MONOGRAPH

QUADROTOR CONTROLLER FOR COOPERATIVE INTERACTION WITH JUGGLER

Gabriel Pires Iduarte
Luan Haickel Araújo

Brasília, May 2021

ENGENHARIA MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Undergraduate Course in Control and Automation Engineering

MONOGRAPH

QUADROTOR CONTROLLER FOR COOPERATIVE INTERACTION WITH JUGGLER

Gabriel Pires Iduarte
Luan Haickel Araújo

Report submitted as a partial requirement to obtain the degree of Control and Automation Engineer.

Brasília, May 2021

FICHA CATALOGRÁFICA

Gabriel, Pires Iduarte
Luan, Haickel Araújo
QUADROTOR CONTROLLER FOR COOPERATIVE INTERACTION WITH JUGGLER,

[Distrito Federal] 2021.

xiv, 40p., 297mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação - Universidade de Brasília. Faculdade de Tecnologia.

1. Quadrotor	2. Juggle
3. E-FRIT	4. Crazyflie
I. Mecatrônica/FT/UnB	II. Quadrotor Controller For Cooperative Interaction With Juggler

REFERÊNCIA BIBLIOGRÁFICA

Iduarte, GP, Haickel Araújo, LH, (2021). Quadrotor Controller for Cooperative Interaction with Juggler. Monograph in Control and Automation Engineering, Publication FT.TG-nº 16, Faculty of Technology, Universidade de Brasília, Brasília, DF, 54p.

CESSÃO DE DIREITOS

Dedicatórias

Á você, leitor.

Gabriel Pires Iduarte

Aos meus pais, Dalila e Gorgonio.

Luan Haickel Araújo

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Dalila e Gorgonio, que sempre tiveram a saúde e educação dos seus filhos como primeira prioridade, sendo assim, inquestionavelmente, os principais responsáveis por eu chegar até aqui.

Agradeço à minha irmã Mariah que sempre me apoiou da maneira que pode. Chegou a me emprestar uma máquina da gráfica dela para eu tentar usar para fabricar peça de drone, e o pior, conhecendo os riscos.

Agradeço ao meu parceiro neste trabalho, Gabriel, que pela segunda vez na mesma graduação, aceitou um convite meu para trocar de tema de TG, com sua única pergunta sendo: "Como faz pra trocar?".

À Carol, que sempre me apoiou, e foi uma grande companhia nestes últimos meses. Especialmente nesta reta final quando, por mais de uma vez, ficou até tarde da noite corrigindo meu inglês através de uma tela.

Ao meu melhor amigo, Tarzan, que não apenas acreditou em mim desde a primeira vez que falei neste projeto, como também foi o primeiro a colocar a mão na massa junto comigo para fazê-lo acontecer, e certamente estará no projeto até o fim.

Ao nosso orientador, professor Edil, que mesmo em campanha para pegar menos trabalho devido ao cargo de chefia, aceitou me orientar neste projeto maluco, quando eu ainda não tinha nem um parceiro.

Ao nosso coorientador, professor Fernando, que não apenas aceito de bom grado nos orientar, como foi quem desvendou e nos explicou os mistérios dos métodos que queríamos usar, mas não entendíamos.

À toda minha família que, de sangue ou não, de perto ou de longe, sempre compreenderam e apoiaram minhas escolhas, raramente reclamaram da minha ausência e nunca negaram nenhuma ajuda.

A UnB... Nunca pensei que agradeceria uma instituição, mas essa universidade foi a responsável pelo que hoje parece ser a maior transformação da minha vida. Foi graças a ela que empreendi na Mecajun, onde passei a me enxergar como um líder. Foi também na UnB que me doei para o CATRON, onde aprendi que algumas batalhas são muito grandes para serem travadas sozinho. Também na UnB foi onde competi com e pela DROID, no ambiente mais estimulante e desafiador que já estive, e do qual eu sinto mais saudades. As vezes até eu me esqueço, mas foi na UnB, mesmo antes de ser aluno, que mergulhei de vez no universo circense, parte tão importante da minha vida, que inclusive motivou este trabalho. Mesmo agora perto do fim, quando eu já pensava apenas em me formar, a UnB continuou a me

surpreender, quando, através da Descontrolada, me apresentou uma nova geração de pessoas incríveis, que eu tenho certeza de que cuidarão muito bem da minha universidade.

E, por último, mas talvez mais importante, agradeço a todos os meus amigos, com os quais eu as vezes converso sobre meus sonhos e ambições, como que para ver se eu não estou louco em achar que consigo... E eles sempre realimentam as minhas loucuras. Eu sei que na cabeça deles, pensam que acreditam em mim porque eu faço o impossível, mas a verdade é que eu faço o impossível porque eles acreditam em mim.

Luan Haickel Araújo

Eu acredito que toda e qualquer interação que tivemos com outra pessoa é como um pequeno tijolo que foi usado para construir quem somos e que também pavimentou nosso caminho até aqui, aonde chegamos, portanto, se eu fosse agradecer devidamente a todos que me ajudaram a chegar aqui...bom, precisaria de muitas páginas, mas vou me esforçar para fazer caber todas as pessoas mais importantes.

Primeiramente agradeço a minha dupla e amigo, Luan Haickel Araújo, quem me convidou para essa jornada, com esse tema interessante. Seu apoio para esse finalzinho de graduação deveras cansativo foi importante. Obrigado por ter sido sempre paciente com meus eventuais atrasos (me desculpa, não eram de propósito).

Agradeço aos meus orientadores Prof. Edil e Prof. Fernando por todo o apoio, conhecimento e dedicação para reservar espaço em suas rotinas puxadas para nos orientar.

Agradeço aos funcionários do SG11, em especial ao ex-técnico João, que me ensinou a não perder meus dedos ao usar uma furadeira de bancada, Seu Cicero e ao Valter.

Agradeço a todos os membros da equipe de competição de robótica DROID, por todos os momentos de aprendizado e felizes que me forneceram.

Agradeço a todas as minhas amizades, sejam elas feitas antes ou durante o curso, principalmente as minhas amizades de qualidade duvidosa. Não vou agradecer um a um não, se acharem ruim, mandem reclamações para o Milton.

Agradeço a todos os meus familiares pelo apoio que me deram, em especial a minha tia Solange por fornecer moradia, aos meus pais Verni e Nilso que sempre me apoiaram e deram todo o suporte necessário para que eu focasse em estudar agradeço ao meu irmão Gustavo Pires Iduarte que foi uma peça fundamental que me permitiu ser quem eu sou hoje e a agradeço a todos os demais familiares que me apoiaram.

Finalizando, espero que todos vocês consigam trazer o melhor de si à tona, lembre-se que vocês são feitos de partículas com bilhões de anos, feitos até com átomos forjados em estrelas, façam jus as suas origens.

Gabriel Pires Iduarte

ABSTRACT

This text presents the work progress of the project which aims to create an autonomous juggling prop using a quadrotor and a carbon fiber structure. The text also details the current status and future plans, describing the next steps to be followed.

In the first chapter we introduce the main goal of this project. And this work will start such a project aiming to answer the most fundamental questions about its viability. Also, in this chapter we present a smaller goal in order to be the first big milestone of the project.

In the second chapter we will describe the Crazyflie, the platform we chose. We also will describe the format, materials and manufacture method chosen for making the dodecahedron structure that will allow the drone to be handled during flight. In the end of the second chapter, we will describe why the new structure requires the drone controller to be calibrated.

In the third chapter we will describe the cascade PID implemented in Crazyflie firmware. Then we will discuss some approaches from how to recalibrate the control system of the drone followed by a detailed description of the FRIT and E-FRIT algorithms, the chosen methods to be used in this work.

In the fourth chapter we will present our methodology and its results, showing how well the regulated the controller respond and how it allows the drone to fly with the new structure. Finally, in in the last chapter, we will discuss those results and compare with our initial goal. Also, in the last chapter we will present some suggestions for future work.

Keywords: Drone, juggling, E-FRIT, Crazyflie.

RESUMO

O presente texto apresenta a evolução do trabalho cujo objetivo final é a criação de um malabar autônomo, para tal será utilizado um drone que possui uma estrutura de fibra de carbono. O texto não só detalha o estado atual projeto, mas também descreve o futuro do mesmo, descrevendo os próximos passos a serem seguidos.

No primeiro capítulo, começamos contextualizamos a interação de humanos com drones citando como exemplo o filme SPARKED. Apresentamos a contribuição do nosso objetivo principal que é uma apresentação de fato interativa e não uma coreografia pré-configurada. Além disso, neste capítulo, apresentamos um objetivo mais modesto para ser a primeira grande meta do projeto: Simular malabares entre duas pessoas substituindo a segunda pessoa pelo planejamento de trajetória do drone, ou seja, após a primeira pessoa lançar o drone, o mesmo irá planejar sua trajetória de modo que simule o lançamento da segunda pessoa.

No segundo capítulo descreveremos o drone escolhido, Crazyflie. Descreveremos brevemente suas características, como a linguagem de programação usada no firmware e o rádio usado para a comunicação do drone com client software. Também apresentamos os possíveis sensores que podem ser usados. Descreveremos a estrutura que será usada para permitir que a pessoa possa manipular o Crazyflie, ou seja, vamos apresentar o formato escolhido, os materiais e o método de fabricação escolhido para fazer a estrutura em forma. No final do segundo capítulo, descreveremos que devido a adição da nova estrutura o controlador com parâmetros de fábrica já não é o suficiente para que o drone decole, portanto devemos reajustar os seus parâmetros.

No terceiro capítulo, apresentaremos de forma sucinta o que é um controlador. Descreveremos o controlador PID e o PID em cascata implementado no firmware Crazyflie. Em seguida, discutiremos sobre as coordenadas de referência usadas pelo drone e apresentaremos a implementação do controle do mesmo. Abordaremos então os requisitos de escolha e os possíveis método de calibração de controlador, e em seguida explicaremos o método escolhido para o nosso trabalho, E-FRIT. Finalizando o capítulo, apresentaremos alguns detalhes relevantes para a implementação do E-FRIT.

No quarto capítulo apresentaremos nossa metodologia. Como abordamos o fato de que o E-FRIT é válido apenas para sistemas SISO enquanto o drone é do tipo MIMO. Também apresentaremos nossa bancada de testes e em seguida, mostramos que não é necessário ajustar os parâmetros do controlador de velocidade angular e que o controlador de posição angular se tornou um sistema oscilante na bancada de testes. Mostramos nossa escolha de função de transferência desejada e como a encontramos. Como não é possível usar o E-FRIT em sistemas instáveis, tivemos que trocar os valores originais do controlador de posição angular para que o mesmo se torne ao menos estável. Com os dados, aplicamos o algoritmo E-FRIT, mostramos nossos resultados, discutimos o quão bem o controlador regulado responde e como ele permite que o drone voe com a nova estrutura. Finalmente, no último capítulo, discutiremos esses resultados e os compararemos com nosso objetivo inicial. Além disso, apresentaremos algumas sugestões para trabalhos futuros.

Palavras-Chave: Drone, malabares, E-FRIT, Crazyflie.

CONTENTS

AGRADECIMENTOS v

ABSTRACT viii

RESUMO ix

CONTENTS xi

FIGURES LIST xii

ALGORITHMS LIST xiii

SYMBOL LIST xiv

CHAPTER 1 – INTRODUCTION 1

1.1 BREAKING DOWN THE PROBLEM 2

CHAPTER 2 – HARDWARE 4

2.1 STRUCTURE 6

CHAPTER 3 – CONTROLLER ALGORITHMICS 10

3.1 REFERENCE FRAMES 12

3.2 CONTROLLER FIRMWARE 14

3.3 TUNING METHODS 15

3.4 FRIT 16

3.5 E-FRIT 19

3.6 E-FRIT IMPLEMENTATION 20

CHAPTER 4 – PERFORMANCE EVALUATION 23

CHAPTER 5 – CONCLUSIONS 31

6.1 FUTURE WORKS 31

BIBLIOGRAPHY 33

ANNEX I - THE E-FRIT SCRIPT 36

ANNEX II - CHOOSING TD 39

FIGURES LIST

Figure 1- SPARKED: A Live Interaction Between Humans and Quadcopters [2] 1

Figure 2 - First interaction goal proposal. 3

Figure 3 - Crazyflie 2.1(A), Crazyradio PA 2.4 GHz USB dongle (B), Flow deck v2 (C), Multi-ranger deck (D), Lighthouse positioning deck (E), Lighthouse V2 base station (F). [6] 4

Figure 4 - Block diagram of the Crazyflie firmware Kalman Filter [9]. 5

Figure 5 - Complete assembly of the dodecahedron without the drone. 6

Figure 6 – The 3D model made to confirming that the calculated size of the structure would fit the drone inside. 7

Figure 7 – The 3D model of the vertices connections to be printed, the dimensions are in millimeters. 7

Figure 8 – The complete assembly 3D model of the dodecahedron without the drone. 7

Figure 9 - 3D model of the parts for coupling the structure to the Crazyflie. How it docks to the drone and the dodecahedron can be seen in Figure 10. 8

Figure 10 - Complete assembly of the dodecahedron structure with the drone docked. 9

Figure 11 - Illustration of a rola-bola with an θ angle to be compensated. 10

Figure 12 – Simplified blocks diagram of rola-bola 10

Figure 13 – PID blocks diagram 11

Figure 14 – Block diagram of a simplified version of the Crazyflie cascaded PID control structure [19]. 11

Figure 15 – Bottom of the Crazyflie main board, that also is the main structural component of que drone. 12

Figure 16 - Crazyflie body reference frame [22]. 13

Figure 17 - Block diagram of the detailed operation of the firmware controller. This diagram is not available from the manufacture, it was made by us through a reverse engineering of the Crazyflie firmware. 15

Figure 18 – Diagram block of a generic closed loop system. 16

Figure 19 – Crazyflie, without the dodecahedron, in the test stand that's restrain all the degrees of freedom except the pitch one. 23

Figure 20 – Response to a step input from the attitude pitch rate PID controller of the quadrotor in the constrained test stand without the dodecahedron struct (in (A)) and with the structure (in (B)), in both cases with the original PID parameters. 24

Figure 21 – Response to a step input from the attitude pitch PID controller of the quadrotor in the test stand without the dodecahedron structure (in (A)) and with the structure (in (B)), in both cases with the original PID parameters. 25

Figure 22- Response of the drone without the structure, System response, on the test stand with input, r_0, on the pitch attitude PID, with original factory control parameters. Contribution of the roll and yaw rate attitude PID have been disabled. 26

Figure 23- Simulated response of the desired transfer function, T_d response, in pitch axis. 26

Figure 24 - Response of the drone with the structure, System response, on the test stand with input, r0, on the pitch attitude PID, with original factory control parameters, $\rho original$. Contribution of the roll and yaw rate attitude PID have been disabled. 27

Figure 25 - Response of the drone with the structure, System response, on the test stand with input, r0, on the pitch attitude PID, with $\rho 0E$ control parameters. Contribution of the roll and yaw rate attitude PID have been disabled. 27

Figure 26 - Response of the drone with the structure, System response, on the test stand with input, $r0$, on the pitch attitude PID, with $\rho *$ control parameters. Contribution of the roll and yaw rate attitude PID have been disabled. 28

Figure 27 - Crazyflie in about 3 meters height, outdoor flight, with structure and using $\rho *$ as controller parameters for the pitch and row attitude PIDs. 28

Figure 28 – Portion of data collected from drone pitch response in outdoor flight. (A) Drone with structure and original factory control parameters $\rho original$. (B) Drone without structure and original factory control parameters $\rho original$; (C) Drone with structure and control parameters to $\rho *$. 30

ALGORITHMS LIST

Algorithm 1 - Excerpt of the Crazyflie firmware's PID code implementation [24]. 14

SYMBOL LIST

Greek Symbols

φ	Pitch	[deg]
θ	Roll	[deg]
ψ	Yaw	[deg]
λ	Weighting coefficient	
Δ	Difference between the current instant and the previous instant of a variable	
ρ	Control parameter vector	

Acronyms

3D	Three Dimensions
ABS	Acrylonitrile Butadiene Styrene
ETH	Eidgenössische Technische Hochschule
E-FRIT	Extended Fictitious Reference Iterative Tuning
FRIT	Fictitious Reference Iterative Tuning
IFT	Iterative Feedback Tunning
IMU	Inertial Measurement Unit
LED	Light-Emitting Diode
LPS	Loco Position System
MIMO	Multiple-Inputs Multiple-Output
PCB	Printed Circuit Board
PID	Proportional Integral Derivative
VRFT	Virtual Reference Feedback Tuning
SISO	Single-Input Single-Output
SQP	Sequential Geometric Programming

CHAPTER 1 – INTRODUCTION

Juggle is present in human culture for at least 4000 years [1], and still, there is a huge potential for innovations in juggling performances, especially with the adoptions of modern technologies. Drones are great candidates to make appearances in modern juggling performances because of their capacity to fly precisely in any direction, and even stay still in the air. Indeed, drones have made some appearances in this area in the last decade. One of them is SPARKED [2], a film made by a partnership between Cirque du Soleil, Eidgenössische Technische Hochschule (ETH) Zurich, and Verity Studios. However, even in an amazing work like this, the juggler did not handle the flying props[1].

Figure 1- SPARKED: A Live Interaction Between Humans and Quadcopters [2]

The present work aims to change this scenario or, at least, start that change. We envision a juggling prop that can fly by itself and also be safely handled by the juggler like a regular prop, or as close as possible. If such a goal is achieved, the possibilities are endless.

Just like music, juggle have a well-defined rhythm, and there is a very precise way to mathematically express the juggle patterns[2], hence, we could easily program certain behavior into these props in a way that it will be intuitive to the juggler to interact with them. This mathematical way to express juggle patterns is called siteswap [3], and the quantization of the throws is basically based in which hand the prop will land on and how long it will take to do it. Therefore, in the first stages we will tell the props which kind of throw they should do and based on this information the juggler can incorporate that in his patterns.

[1] Props: The objects that are juggled [1].
[2] Patterns: repeating sequences of throw [3].

At this point we could make a choreography that could freeze in time, go backwards or in slow-motion, we could reverse gravity or change its direction, or turn it off entirely, all of that with the props in the air. However, the real magic will start when we manage to use the drone sensors, and maybe some external sensors as well, in order to predict in real time which kind of throw the juggler is intending. In that scenario, the algorithm will be able to take control of the prop right after it leaves the jugglers hand, and no matter the height, speed or path that the props follow, as long it lands in the right hand at the right time, it will not interfere in the pattern. Thus, the performances could have some level of freestyle interaction between the props and the juggler, which is very common in juggling presentations. And now we could do all those things mentioned earlier without having to precisely time everything. However, we can go even further. Note that the prop which was thrown is not necessarily the one that have to land in the juggler's hand. As long as the juggler always has a prop in his hand at the right time, the props can change positions in the air freely. Add the ability to make props appear and disappear by controlling on board LEDs and stage lights and will be very unlikely to someone to describe what is going on in the stage without using the word magic.

1.1 BREAKING DOWN THE PROBLEM

Although it is clear that some type of structure needs to be built around the drone to enable it to be handled in flight, many other questions come to mind: what kind of structure can we build around the drone that is dense enough to be handled? How well will the drone fly with such a structure? How can the drone detect when it is being handled in order to turn off its controller? How can the system keep track of the hand's position if the juggler starts to move in the stage? How silent can the prop be? Will it be possible to mask the sound? How portable can we make the system? Will it be possible to perform in any stage or only in pre-prepared ones? How long can the battery go? We could go on forever making questions.

In order to start to answer those previous questions, we will first propose a smaller and more tangible goal. In the Figure 2, we illustrate a more modest goal, a single throw where the drone task is to simulate an invisible second juggler that would catch it and throw it back.

In the scheme shown in Figure 2 we can see three stages in the trajectory. At the start of the movement the drone will be thrown by the juggler at a certain distance (blue trajectory). During this part of the path the drone must maintain its orientation (parallel to the ground) without interfering in the free fall trajectory intended by the juggler.

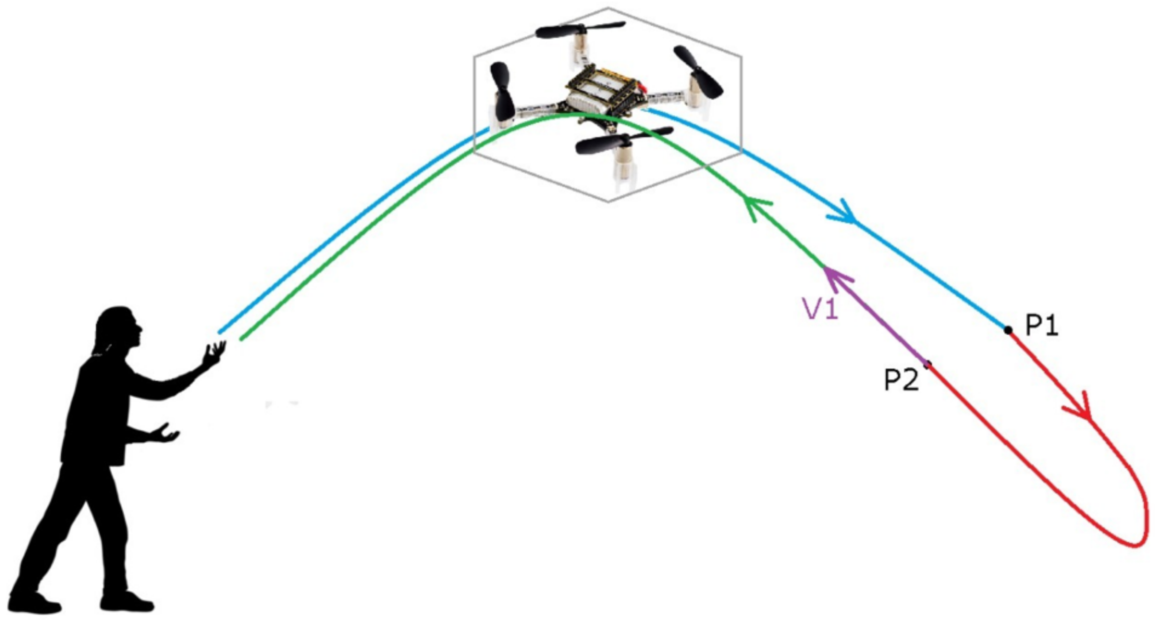


Figure 2 - First interaction goal proposal.

When reaching a certain distance from the ground, the system must change its mode of operation and plan a trajectory for the drone to execute in order to reach point P2 with a pre-determined speed V1 (red trajectory). And then, the drone must turn off the engines to complete the last part of the path in free fall, until it reaches the juggler again (green trajectory).

However even with this goal being more modest it is still too complex for the scope of this work. Thus, in this work, we are going to focus on the first two questions: What kind of structure can we build around the drone that is dense enough to be handled? How well will the drone fly with such a structure?

In the next chapter we will describe the chosen platform with all its advantages and drawbacks. We also will describe the format, materials and manufacture method chosen for making the structure that will allow the drone to be handled during flight. At the end of the second chapter, we will describe why the new structure requires the drone controller to be recalibrated.

In the third chapter we will describe the control architecture implemented in Crazyflie firmware. Then we will discuss some approaches from how to recalibrate the control system of the drone followed by a detailed description of the chosen method, and the key aspects of our script implementation.

In the fourth chapter we will present our methodology and its results. Finally, in in the last chapter, we will discuss those results and compare with our initial goal. Also, in the last chapter we will present some suggestions for future work.

CHAPTER 2 – HARDWARE

The chosen platform was a nano[3] drone, named Crazyflie [4] (Figure 3) sold by a company named Bitcraze [5], that is vastly used in researches with drones, especially in swarms of nano drones.

The Crazyflie is a ready-to-fly drone, its firmware is written in C++ and it can be uploaded via the included radio, it has out-of-the-box wireless log capability, and support to receive commands by a number of different ways, like a client software that runs in Linux, Windows and Macs, or a mobile app (Android and iOS), or python scripts using a library, all this software are open source and are available in the Bitcraze GitHub. This drone also supports expansion decks with automatic detection that add many sensors to the drone in addition to the inertial measurement unit (IMU) of the main board. All this and the small size of the drone made the Crazyflie the perfect fit for this project.

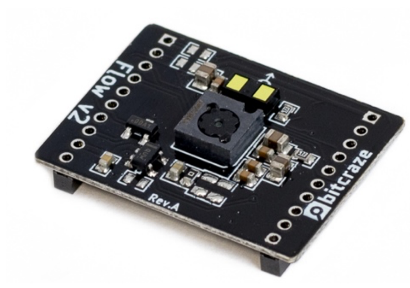
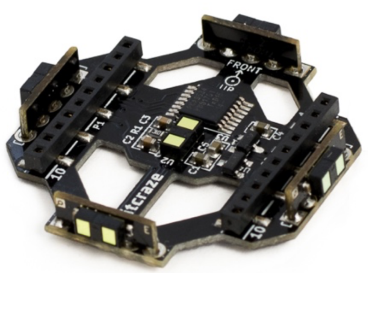
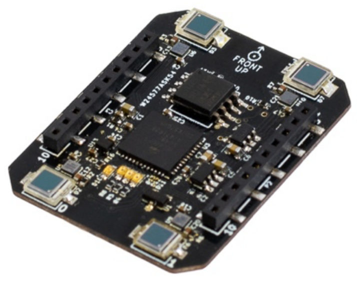
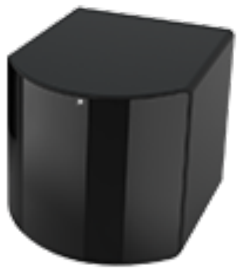
A	*B*	*C* 
D 	*E* 	*F* 

Figure 3 - Crazyflie 2.1(A), Crazyradio PA 2.4 GHz USB dongle (B), Flow deck v2 (C), Multi-ranger deck (D), Lighthouse positioning deck (E), Lighthouse V2 base station (F). [6]

Actually, we have one main concern with this drone, its thrust capability. With only 15g of recommended payload, we fear that it will not have enough power to do the necessary maneuvers with a structure coupled to it. However, if that's proved to be the case, we could use in the future a Crazyflie Bolt [7], which is basically a Crazyflie board without the motors

[3] Nano drone: weight las then 200g [39] [40]

and drivers. It would enable us to build our own quadrotor by choosing the appropriate motors and drivers, perhaps with the structure already permanently attached on it. Given that the Crazyflie Bolt has the same sensors and the same decks compatibility as the Crazyflie itself we expect that everything we develop for one would work for the other. Therefore, we acquire for this project a Crazyflie 2.1 with the radio, a flow deck, a multi-ranger deck, a lighthouse positioning deck and two lighthouse V2 base station, shown in Figure 3.

The flow deck is a deck put under the drone that has one distance laser sensor that measures the distance to the ground and one optical flow sensor that measure the xy-speed of the drone relative to the ground. With only this deck the drone is able to perform autonomous flight, although the accuracy rapidly decreases in high speeds [8].

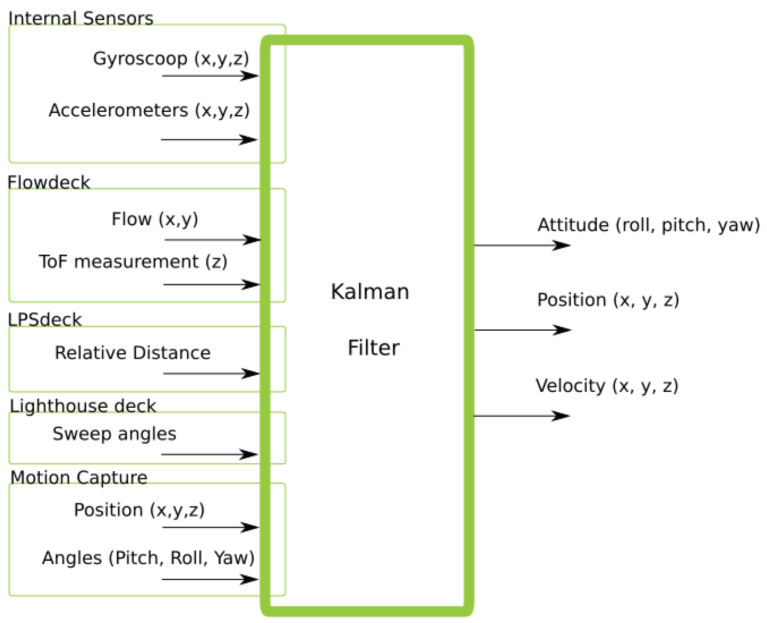


Figure 4 - Block diagram of the Crazyflie firmware Kalman Filter [9].

If the Flow deck proves to be sufficiently accurate for the planned maneuvers, it would be ideal to use only it, since all the sensors would be embedded, and consequently the stage would not have to be prepared with any markers or camera system. However, it is not prohibitive the usage of an external measurement system, if needed for the functionality. In that case, we will try to use the Lighthouse positioning deck with two SteamVR V2 base station positioned across the stage.

One can question why use the Lighthouse deck, that during the elaboration of this work it is still under development, instead of a method with safer results like a motion capture system with visual marks, which is the most used method for autonomous indoor flights [10] [11] [12] [13]. The advantage and decisive factor of the Lighthouse deck is the cost, which is a tenth of a motion capture system cost. The other solution offered by the Bitcraze for indoor navigation is the Loco Positioning System, LPS system, which has a similar price once the base station for the lighthouse is accounted for, however it is not an attractive option for the project, as it requires more markers than the lighthouse, which results in a more expensive system.

2.1 STRUCTURE

The structure that will go around the drone needs to meet certain criteria: be light-weighted, don't block the air flow and enable safe handling. The maximum recommended payload weight of the Crazyflie 2.1 is 15g [4], and the weight of the Flow deck V2 and the Lighthouse positioning deck are 1.6g [14] and 2.7g [15], respectively. Whereas it is possible to use both decks simultaneously, therefore the available payload comes down to 10.7g. After some tests and inspired by the work of Bruno Gabrich e David Saldaña [16], we assembled a dodecahedron with carbon fiber rod as edges and 3D printed connections as vertices, that can be seen in Figure 5.

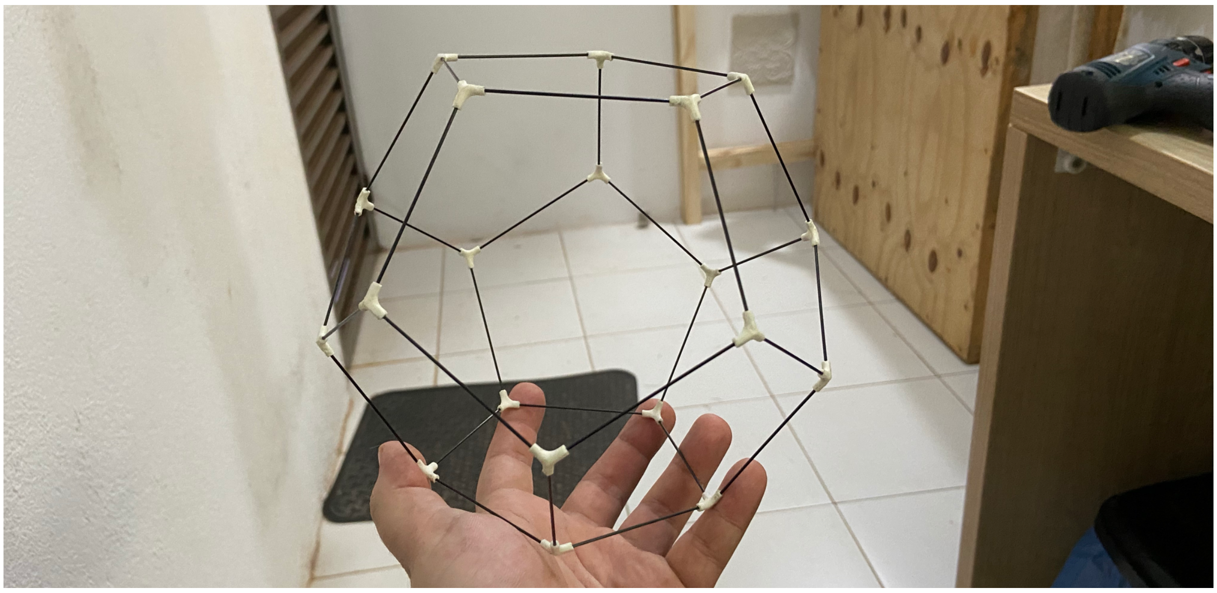


Figure 5 - Complete assembly of the dodecahedron without the drone.

We chose the dodecahedron shape because it would be very difficult to juggle with a cuboid one, and a spherical shape would be much harder to manufacture since the process to shape the carbon fiber is not trivial. We calculated that the edges needed to be 61mm long for the drone to fit entirely into the structure. A 3D model was developed to confirm and is shown in Figure 6.

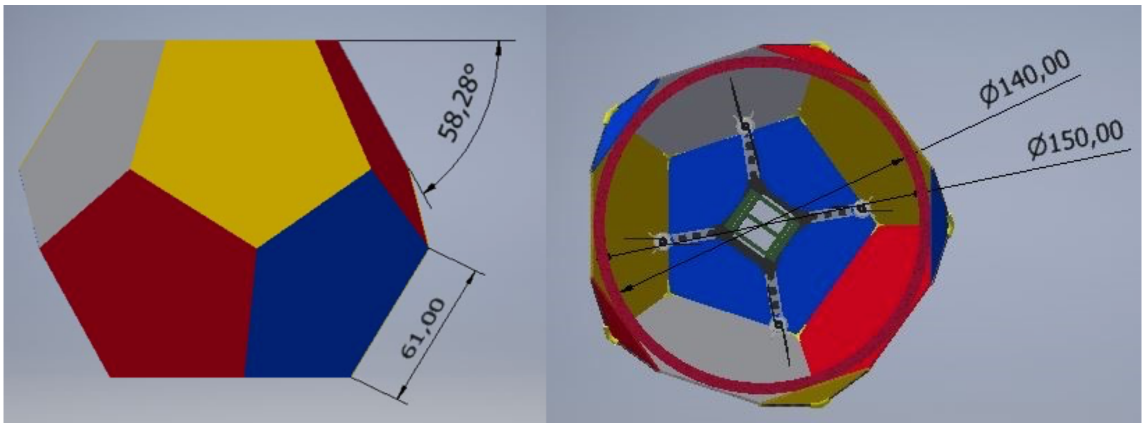


Figure 6 – The 3D model made to confirming that the calculated size of the structure would fit the drone inside.

From the 3D model we modeled connection pieces. Figure 7 shows the proposed geometry after a few iterations of manufacturing and evaluation of the real structure rigidity and weight.

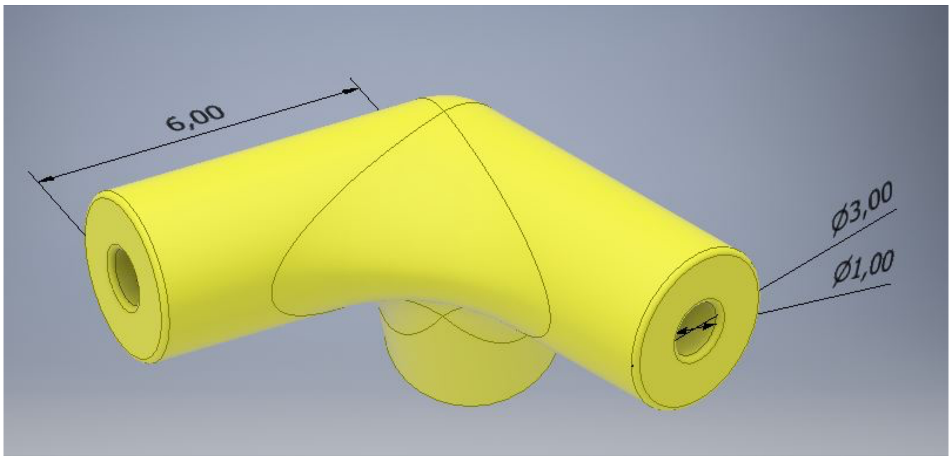


Figure 7 – The 3D model of the vertices connections to be printed, the dimensions are in millimeters.

A 3D model of the assembled dodecahedron can be seen Figure 8.

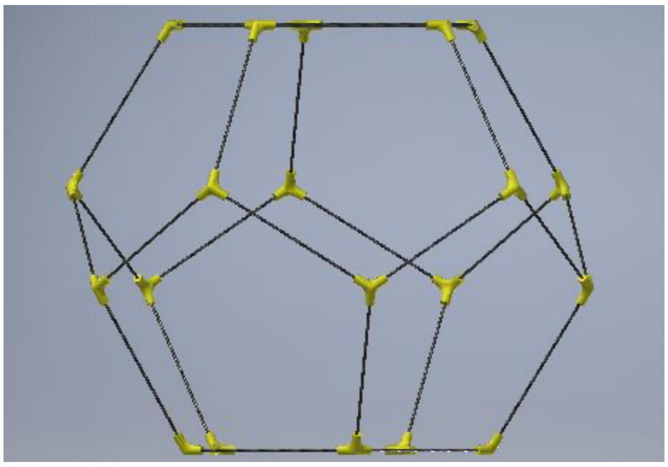


Figure 8 – The complete assembly 3D model of the dodecahedron without the drone.

To mount the Crazyflie on the structure we model the parts that attach the motors of the quadrotor to the carbon fiber rods. It can be seen in Figure 9.

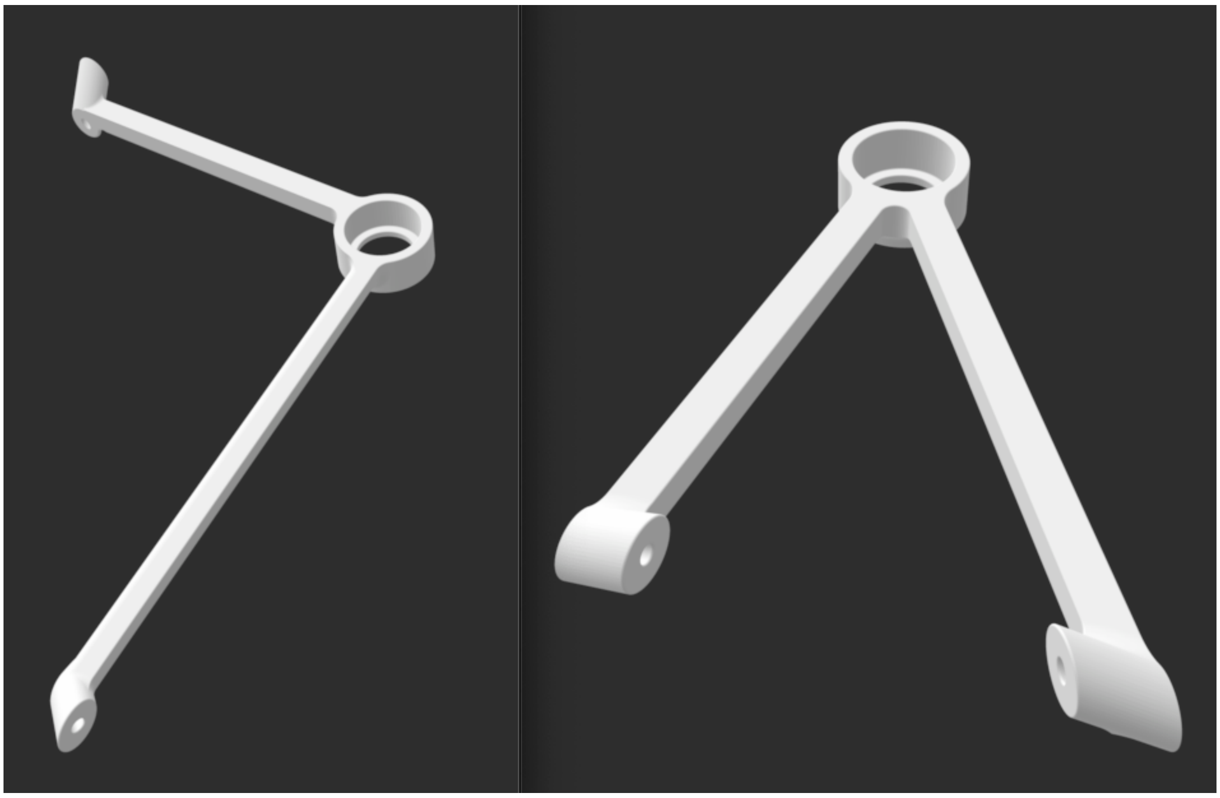


Figure 9 - 3D model of the parts for coupling the structure to the Crazyflie. How it docks to the drone and the dodecahedron can be seen in Figure 10.

We weighted and calculated the density of the carbon fiber rods to be 1.5279 g/cm^3, and the density of the filament we used is 1.06 g/ cm^3. Given this value and the geometry of the structure we used a tool of the 3D modeling software to calculate the predicted mass to be 7g and the inertial moment to be the one shown in Eq. 1.

$$I = \begin{bmatrix} 3.1956 & 0.0018 & -0.0088 \\ 0.0018 & 3.6768 & -0.0143 \\ -0.0088 & -0.0143 & 2.9556 \end{bmatrix} 10^{-5}\ [kg\ m^2]. \tag{1}$$

For the printed parts we used ABS filament, since it is stronger and lighter than other materials commonly used for 3D printing. It is worth noting that we used an Ender 3, a very popular and open-source 3D printer, and in order to achieve the necessary precision for the parts, we changed the standard 4mm nozzle for a 2mm one. The dodecahedron mounted to the Crazyflie in Figure 10.

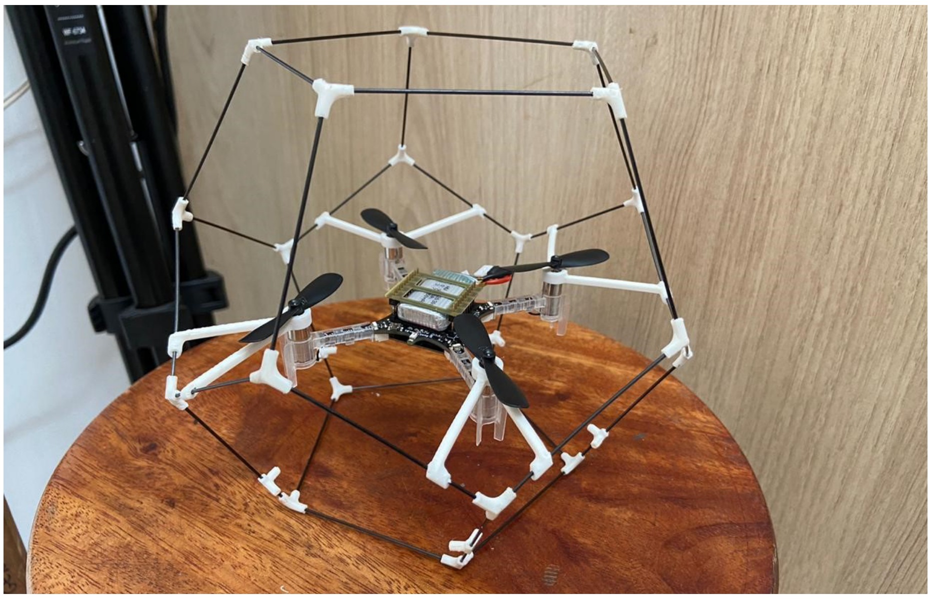


Figure 10 - Complete assembly of the dodecahedron structure with the drone docked.

The structure with the four coupling parts weighs 6.3g, which is 10% less than the calculated weight and only 16.7% of the whole system mass. Due to the ABS expansion during the printing process, all the holes had to be adjusted with a drill, which removed a lot of material from the final parts, which explains the difference in weight between the model and the real one. Unfortunately, we couldn't compare the real inertial moment with the calculated one since we don't have any measuring equipment.

Despite the structure mass being only 58% of the available payload, it is mainly distributed far from the center of mass, which makes the inertial moment increase considerably. When we compare the estimated inertial moment of the structure, shown in Equation (1), to the one of the Crazyflie [17], we can see that the moment of inertia of the drone with the structure is twice the one of the quadrotors alone. Therefore, it is very likely that the drone control loop has to be adjusted in order to fly with the new structure. Indeed, when we try fly without modifying the controller, the drone falls to one side and fails to takeoff. Therefore, we realized that the parameters of the proportional–integral–derivative (PID) controller were no longer enough for the drone takeoff and fly.

With this problem comes two possible solutions. We could choose to remake the controller using a more modern one, or we could simply adjust the parameters of the original controller. During this work, we always try to use the practicality and speed of development as the criterion of choice to reach the goal of juggling the quadrotor as soon as possible. Given that we don't have a model of the Crazyflie flying dynamics, or the components parameters to make our own model, remaking the controller would not be a fast solution. Therefore, we chose to recalibrate the control algorithm already implemented on the Crazyflie.

CHAPTER 3 – CONTROLLER ALGORITHMICS

A drone without a control algorithm is an unstable system, meaning that it will not stay in the air effortlessly, much like an untrained person in a rola-bola (Figure 11).

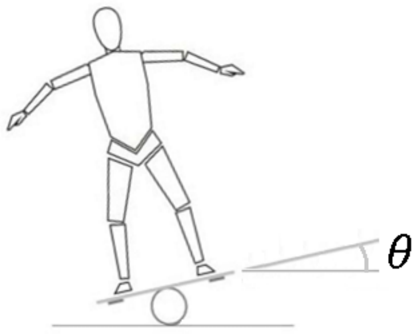


Figure 11 - Illustration of a rola-bola with an θ angle to be compensated.

Modified image from [18].

Hence, to continue with our analogy, we will consider the simplified blocks diagram of the rola-bola problem as shown in Figure 12. The muscle memory of a well-trained artist is the Controller, that allows him to stand on the rola-bola. His body on the rola-bola is the Plant. And the inner ear balance system is the sensor. How much the artist must move his muscle will depend on how much he is tilted, we call this error, e. The error is the difference between the perceived value and the desired value of the variable to be controlled $e = \theta_r - \theta$, hence, in the rola-bola example, the error would be the angle between the board and the floor, since the desired position would be zero degrees (parallel to the ground).

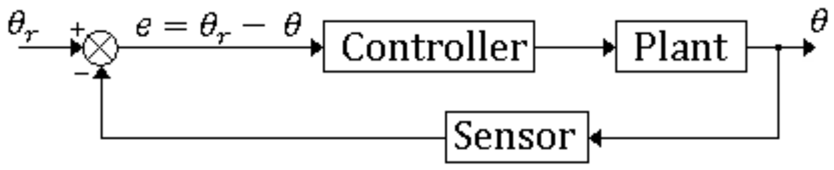


Figure 12 – Simplified blocks diagram of rola-bola

However, in the drone case, the organic muscles are replaced by electrical motors, the inner ear balance system is replaced by an electronic gyroscope sensor and the neural network is replaced by a control algorithm running in a microcontroller.

One of the most common control algorithms is the PID, Figure 13, in which, integrator is an integral operator and Derivative is a derivative operator. PID will multiplies the error by a constant usually called K_p, multiplies the error integral by another constant usually called K_i, multiplies the error derivative by a third constant, K_d, adds all these values and uses the result as the control value for the actuators, hence the name.

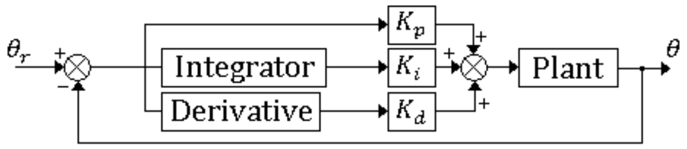


Figure 13 – PID blocks diagram

Just like an untrained person would fall from the rola-bola, a PID controller with wrong constants values will not be able to follows the reference signal. Then, when we say that the control needs to be calibrated or tunned, that means we need to find the correct values (correct enough) of the $\mathrm{K_p}$, K_i and K_d constants for the controller to work properly.

Another commonly used technique for controlling more complex systems is to concatenate several PID controllers, where one gives the reference value for the next one, in a struct called PID cascade. The Crazyflie control algorithm is a cascaded PID and can be seen in Figure 14.

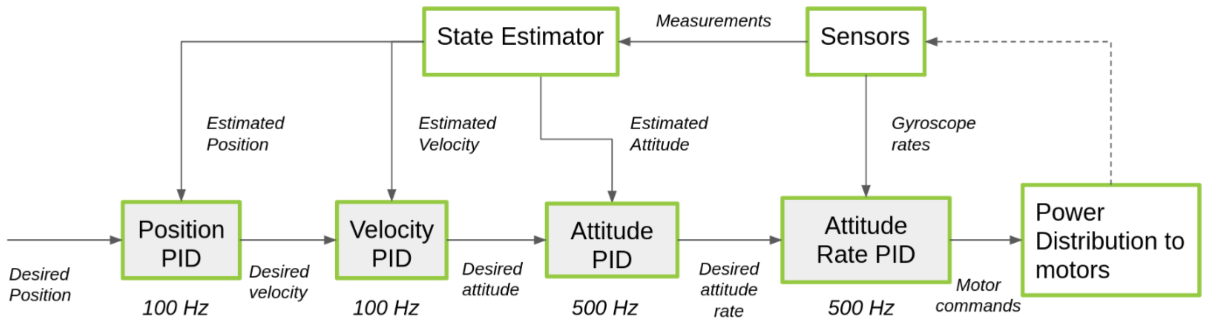


Figure 14 – Block diagram of a simplified version of the Crazyflie cascaded PID control structure [19].

In this cascaded PID the Position PID receives the desired and actual positions and calculate the velocity needed to arrive at the desired position as quickly as possible. Although, note that, even if the desired position is constant, the desired velocity will change over time, since the Position PID goal is to arrive at the desired position and stay there, and not zip through the position at a constant speed. Similarly, the Velocity PID receives that desired velocity and the current velocity to calculate a desired attitude (angle), since the vertical velocity of a drone is proportional to its angle. The attitude PID calculate the attitude rate (angular velocity) and finally the Attitude Rate PID sends the desired thrust to each motor. The figure simplifies the representation of a three-dimensional system, since the drone is able to move and rotate in all three directions each one of these blocks actually represents three PID controllers. Therefore, there are 12 PID controllers in total and 36 parameters that can be adjusted. In the next sections we will discuss a strategy to tunning such a complex system, but we still need to name those dimensions in order to understand further this control structure and we need to define our referential systems and how they are associated.

3.1 REFERENCE FRAMES

We have two coordinate frames related to explain. One is the local coordinate that is in the body of Crazyflie, the other is the global coordinate attributed to the room, this is illustrated in figure 14 in a simplified way. Hence, when two coordinate frames are related it means there is a successive sequence of rotations about the coordinate axes that will make their attitudes be the same. The angle of rotation about a coordinate axis is called a Euler Angle and a sequence of such rotations is often called a Euler Angle Sequence. The rotational sequence is an essential information for the Euler Angles, in Crazyflie firmware the sequence is XYZ. The sequence be XYZ means to turn the attitude of the first frame into the second one, first rotate angle about the X-axis (pitch) then a rotation angle on the new Y-axis (roll) and finally a rotation angle in the new Z-axis (yaw) [20].

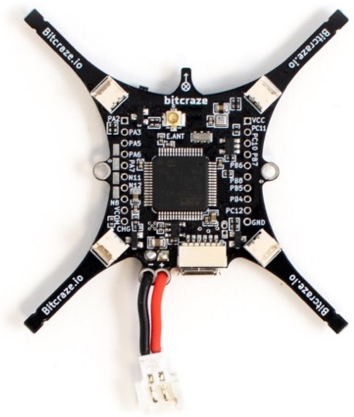


Figure 15 – Bottom of the Crazyflie main board, that also is the main structural component of que drone.

The body of the Crazyflie is a single Printed Circuit Board (PCB), as can be seen in Figure 15. The plane on which it is contained is defined as the xy-plane of the body reference frame. Since this quadrotor has a cross-configuration, the x and y axis are rotated 45 degrees in relation to the lines that connect the opposite-sided propellers [21]. The x-axis goes from back-to-front, where the back is the side that contains the large battery cable. The z axis direction is bottom-up placing the Crazyflie on the ground with its propellers facing upwards, in takeoff position. The y axis completes the cross product $\vec{x} \times \vec{y} = \vec{z}$, with its direction pointing from right to left, as illustrated in Figure 16. Furthermore, marks with this body frame orientation are printed in the main board as well as in all the extra boards that can be coupled to the Crazyflie. In Figure 15 one of this marks can be seen on the top, just above the name “bitcraze”, the arrow indicates the front of the drone, positive x direction, and the circle with the cross indicates that the z-axis is entering the PCB~~.~~

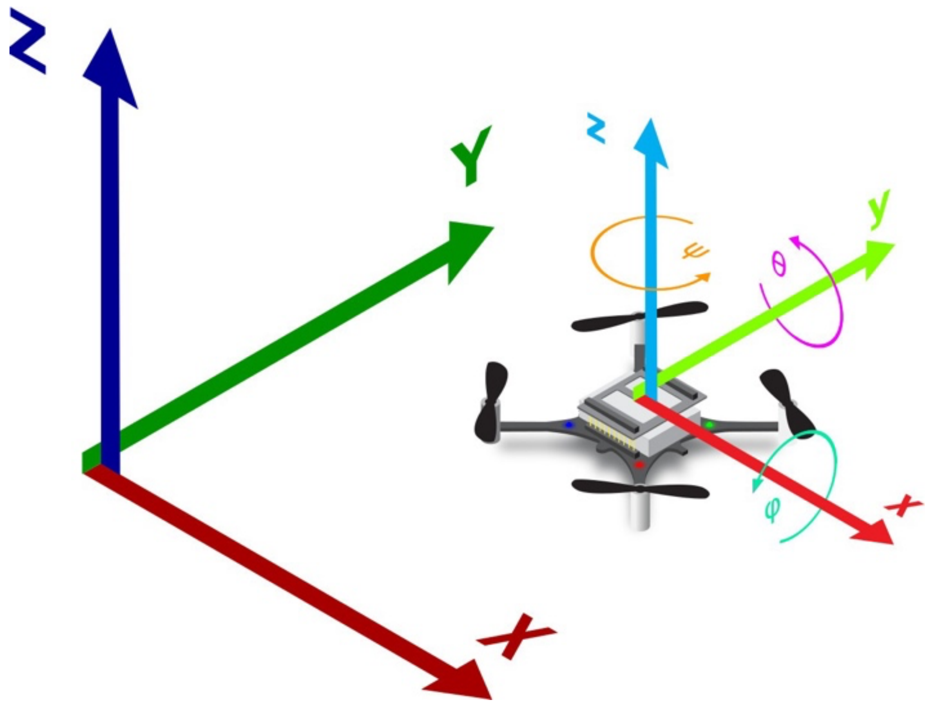


Figure 16 - Crazyflie body reference frame [22].

We will consider the room as our inertial reference system, since the room moves only with the rotation and translation of the earth, which is a negligible for the distance and durations of ours flights. That means the Newtons laws of motion will be valid in this reference frame [23]. Besides being fixed to the room, the xy-plane of the inertial reference frame will always be perpendicular to the Earth's gravitational field, that fixes two of the three orientations. The other orientation (around z), as well as the coordination system origin, will depend on the sensors that are coupled to the drone. If no additional sensor boards are used the z orientation and the origin position of the reference frame will be set to be equal to the body reference frame in the moment that the State Estimator is reset, which happens every time the drone is turned on and whenever we send the reset state estimator command by radio. State Estimator in our case is the extended Kalman filter showed in Figure 4. However, if the Flow deck sensor is coupled, the distance between the drone and the ground is measured using the laser distance sensor included in the deck. With this information and assuming that the floor of the room is leveled (perpendicular to the Earth's gravitational field) the xy-plane of the inertial reference frame is defined as the plane that contains the floor. And, finally, if the Lighthouse positioning deck is coupled to the Crazyflie and the base stations are set on the room, the position and orientation of the inertial reference frame will be the same as the body reference frame of the drone in the moment of the Lighthouse system setup, which only need to be done when the position of the bases change in relation to which other. This last setup is more convenient, since the inertial frame is preserved between power cycles, but is the only one that requires external hardware to the drone, what would mean not being able to use the equipment on a stage that was not pre-prepared. Thereby, the reference for all the measurements of position and velocity (angular and linear) mentioned on this, and the following sections is the inertial frame.

3.2 CONTROLLER FIRMWARE

To understand in more detail the control structure implemented in the Crazyflie, represented in a simplified way in the Figure 14, we reverse engineered the firmware code and represented how it works on a block diagram that can be seen in Figure 17. In this diagram each one of the PID(z) block represents different calls of the same function in the code, therefore all the controller blocks have the same implementation. The PID source code excerpt, Algorithm 1, show the PID implemented in Crazyflie, from it we calculate the transfer function that's represent it in the Z domain that is express in Eq. 2. The full open-source code can be found on the Bitcraze's GitHub page [24].

$$C_{PID} = K_p + K_d \frac{(1 - z^{-1})}{T} + K_i \frac{T}{(1 - z^{-1})} \,. \tag{2}$$

```
...
pid->outP = pid->kp * pid->error;
output += pid->outP;

float deriv = (pid->error - pid->prevError) / pid->dt;
...
pid->deriv = deriv;
...
pid->outD = pid->kd * pid->deriv;
output += pid->outD;

pid->integ += pid->error * pid->dt;
...
pid->outI = pid->ki * pid->integ;
output += pid->outI;
...
pid->prevError = pid->error;

return output;
...
```

Algorithm 1 - Excerpt of the Crazyflie firmware's PID code implementation [24].

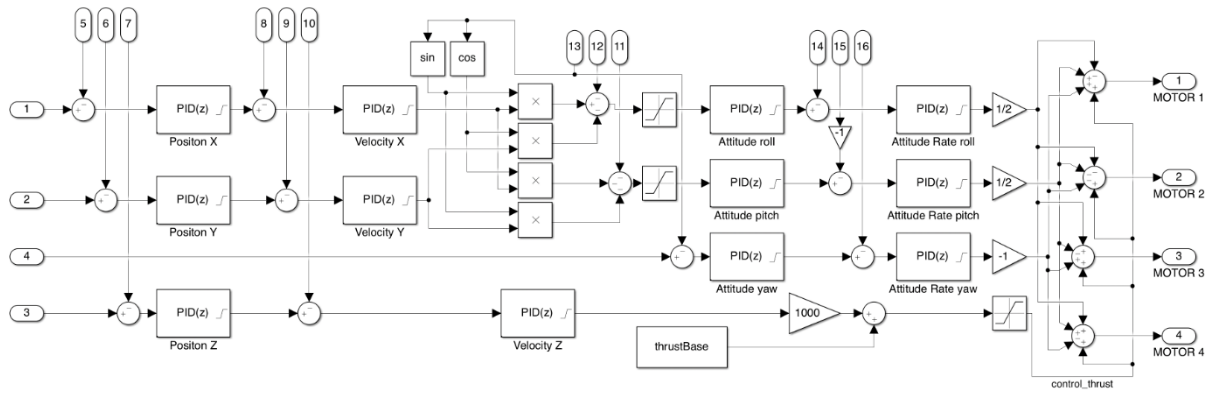


Figure 17 - Block diagram of the detailed operation of the firmware controller. This diagram is not available from the manufacture, it was made by us through a reverse engineering of the Crazyflie firmware.

3.3 TUNING METHODS

In the future of this project, we intend to update the hardware. Perhaps changing the rotors for more efficient ones and/or upgrading the cage. Actions that will require re-adjusting the controller. Thus, the tuning of the PID controller parameters will be a recurring task. Therefore, the tuning method of choice should be something practical that can be easily repeated for each new upgrade of the quadrotor, whence we discard methods like the Ziegler Nichols, that's require individual analysis each time we run it. With this requirement in mind, we analyzed some direct design methods based on input-output measurements without a plant model. A measurement of input-output contains expressive information about the plant and with the correct algorithm it can generate results as good as or even better than those obtained through models [25].

Some of those methods are the iterative feedback tunning (IFT) [26], the fictitious reference iterative tuning (FRIT) [25] and the virtual reference feedback tuning (VRFT) [27]. Among these methods, we will exclude the IFT, because it requires a new input-output measurement for each new iteration of the algorithm. The VRFT and FRIT methods require only one individual experiment in the plant. Despite those two last methods being based on similar ideas [28], the VRFT is suboptimal for restricted controller classes [27] and FRIT gives a better tuning parameter [29].

Initially, we chose the FRIT method and afterwards we changed to the extended fictitious reference iterative tuning (E-FRIT). The differences between those two methods will be explained in upcoming sections, for the time being is sufficient to know that both methods consider a time invariant, one-degree-of freedom, single-input single-output (SISO) closed loop system. However, considering the system represented in Figure 17 it is clear that the complete Crazyflie controller is a multiple-input multiple-output (MIMO) system, therefore we must restrict and isolate each PID what will be discussed later.

3.4 FRIT

To explain how FRIT works let's use as an example the generic control system shown in Figure 18. Denoting the plant model as $\mathrm{P}(s)$, the controller as $\mathrm{C}(s, \rho)$, ρ is the vector controller parameter, $r(s)$ is the reference signal, $\mathrm{u}(s, \rho)$ is the controller signal and $\mathrm{y}(s, \rho)$ is the system output signal. s is the complex variable used in Laplace transform, since it is not influencing the results, it will be omitted from now on in order to result in a clear notation.

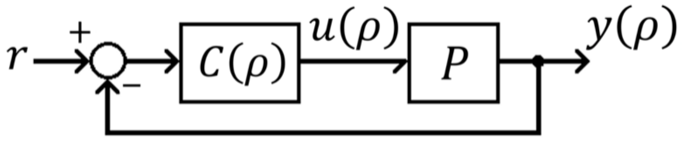


Figure 18 – Diagram block of a generic closed loop system.

The controller C may be any controller parametrized by ρ, in our case C is a PID controller and ρ is a vector defined by Eq. 3:

$$\rho = \begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix}. \tag{3}$$

The transfer function, $T(\rho)$, of the this closed loop system in given by

$$T(\rho) = \frac{P.C(\rho)}{1 + P.C(\rho)}, \tag{4}$$

its output $\mathrm{y}(\rho)$,

$$y(\rho) = T(\rho).r. \tag{5}$$

The output signal of the controller $\mathrm{C}(\rho)$, can be express by:

$$u(\rho) = T(\rho).\left(r - y(\rho)\right). \tag{6}$$

The objective of the method is to find a configuration that makes the closed loop system achieve the desired behavior. Therefore, it is necessary to define in advance this behavior through a transfer function that we will call T_d. The output of the T_d will be $y_d$with an input signal, r,

$$y_d = T_d.r. \tag{7}$$

Our objective is to find the parameters, ρ^*, that makes the output, $y(\rho^*)$, match with the reference model output, y_d, i.e., $y(\rho^*) = y_d$. Hence, if the optimal controller $C(\rho^*)$ really exists, we expect $T(\rho^*)$ to be identical to the desired model T_d as shown by Eq. 10. Ergo, if we know the plant model it is possible to find ρ^* by minimizing the performance index, $J(\rho)$:

$$\rho^* = arg\, min_\rho\ J(\rho), \tag{8}$$

in which, $\arg\min_\rho$ is the arguments of the minimum, which gives the value of ρ at which $J(\rho)$ attains the minimum value. The minimized function, $J(\rho)$, is defined as:

$$J(\rho) = \int_0^T (y(\rho) - y_d)^2\, dt\,. \tag{9}$$

Therefore, if $J(\rho^*) = 0$, we expect that $y_d = y(\rho^*)$, hence we conclude

$$T_d = \frac{P.C(\rho^*)}{1 + P.C(\rho^*)}. \tag{10}$$

We can find ρ^* with Eq. 8 through a nonlinear approach method, such as the Gauss-Newton method [28] [29]. A standard approach would be one of the following two options:

1) If we have the model plant, P, we could use the iterative optimization method by simulating the system. This approach is valid even if the system is nonlinear. But we do not have the P and it is not in the scope of this work modelling P;
2) Same as before, but instead of simulating the response in each iteration, we run a test in the real system. This way we avoid the need to know the plant model. But this would result in a new collection of data at each iteration to estimate ρ^*.

Neither approach is attractive for us to use in Eq. 8. The FRIT method can overcome the need for multiple data collections as it only needs a single data, and also don't need to know P. Let's understand how this is done.

First, we will denote the value of the parameter ρ used during the data collection as ρ_0, note that the only requirement for choosing ρ_0 is $T(\rho_0)$ must be a stable system, even if it is not well tuned. The collected output signal and control signal will be called y_0 and u_0 and will be denoted, respectively, by:

$$y_0 = T(\rho_0).r_0, \tag{11}$$

$$u_0 = C(\rho_0).(r_0 - y_0), \tag{12}$$

in which, r_0 is the input signal used, that can be arbitrary.

Let's start by assuming that in fact there is an optimal controller, $C(\rho^*)$ like showed in Eq. 10. The FRIT method uses a fictitious reference, r^*, input to overcome the need for multiple experiments. The r^* is defined as the reference signal that when applied to the closed loop system parameterized by ρ^*, $T(\rho^*)$, will produce the output system equal to y_0, i.e.,

$$y_0 = T(\rho_0).r_0 \;= T_d.r^* = T(\rho^*).r^*, \tag{13}$$

and since P has not been changed and output remains y_0 the control signal must be equal to u_0 [27], therefore

$$u_0 = C(\rho^*).(r^* - y_0). \tag{14}$$

From Eq. 14 we can isolate r^* and express it as:

$$r^* = \; C(\rho^*)^{-1}.u_0 + y_0. \tag{15}$$

Replacing Eq. 15 in $\mathrm{y}_0 = \mathrm{T}(\rho^*).\mathrm{r}^*$ from Eq. 13, we have

$$y_0 = \; T_d.C(\rho^*)^{-1}.u_0 + T_d.y_0. \tag{16}$$

Last's pause for a second to understand the meaning of all those expressions, get a cup of coffee if you will. The parameter ρ^* is what we want to find, it is the controller parameters that will make the close-loop system behave as desired, this is the idea expressed in Eq. 10. As can be seen in Eq. 3, in our particular case finding the ρ^* means to find the correct values of the K_p, K_i, and K_d constants for the controller to give the appropriate response, which is our goal after all. To avoid making a new experiment every time we change the controller parameters, we work with this clever idea of fictitious reference r^*. From Eq. 15 it is simple to see that if we have the r^* we could find the ρ^* with only one-shot experiment. Thus, now the problem become to find r^*. For this we could apply a similar no linear approach as before, but as we see in the rest of this section, we now have all the variables need for the calculation in each iteration.

FRIT identifies the optimal controller parameter, ρ^*, based on Eq. 16. Then, in order to use an iterative optimization method to find the ρ^*, we need to define some variables, $\hat{r}_i$ and $\hat{\mathrm{y}}_i$. $\hat{r}_i$ is the *i-th* candidate of the optimization algorithm to be r^*,

$$\hat{r}_i = C(\rho_i)^{-1}.u(\rho_0) + y(\rho_0), \tag{17}$$

and $\hat{\mathrm{y}}_i$ is the output of T_d using $\hat{r}_i$ as input in the *i-th* round of the optimization algorithm, i. e.,

$$\hat{\mathrm{y}}_i = T_d.\hat{r}_i, \rightarrow \hat{\mathrm{y}}_i = T_d.C(\rho_i)^{-1}.u_0 + T_d.y_0. \tag{18}$$

Remember that we want $\hat{\mathrm{y}}_i$ to be equal to y_0.

Since we can calculate $C(\rho_i)^{-1}$ and we have T_d, u_0 and y_0, then, we can find ρ^* minimizing the performance index introduced by FRIT, $J_F(\rho)$. We define the error $e(\rho_i)$ as:

$$e(\rho_i) \coloneqq y_0 - \; \hat{\mathrm{y}}_i, \tag{19}$$

then, we define $J_F(\rho)$ as

$$J_F(\rho) = \int_0^T e(\rho)^2 \, dt. \tag{20}$$

Now we can find ρ^* using Eq. 21:

$$\rho^* = arg\, min_\rho \, J_F(\rho)\,. \tag{21}$$

To show that we can use Eq. 21 instead of Eq. 8 we will assume that Eq. 10 is satisfied. Using Eq. 10, Eq. 7, Eq. 5 and Eq. 4 we can rewrite Eq. 8 as:

$$J(p) = \int_0^T (y(\rho) - y_r)^2 \, dt = \int_0^T \left(\frac{C(\rho^*) - C(\rho)}{1 + P\, C(\rho^*)} \cdot \frac{P}{1 + P\, C(\rho)} r \right)^2 dt\,, \tag{22}$$

and, using Eq. 10, Eq. 13 and Eq. 18 we can rewrite Eq. 20 as:

$$J_F(\rho) = \int_0^T e(\rho)^2 dt \; = \int_0^T \left(y_0 - \hat{y}(\rho)\right)^2 dt = \int_0^T \left(\frac{C(\rho^*) - C(\rho)}{1 + P\, C(\rho^*)} \cdot \frac{1}{C(\rho)} y_0 \right)^2 dt\,. \tag{23}$$

We can see that $C(\rho^*)$ in $J(\rho)$ is equal to $J_F(\rho)$, thus the controller $C(\rho^*)$ can be derived from the minimization of $J_F(\rho)$ through an iterative nonlinear optimization approach instead of minimizing $J(\rho)$ [29]. And even if there is no $C(\rho^*)$ to satisfy Eq. 16, $J(\rho)$ could be sufficiently approximated to $J_F(\rho)$ using a pre-filter in y_0 and u_0 [30].

3.5 E-FRIT

As mentioned earlier the FRIT approach requires that the reference model T_d must be determined beforehand. However, the FRIT will have an undesirable performance if we choose a reference model faster than it is possible. To avoid this problem, we can add a delay parameter L to the reference model T_d, therefore, the new reference model, T_L, becomes [29]:

$$T_L(s, L) = T_d(s) \cdot e^{-Ls}. \tag{24}$$

The evaluation function proposed on Eq. 21 is still the same. However, choosing an appropriate value of L can become a complex task without any knowledge of the plant. Thus, in order to determine the reference model properly, it is useful to provide a L parameters that can be optimize with the control parameters, then ρ [31] used in Eq. 3 becomes:

$$\rho_C = \begin{bmatrix} K_p \\ K_i \\ K_d \\ L \end{bmatrix}, \tag{25}$$

and $\hat{y}_i$ becomes

$$\hat{y}_i = T_L \cdot C(\rho_i)^{-1} \cdot u_0 + T_L \cdot y_0. \tag{26}$$

Therefore, we will use the E-FRIT instead FRIT.

Note that all this math could also be done in the discrete domain. Thus, considering that our data will be given in vector forms, the E-FRIT performance index, $J_{F2}(\rho)$, used will be:

$$J_{F2}(\rho) = \frac{1}{N}\sum_{k=1}^{N}\{(y_0[k] - \hat{y}_i[k])^2 + \lambda\Delta\tilde{u}(\rho_i, [k])\}, \tag{27}$$

in which, λ is a weighting coefficient and it is recommended to be equal to 1 [31] and $\Delta\tilde{u}(\rho_i, k)$, is a term added to penalize sudden variations in the controller signal, ũ, between two instants, defined by.

$$\Delta\tilde{u}(\rho_i, k) = \tilde{u}(\rho_i, k) - \tilde{u}(\rho_i, k-1), \tag{28}$$

considering we are working in the discrete time, $\tilde{u}(\rho_i, k)$ can be calculate as:

$$\tilde{u}(\rho_i, k) = C(\rho_i, Z).(\hat{r}_i[k] - \hat{y}_i[k]). \tag{29}$$

Therefore, with the E-FRIT method we can change the values of ρ_C, calculate $\hat{r}_i$ and consequently T_L, $\tilde{u}(\rho_i, t)$ and $\hat{y}_i$ hence we will have all the parameters of Eq. 27, thus, we can use a nonlinear optimization method and we can find ρ^* using the Eq. 27.

3.6 E-FRIT IMPLEMENTATION

To execute the E-FRIT method we use scripts in the MATLAB® software [32]. In this work we use version R2020b update 5 with a student license running on a MacBook Air (M1, 2020).

Our goal is not to make the low-level implantation of e-frit in the MATLAB® language, hence we will stick to a general explanation of how the implementation problem was addressed and give an in-depth look at just two functions that we consider most important, `lsim` and `fmincon`. We are inspired by the code available at [33], ergo, we wrote two scripts for MATLAB® to be able to use the e-frit method. The first one is `perfomance_index.m` that we write as a function to MATLAB®, and it will take the vector ρ as an argument and execute all the necessary equations to return the value of $J_{F2}(\rho)$. The second script will be `e_frit.m` (ANNEX I) which will pull the input-output data collected from the experiment, set the initial variables and call the iterative optimization function, `fmincon`, that will use the function `perfomance_index.m` as a cost function for optimization.

The two most relevant functions in the implementation, as already mentioned, are the *lsim* function and the `fmincon` function. The "`y = lsim(sys, rs)` *function returns the system response* `y`*, sampled at the same times t as the input. For single-output systems,* `y` *is a vector of the same length as t. This syntax does not generate a plot"* [34]. In the syntax used, the input arguments are *`sys`* and `r`, where sys is a dynamic system model, `rs` is the input signal for simulation, specified as a vector, and `t` is omitted, hence its value is equal to the period used in sys discretization. y is the simulated response data, returned as an array.

The algorithm is based on recursive digital filter [34]. Hence, considering the input argument sys defined by

$$sys(Z^{-1}) = \frac{a_1 + a_2 Z^{-1} + \cdots + a_n Z^{-n}}{1 + b_1 Z^{-1} + \cdots + b_m Z^{-m}} \tag{30}$$

then the *k-th* element of output y becomes

$$y[k] = sys(Z^{-1}) rs[k], \tag{31}$$

replacing Eq. 30 in Eq. 31, we have

$$y[k] = \frac{a_1 + a_2 Z^{-1} + \cdots + a_n Z^{-n}}{1 + b_1 Z^{-1} + \cdots + b_m Z^{-m}} rs[k] \tag{32}$$

hence, multiplying both sides by the denominator of $\mathrm{sys}(\mathrm{Z}^{-1})$, we have

$$y[k] \cdot (1 + b_1 Z^{-1} + \cdots + b_m Z^{-m}) = (a_0 + a_1 Z^{-1} + \cdots + a_n Z^{-n}) rs[k], \tag{33}$$

then, applying distributive property of multiplication, the Eq. 33 becomes

$$y[k] + b_1 \cdot y[k] \cdot Z^{-1} + \cdots + b_m \cdot y[k] \cdot Z^{-m} = rs[k] \cdot a_0 + a_1 \cdot rs[k] \cdot Z^{-1} + \cdots + a_n \cdot rs[k] \cdot Z^{-n}, \tag{34}$$

hence, isolating $y[k]$, we have the equation:

$$y[k] = \ u[k] \cdot a_0 + a_1 \cdot rs[k] \cdot Z^{-1} + \cdots + a_n \cdot rs[k] \cdot Z^{-n} - b_1 \cdot y[k] \cdot Z^{-1} - \cdots - b_m \cdot y[k] \cdot Z^{-m}. \tag{35}$$

We set all initial conditions to zero, in Eq. 35.

Another important function used is `fmincon`, that find the minimum of constrained nonlinear multivariable function [35]. Its syntax is:

```
[x,fval,exitflag,output]=fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,op
tions).
```

Input arguments are:

- `fun`: Function to minimize, in our case its `performance_index.m`, i.e., `fmincon` will find the input argument of `performance_index.m`, ρ, that will return the minimal value.
- `x0`: Initial point, $x_0 = [\rho_0 \ L_0]$, ρ_0 used to collect eq. (11) and (12) and L_0 is our initial guess.
- `A, b, Aeq, beq` are linear inequality constrains and for our work they are all null.
- `lb` it is a lower bound and `ub` is an upper bound, i.e., this will limit the maximum and minimum value that ρ can be.

- `nonlcon` is nonlinear constraints, it is specified as a function name, `nonlcon.m`, which receives as input argument a vector or array x and return ceq and c, where, ceq is an array of nonlinear inequality constraints at, $ceq(x) \leq 0$, and c is the array of nonlinear equality constraints, $c(x) = 0$ [35]. `fmincon` attempts to satisfy both ceq and c, in our work we have no need to define such limitations, hence our implementation of `nonlcon.m` returns ceq and c as null vectors.
- Options is the optimization options parameters. We use the option `Algorithm` set on `interior-point` [36] [37].

With the understanding of the FRIT method and its implementation in the ANNEX I scripts, we will do the experiments and the calibration of the controller in the next chapter.

CHAPTER 4 – PERFORMANCE EVALUATION

As mentioned in the previous chapter, the first task is to choose the order in which each one of the 12 SISO controllers in the diagram shown in Figure 17 will be calibrated, and if in fact all 12 will need to be tuned. As the only significant characteristic of the Crazyflie altered by the addition of the structure, besides the shape, was the inertial moment, we foresee that the six most external controllers (left of the diagram) will not need tuning, since they control linear velocity and positions and those are not influenced by the inertial moment, only by the mass. Furthermore, we will not attempt to calibrate the yaw and yaw rate controllers at first, since the yaw orientation isn't critical to the flight stability and our most immediate goal is to make the drone do a stable flight. Therefore, we start with the following four controllers: Attitude pitch, attitude rate pitch, attitude roll and attitude rate roll.

Moreover, due to the symmetry of the structure and of the Crazyflie, we will only use data for one axis to run the FRIT algorithm and we will use the result parameters in both axes. It is highly likely that this assumption was also made in the calibration of the original firmware controller since the controller constants of the pitch and row controllers are the same in the firmware source code. Since there is no objective reason to choose one over the other between those two axes, we chose to run the experiments on the pitch axis because it was easier to attach the quadrotor in this orientation doe to the pins position in the PCB board. Finally, since the dynamics of the attitude rate controller affects the dynamics seen by the attitude controller, the inner controllers (attitude rate ones) need to be tunned first.

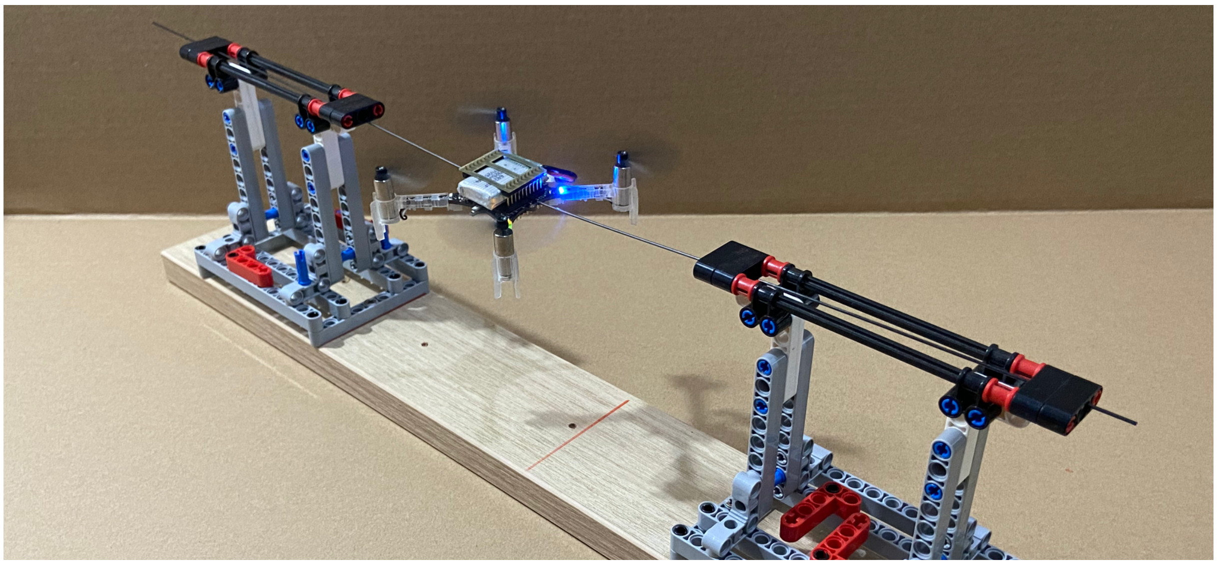


Figure 19 – Crazyflie, without the dodecahedron, in the test stand that's restrain all the degrees of freedom except the pitch one.

We assembled a test stand to constrain the drone movements other than the pitch, which can be seen in Figure 19. Beyond that we also disabled the other controllers during the tests to ensure that only the pitch PID controllers was contributing to the control signal u_0.

Although we did not use step signal as an input to use the FRIT and E-FRIT algorithms, we run some tests with step inputs to get a more intuitive sense of the current system response. The response of the rate pitch controller of the drone without the dodecahedron structure is shown in Figure 20 (A), and with the structure in Figure 20 (B). The rise time with the structure is equal to 0.057 seconds and the response without the structure is equal to 0.110 seconds, practically twice as slow, that is expected since the estimated inertial moment of one is twice of the other. The overshooting has also been attenuated, from 70,5% with structure to 60,9% without structure. Since changes in system response are caused by increased inertia, we have concluded that there is no need to tunning the pitch rate PID since the only possible solution in this case would be to increase the motor thrust capacity

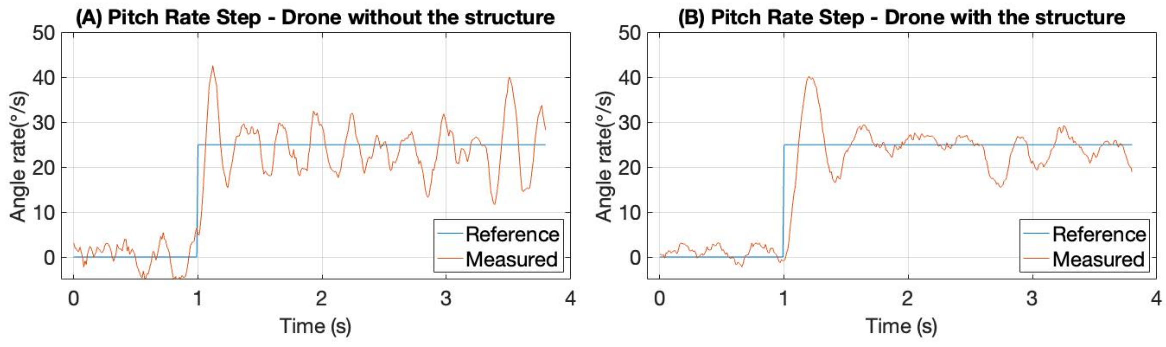


Figure 20 – Response to a step input from the attitude pitch rate PID controller of the quadrotor in the constrained test stand without the dodecahedron struct (in (A)) and with the structure (in (B)), in both cases with the original PID parameters.

Moving on to the next controller in the PID cascade, Figure 14, the angle one, we also ran some steps experiments to compere the system response with and without the dodecahedron structure, the data are shown in Figure 21 (B) and (A), respectively. The response signal of the Figure 21 (A) has an overshooting of 20.5%, a rise time of 0.173 seconds and a settling time of 0.95 seconds with tolerance set in 15%. And the response signal of the Figure 21 (B) has a overshooting of 31.57%, a rise time of 0.195 seconds and no settling time with tolerance set in 15%. In this case we can see that the two responses are essentially different, the one without the structure (Figure 21 (A)) is underdamped and eventually settle on the reference, meanwhile the response with the structure is a undamped one (Figure 21 (B)), thus will oscillate indefinitely, which is a very compelling reason for the drone to refused to fly.

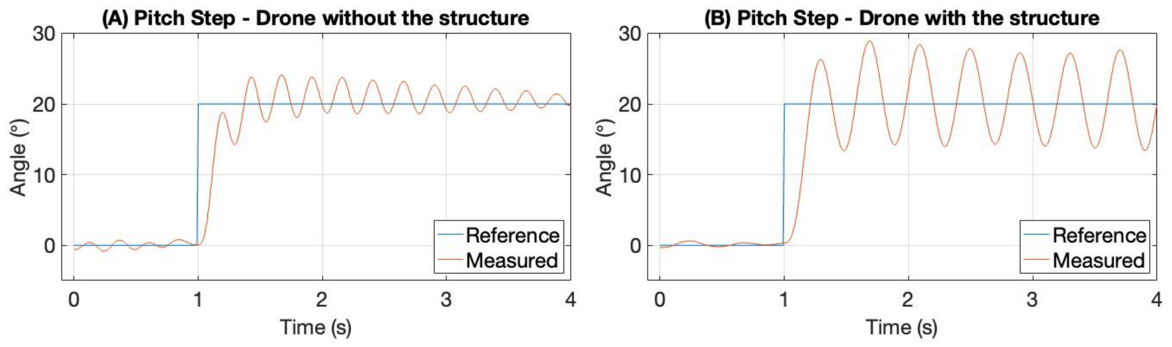


Figure 21 – Response to a step input from the attitude pitch PID controller of the quadrotor in the test stand without the dodecahedron structure (in (A)) and with the structure (in (B)), in both cases with the original PID parameters.

Then we pursue to apply the E-FRIT method to better tune the attitude pitch PID. Inspired by previous results we decided to use a pulse sequence with exponential decay as our input reference signal [28]. The exact signal that we used is the r_0 shown in Figure 22. In this figure we can also see response to this input of the original system, the Crazyflie without the dodecahedron structure with the original PID parameters $\rho_{original}$, in which:

$$\rho_{original} = \begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 0 \end{bmatrix}. \tag{36}$$

The high frequency component in the response of the original system, that can be seen in Figure 21 and in Figure 22, is not present when the drone is freely flying, as can be seen in Figure 28 (B) on the next chapter, therefore, this behavior must be induced by the test stand. Considering that the addition of the dodecahedron structure does not change the effect induced by the test stand, if we desire that the final version of the quadrotor controller have a similar behavior to the original one outside the test stand, then they also must have similar behavior in the test stand. Therefore, we must use a desired transfer function T_d, that has a similar response to the one shown in Figure 22.

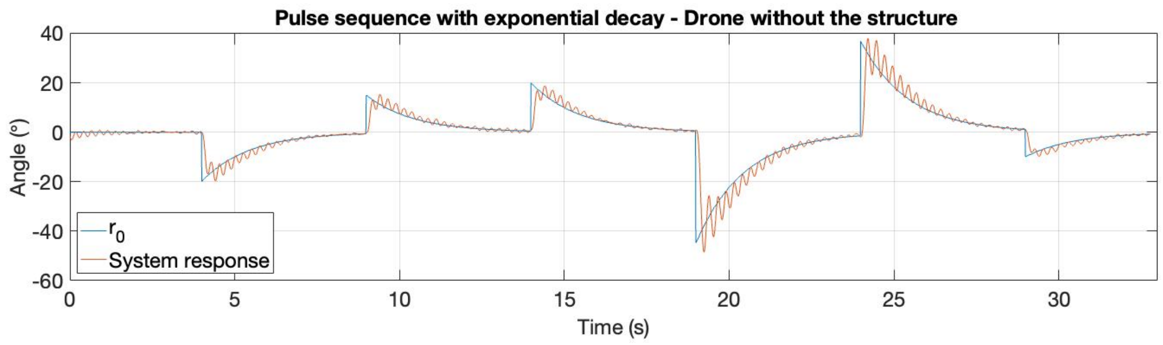


Figure 22- Response of the drone without the structure, System response, on the test stand with input, r_0, on the pitch attitude PID, with original factory control parameters. Contribution of the roll and yaw rate attitude PID have been disabled.

To generate a desired transfer function T_d that would have de desired effect we use the `tfest`, function of MATLAB® [38], using the data shown in Figure 22 as input. The script can be seen in Annex II and the result was the following:

$$T_d = \frac{8.5637\ (s+18.79)(s+3.13)(s+0.1996)(s^2-279.3\ s+7.456\cdot 10^4)}{(s+58.44)(s+14)(s+3.53)(s+0.2035)(s^2-35.81\ s+1.293\cdot 10^4)}. \tag{37}$$

To confirm that the $\mathrm{T_d}$ has a behavior similar to the intended one we ran a simulation with the same input r_0, that can be seen in Figure 23. One can ask why we do not use the results shown in equation (37) to identify the system model and use a traditional controller design, since the controller transfer function is known. Besides the fact that the T_d has not the same exact response to the one of the real system, which can be seen by comparing the Figure 22 and Figure 23, we have no way to separate the behavior caused by the test stand from our results.

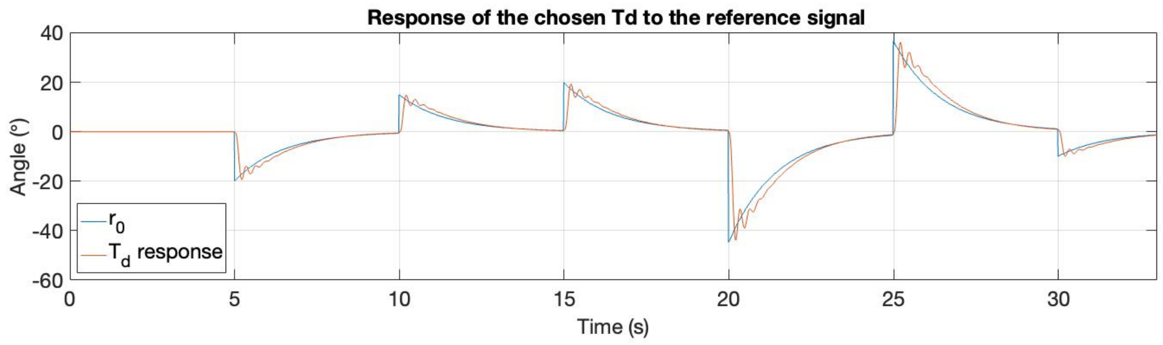


Figure 23- Simulated response of the desired transfer function, T_d response, in pitch axis.

To collect the initial data required to run the E-FRIT algorithm we have to chose K_p and K_i values lower than the original ones, because when we run the test with the original values the system proves to be instable as can be seen in Figure 24, and as discussed in previous chapters the E-FRIT and FRIT algorithm require a stable one.

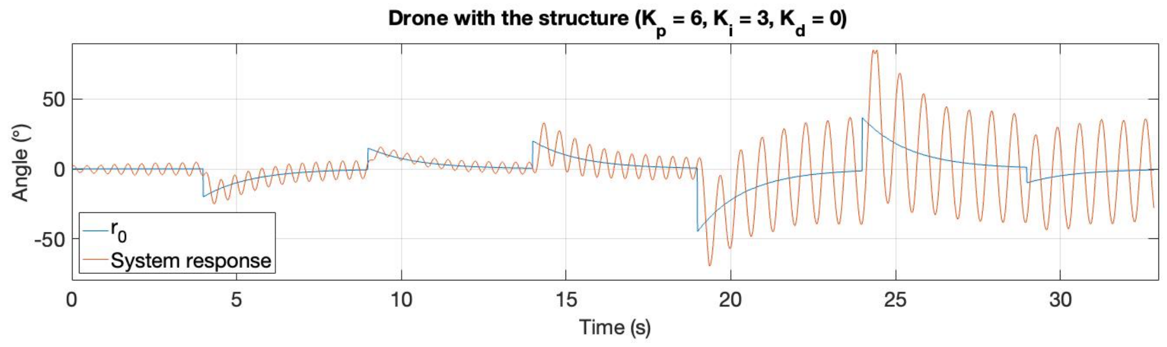


Figure 24 - Response of the drone with the structure, System response, on the test stand with input, r0, on the pitch attitude PID, with original factory control parameters, $\rho_{original}$. Contribution of the roll and yaw rate attitude PID have been disabled.

We also decide to change the value of the K_d, that was zero previously, turning the controller into a PI. We suspect that it could cause the algorithm to be trapped in a local minimum. Using the new values shown on Eq. 36, we collect que data to be used in the algorithm, the data is shown on Figure 25,

$$\rho_{0E} = \begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 1.7344 \\ 1.445 \\ 0.001 \end{bmatrix}, \tag{38}$$

Note that as we defined in Eq. 25, we also need to define L in addition to K_p K_i and K_d in $\rho_{original}$. In all test runs, we will use L equal to 0.01 seconds, which was chosen arbitrarily.

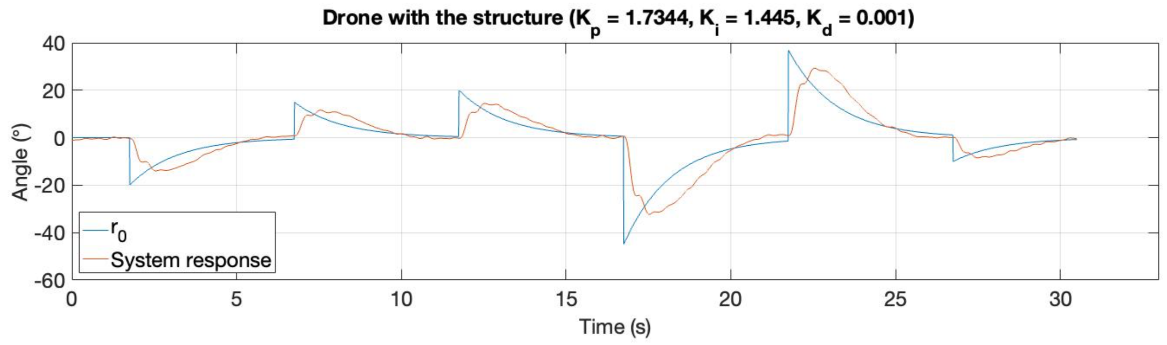


Figure 25 - Response of the drone with the structure, System response, on the test stand with input, r0, on the pitch attitude PID, with ρ_{0E} control parameters. Contribution of the roll and yaw rate attitude PID have been disabled.

We run the algorithm available, on the ANNEX I, it generated the estimated optimal parameters $\hat{\rho}^*$, shown on Equation (39).

$$\hat{\rho}^* = \begin{bmatrix} K_p \\ K_i \\ K_d \\ L \end{bmatrix} = \begin{bmatrix} 4.2039 \\ 4.7822 \\ 0 \\ 0.0094 \end{bmatrix} \tag{39}$$

Our interest is in the values that we will introduce in the PID in the Crazyflie firmware, i.e., K_p K_i and K_d. Hence, note that, even with we force a non-zero start point for the K_d it converges to zero again. Which suggest that the optimal controller is indeed a PI.

We run the experiment one more time with the generated parameter for confirmation, and the result is shown on Figure 26. As the answer obtained was very similar to the desired one, we preceded to the flight tests.

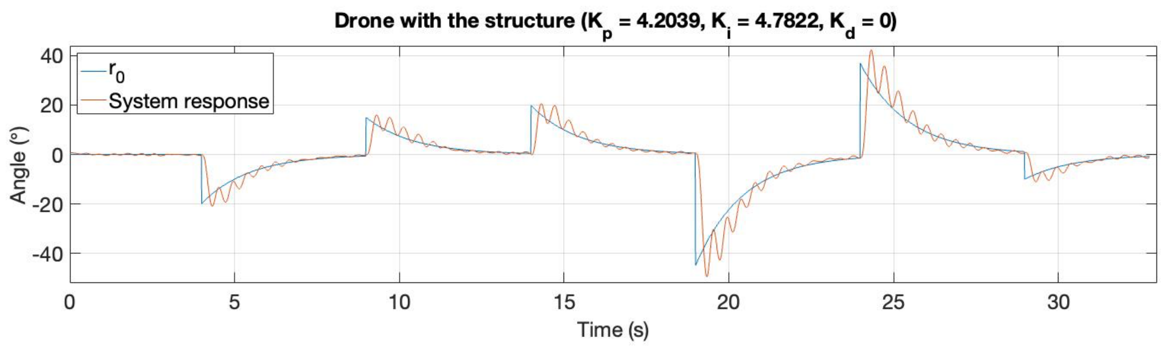


Figure 26 - Response of the drone with the structure, System response, on the test stand with input, r_0, on the pitch attitude PID, with $\hat{\rho}^*$ control parameters. Contribution of the roll and yaw rate attitude PID have been disabled.

As can be seen in Figure 27 the drone is able to fly in the structure using the new parameters $\hat{\rho}^*$ in the pitch and roll attitude PID controllers.

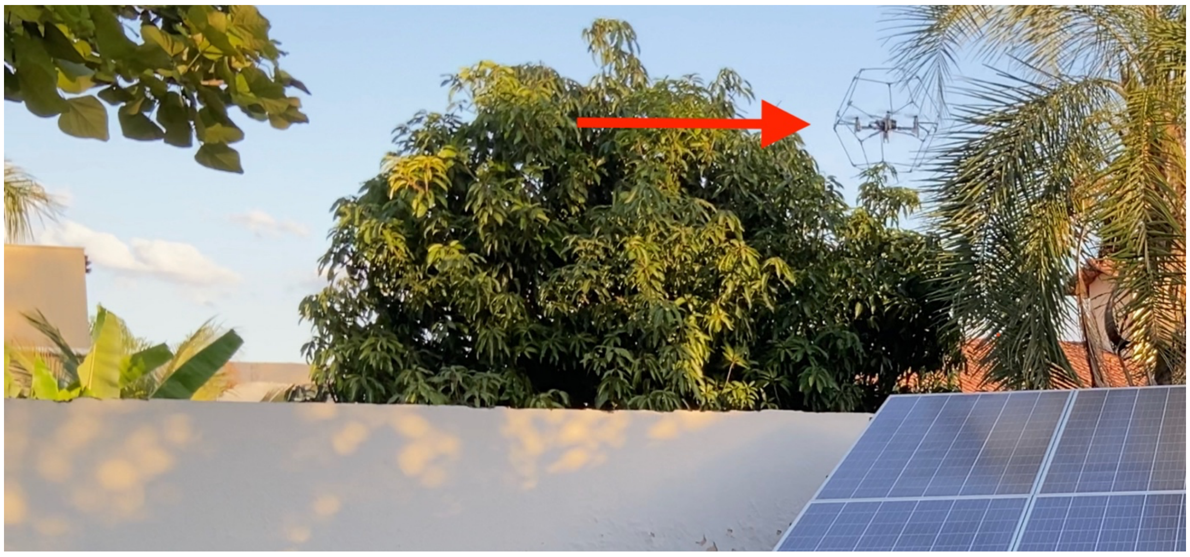


Figure 27 - Crazyflie in about 3 meters height, outdoor flight, with structure and using $\hat{\rho}^*$ as controller parameters for the pitch and row attitude PIDs.

However, because we do not yet regulate the yaw controllers, it was interfering with the drone's ability to fly as it saturates two of the engine's signals. Therefore, in all final flight tests we disabled the yaw controller, including in the flights with the drone without the structure that we made for comparison. Although the Crazyflie flights were stable, we basically have no steering since the drone flew spinning in the z axis due to the missing yaw controller. Because of that, we made the final flights outdoors and short-lived to prevent the drone to crash in a tree or a wall.

Figure 28 shows the results of this work in form of reference and system responses of the attitude pitch PID controller during outdoors flights and flights attempts. In Figure 28 (A) we can see that even with the yaw controller disabled the drone is not able to fly in the dodecahedron structure with the original PID parameters $\rho_{original}$ (Equation (34)), since in the presence of the smallest stimulus at the input of the pitch controller the system responds with an increasing oscillation that causes the drone to fall in less than a fifth of a second. Then in Figure 28 (C) we can see that with the new parameters $\hat{\rho}^{*}$ (equation (37)), the Crazyflie not only is able to fly in the structure, but also the PID pitch controller response follows closely the reference, in a very similar way to the response of the original controller in flight without the structure, as can be seen in Figure 28 (B).

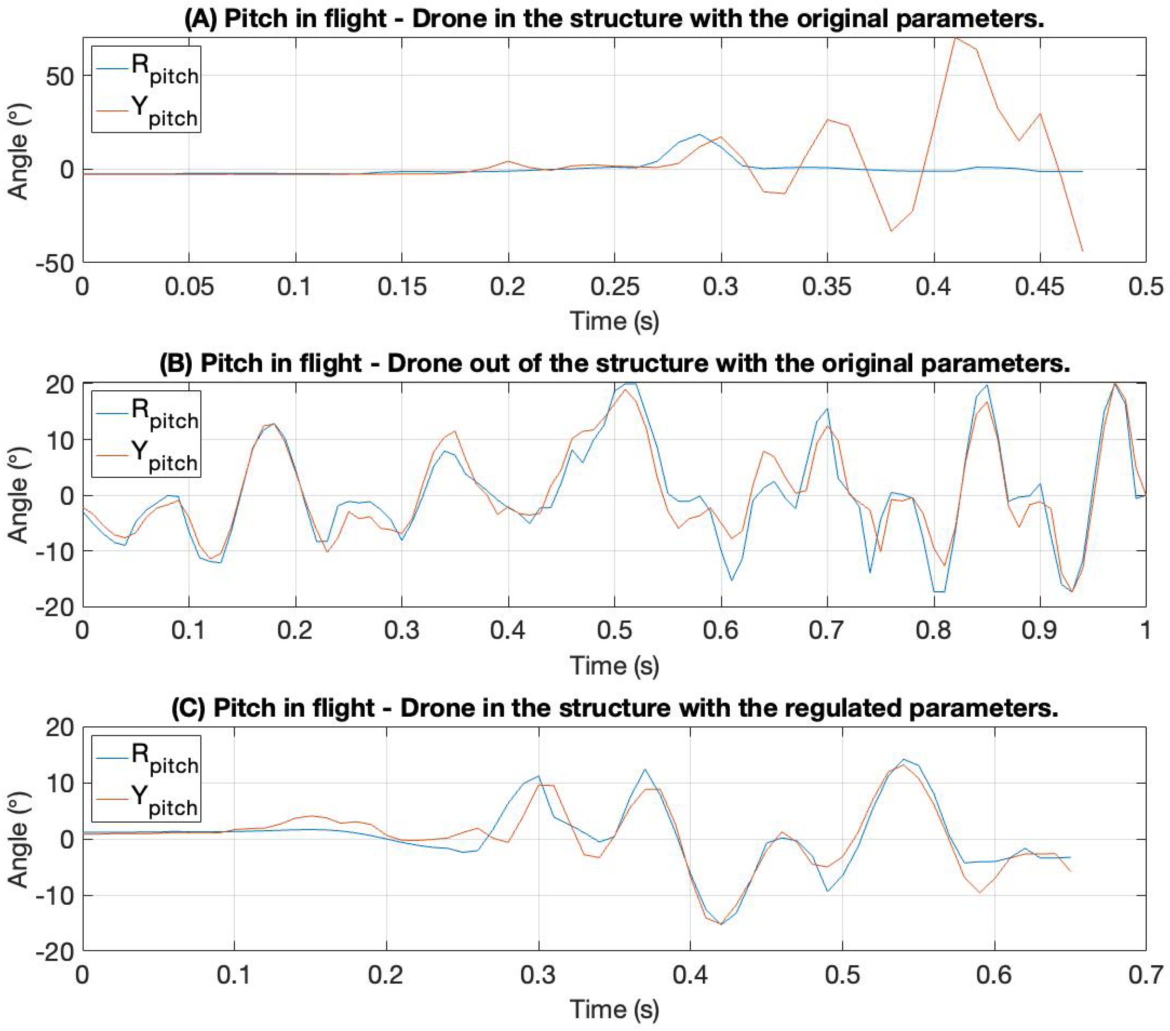


Figure 28 – Portion of data collected from drone pitch response in outdoor flight. (A) Drone with structure and original factory control parameters $\rho_{original}$. (B) Drone without structure and original factory control parameters $\rho_{original}$; (C) Drone with structure and control parameters to $\hat{\rho}^{*}$.

CHAPTER 5 – CONCLUSIONS

In this work we aimed to transform a nano drone in a handleable flying object, as a first step towards an ambitious goal of creating a revolutionary juggling prop. Then we peruse to resolve the problem of creating a rigid and light-weight structure to be attached to the drone. Which provoked the need to recalibrate the drone control algorithm.

Using carbon fiber rods and 3D print parts, we manage to design and build a structure that weighs less than 7 grams and fits the chosen drone inside. It fulfilled the role of protect the juggler hands and also protect the quadrotor form occasional crashes. Additionally, the structure modular design makes very easy to turn it in a much denser structure once we have the hardware with more lift capability.

And, using the E-FRIT algorithm, we menage to recalibrate the drone controllers hence it could fly with the new structure. And because this algorithm doesn't need a lot of experimental data, it will be easily replicated for any future iteration of this project hardware.

However, even after we calibrated all the other controllers, we are not sure that the current prop will be able to meet the performance needed for the proposed show. Meanwhile, the versatility and modularity of the developed solutions will ensure that these solutions can be used regards of the future hardware modifications that certainly will be made in order to active the proposed performance. And even knowing that until this prop arrives on a stage that will be a lot of effort, the progress developed during this work was a solid start for this project final goal.

6.1 FUTURE WORKS

Definitely the next step of this project is regulating the yaw axis for the drone to recover its steering capability. In order to do that, we will need to build another test stand to isolate the yaw movement. However, now that the drone has regained its ability to fly, it may be possible to acquire the yaw data during flight, therefor spare the creation of another test stand. Then, it would be beneficial to mount the additional sensors already acquired, to investigate if the Crazyflie remains with the same autonomous flight capability. If hence, investigate if the drone has the trust capability to perform the maneuver illustrate on Figure 2. And surly, we could not fail to mention the build of a more capable drone using the Crazyflie Bolt [7] and bigger motors and battery.

We have a few suggestions for anyone how decides to build on this work. One could try to use carbon fiber manufacture techniques to bend the rods and then make a spherical structure. Would also be interesting to investigate if there is really no advantage in adjust the pitch and row controllers with different parameters. In this work we were also assuming that the original controllers of the drone were the optimal ones, this was translated in the way that we chose the desired transfer function T_d, but this may not be the case, and would be interesting to investigate others values for T_d.

BIBLIOGRAPHY

[1] A. Lewbel, "History of Juggling," Boston College, March 2002. [Online]. Available: https://sites.google.com/bc.edu/arthur-lewbel/history-of-juggling. [Accessed May 2021].

[2] Cirque du Soleil, ETH Zurich and Verity Studios, "SPARKED: A Live Interaction Between Humans and Quadcopters," Set 2014. [Online]. Available: https://www.youtube.com/watch?v=6C8OJsHfmpI. [Accessed December 2020].

[3] G. S. Warrington, "Juggling Probabilities," *Taylor & Francis, Ltd. on behalf of the Mathematical Association of America,* pp. 105-118, February 2005.

[4] Bitcraze, "Crazyflie 2.1," [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/. [Accessed December 2020].

[5] Bitcraze, "Bitcraze," [Online]. Available: https://www.bitcraze.io/about/bitcraze/. [Accessed January 2020].

[6] Bitcraze, "Bitcraze Store," [Online]. Available: https://store.bitcraze.io. [Accessed January 2021].

[7] Bitcraze, "Crazyflie Bolt," [Online]. Available: https://store.bitcraze.io/products/crazyflie-bolt. [Accessed December 2020].

[8] C. Chadehumbe and J. Sjöberg, "Autonomous flight of the micro drone Crazyflie 2.1 through an obstacle course," *Examensarbete 15 hp,* Jun 2020.

[9] Bitcraze, "State estimation: To be or not to be!," [Online]. Available: https://www.bitcraze.io/2020/01/state-estimation-to-be-or-not-to-be/. [Accessed December 2020].

[10] A. Kumar and A. Zheng, "Using Reinforcement Learning for Real-Time Trajectory Planning of Aerial Multi Agent Systems," *TJHSST Computer Systems Research Lab,* 2019.

[11] W. Hönig and N. Ayanian, "Flying Multiple UAVs Using ROS," *Springer International Publishing AG,* pp. 83-118, 2017.

[12] J. A. Preiss, W. Hönig, G. S. Sukhatme and N. Ayanian, "Crazyswarm: A Large Nano-Quadcopter Swarm," *IEEE International Conference on Robotics and Automation (ICRA),* pp. 3299-3304, 3 June 2017.

[13] J. Xu, "Experiments of Constrained Distributed Optimization by Using UAVs," California Digital Library, 2020.

[14] Bitcreze, "Flow deck v2," [Online]. Available: https://store.bitcraze.io/collections/decks/products/flow-deck-v2. [Accessed December 2020].

[15] Bitcraze, "Lighthouse positioning deck," [Online]. Available: https://store.bitcraze.io/collections/decks/products/lighthouse-positioning-deck. [Accessed December 2020].

[16] B. Gabrich, D. Saldaña, V. Kumar and M. Yim, "A Flying Gripper Based on Cuboid Modular Robots".

[17] J. F.¨orster, "System Identification of the Crazyflie 2.0 Nano Quadrocopter," *Institute for Dynamic Systems and Control Swiss Federal Institute of Technology (ETH) Zurich,* 2015.

[18] M. A. C. Bortoleto, "Rola-bola: iniciação," *Movimento & Percepção,* vol. 4, no. ISSN 1679-8678, pp. 100-109, 2004.

[19] Bitcraze, "Controllers in the Crazyflie," [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/. [Accessed January 2021].

[20] J. B. Kuipers, QUATERNIONS AND ROTATION SEQUENCES. A Primer with Applications to Orbits, Aerospace, and Virtual Reality., Princeton, New Jersey: PRINCETON UNIVERSITY PRESS, 1999.

[21] R. Niemiec and F. Gandhi, "Multirotor Controls, Trim, and Autonomous Flight Dynamics of Plus- and Cross-Quadcopters," *Rensselaer Polytechnic Institute, Troy, New York,* vol. 12180, no. DOI: 10.2514/1.C034165.

[22] Bitcraze, "The Coordinate System of the Crazyflie 2.X," [Online]. Available: https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/. [Accessed January 2021].

[23] F. L. Markley and J. L. Crassidis, "2.6.2 Inertial Reference Frames," in *Fundamentals of Spacecraft Attitude Determination and Control*, SPACE TECHNOLOGY LIBRARY; Microcosm Press; Springer, 2014.

[24] Bitcraze, "crazyflie-firmware/src/modules/src/pid.c," [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/pid.c. [Accessed January 2021].

[25] S. Soma, O. Kaneko and T. Fujii, "A NEW METHOD OF CONTROLLER PARAMETER TUNING BASED ON INPUT-OUTPUT DATA -FICTITIOUS REFERENCE ITERATIVE TUNING (FRIT)-," in *IFAC Workshop on Adaptation and Learning in Control and Signal Processing, and IFAC Workshop on Periodic Control System*, Yokohama, Japan, 2004.

[26] H. Hjalmarsson, M. Gevers, S. Gunnarsson and O. Lequin, "Iterative feedback tuning: theory and applications," *IEEE Control Systems Magazine,* vol. 18, pp. 26-41, 8 1998.

[27] M. C. Campi, A. Lecchini and S. M. Savaresi, "Virtual reference feedback," *Automatica,* pp. 1337-1246, 2202.

[28] A. JULKANANUSART, “Quadrotor Tuning for Attitude Control based on PID Controller using Fictitious Reference Iterative Tuning (FRIT),” *Sirindhorn International Institute of Technology - Thammasat University,* p. 105, 2016.

[29] S. Masuda and Y. Yasuda, "A PID Gain Tuning Using Fictitious Reference Iterative Tuning Approach with Simultaneous Tuning for the Delay Parameter of the Reference Model," in *SICE Annual Conference*, The University Electro-Communications, Japan, 2008.

[30] O. Kaneko, K. Yoshida, K. Matsumoto and T. Fujii, "A New Parameter Tuning for Controllers Basead on Least-Squares Method by using One-Shot Closed Loop Experimental Data - An Extension of Fictitious Reference Iterative Tuning," *Transactions of the Institute of Systems, Control and Information Engineers,* pp. 400-409, 2005.

[31] M. Kano, K. Tasaka, M. Ogawa, S. Ootakara, A. Takinami, S. Takahashi and S. Yoshii, "Practical Direct PID/I-PD Controller Tuning and Its Application to Chemical Processes," in *2010 IEEE International Conference on Control Applications Part of 2010 IEEE Multi-Conference on Systems and Control*, Yokohama, Japan, September 8-10, 2010, 2010.

[32] "What Is MATLAB?," MathWorks, [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html. [Accessed May 2021].

[33] M. Kano and M. Ogawa, "E-FRIT MATLAB Toolbox/Program," JSPS PSE 143rd committee, 22 May 2010. [Online]. Available: http://e-frit.chase-dream.com/matlab-ENG.html. [Accessed February 2021].

[34] "lsim," MathWorks, [Online]. Available: https://www.mathworks.com/help/control/ref/lti.lsim.html. [Accessed May 2021].

[35] "fmincon," MathWork, [Online]. Available: https://www.mathworks.com/help/optim/ug/fmincon.html?s_tid=doc_ta. [Accessed May 2021].

[36] "Constrained Nonlinear Optimization Algorithms," MathWorks, [Online]. Available: https://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html#brnox01. [Accessed May 2021].

[37] "Choosing the Algorithm," MathWorks, [Online]. Available: https://www.mathworks.com/help/optim/ug/choosing-the-algorithm.html#brppuoz. [Accessed May 2021].

[38] MathWorks, "tfest," [Online]. Available: https://www.mathworks.com/help/ident/ref/tfest.html. [Accessed May 2021].

[39] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences,* 2017.

[40] L. Brooke-Holland, "Unmanned Aerial Vehicles (drones): an introduction," *International Affairs and Defence,* no. SN06493, 2012.

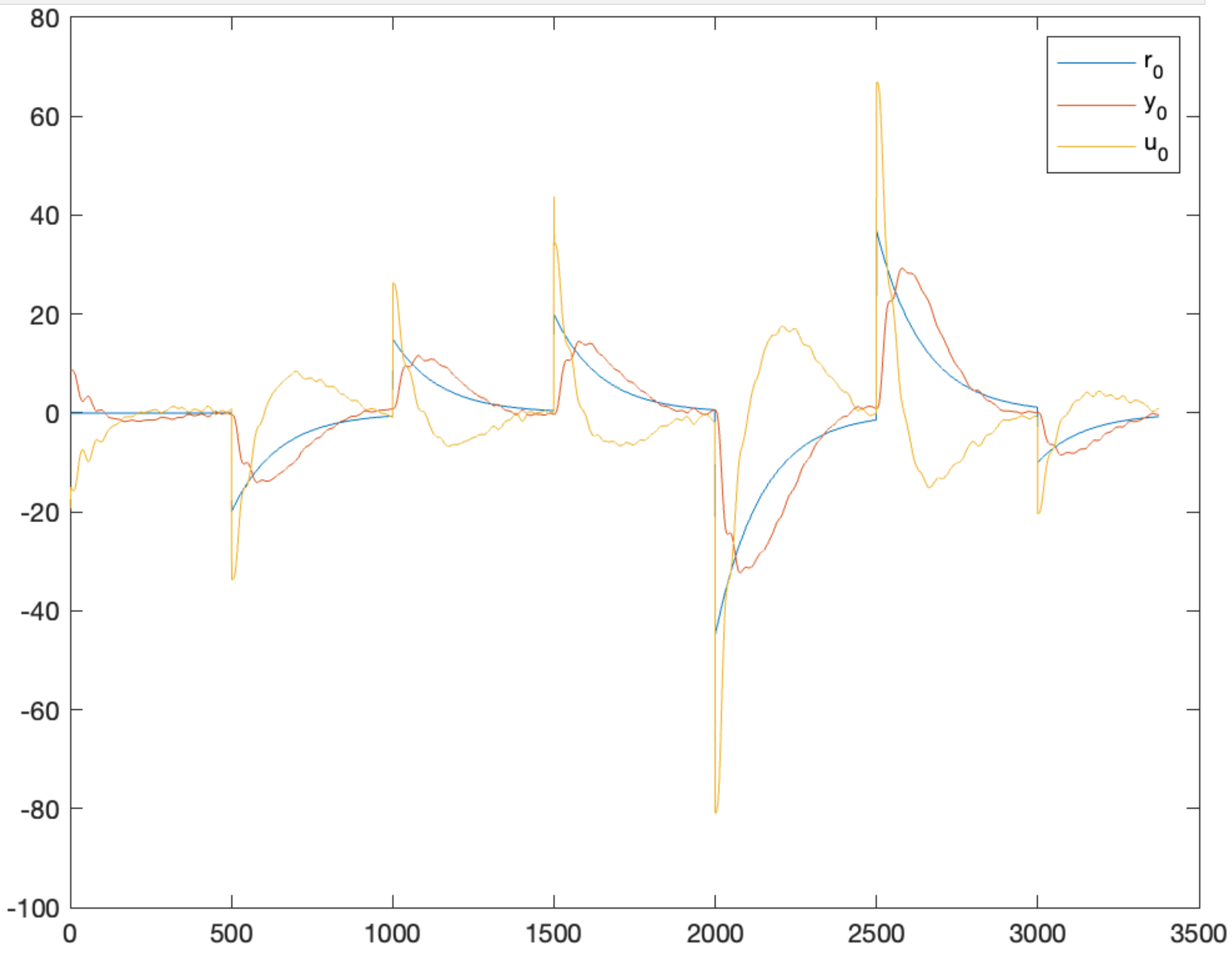
ANNEX I - THE E-FRIT SCRIPT

```
clear
global y0 Tds T u0 z lambda fs

load("data/e_frit_data.mat");
T = 1/500;
z = tf([1 0], 1, T);
lambda = 1;
```

I/O data use in the e-frit:

```
plot(r0)
hold on
plot(y0)
plot(u0)
legend("r_0","y_0","u_0")
```



Desired transfer function used undelayed:

```
Tds
```

```
Tds =

  From input "u1" to output "y1":
     8.564 s^5 - 2202 s^4 + 5.862e05 s^3 + 1.397e07 s^2 + 4.031e07 s + 7.493e06
  ---------------------------------------------------------------------------------
  s^6 + 112 s^5 + 1.675e04 s^4 + 1.027e06 s^3 + 1.42e07 s^2 + 4.021e07 s + 7.601e06

Continuous-time identified transfer function.

Parameterization:
   Number of poles: 6   Number of zeros: 5
   Number of free coefficients: 12
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "data".
Fit to estimation data: 86.54%
FPE: 2.05, MSE: 2.026
```

Initial Kp, Ki, e Kd values:

```
ro0(1:3)
```

```
ans = 1×3
    1.7344    1.4450    0.0010
```

Call the optimization function to find the best ro value:

```
ro = ro0;
options = optimset();
[ ro, fval, exitflag, output] =
fmincon('performance_index',ro,[],[],[],[],[0, 0, 0, 0.0001],300*[1, 1, 1,
1],'funcon',options);
```

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

Estimated optimal parameters:

```
ro(1:3)
```

```
ans = 1×3
    4.2039    4.7822    0.0000
```

Function that will calculate the performance index of the E-FRIT method.

This value will be used in the nonlinear optimization function:

```
function [J] = performance_index(ro)
    global Tds y0 T u0 z lambda

    Tdsd = Tds * tf(1,1,'ioDelay',ro(4));
    Tdd = c2d(Tdsd,T,'zoh');
    C = ro(1) + ro(3)*(1 - z^(-1))/T + ro(2)*T/(1 - z^(-1));
    ri = lsim(1/C,u0) + y0;
    yi  = lsim(Tdd,ri);
    e   = y0 - yi;
    Je  = ( e' *  e ) / length(y0);
    ei = ri - yi;
    ui = lsim(C,ei);
    dui = ui - [0; ui(1:end-1)];
    Ju = ( lambda * ( dui' * dui ) ) / length(dui);
    J  = Je + Ju;
end
```

Nonlinear constraints used by the optimization function:

```
function [ c, ceq ] = funcon(theta)
           c = [];         % no constraints
           ceq = [];       % no constraints
end
```

ANNEX II - CHOOSING TD

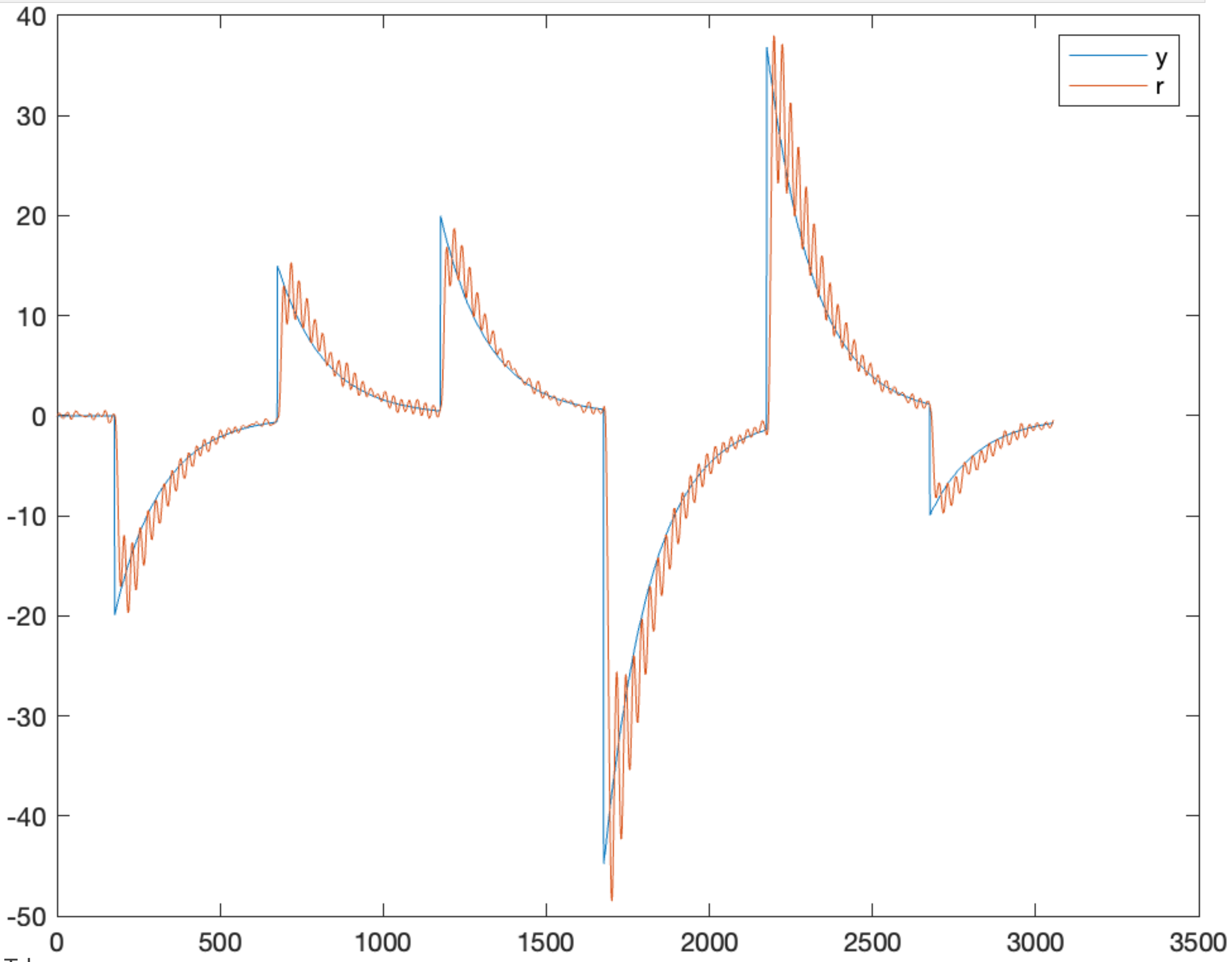
```
clear
load("data/dataset_to_find_Td.mat")
T = 1/500;
```

Loaded data:

```
plot(r)
hold on
plot(y)
legend("y","r")
```

Estimated Td:

```
data = iddata(y,r,T);
Tds = tfest(data,6)
```



```
Tds =

  From input "u1" to output "y1":
     8.564 s^5 - 2202 s^4 + 5.862e05 s^3 + 1.397e07 s^2 + 4.031e07 s + 7.493e06
  ---------------------------------------------------------------------------------
  s^6 + 112 s^5 + 1.675e04 s^4 + 1.027e06 s^3 + 1.42e07 s^2 + 4.021e07 s + 7.601e06
```

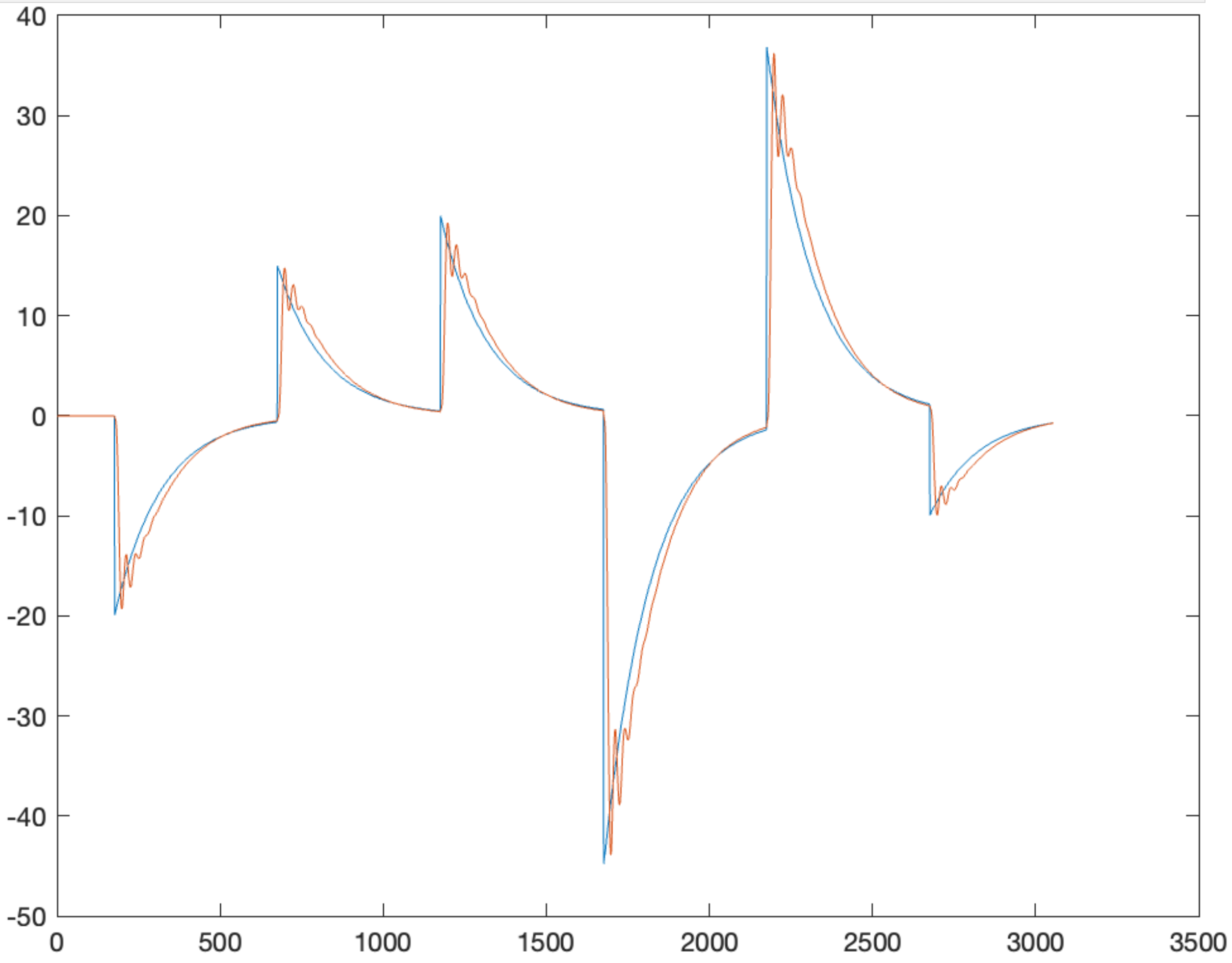
```
Continuous-time identified transfer function.

Parameterization:
   Number of poles: 6   Number of zeros: 5
   Number of free coefficients: 12
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "data".
Fit to estimation data: 86.54%
FPE: 2.05, MSE: 2.026
```

Td response:

```
vecT = 0:T:T*(size(r, 1) - 1);
Yd = lsim(Tds,r,vecT);
figure
plot(r)
hold on
plot(Yd)
```



40
30
20
10
0
-10
-20
-30
-40
-50
0
500
1000
1500
2000
2500
3000
3500