. N is called the horizon length. Because the model is often inacurate, or because the environment is evolving, after executing the first action command, the controller measures the state and/or its surrounding and re-plans a serie of N actions acoordingly, which act as feedback. The longer the horizon length, the closer the prediction (the N planned actions) from the closed-loop response. Despite it's power, MPC suffers from chalenges to attain: feasibility, stability, robustness and implementation. A DEVELOPPER.

Implementation

In optimization-based control, such as MPC, a necessary condition for optimal trajectory in a control system Σ is the Pontryagin maximum principle [5, 6, 7]. A control system is defined as follow, with $\mathcal{X} \subset \mathbb{R}^n$, the state set and $\mathcal{U} \subset \mathbb{R}^m$, the input set

$$\Sigma = (\mathcal{X}, f, \mathcal{U})$$

$$\Sigma : \dot{x} = f(\boldsymbol{x}, \boldsymbol{u}),\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{u} \in \mathcal{U}$$

Two important objects need to be presented, the Lagrangian $\mathcal{L}$ and the extended Hamiltonian $\mathcal{H}_{\Sigma,\mathcal{L}}$. The Lagrangian is a continuous function defined as follow, and subsequently used to define an objective function $J_{\Sigma,\mathcal{L}}$:

$$\mathcal{L} : \mathcal{X} \times \bar{\mathcal{U}} \to \mathbb{R}$$

$$\mathcal{L} := \mathcal{L}(\boldsymbol{x}, \boldsymbol{u}),\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{u} \in \mathcal{U}$$

$$J_{\Sigma,\mathcal{L}} : \mathcal{X} \times \mathcal{U} \times \mathbb{R}_+ \to \mathbb{R}$$

$$J_{\Sigma,\mathcal{L}}(\boldsymbol{\xi},\boldsymbol{\mu},t)=\int_t^{t+T}\mathcal{L}(\boldsymbol{\xi}(t),\boldsymbol{\mu}(t))dt,\ (\boldsymbol{\xi}(t),\boldsymbol{\mu}(t))\in\mathcal{X}\times\bar{\mathcal{U}}$$

The couple $(\boldsymbol{\xi}(t),\boldsymbol{\mu}(t))$ is called a trajectory.

In order to minimize $J_{\Sigma,\mathcal{L}}$, three necessary conditions can be derived. However another mathematical object can be used, the extended Hamiltonian. It requires also the introduction of the co-state variable, or Lagrange multipliers $\boldsymbol{p}$:

$$\mathcal{H}_{\Sigma,\mathcal{L}}:\mathcal{X}\times\mathcal{U}\times\mathbb{R}^n\to\mathbb{R}$$

$$\mathcal{H}_{\Sigma,\mathcal{L}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p})=\boldsymbol{p}^T f(\boldsymbol{x},\boldsymbol{u})+\mathcal{L}(\boldsymbol{x},\boldsymbol{u})$$

To simplify equation and improve comprehensibility, the extended Hamiltonian $\mathcal{H}_{\Sigma,\mathcal{L}}$ will simply be called Hamiltonian and written $\mathcal{H}$. As well as $J_{\Sigma,\mathcal{L}}$ will be simplified as J

The maximum principle states that, for an optimal trajectory $(\boldsymbol{\xi}^*,\boldsymbol{\mu}^*)$ for cost function:

$$J(\boldsymbol{x},\boldsymbol{u},t)=V(\boldsymbol{x}(t+T))+\int_t^{t+T}\mathcal{L}(\boldsymbol{x},\boldsymbol{u})dt$$

$$V:\mathcal{X}\to\mathbb{R}$$

With $V(\boldsymbol{x}(t+T))$, the terminal cost. Subject to:

$$\dot{\boldsymbol{x}}=f(\boldsymbol{x},\boldsymbol{u})$$

$$\psi_i(\boldsymbol{x})=0,\ i=1,\cdots,q$$

With $\psi(x)$, equality state constraints (often only for the terminal state). These constraints can be adjoined to the Hamiltonian with the help of another multiplier $\boldsymbol{v}$:

$$\mathcal{H}_{\Sigma,\mathcal{L}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{v})=p^T f(\boldsymbol{x},\boldsymbol{u})+v^T\psi(\boldsymbol{x})+\mathcal{L}(\boldsymbol{x},\boldsymbol{u}),\ \boldsymbol{v}\in\mathbb{R}^q$$

Then, there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\nu}^*$ such that:

$$\dot{\boldsymbol{\xi}}^*=\mathcal{H}_p^T(\boldsymbol{\xi}^*,\boldsymbol{\mu}^*,\boldsymbol{\lambda}^*,\boldsymbol{\nu}^*)\qquad\dot{\boldsymbol{\lambda}}^*=-\mathcal{H}_x^T(\boldsymbol{\xi}^*,\boldsymbol{\mu}^*,\boldsymbol{\lambda}^*,\boldsymbol{\nu}^*)$$

$$\boldsymbol{\lambda}(t+T)^*=V_x^T(\boldsymbol{\xi}^*(t+T))$$

And:

$$H(\boldsymbol{\xi}^*,\boldsymbol{\mu}^*,\boldsymbol{\lambda}^*,\boldsymbol{\nu}^*)\leq H(\boldsymbol{\xi}^*,u,\boldsymbol{\lambda}^*,\boldsymbol{\nu}^*)\forall\boldsymbol{u}\in\mathcal{U}$$

Moreover, if the input are unconstrained (i.e.: $\mathcal{U}=\mathbb{R}^m$), a necessary condition for optimal input sequence is:

$$\mathcal{H}_u(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{v})=\mathbf{0}_m$$

Where A_b denotes either the gradient of a scalar field or the Jacobian of a vector function A with respect to $\boldsymbol{b}$.

$$A_b=\nabla A=\left[\frac{\partial\mathcal{A}}{\partial b_1}\cdots\frac{\partial\mathcal{A}}{\partial b_n}\right]$$

$$A_b = \begin{bmatrix} \frac{\partial A_1}{\partial b_1} & \frac{\partial A_1}{\partial b_2} & \cdots & \frac{\partial A_1}{\partial b_s} \\ \frac{\partial A_2}{\partial b_1} & \frac{\partial A_2}{\partial b_2} & \cdots & \frac{\partial A_2}{\partial b_s} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial A_r}{\partial b_1} & \frac{\partial A_r}{\partial b_2} & \cdots & \frac{\partial A_r}{\partial b_s} \end{bmatrix}$$

This last characteristic is part of the core of the Continuation-Generalized Minimum Residual Method (C/GMRES)[8] used to solve the optimization process. Therefore, instead of constraining the input set, a penalty cost will be included in the Lagrangian such that the previous condition is satisfied.

Note that Pontryagin maximum principle is only a necessary condition of optimal trajectory. A sufficient and necessary condition is the Hamilton-Jacobi-Bellman equation, but it must be satisfied on the whole state space. Note also that all the result are presented on a continuous time scale but they can be discretized.

The C/GMRES method is a fast algorithm for nonlinear MPC exploiting the fact that the optimal trajectory varies smoothly with respect to time. This part of the algorithm is what is called the continuation, or homotopy, method. It is combined with GMRES in order to quickly solve large linear equations. The main idea of C/GMRES is to compute the derivative of the input sequence and to integrate it other time, instead of computing at each step a new input sequence using standard iterative methods.

Starting from a approximated version of the optimization problem using forward discretization. For simplification,a mapping from $[t; t+T]$ to $[0; T]$ is performed and $\boldsymbol{x}(t = kT_s)$ is denoted $\boldsymbol{x}(k)$ with T_s the sampling time.

$$J_k = V(\boldsymbol{x}(N)) + \sum_{k=0}^{N-1} \mathcal{L}(\boldsymbol{x}(k), \boldsymbol{u}(k))T_s$$

Subject to:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + f(\boldsymbol{x}(k), \boldsymbol{u}(k))T_s$$
$$\psi_i(\boldsymbol{x}(k)) = 0,\ i = 1, \cdots, q_k$$

Then we can rewrite a discretized version of the Pontryagin principle for unconstrained input:

$$\mathcal{H}_u(\boldsymbol{\xi}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \mathbf{0}_m$$
$$\boldsymbol{\lambda}^*(k) = \boldsymbol{\lambda}_i^*(k+1) + \mathcal{H}_x^T(\boldsymbol{\xi}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)T_s$$
$$\boldsymbol{\lambda}^*(N) = V_x^T(\boldsymbol{\xi}^*(N))$$

The control input $\boldsymbol{\mu}^*$ and the multipliers $\boldsymbol{\nu}^*$ can be computed solving a set of N(m+q) equations. We introduce also the vector U:

$$\boldsymbol{U} = [\boldsymbol{u}(0), \boldsymbol{v}(0), \boldsymbol{u}(1), \boldsymbol{v}(1), \cdots, \boldsymbol{u}(N-1), \boldsymbol{v}(N-1)]^T$$

$$F(\boldsymbol{U}, \boldsymbol{x}) := \begin{bmatrix} \mathcal{H}_u(\boldsymbol{x}(0), \boldsymbol{u}(0), \boldsymbol{p}(1), \boldsymbol{v}(0)) \\ \psi(\boldsymbol{x}(0)) \\ \vdots \\ \mathcal{H}_u(\boldsymbol{x}(N-1), \boldsymbol{u}(N-1), \boldsymbol{p}(N), \boldsymbol{v}(N-1)) \\ \psi(\boldsymbol{x}(N-1)) \end{bmatrix} = \mathbf{0}_{N(m+q)}$$

To simplify notation, $F = F(\boldsymbol{U}, \boldsymbol{x})$. Continuation method consists of, starting with a solution of a known problem, for example $F = 0$ at time t, computing the solution of another problem, about which we have few knowledge concerning solutions, for example $F = 0$ at time $t + T_s$ [9]. Applied to our controller, it means that the equations $F = \mathbf{0}_{N(m+q)}$ are not computed every sampling time. Instead $\dot{\boldsymbol{U}}$ is computed such that:

$$F(\boldsymbol{U} + \dot{\boldsymbol{U}}T_s, \boldsymbol{x} + \dot{\boldsymbol{x}}T_s) = \mathbf{0}_{N(m+q)}$$

$$\dot{\boldsymbol{U}} = F_U^{-1}(A_s F - F_x \dot{\boldsymbol{x}})$$

A_s is a matrix whose role is to stabilize $F = \mathbf{0}_{N(m+q)}$ and F_u is required to be nonsingular. conditions on A_s. A noticeable constraint of this method is the requirement to solve the system $F = 0$ at least once at $t = 0$.

In order to reduce the cost of solving the linear equations, two additional tools are used, forward difference approximation for Jacobian and vector products, and the GMRES method for solving linear equations. The product approximation is computed as follow:

$$F_U(\boldsymbol{U}, \boldsymbol{x})\boldsymbol{W} + F_x(\boldsymbol{U}, \boldsymbol{x})\boldsymbol{w} \approx D_h F(\boldsymbol{U}, \boldsymbol{x} : \boldsymbol{W}, \boldsymbol{w}) := \frac{F(\boldsymbol{U} + h\boldsymbol{W}, \boldsymbol{x} + h\boldsymbol{w}) - F(\boldsymbol{U}, \boldsymbol{x})}{h}$$

Then the linear equation to solve to compute $\dot{\boldsymbol{U}}$ is:

$$D_h F(\boldsymbol{U}, \boldsymbol{x} + h\dot{\boldsymbol{x}} : \dot{\boldsymbol{U}}, 0) = A_s F(\boldsymbol{U}, \boldsymbol{x}) - D_h F(\boldsymbol{U}, \boldsymbol{x} : 0, \dot{\boldsymbol{x}}) := \boldsymbol{b}(\boldsymbol{U}, \boldsymbol{x}, \dot{\boldsymbol{x}})$$

The GMRES tool (or Forward Difference GMRES, FDGMRES, in [10]) is an iterative Krylov subspace method to solve linear systems of equations. It can be decomposed in three steps.

First, the initialization of the residual $\hat{r}$ and the Krylov basis $\mathcal{V}_\mathcal{K}$ with an initial guess $\hat{\dot{\boldsymbol{U}}}$ on the control input derivative:

$$\hat{\boldsymbol{r}} := \boldsymbol{b}(\boldsymbol{U}, \boldsymbol{x}, \dot{\boldsymbol{x}}) - D_h F(\boldsymbol{U}, \boldsymbol{x} + h\dot{\boldsymbol{x}} : \hat{\dot{\boldsymbol{U}}}, 0)$$

$$\boldsymbol{v_1} := \frac{\hat{\boldsymbol{r}}}{\|\hat{\boldsymbol{r}}\|}, \ \rho = \|\hat{\boldsymbol{r}}\|, \ \beta := \rho$$

.

Then, the loop iteration while $k_{loop} < k_{loop,max}$:

1. $k = k + 1$
2. Compute a new basis vector for $\mathcal{V}_\mathcal{K}$: $\boldsymbol{v}_{k+1} = D_h F(\boldsymbol{U}, \boldsymbol{x} + h\dot{\boldsymbol{x}} : \boldsymbol{v_k}, 0)$.
3. Normalization, orthogonalization using Gramm-Schmidt process and construction of matrix H_k:

$$Loop : h_{j,k} = \boldsymbol{v}_{k+1}^T \boldsymbol{v}_j, \ \boldsymbol{v}_{k+1} = \boldsymbol{v}_{k+1} - h_{j,k}\boldsymbol{v}_j, \quad for \ j = 1, \cdots, k$$

$$h_{k+1,k} = \|\boldsymbol{v}_{k+1}\|$$

$$h_{k+1,j} = 0 \quad for \ j = 1, \cdots, k-1$$

$$\boldsymbol{v}_{k+1} = \frac{\boldsymbol{v}_{k+1}}{\|\boldsymbol{v}_{k+1}\|}$$

4. Solve $\min_{y_k} \|\beta \boldsymbol{e}_1 - H_k \boldsymbol{y}_k\|$, with $\boldsymbol{y}_k in \mathbb{R}^k$.
5. Residual length $\rho = \|\beta \boldsymbol{e_1} - H_k \boldsymbol{y}_k\|$.

The minimization step can be solved efficiently using Givens rotations [11]. Which consists of applying consecutive rotations $G_i(\theta_i)$ to the system untill $\prod_i G_i(\theta_i) H_k$ is upper triangular. We define a Givens rotation matrix $G(p, q, \theta)$ as follow:

$$g_{ij} = \begin{cases} \cos\theta & if\ i = j = p, q \\ 1 & if\ i = j \neq p, q \\ -\sin\theta & if\ i = p,\ j = q \\ \sin\theta & if\ i = q,\ j = p \\ 0 & else \end{cases}$$

For our algorithm, at iteration k, we compute the matrix G_k:

$$G_k := G_k(k, k+1, \theta_k) \quad with\ \theta_k = atan2(h_{k+1,k}, h_{k,k})$$

And we solve easily:

$$\min_{y_k}\left(\left\|\prod_{i=1}^{k} G_i(i, i+1, \theta_i)(\beta \boldsymbol{e}_1 - H_k \boldsymbol{y}_k)\right\|\right.$$

Moreover, despite being iterative, this step is limited in number of iterations (k_{max}) to keep its influence small on the whole algorithm duration

The final step is the computation of solution:

$$\dot{\boldsymbol{U}} = \hat{\dot{\boldsymbol{U}}} + V_k \boldsymbol{y}_k$$

With $V_k = [v_1, \cdots, v_k]$ the Krylov subspace basis $\mathcal{V}_\mathcal{K}$.

Now, continuation and GMRES method can be combined to produce C/GMRES:

1. Initialization: compute $\boldsymbol{U}(0)$ numerically or anatically.
2. Loop over k:
   (a) Apply first input of sequence: $\boldsymbol{u}(k) = \boldsymbol{U}_{1:m}(k)$.
   (b) Measure $\boldsymbol{x}(k)$, Compute $\Delta x = \boldsymbol{x}(k) - \boldsymbol{x}(k-1)$.
   (c) Compute $\dot{\boldsymbol{U}}(k)$ using GMRES with $\dot{x} = \frac{\Delta x}{Ts}$ and $\hat{\dot{\boldsymbol{U}}}(k) = [\dot{\boldsymbol{U}}^T_{m+q+1:N(m+q)}(k-1)\ \boldsymbol{0}_{m+q}]^T$ (i.e.: the input of last time step, without the first elements corresponding to the input applied and padded with zeros).
3. k = k+1.

EFFICIENCY, CONVERGENCE SPEED, ROBUSTNESS, APPROXIMATION ERROR

Model

The outer MPC controller controls the position and its derivative and the yaw angle, therefore a 7-states model is considered.

Moreover, to remove the dependency of the position on the attitude and on the flying platform, the thrust force is represented as:

$$R_{BW}^{T}(x_7, x_8, x_9) \begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix}_B = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \boldsymbol{u_{1:3}}$$

The inner controller, taking as references $\boldsymbol{r_{ic}} = [F_{zB}\ \phi\ \theta\ \dot{\psi}]^T$, an intermediary block, proper to quadrotors will be implemented to translate the output of the reduced model into adequate commands for the quadrotor.

The drag force, in a turbulent air flow, is usually defined as:

$$\boldsymbol{D} = -(\frac{1}{2} \left\|\dot{\boldsymbol{r}}\right\|^2 C_D A) \frac{\dot{\boldsymbol{r}}}{norm\dot{\boldsymbol{r}}}$$

Where $\dot{\boldsymbol{r}}$ is the relative speed of the quadcopter to the fluid, ρ is the air density, C_D the drag coefficient and At the cross-sectional area (i.e.: the maximal area of the quadrotor in the plane perpendicular to the speed vector $\dot{\boldsymbol{r}}$). For modeling purpose, it can be simplified as follow:

$$\boldsymbol{D} \approx -K_d(\psi) \left\|\dot{\boldsymbol{r}}\right\| \dot{\boldsymbol{r}}$$

A DEVELOPPER

Then, to linearize the model, a working is required. Assuming we stay around the hovering equilibrium, then only gravity is compensated.

$$\boldsymbol{u_s} = \begin{bmatrix} f_x \\ f_y \\ f_z \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \\ 0 \end{bmatrix}$$

The reduced linearized model is:

$$\boldsymbol{x_r} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \psi]^T$$

$$\dot{\boldsymbol{x}}_{\boldsymbol{r}} = A_r \boldsymbol{x_r} + B_r \boldsymbol{u}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \psi \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x \\ f_y \\ f_z \\ \dot{\psi} \end{bmatrix}$$

Finally, the model is discretized, with sampling time T_s:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{z}_{k+1} \\ \psi_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_s & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & T_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & T_s & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \\ \psi_k \end{bmatrix} + \begin{bmatrix} \frac{T_s^2}{2m} & 0 & 0 & 0 \\ 0 & \frac{T_s^2}{2m} & 0 & 0 \\ 0 & 0 & \frac{T_s^2}{2m} & 0 \\ \frac{T_s}{m} & 0 & 0 & 0 \\ 0 & \frac{T_s}{m} & 0 & 0 \\ 0 & 0 & \frac{T_s}{m} & 0 \\ 0 & 0 & 0 & T_s \end{bmatrix} \begin{bmatrix} f_{x,k} \\ f_{y,k} \\ f_{z,k} \\ \dot{\psi}_k \end{bmatrix}$$

Constraints

6 Results - Experiment

6.1 Radio Delay Estimation

6.2 Motion Capture Caracterization

6.3 Yaw Control

6.4 Posiiton Control

6.5 Controlled Trajectory

6.6 Without Speed Reference

6.7 With Speed Reference

6.8 Control Loop Timing

6.9 Other Results

7 Improvement

8 Conclusion

Bibliography

[1] S. Suzuki, T. Ishii, Y. Aida, Y. Fujisawa, K. Iizuka, and T. Kawamura, "Collision-free guidance control of small unmanned helicopter using nonlinear model predictive control," *SICE Journal of Control, Measurement and System Integration*, vol. 7 (6), November.

[2] BitCraze, "bitcraze.io," Oct. 2016.

[3] J. Diebel, "Representing attitude: Euler angles, unit quaternions and rotation vectors," tech. rep., Stanford University.

[4] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for linear and hybrid systems*. Cambridge University Press, 2015.

[5] L. Pontryagin, *Mathematical Theory of Optimal Processes*. CRC Press, 1987.

[6] A. D. Lewis, "The maximum principle of pontryagin in control and optimal control," tech. rep., University of Catalonia.

[7] R. M. Murray, "Optimization-based control," tech. rep., California Institute of Technology.

[8] T. Ohtsuka, "A continuation/gmres method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40 (4), pp. 563–574, 2004.

[9] E. L. Allgower and K. Georg, *Introduction to numerical continuation methods*. Society for Industrial Mathematics, 1987.

[10] C. T. kelley, *Iterative methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.

[11] A. Bjõrck, *Numerical methods for least squares problems*. Society for Industrial and Applied Mathematics, 1996.

[12] J. Krause and G. D. Ruxton, *Living in Groups*. Oxford University Press, 2002.

[13] J. K. Parrish and H. W. M., *Animal Groups in Three Dimensions*. Cambridge University Press, 1997.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[15] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics, volume= 21 (4), month=July, year= 1987, pages=25-34.*

[16] G. Beni, "From swarm intelligence to swarm robotics," in *2004 Workshop on Swarm Robotics, Proceedings of SAB*, pp. 1–9, 2004.

[17] T. Balch and R. C. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transactions on Robotics and Automation*, vol. 14 (6), pp. 926–939, 1999.

[18] J. Fredslund and M. J. Matarić, "A general algorithm for robot formations using local sensing and minimal communication," *IEEE Transactions on Robotics and Automation*, vol. 18 (5), pp. 837–846, October 2002.

[19] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast onboard relative positioning module for multirobot systems," *IEEE/ASME Transaction on Mechatronics*, vol. 14 (2), pp. 151–162, April 2009.

[20] S. Gowal, *A Framework for Graph-Based Distributed Rendezvous of Nonholonomic Multi-Robot Systems*. PhD thesis, EPFL Thesis no. 5845, 2013.

[21] R. Falconi, S. Gowal, and A. Martinoli, "Graph based distributed control of non-holonomic vehicles endowed with local positioning information engaged in escorting missions," in *Robotics and Automation, 2010 IEEE International Conference on*, pp. 3207–3214, IEEE, 2010.

[22] W. Ren, R. W. Beard, and E. M. Atkins, "A survey of consensus problems in multi-agent coordination," in *Proceedings of the 2005, American Control Conference*, pp. 1859–1964, IEEE, 2005.

[23] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on Automatic Control*, vol. 50 (2), pp. 169–182, February 2005.

[24] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *IEEE/RSJ Intl Conf. Intelligent Robots and Systems*, pp. 5382 – 5387, Sept 2015.

[25] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, p. to appear, 2012.

Appendices

A Attitude estimation

B Install ROS package from Git

crazyflie ROS doc [24]
hector_quadrotor doc [25]

```
  # make sure you have sourced the correct setup.bash file for your ROS distribution already

  # go to workspace src space
cd /path/to/your/catkin/src

  # checkout the desired version of the descartes repository.
git clone -b [branch] https://github.com/[..].git

  # we need to make sure you have all dependencies installed.
cd /path/to/your/catkin
rosdep install –from-paths src –ignore-src –rosdistro kinetic

  # now build
catkin_make

  # source
source devel/setup.bash
```