Rescue Using Constrained Drone Formations

Pedro José Loureiro Neves
pedro.jose.neves@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

Jul 2019

Abstract

War and technological advancements have always had a visible correlation throughout human history. From something as simple as duct tape, to huge game-changing inventions such as the internet, many of humanity's greatest creations were ironically born from their inclination towards destruction.

The drone is a *warborne* technology that has seen great breakthroughs during the later half of the past century. While drones are already part of the modern battlefield scenery, front-line resupplying and medical evacuation are still often performed by human operators and under dangerous circumstances. This dissertation showcases a possible solution: A formation of UAVs specifically tailored to act in these harsh environments, where being shot down is a very real possibility. The group is to adapt in real time to drastic changes in its flight condition.

This thesis focuses on the need for this flight formation to be able to rearrange itself and fulfill its goal with less agents than it had at mission start, while possibly sharing a payload. As such, there's a restriction to the group's flight pattern: it has to follow rules favouring the transportation of a common cargo. The solution has to take into account that, in these specific situations, the agents' relative positions to each other leave very little room for maneuver.

A Multi-PID based quadcopter controller was created, working in tandem with an originally designed autopilot that effectively drives the drone from point A to point B, including takeoff. It offers hovering capabilities, has built-in error correction and shows tolerance to changes in mass, to an extent. A controller was designed to calculate the drones' objective coordinates. This allows for multiple flight patterns and is able to output tightly-knit formations, ensuring the agents' relative positions to one another. Furthermore, the simulations with a drone being shot down mid-flight show the agents faithfully respecting their bounding restrictions and a new flight pattern being swiftly applied.
Keywords: Drone, UAV, Flight Formation, Multi-UAV load transportation, PID controller

1. Introduction

This thesis proposes studying a constrained Multi-UAV Formation for (originally) military applications. The solution consists in a flock of drones flying in tight formation, applying built-in redundancy techniques to increase mission fulfillment probability. It focuses on the real-time realignment of the flight formation in the case of the sudden loss of one/some of its agents while still flying towards the mission's destination. These realignments, and all the flight patterns formed have to allow for the possible hauling of a body, bound to the group by cables. This load, being it cargo or an actual possibly injured person, is to be successfully transported from point A to point B in a rough belligerent environment and admitting real-time changes to the mission plan. This system has to be able to both takeoff and cruise in these harsh conditions, in a tightly-knit formation.

1.1. Motivation

The idea featured in this thesis originally came as an intended answer to the (nonsensical) disparity in some aspects of modern warfare.

In an age where war is fought mostly at the cost of materials and resources, a paradigm shift from the wars fought till this past century, it seems contradictory that many high-risk operations are still (unnecessarily) conducted by jeopardizing human lives. Situations such as front-line resupplying and medical evacuation - Medevac - are still carried out mostly by human operators, putting both theirs and possibly the injured subject's lives in more peril. This obviously collides with the Zeitgeist of leaving dangerous jobs to machines when possible.

The original idea came from an on-going DARPA project (the 'TX Program' [5] [2]) which had as its main objectives tackling "Combat Reconnaissance and Combat Delivery & Evacuation" [3].

The idea first came up on the topic of military medical evacuation but was instantly expanded to front-line resupplying due to the identical nature of both situation. The later being a special case of the first but with seemingly less constraints and symmetrical objectives: the cargo is being carried to the battlefield instead of away from it.

The military influence of this thesis is solely due to the DARPA project that gave birthed it, by no means restricting any possible usages for regular civil purposes.

1.2. Objectives

The main objective of this project is laying down a solid groundwork for a feasible solution, that might've been applicable to the DARPA contest.

The myriad of conditions and variables that a solution like this would face is countless and can't possibly be contemplated in a single project. It being an original idea, the solution would have to be built from the ground up, starting from a **Single Drone** and naturally growing to have **Multiple Drones**.

- **Single Drone:** To end up with a flight formation of multiple drones, it is a basic requirement to have agents that will act as expected to and when needed to. A system that feels completely natural to the user tackling the flight formation problem. A complex and responsive, yet familiar model that has to be flexible enough to adapt to new situations, maybe even receiving new features as the development calls for it. With these requirements, the choice was creating an original drone. This will have to both include a realistic model and a controller. It will be expected to follow any and all realistic orders given, effectively taking off from location *A*, and driving itself to place *B*. The objective is to have an effective transportation system that doesn't make any unnecessary deviations, has no drastic unscheduled changes in altitude, and is capable of closely hovering around target coordinates. It's expected to swiftly correct all detected errors in any of its 6 degrees of freedom. This means not only correcting its position in the world, but also its inclination or rotation in all axis while maintaining all flight capabilities.

- **Multiple Drones:** Having successfully deployed a fully controllable drone that fulfills the previous requirements, it's necessary to program a controller to ensure the mission's fulfillment. If the drone already flies from *A* to *B*, it is now necessary to have something that takes this *A* and calculates the *B*. One of the possibilities brought by using a group of agents instead of a single one is expandability and scalability. If a drone can't carry a load, maybe 2 or more working together can. As such, a flight formation for this group of quadcopters will have to take into account the possibility of physical restrictions beyond the many ones already present in nature. The hostile environment in which this system is to be deployed in also requires this controller to be adaptable to spontaneous changes that corrupt the formation. As such, the system will be expected to correct the drones' objectives in real time when faced with the sudden loss of an agent. All of this is to be performed autonomously, without any inputs from an external entity.

2. Drone Model

The drone dynamics were based of an already existing project, by an indian group [4].

There's a peculiarity in their model. Instead of the classic "rotors on the corners" quadcopter build, this drone had its motors on the sides, front and back.

While sounding counter intuitive, this actually gives birth to simplified momentum equations:

$$M_y = (F_1 - F_2) \times l \tag{1}$$

$$M_x = (F_3 - F_4) \times l \tag{2}$$

In order to use these dynamics in the intended kind of drones, some transformations were in order. Changing the X and Y axis and assuming a drone where the motors are placed on the corners generated the new equations that would be used throughout the project:

$$M_y = ((F_3 + F_4) - (F_1 + F_2))\frac{\sqrt{2}}{2}l \tag{3}$$

$$M_x = ((F_2 + F_3) - (F_1 + F_4))\frac{\sqrt{2}}{2}l \tag{4}$$

This is actually an assumption where the drone is considered to be symmetric alongside both X and Y axis.

The other forces and momentum equations remained unchanged from the indian model:

$$F_i = K_f \times \omega_i^2 \tag{5}$$

$$M_i = K_m \times \omega_i^2 \tag{6}$$

2.1. Model dynamics and projections

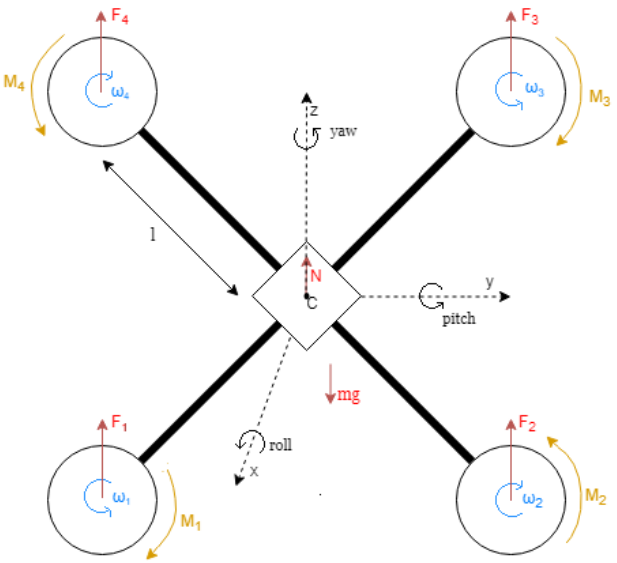


Figure 1: Drone Dynamics

2.2. Equations and constants

The equations above include some constants that haven't been explained: K_f and K_m. These are the proportionality constants that transform the rotation of the ro-

tors into their actual outputs (effective lift). They're inherent to the propellers' size, shape and weight. As such, they couldn't be chosen at random. Besides this, the equations to calculate the drone's Euler angles from the data already at hand still needed to be deduced from eq 3 and eq 4. Answers to this were taken from a small course [1] on drone control at the university. This was an introductory course on drone control and dealt with quadcopters on the same scale as the ones pictured for this project. As such, the same values were used in this model: K_f = 3.5897 $\times 10^{-5}$ & K_m = 1.2191 $\times 10^{-5}$. It also helped arriving at the Euler angle equations:

$$\begin{cases} I_{xx} \times \ddot{\theta} = M2 + M4 - M1 - M3 \\ I_{yy} \times \ddot{\phi} = M3 + M4 - M1 - M2 \\ I_{zz} \times \ddot{\psi} = M2 + M3 - M1 - M4 \end{cases} \tag{7}$$

These equations actually come from a simplification, where:

$$\begin{bmatrix} I_{xx}, I_{xy}, I_{xz} \\ I_{yx}, I_{yy}, I_{yz} \\ I_{zx}, I_{zy}, I_{zz} \end{bmatrix} \simeq \begin{bmatrix} I_{xx}, 0, 0 \\ 0, I_{yy}, 0 \\ 0, 0, I_{zz} \end{bmatrix} \tag{8}$$

This simplification indicates that rotations around one axis won't directly affect rotations around the others. While not completely true, the diagonal values were deemed to be sufficiently greater than the others to warrant the approximation. This greatly eases the equations.

From these, it becomes possible to deduce the accelerations at play around the drone's axis:

$$\begin{cases} I_{zz} \times \ddot{\psi} = M2 + M4 - M1 - M3 \\ F_i = K_f \times \omega_i^2 = m \times a_i \\ M_i = K_m \times \omega_i^2 \end{cases} \\ => \begin{cases} \ddot{\psi} = \frac{K_m \times m}{K_f \times I_{zz}} \times (a_2 + a_4 - a_1 - a_3) \\ \omega_i^2 = a_i \times \frac{m}{K_f} \\ M_i = \frac{K_m \times m}{K_f} \times a_i \end{cases} \tag{9}$$

$$\begin{cases} I_{yy} \times \ddot{\phi} = \frac{\sqrt{2}}{2} \times l \times (F_4 + F_3 - F_2 - F_1) \\ I_{xx} \times \ddot{\theta} = \frac{\sqrt{2}}{2} \times l \times (F_2 + F_3 - F_1 - F_4) \end{cases} \\ => \begin{cases} \ddot{\phi} = \frac{l \times m \times \sqrt{2}}{2 \times I_{yy}} \times (a_4 + a_3 - a_2 - a_1) \\ \ddot{\theta} = \frac{l \times m \times \sqrt{2}}{2 \times I_{xx}} \times (a_2 + a_3 - a_1 - a_4) \end{cases} \tag{10}$$

Integrating these values twice produces the Euler angles: Roll - θ, Pitch - ϕ & Yaw - ψ

3. Simulation of a Drone

The software chosen to represent and test the drone model was MatLab + Simulink.

The Simulink Block diagram firstly started as a simple representation of the drone model, but quickly evolved to a more realistic approach which included the possibility of adding noise to both the sensors and the rotors, and a "motor model" comprised of a saturation and a rate limiter. This greatly improves on a simplistic motor model and had to be added at the start as these limits heavily affect the performance of the PID controllers in the future. This first implementation already includes filters for the added noise. There's also access to every variable of interest.

3.1. Altitude Control

The first controller designed was the one handling the drone's altitude. Starting with only one degree of freedom facilitates the study of the drone's responses and how each property and variable affects its flight. Properties such the quadcopter's weight, the motors' limits, noise and air resistance will be tested throughout this section, in addition to its (the drone's) reaction to the altitude control. This makes it possible to get some understanding of how the quadrotor will behave when the complete control system is designed and being tested.

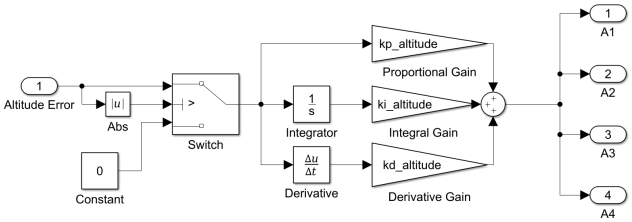


Figure 2: Altitude PID diagram

3.1.1 PID Tuning

The structure of this type of controller consists in attributing individual weights to both the altitude error, its derivative and its integral. A general rule of thumb is attributing a higher value to proportional gain (the first weight mentioned) than to the other 2. These 2 gains are actually not always found in these types of controllers when deemed not necessary, or more prejudicial than beneficial to the situation.

After getting some sensibility on each constant's influence on the systems response, positive outcomes started coming forward. The higher the values of the constants the higher response speed, but high derivative gains would often cause instability, high proportional gains would lead to overshooting, and high integral gains sometimes caused the system to over-correct, not stopping at 0 error.

Using these sensibilities gained from constant testing, "sweet spot" values were finally reached:

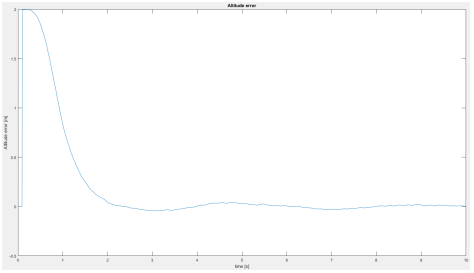


Figure 3: Altitude control for PID constants: K_p=1; K_i=0.004; K_d=0.9

3.2. Motor model imposed restrictions

The motors are simulated by a sequence of a Saturation and a ”Slew rate limiter” blocks. While this doesn't take into account the inputs a motor needs to be controlled (namely Voltage, for example), it does simulate well enough the effects a motor's restraints have in the real system.

3.3. Sensor noise

Assuming a robust enough control system and sensors of high enough quality (good readings without much noise), the actual effects of noise aren't too alarming. They're very present and visible, but tolerable:

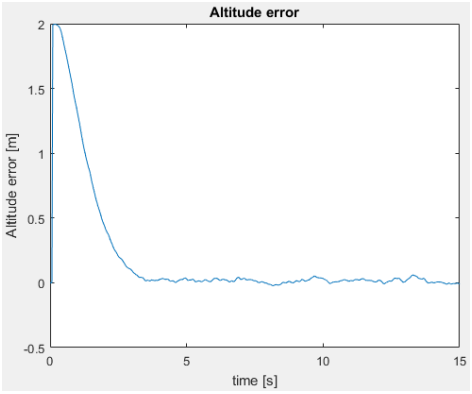
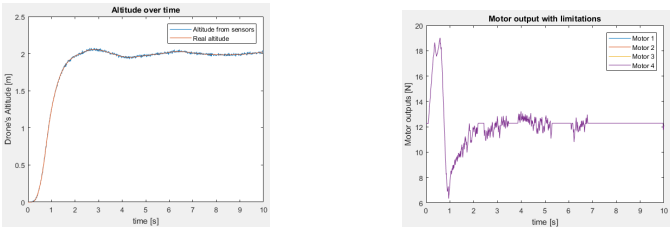


Figure 4: Altitude error test - with noise

3.3.1 Countermeasures - filters and Deadzone

As a response to the noise, a deadzone was created: A small area around ”0 error” where the controller would always assume 0 and not correct. For very small errors, these differences would likely be attributed to noise.

Additionally, median filters were installed right after where the error was measured.



(a) Altitude (b) Control signal

Figure 5: Altitude control with a 0.02 m error deadzone and median filter

3.4. Division of space into 2 control zones

This controller makes use of 2 PIDs for each euler angle to be handled. When far away from the goal, the drone is expected to fly swiftly in the target's direction. On the other end of the spectrum, when close to the target, corrections are expected to be slower but more precise.

The drone is considered to be in Zone *1* when it's far away from the target, and enters zone *2* when it gets close. The separation between the zones (i.e. the size/radius of zone 2 around the target) is decided by the user. This controller has the stock value for the radius of zone *2* at 1 meter. Altitude isn't taken into account for this measurement, so instead of a sphere around the target, zone 2 is actually a vertical cylinder centered on the goal coordinates. Another major reason for this division between ”far” and ”close” zones is the control of the yaw angle. It is possible for the drone to take off facing the opposite direction of the goal (an ”yaw error” of pi/2 rad). Of course the controller is expected to be able to correct this and turn the drone around. But when finally arriving at the destination coordinates, due to the nature of drones (not having any sort of direct braking system like for example *air brakes*) and also due to the quadcopter's momentum, overshoots are to be expected. Overshooting the target causes the drone to be momentarily facing the opposite direction of the target i.e. it instantly goes from around 0 ”yaw error” to pi/2 (or -pi/2) at which point the PID controlling the yaw angle begins to act (unnecessarily, and with actual hurtful effects). Dividing the space into 2 zones, which having their own controller/set of PIDs allows for a distinction between when to correct these errors and when not to.

This way, there's a set of PID controllers specifically made for when the drone is flying through zone 1 and another for zone 2. They don't work in tandem and switch automatically from one to the other when the frontier (between zones) is crossed - unlike cascading PIDs.

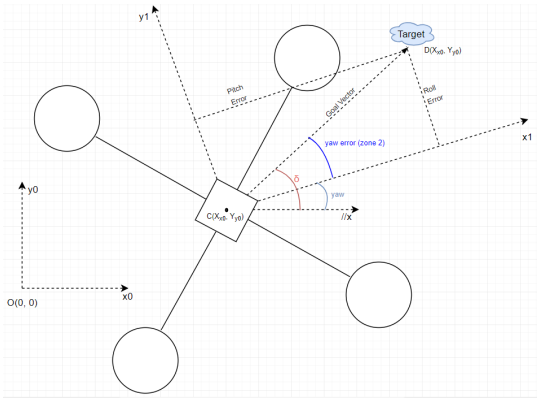


Figure 6: Error calculations in zone 2

$$\begin{cases} Goal\_Vector = D - C \\ \delta = atan2(D_Y, D_X) \\ yaw\_error(zone2) = \delta - yaw \\ Roll\_Error = |Goal\_Vector| \times sin(yaw\_error(zone2)) \\ Pitch\_Error = |Goal\_Vector| \times cos(yaw\_error(zone2)) \end{cases} \tag{11}$$

3.5. ”Yaw” control

Creating the controller for the drone's yaw angle came in obvious succession to the the altitude controller. At this point it made no sense to advance the controller that would handle the pitch angles to make the drone move

forward, if there were no guaranties the drone would be moving toward the right direction.

The steps taken were in everything similar to the ones followed for the altitude PID. There was some interference from the sensor noise, but not to a level where it'd become too disruptive. In any case, a deadzone was also put in place for this PID controller, to avoid stress on the motors from over-corrections. After some PID tuning, satisfactory responses were achieved:

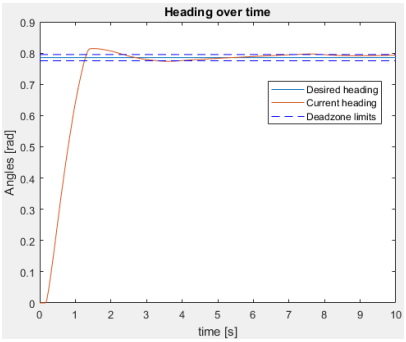


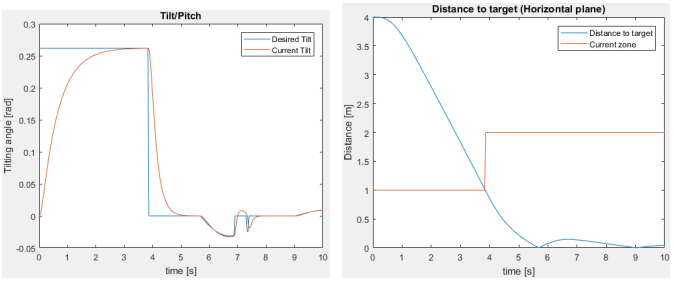
Figure 7: Response of the system to a pi/4 [rad] yaw angle reference after PID tuning

The deadzone was settled at $\pm$ 0.01 rad ($\pm$ 0.57 degrees).

3.6. "Normal" control

The drone's normal is controlled by 2 of its Euler angles: Pitch and Roll. As such, a controller was created for each of these. Due to the differences between zone 1 and 2, 4 PID controllers were created in total. Following the same steps as in for the previous PIDs, after some tuning good convergences were achieved for both.

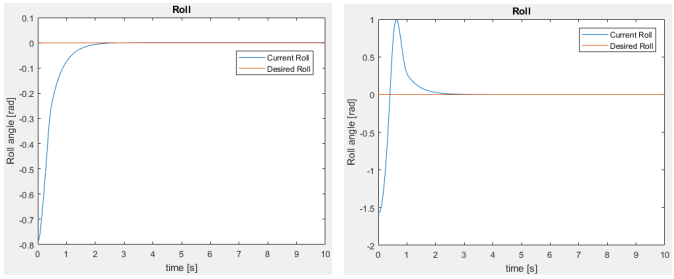
3.6.1 Pitch control



(a) Pitch Correction (b) Distance to target

Figure 8: Pitch angle control

3.6.2 Roll control



(a) Initial roll angle = pi/4 rad (b) Initial roll angle = pi/2 rad

Figure 9: Roll angle control

3.7. PID tuning algorithm

In order to optimize the PID constants' value and their tuning speed an algorithm was devised.

For each PID variable being tuned, a cycle was created. This cycle would take a chosen stock value for that variable (already known to show some results, while not yet satisfactory) for its middle point and perform a linear sweep of all the values around it.

To arrive at the best PID values from these iterations, a cost function was devised. The chosen function was the sum of the root mean square of the error in each point. This seemed more punishing for both slow and overshooting solutions and gave less weight to small errors:

$$cost = \sqrt{\frac{\sum e^2}{t}} \tag{12}$$

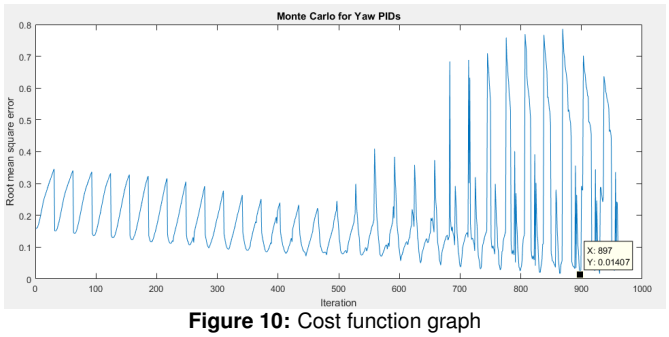


Figure 10: Cost function graph

4. Flying in Formation

When talking about flight formations, people instinctively think of either flight shows or migrating birds formations. The second is a good example of the advantages of flying in formation. The "V" shape easily recognized in migrating flocks is known to reduce drag and thus greatly improving aerodynamic efficiency. This allows for longer flight times with less energy expenditure. The same principle can be used for man-made aircraft, and is already its own study field.

In the specific case of multiple airborne agents carrying a common cargo, the necessity of a clearly defined and strictly followed flight formation can't be overlooked.

An assumption was made that the most obvious way of flight formation for a flying vehicle assembly would be following a symmetrical geometric pattern. In the case of agents with omnidirectional flight capabilities such as quadcopters, and as per the possible cargo carrying constraint, the flight formation was settled in the shape of a regular polygon with as many sides as the number of drones partaking in the formation. This polygonal flight pattern would be centered on a (virtual or not) load being carried and allow instant changes of direction while avoiding severe damage to the flight formation caused by the sudden shift in the weight due to said cargo's momentum.

This pattern then allows for 2 types of construction. One where there are 2 drones leading the formation, and another with only one.

4.1. Trajectory following possibilities

Besides the 2 types of construction, 4 different "behaviours" were also considered for the trajectory following. The first 2 consist on patterns that don't adapt to curves, "static" flight patterns. This means the formation will be facing the same way independently of the direction it's actually flying towards. This is made possible and easy due to the quadcopter's omnidirectional flight nature.

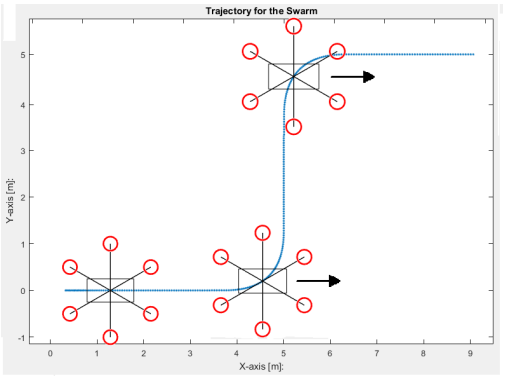


Figure 11: Example of a 'static' flight pattern

The remaining 2 flight behaviours could be described as dynamic. One of them has the formation always facing the tangent of the trajectory i.e. turning alongside the curves of the trajectory itself. This solution involves lots of "unnecessary" movement by the drones and doesn't fully make use of the advantages of using quadcopters as its elements. The final behaviour has the group always facing the destination, independently of where it's located. This solution has a clear advantage against the previous one when it comes to energy and movement waste.

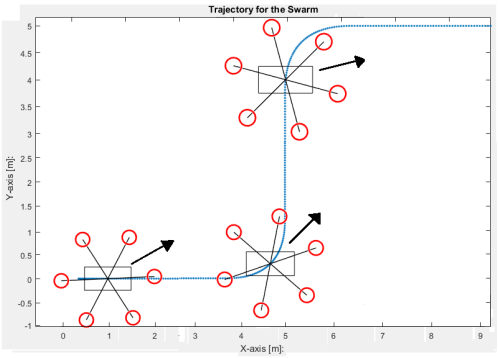


Figure 12: Example of a 'dynamic' flight pattern

4.2. Real-time adaptation to sudden loss of random agent

This step does not stop at simply choosing new positions for the drones to occupy in space. In this specific case, there are some aspects that require extra strenuous effort: Having a mass binding the drones by cables restricts their range of movement, and more importantly, each drone not keeping its cable stretched is a cause for extra stress to the other drones, that will find themselves hauling even more weight. This way, when driving the agents to their new position after an agent loss, measures have to be taken into account so that the new trajectories enable all the drones to keep their cables stretched while changing position, to prevent further unbalance when carrying cargo:

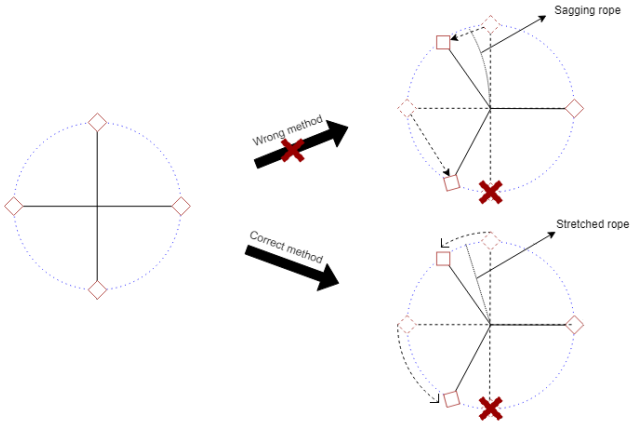


Figure 13: Circular trajectory to keep cables stretched (after loss of one drone)

In the even of a drone loss, the assembly will create a new formation. Again based on a regular polygon, this time with N-1 sides while still respecting the path-following algorithms. This process is repeatable while there are enough drones to produce the lift necessary to keep the group in the air. The way to keep the cables stretched while moving to the same positions implied an arc-like trajectory. Each drone out of place follows a circular path to its new position. All these arcs are superpositioned with a circle of a defined radius. Having all drones move along a circumference of unchanging radius allows for the reduction of the problem from 2 degrees of freedom to one, in polar coordinates. Fixing the radius,

all positions can be defined by their angle relative to a reference direction - the *Azimuth*:

$$\sigma = \frac{2 \times \pi}{N} \tag{13}$$

With σ being the angle between each drone on the circumference they create, and N the number of drones at the moment.

This means the i-th agent (a_i) will occupy the position given by $\sigma \times (i-1)$ (as the indexes start at 1 and not 0) + an offset that's either 0 or σ/2.

4.3. Drone Position: Relative and Absolute coordinates

To get the relative X'Y' coordinates for each agent there's a transformation function from the polar coordinates:

$$a_i[X', Y'] = [radius \times cos(\sigma), radius \times sin(\sigma)] \tag{14}$$

where the radius can be decided either directly or indirectly:

$$S = radius \times 2 \times sin(\sigma/2)) \Leftrightarrow radius = \frac{S}{2 \times sin(\sigma/2))} \tag{15}$$

With "S" being a variable that controls the "safe" space between the drones. This way, the group will constantly try to keep a distance of *S*[m] between each drone to avoid inner collisions.

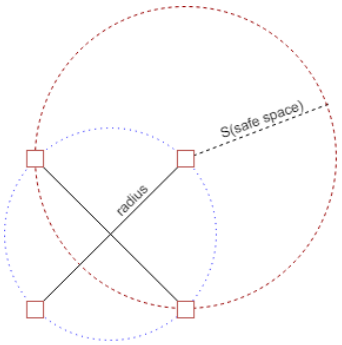


Figure 14: Representation of the "Safe space" and the "radius" of the group's formation

Having the means to obtain the relative coordinates of each drone, they need only to be transformed into their actual coordinates in the world's coordinate system:

$$a_i[X, Y] = a_i[X', Y'] + C[X, Y] \tag{16}$$

Here 'C' represents the real coordinates of the group's center of mass. This equation performs a linear translation of each agent's coordinates.

4.4. Adaptive trajectory following

Applying the techniques above, and the algorithm that calculates the new positions for each drone in the case of a sudden loss:

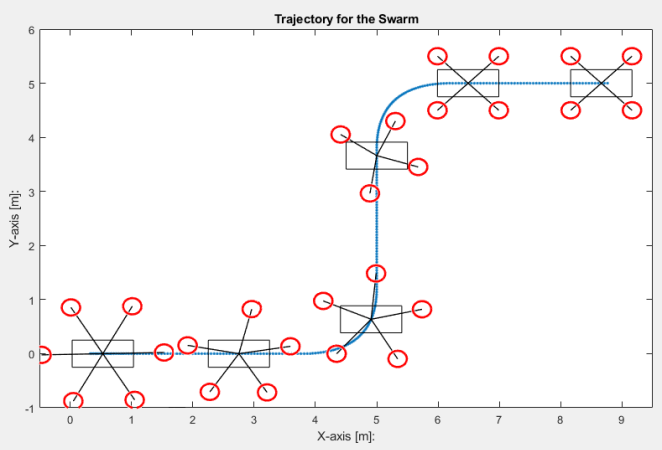


Figure 15: "Dynamic" flight behaviour taking into account random casualties

5. Group Flight

In the previous simulations, some pre-processing was done via a script that was run *before* the simulink simulation. This script calculated some of the constants the drone model would use that are still needed, but most importantly, it also plotted its fixed trajectory.

In the case of the whole assembly, some of these pre-calculations have to be taken away and replaced by real-time data processing: If the objective is to have an adaptive system that reacts to unexpected inputs, extra on-site processing is required. This includes the real-time plotting of the trajectory for each drone in the group, provided that some agents might be lost during the flight mission. This highlights the necessity for the system to have the power to adapt each agent's own trajectory, allowing for the formation to finish its mission as a whole.

This type of on-site approach allows for some liberty and swiftness in the reaction to outside inputs as no communication to a home-base is required, given the final objective is still the same.

5.1. Flying in Formation

In the previous chapter, a few flight formations were studied. Their geometrical nature provided great compatibility and ease of implementation with the already existing blocks. A circular space of a fixed radius was virtually drawn around the formation's center. This circumference was then equally divided into as many parts as there are active agents, in real-time, with every iteration.

5.1.1 Trajectory Following as a Flight Formation

It's possible to draw a parallelism between this batch setup and the previous situations with just one drone. Abstracting the individual agent's coordinates, and treating the formation as a monolith, enables treating the new *simpler* paradigm as a regular trajectory following situation.

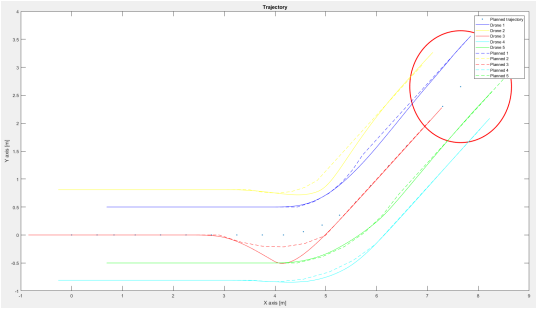


Figure 16: Trajectory plotted in 2D

One of the reasons the deviation from the drone colour coded in Red (agent 3) is so exacerbated is that its own trajectory already had a visible outwards ”hump” before and during the curve. This happens due to the ’flight pattern’ chosen. It was explained before that 2 types of ”behaviours” were studied - Static and Dynamic. This simulation employed the last one, causing what can be considered extra and unnecessary movement. The other reason for this deviation was an error in the ’Normal’ error calculation, on the roll angle component: In earlier stages, during the implementation of the 2 zones of flight, it was decided that roll error corrections were unnecessary in zone 1, and were implemented for zone 2 only. That ceased to be the case here, and roll error corrections were promptly added:

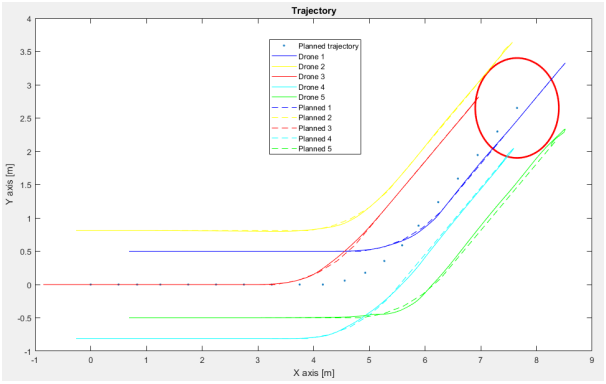


Figure 17: Group’s trajectory with Roll correction

Each agent is still a complete and autonomous drone, with the exact same model perfected in the previous chapters. This means each drone still implements the 2 ’zones of flight’. And agents that are further ahead and closer to their local goal aren’t being fed new objectives (while some are still too far behind). This way, faster drones find themselves entering ’zone 2’ of flight, where they automatically tune down their acceleration.

This synergy allows for the flight formation to perform a primitive but effective form of self-regulation of its agents, not allowing any to get too far ahead of themselves and keeping the formation tight around the intended geometric shape.

5.1.2 Real-time reaction to sudden agent loss

As no changes in altitude were intended for this simulation, the 2D formation control algorithm could be directly adapted to the 3D simulation without any extra mapping.

Only one drone is made to ’fall’ in these simulations, though the algorithm does account for scalability. Which drone falls, and when, were randomly selected variables:

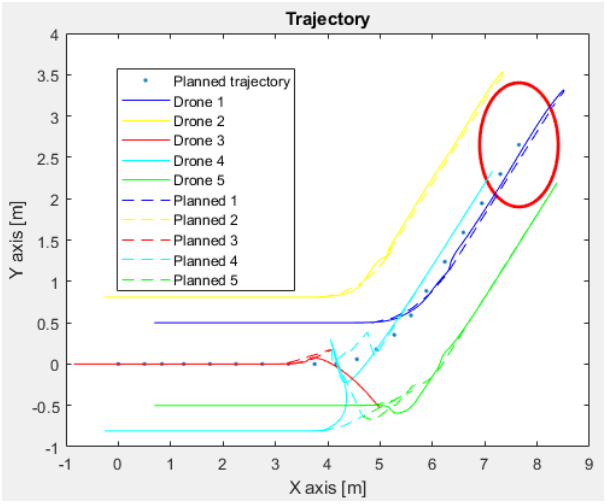


Figure 18: Drone 3 is disengaged

5.2. Algorithm application to different trajectories

The previous chapter applies all the knowledge gathered so far in order to arrive at a system that successfully completes its objectives. But it does so under an ever unchanging trajectory. This kind of testing often incurs in overfitting, where the controllers start being fine-tuned to a specific set of course characteristics. To make sure this wasn’t the case, and also to further validate this model, several more tests were run. These used different kinds of trajectories, and kept the randomness of the drone falls:

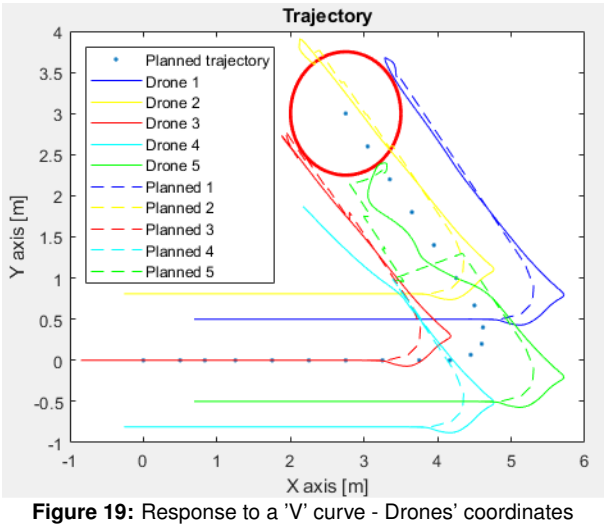


Figure 19: Response to a ’V’ curve - Drones’ coordinates

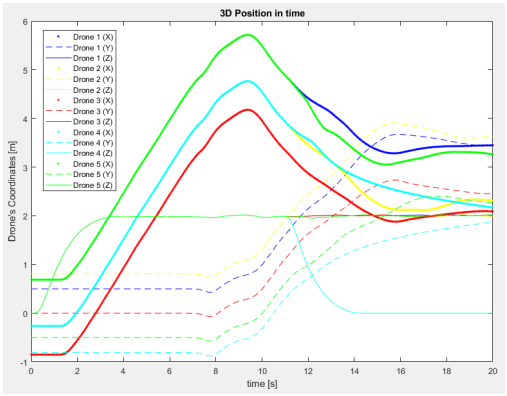


Figure 20: Response to a 'V' curve - Trajectory from above

5.3. Flying in Formation - with extra cargo

An attempt was made to bound the drones to a load below them, via cables:

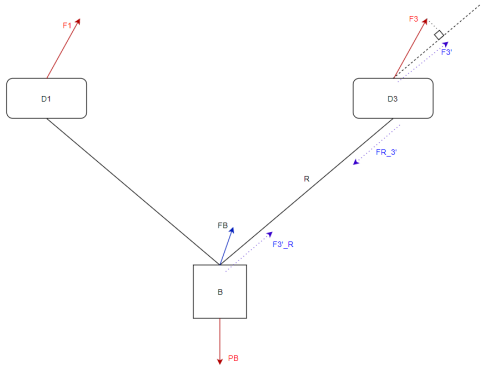


Figure 21: Cross-section of the formation carrying a cargo and its Dynamics

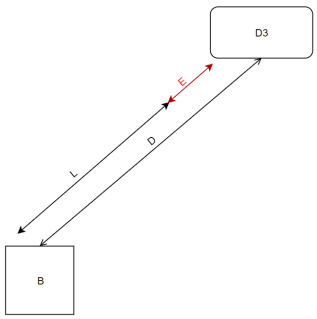


Figure 22: Representation of the cable

$$L - E <= D <= L + E \tag{17}$$

F_i is the force the agent i is generating, F_i' are the components of these forces on the Rope, i.e the effective force the drone exerts on the cable. $F_i'\_R$ and FR_i' are the forces exerted by the cable on the body and the agents, respectively. FB is the resultant of all the forces applied to the Body:

$$FB = PB + \sum F_i'\_R \tag{18}$$

$$F_i'\_R = F_i' + k_e \times (D - L) - k_d \times \frac{D - L}{dt} \tag{19}$$

The current model was unable to provide enough lift to carry an extra cargo. Boosting the drones' lift generation would bring them outside the bounds for which they'd been programmed/tuned for, which would cause negative outcomes, i.e. mission failures.

Testing the solution at an altitude where this extra lift was no longer required (where the load effectively doesn't leave the floor) did generate some results, albeit unusable ones:

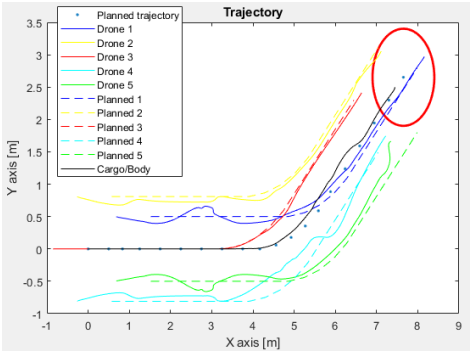


Figure 23: Cargo carrying simulation without enough lift

6. Mission accomplishment statistical study

It seemed plausible there were some correlations between the timing of the drone fall and the outcome of the mission.

As such, a statistical study was performed, with N=2000 where N equals the number of iterations the simulation is repeated.

First, these 2000 experiences were divided between "successes" and "failures". A success would correspond to a simulation where the formation reached its objective and failures when it wouldn't. These were repeated to both situations with noise and without.

Noiseless simulations originated no good graphs, with an overall accomplishment rate of 96%. And the remaining 4% were all situations where the group was 1 second away from the objective, but simulations with noise yielded different results:

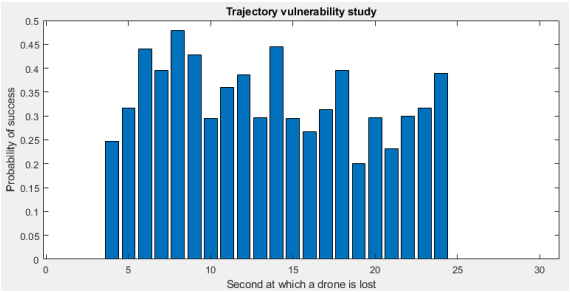


Figure 24: Probability of success by second of failure. N=2000

It's dangerous to take many illations from this, but an argument could be made there's a pattern. A slight drop starting at around second 9 or 10, which would coincide

with when the group is starting the curve - a sensitive section of the trajectory. The results also slowly start climbing when the drone falls only on the last seconds of the simulation. These situations are progressively getting closer to the simulation without a loss, one with arguably better chances of success.

The position of the sacrificed drone on the flight-formation seems to also influence the outcome of the simulation:

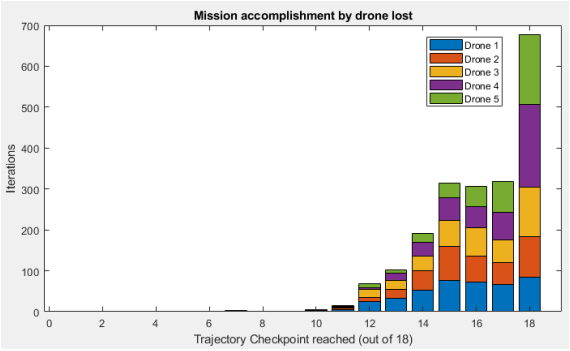


Figure 25: Group's progress by drone lost. N=2000

A quick look at the trajectory plotting algorithm shows a plausible answer for this correlation. The algorithm operates by representing the drones as entries in an array. When one falls, it's removed from the array, and the entries that come both before and after i.e. the drones around it try to fill in the blank left. No exception was made for the first or last drones, which only had drones on one side (due to the representation as an array, with a defined beginning and end). It's possible that remaking this algorithm in order to have this vector act as a closed circle, where the first and last agents are as connected as any other, would show visible improvements for the worst case scenarios as drone 1 falling.

7. Conclusions

With autonomous technology continuously overtaking dangerous or repetitive jobs away from humans, this project intended to showcase a possible way to make use of current **autonomous systems** and expand it to an area where it still hasn't displayed its full potential: **Front-line resupplying and medical evacuation**. The solution presented here consists on a set of specially designed autonomous **quadcopters that are able to fly in a tight formation and self-adapt** to the sudden loss of any of them, redeploying themselves in a new pattern favourable to the mission's completion.

The first half of the project required the development of a quadcopter able to takeoff on its own, arrive at the objective's coordinates without great deviation from the plotted course and be able to keep itself there. The final solution was able to **visibly pass all these requirements consistently**, not failing to deliver even when lifting off from uneven grounds. During the roll and pitch angle tests the **drone would always successfully takeoff and balance itself**, given a starting angle within common sense. In some cases it would even recover from a 90º starting tilt.

The second half took instances of the model obtained and deployed them on the same simulation space connected by a mutual controller. The deployed solution effectively generated objective coordinates that placed each agent on the right position to design a pre-programmed desired flight pattern. The controller showed the capacity to consistently keep this tightly-knit formation from mission start to end with little to no deviations from the goal. These **undisrupted simulations held a 100% mission accomplishment rate**, albeit in unrealistically ideal conditions. Introducing drone failures to the system expectedly dropped the completion rate, though only slightly. The group would take longer to reach its goal, but it would do so consistently, after losing any one of its agents and at any given point in the mission. The accomplishment rate dropped to 96% in simulations without noise and 40% with, although this number held a high correlation to the length of the simulation. Longer simulations having higher probability of success and shorter ones undoubtedly lower. **67% of the simulations with noise reached a state of at least nigh-completion**. In all these situations, drones that weren't crashed on purpose would **never crash on their own**, always having little to no deviations from their target altitude. Every simulation would have reached its objective given enough time, but the ones that took longer were occurrences where one or more agents temporarily deviated too far from the formation. These were purposely considered unsuccessful missions.

Attempts at carrying cargo by tying it up to the drones via cables were unsuccessful, excluding situations where the load was being dragged through the floor. These produced high probability of reaching the target destinations but were effectively mission failures.

References

[1] Simulação e controlo de drones. Instituto Superior Técnico.

[2] New darpa project focuses on future vehicles, armor. CompositesWorld, Jun 2010.

[3] S. Ackerman. Darpa offers a million dollars for crowd-sourced amphibious tank. Wired. co. UK, 2012.

[4] P. N. S. . L. S. Sakhare, A. Quadcopter dynamics. Cultures Of Communication, November 2016.

[5] B. Spice. Nov. 9: Darpa chooses carnegie mellon to develop autonomous capability for "flying car". Carnegie Mellon University, November 2010.