Research Internship
Final report

Distributed consensus for quadcopter formation control

26 May - 5 September 2025

Department : Computer Science
Author : Thomas Wanchai MENIER
Internship Supervisor : Tatsuya IBUKI
Academic Supervisor : Julien ALLALI

Contents

1 Summary of the internship setting (in French)

Le stage a été réalisé à l'étranger au Japon, dans un des campus de l'Université de Meiji, en laboratoire de recherche. Dirigé par le professeur Tatsuya Ibuki, chercheur en théorie du contrôle, la salle dispose d'un espace pour faire voler des drones en intérieur.

L'objectif du stage était double : implémenter un algorithme de consensus pour faire réaliser des formations à des drones en temps réel, puis déterminer si celui-ci peut être implémenté de manière décentralisée (ou distribuée). Cela signifie qu'une version modifiée de l'algorithme existe qui peut être lancée sur chaque drone leur permettant de communiquer leur position localement, sans avoir à passer par un ordinateur central, pour atteindre le même résultat que l'algorithme centralisé. La motivation centrale était de pouvoir réaliser ces formations sans demander aux robots de se placer à un point spécifique dans le monde. La formation atteinte serait alors relative aux positions actuelles des drones.

Pour tester l'implémentation réalisée, des drones étaient fournis par le Professeur Ibuki. Afin de ne pas endommager le matériel, il était aussi demandé d'implémenter un autre algorithme d'évitement, appelé les *Control Barrier Functions*, afin de le combiner avec le contrôle de flotte, pour que les drones s'évitent en plein vol.

Les deux objectifs ont été atteints et validés par le Professeur Ibuki, et le fonctionnement des algorithmes employés sont détaillés dans ce rapport. Une expérience en réelle a pu être conduite avec succès sur les drones fournis, avec l'évitement d'obstacle implémenté.

2 Presentation of the Ibuki laboratory at Meiji University

2.1 Meiji University

The current internship is hosted in Meiji University, a private Japanese research university divided in four different campuses around Tokyo : Surugadai, Izumi, Nakano and Ikuta. It is a university for graduate and post graduate students. While it was originally a school of law when it was created in 1881, it became a university in 1920. The research internship is located on the Ikuta campus, in the laboratory of Tatsuya Ibuki.

2.2 Laboratory of Tatsuya Ibuki

This laboratory is focused on research around robotics, while hosting bachelor and master students. It is responsible for supervising the graduation of these students, as being part of the laboratory is mandatory for students of Meiji University.

Additionally, Professor Ibuki wants the students to develop project management skills by studying previous works performed in the laboratory, by creating new results and verification of previous experiments. By presenting their progress monthly to each other, the goal is for students to cooperate together, and discuss results obtained, as well as their experiments and methods employed.

Apart from this mandatory meeting every Monday, students must continue working on their research project on their own, without ordered directives. Of course, they are still allowed to ask for directives from Professor Ibuki, but they have much more independence than ENSEIRB-MATMECA students during their projects, as they have to work on it when they do not have class. Master students even came during the summer vacation to progress on their projects.

3 Introduction and motivation

Given a number of drones and the software to control their position and speed in the world frame, the goal is to control them so that they achieve formations, such as alignment or forming a triangle. Common examples include the famous Chinese drone shows in the sky, or even in Bordeaux on the 14th of July. An obvious solution is to compute exact positions for the drones to attain, seemingly fulfilling the objective. But it is not flexible enough : positions need to be computed in advance, and the formation is always achieved at the same location.

The goal of the internship is to discover the usage of the consensus algorithm to achieve relative formation control, that is, Formation must be achievable at any location in the world. While being more flexible, it also means that moving a single drone will impact all of the drones.

Finally, we need to prove whether such an algorithm can be computed in a distributed fashion. Instead of computing the algorithm on a central computer, can it be computed by each drone individually so that they communicate together instead ? Will we obtain the same result ?

As safety is a primary requirement, implementing a collision avoidance algorithm is mandatory before running any experiment. Professor Ibuki has explicitly requested the usage of the *Control Barrier Function* technique that is employed in the laboratory, before running real-life experiments.

Before understanding why and how a consensus algorithm can help achieve formations, there are some definitions and formulas that need to be thoroughly defined beforehand.

4 The consensus problem

4.1 What is consensus ? An analogous example

We define consensus as a collection of agents with their own value who are trying to attain a common decision together over time. Take the example of a bakery for example, driven by 3 different persons : Alice, Bob and Charlie. These chefs want to debate the price of a new cookie that will be sold at the

bakery. Everyone has its own opinion, and might not be able to influence each other. Maybe Alice listens to the opinion of Bob, but not Charlie. To simplify, we will consider that every person listens to at least someone.

This influence can be modeled using a graph where each node represents one person, and the directed arcs represent whether a person influences the choice of another. The following graph in figure 1 represents how their discussions went over time (under the assumption the influence graph does not change).

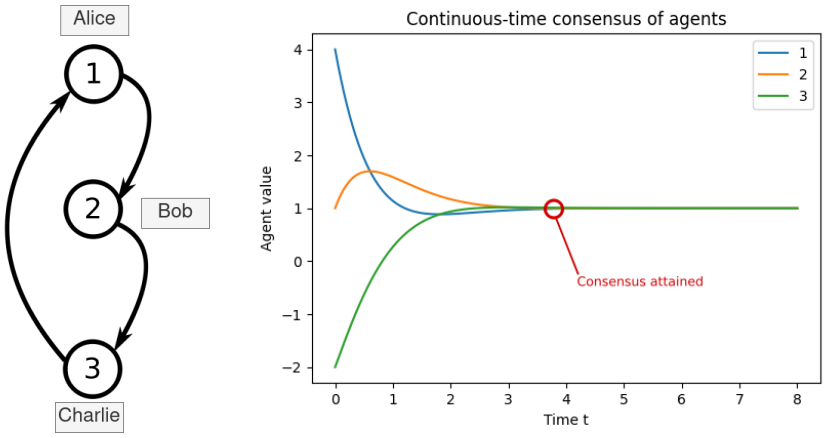

Figure 1: Continuous-time consensus algorithm example in one dimension. Three agents in a directed network influence each other's value, until attaining agreement.

We can see the negotiations[1] led everyone the same value, called the consensus value, and that it does not change over time. With this goal in mind, let's define clearly the algorithm used to obtain this result.

4.2 Mathematical definition

Consider that we have $n \in \mathbb{N}$ agents, and each of them has a given value $x_i \in \mathbb{R}^2$ for each agent $i \in \{1, 2, ..., n\}$. Each agent will be defined as an integrator agent, meaning that its current value changes in respect to time. Thus the current value of an agent is denoted $x_i(t)$. The algorithm employed to achieve consensus is given by the ordinary differential equation (1)

$$\dot{x}_i(t) = \sum_{j \in N_i} x_j(t) - x_i(t) \tag{1}$$

where the directed graph $G = (V, E)$ is defined, along with the adjacency matrix $A = [a_{ij}]$ where, if agent i is linked to agent j, then $a_{ij} \neq 0$. Thus, the

[1] Of course, this is an ideal case of what can happen in real-life. The example of the bakery is just here to help the reader understand the definition of consensus here.

neighbours of an agent i is given by $N_i = \{j \in V \mid a_{ij} \neq 0\}$. This is the continuous-time consensus algorithm that can be employed for any graph G. Convergence is assured as long as the graph G is strongly connected, i.e. there exists a path from any node to any other node in the graph. Nodes with no neighbours are not taken in account (as they do not participate in the consensus algorithm). Note that it is possible to express the dynamics[2] of this system using the Laplacian matrix denoted as L, as follows in (2)

$$\dot{x}_i = -Lx \tag{2}$$

This form has been used for stability analysis and convergence of the consensus algorithm in [1]. Using this tool, how can we change it to achieve formations ?

4.3 Deriving consensus for formation control

At the moment, we can only converge different values into a common one. Formation control will be achieved by adding offset terms between agents that are connected. With the configuration in figure 1, for each neighbour $j \in N_i$ of an agent i, we add a term named d_{ij} which is a relative offset from i to j. All offsets must be represented in the same coordinate basis. For consensus to still converge[3], the sum of all relative offsets must be zero. This gives us the rewritten consensus algorithm in (3).

$$\begin{aligned} \dot{x_i}(t) &= \sum_{j \in N_i} x_j(t) - x_i(t) - d_{ij} \\ s.t &\sum_{\substack{i \in V, \\ j \in N_i}} d_{ij} = 0 \end{aligned} \tag{3}$$

In the configuration that was mentioned earlier, applying this algorithm will give the result displayed in figure 2, with fixed offset values that do not change over time.

This mathematical tool alone is sufficient to achieve formations. If we consider the agent's value to be the (x, y) coordinates of a drone, consensus can be used to achieve relative position-based formations. An example of application is shown in figure 3, being a simulation of how a group of three drones would achieve a triangular formation.

While this can be used for a real-time application, there is one disadvantage to using a continuous-time algorithm. Simulation is a perfect world with no interference, but this is not true for the real world. If we were to command

[2] In this report, "dynamics" does not refer to the term used in robotics, that studies the causes of motion. Rather, we use the definition employed in control theory, where "dynamics" refer to a system evolving in respect to time.

[3] Professor Ibuki mentioned that there is no proof of convergence when using offset terms, this constraint has been determined numerically.

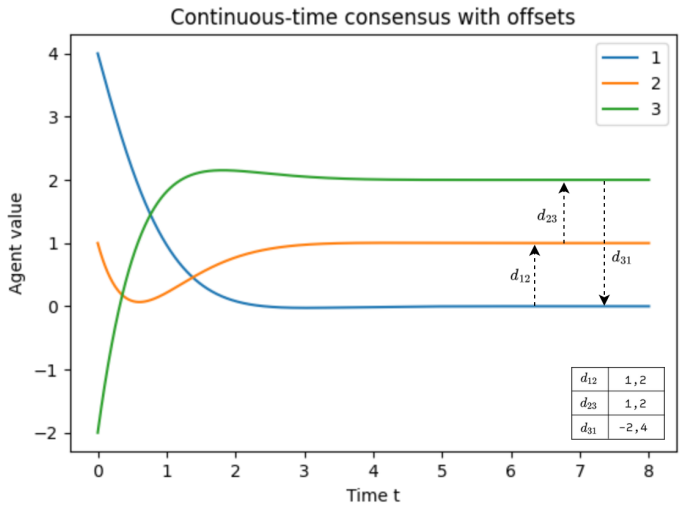

Figure 2: Continuous-time consensus algorithm in one dimension with offsets d_{ij} defined between each agent, as defined in (3), with the offset values used in a side table.

drones in an environment comprised of obstacles, the final solution obtained could not be feasible, requiring to compute it again every time the formation becomes impossible to achieve. The target positions that were computed right when the drones started flying would become useless, which means that we've spent computational power and time for nothing. A solution to this problem is using a discrete-time algorithm instead.

4.4 Discrete-time consensus

The discrete time version of the consensus algorithm achieves the same results as its continuous counterpart, as proven in [1]. The key difference lies in the time-discretization of the method. In the continuous-time algorithm, we need an ODE solver to obtain the different states to achieve over time, considering the step size dt is infinitely small. In the discrete-time version, this step size is defined inside the algorithm and controlled by the value ϵ in equation (4), which is constrained by (5). $x_i[k]$ denotes the value of agent i at "discrete time step"[4] k.

[4]Brackets instead of parenthesis have been used to highlight that we consider discrete time steps. This style of writing is taken directly from [1].

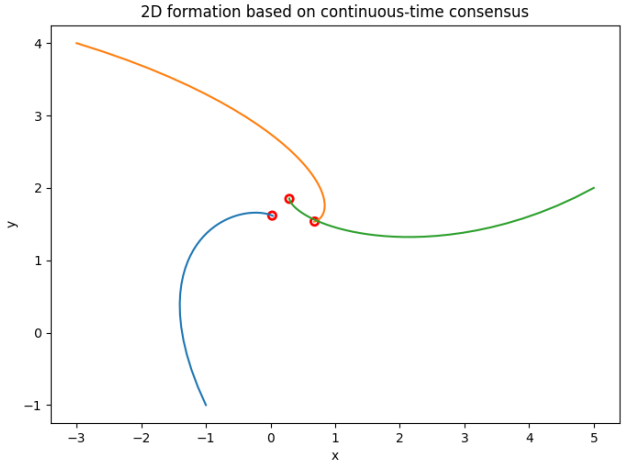

Figure 3: Continuous-time consensus algorithm with offsets, achieving a triangular formation. The red circle highlights the final position of each agent.

$$x_i[k+1] = x_i[k] + \epsilon \sum_{j \in N_i} x_j[k] - x_i[k] - d_{ij} \tag{4}$$

$$s.t \quad 0 < \epsilon < \Delta \tag{5}$$

The algorithm thus computes the next state using two terms : the previous state and a similar term from its continuous counterpart. Instead of computing the whole solution at a given time step, this version only computes the very next state of the system of agents, as well as not requiring the usage of an ODE solver, making it more efficient in terms of computing speed.

The constraint over the ϵ term inside (4) is necessary to ensure convergence. Intuitively, ϵ needs to be strictly positive for the states to change. The upper-bound Δ is the maximum degree in the graph limits the maximum influence on the node with the most neighbours. In a minimalist example, considering a strongly connected graph of two nodes A and B, if $\epsilon = \Delta = 1$, then A would have the value of B in the next state, and similarly for B. By plugging in the values inside the consensus algorithm with (6), divergence becomes apparent. A more generic proof is available in [1].

$$\begin{aligned} x_A[k+1] &= x_A[k] + 1 * (x_B[k] - x_A[k]) \\ &= x_B[k] \end{aligned} \tag{6}$$

To this point, the presented formulas were already available in the world of control theory. While the objective of the internship was also to understand

these topics clearly, the final goal is to understand whether we can achieve a distributed version of the consensus algorithm.

4.5 Achieving distributed consensus

Let us start by defining the meaning of distributed. Consider that the consensus algorithm is now used on multiple processors. Instead of having a central unit collecting the values of each processor (or agent), we would like the processors to only communicate their value to their neighbours instead periodically, removing the need for synchronization between all agents. Figure 4 tries to showcase the difference between centralized and distributed algorithms. In other terms, one agent could be computing its value at step $k+2$ while another one is still at step k. When this kind of behaviour occurs, would the consensus algorithm presented still converge properly ?

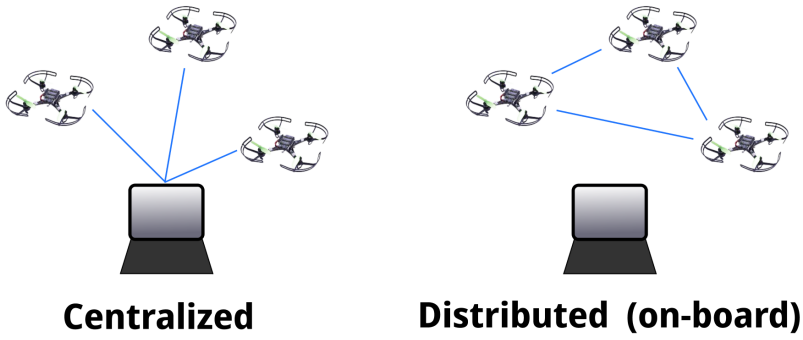

Figure 4: Difference between centralized and distributed computation

Without having found prior research on this algorithm in particular, a completely different approach was taken to solve this problem. With the current progression in the field of artificial intelligence, there are papers that exist (such as [5] on the convergence of distributed gradient descent. If we manage to find an equation to minimize that results in computing the consensus value, then we can use convergence results of distributed gradient descent.

Let us focus on a single agent i first, without considering the offsets d_{ij} (such that $\forall i, j \in V \mid d_{ij} = 0$). Recall that the general form of gradient descent is given by (7). The objective function for the classical gradient descent is f. As there is a single variable in our given case, we can simplify ∇f to $\frac{df}{dx}$.

$$\begin{aligned} x_{n+1} &= x_n - \eta \nabla f(x_n) \\ &= x_n - \eta \frac{df}{dx}(x_n) \end{aligned} \tag{7}$$

Looking back at the consensus algorithm in equation (1), all agents in the consensus algorithm try to minimize the difference they have with other agents,

thus it is a good candidate as an objective function. We will take the discrete-time version as described in (4) and compute its primitive, which gives us (8), referred to as $G_i(x_i)$. This objective function[5] is defined for each agent in the graph.

$$G_i(x_i) = -\frac{1}{2} \sum_{j \in N_i} (x_j - x_i)^2 \tag{8}$$

$$\dot{G}_i(x_i) = \sum_{j \in N_i} (x_j - x_i) \tag{9}$$

We can see that $\dot{G}_i$ is equal to the right-hand term of the discrete consensus algorithm in (4). The global objective function for distributed gradient descent can be defined by the sum of all agents' objective function, such that $f_{global}(x) = \sum_{i \in V} G_i(x_i)$. By redefining η and noting that $f = G_i$, we can rewrite the local objective function of one agent i, which gives (10).

$$\begin{aligned} x_{n+1} &= x_n - \eta \nabla f(x_n) \\ &= x_n - \eta \dot{G}(x_n) \\ &= x_n + \epsilon \sum_{j \in N_i} (x_j - x_i) \\ &\text{where} \quad -\eta = \epsilon \end{aligned} \tag{10}$$

Finally, each step of the gradient descent becomes a step the discrete time consensus algorithm of (4), so we may use convergence results of the distributed gradient descent algorithm. Because [1] proves the equivalence of continuous and discrete-time consensus algorithms, we have shown that both versions of consensus can converge properly.

5 Ensuring safety using Control Barrier Functions (CBF)

The software provided by the laboratory requires velocity commands to be sent to the quadcopters to control them. While we have an algorithm to achieve formations, we need to ensure that the drones avoid each other when flying. Because the reference papers [2] and [3] define multiple notions, the main result behind CBFs will be given for conciseness.

[5]The objective function of each agent G_i actually holds the same formulation for each agent, this is only to put an emphasis on the fact that each objective function is deeply linked to the neighbours of each agent.

5.1 CBF definition with forward-set invariance

There are already ways of ensuring safety, by using path finding algorithm such as A* or RRT* to determine where the drone should go over time, requiring a definition of collision with obstacles to avoid. The thought process behind CBFs is different, defining instead a safe set of positions (or more generally, a set of states) that the drone can attain. This set is defined by (11).

$$C := \{x \in \mathbb{R}^n \mid h(x) \geq 0\} \tag{11}$$

where $h(x)$ is a smooth function used to define the safe set of states. In our case, $h(x)$ can simply define collision between the drone and an obstacle (e.g another drone), as this is what we want to avoid. Thus whenever $h(x) \geq 0$ there is no collision occurring.

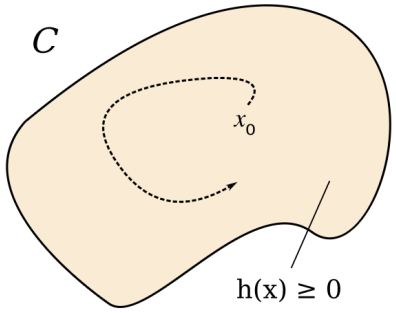

Figure 5: Representation of the set forward invariance property, applied to the CBF $h(x)$, in the safe-set C, when the drone starts at position x_0

The idea is to use the property of forward-invariance of a set to ensure safety. Figure 5 is a diagram representing this concept. When inside the working set, set forward-invariance will ensure that the state x of the drone (its position for example) never leaves the set C. Using the findings in [3], if the function $h(x)$ satisfies the following condition[6] 12

$$\dot{h}(x) + \alpha(h(x)) \geq 0 \tag{12}$$

then $h(x)$ is a *Control Barrier Function* (or CBF) and ensures that the set C will be forward-invariant, i.e. after entering the set once, any output from the control function $u(x)$ will stay in the set C. (12) is referred to as the CBF condition. α is referred to as a K-class function, meaning it is strictly increasing and $\alpha(0) = 0$. While this is defined in the general case, you can satisfy this condition by making α act as a coefficient, changing the CBF condition to (13).

[6]The paper in [3] defines control barrier functions in a more generic way using the Lie derivative. In a 3-dimensional state space, the generic formula has no additional value and was removed for ease of understanding.

$$\dot{h}(x) + \alpha h(x) \geq 0 \tag{13}$$

This satisfies the K-class function requirement, and will be used later in section 5. With this definition of collision, we now need to design a velocity-input controller that uses the CBF.

5.2 Optimization-based controller for the CBF

A simple way to determine collision is to use the distance to a circular obstacle, i.e. given a rock at location p_o with radius D, we can define $h(x) = \frac{1}{2}(||x - p_o||^2 - D^2)$, defined arbitrarily here because it is simple to derive. This safe set C only defines the safe set of positions the drone may attain, but we still need to change the velocity input $u(x)$ using condition (12). To do so, [2] explores the use of Quadratic Programs to change as little as possible the original control input. In short, it tries to solve the following QP program given in (14).

$$u^*(t) = \arg\min_{u \in R^2} || \ u(t) - u_{nom} \ ||^2 \qquad s.t \ \ (*) \tag{14}$$

The constraints of this QP program are the CBF conditions defined in (12) for each obstacle in the world the drone moves in. That way, we can achieve obstacle avoidance in a minimally invasive way, using solvers for the defined QP program.

5.3 Discretization of CBF

Another student from ENSEIRB-MATMECA in the laboratory, Paul RAJOT, was working on implementing a discrete-time version of the CBF technique. Professor Ibuki has requested the usage of his results, which are based off of [4].

Without detailing all the steps[7], the essence of his findings is that the CBF condition can be adapted into a discrete-time compliant variant (15), that includes the factor dt into the condition.

$$\dot{h}(x) + \frac{\alpha}{dt} h(x) \geq 0 \tag{15}$$

6 Experimental application on quadcopters

With the help of consensus and CBF together, there are enough tools to be applied in a real-life experiment. The laboratory provides an experiment area, the quadcopters to use and a vision-based system to locate them using multiple cameras. All of the configuration is already set up for use by students, alongside some basic skeleton in Python to send velocity orders to each drone used. We will dive into how to implement the discrete-time consensus algorithm and CBF for the experiment.

[7] Discrete-time CBF was not one of the main focus of this internship, so the time left was focused on the subject I was given instead. The goal was only to use the results of Paul RAJOT.

6.1 Computing the driving speeds

Consider that the position (x, y) of a drone is equivalent to the state $x[k] \in \mathbb{R}^2$ of a consensus agent at time step k. The discrete consensus algorithm in (4) directly drives the state value $x[k]$, so it cannot be implemented as such. Looking at both terms, the next state $x[k+1]$ can be interpreted as taking the current state $x[k]$, added with the sum of vectors from $x[k]$ towards the value of each neighbour, minus a relative offset. This is showcased by figure 6, where the red vector is the resulting sum of vectors to each neighbour.

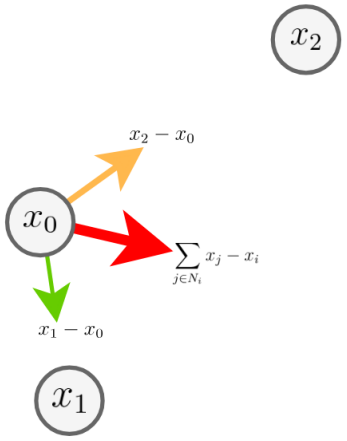

Figure 6: Three agents where the next state $x_0[k+1]$ is being computed, while linked to the other nodes. Vector lengths are normalized and their norm is shown by color instead, from green (smallest) to red (biggest).

Thus this sum of vectors can be used as a speed vector to drive a drone with, defining our control function $u[k] \in \mathbb{R}^2$ as (16).

$$u_{nom}[k] = \epsilon \sum_{j \in N_i} x_j[k] - x_i[k] - d_{ij} \tag{16}$$

We refer to $u_{nom}[k]$ as the nominal output speed to drive a drone at time step k. This value is then passed to the CBF quadratic program defined in (14), where the obstacles are every other flying drone, with the control barrier function $h(x)$ defined in section 5.2. This outputs an optimal speed $u^*[k]$ that achieves formation and avoids other drones at the same time. The complete control law is summed up in figure 7.

The drones' positions are then centralized as a single state vector $x[k]$ by a central computer, and the algorithm runs again. Since the drones of the laboratory cannot be reprogrammed (as the code used inside the drones is proprietary), distributed consensus could not be tested.

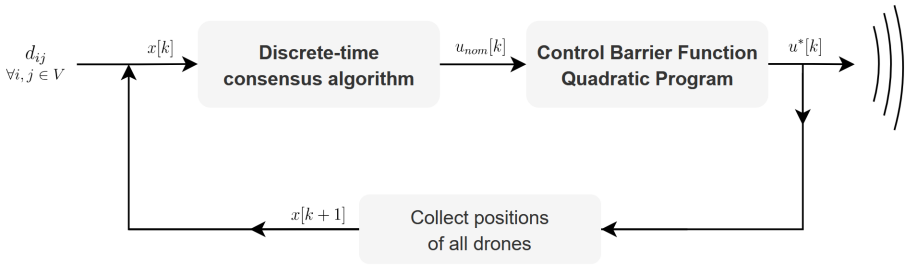

Figure 7: Control law diagram of the system used to command the quadcopters. Speed output $u^*[k]$ is then sent using a radio antenna to all drones in the real-life experiment.

6.2 Simulation on holonomic robots using grSim

Testing the algorithms implemented directly in real-life is risky, considering that the price of one drone is up to 600 euros. Before hurting the drones, the code was tested on the grSim simulator [6], where we can directly send velocity inputs to holonomic robots, so the system is very similar to the drones. The code used to send commands to the simulator was taken from GitHub [7] and slightly adapted to gain time. After confirming that there were no errors, we moved on to set it up for the real drones.

6.3 Real experiment on drones

Using the configuration provided that works on the ROS (Robot Operating System) library and an Optitrack-based capture system, the control law was ran asking the drones to form a diagonal in the air. Everything was implemented using the Python programming language, as well as the base code that was provided for drone control. The drone used is the Crazyflie 2.1 model from the Bitcraze company.

Considering the ground is the (x, y) plane and z the height, formation control was performed on the (x, z) coordinates of each drone, while driving their y coordinate to a fixed position (so that the airflow produced by their propellers). The results obtained are as follows, with the pictures in 8 showcasing the formation.

Figures 9 and 10 display respectively the percentage of achievement of formation and the drones positions over time. The achievement rate was computed by calculating the square error between the sum of offsets to attain, minus the sum of actual offsets between drones, based on the neighbours graph G. At the end of the experiment, drones were close to the ground, affecting their flight, which may explain the difficulty to maintain a high achievement percentage.

A notable fact when looking at the locations of the drones over time is that they start to go down at a certain point. Because their z coordinate was driven by the consensus algorithm, there was no external input that requested the

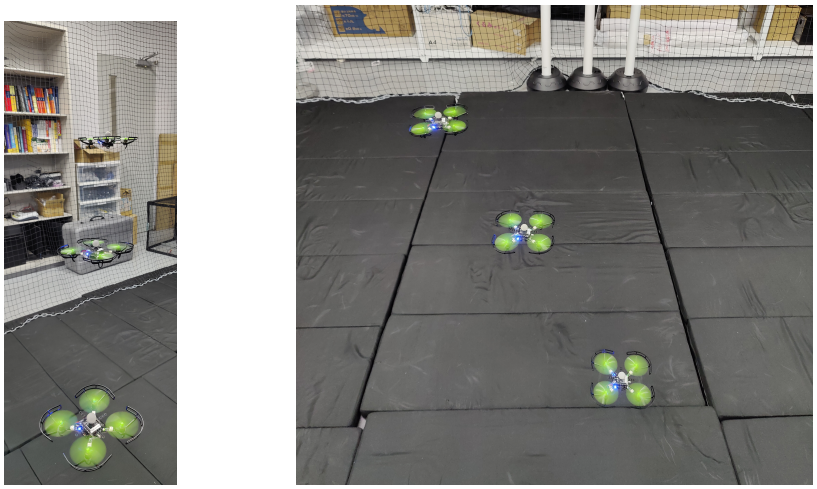

Figure 8: Pictures of the three drones after having attained their formation

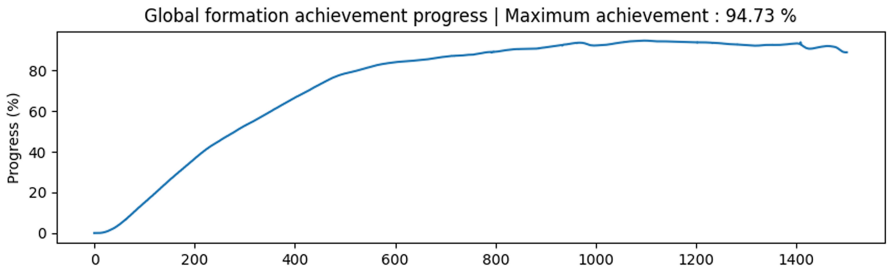

Figure 9: Percentage of achievement of the progression by the three drones over time. The air conditioner was turned off for this test as the drones are very sensitive to its airflow.

drones to stay at a certain height. One of the drones started falling slowly, and every other drone in the formation responded by changing their height. This is a direct consequence of the consensus algorithm, and is exactly the behaviour targeted. We now have a relative formation that is maintained over time even if the drones are slightly disturbed.

7 Conclusion and evaluation of the objectives attained

Implementation of formation control based on the consensus algorithm was a success, and an advancement on proving the convergence of the distributed consensus algorithm has been found, so these objectives are complete as they were validated by Professor Ibuki. There are still advancements that can be made,

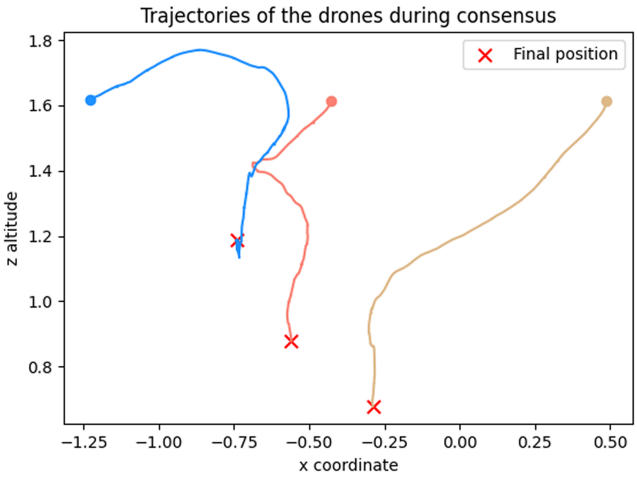

Figure 10: Positions of the drones over time. The neighbours graph is the same as in figure 1. Consensus was ran with value $\epsilon = 0.3$, and the CBF condition coefficient $\alpha = 0.3$

such as controlling the entire formation using a common input to all agents, or comparing the efficiency of the usage of CBFs instead of path planning.

Although the consensus algorithm converges properly, a study of the convergence speed based on the connectivity of the graph, as well the value of the coefficient ϵ in the discrete-time version, could be further studied. Plus, the CBF is not adapted for usage if the drone enters a collision zone, so other ways of achieving collision avoidance could be explored.

The sketch of proof provided in this report could be reworked to ensure mathematical accuracy, as there wasn't enough time left to thoroughly study papers that discuss the convergence of the distributed gradient descent algorithm. Nonetheless, this gives a step in the right direction.

8 Personal conclusion

First of all, I'd like to remind the reader that I come from the IUT of Bordeaux, so I have a weak level in mathematics compared to other students. I selected this internship because it included control theory and a reasonable amount of notions that I did not know of, so I wanted to challenge myself to improve my general ease of understanding, as I was more familiar with numerical examples than formulas.

This internship was very fulfilling but required me to invest a lot of time re-reading the same papers, understanding the intuition when reading formulas or equations, and learning new notions in general, notably in control theory. I could've implemented the algorithms by just following the equations, but this

was not my goal. I managed to get used to reading and manipulating formulas over time, reading multiple different papers to get different point of views on the same subject, and also discovering new notions. In my opinion, the objectives are attained and are satisfying enough (though I did give up on understanding the Lie derivative, brackets and algebra).

This is the first report I've ever written where the main focus is around mathematical formulas and not code architecture. As I come from a technician diploma, I am very glad to have been able to improve my theoretical knowledge and toolbox, because I assume I have performed enough technical development solely revolving around architecture and code.

The management style in the laboratory is very different but adapted to my style of working. Appointments with Professor Ibuki are optional but can be requested, but since he would come from time to time in the laboratory to check up on my work, I was able to progress easily. This may not be suitable for someone who requires (or appreciates) being overseen regularly.

I am grateful for this opportunity and very happy with the results and work I have provided. Thank you to Meiji University, ENSEIRB-MATMECA, Mathie CHEVRIE and Professor Tatsuya Ibuki for this opportunity, as well as the students of Meiji University that helped me with my subject.

9 CREGE appendix

This section will focus on the project lifecycle in Professor Ibuki's laboratory as well as the studies of Meiji University students : how do new students join the laboratory, what is the selection process, and how are the projects defined ?

9.1 Bachelor years

During the third year of their bachelor degree, students will have the opportunity to visit the different laboratories available in the campus, across the different buildings. They get to learn more about the daily life inside each laboratory, the topics that are studied, and the different events and experiments conducted. It is mandatory for them to select one to finish their bachelor diploma. After that, each student must select which laboratories they would like to go to, ranking them by preference. Whether they get their first choice or not depends solely on the grades they obtained during their studies (a system similar to Parcoursup in France).

At the end of April, the start of the new school year, the selections end and each student (now in their 4th year of bachelor) is assigned to a laboratory. Students continue to have regular classes, but a new class is added to their schedule. This class is managed by the reference professor of their assigned laboratory. Each laboratory is free to choose the learning topic, some decide to make students read papers, where others conduct experiments. The following explanations are specific to Professor Ibuki's laboratory.

In the case of Professor Ibuki, control theory is the main topic in the laboratory. He provides a standard class on this topic so that students get the basic knowledge, learning how to run Matlab simulations, as well as conducting the experiments in the laboratory. He also teaches the mathematical tools students will need for their projects : linear control theory, algorithms and theory, as well as convergence theorems.

Alongside this class, students are given two "graduation projects" that they must fulfill to graduate. Professor Ibuki uses the first month of the first project to let students decide on their project. Students must read papers, read about the current state of the art around their project, and learn about the current research. With this collection of data, the first graduate project consists of achieving the same results presented in the reference papers studied. The second graduation project is focused on the same research topic, with small incremental steps that consist of actual research from the students.

Additionally, students are required to participate in a common meeting every Monday, as each student needs to show his or her current progress on his topic of research, helping students to collaborate together and help each other. Apart from this mandatory meeting, students are free to come or not at the laboratory to progress on their graduate projects. Professor Ibuki trusts the seriousness of each of his students, and lets them organize themselves as they please. Not checking attendance of students is actually not the standard at Meiji University, and many other laboratories do not follow this. If one of the students needs assistance, they can schedule a meeting with Professor Ibuki to receive assistance.

9.2 Carrer pursuit or master years

At the end of their fourth bachelor year, students of Meiji University have multiple choices. They may either go work for a company, or pursue a master's degree. This degree can be performed in another university, or continued inside Meiji University. In fact, in the Electronic Engineering department (which Professor Ibuki's laboratory belongs to), there are on average 256 fourth year bachelors. Among them, 30-40% of students remain in the university, pursuing the master's degree in the same laboratory. 5% of students choose to leave, to aim for a higher ranked university, and more than 50% decide to work at a company. In Professor Ibuki's case, there are 8 fourth year bachelors in average each year, and half of them stay to pursue a master's degree.

But this is only true for Meiji University. For example, at the most prestigious university of Japan, called the University of Tokyo (often referred to as "Todai"), 99% of students will choose to stay to pursue their master's degree. This was the case of Professor Ibuki. This difference can be explained by the high education fee required to study at Meiji University. Most students need to work a part-time job to continue studying there.

Let's focus on the future of master's students inside Meiji University. Most of the time, master students will keep the same project topic they studied during their bachelor years, so as to iterate on it and improve it. Again, this is particular

to Meiji University. At the University of Tokyo, students are required to change supervisors and their research topic.

The general consensus in the laboratories at Meiji University is that their master's students should attend at least one domestic to present their own research. This is not an official rule but most laboratories enforce it, as it is considered a small requirement. Although if the student continues with a doctoral degree (or PhD), official rules stipulate that he or she will have to publish two papers and attend two international conferences.

Master students continue to have standard classes, but they put more focus on the master's thesis. Professor Ibuki opens opportunities for his students to participate at domestic and international conferences. This way, Meiji University students contribute to the world of research by achieving small steps, which can sometimes be presented at domestic or international conferences around control theory, on the recommendation of Professor Ibuki.

10 References

[1] R. Olfati-Saber, J. A. Fax and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," in Proceedings of the IEEE, vol. 95, no. 1, pp. 215-233, Jan. 2007, doi: 10.1109/JPROC.2006.887293

[2] A. D. Ames, X. Xu, J. W. Grizzle and P. Tabuada, "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," in IEEE Transactions on Automatic Control, Aug. 2017, doi: 10.1109/TAC.2016.2638961

[3] A. D. Ames et. al, "Control Barrier Functions: Theory and Applications", 2019, http://arxiv.org/abs/1903.11199

[4] A.Agrawal and K. Sreenath, "Discrete Control Barrier Functions for Safety-Critical Control of Discrete Systems with Application to Bipedal Robot Navigation", Robotics: Science and Systems XIII, https://roboticsproceedings.org/rss13/p73.pdf

[5] B. Swenson, R. Murray, H. V. Poor and S. Kar, "Distributed Stochastic Gradient Descent: Nonconvexity, Nonsmoothness, and Convergence to Local Minima" in Journal of Machine Learning Research, volume 32, no. 328, pp. 1-62, 2022, http://jmlr.org/papers/v23/20-899.html

[6] grSim is a simulator of holonomic robots developed for the RoboCup. More info on https://github.com/RoboCup-SSL/grSim
Reference paper : Monajjemi, Valiallah & Koochakzadeh, Ali & Ghidary, Saeed. (2012). grSim – RoboCup Small Size Robot Soccer Simulator. 7416. 450-460. 10.1007/978-3-642-32060-6_38.

[7] Original code written by Grégoire Passault on GitHub. The adapted version of his code used in this internship can be found at https://github.com/Wanchai290/ssl_traj whereas the original code can be found at https://github.com/Gregwar/ssl_traj