TREBALL DE FI DE GRAU

TÍTOL DEL TFG: Improvement of AOCS for nanaosatellite based on Helmholtz coils software design using a GUI and the implementation of a new system for measure angular velocities.

TITULACIÓ: DG ENG AERO/SIS TEL

AUTOR: Ulises Ortega Collado

DIRECTOR: Hyuk Park

SUPERVISOR: David Llaveria Godoy

DATA: September 6, 2022

Title : Improvement of AOCS for nanaosatellite based on Helmholtz coils software design using a GUI and the implementation of a new system for measure angular velocities.
Author: Ulises Ortega Collado

Advisor: Hyuk Park

Supervisor: David Llaveria Godoy

Date: September 6, 2022

Overview

This work contains the analysis and improvements regarding attitude nanosatellite testing procedures. An utterly new system for measuring angular velocities has been designed from scratch in order to be applied to Cubesat testing campaigns. This system will help to track the rotations of the nanosatellite in the different tests. Knowing the angular velocity of the nanosatellite while performing the tests is so important. It allows to calibrate the gyroscopes as well as set a perfect and knows conditions to the satellite. Also, the measurements can be compared with the gyroscope angular velocityes obtained in order to recalibrate the system if necesary. Furthermore, a new design for Helmholtz coils software has been proposed using a user Interface in C++. This new software allows the user to initialize the coils with the desired magnetic fields required for testing. Two main operating ways has been considered: Static and Dynamic orbit magnetic field generation. Both systems can work altogether in order to check the attitude determination and control of the nanosatellite during testing campaigns. Nanosatellite needs to perform testing before its launch. Thus, the whole project avoids expensive risks which could doom the mission. The results obtained in this work show that the angular tracking works as desired for the testing purposes and the Helmholtz coils GUI simplifies a lot the testing process from the user point of view.

Títol: Improvement of AOCS for nanaosatellite based on Helmholtz coils software design using a GUI and the implementation of a new system for measure angular velocities.
Autor: Ulises Ortega Collado

Director: Hyuk Park

Supervisor: David Llaveria Godoy

Data: 6 de setembre de 2022

Resum

Aquest treball conté l'anàlisi i les millores pel que fa als procediments de prova de nano satèl·lits d'actitud. Un sistema completament nou per mesurar velocitats angulars ha sigut dissenyat des de zero per aplicar-lo a les campanyes de proves de Cubesat. Aquest sistema ajudarà a seguir les rotacions del nano satèl·lit en les diferents proves. Conèixer la velocitat angular del nano satèl·lit mentre es realitzen les proves és molt important, ja que permet calibrar els giroscopis així com establir unes condicions perfectes i conegudes al satèl·lit. A més a més, les mesures es poden comparar amb les velocitats angulars obtingudes del giroscopi per recalibrar el sistema en cas que sigui necessari. Per una altra banda, s'ha proposat un nou disseny per al programari de bobines Helmholtz utilitzant una interfície d'usuari en C++. Aquest nou programari permet a l'usuari iniciar les bobines amb els camps magnètics desitjats i necessaris per a la prova. S'han considerat dues vies de funcionament principals: la generació de camps magnètics d'òrbita estàtica i dinàmica. Ambdós sistemes poden funcionar conjuntament per comprovar la determinació de l'actitud i el control de les campanyes de proves de nano satèl·lits. Els nano satèl·lits han de dur a terme proves abans del seu llançament per tal d'evitar perills que posarien en risc la missió. Els resultats assolits en aquest treball mostren que el seguiment angular funciona satisfactòriament per a les proves i la GUI de les bobines de Helmholtz simplifica molt el procés de prova des del punt de vista de l'usuari.

Gracias a dios por llevarnos al cielo y poderlo contemplar.

CONTENTS

Introduction **1**

CHAPTER 1. Introduction **3**

1.1. CubeSats **3**

1.2. Attitude Determination and Control System (ADCS) **4**

1.3. Sensors and actuators **6**

1.3.1. Magnetometers 6
1.3.2. Gyroscopes 6
1.3.3. Magnetorquers 9
1.3.4. ADCS testing 10

1.4. Helmholtz Coils **13**

1.4.1. Structure 14
1.4.2. Magnetic field generations 18

1.5. Orbit Propagators **23**

1.5.1. SGDP4 explanation and some necessary coordinate transformation explanations. 23
1.5.2. IGRF magnetic fields simulator. Explanation and some necessary explanations. 26

CHAPTER 2. Software Design **29**

2.1. Current design **29**

2.2. New design working Principle **32**

2.2.1. From matlab to Code structure in C++. 32
2.2.2. Software Architecture 36

CHAPTER 3. Angular velocity measurements **51**

3.1. Working principles **51**

3.1.1. Angle measurement 52

3.2. Auto shape detector **56**

3.2.1. Linear least squares fitting. 56
3.2.2. Auto shape detector with Hough circle 61

3.3. Trackers 68

3.4. Tracker selection, patterns and tests. 71

3.5. Correlation filter trackers limitation and FPS. 83

3.6. Code explanation 91

CHAPTER 4. Future work 95

Conclusions 97

Bibliography 99

APPENDIX A. C++ Helmholtz coils code 105

APPENDIX B. Coils study 169

APPENDIX C. Angular tracking Algorithm code 173

APPENDIX D. Circular fit 199

APPENDIX E. Angular tracking study 201

LIST OF FIGURES

1.1 3-Cat-4 . . . 3
1.2 Attitude control modes [1] . . . 4
1.3 Simplified block diagram of the Attitude Determination and Estimation mode. [1] 5
1.4 Gyroscope working principle. [2] . . . 7
1.5 . Different Noise processes on an Allan Variance Plot [6] . . . 8
1.6 Magnetorquers for nanosatellite porpuses [3] . . . 10
1.7 Helmholtz coils in NanoSatLab . . . 13
1.8 Helmholtz structure [23] . . . 14
1.9 Helmholtz circuit design[**?**] . . . 15
1.10 Helmholtz circuit structure [23] . . . 15
1.11 Helmholtz circuit structure with bridge . . . 16
1.12 Helmholtz circuit structure with bridge . . . 16
1.13 Helmholtz circuit structure with bridge in Z-axis . . . 17
1.14 Front and rear side of the aluminium loop connection . . . 17
1.15 Helmholtz circuit structure with bridge in Z-axis [25] . . . 18
1.16 Slope of the currents vs Magnetic fields . . . 19
1.17 Analysis of the theoretical precision . . . 20
1.18 Bx magnetic field generated . . . 21
1.19 By magnetic field generated . . . 21
1.20 Bz magnetic field generated . . . 22
1.21 Two Line Elements example . . . 23
1.22 ECI and ECEF reference frames [28] . . . 24
1.23 Orbit frame [28] . . . 25
1.24 Target frame [28] . . . 25
1.25 IGRF Hystory [32] . . . 26
1.26 Propgataed orbit from the TLE . . . 27
1.27 Propagated orbit from the TLE with IGRF magnetic fields X-axis . . . 27
1.28 Propagated orbit from the TLE with IGRF magnetic fields Y-axis . . . 28
1.29 Propagated orbit from the TLE with IGRF magnetic fields Z-axis . . . 28

2.1 Matlab code structure . . . 29
2.2 Static Orbit simulation overview code (Matlab) . . . 30
2.3 Matlab usage of the serial communication . . . 30
2.4 Matlab send currents to powerSupply function . . . 31
2.5 Matlab function to send the signs . . . 31
2.6 Dynamic orbit simulation loop . . . 32
2.7 Arduino communication structure . . . 33
2.8 communication structures . . . 33
2.9 Fuction to send signs via serial . . . 33
2.10 send characters via Serial fuctions . . . 34
2.11 Open a serial object command . . . 34
2.12 Serial used functions . . . 35
2.13 Send currents via serial function . . . 35
2.14 magnetic fields file format . . . 37

2.15 static orbit simulation functions . . . 38
2.16 static orbit simulation functions . . . 39
2.17 static orbit simulation functions . . . 40
2.18 static orbit simulation functions . . . 41
2.19 Interface process design . . . 42
2.20 static orbit simulation functions . . . 43
2.21 Included headers . . . 44
2.22 Interface structure . . . 46
2.23 Interface structure . . . 47
2.24 Interface structure . . . 48
2.25 File data selection . . . 49

3.1 general flow chart for the angular velocity algorithm . . . 51
3.2 System calibration . . . 52
3.3 code to select the window template (Python) . . . 53
3.4 code to select the window template (Python) . . . 53
3.5 scheme to limit the dimensions of the template . . . 54
3.6 Final lap problem scenario . . . 55
3.7 Circular fit scheme . . . 56
3.8 Parabola and its circular fit . . . 58
3.9 Conics sections which would be observed by the camera . . . 59
3.10 Ellipse: most common conic section captured by a camera . . . 59
3.11 Matrix with the data points [14] . . . 60
3.12 Ellipse condition [14] . . . 60
3.13 Data set to find its centre . . . 62
3.14 Set of cones used to find the shape center . . . 62
3.15 Set of cones used to find the shape center (variable radius) . . . 63
3.16 Hough circle algorithm pseudocode . . . 63
3.17 Representation of the algorithm for non perfected shapes . . . 64
3.18 Hough circles steps . . . 64
3.19 Auto shape detect without previous design . . . 66
3.20 Auto shape detected succesfully . . . 67
3.21 MOSSE flowchart . . . 70
3.22 Basic pattern analysis . . . 71
3.23 2D FFT of the chess pattern using fftshift() . . . 72
3.24 First frequencies for the FFT 2x2 chess . . . 72
3.25 FFT chees pattern evaluated at the origin (f1=0) . . . 73
3.26 Tracking patterns used . . . 74
3.27 Circle pattern and FFT . . . 75
3.28 2D FFT of the template pattern . . . 76
3.29 FFT of the template pattern . . . 76
3.30 First frequencies for the FFT circle . . . 77
3.31 Circle pattern and FFT . . . 77
3.32 2D FFT of 3.31(a) . . . 78
3.33 2D FFT of 3.31(b) . . . 78
3.34 Trackers list (Python) . . . 79
3.35 Tracking lose analisys . . . 80
3.36 Circle patterns colors tested . . . 81

3.37 BOOSTING RedYellow . . . 81
3.38 Performance analysis for pattern 1 . . . 82
3.39 Performance analysis for pattern 2 . . . 82
3.40 Camera used: Logitech - P/N: 960-001063 . . . 83
3.41 Scenario proposed at each sample time . . . 83
3.42 Maximum velocities reached . . . 85
3.43 Tracking algorithm comparison and testing . . . 85
3.44 Tracking algorithm comparison and testing (moving average) . . . 85
3.45 Overall error analysis . . . 87
3.46 Tracking lose analisys . . . 87
3.47 error analysis by steps . . . 88
3.48 Designed tracking algorithm with vertex sampling . . . 91
3.49 Designed tracking algorithm with time sampling . . . 92

4.1 Camera not located at center . . . 95
4.2 Camera located at the center . . . 96

INTRODUCTION

Space missions involving small satellites require high attitude determination and control systems (ADCS) performances. However, launching satellites without avoiding risks is a very expensive task due to the problems and experience the team could have while the satellite reaches the hash space environments. The solutions to avoid any type of failure is to test all the systems on the ground. Nowadays, testing campaigns are mandatory before any type of launch if the mission requires succed. Therefore, the only way to test the behaviour on ground is to create the most similar environment on the laboratory, that the satellite will have on orbit. The responsible of creating those artificial environments is the so called, Helmholtz coils equipment. This work covers the demonstration of the Helmholtz coils' working principle and the new design from scratch of the a new application that controls the coils with the aim of automate all the testing process. Indeed, testing can be a slow process since the coils in its early origins, it was coded in Matlab and was not so intuitively for the testing non expert users. Currently, the new design designed in C++ allows the testing process to be more fluent and thus, focus more carrefuly on the real satellite features while testing it. Thus, the application code has the purpose of ease all the testing process and standardize a solution in C++ that will allow future generations of Helmholtz coils software to upgrade with even more intuitive interfaces. Also, the idea of coding in C++ the coils allowed from the beginning to embedde the system to microcontrollers such as raspeberyPi that could control Helmholtz coils in less space and with different and more intuitively features. Helmholtz coils allow to place the small satellite at the disk allowing almost frictionless environments ready to test the sensors and actuators that where packed in the satellite. Therefore, once the Helmholtz coils have created the desired orbit condition, the actuators can start performing the desired actions while the sensors read the Helmholtz artificial conditions. Furthermore, ADCS testing is analysed carefully from the angular movements point of view. Indeed, gyroscopes are fundamental to perform a correct following of the satellite control. However, gyroscopes need information on the angular velocities in order to be calibrated and thus, perform some testing procedures as detumbling or general estimation of the satellite position. This work shows the design from scrath of a new method used to measure angular velocities that will be used for calibrating and general testing purposes. Image processing and Python are the proposed candidates to solve the problem with all the obstacles that a visual system can have while operat- ing in real time. Indeed, real time video algorithm processing shows the problems that will be analysed in order to prove that the system works even though other different solutions could have been applied. The solution has been analised considering its strengths and weaknesses. Focusing on the most important flaw, the system must track correctly a template or target inside the coils disk. If the tracking is lost the system can lose all the further measurements. On the othe hand, the first strength is adaptability: the system designed can measure angular velocities further than the offered by the helmholts coils disk... it can measure any type of circular angular movement! For any Helmholtz coil in the world or other type of industrial purpose. This solution in conjuction with the Helmholtz coil, will allow to perform ADCS testing in order to prepare the Cubesat correctly before launch.

CHAPTER 1. INTRODUCTION

1.1. CubeSats

CubeSats are the type of nanosatellites that California Polytechnic State University and Leland Sandford Junior University began developing in 1999. They are characterized mainly by their cubic unitary shape (100x100x100 mm) and their low weight (less than 1.33 kg). These cubic satellite units can be used as stand-alone units or in groups of multiple units. The unit of CubeSat can weight up to a maximum of 1.33 kg. Forming groups of multiple units up to a maximum of 24.

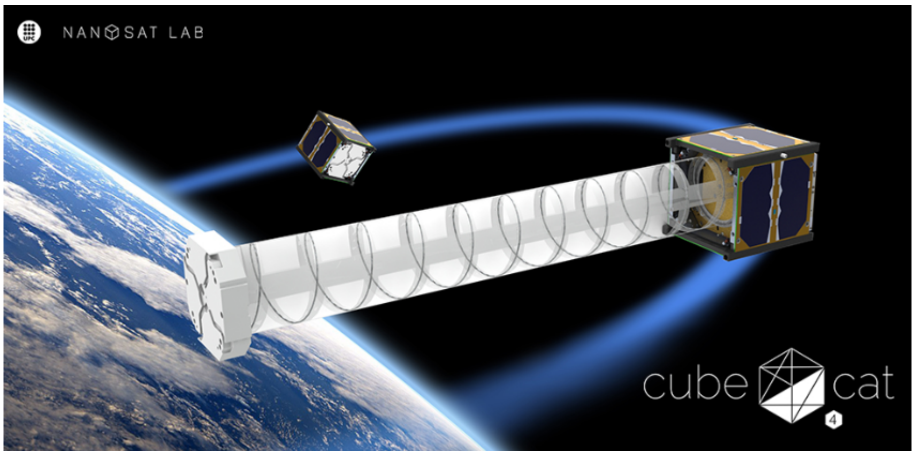


Figure 1.1: 3-Cat-4

The main purpose of the development of this standard began with the aim of bringing together space projects and research developed by universities. With the evolution of satellites to date, the complexity of these systems did not allow students to be involved in the systems, during their training, in the development of one of them from the beginning to the end. The development of the CubeSat design made it possible to reduce the costs and the amount of time required for its manufacture. These advantages led to the establishment of a design standard, which is included in the CubeSat Design Specification [18]. Nowadays, these nanosatellites have been developed to cover different objectives in their use and operation. Thus, they are mainly used in missions of technological demonstrations, scientific research, international projects, and even for commercial purposes. In most cases, to reduce the costs of launching these satellites, space launches of larger satellites are used.

CubeSats are formed by a group of systems in charge of the nanosatellite integrity and the mission requieremnts. Although payloads as cameras or RF antennas can vary depending on the mission. The designed systems of CubeSat for the ongoing mission are well defined as Attitude control modes.
1) Attitude Determinationand Estimation(ADE) is in charge of know the position of the satellite as well as its attitude
2) Attitude Stabilization Mode is in charge of change dynamically the orientation of the satellite throughout the orbit
3) Earth pointing Mode needs to point the satellite to a desired location

4) Sensors are space readers
5) Actuators must be activated when the other attitude control systems need to operate.

1.2. Attitude Determination and Control System (ADCS)

Attitude Determination and Control System(ADCS) is often divided in the determination and control processes. Determination consists on measuring with the sensors some physical parameter as the sun position, magnetic fields or rotation and process the data with different algorithms in order to estimate the satellite orientation with respect of a given reference frame. Magnetometer, gyroscopes and sun sensors measures attitude data from the CubeSat and the measurements are combined to estimate attitude.

On the other hand, attitude control is in charge of stabilizing and canceling the angular velocity of the CubeSat. Furthermiore, it must control the orientation of the satellite with respect of a reference frame in order to have line of sight with a defined target, orientate the solar panels, or change the temperature of the satellite among other functions.

Although the algorithm for attitude determination have not been studied in this work, the testing required for CubeSat mission can be well understood. The sensors involved in the work are gyroscope and both magnetorquers and magnetometers. Thus, this work focuses on the tests performed by the magnetometers and magnetorquers while are inside the Helmholtz coils and the gyroscope measurement using the new tracking algorithm.

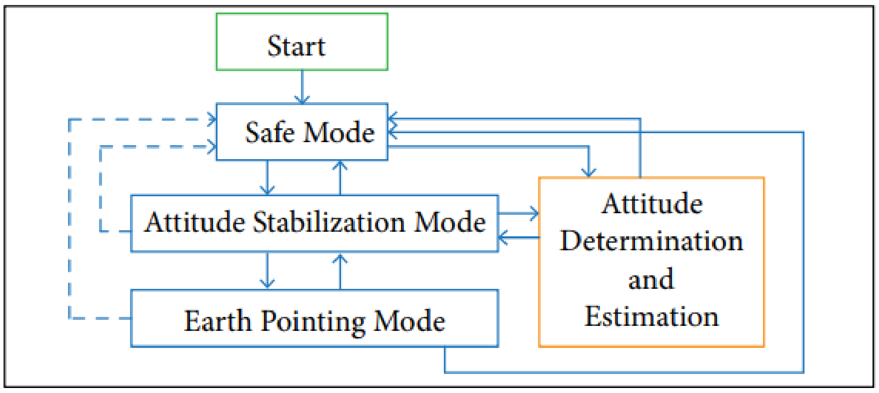

Start
Safe Mode
Attitude Stabilization Mode
Attitude Determination and Estimation
Earth Pointing Mode

Figure 1.2: Attitude control modes [1]

Control requirements needs to overcome 4 possible modes in order to operate correctly. Firstly, after orbit injection and starting the onboard computer, the first state the satellite takes is the Safe Mode (SM), in which the CubeSat does not take any action regarding ACS processes. In this mode is used to send onboard telemetry and general ground command. Although the satellite can communicate with ground, it still cannot switch on sensor or actuators. Using ground command, the SM can change to Attitude Stabilization Mode(ASM) or to Attitude Determination and Estimation Mode(ADE). Notice that ADE is operating in all modes except the SM. Instead of an operation mode, it is considered a true mode since the satellite needs attitude knowledge even when the attitude is not being stabilized or controlled. Earth pointing mode, or Target pointing can start by checking the attitude determination error with the telemetry. In figure 1.2, dotted lines means autonomous reconfiguration onboard commands. Conversely, filled lines represents ground commands. Once the ADE have data enough, Target Pointing Mode can operate in order to properly point the satellite to Earth or other target. Two conditions must be fulfilled in order to operate in Earth Pointing Mode (EPM) when only the ADE in active. Satellite Angular rate must be small and attitude must be well known. The ASM must dump the angular rate up to 0.6 degrees/s. Two algorithms can be implements in ASM: B-Dot or Rate Reduction Mode(RR)[1] [2].

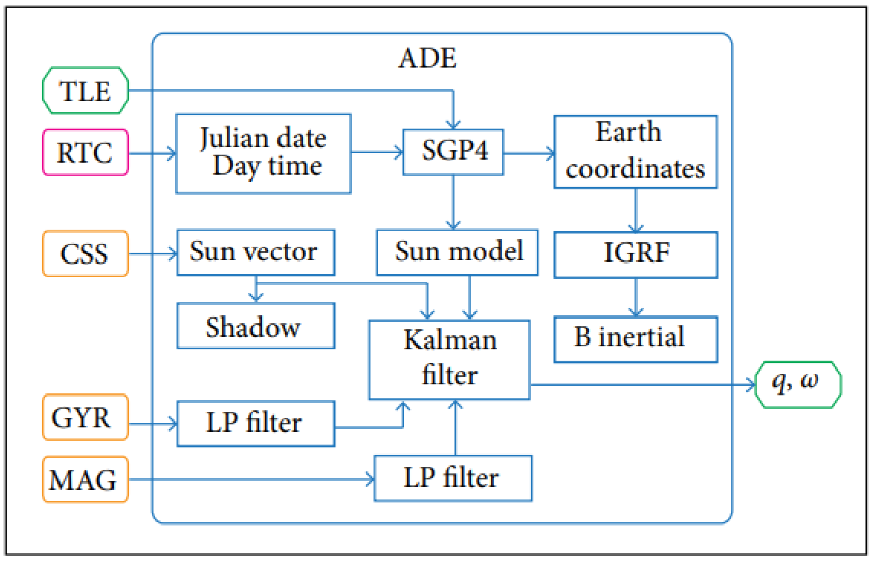


Figure 1.3: Simplified block diagram of the Attitude Determination and Estimation mode. [1]

Two methods are traditionally used for compute satellite attitude determination and estimation (ADE): TRIAD [20] and QUEST [21] . Alternatively, Kalman filter [1] can also estimate the disturbance torques. The angular rate can be computed both using a Low pass filter(LPF) or computing the numerical derivative of the attitude. Also, the number combinations to compute the attitude is generally high since the methods to compute ADE is wide ranging.

Furthermore, two vectors are used to perform attitude determination: Solar sensor reading and magnetometers sensors measures the data in the vector form in a given reference frame, usually the inertial or the orbital frame. Often this vector is computed in the onboard computer based on the IGRF algorithm and the Sun pointing vector in the inertial coordinates.

Earth magnetic field rotates sincronouslly with the Earth. The model computes the field strenght in Earth centered coordinates as function of the LLA corrdinates (latitud, longitud, altitud).Where the altitud make reference to the radius from the center of the earth to the true satellite position on orbit. Thus, the satellite must compute its propagated orbit using SGP4 algorithm to send the proper data to the geomagnetic model. SGP4 [22]takes as input a Two Line element(TLE) [38] block with a defined time delay. Finally, the magentic field will be calculated with the International Geomagnetic Reference Field (IGRF) model [31]. Figure 1.3 shows a simplified model of ADE model using kalman filter which estimated the inertial attitude rotation transformation in the quaternion form. Also finds the gyroscope bias. Finally, the angular velocity can be calculated from the gyroscope measurements applying the bias previously computed. Coarse Sun Sensor does not provide measurement if the orbit is in Earth's shadow. Thus, the satellite's onboard real time clock (RTC) is in charge of provide time and date from computations.

1.3. Sensors and actuators

The measurement of physical parameters that might change throughout the orbit employing sensors is used to determine the satellite's attitude. The determination algorithm chosen to compute the attitude and the pointing requirements for the mission specification determines the choice of the determination sensors. There are sources of interference and noise in every sensor. To get a measurement as close to the actual values as possible, they must be calibrated. The volume/mass of the sensors and their power consumption are crucial factors when choosing the proper sensor. Most of these sensors are constantly active to provide the orientation systems with as much information as possible regarding the state of the satellites to correct the attitude.

To correct rotation and orientation, the actuators are responsible for imparting precise torque to the satellite axes. Actuators can be divided into passive and active systems. Passive mechanisms stabilizes its orientation regarding the magnetic or gravitational fields of the Earth. Magnets or gravity booms are examples of passive actuators . They merely require mass and space and have no power requirements. However, it is not possible to provide them with any additional control. Due to the lack of electronic components, these systems are more dependable than active systems, but the ultimate attitude cannot be altered.On the other hand, active systems as manetorquers or reaction wheels, are widely used due to the general correct performances. However, the consume energy. For instance, magnetorquers are used at testing campaigns of 3-cat4 Nanosatellite since its performance can be checked using Helmholtz coils environments.

1.3.1. Magnetometers

Magnetometers measure the Earth's magnetic field in the sensors' reference frame while the satellite is in orbit in order to process the currents that magnetorquers must apply to the coils to compensate the magnetometer measurements in that position in orbit. Magnetometers are usually capable to operate with low power consumption and with low volume. However, they can be interfered by other electronic components that create magnetic fields as well.

1.3.2. Gyroscopes

Attitude propagation requires Gyroscopes able to track orientation or satellite angular behavior from a known point. It is the aim of the gyroscope to maintain angular velocity measurements throughout the propagation orbit. Using a joint where a rotation can occur

called gimbals, the spinning mass can rotate along the 3 axes maintaining a constant angular momentum vector aligned with the spin axis. Indeed, the gyroscope can measure the angular change of the spacecraft by the rotation of the gimbals. However, the gimbals lock is a known problem that can occur when 2 or more gimbals are aligned in the same axes. When this phenomena occurs it appears both angular measurement in both axes. This results in the lost of a degree in attitude control.
Mechanical gyroscopes implementation where traditionally used for large spacecraft. However, due to capacity and mass limitation in nano-satellites, other tiny technologies are used for ADCS. MEMS (Microelectronic Mechanical Systems) gyroscopes must be used for nano satellite missions. MEMS gyros contain a miniature piezoelectric oscillating mass that when rotated suffers the Coriolis effect and the mass is displaced from its center allowing to measure with the sensitivity the angular changes in time. When the angular changes are integrated, the angular velocity can be computed. The centrifugal forces creates the necessary motion of the mass in order to find the sensitivity of the sensor. Although, the system is robust enough to measure angular changes, it involves small piezoelectric structure which are sensitive to temperature changes. This phenomena is called temperature drifts and due to the vibrating thermal mass's mechanical properties it can lead to considerable errors in the measurements.

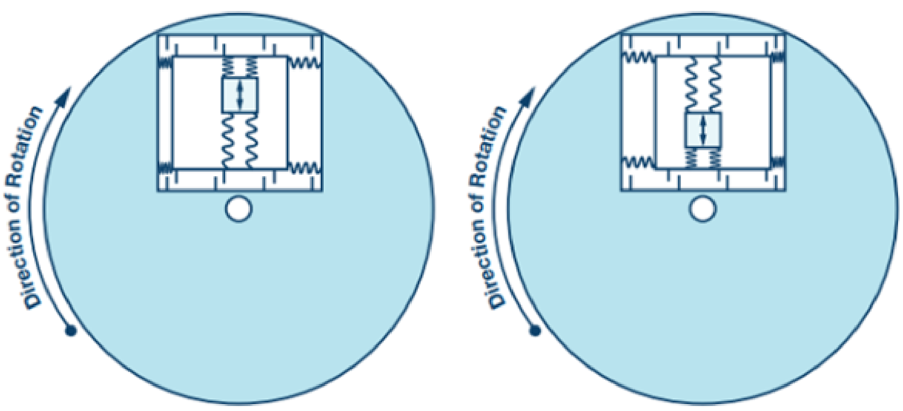


Figure 1.4: Gyroscope working principle. [2]

The entire structure is shown in Figure 3.4, which illustrates that the resonating mass experiences Coriolis acceleration as it moves and as the surface to which the gyroscope is affixed spins. Both the mass displacement and the signal resulting from the accompanying capacitance change grow along with the rate of rotation. As long as the gyroscope's detecting axis is parallel to the rotational axis, it can be positioned anywhere on the rotating object and at any angle.

Gyroscope mathematical model and calibration

Gyroscope measurements are composed of calibration terms as well as noise components in the data collected. The aim of the gyroscope sensor is to find the correct terms that aproximates the ouotput measurements to the real ω_{body} angular velocity. Some techniques are used in order to reduce noise.

$$\overline{\omega_r} = [\overline{\overline{SP}}]([\overline{\overline{SF}}](\overline{\omega_{bias}} + \overline{\omega_{body}} + \overline{e}) \tag{1.1}$$

where:

$\overline{\omega_r}$ = is the rotation reading by the sensors in the body frame

$[\overline{\overline{SP}}]$ = is the transformation matrix produced in the mechanical integration of the sensors in the satellite. It transforms from the sensors frame into the body frame.

$[\overline{\overline{SF}}]$ = scaling factor and non-orthogonally corrections.

$\overline{\omega_{bias}}$ = Rotation bias [º/s].

$\overline{\omega_{body}}$ = Rotation rate in body frame [º/s].

$\overline{e}$ = Noise [º/s].

The bias term b(t) is referred to as RRW, and the additive noise term n(t) is referred to as ARW. Therefore, noise in gyroscopes is not modeled as Gaussian white noise (GWN).

$$\overline{e} = RRW + ARW \tag{1.2}$$

where:

RRW = is the integration of the white noise. Qualitatively it shows a variance in the change of the bias. Thus, is one of the objectives to solve in gyroscopes. In order to solve bias time rates of changes, the solution is to apply a determination algorithm computien the Allan variance and other features. [4].

ARW = involves the white noise

The calibration of MEMS gysoscope is performed by computing the Allan variance algorithm[5]. Allance variance consist on computing the acomulated angular velocity sensor output and apply it into designed variance formulas.

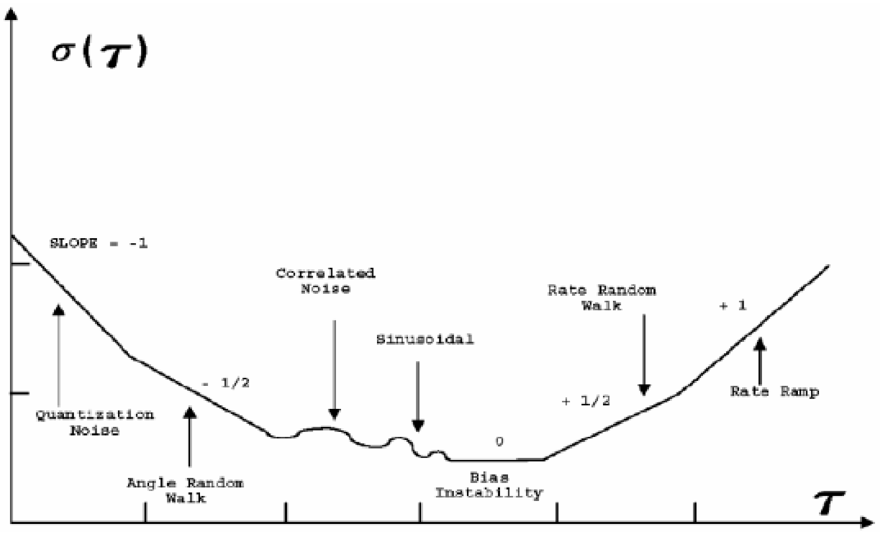


Figure 1.5: . Different Noise processes on an Allan Variance Plot [6]

Figure 1.5 shows different noise porcesses for allan Variance(Y-axis) in function of the average time of the sensor readings $\tau = m\tau_0$ where m are the samples and τ_0 the time between 2 consecutive samples.
Equation 1.1 is the gyroscope model that must to be fullfilled with the correct data. Therefore, calibration is done in this expresion in order to perform accurate data measurements.

Some parameters are automatically computed by the gyroscope during calibation, others must be measured and given to the sensor externally.

The idea is to change or utterly substract known parameters given the sensor readings $\overline{\omega_r}$

$[\overline{\overline{SP}}]$ = is computed by the sensors during the integration phase

$\overline{\omega_{bias}}$ = is measured when the sensor is not moving. the sensor outout can be modeled as $y(t) = \omega(t) + b(t) + n(t)$ where $\omega(t)$is teh true rate signal, b(t) teh sensor bias and n(t) the observation noise. When the sensor is static $\overline{\omega_{body}}$, the measurments consist in the bias and the noise. In order to reduce noise, the measurement can be averaged. However, data averages is linked with Allan variance so the maximun in signal averaging is the minimun Allan variance effect.

$[\overline{\overline{SF}}]$ = scaling factor and non-orthogonally corrections is the most importatn value to estimate from the user point of view. It is stimated for each axis by rotating the sensor in 3 orthogonal states. The purpose is to substract the noise and bias when the integration matrix is constant and known. Therefore, if the angular velocity is known $\overline{\omega_{body}}$

$$\frac{\overline{\omega}_r}{\overline{\omega}_{body}} = \overline{\overline{SF}} \tag{1.3}$$

$$[\frac{\overline{\omega}_{r_x}}{\overline{\omega}_{bodyx}} \frac{\overline{\omega}_{r_y}}{\overline{\omega}_{bodyy}} \frac{\overline{\omega}_{r_z}}{\overline{\omega}_{bodyz}}] = [\overline{\overline{SF_x}}\,\overline{\overline{SF_y}}\,\overline{\overline{SF_z}}] \tag{1.4}$$

Therefore, the angular tracking can be used in order to know the desired constant angular velocity of the gyroscopes in order to calibrate. The nanosatellite can be placed at the Helmholtz's disk and rotated witht he air bearing producing almos a constant angular velocity. The angular tracking algorithm can measure the external angular velocity of the rotating nanosatellite and averaging the value the user will know the actual angular velocity since the angular tracking operates at real time with all its features.

1.3.3. Magnetorquers

The actuator depends on the measurements of the sensors measurements that is applied to determination algorithm in order to estimate the actuators responses. Therefore, it applies the necessary moment to change the orientation of the nanosatellite based on the Earth's magnetic fields measurements.

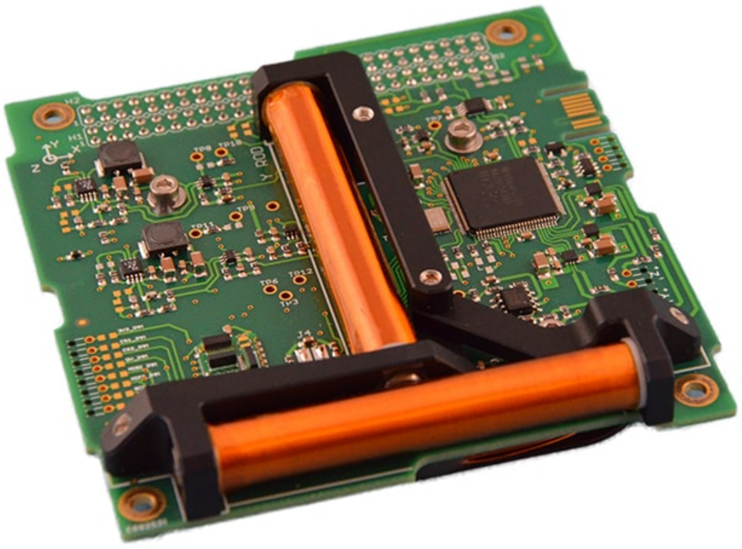


Figure 1.6: Magnetorquers for nanosatellite porpuses [3]

The total dipole moment can be computed as:

$$\vec{M} = \mu_r I \sum_{i=1}^{N} A_i \hat{n} \tag{1.5}$$

Where μ_r is the permeability of the magnetic core, I the current along the winding coil and the sumatyory denotes the effective area vector that represents the sum of enclosed areas [27].

$$\vec{\tau} = \vec{M} x \vec{B} \tag{1.6}$$

1.3.4. ADCS testing

Nanosatellite Testings results could not be provided since the testing campaign delay. This work will provide the helpful tools done when testing campaigns starts. The 2 testing procedures which will be involved in the the work done are Helmholtz coils related procedures and gyroscopes. In general, testing proceds with the set up required components and test facilities required for test realization:

- AOCS board
- Computer with Linux configured for connecting wirelessly to Raspberry Pi 3B through SSH and Matlab installed with the AOCS_test Matlab software prepared.
- Raspberry 3B configured for connecting to same Wi-Fi network than the computer.
- 5V battery.
- Helmholtz coils system
- External calibrated magnetometer
- File with the magnetic field simulated for the satellite's TLE

It proceed seting 45nT to the Helmholtz coils and then Raspberry starts saving data from sensors on the AOCS board in order to compared with the magnetic fields generated by the coils.
Then some signs in the coils are done.

The test will be considered successful when the measured magnetic field has an error of less than ±10

Detumbling test
After the CubeSat is ejected from the P-POD, the CubeSat may be affected initially by uncontrolled rotations. In particular, an initial rotation of 30º/s has been considered as worst case in each of the axes. The objective of the Detumbling Mode is to reduce this initial angular velocity to below 2º/s on each of the axes by using magnetic control in less than 24 hours.

Therefore, the tracking algorith can help alot in the determination of detumbling condition at 30º/s. Generally, detumbling uses Bdot algorithm to control the unstability generated.

Emplying the time derivative of the Earth's magnetic field measured by magnetorquers, Bdot generates a torque by the magnetorquers that try to stabilize the rotation. The magnetic torque is given by the cross product between coil magnetic moment and Earth's magnetic field.

$$g_m = m_{mag} \times B \tag{1.7}$$

Magnetic moment vector can be obtained

$$g_m = -k_{bdot} \frac{\dot{B}}{|B| \left|\dot{B}\right|} \tag{1.8}$$

In general, the time derivative of a vector in a rotating frame can be expresed as follows:

$$\dot{B}|_i = \dot{B}|_b + \omega \times B \tag{1.9}$$

Where i and b means inertial and body frames. Finally, it results after substituing Eqs. 1.8 and 1.9 into 1.7

$$g_m = \frac{k_{bdot}}{|B| \left|\dot{B}\right|} (\omega \times B) \times B) \tag{1.10}$$

The magnetic torque is opposed to the angular rate component perpendicular to the B magnetic field. Thus it will reduce the angular component.

The correct tracking of the Bdot can be now studied using the new GUI and angular tracking algorithm. By starting the coils at a Earth's constant magnetic field and tracking the angular velocities obtained while applying the Bdot algorithm in the satellite. After the simulation, both measurements can be analysed.

Gyroscopes test
Gyroscopes output is necessary for determining and controlling the attitude quaternion. Thus, this test tries to verify the correct gyro performances. All the required components and test facilities required for test: - AOCS board

- Computer with Linux configured for connecting wirelessly to Raspberry Pi 3B through SSH and Matlab installed with the AOCS_test Matlab software prepared.
- Raspberry 3B configured for connecting to same Wi-Fi network than the computer.
- 5V battery.
- Gyro table.
- USB Type A/B cable

The test will be considered successful if the measured angular rate differs from the expected angular rate within a margin of ±10

The new tracking algorithm is very important for this test since could be the substitute of teh gyro table for any give input velocity. This means that the test can be performed at any velocity inside the Helmholtz coils disk since the angular verlocity can be well known.

1.4. Helmholtz Coils

The aim of this magnetic field generator is to create artificial magnetic fields with order of magnitude close to the real space missions. Thus, Helmholtz coils must be able to generate around 100 nT with stability and control. Furthermore, the center of the coils should be stable enough to consider constant magnetic fields along all the nanosatellite. In this section, the design of the current Helmholtz coil is analized and also it is explained the reason why the structure must have the physical properties with which the coils were built.

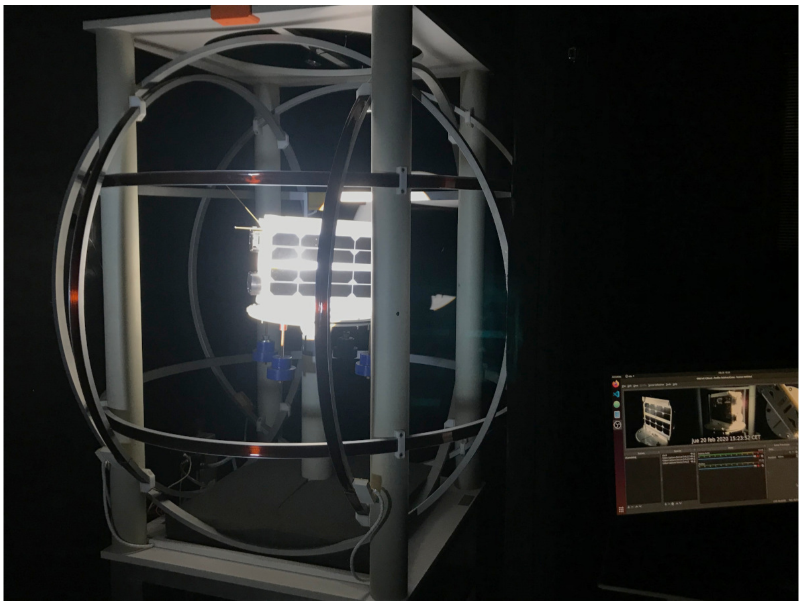


Figure 1.7: Helmholtz coils in NanoSatLab

Inside the coils, the structure contains the air bearing tube which will allow the disk to sustain in the air close to frictionless conditions. The coils are controled via Linux using 2 ports connected to one Arduino and th eother to the powerSupply. The disk contains some weights to stabilize the platform while doing the tests and usually the bottom is covered with some web in order to minimize undesirable risks.

1.4.1. Structure

NanoSatellites sensors must detect correctly 3-Axis magnetic fields. Thus, 3 pairs of orthogonal coils must be able to generate controlled magnetic fields in each axis.

Three axis 1300 mm Helmholtz Coils - Ferronato BH1300-3-A (serviciencia.es)

The structure of the coils is the following:

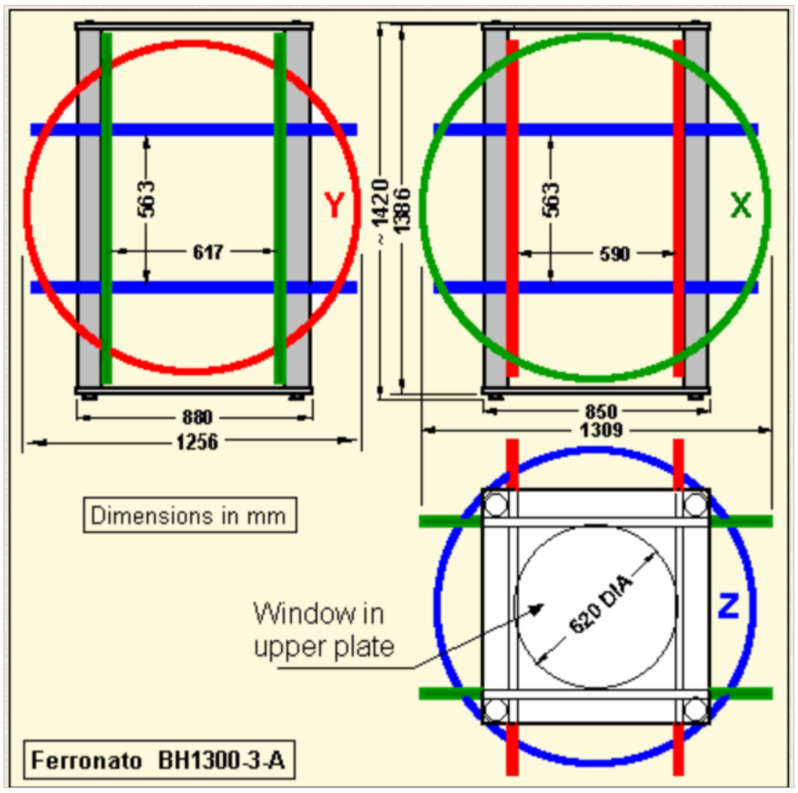


Figure 1.8: Helmholtz structure [23]

Notice how dimesions in each axis slightly diverge. However, in order to be able to have the same Magnetic field / Current ratio in each Axis the number of turns in each pair of coil also is different. BH1300-3-A SET has a ratio of 200 µT/A (2.0 Gauss/A) [23]. For each pair, X, Y or Z. The maximum error stabilized is : ±1

Optionally another ratio can be set if the connection is changed: 400 µT/A, or 2 x 100 µT/A, by modifying the wiring at the terminal block. However, for simplicity, we have worked with 200 µT/A . The maximum magnetic field allowed is 800 µT(8 Gauss) in steady way or 2.0 mT (20 Gauss) during 2 minutes at maximum for each axis.

The circuit for 1 standard Helmholz coil would look like the one proposed:

A power supply connected into the entrance of one coil and the exit of the opposite coil. Although this may look so simple, our coils have 2 more virtual coil in each pair of coils. The aluminium structure allows to generate magnetic fields as well. So in order to take benefit of the built structure, the coils are connected electrically to the looped aluminium structure in each axis. Thus, each aluminium loop constitutes a coil of one turn. This phenomenon allows to generate small controlled magnetic fields.

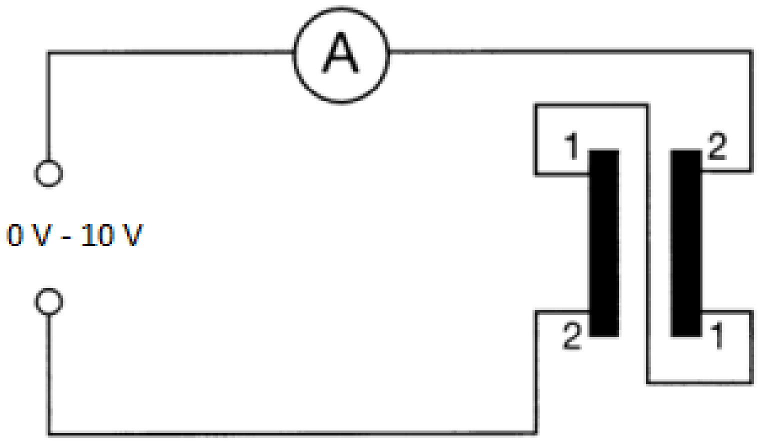


Figure 1.9: Helmholtz circuit design[**?**]

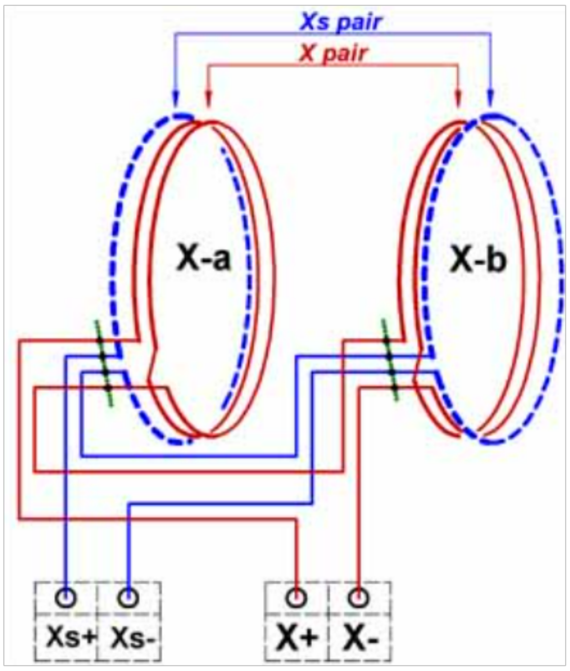


Figure 1.10: Helmholtz circuit structure [23]

Generally, X+ and X- are the inputs of the circuit. See figure 1.13. Althought it is simpler to connect directly the power supply to the coils using X+ and X-, Helmholtz coil BH1300 familly depict 2 winding coil in each pair. For instance, X+ is formed by X1+ and X2+, and X- is formed by X1- and X2. See 1.11. All the connections are made in the preassure control site in which the air at high preasurre passes by measuring the preassure just before the air bearing tube.

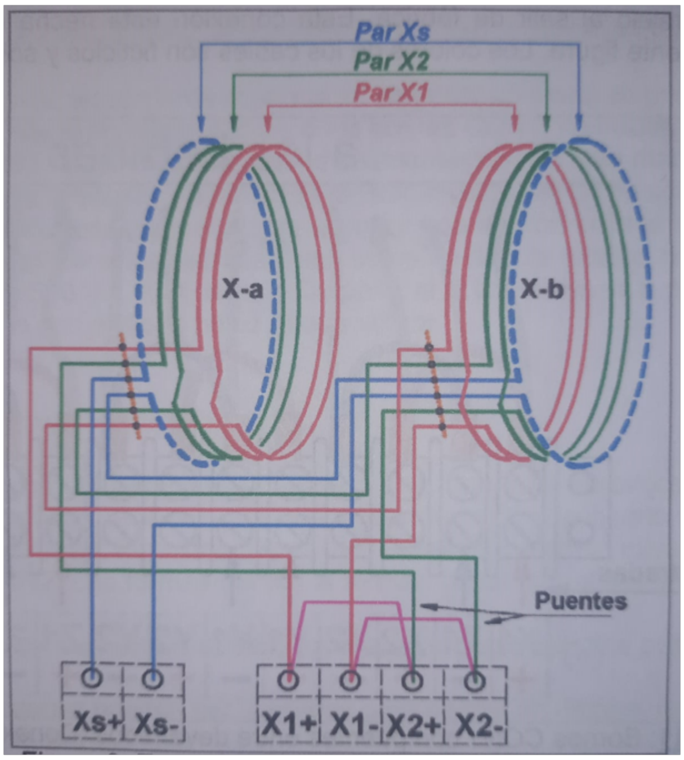


Figure 1.11: Helmholtz circuit structure with bridge

Furthermore, by default, the coils conection have a bridge from X1 to X2 in each sign. Notice each axis has the 2 conections of the aluminium loop and the 4 conection of the winding coils.

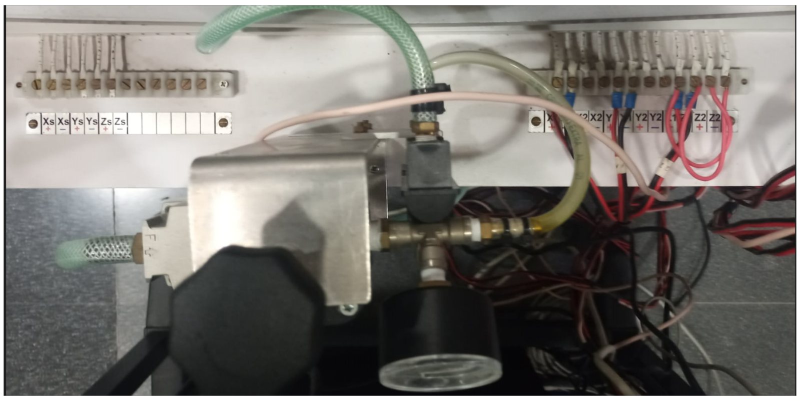


Figure 1.12: Helmholtz circuit structure with bridge

zooming up in the current Helmholtz conection we can see that Z-axis is connected using the bridge

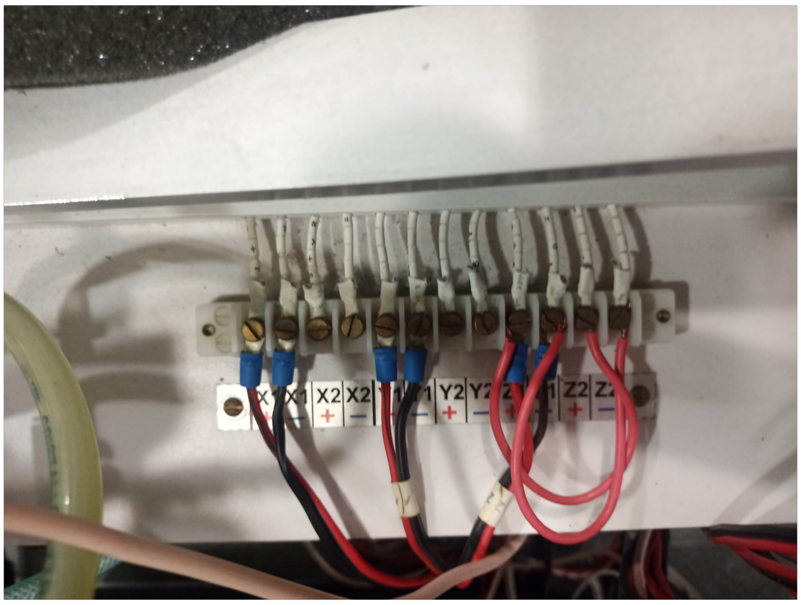


Figure 1.13: Helmholtz circuit structure with bridge in Z-axis

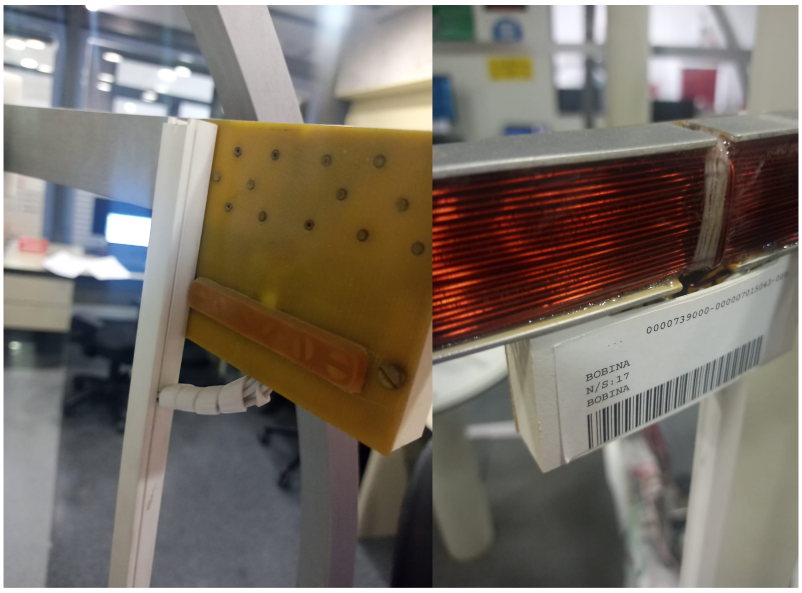


Figure 1.14: Front and rear side of the aluminium loop connection

Figure 1.14 shows how Helmholtz coils connect the structure to the winding generating one more loop at each axis.

1.4.2. Magnetic field generations

The problem can be simplified by considering a single circular loop coil with a constant current value I. Then, the problem consist in calculate the magnetic field at alomng the center of the circular llop [23].

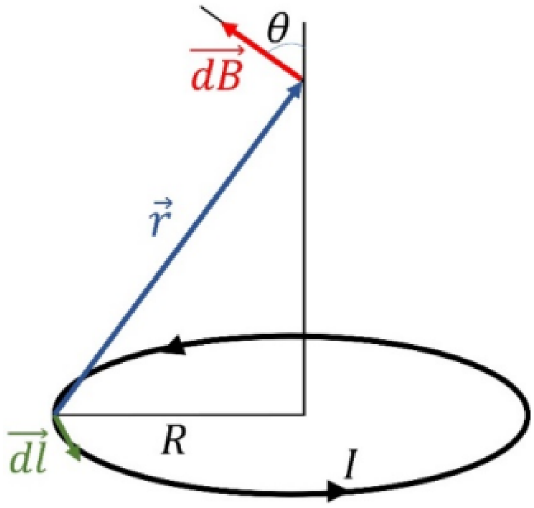


Figure 1.15: Helmholtz circuit structure with bridge in Z-axis [25]

From Maxwell equation we can obtain a more practical expression in order to relate the current in a coil with the magnetic field that generates throughout the space:

$$\oint_K \vec{H}\,\vec{ds} = I + \iint_F \vec{D}\,d\vec{f}\,dt \tag{1.11}$$

Where K denotes the close curve around the surface F. Thus, the first derivative of the direct current D is null.

$$\oint_K \vec{H}\,\vec{ds} = I \tag{1.12}$$

Which is usually written in form of Biot-Savart's law

$$\vec{dB} = \frac{\mu_o}{4\pi}\frac{\vec{dl}\times\vec{r}}{r^3} \tag{1.13}$$

where $\mu_o = 4\pi 10^{-}7$ Tm/A
The practical purpose of using circular structures is to be able to generate a magnetic field along one single axis. Notice how, by symmetry, horizontal pairs of the magnetic field swill cancel each other. Therefore, with 3 pairs of orthogonal coils, any space magnetic field vector can be created by adding the 3 designed components of the magnetic field generated. Other coil structures along a non astrometric wire will not apply for our purpose. For instance, structures such as dipoles, parabolas or superposition of those as yagis wont be able to create our constant magnetic fields. Due to the non constant directivity of the magnetic fields along one single axis.

From the definition of the cross product: $\vec{dl} \times \vec{r} = |dl||r|sin\theta$

$$\vec{dB} = \frac{\mu_o}{4\pi} I \frac{dl sin\theta \vec{k}}{r(R^2+z^2)} \tag{1.14}$$

applying the pitagorean theorem in the denominator and knowing that dlRresults in $2\pi R^2$ from the circle definition since sin$\theta = R/r$

For two coils with N turns each separated a distance h with the origin centered at one coil. By superposition of the magnetic fields we obtain the following expresion:

$$\vec{Bz} = \frac{\mu_o NIR^2\vec{k}}{2(R^2+z^2)^{\frac{3}{2}}} + \frac{\mu_o NIR^2\vec{k}}{2(R^2+(h-z)^2)^{\frac{3}{2}}} \tag{1.15}$$

If the maximum of the fuction is analized, the magnetic field must be almost constant near the center of the helmholtz coils for each axis. This can be done by finding the partial derivative with respect to the x,y, or z axis and finding its minimun. Resulting in the Helmholtz magnetic field generation expresion.

$$B_z = \frac{8\mu_0 NI}{R125^{0.5}} \tag{1.16}$$

In order to analize how close is the analitical expresion of the Helmholtz coils with the actual code. It has been compared the values that the power supply gives for a given input values. See appendix B

Bmeasured = [5 5 5; 10 10 10; 15 15 15; 20 20 20; 25 25 25; 30 30 30; 35 35 35; 40 40 40; 45 45 45; 50 50 50; 55 55 55; 60 60 60; 65 65 65; 70 70 70; 75 75 75;]; nT and the currents are compared with the theoretical current values.

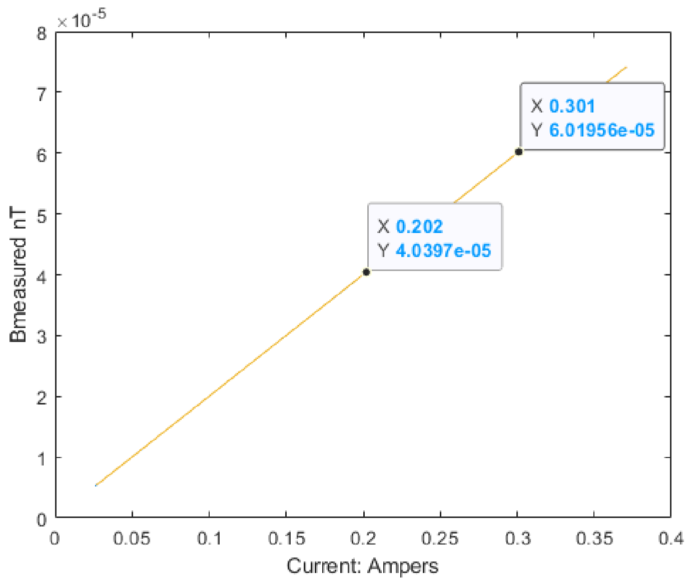


Figure 1.16: Slope of the currents vs Magnetic fields

Indeed, the relationship that the power supply must have is the one represented by this slope graph. By taking 2 consecutive point the slope can be computed. This must be the one that the coils especifications contains.

$$slope = \frac{6.01956e-05-4.0397e-05}{0.301-0.202} = 200nT/A \tag{1.17}$$

This is the computation that allows the code to operate the coils autonomously. Thus, instead of entering each magnetic field at a time, the code is prepared to compute the relationship and communicate directly with thte powersupply the current computed by this relationship. Although, this is a value that often the manufacturer gives to the customer, it is important to know from where does it comes. For instance, although it is not so natural, if the structure is modified by adding winding turns or by changing coils space. this could be taken into account by recomputing the expresion with the new changed values as done before.

By computing the relative error of the estimation analized, the precision of the coils can be estimated.

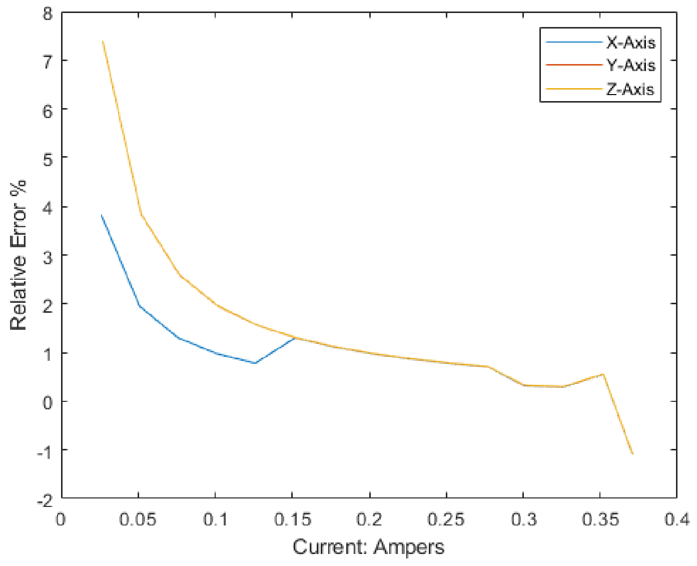


Figure 1.17: Analysis of the theoretical precision

Figure 1.17 shows that the relative error of the formula with the current code measures is lower when the currents goes aproximately from 0.15 to 0.3 Ampers. Which means that the Helmholtz coils are more precise with the theorical calculations when the magnetic fields genrated are close to 30 to 60 nT . Indeed, when doing the nanosatellite ADCS tests, the magnetic fields generated are often close to 50nT . This, shows how the Helmholtz coils are not so stable when the magnetic fields are higher. Other structures must be proposed in order to generate constant higher magnetic fields.

Applying to the formula 0.026 Ampers at each axis, it can be seen that since each axis has differetn dimensions, only the x-Axis reaches 5 nT at the center. However, the other axis have reacher different magnetic fields at the center of the theoretical Helmholtz coils.

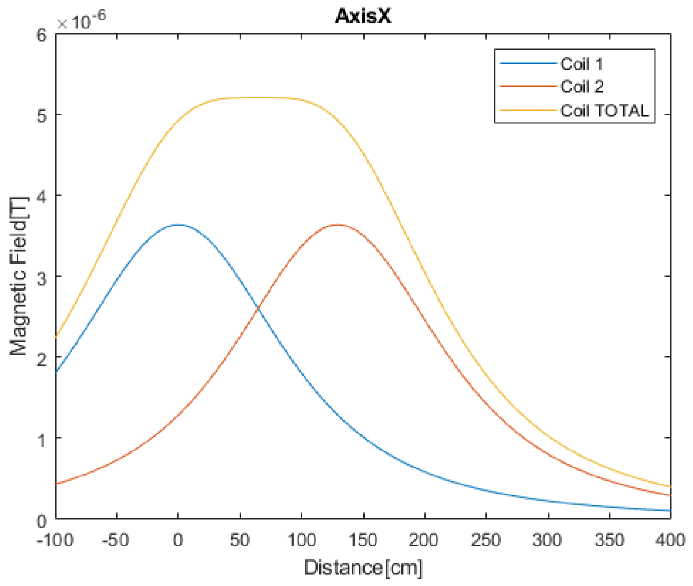


Figure 1.18: Bx magnetic field generated

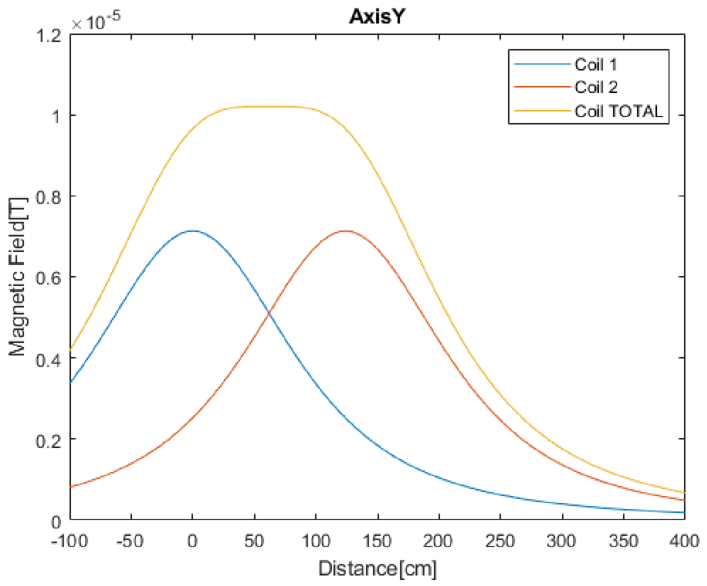


Figure 1.19: By magnetic field generated

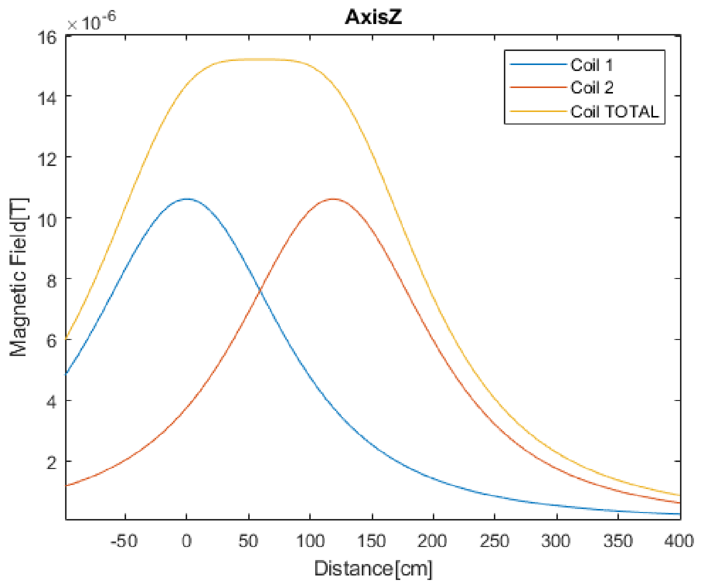


Figure 1.20: Bz magnetic field generated

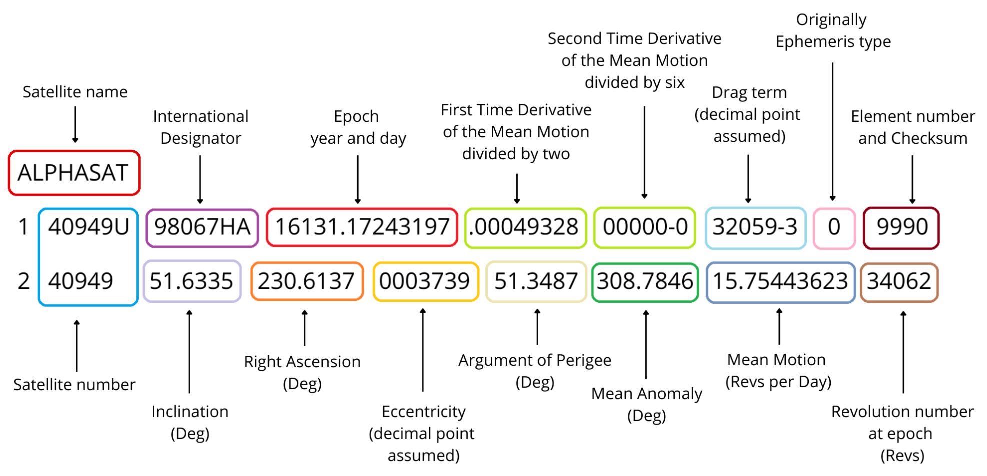
Notice that each axis is not centered at the physical Helmholtz coils center. The zero is located at the center of one coil. However, the constant magnetic field can be easily observed along the maximum. The 2 coils of each axis would represent the position where the two separated coils have their local maximum without any superposition. This proves that Helmholtz coils can be used easily for nanosatellite testings since the constant magnetic field is much higher than the nanosatellite dimesions close to 100mm at each axis. So the satellite will receive constant magnetic field along all the structure and the magnetic fields magnitude are correct compared with real orbit magnitudes.

1.5. Orbit Propagators

The aim of this section is not to explain the complete functionality of the SGPD4 propagator or IGRF or other version. The detailed explanation for each algorithm can be found at [10]. However, a qualitative explanation of how an orbit propagator works will be described. Some reference frames will be explained although the choice of each one depends on the mission.

1.5.1. SGDP4 explanation and some necessary coordinate transformation explanations.

An orbit propagator will be used as a black box with two input and 1 output. The input is the so called Two line elements TLE which contains the orbit information the other input is the propagation time which will define the time starting point and the end of the simulation.



Satellite name
International Designator
Epoch year and day
First Time Derivative of the Mean Motion divided by two
Second Time Derivative of the Mean Motion divided by six
Drag term (decimal point assumed)
Originally Ephemeris type
Element number and Checksum
ALPHASAT
1
40949U
98067HA
16131.17243197
.00049328
00000-0
32059-3
0
9990
2
40949
51.6335
230.6137
0003739
51.3487
308.7846
15.75443623
34062
Satellite number
Inclination (Deg)
Right Ascension (Deg)
Eccentricity (decimal point assumed)
Argument of Perigee (Deg)
Mean Anomaly (Deg)
Mean Motion (Revs per Day)
Revolution number at epoch (Revs)

Figure 1.21: Two Line Elements example

Figure 1.21 shows an example of ALPHASAT satellite two lines element TLE. This information is enough in order to define the orbit and propagate the satellite along the given orbit.

The output is a matrix containing the coordinates of the object propagated usually in eci or ecef coordinates reference frames. Some considerations that the newest algorithm have are the following: the spherical gravitational field of the Earth, the Luni-Solar third body gravitational perturbations, the Solid Earth tides among the mass forces, atmospheric drag and solar radiation pressure (SRP) among the surface forces and thrust forces [33].

ECI: Earth-centered inertial frame

The Earth-centered inertial, or the ECI frame, is a non-rotational frame which has its origin in the center of the Earth. This referece frame allow to have nice target following for object located at the equator. Thus, each sidereal day will share ECI coordinates. The z vector has the same direction as the Earth rotation axis, and it points towards the North Pole(NP). xECI vector points towards the vernal equinox direction. Which can considered a static point. Finally, the y vector I completes the orthogonal triad, following the righ-hand rule. Points located in the north pole or south pole have the same ECI coordinates along the time.

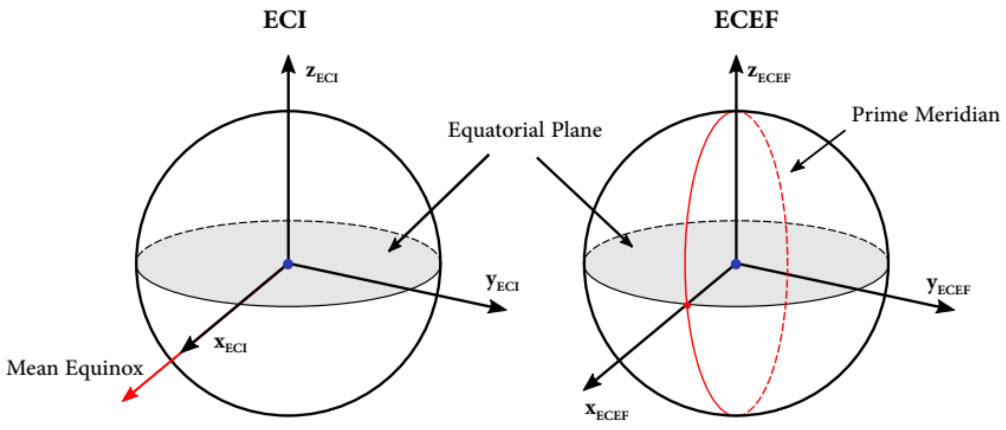


Figure 1.22: ECI and ECEF reference frames [28]

ECEF: Earth-centered fixed frame

The Earth-centered fixed frame, or the ECEF frame has the same origin than the ECI frame and also the same z vector pointing to the North Pole. However, xECEF points towards the 0○ latitude 0○ longitude Earth surface point.In other words, x ECEF point to the intersection of the Equatorial and the Meridian Planes in the positive side. Again, the yECEF completes the orthogonal triad. ECEF reference frame is not an inertial frame. Therefore, it rotates with the Earth around the zECEF with angular speed $\omega ECEF = 7.2921e{-}5$ rad/s.Constant points located at the earth will have constant coordinates while using ECEF reference frame. Geoestationary satellites will offer constant ecef coordinates.

Rotation Between ECI and ECEF

The rotation from ECI to ECEF is given by the rotation of the Earth. The e z-axis is rotating around the i z with a period time of a sidereal day denoted by td. This rotation is described by a direction cosine matrix with a rotation around the z-axis.

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where R is the rotation matrix from ECI to ECEF and $\theta(t)$ is the angle between the ECI and ECEF and is given as:

$$\theta(t) = \frac{2\pi t}{t_d} \tag{1.18}$$

Where t is the time since last alignment with the ECI frame.

Other Frames: Orbit Frame:

Z-axis points to the Erath's center and its origin is located at the satellite center of mass CoM. X-axis points to the orbit tangent and y-axis is orthogonal to the orbit plane.

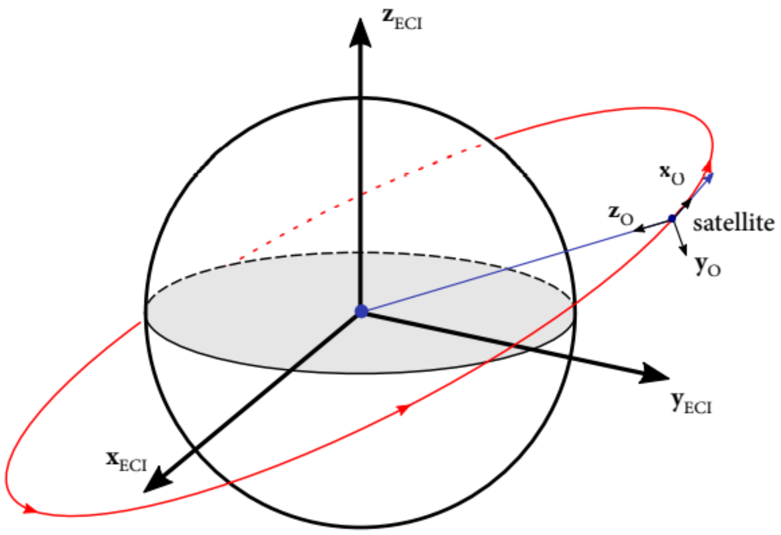


Figure 1.23: Orbit frame [28]

Target frame:

This frame is usally used for tracking earth targets of interest. Z-Axis point directly to the desired tracking point and x-axis point to the velocity vector in orbit.

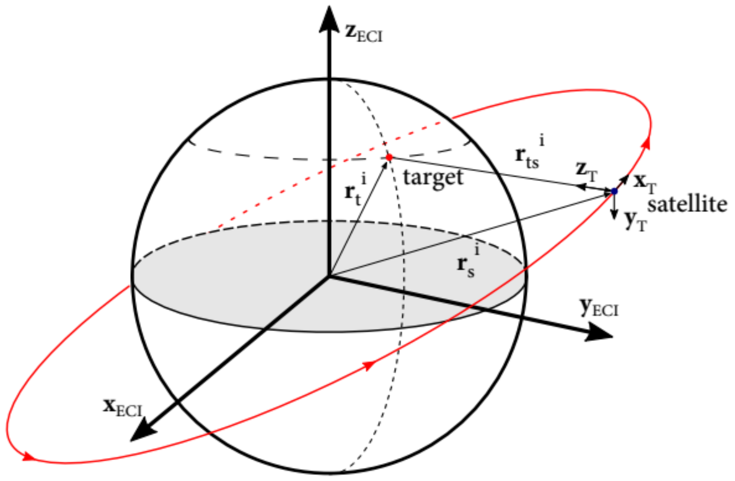


Figure 1.24: Target frame [28]

Body frame: This frame is similar to the target frame but with a constant coordinate frame located at the center of mass(CoM) of the satellite and with a designad orientation in order to rotate with the satellite. Usually is used for control and orientation purposes.

1.5.2. IGRF magnetic fields simulator. Explanation and some necessary explanations.

The International Geomagnetic Reference Field (IGRF) is a series of mathematical models of the Earth's main field and its annual rate of change (secular variation). IGRF computes the magnetic field intensity in 3 dimensions given a spacial coordinate usually expresed as latitude, longitude and altitude(above MSL). It consist in a set of Gauss coefficients associated to solid spherical harmonic functions [31].

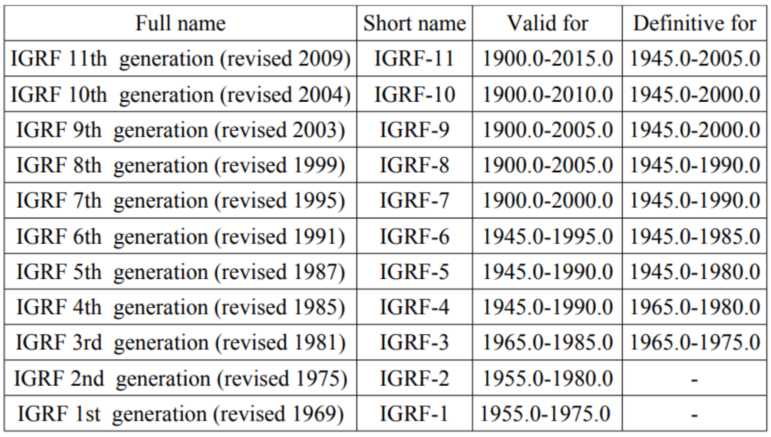
$$V(r,\theta,\varphi)=R\sum_{n=1}^{N}\left(\frac{R}{r}\right)^{n+1}\sum_{m=0}^{n}\left(g_n^m\cos(m\varphi)+h_n^m\sin(m\varphi)\right)P_n^m(\theta)$$

where r, θ, ϕ are geocentric coordinates according to a reference coordinate system. r is the distance from the centre of the Earth,θ is the colatitude (i.e. 90 ° - latitude), and ϕis the longitude , R is a reference radius (Earth mean radius) set to 6 371.2 km; g n m and h n are the Gauss internal coefficients and P n m(θ) are the Schmidt semi-normalised associated Legendre functions of degree n and order m.

Therefore the magnetic field can be computed using the gradient of the potential defined.

$$B = -\nabla V \tag{1.19}$$

Notice that Earth magnetic fields are time dependent. However, the model asumes time variation linear dependent up to time intervals of 5 years. Thus, the model must be updated each 5 years in the best case.



Full name	Short name	Valid for	Definitive for
IGRF 11th generation (revised 2009)	IGRF-11	1900.0-2015.0	1945.0-2005.0
IGRF 10th generation (revised 2004)	IGRF-10	1900.0-2010.0	1945.0-2000.0
IGRF 9th generation (revised 2003)	IGRF-9	1900.0-2005.0	1945.0-2000.0
IGRF 8th generation (revised 1999)	IGRF-8	1900.0-2005.0	1945.0-1990.0
IGRF 7th generation (revised 1995)	IGRF-7	1900.0-2000.0	1945.0-1990.0
IGRF 6th generation (revised 1991)	IGRF-6	1945.0-1995.0	1945.0-1985.0
IGRF 5th generation (revised 1987)	IGRF-5	1945.0-1990.0	1945.0-1980.0
IGRF 4th generation (revised 1985)	IGRF-4	1945.0-1990.0	1965.0-1980.0
IGRF 3rd generation (revised 1981)	IGRF-3	1965.0-1985.0	1965.0-1975.0
IGRF 2nd generation (revised 1975)	IGRF-2	1955.0-1980.0	-
IGRF 1st generation (revised 1969)	IGRF-1	1955.0-1975.0	-

Figure 1.25: IGRF Hystory [32]

Currently the model updated is the 13th generation. Each 5 years the IAGA Working Group- V Mod. Makes the decision of standarize the newest model.[12].

Overall procedure
given a TLE, the orbit information(TLE) is passed to the SGDP4 to obtain the coordinates in the time interval set. Notice that in order to plot the cooordinate into the earth map it is first mandatory to convert the cordenates from ecef to lla.

TLE:
1 41603U 16040E 18025.23966005 .00000144 00000-0 99713-5 0 9995
2 41603 97.4319 89.2031 0014438 26.9018 44.8694 15.19501926 88378 Then, the coor-

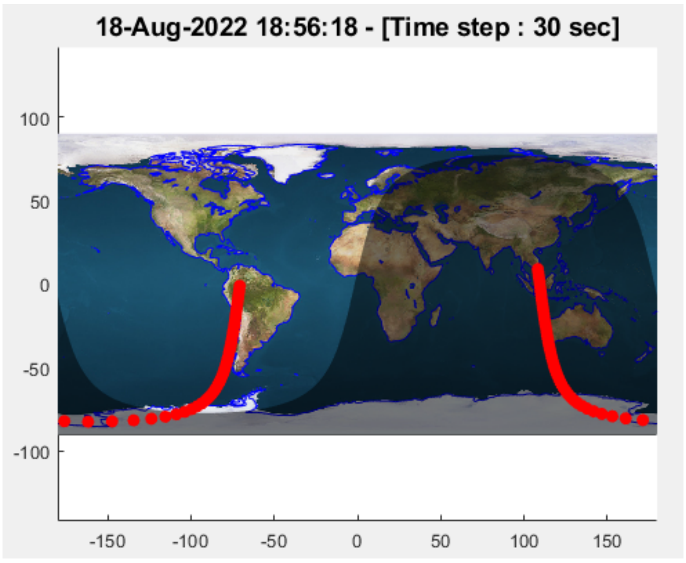


Figure 1.26: Propgataed orbit from the TLE

dinates are set to the IGRF function obtaining finally the 3 magnetic components at each propagated point in time.

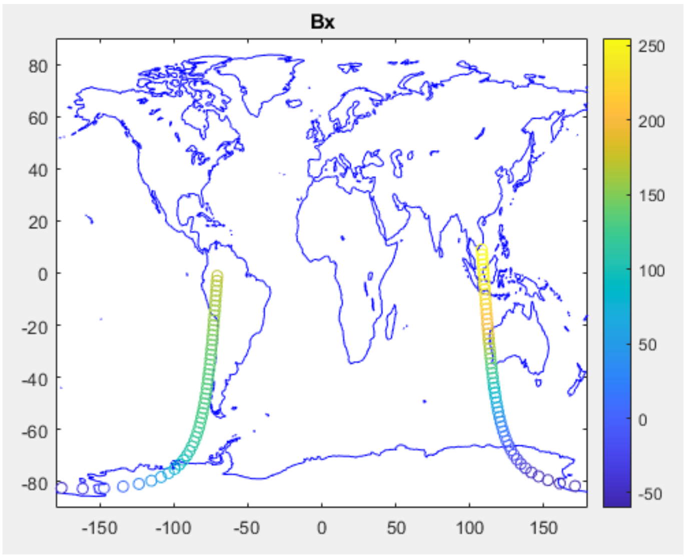


Figure 1.27: Propagated orbit from the TLE with IGRF magnetic fields X-axis

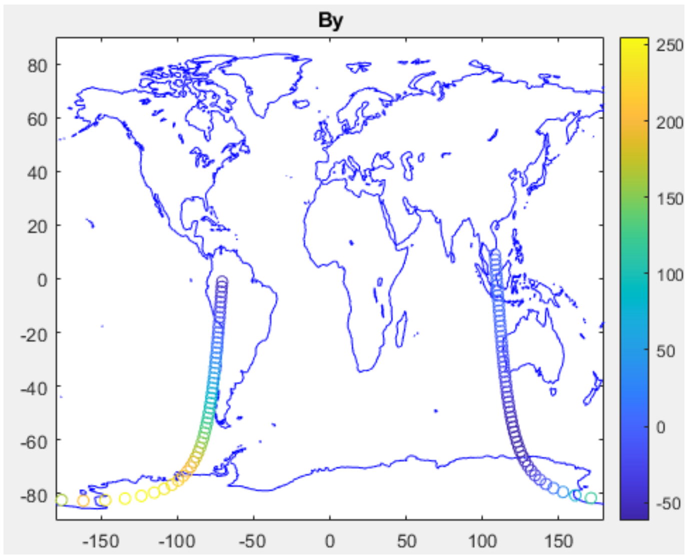


Figure 1.28: Propagated orbit from the TLE with IGRF magnetic fields Y-axis

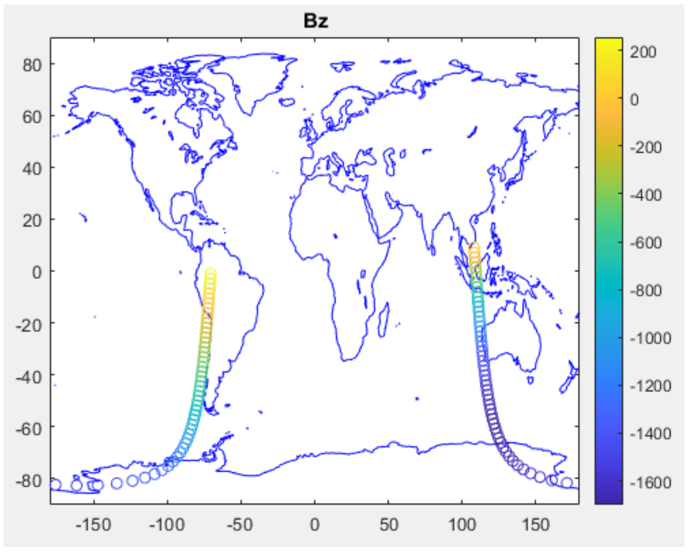


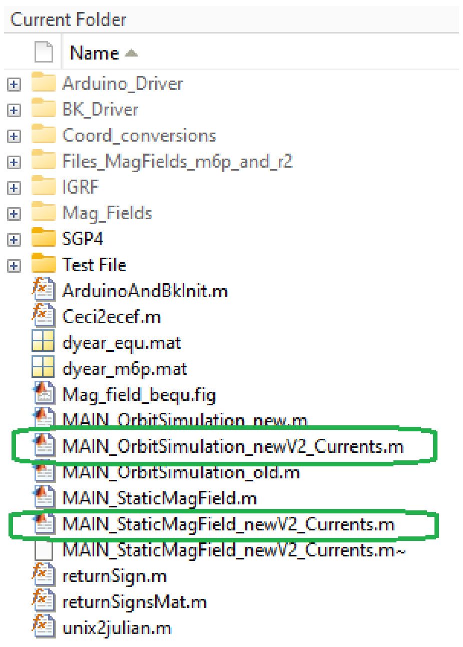
Figure 1.29: Propagated orbit from the TLE with IGRF magnetic fields Z-axis

Those magnetic fields would be loaded to a file with the format Bx;By;Bz ready to use in the Helmholtz coils software explained in the next section. IGRF can be checked at [30] for a give position online.

CHAPTER 2. SOFTWARE DESIGN

2.1. Current design

NanoSat Lab provides the code used to work with the Helmholtz coils written in Matlab. The code structure is organized as shown in Figure 2.1



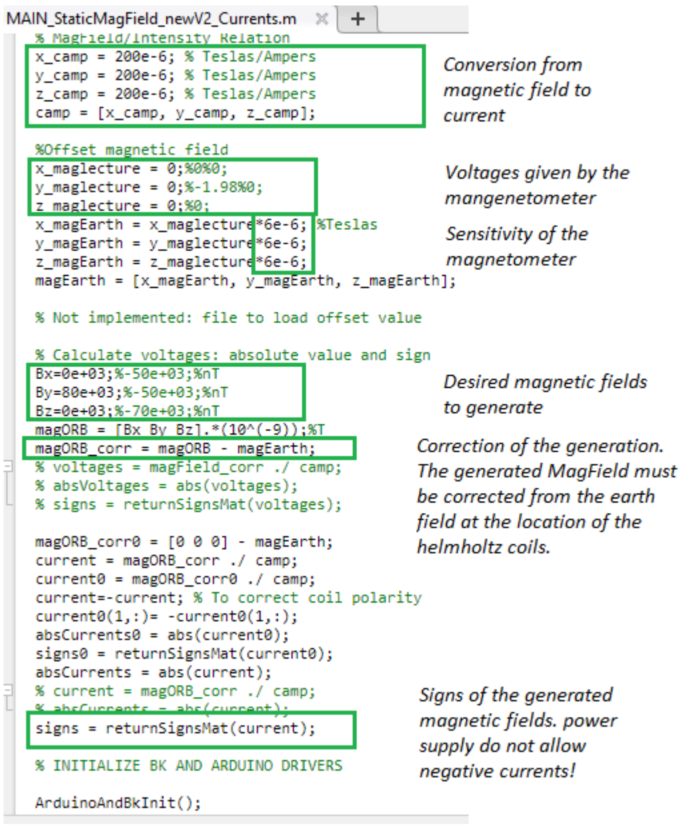
```
Current Folder
Name
Arduino_Driver
BK_Driver
Coord_conversions
Files_MagFields_m6p_and_r2
IGRF
Mag_Fields
SGP4
Test File
ArduinoAndBkInit.m
Ceci2ecef.m
dyear_equ.mat
dyear_m6p.mat
Mag_field_bequ.fig
MAIN_OrbitSimulation_new.m
MAIN_OrbitSimulation_newV2_Currents.m
MAIN_OrbitSimulation_old.m
MAIN_StaticMagField.m
MAIN_StaticMagField_newV2_Currents.m
MAIN_StaticMagField_newV2_Currents.m~
returnSign.m
returnSignsMat.m
unix2julian.m
```

Figure 2.1: Matlab code structure

Helmhotlz coils can be used in two different ways: either by generating a constant magnetic field or by dynamic one. MAINStaticMagFieldnewV2Currents and MAINOrbitSimulationnewV2Currents are the main codes that execute both processes respectivelly. Using the explanations of Helmholtz coil working principle section, the Matlab code can be better undertood in order to change all the structure to a new language. Therefore, with the use of teh power supply and the current switcher device correctly programmed. The new software must operate as the Matlab orignal code.

MAIN_StaticMagField_newV2_Currents:

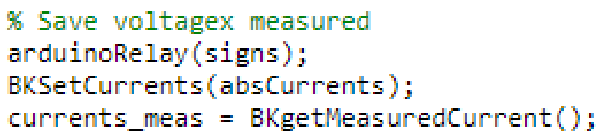
Magnetic fields generation by a winding coil can be studied both from the current or the voltage point of view. Therefore, the code program can set those magnetic fields communicating to the power supply the voltages or the currents. Usually, the manufacturer will provide the resistance value of each pair of coils. Using Hom's theory, it is possible to easily set the voltage that creates the desired magnetic fields within the dynamic range of our Helmhotz coils (max 10 volts). Previously, this had been done, but the magnetic field generation was not accurate enough for the nanosat tests. Section 1.4.2. shows how to find the conversion from magnetic field magnitude to currents. this conversion is the one used in Figure 2.2


MAIN_StaticMagField_newV2_Currents.m
Conversion from magnetic field to current
Voltages given by the mangenetometer
Sensitivity of the magnetometer
Desired magnetic fields to generate
Correction of the generation. The generated MagField must be corrected from the earth field at the location of the helmholtz coils.
Signs of the generated magnetic fields. power supply do not allow negative currents!

```
% MagField/Intensity Relation
x_camp = 200e-6; % Teslas/Ampers
y_camp = 200e-6; % Teslas/Ampers
z_camp = 200e-6; % Teslas/Ampers
camp = [x_camp, y_camp, z_camp];

%Offset magnetic field
x_maglecture = 0;%0%0;
y_maglecture = 0;%-1.98%0;
z_maglecture = 0;%0;
x_magEarth = x_maglecture*6e-6; %Teslas
y_magEarth = y_maglecture*6e-6;
z_magEarth = z_maglecture*6e-6;
magEarth = [x_magEarth, y_magEarth, z_magEarth];

% Not implemented: file to load offset value

% Calculate voltages: absolute value and sign
Bx=0e+03;%-50e+03;%nT
By=80e+03;%-50e+03;%nT
Bz=0e+03;%-70e+03;%nT
magORB = [Bx By Bz].*(10^(-9));%T
magORB_corr = magORB - magEarth;
% voltages = magField_corr ./ camp;
% absVoltages = abs(voltages);
% signs = returnSignsMat(voltages);

magORB_corr0 = [0 0 0] - magEarth;
current = magORB_corr ./ camp;
current0 = magORB_corr0 ./ camp;
current=-current; % To correct coil polarity
current0(1,:)= -current0(1,:);
absCurrents0 = abs(current0);
signs0 = returnSignsMat(current0);
absCurrents = abs(current);
% current = magORB_corr ./ camp;
% absCurrents = abs(current);
signs = returnSignsMat(current);

% INITIALIZE BK AND ARDUINO DRIVERS

ArduinoAndBkInit();
```

Figure 2.2: Static Orbit simulation overview code (Matlab)

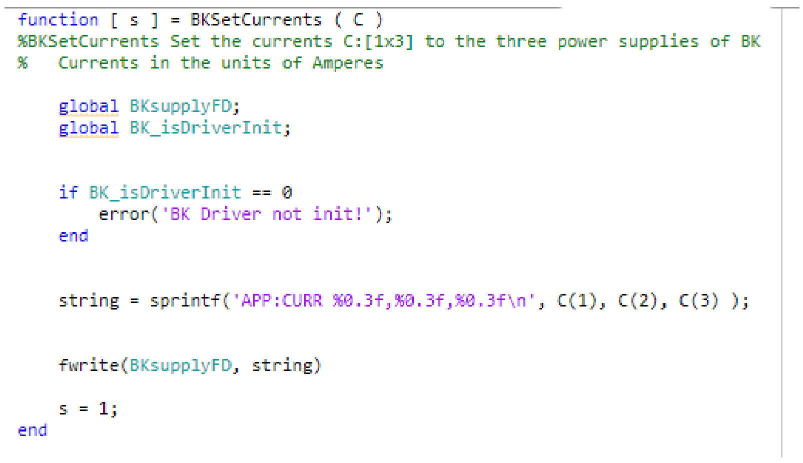
There is a static magnetic field generator which takes the 6 incoming inputs as Bx,By,Bz from the user as well as the voltages measured with the magnetometer placed at the center of the Helmholtz coils. In order to convert those voltages to currents, they need to be divided by the sensitivity of the sensor and then subtracted from the desired magnetic field. Thus, allowing the user to generate a magnetic field without the components of the magnetic field present due to the earth one. This situation may result in a negative current. However, the power supply will not allow negative currents, and the Helmholtz coil electric circuit should be set up in accordance with the section on Helmholtz structure. Based on the variable signs Arduino receives from the power supply connection, Arduino automatically changes the sign of the current. There is an output that is the desired magnetic field with its signs, but without the presence of the earth's magnetic field as well.



```
% Save voltagex measured
arduinoRelay(signs);
BKSetCurrents(absCurrents);
currents_meas = BKgetMeasuredCurrent();
```

Figure 2.3: Matlab usage of the serial communication

In conclusion, code shown in Figure 2.3 allows Matlab to communicate with the power supply and the Arduino to set the final signs and currents. Finally, the BKDriver and ArduinoDriver libraries enable the system to communicate with the hardware. Figure 2.4 is a detailed description of the most important functions in the libraries that can be found in the code section



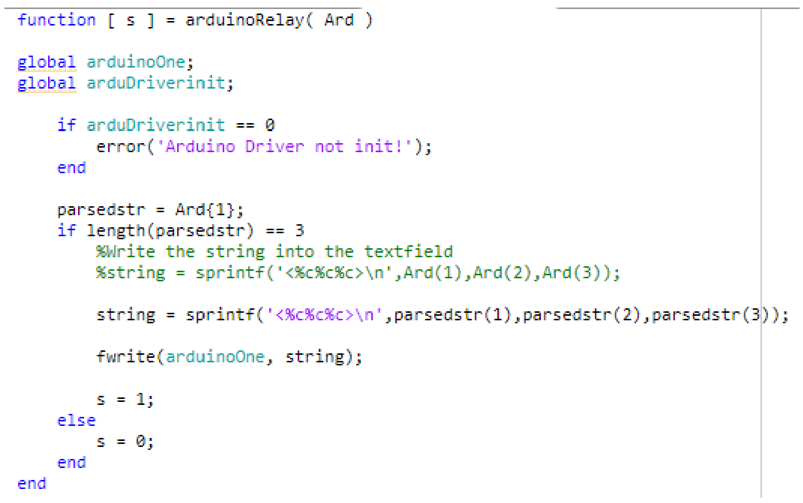
```
function [ s ] = BKSetCurrents ( C )
%BKSetCurrents Set the currents C:[1x3] to the three power supplies of BK
%   Currents in the units of Amperes

    global BKsupplyFD;
    global BK_isDriverInit;


    if BK_isDriverInit == 0
        error('BK Driver not init!');
    end


    string = sprintf('APP:CURR %0.3f,%0.3f,%0.3f\n', C(1), C(2), C(3) );


    fwrite(BKsupplyFD, string)

    s = 1;
end
```

Figure 2.4: Matlab send currents to powerSupply function

BKSetCurrents function sends the message to the powerSupply using the following text format:

`'APP:CURR %0.3f,%0.3f,%0.3f\n', C(1), C(2), C(3)`

It should be noted that 0.3 means that one 3 digit number will be sent after the comma. This is due to the fact that the power supply used only allows that resolution. APP:CURR message is used to set currents.



```
function [ s ] = arduinoRelay( Ard )

global arduinoOne;
global arduDriverinit;

    if arduDriverinit == 0
        error('Arduino Driver not init!');
    end

    parsedstr = Ard{1};
    if length(parsedstr) == 3
        %Write the string into the textfield
        %string = sprintf('<%c%c%c>\n',Ard(1),Ard(2),Ard(3));

        string = sprintf('<%c%c%c>\n',parsedstr(1),parsedstr(2),parsedstr(3));

        fwrite(arduinoOne, string);

        s = 1;
    else
        s = 0;
    end
end
```

Figure 2.5: Matlab function to send the signs

ArduinoRelay in Figure 2.5 sends the signs of the currents to the arduino module, which sets the sense of the currents using its relays.
The message format for this purpose is the following:

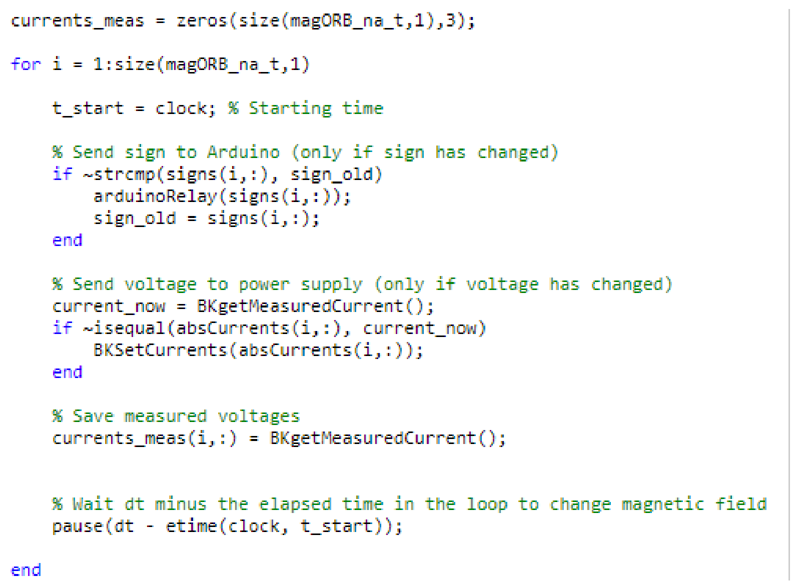
```
'<%c%c%c>\n',parsedstr(1),parsedstr(2),parsedstr(3)
```

the message must be sent as a char so in order to convert the string input Ard to char it uses $parsedstr = Ard1$ for instance the following signs +-+ would be converted into this.

```
"+-+" → '+-+' → '<+-+>'
```

MAIN_Orbit_Simulation_newV2Currents :

The dynamic code uses the same idea as the static explained above but changing the currents at a constant rate.



```
currents_meas = zeros(size(magORB_na_t,1),3);

for i = 1:size(magORB_na_t,1)

    t_start = clock; % Starting time

    % Send sign to Arduino (only if sign has changed)
    if ~strcmp(signs(i,:), sign_old)
        arduinoRelay(signs(i,:));
        sign_old = signs(i,:);
    end

    % Send voltage to power supply (only if voltage has changed)
    current_now = BKgetMeasuredCurrent();
    if ~isequal(absCurrents(i,:), current_now)
        BKSetCurrents(absCurrents(i,:));
    end

    % Save measured voltages
    currents_meas(i,:) = BKgetMeasuredCurrent();


    % Wait dt minus the elapsed time in the loop to change magnetic field
    pause(dt - etime(clock, t_start));

end
```

Figure 2.6: Dynamic orbit simulation loop

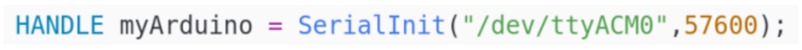
In the loop of Figure 2.6, the currents and the signs are set at each position of the matrices absCurrents and signs. The matrices are obtained by reading the files that contain the magnetic fields in three columns and processing each row individually as static code would. The code sends new signs only if they differ from the previous magnetic fields in the list in order to increase speed.

2.2. New design working Principle

2.2.1. From matlab to Code structure in C++.

The code of Helmholtz coils in C++ can be obtained at appendix A in which all the used classes and functions have been included. One of the most significant aspects of the project is the communication with both Arduino and powerSupply. Consequently, the objective of this section is to find a correct C++ structure code capable of communicating the processed inputs to the power supply and the signs to the Arduino. For this purpose, the class Serial will be created as Serial.cpp and Serial.h files, which will be invoked from the main script that will initiate the process.

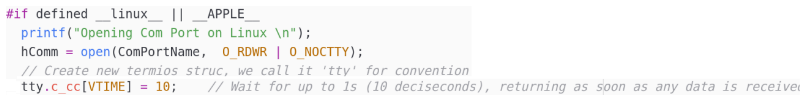
2.2.1.1. Arduino Serial



```
HANDLE myArduino = SerialInit("/dev/ttyACM0",57600);
```

Figure 2.7: Arduino communication structure

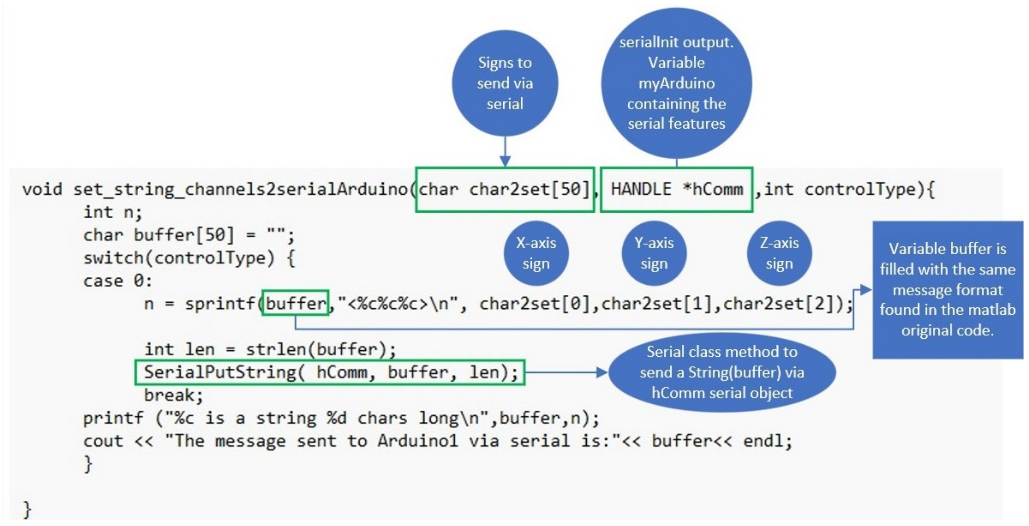
Within the Serial class, the SerialInit function is the main function. After processing the open() function in C++, the system communicates with the serial port through an object called hComm shown in Figure 2.8. Then, hComm object is edited for general serial communication purposes and to set control protocols. It is created a new termios structure [34] called 'tty' for convenience. tty.c code in Figure 2.8 makes the system wait for up to 1s (10 deciseconds), returning as soon as any data is received. Generaly those conditions are made to prevent any danger to the hardware.



```
#if defined __linux__ || __APPLE__
  printf("Opening Com Port on Linux \n");
  hComm = open(ComPortName,  O_RDWR | O_NOCTTY);
  // Create new termios struc, we call it 'tty' for convention
  tty.c_cc[VTIME] = 10;    // Wait for up to 1s (10 deciseconds), returning as soon as any data is receive
```

Figure 2.8: communication structures

Once the hComm object is filled with all the port condition, the serial class is ready to listen and write via serial port. The serial code works for Linux using the condition shown in Figure 2.8. Despite the fact that it contemplates working on Windows as well, for NanoSat Llab and general C++ purposes, it is more efficient to use the tty ports nomenclature for Linux systems. The main code has been designed to call Serial functions in a more readable manner. If the code needs to send the signals to Arduino in order to set the signs, the set_string_channels2serialArduino() function is used. The purpose of this function is to

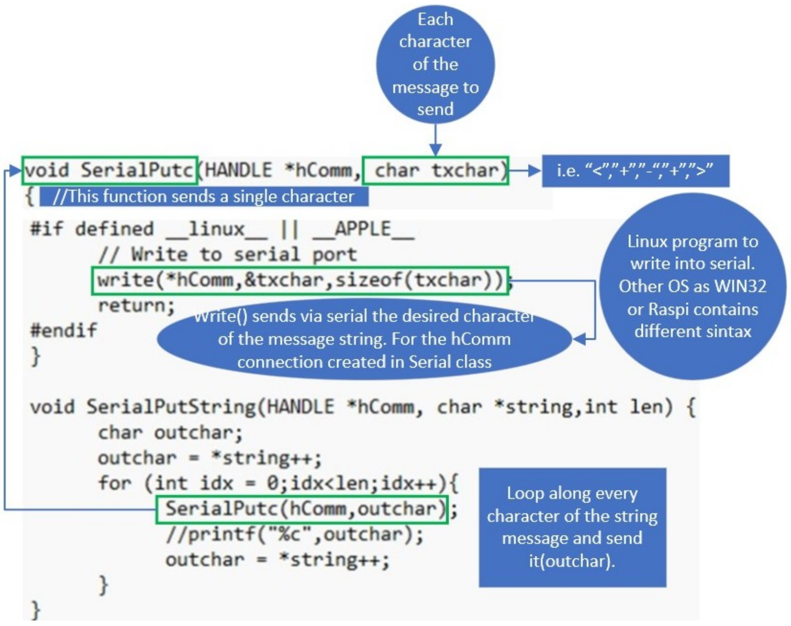


```
void set_string_channels2serialArduino(char char2set[50], HANDLE *hComm ,int controlType){
      int n;
      char buffer[50] = "";
      switch(controlType) {
      case 0:
            n = sprintf(buffer,"<%c%c%c>\n", char2set[0],char2set[1],char2set[2]);

            int len = strlen(buffer);
            SerialPutString( hComm, buffer, len);
            break;
      printf ("%c is a string %d chars long\n",buffer,n);
      cout << "The message sent to Arduino1 via serial is:"<< buffer<< endl;
      }

}
```

Signs to send via serial
serialInit output. Variable myArduino containing the serial features
X-axis sign
Y-axis sign
Z-axis sign
Variable buffer is filled with the same message format found in the matlab original code.
Serial class method to send a String(buffer) via hComm serial object

Figure 2.9: Fuction to send signs via serial

mimic the ArduinoRelay() function in Matlab. At some high level of coding, the function used is set_string_channels2serialArduino(), which utilizes a method of the Serial class, SerialPutString. A full string cannot, however, be sent directly to a serial channel in C++. In order to send the data in C++, the message must consist of only one character, which is the number of hexadecimal digits that represents the data.


Each character of the message to send
i.e. “<”,”+”,”-”,”+”,”>”
//This function sends a single character
Linux program to write into serial. Other OS as WIN32 or Raspi contains different sintax
write() sends via serial the desired character of the message string. For the hComm connection created in Serial class
Loop along every character of the string message and send it(outchar).

```
void SerialPutc(HANDLE *hComm, char txchar)
{
#if defined __linux__ || __APPLE__
      // Write to serial port
      write(*hComm,&txchar,sizeof(txchar));
      return;
#endif
}

void SerialPutString(HANDLE *hComm, char *string,int len) {
      char outchar;
      outchar = *string++;
      for (int idx = 0;idx<len;idx++){
            SerialPutc(hComm,outchar);
            //printf("%c",outchar);
            outchar = *string++;
      }
}
```

Figure 2.10: send characters via Serial fuctions

To summarize, the process to send the signs to Arduino uses the function set_string_channels2serialArduino() that is invoked by serialPutString(), which loops through each character of the message phrase and calls SerialPutC() at each character. This last function calls the function write(), which is an owner function in the C language.

2.2.1.2. Power supply communication

In order to set the current into the Helmholtz coils, the process is similar. It should be noted, however, that the Serial class does not work externally when called from the main code, as it did on the Arduino with the SerialInit() function. As a result, a piece of the Serial code was incorporated into the main code in order to obtain a similar hComm object with the desired communication features and protocols. int open_powerSupply_serial() function will create most of the features created inside the Serial class, for instance opening the port. It is necessary to write the use of the serial port RS 232 with the USBX syntax where

```
int myPowerSupply = open("/dev/ttyUSB0", O_RDWR);
```

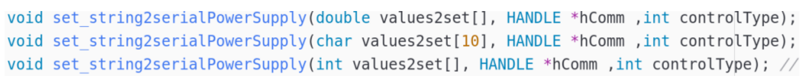
Figure 2.11: Open a serial object command

X starts at 0. it will be created a new termios struct called struct termios tty, and we will fill it with the properties of the communication serial port simillarly as it was done for Arduino communication in Figure 2.8.

tty.c_lflag &= ECHO; // Disable echo

tty.c_cc[VTIME] = 10;

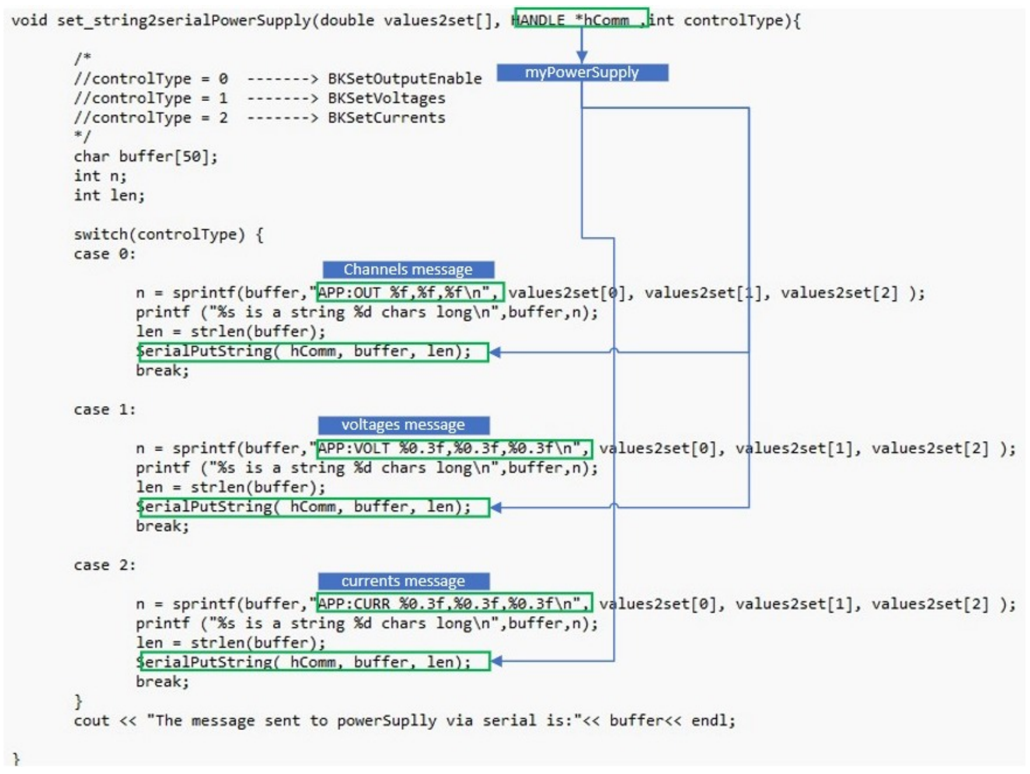
In addition, some other features are available on the Arduino serial port. As with the Arduino code, this communication is done directly in the main in order to resolve the problems encountered while implementing the communication using the complete Serial class. As soon as the myPowerSupply object has been filled, it can be used to send and receive data in the same manner as the previously described hComm communication object. String2serialPowerSupply() 2.13 mimics the functionality of BKSetCurrents from Matlab, and it is overloaded with voltages and currents in doubles, and channels in integers. Also chars has been considered if necessary.



```
void set_string2serialPowerSupply(double values2set[], HANDLE *hComm ,int controlType);
void set_string2serialPowerSupply(char values2set[10], HANDLE *hComm ,int controlType);
void set_string2serialPowerSupply(int values2set[], HANDLE *hComm ,int controlType); //
```

Figure 2.12: Serial used functions

BKsetOutputEnables([1 1 1]);
BKSetVoltages([10 10 5]);
BKSetCurrents([1 1 1]);%
Matlab implementation set channels, Voltages and currents to the power supply with BKSet library. Therefore Set_string2serialPowerSupply() must mimic the behaviour of its functions. When calling the function, controlType = 0,1,2 sets channels, voltages and currents



```
void set_string2serialPowerSupply(double values2set[], HANDLE *hComm ,int controlType){

        /*
        //controlType = 0  -------> BKSetOutputEnable
        //controlType = 1  -------> BKSetVoltages
        //controlType = 2  -------> BKSetCurrents
        */
        char buffer[50];
        int n;
        int len;

        switch(controlType) {
        case 0:

                n = sprintf(buffer,"APP:OUT %f,%f,%f\n", values2set[0], values2set[1], values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 1:

                n = sprintf(buffer,"APP:VOLT %0.3f,%0.3f,%0.3f\n", values2set[0], values2set[1], values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 2:

                n = sprintf(buffer,"APP:CURR %0.3f,%0.3f,%0.3f\n", values2set[0], values2set[1], values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;
        }
        cout << "The message sent to powerSuplly via serial is:"<< buffer<< endl;
}
```

Figure 2.13: Send currents via serial function

respectively. The message sent has the following format:

```
"APP:OUT %f,%f,%f\n"
"APP:VOLT %0.3f,%0.3f,%0.3f\n"
"APP:CURR %0.3f,%0.3f,%0.3f\n".
```

Notice the resolution of 3 digits for the currents and voltages

2.2.2. Software Architecture

2.2.2.1. Explanation of all the classes and the overview of the code.

During the conversion of the original matlab code to C++, the main challenge was that the program was developed in the form of a script without any interface other than the Matlab console interface. Thus, it is easier to verify that the code is functioning properly. Focusing on serial communication and parallel processing. As soon as all the code was able to start the Helmholtz coils, a GUI [37] design was developed in order to make the block usable. All objects and data structures related to the FLTK [36] interface are contained in main_v2.cpp. The processing block is contained in the Start.cpp class. This class contains both static and dynamic processing blocks. Serial.cpp is responsible for communicating both signals and currents with the Arduino and power supply. Finally, the File.cpp class can read and prepare the .txt file including magnetic fields in 3 axes.

Serial Class:

void InitSerialPort();
Use this if you want to use default ComPort and BaudRate

HANDLE SerialInit();
Use this if you want to specify ComPortName and BaudRate

char SerialGetc();
Get and Receive 1 character

void SerialPutc();
Get and send 1 character

void SerialPutString();
This sends a string rather than 1 character

*void SerialPutString(HANDLE *hComm, char *string,int len() ;*
This is an overloaded function that uses a for loop instead of a while loop

void SerialGetAll()
This routine reads everything from serial and prints it to screen. not just 1 character

void SerialGetAllPowerSupply(
This routine reads everything from serial powesupply and prints it to screen. not just 1 character

void SerialSendArray()
So this routine here sends an array in the 1 Hex format

It has some other functions but the most interesting and widely used in the code are the proposed above.

File class:

void openFile(char fileName[20]);
Open file given a name and its path if is not in folder

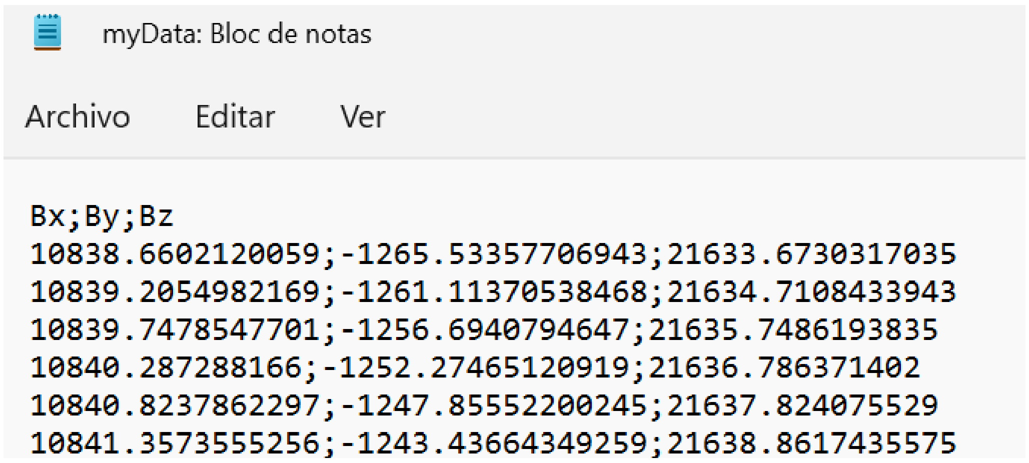
void readFile(char fileName[20]);
reads the information inside the opened file

void setBxx(double bxx);
This function sets the column of the X-axis magnetic field into array

void setByy(double byy);
This function sets the column of the Y-axis magnetic field into array

void setBzz(double bzz);
This function sets the column of the Z-axis magnetic field into array

The purpose of this class is to provide basic file reading skills. In order for the file to be read correctly, it must be in a specific format. The magnetic field components must be separated by a ; sign, and the units must be written in nanoteslas.



```
Bx;By;Bz
10838.6602120059;-1265.53357706943;21633.6730317035
10839.2054982169;-1261.11370538468;21634.7108433943
10839.7478547701;-1256.6940794647;21635.7486193835
10840.287288166;-1252.27465120919;21636.786371402
10840.8237862297;-1247.85552200245;21637.824075529
10841.3573555256;-1243.43664349259;21638.8617435575
```

Figure 2.14: magnetic fields file format

The time step will be set further in the start.cpp processing block.

Start class: This class is in charge of computing all the signs and currents from the magnetic fields inputs. It contains 2 important methods for the static and dynamic simulation:

void start_static_orbit_simulator();
This method used the 6 Helmholtz coills magnetic fileds inputs and computes the currents. Finally it communicates the values to the power supply and Arduino

void start_dynamic_orbit_simulator()
This method used the file and the earth magnetic field inputs and computes the currents. Finally it communicates the values to the power supply and Arduino dynamically along time.

static_orbit_simulator function firts step is to take the 6 inputs. 3 desired magnetic fields to create in the helmholtz coils and 3 earth magnetic fields at the moment of the simulation. Then the process jumps into processing of those inputs. The the process jumps into processing the initial variables as init_variables() function:

- double Bx,double By,double Bz
- double x_maglecture, double y_maglecture, double z_maglecture
- char signs0[50], char signs[50]
- double absCurrents0[3], double absCurrents[3]
- string time

init_variables function writes into variable signs0, signs, absCurrents0, absCurrents the values computed Some funtions used for the different operations of the code have been writen. C++ makes the code more complex when compared with matlab in the processing chain.

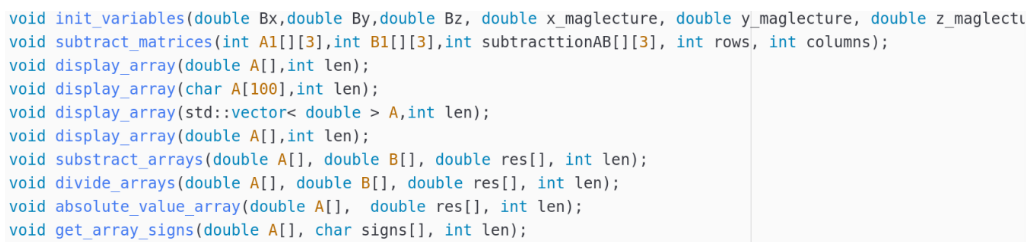
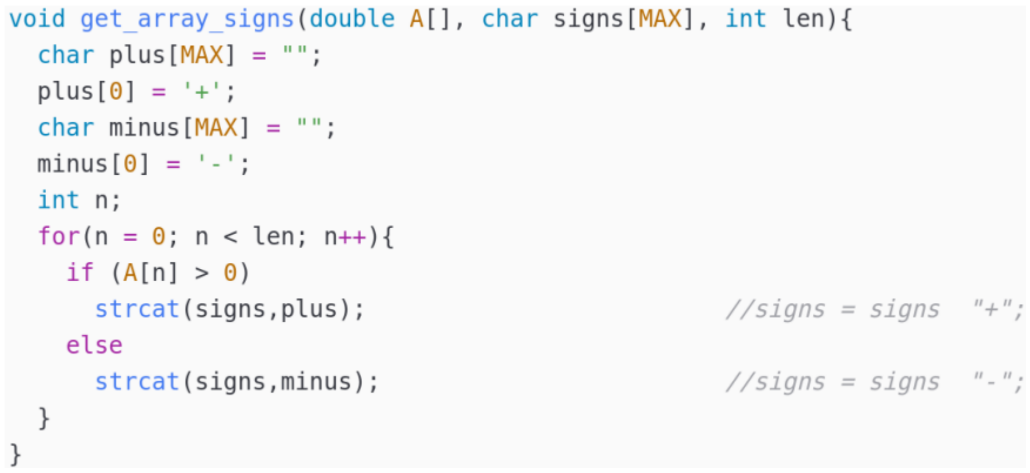

```
void init_variables(double Bx,double By,double Bz, double x_maglecture, double y_maglecture, double z_maglectu
void subtract_matrices(int A1[][3],int B1[][3],int subtracttionAB[][3], int rows, int columns);
void display_array(double A[],int len);
void display_array(char A[100],int len);
void display_array(std::vector< double > A,int len);
void display_array(double A[],int len);
void substract_arrays(double A[], double B[], double res[], int len);
void divide_arrays(double A[], double B[], double res[], int len);
void absolute_value_array(double A[],  double res[], int len);
void get_array_signs(double A[], char signs[], int len);
```

Figure 2.15: static orbit simulation functions

When getting the signs of the three currents, the absolute_value_array() function is used to save the currents into absCurrents variable.

In general all the fuctions or operators in matlab have been coded from zero in C++.



```c
void get_array_signs(double A[], char signs[MAX], int len){
  char plus[MAX] = "";
  plus[0] = '+';
  char minus[MAX] = "";
  minus[0] = '-';
  int n;
  for(n = 0; n < len; n++){
    if (A[n] > 0)
      strcat(signs,plus);                         //signs = signs  "+";
    else
      strcat(signs,minus);                        //signs = signs  "-";
  }
}
```

Figure 2.16: static orbit simulation functions

In addition to the dynamic simulation, the same variable changes have been implemented for the different serial sent variables as well. Similarly, a dynamic orbit simulation was prepared using the same functions and others adapted to the new data.

As a result of this code, the functions start_dynamic_orbit_simulation are called to obtain the signs as well as currents.
void start_dynamic_orbit_simulator(double Bxx_mag, double Byy_mag, double Bzz_mag, char fileName[50]);

With this function, the file class is used to start reading the file independently of the static function.

Using the function cut_vector(), it fills the vectors Bxx,Byy,Bzz with the magnetic fields from the file and cuts them from a given start to a given end

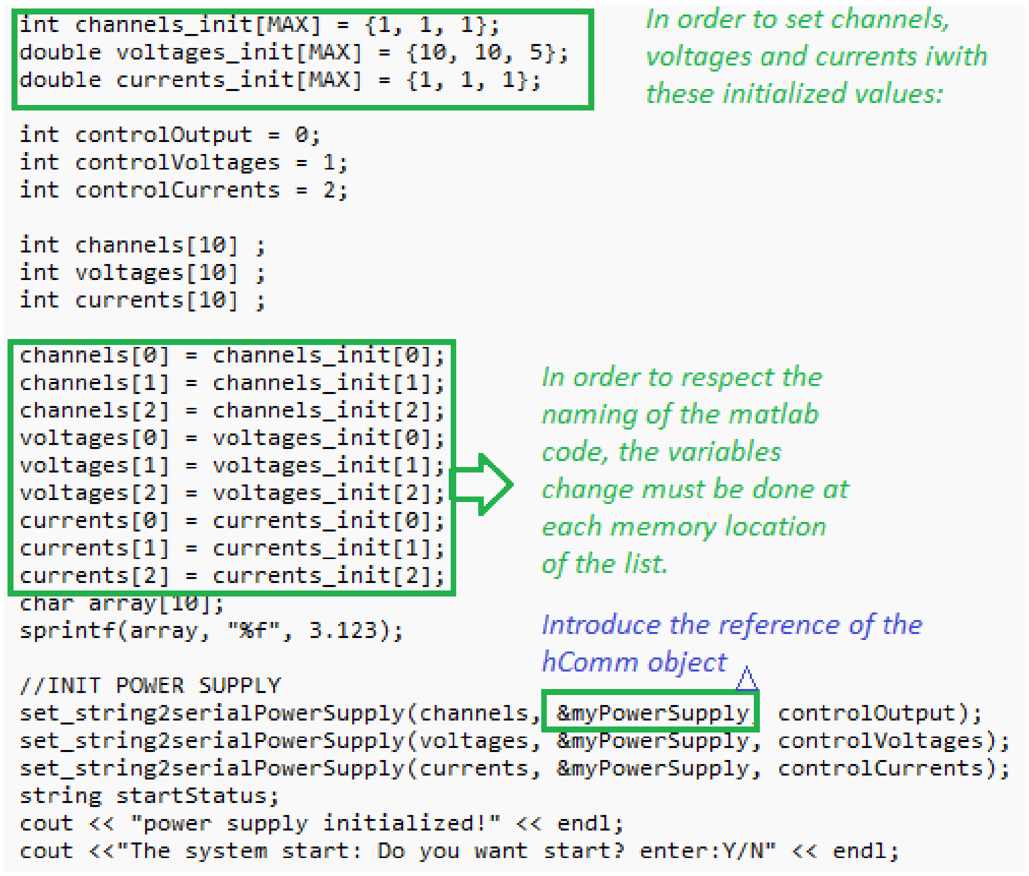
The magnetic fields resulting from this are then transformed into the ECEF coordinate system (see block 1.5.1).

Using the function ceci2ecef, the time is transformed into a Julianday and the transformation matrix is computed. The transformation matrix [35] is then applied to each row of the vector containing the magnetic fields of the file.

In order to obtain the values of the time transformed into Julian time, we use unixTime values, which may vary depending on the orbit:

unix = 1584662400 Orbits: M6P and Equ

unix=1583887278; Orbit: R2



```
int channels_init[MAX] = {1, 1, 1};
double voltages_init[MAX] = {10, 10, 5};
double currents_init[MAX] = {1, 1, 1};

int controlOutput = 0;
int controlVoltages = 1;
int controlCurrents = 2;

int channels[10] ;
int voltages[10] ;
int currents[10] ;

channels[0] = channels_init[0];
channels[1] = channels_init[1];
channels[2] = channels_init[2];
voltages[0] = voltages_init[0];
voltages[1] = voltages_init[1];
voltages[2] = voltages_init[2];
currents[0] = currents_init[0];
currents[1] = currents_init[1];
currents[2] = currents_init[2];
char array[10];
sprintf(array, "%f", 3.123);

//INIT POWER SUPPLY
set_string2serialPowerSupply(channels, &myPowerSupply, controlOutput);
set_string2serialPowerSupply(voltages, &myPowerSupply, controlVoltages);
set_string2serialPowerSupply(currents, &myPowerSupply, controlCurrents);
string startStatus;
cout << "power supply initialized!" << endl;
cout <<"The system start: Do you want start? enter:Y/N" << endl;
```

In order to set channels, voltages and currents iwith these initialized values:
In order to respect the naming of the matlab code, the variables change must be done at each memory location of the list.
Introduce the reference of the hComm object

Figure 2.17: static orbit simulation functions

2.2.2.2. C++ GUI using FLTK.

The interface has been implemented in the first executable file in the whole code named as main_v2.cpp which keeps calling the explained functions in the others files. The int main (int argc, char *argv[]) function executes the Interface with the function init_Interface().

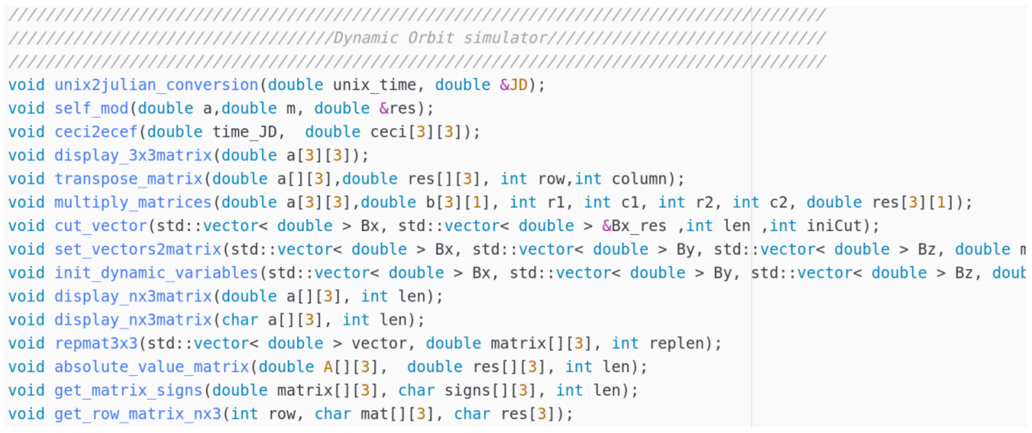
Init_interface performs the first task by creating a window that allows all possible interactions with the user to be displayed as *double Byy_mag, double Bzz_mag, char fileName[50]);*
In the template, all the buttons can be declared and initiated using *auto btnStatic = new MyButton(0, 200, BARWIDTH, 50, "Static Simulation");*

This command uses the class MyButton, which simplifies the creation of buttons. The process is repeated for each of the buttons Dynamic, close, open file, and stepped processing.

Once all the buttons have been created, they can be entered into a structure[3] containing dynamically all of them using the

code *struct inputs userInputs;;* to create the structure and *userInputs.load_btn = runbtn;*

to insert a given button. It is also possible to create and insert input boxes that allow FLTK into the structure by using the following syntax: *userInputs.new_input1 = new Fl_Input(BARWIDTH+100, 300, 250, 25,"Input Bx:");*



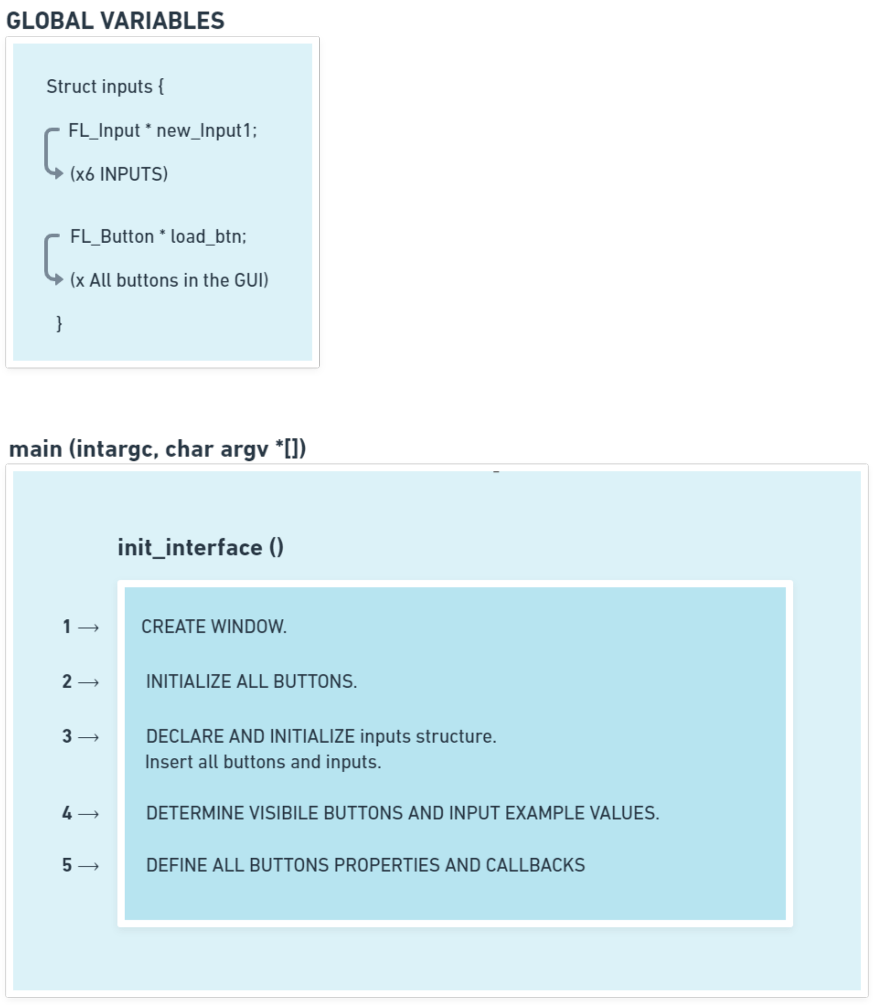
```
////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////Dynamic Orbit simulator//////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
void unix2julian_conversion(double unix_time, double &JD);
void self_mod(double a,double m, double &res);
void ceci2ecef(double time_JD,  double ceci[3][3]);
void display_3x3matrix(double a[3][3]);
void transpose_matrix(double a[][3],double res[][3], int row,int column);
void multiply_matrices(double a[3][3],double b[3][1], int r1, int c1, int r2, int c2, double res[3][1]);
void cut_vector(std::vector< double > Bx, std::vector< double > &Bx_res ,int len ,int iniCut);
void set_vectors2matrix(std::vector< double > Bx, std::vector< double > By, std::vector< double > Bz, double
void init_dynamic_variables(std::vector< double > Bx, std::vector< double > By, std::vector< double > Bz, dou
void display_nx3matrix(double a[][3], int len);
void display_nx3matrix(char a[][3], int len);
void repmat3x3(std::vector< double > vector, double matrix[][3], int replen);
void absolute_value_matrix(double A[][3],  double res[][3], int len);
void get_matrix_signs(double matrix[][3], char signs[][3], int len);
void get_row_matrix_nx3(int row, char mat[][3], char res[3]);
```

Figure 2.18: static orbit simulation functions

This will be repeated six times. When all objects are contained in the input created structure, it can be edited to have some specific properties as shown in step 4 of the figure 2.19. For example, userInputs.load_btn- hide() will hide the button and - show() will make it visible whenever desired. also, *userInputs.new_input1- value(exampleBxx);* gives a determined value to the box. Some example values have been entered:

char exampleBxx[20] = ”50000.0000”;
char exampleByy[20] = ”50000.0000”;
char exampleBzz[20] = ”-50000.0000”;
This will help the user to know the units that the magnetic fields must contain (nT).

Finally, in order to determine each button functionality when it is pressed it has been used the structure created as follows step 5 in Figure 2.19:



GLOBAL VARIABLES

```
Struct inputs {
  FL_Input * new_Input1;
  (x6 INPUTS)

  FL_Button * load_btn;
  (x All buttons in the GUI)
}
```

main (intargc, char argv *[])

init_interface ()

1 ⟶ CREATE WINDOW.

2 ⟶ INITIALIZE ALL BUTTONS.

3 ⟶ DECLARE AND INITIALIZE inputs structure.
Insert all buttons and inputs.

4 ⟶ DETERMINE VISIBILE BUTTONS AND INPUT EXAMPLE VALUES.

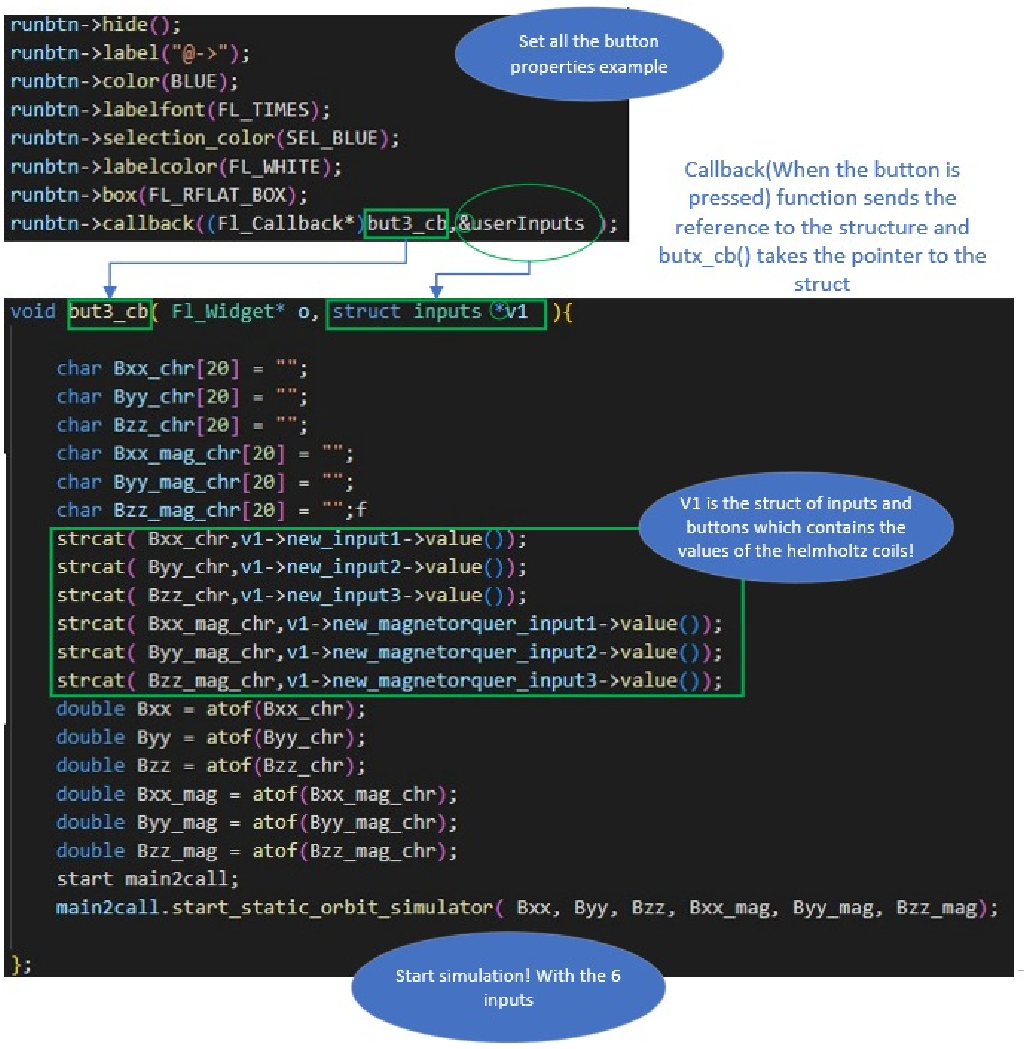
5 ⟶ DEFINE ALL BUTTONS PROPERTIES AND CALLBACKS

Figure 2.19: Interface process design

Each button and widget (inputs and boxes) has been updated in this manner. Once all the properties are set and the callbacks are defined with their button_changebutton functions, the interface can be displayed and executed repeatedly until the process is complete.
win.show();
Fl::run();

2.2.2.3. FLTK Principles and features.

UNIX, Linux, Microsoft Windows and MacOS are supported by FLTK (fulltick), a cross-platform C++ GUI toolkit. FLTK provides a modern GUI without the bloat and supports 3D graphics via OpenGL. Helmholtz does not use the most modern visualizations, but it is still an appropriate candidate for performing an easy GUI simply enough to automate the entire magnetic field generation process. FLTK is designed to be small and modular enough to be statically linked [37]. It uses all the headers in the library stack to link all the objects and create specific widgets and objects that can be used in the interface. It is a shared library that works as expected. Furthermore, FLTK contains an excellent UI builder called FLUID, which will be used to build the application.


Set all the button properties example
Callback(When the button is pressed) function sends the reference to the structure and butx_cb() takes the pointer to the struct
V1 is the struct of inputs and buttons which contains the values of the helmholtz coils!
Start simulation! With the 6 inputs

```
runbtn->hide();
runbtn->label("@->");
runbtn->color(BLUE);
runbtn->labelfont(FL_TIMES);
runbtn->selection_color(SEL_BLUE);
runbtn->labelcolor(FL_WHITE);
runbtn->box(FL_RFLAT_BOX);
runbtn->callback((Fl_Callback*)but3_cb,&userInputs );

void but3_cb( Fl_Widget* o, struct inputs *v1 ){

    char Bxx_chr[20] = "";
    char Byy_chr[20] = "";
    char Bzz_chr[20] = "";
    char Bxx_mag_chr[20] = "";
    char Byy_mag_chr[20] = "";
    char Bzz_mag_chr[20] = "";f
    strcat( Bxx_chr,v1->new_input1->value());
    strcat( Byy_chr,v1->new_input2->value());
    strcat( Bzz_chr,v1->new_input3->value());
    strcat( Bxx_mag_chr,v1->new_magnetorquer_input1->value());
    strcat( Byy_mag_chr,v1->new_magnetorquer_input2->value());
    strcat( Bzz_mag_chr,v1->new_magnetorquer_input3->value());
    double Bxx = atof(Bxx_chr);
    double Byy = atof(Byy_chr);
    double Bzz = atof(Bzz_chr);
    double Bxx_mag = atof(Bxx_mag_chr);
    double Byy_mag = atof(Byy_mag_chr);
    double Bzz_mag = atof(Bzz_mag_chr);
    start main2call;
    main2call.start_static_orbit_simulator( Bxx, Byy, Bzz, Bxx_mag, Byy_mag, Bzz_mag);

};
```

Figure 2.20: static orbit simulation functions

The FLTK libraries can be called using the following syntax:

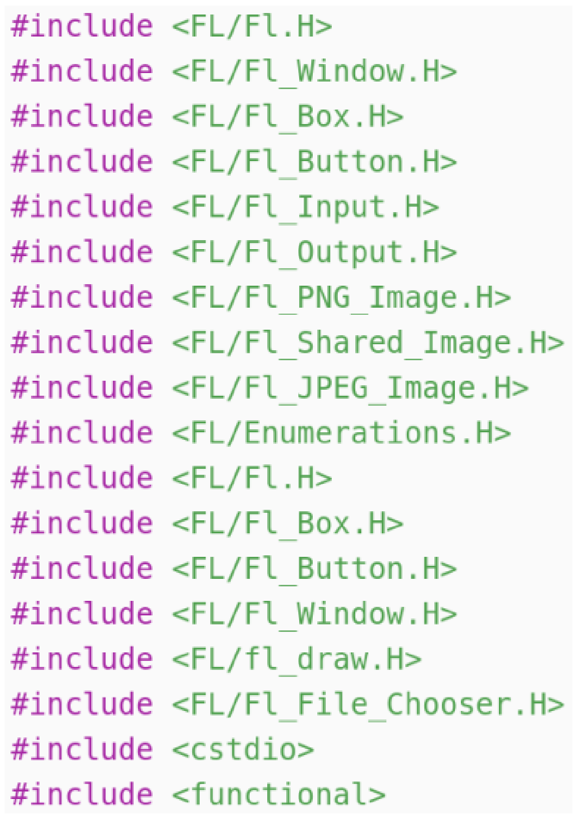
#include <FL/xxx.h>or these other #include <fltk/Xyz.h >. Helmholtz code is based on the first syntax type library.

In FLTK, each class has its own header file. It is common for one header file to include another. As an example, if Window.h is used, then Widget.h is not required. In other words, it is not always necessary to include all the classes, although it is a good practice to do so.

The use of namespace FL eliminates the need to type ”fl::” in front of all the symbols when calling internal FLTK functions. It is possible for the reader to find documentation that does not use the FL: or FLTK: nomenclature.

FL is the FLTK global (static) class containing state information and global methods for the Helmholtz application. As a result, FL is used instead of FLTK for the development of the application [36].

Widgets are created as soon as all the header files are included, the program must create a window in order to display the buttons and boxes. Fltk can do it with the following code:



```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_JPEG_Image.H>
#include <FL/Enumerations.H>
#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Window.H>
#include <FL/fl_draw.H>
#include <FL/Fl_File_Chooser.H>
#include <cstdio>
#include <functional>
```

Figure 2.21: Included headers

*fltk::Window *window = new fltk::Window(WIDTH,HEIGHT);*

altough in the Helmholtz code it has been used a modified class to do so. Then it results as:

Fl_Window win(WIDTH, HEIGHT,"Testing");

To create widgets of the type button, the class myButton has been used.

auto btnStatic = new MyButton(0, 200, BARWIDTH, 50, "Static Simulation");

This is the reason why the code does not execute a general widget of the type
*fltk::Widget *button = new fltk::Widget(20,40,260,100,"Hello, World!"););*
Most of the widget constructors have the following sintaxis: *fltk::Widget(x, y, width, height, label).*

Where x and y are the position of the upper-left corner of the widget. Then the widtha nd height represents the dimensions in pixels and finally the label is the text that the widget will show. If the label is not given it defaults set NULL.

Lastly, after all widgets are created and their properties are set, the window can be displayed using window $\rightarrow$ show(argc, argv); or for the Helmholtz code, using win.show(), which results from the creation of the object FL_window called win. FLTK::run() can be used to execute the main event loop. This will repeatedly wait for events from the user and then process them. In Helmholtz code it has been used Fl::run(); nomenclature, and does not return until all of the windows under FLTK control are closed (either by the user or your program).

2.2.2.4. *Interface structure.*

This application has been designed to have a simple and clean user interface. So, the code starts with three simple buttons: two buttons for Helmholtz activations and magnetic field generation, and one more button for propagating one orbit if necessary.

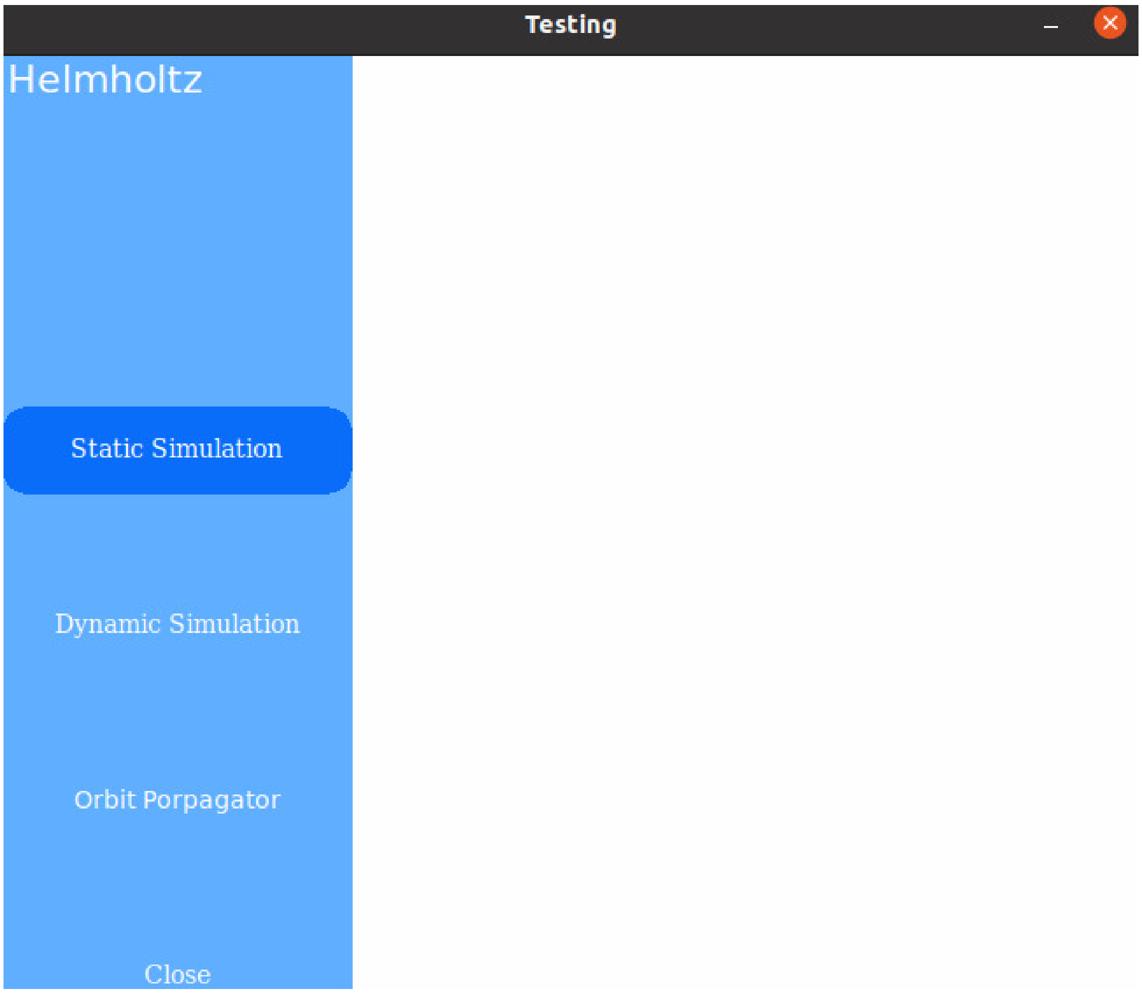
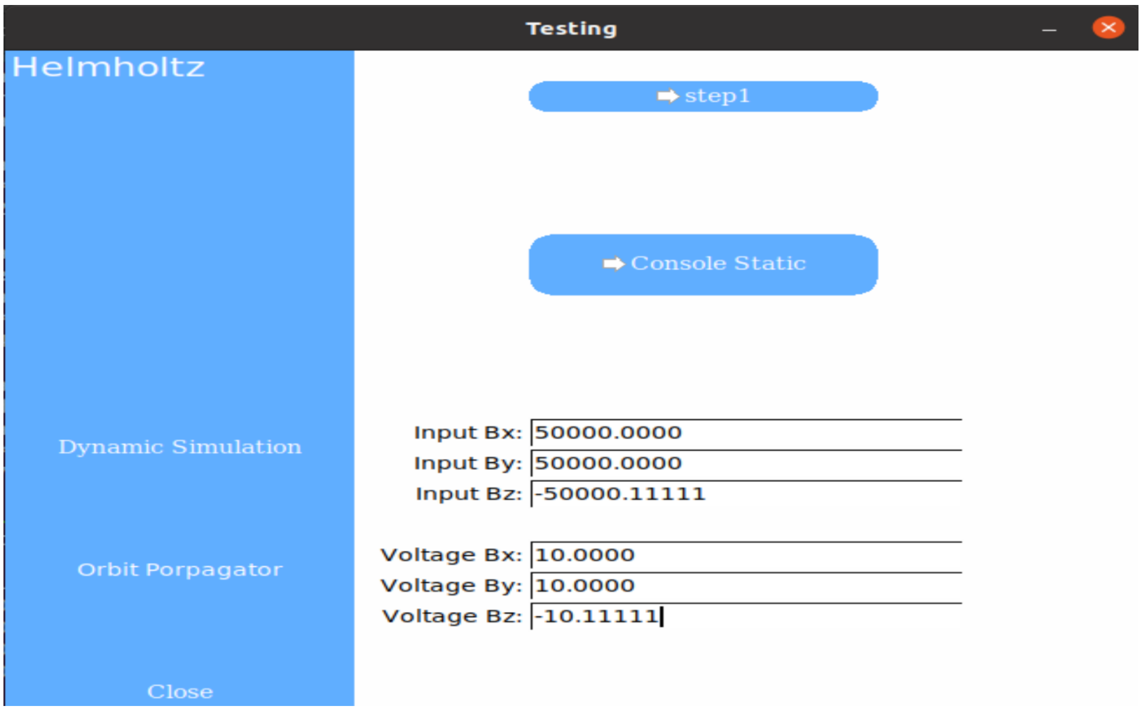


Figure 2.22: Interface structure

When the staticSimulation button is selected, two buttons will appear with six input boxes. The user has the option of simulating directly from the GUI by pressing the step1 button, which will change up to four steps. Each time the system sends data to the Arduino and power supply, the communication must rest for a period of time in order not to overcharge the power supply or Arduino. Notice in matlab this was done by using a running step simulation in which the code was executed step by step using run section MATLAB option. Using the console simulation button, the linux console will emerge displaying some useful data such as the exact serial messages sent or the computed currents if the console simulation button is used.

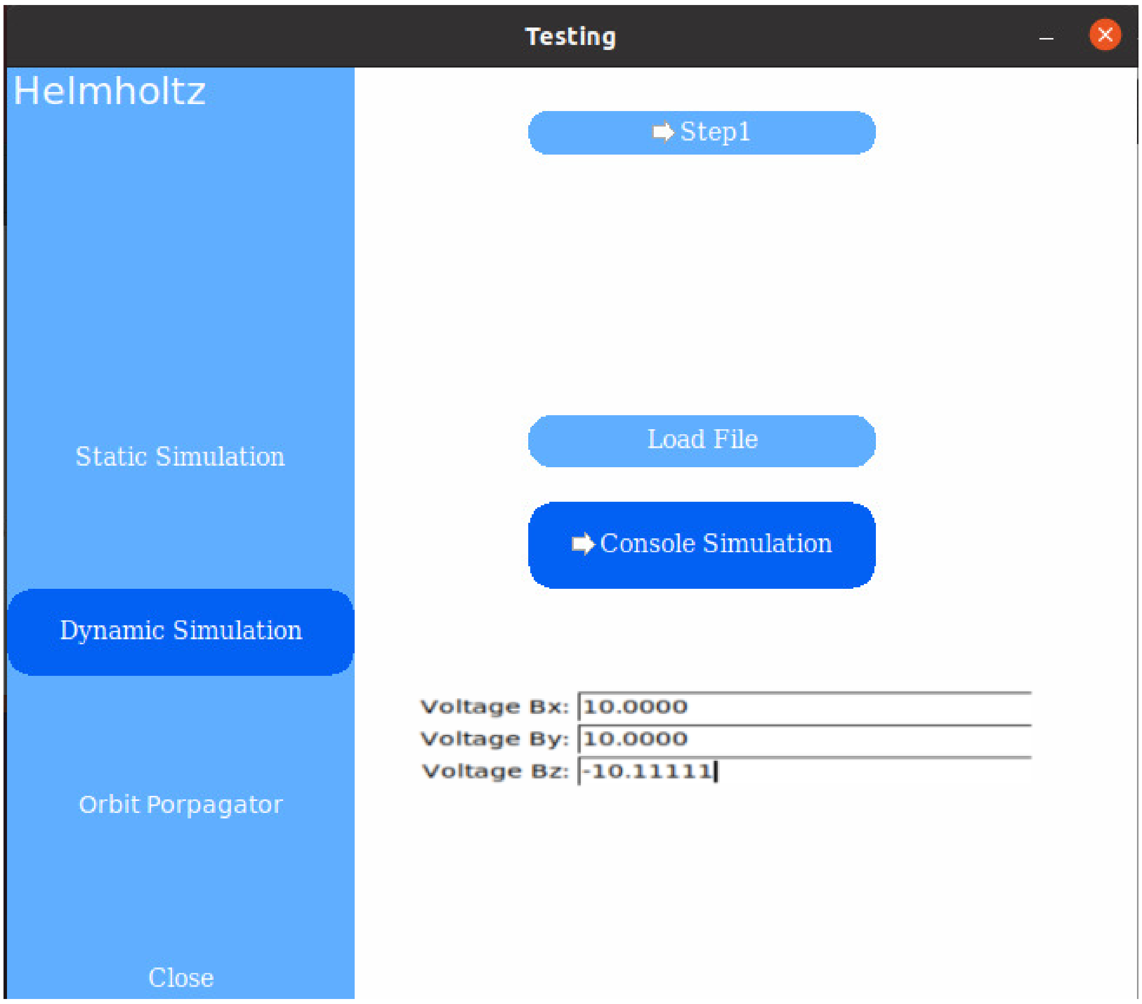


Input Bx: 50000.0000
Input By: 50000.0000
Input Bz: -50000.11111

Voltage Bx: 10.0000
Voltage By: 10.0000
Voltage Bz: -10.11111

Figure 2.23: Interface structure

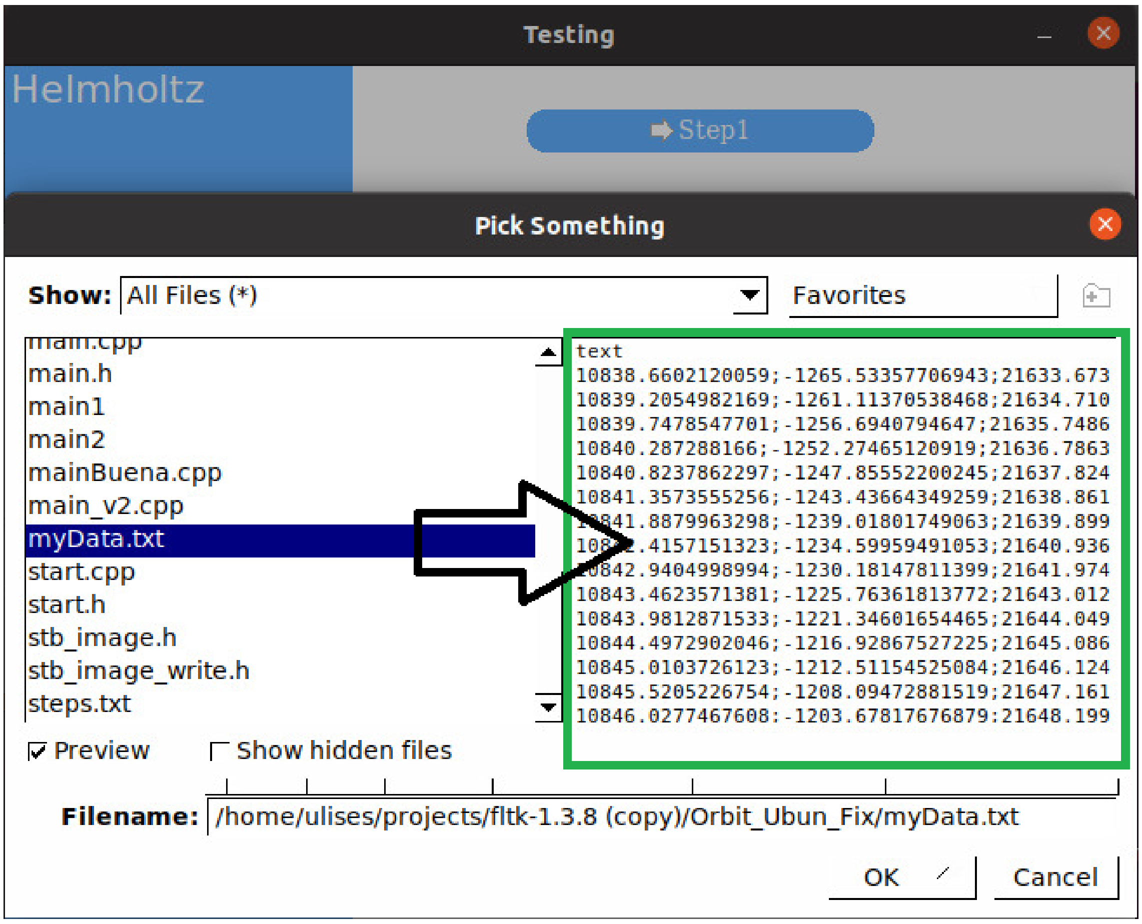
When the dynamic simulation button is selected, 3 inputs to set from the user are available (mangetometer voltages) and 3 buttons allows the user to switch betweeen a console simulation or a 2 step simulation.



Voltage Bx: 10.0000
Voltage By: 10.0000
Voltage Bz: -10.11111

Figure 2.24: Interface structure

Before the dynamic simulation start, it is mandatory to select the file containing the magnetic fields for the simulation. This file will have the format explained in the previous section.



```
text
10838.6602120059;-1265.53357706943;21633.673
10839.2054982169;-1261.11370538468;21634.710
10839.7478547701;-1256.6940794647;21635.7486
10840.287288166;-1252.27465120919;21636.7863
10840.8237862297;-1247.85552200245;21637.824
10841.3573555256;-1243.43664349259;21638.861
10841.8879963298;-1239.01801749063;21639.899
10842.4157151323;-1234.59959491053;21640.936
10842.9404998994;-1230.18147811399;21641.974
10843.4623571381;-1225.76361813772;21643.012
10843.9812871533;-1221.34601654465;21644.049
10844.4972902046;-1216.92867527225;21645.086
10845.0103726123;-1212.51154525084;21646.124
10845.5205226754;-1208.09472881519;21647.161
10846.0277467608:-1203.67817676879:21648.199
```

Filename: /home/ulises/projects/fltk-1.3.8 (copy)/Orbit_Ubun_Fix/myData.txt

Figure 2.25: File data selection

CHAPTER 3. ANGULAR VELOCITY MEASUREMENTS

3.1. Working principles

Measuring the angular speed of a nanosatellite while performing some tests can be a difficult task if the Helmholtz coils structure does not have space enough to set some hardware to measure periodic samples of the rotatory disk. One easy solution to solve the problem would be to use RF sensors and sample each time the disk crosses the infrared sensor. Although it is simple, the angular velocity would not be so accurate. Moreover, AOCS testings are usually performed with speeds from 2 to 60 º/s, so it would not have enough samples for the desired solution. Still, it exists a way to find the angular velocities at a constant rate and accurate enough: using image processing. The idea of using a camera to solve this problem is thrilling since it could measure the angular velocities of one rotatory disk at a high sampling rate. This chapter will discuss one solution for the problem proposed, its flaws, and its advantages.

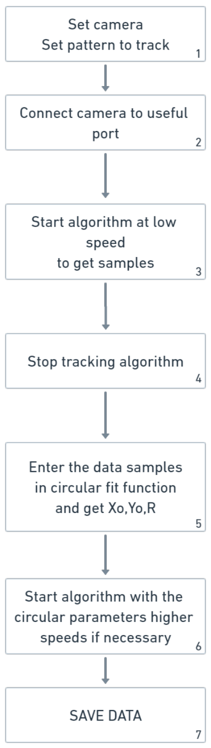

Set camera
Set pattern to track 1
Connect camera to useful port 2
Start algorithm at low speed to get samples 3
Stop tracking algorithm 4
Enter the data samples in circular fit function and get Xo,Yo,R 5
Start algorithm with the circular parameters higher speeds if necessary 6
SAVE DATA 7

Figure 3.1: general flow chart for the angular velocity algorithm

Figure 3.1 shows all the general process that has been designed to reach the solution. First of all, it is important to set the camera to a location in which the direction of the lens is pointing to the center of the disk. Although centering the camera is difficult, the problem has been designed to avoid this issue by solving some math problems in a further discussion. Also, one pattern must be set inside the disk. This pattern will be used as the target while performing the tracking. Once the camera is centered outside the Helmholtz coil, it is mandatory to find the camera ports (2). To know the movement properties of the disk, the imaginary circle at a distance R can be found by using a circular fit process to obtain the circular parameters of the pattern we previously pasted in the disk (3).

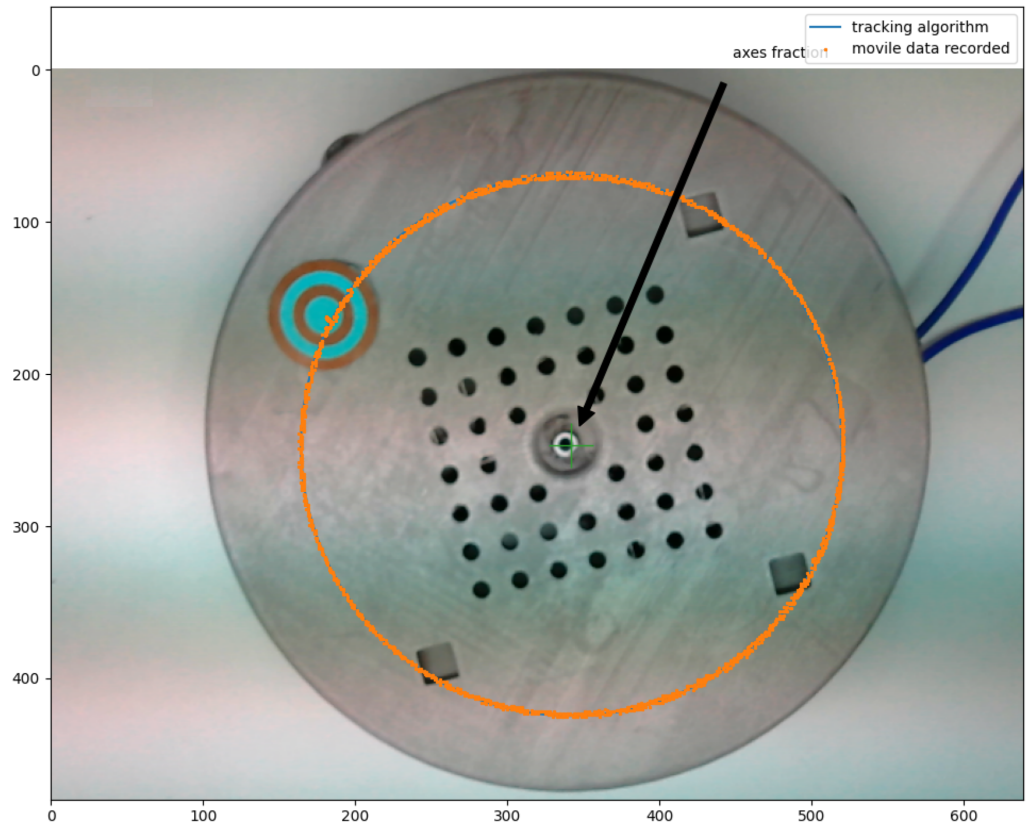


Figure 3.2: System calibration

It is recommended to use low angular velocities to calibrate the system. Since it is not mandatory to measure any angular velocity at this step, the best option is to operate at a low speed. Thus, the tracker can obtain samples close to the imaginary circle which will give better and more accurate information in the fit function. Figure 3.2 shows how the calibrating process is done: first, take some samples very close to the circle and use the fit to obtain the center of the circle and the radius. Notice that the center can be different from the physic center of the disk. But it is recommended to obtain a center close to the real one. Otherwise, consider rotating and center the camera to have the lens center pointing directly at the midpoint of the disk. In blue the circle is plotted, it can be seen in the background of the orange sample as the best circular fit.

Once the circle is known with its center and the radius [5], the algorithm can restart at higher speeds while the tracking will be activated steadily until the end of the whole algorithm [6].

3.1.1. Angle measurement

Once the circle is defined, the next step is to compute the angle ϕ at each desired sample both time sampling or spacial. This angle consist on the angle formed by two consecutive samples, in 3.44 ϕ would be 40 degress. Therefore, the angular velocity can be comupted as the ratio between $\phi$and the time elapsed from the measurement of the first and the second sample. However, some problems have to be considered while computing the angle.

Bbox (template) coordinates change and zoom bbox:

In order to track the desired object, the user can select the area that the system will automatically follow during the tracking process. This is called a Bbox and it is not mandatory to be a square shape. Rectangular shapes can be also considered.

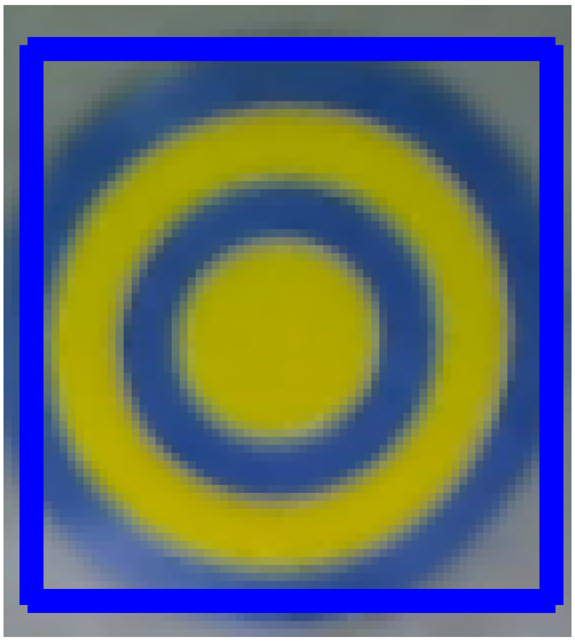
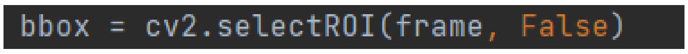


Figure 3.3: code to select the window template (Python)

In order to use Bbox using ROI function or other functions throughout this project, OpenCV has been used. OpenCV refers to Computer Vision and is currently one of the the biggest CV library in the world. It provides packages realted with image procesing and AI. OpenCV allows us to select a region in the image domain which will be a candidate to determine the reference for the overall algorithm computations [7]. All the configuration and installation of cv2 libraries has followed by the link cited. However, different versions on the libraries could be used for different tracking algorithms. So notice that not all the further releases executes all the libraries of the code.



```
bbox = cv2.selectROI(frame, False)
```

Figure 3.4: code to select the window template (Python)

Some trackers are sensitive to deep changes such as zooms or shape-changing objects. However, our pattern object should not change its size throughout the test. Furthermore, the camera is always in a static location and no zoom is used in the process of tracking. Thus, any change in the bbox surface is not welcome to track the one object which ideally would be the same as the first bbox the user selects when the algorithms start running.

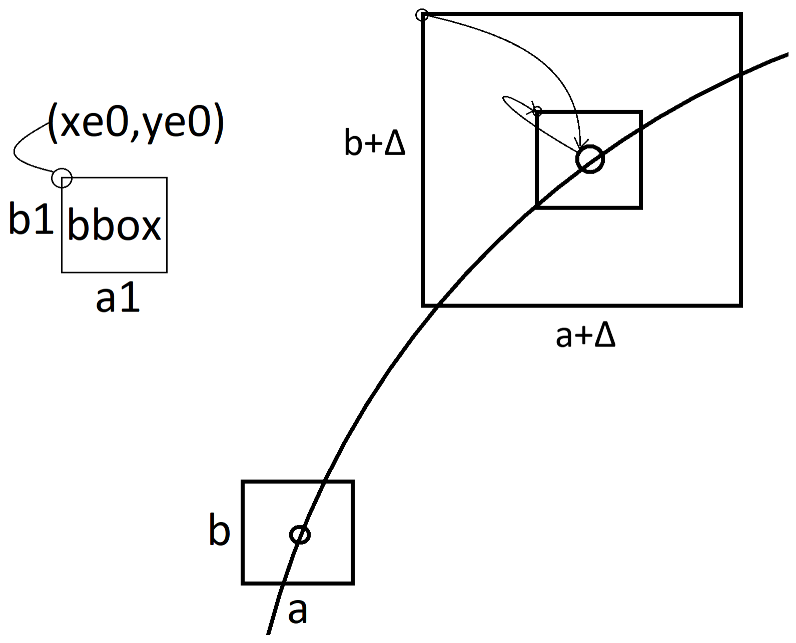


Figure 3.5: scheme to limit the dimensions of the template

One bbox is defined with the coordinates of the left upper edge and the length of both sides as a tuplet

$$bbox1 = (xe1, ye1, a1, b1) \tag{3.1}$$

If the size changes while moving the system should resize the bbox. To do so, find the bbox changed center as

$$bboxnewcenter = (xe1 + (a1 + \Delta)/2, ye1 - (b1 + \Delta)/2) \tag{3.2}$$

Notice the Y-axis goes downwards in image processing.

$$bboxnew = (xe1 + (a1 + \Delta)/2 – a1/2, ye1 - (b1 + \Delta)/2 – b1/2, a1, b1) \tag{3.3}$$

Also, at each turn, the system should consider cyclic loops in which the maximum possible angle is 360º.

Finish lap problem:

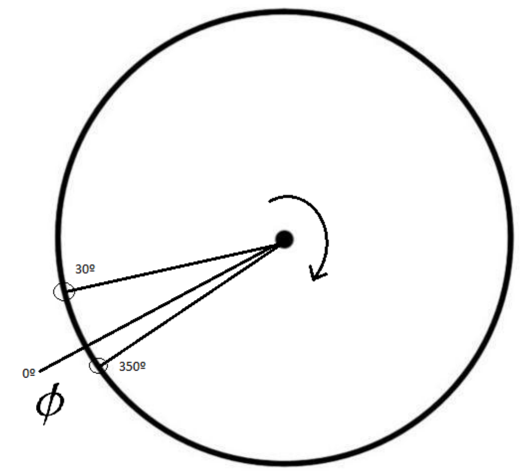


Figure 3.6: Final lap problem scenario

Imagine the scenario proposed above, in which two consecutive samples are taken at 350º and 30º when crossing the reference angle where the algorithm started (the location where the bbox was selected). The angular speed would be computed as the ratio between phi – pri previous$(T1-T2)$ where T1 and T2 are the times in which the samples were taken. Notice how the algorithm will compute a wrong angular speed when the samples are taken close to the reference point. The correct angular increment would be then$abs(phi-phiprev-360)$ for instance in this case abs$(350-30-360)=40$ So it is necessary to know when the samples are crossing the reference. In other words, it is necessary to count the number of turns the disc does. If the tracked pattern does increment its number of turns, this problem must be taken into account.

3.2. Auto shape detector

3.2.1. Linear least squares fitting.

3.2.1.1. Linear least squares fitting of a circle

Calibrating the angular velocity algorithm to know the disk properties can be solved easily by measuring the real physical dimensions of the disk and the lens' length of the camera. However, this is a rough process because not all the laboratories have the same Helmholtz coils and the camera can be moved easily or other issues could be found. The solution proposed is more versatile and avoids camera undesired movements. The most important feature is that can be adapted to be used at any circularly rotatory object.

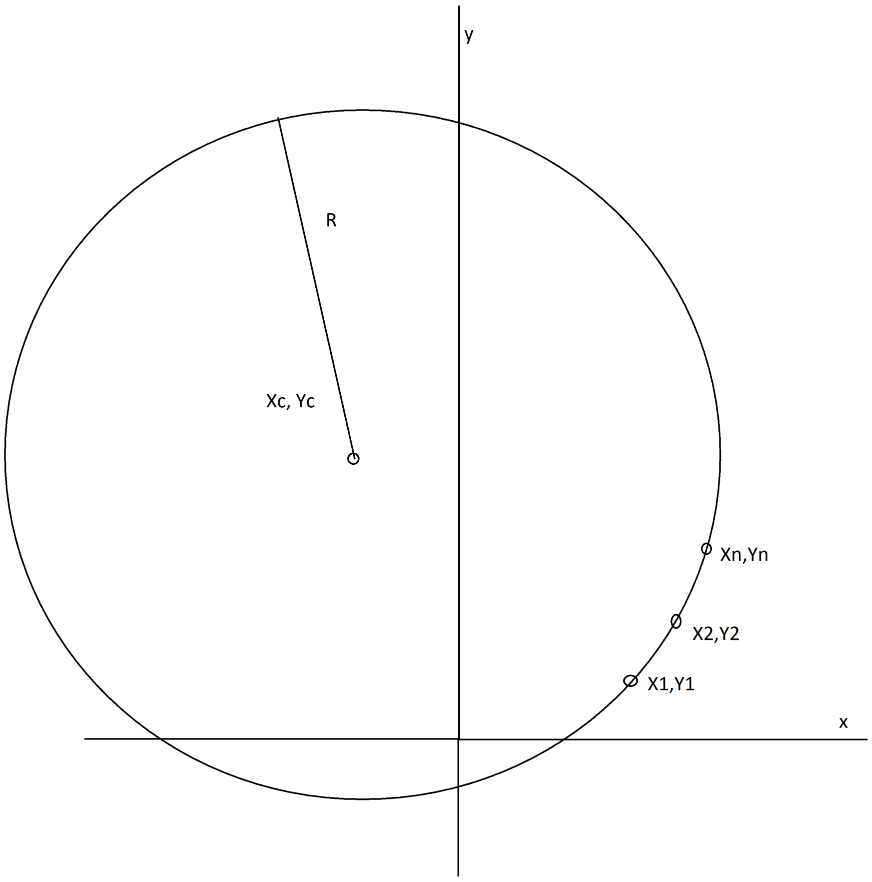


Figure 3.7: Circular fit scheme

Imagine the camera tracks some circular samples. The aim of the linear least squares fitting for the circle is to find the center and the radius that best fits those sampled points [10]. Therefore, can be treated as an optimization problem where the objective is to minimize the error of the circular algebraic expression.

$$g(x_i, y_i) = (x_i - x_c)^2 + (y_i - y_c)^2 - R^2 \tag{3.4}$$

$$\overline{x} = \frac{1}{N}\sum_i x_i \tag{3.5}$$

$$\overline{y} = \frac{1}{N}\sum_i y_i \tag{3.6}$$

For simplicity let's define $\alpha = R^2$

The problem must minimize S:

$$S = \min\left\{\sum_i g(x_i, y_i)^2\right\} \tag{3.7}$$

Let's now take the partial derivatives of the 3 output variables and set them to zero:

$$0 = \frac{\partial S}{\partial \alpha} = 2\sum_i \frac{\partial g(x_i, y_i)}{\partial \alpha} \cdot g(x_i, y_i) = -2\sum_i g(x_i, y_i) \tag{3.8}$$

$$\begin{aligned}
0 &= \frac{\partial S}{\partial x_c} \\
&= 2\sum_i \frac{\partial g(x_i, y_i)}{\partial x_c} \cdot g(x_i, y_i) \\
&= 2\sum_i g(x_i, y_i) \cdot [2(x_i - x_c)(-1)] \\
&= -4\sum_i g(x_i, y_i) \cdot (x_i - x_c) \\
&= -4\sum_i x_i g(x_i, y_i) + 4\sum_i x_c g(x_i, y_i) \\
&= -4\sum_i x_i g(x_i, y_i) + 4x_c \sum_i g(x_i, y_i) \\
&= -4\sum_i x_i g(x_i, y_i)
\end{aligned}$$

Notice that $4\sum_i x_c g(x_i, y_i)$ is null since x_c is a constant and the value of the expression has been proved to be zero in the first derivative.

Now, expanding the expression obtained results:

$$0 = \frac{\partial S}{\partial x_c} = -4\sum_i x_i g(x_i, y_i)^2 \tag{3.9}$$

$$\sum_i x_i g(x_i, y_i)^2 = 0 = \sum_i x_i \left(x_i^2 - 2x_i x_c + x_c^2 + y_i^2 - 2y_i y_c + y_c^2 - \alpha\right) \tag{3.10}$$

Defining: $S_u = \sum_i x_i$, $S_{uu} = \sum_i x_i^2$ and $S_{uuu} = \sum_i x_i^3$ $S_v = \sum_i y_i$ $S_{vv} = \sum_i y_i^2$ and $S_{vvv} = \sum_i y_i^3$

$$S_{uuu} + S_u x_i^2 + S_{uvv} + S_u y_c^2 = 2S_{uu} x_i + 2S_{uv} y_c + S_u \alpha \tag{3.11}$$

This leads to a system of an equation that is not linear. However, the system can be linearized using the following linear transformation:

$$u_i = x_i - \bar{x}, v_i = y_i - \bar{y} \tag{3.12}$$

with this transformation the term Su results in a null sum. Thus, $Su = 0$

$$\frac{1}{2}(S_{uuu} + S_{uvv}) = S_{uu}x_c + S_{uv}y_c \tag{3.13}$$

Simillarly using $S_v = 0$ gives:

$$\frac{1}{2}(S_{vvv} + S_{vuu}) = S_{vv}y_c + S_{vu}x_c \tag{3.14}$$

Finally to obtain the radius:

$$\begin{aligned}\sum_i g(x_i, y_i) = 0 = \sum_i x_i^2 - 2x_ix_c + x_c^2 + y_i^2 - 2y_iy_c + y_c^2 - \alpha \quad &= S_{uu} + \sum_i x_c^2 + S_{vv} + \sum_i y_c^2 - \sum_i \alpha \\ &= S_{uu} + S_{vv} + N\left(x_c^2 + y_c^2 - \alpha\right)\end{aligned}$$

$$\alpha = \frac{S_{uu} + S_{vv}}{N} + x_c^2 + y_c^2 \tag{3.15}$$

Finally reconvert to the original coordinates: $x_c = x_c' + \bar{x}$ $y_c = y_c' + \bar{y}$

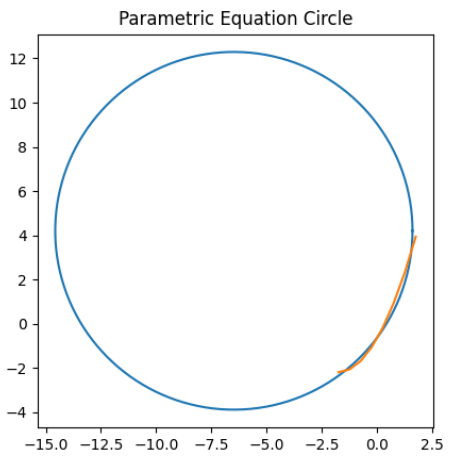


Figure 3.8: Parabola and its circular fit

In Figure 3.8 a parabolic function of the type $1/2x^2$ has been plotted and also the fitted circle which best adapts to the parabola. Notice how the algorithm works even though both curves have features utterly different. Therefore, if a set of points is given by the camera when calibrating the algorithm, the fit algorithm will be ready to find the best circular fit. The circular fit fuction coded with the explanation seen has been writen in Python. See appendix D

3.2.1.2. *Linear least squares fitting of an ellipse*

It is comprehensive to think that since the pattern object moves circularly in a rotatory disc, the model to calibrate the algorithm must be a circular one. However, it is difficult to set a camera at a desired location in the Helmholtz coils, centred and plain.

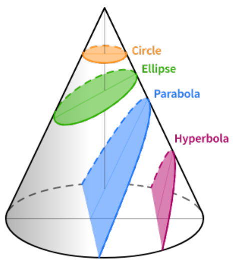


Figure 3.9: Conics sections which would be observed by the camera

Notice how the camera must be centered at the focus of the disc in order to operate with circular fitting algorithms. The general perspective of the disc from the camera's point of view is represented in the figure above: where the camera would be located at the vertex of the cone and the disc would be placed at its relative position as follows. Although the shapes found are circles, ellipses, parabolas and hyperbolas, for the general case of measuring the angular speed the problem can be solved for circles and ellipses. Notice the ideal case is the circle in which the algorithm works properly for the requirements of the tests.

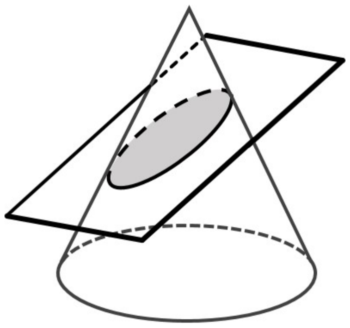


Figure 3.10: Ellipse: most common conic section captured by a camera

Despite the effort of centering the camera to obtain perfect circles, the camera lens often has shape distortions as fisheyes which could operate more efficiently with ellipse algorithms. Although the model using ellipses has not been executed, it is convenient to understand how it works for further studies and possible improvements of the algorithm accuracy.

Fitting an ellipse to dataset of xy plane is commonly analyzed as an optimization problem in which the objective of the problem is to minimize the sum of the squares of the algebraic distances between the fit and the data. This problem will be solved in a different way than the linear least squares fitting of a circle since the analytical expression of the partial derivatives are more complex [16]. The general expression of a conic is the following [13]:

$$F(x,y) = ax^2 + bxy + cy^2 + cy^2 + dx + ey + f = 0 \tag{3.16}$$

$$\boldsymbol{D} = \begin{pmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & x_ny_n & y_n^2 & x_n & y_n & 1 \end{pmatrix}$$

Figure 3.11: Matrix with the data points [14]

$$\boldsymbol{C} = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 3.12: Ellipse condition [14]

Notice that the conic depicts a linear equation with parameters a, b , c, d, e and f to be found. The polynomial F(x,y) is often called the algebraic distance of any point (xi,yi).
To avoid the symmetry of the sum in the expression, the function to minimize must be squared. Thus, one possible algorithm may be to minimize the sum of the squares of the algebraic distances F(x,y).

$$d(a) = \sum_i F(x,y)^2 \tag{3.17}$$

To avoid the trivial solution in which all the coefficients are zero, we can define f=1. Another problem is that F(x,y) does not distinguish between different conics... so if a hyperbola fitted more than an ellipse, the solution would converge to the hyperbola fit. This could happen with the case scenario of the circle, in which if ideally, the camera is at the perfect center of the disc axis. Then the algorithm automatically would converge for circular solutions. To avoid this problem, it is important to define the ellipse features in the problem. An ellipse must be subject to the condition $b^2 - 4ac < 0$. However, it can be proved that the parameters a, b, c, d, e and f can be scaled arbitrarily to an equality constraint $4ac - b^2 = 1$ and ensure that F(x,y) is an ellipse. The least-squares fitting problem can be expressed as minimizing $(||Da||^2)$ subject to:

$$a^T Ca = 1 \tag{3.18}$$

D represents the data points. C matrix represents the ellipse condition:

$$(2c, -b, 2a, 0, 0, 0)(a, b, c, d, e, f)' = 2ca – bb + 2ac \tag{3.19}$$

$$(a, b, c, d, e, f)C = (2c, -b, 2a, 0, 0, 0) \tag{3.20}$$

The solution is obtained when solving the system

$$Sa = \lambda Ca \tag{3.21}$$

$$a^T Ca = 1 \tag{3.22}$$

where the scatter matrix S:

$$S = D^T D \tag{3.23}$$

There are up to 6 solutions and it is used the one with the smallest positive eigenvalue. Finally, the corresponding eigenvector corresponds to the coefficients which represent the best fit ellipse in the least squares sense.

3.2.2. Auto shape detector with Hough circle

Selecting the bbox template to track with the selectROI function is indeed very useful. However, sometimes could be interesting to simulate several times different scenarios inside the Helmholtz coils. Therefore, each time the tracker algorithm starts, bbox must be selected to obtain the angular velocities measurements. To avoid repetition processes during the satellite tests and simplify the overall tracking process, auto shape detection has been taken into account. Furthermore, the idea of having in the future an utterly autonomous embedded system forces the system to operate with driverless shape detectors processes like the one proposed.

Hough circle algorithm

Imagine that the system needs to detect a circle contained inside an input image. The desired output of the problem is the circle properties such as center and radius, output=(xo, yo, R). Thus, one possible solution for the problem could be the adapted solution for a linear least squares fitting for the circle case. However, it can not be ensured that the input image has only 1 circle. Furthermore, we can not ensure the possible number of circles in the image have the same dimension. The fitting algorithm for this scenario would be so complex, although if they are treated for some particular cases they could be utterly accurate. The way of solving the problem is simpler if we obtain a clean image with the edges containing circles as candidates.

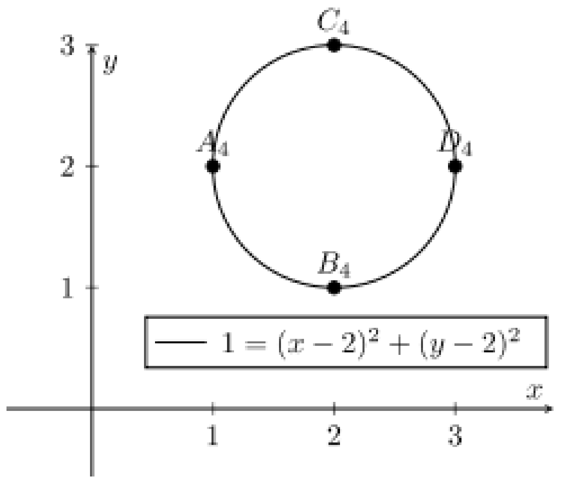


Figure 3.13: Data set to find its centre

Let A4, B4, C4, and D4 edge points of the input image after applying the edge transform(ie. Canny). Mathematically they full fill the circle condition defined by the circle equation. Thus, the algorithm should find the point at which the distance from each edge pixel to that desired point is constant. This point will be defined as the circumference center of radius R, (a0, b0, R).

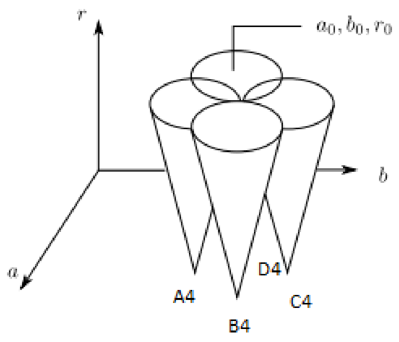


Figure 3.14: Set of cones used to find the shape center

The cones generated from the edge points determine the center of the circle. However, this is simple if we know the exact dimension of the circle we desire to detect. In practice, it is challenging to know the exact length of the shapes to detect. Therefore, the problem consists in a 3 dimensions problem (a,b,r).

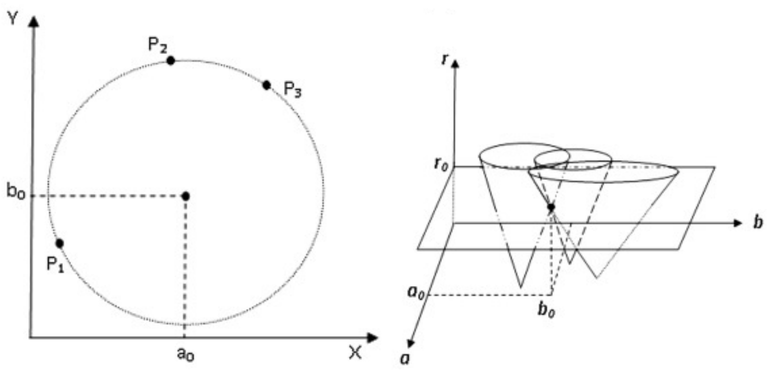
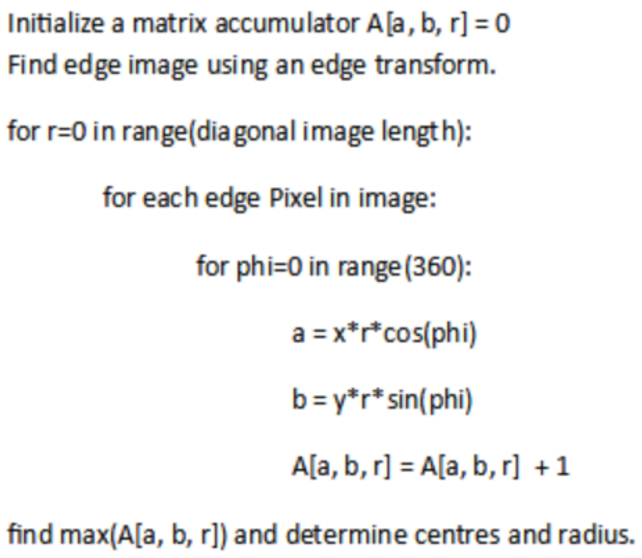


Figure 3.15: Set of cones used to find the shape center (variable radius)

The way to solve this type of problems can be time consuming and computationally complex if 3 dimensions are considered. For our case scenario, it is useful to fence in the radius solution.

Algorithm:



```
Initialize a matrix accumulator A[a, b, r] = 0
Find edge image using an edge transform.
for r=0 in range(diagonal image length):
    for each edge Pixel in image:
        for phi=0 in range(360):
            a = x*r*cos(phi)
            b = y*r*sin(phi)
            A[a, b, r] = A[a, b, r] + 1
find max(A[a, b, r]) and determine centres and radius.
```

Figure 3.16: Hough circle algorithm pseudocode

Figure 3.16 shows the pseudocode of the proposed algorithm. Figure 3.16 represents the simplified edge transform of an image containing a circle.For all the pixels in the image that are contained in the edge set represented by the black line in 3.16. Each point of the set creates a discrete circle defined by the angle step from 0 the 360 degrees. All the circles created by the edge with different radius, are acumuluted and finaly the maximum number of radius represented as the red lines in 3.16 defines the center of the circle at that given radius.

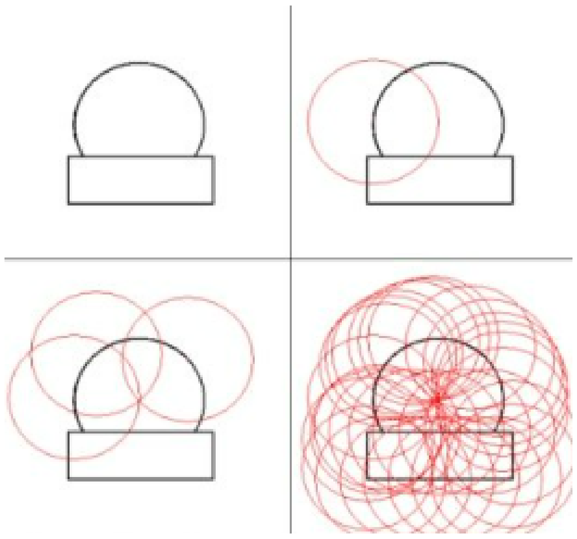


Figure 3.17: Representation of the algorithm for non perfected shapes

Notice that if the image contains more than one circle, instead of evaluation the maximum of the accumulative matrix, it is better to consider a threshold strategy and define a circle every time A[a,b,r] exceeds that one. In general, the whole process can be summed up in 7 steps as flow chart 3.39 shows.

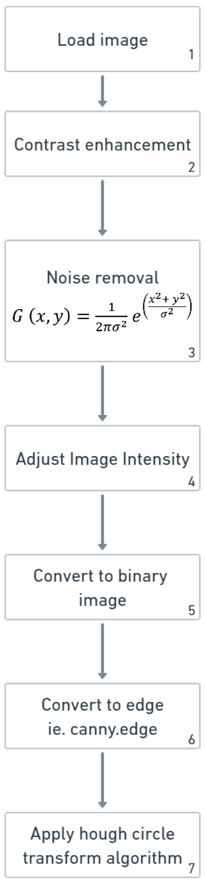

Load image 1
Contrast enhancement 2
Noise removal
$G\ (x,y) = \frac{1}{2\pi\sigma^2} e^{\left(\frac{x^2+y^2}{\sigma^2}\right)}$ 3
Adjust Image Intensity 4
Convert to binary image 5
Convert to edge ie. canny.edge 6
Apply hough circle transform algorithm 7

Figure 3.18: Hough circles steps

The first image the camera receives from the static disk will be used to set the hough circle algorithm. Therefore, that first image is automatically adjusted in the value of intensity and the map of color. Then, the noise removal filter will improve its quality and once the noise is removed, a threshold is applied for those pixels lower than this value. Converting those pixels automatically to value 1 (white) and the rest of the pixels to 0 (black). This process is also called a binary conversion. Usually, this leads to a contrasted image in which the background can be represented as black. Finally, an edge transform is applied. Commonly, canny edge is applied, and after setting the edge of the processed image, the hough algorithm can be applied [7].

Edge detection consist on computing the locat change of all the image and campare that change with a given defined threshold. That locat change can be determined by computing the local gradient magnitude which indicates the edge strength.

$$|grad(f(x,y))))| = \sqrt{\left(\frac{\partial f(x,y))}{\partial x}\right)^2 + \left(\frac{\partial f(x,y))}{\partial y}\right)^2} \tag{3.24}$$

The edge direction can be obtained by finding the angle of the two derivatives computed att each axis.

$$\theta = tan^{-1}(\frac{\frac{\partial f(x,y)}{\partial y}}{\frac{\partial f(x,y)}{\partial x}}) \tag{3.25}$$

Although some other types of derivative are used depending on the edge purpose. Eq 3.24 shows the first derivative implementation. This is translated in image processing on applying a filter(matrix) on the image that usually the center column is set to zero and the right and left sides of the matrix filter are equal but with different signs. When the value that the derivate shows a larger value compared with the threshold, the algorith sets a positive value to the edge transformation. This process is called hysteresis thresholding [9]. This project consider Canny edge detection a the main algorithm to consider any tipe of shape in all the code. cv.Canny() is the python function used in the overall code for edges purposes.

Results:

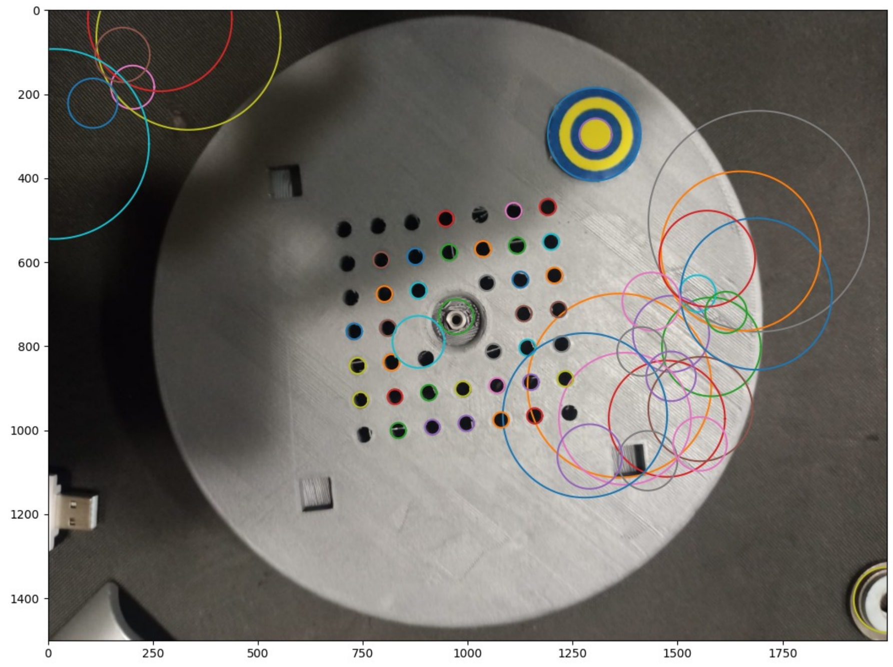


Figure 3.19: Auto shape detect without previous design

Applying the Hough circle method without any dimension reference can have several problems as can be observed in figure 3.19. Many circles are detected, which means that the problem was correctly solved for the 3 variables [a, b, r]. The next step is to fix the problem by using some physic lengths as a reference. The best reference to have into account is that the max length of the disk must be less or equal to the width and height of the image. Another parameter that would help in the design is the radius ratio r/R where r and R are the pattern radius and disk radius respectively. The following code defines the length of the image diagonal which will be used to define the houghCircles function. imagdiagonallength = math.sqrt(img.shape[0]**2+img.shape[1]**2)

The following code allows configuring the houghCircle function. Here the most important parameters are the maximum and minimum detectable radius and the distances between detected circle. Notice that minRadius and maxRadius could change if the pattern is modified or the disk changes as well. However, the distance between the center of detected circles can be set to the diagonal length of the image. Therefore only one circle will be detected.

circles = cv2.HoughCircles(image=img, method=cv2.HOUGHGRADIENT, dp=0.05, minDist=imagdiagonallength, param1=110, param2=39, minRadius=int(np.floor(imgheigth/4)/4), maxRadius=int(np.floor(imgheigth/8)))

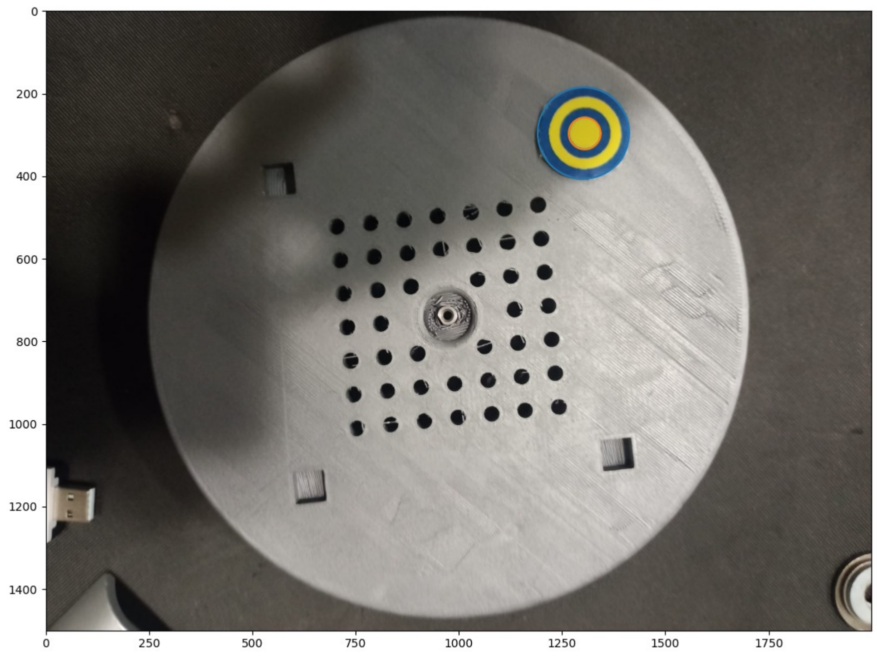


Figure 3.20: Auto shape detected succesfully

Figure 3.20 shows how the algorithm works perfectly if the parameters are designed in the proper way. The center of the target or pattern can be obtained and the next step is the transform that center into a usable template bbox ready to be tracked. In blue, the circle is automatically found. In orange a circle was introduced manualy in order to know the dimensions of the template in pixels. This was done in order to determine the radius threshold.

3.3. Trackers

Nearly a decade has passed since the first time a correlation filter method was used for object tracking purposes. The first correlation filter used in tracking field was the minimum output sum squared error (MOSSE). This filter is easy to design and can track fast objects. Although, it does not guarantee to track accurately when the object tracked changes in shape or dimensions.

MOSSE algorithm uses the frequency domain in 2D to preprocess a filter able to generate one peak at the center location of the moving object. Thus, fast Fourier transform (FFT) is used to avoid correlation time-consuming computation in the time domain.

$$g = f \otimes h \tag{3.26}$$

The main idea of MOSSE algorithm is to find the correlation peak of the input signal with the filter template [11].

$$F(g) = F(f \otimes h) = F(f)F(h)^* \tag{3.27}$$

$$G = FH^* \tag{3.28}$$

Taking benefit of the convolution property in the frequency domain as a dot product the next task of the algorithm is to find template transfer function H. While tracking it is important to consider the influence of factors such as the appearance of the object with the selected template. Therefore, considering m images of the desired object to track can improve accuracy and robustness to the tracking failure. The MOSSE model proposed tries to minimize the output sum of the squared error.

$$S = min \sum_i |H^* F_i - G_i|^2 \tag{3.29}$$

To minimize the solution, the derivatives are calculated and set to zero:

$$0 = \frac{\partial S}{\partial F} = 2\sum_i |H^* F_i - G_i| H \tag{3.30}$$

$$0 = \frac{\partial S}{\partial H} = 2\sum_i |H^* F_i - G_i| F \tag{3.31}$$

$$0 = \frac{\partial S}{\partial G} = -2\sum_i |H^* F_i - G_i|^2 \tag{3.32}$$

taking

$$\frac{\partial S}{\partial H} = 2\sum_i |H^* F_i - G_i| F = 2\sum_i |F_i F_i^* H^* - G_i F_i^*| = 0 \tag{3.33}$$

The solution obtained is the following:

$$H^* = \frac{\sum_i G_i^* F_i}{\sum_i F_i F_i^*} \tag{3.34}$$

The algorithm tracks the object by correlation filters in the next “unknown window” from the next frame. The next position of the bbox or template is located at the correlation peak. Thus, the output’s maximum is the center of the new template. Furthermore, for future frames, it performs a template filter update [12].

$$H_i^* = \frac{A_i}{B_i} \tag{3.35}$$

$$A_i^* = \eta G_i \cdot F_i^* + (1-\eta)A_{i-1} \tag{3.36}$$

$$B_i^* = \eta F_i \cdot F_i^* + B_{i-1} \tag{3.37}$$

The PSR value is used for failure detection

$$PSR = \frac{peak - \mu}{\sigma} \tag{3.38}$$

Steps:
1) The user crops a template from the first frame.
2) The template obtained from the previous step is preprocessed and transformed to the Fourier domain.
3) A synthetic target G is created as

$$g_i = \sum_i \exp \frac{x^2+y^2}{\sigma^2} \tag{3.39}$$

4) The values F of the template and G are substituted to obtain N1 and D1 from the first frame of the video.

$$A_1 = G_i F_i^* \tag{3.40}$$

$$B_1 = F_i F_i^* \tag{3.41}$$

5) Step 1 through 4 are repeated for the next m frames (typical values of m are 6 to 10) Nm and Dm are used to compute the final template filter H.
6) The tracker proceeds to the m+1 frame and retrieves the tracking window where its center is equal to the last frame of the filter initialization (m frame). The tracking window is preprocessed and transformed to the Fourier domain F. The final preprocessed tracking window in the frequency domain is correlated with the conjugate of the filter resulting in the output G where the maximum peak determines the new position of the window in the next frame. In other words, it determines the maximum correlation between the generated filter from a cropped image and the moving object to track.
7) The previous step is used to track the future consecutive frames in which the object will be located but at different pixel positions. However, it is so important to study this algorithm for circular tracking purposes because it is limited by the dimensions of the crop template. The object will be tracked as long as the entire object remains inside the tracking window.

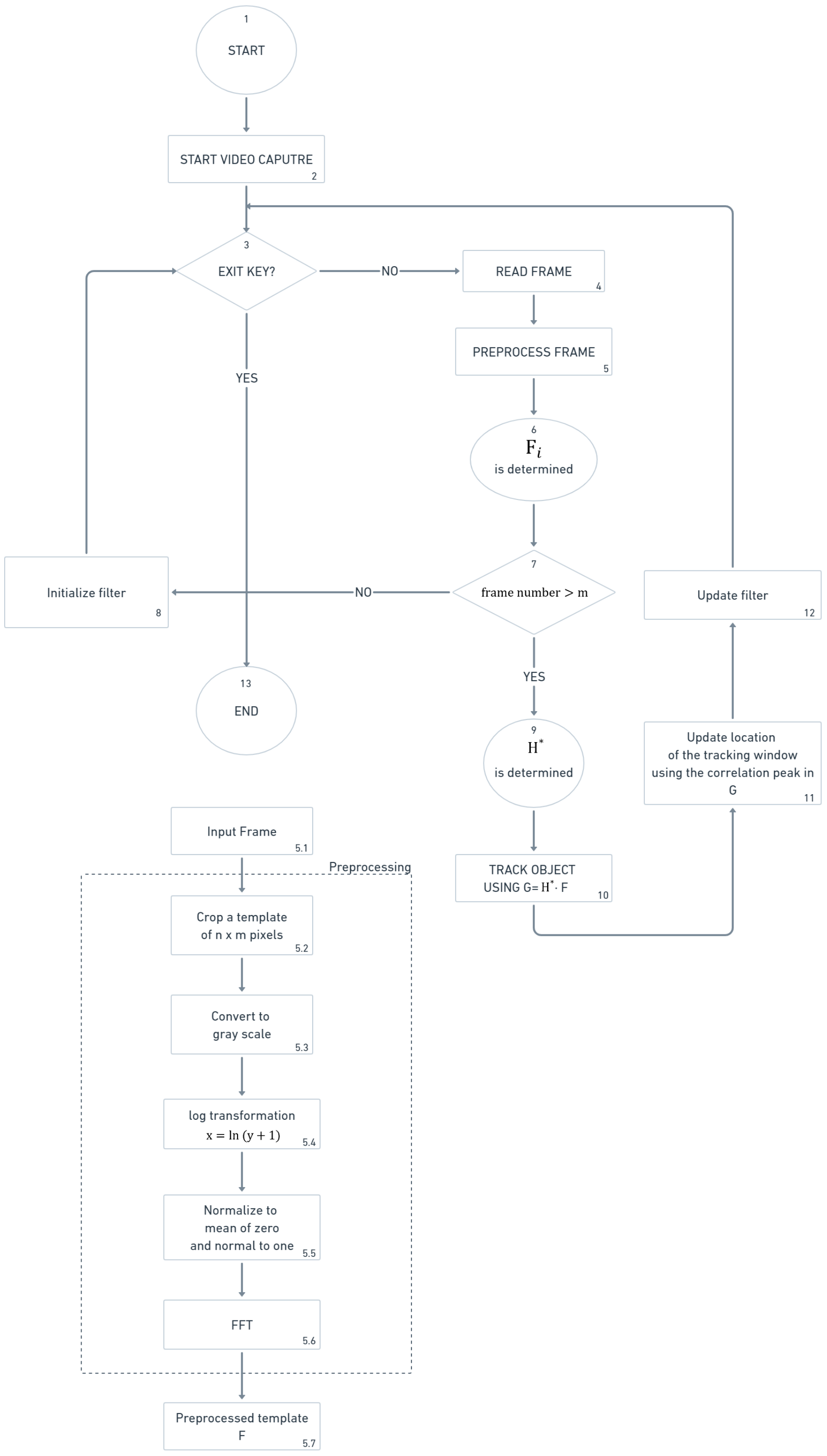
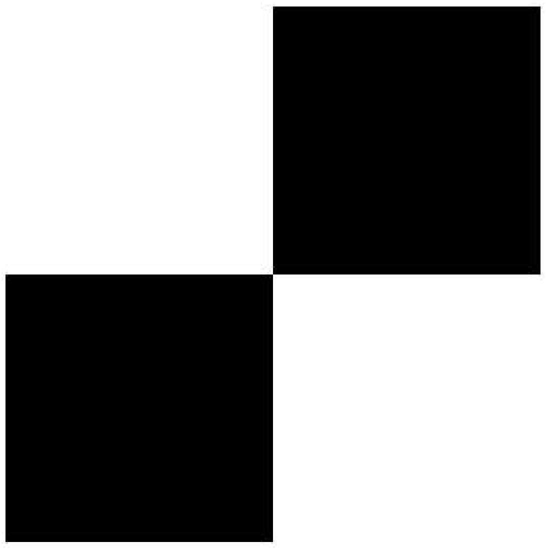
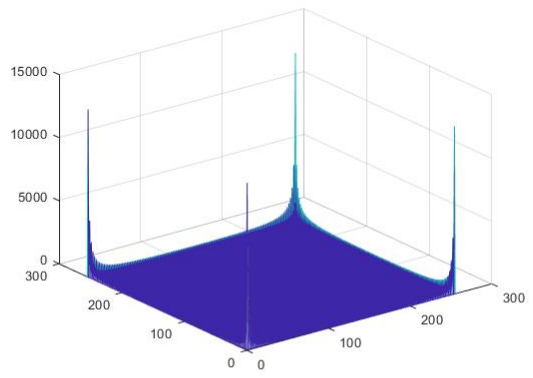

1 START
START VIDEO CAPUTRE 2
3 EXIT KEY?
NO
YES
READ FRAME 4
PREPROCESS FRAME 5
6 F_i is determined
7 frame number > m
NO
YES
Initialize filter 8
13 END
9 H^* is determined
TRACK OBJECT USING G= $H^* \cdot F$ 10
Update location of the tracking window using the correlation peak in G 11
Update filter 12
Input Frame 5.1
Preprocessing
Crop a template of n x m pixels 5.2
Convert to gray scale 5.3
log transformation $x = \ln(y+1)$ 5.4
Normalize to mean of zero and normal to one 5.5
FFT 5.6
Preprocessed template F 5.7

Figure 3.21: MOSSE flowchart

3.4. Tracker selection, patterns and tests.

Before analyzing the trackers, it is mandatory to select a pattern that will be set in the disc to track. This pattern will be selected during the angular velocity algorithm as a crop window or template, and it will be tracked until the end of the process. Therefore, the pattern must fulfill tracking speeds and accuracy minima. Since it is designed to operate for nanosatellite testing scenarios, some considerations have to be taken into account to obtain correct performances. In this section, some patterns are proposed and pros and cons are commented to choose the correct ones for our application.


(a) Square 2x2 chess pattern



(b) 2x2 sqaures FFT

Figure 3.22: Basic pattern analysis

It is very usual to analyze nxn chess patterns in image processing since it is useful to understand frequency components of the FFT of an image and some general features. So chess table could be a good candidate to be tracked since is very intuitive in the frequency spectrum domain and easily analyzed. 2D FFT has been used for the chess pattern applying the fftshift(). Figure 3.23 shows the star pattern obtained.

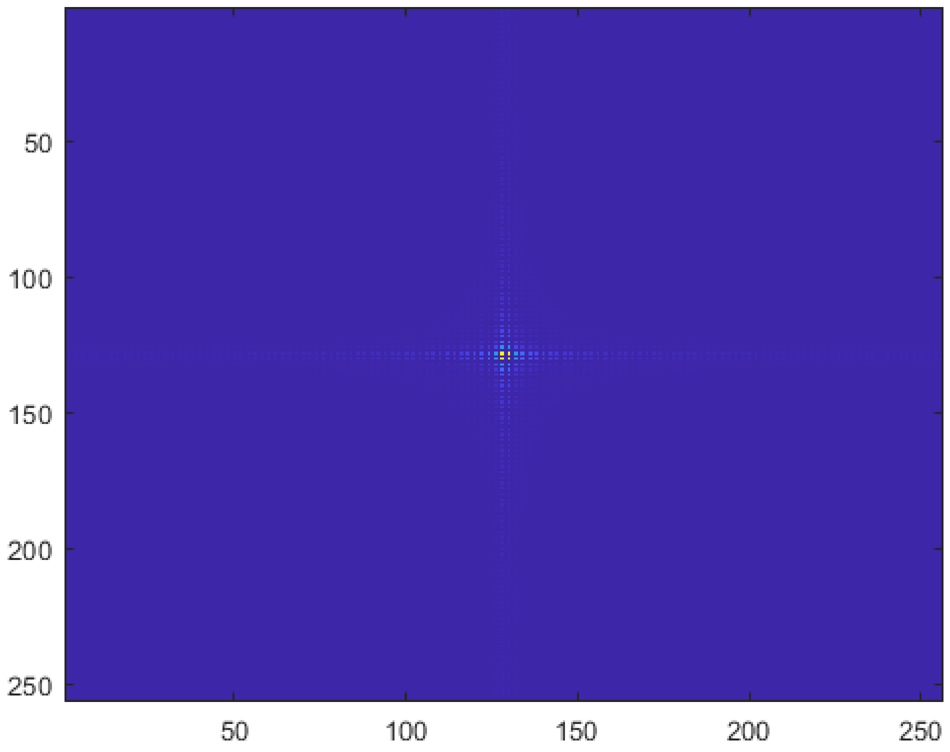
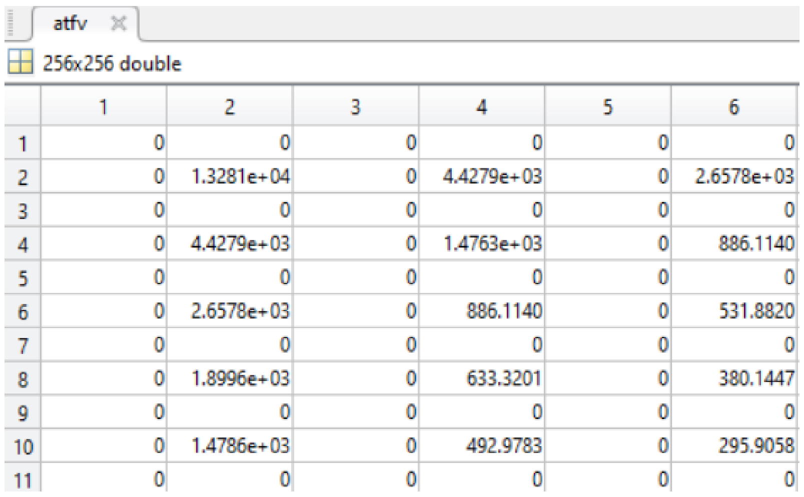


Figure 3.23: 2D FFT of the chess pattern using fftshift()

If the 2D FFT is analized a pattern can be observed in the even frequencies. Thus, just even frequencies contain magnitudes values different from zero. Furthermore, the DC component of the FFT result null. This can be observed numericaly in Figure 3.24


atfv

256x256 double

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	1.3281e+04	0	4.4279e+03	0	2.6578e+03
3	0	0	0	0	0	0
4	0	4.4279e+03	0	1.4763e+03	0	886.1140
5	0	0	0	0	0	0
6	0	2.6578e+03	0	886.1140	0	531.8820
7	0	0	0	0	0	0
8	0	1.8996e+03	0	633.3201	0	380.1447
9	0	0	0	0	0	0
10	0	1.4786e+03	0	492.9783	0	295.9058
11	0	0	0	0	0	0

Figure 3.24: First frequencies for the FFT 2x2 chess

Therefore, the transversal cut of the FFT at the origin towards one given axis(notice the symmetry) has been obtained. It can be observed in figure 3.25 how it exists an even parity in the peaks. indeed, it is interesting the origin DC null value.

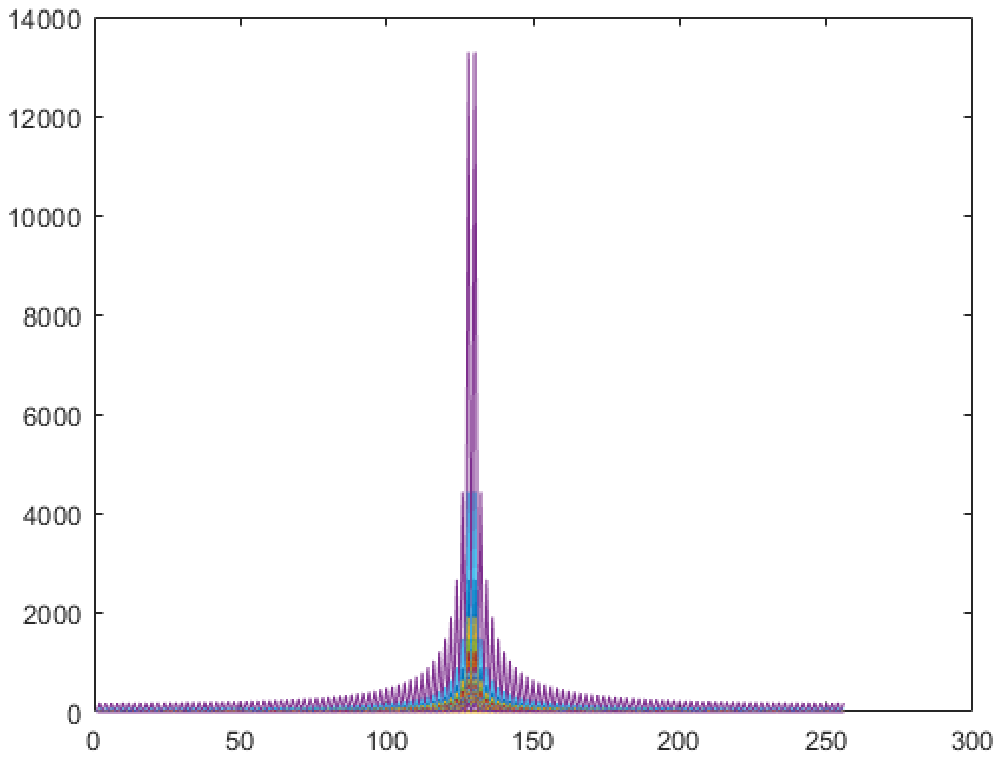


Figure 3.25: FFT chees pattern evaluated at the origin (f1=0)

Notice that the value of the FFT at the position (0,0) can be explained using the DFT definition. Matlab uses (0,0) as (1,1).

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)exp(0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \tag{3.42}$$

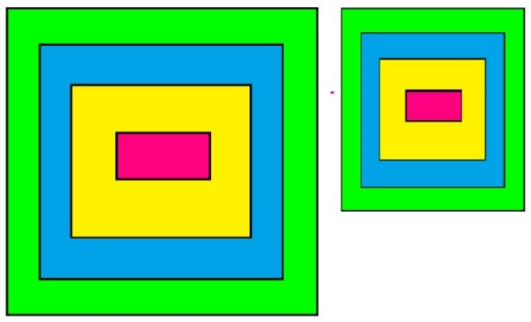
Since the cropped image has been preprocessed and its mean value has been changed to 0, the values at the origin are the DC component of the FFT. So the mean is zero. Thus the DC value is zero too.

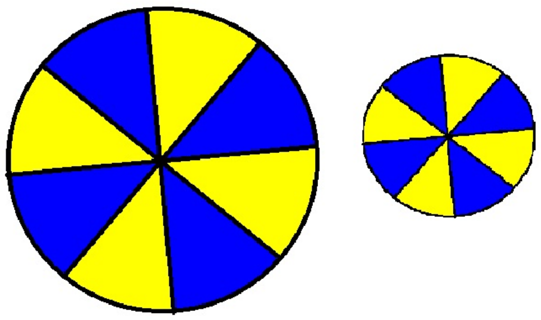
Matlab can perform FFT and graphs of the analysis easily as follows:

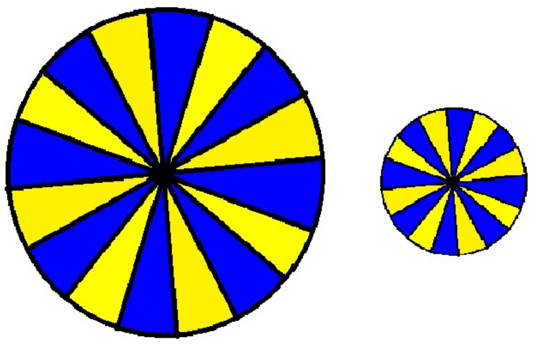
```
close all;
clear all;
uones=ones(128,128);
uzeros=zeros(128,128);
u=[uones uzeros ; uzeros uones];
figure(1)
imshow(u)
v=u-mean(mean(u));
```

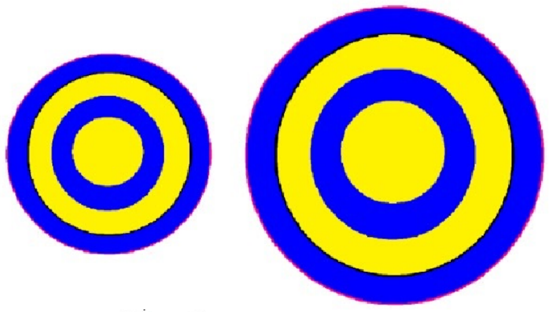
```
figure(2)
imshow(v)
atfv=abs(fft2(v));
figure(3)
meshz(atfv);
figure(4)
plot(atfv)
colorbar
imgfftgrey = abs(fftshift((fft2(v))));
plot((imgfftgrey))
```

However, this pattern is not useful for non-rotatory trackers. Although they exist and are implemented, for non-complex computational trackers, other solutions have been taken into account.


(a) Square pattern


(b) cicle1 pattern


(c) cicle2 pattern


(d) cicle symetric pattern

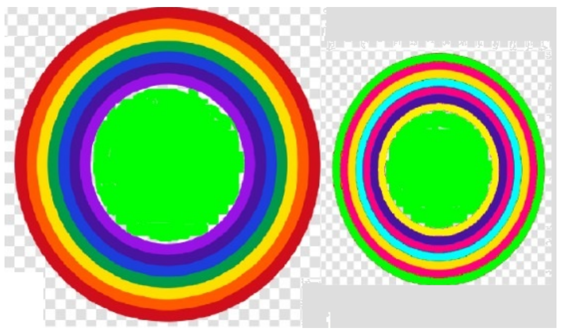
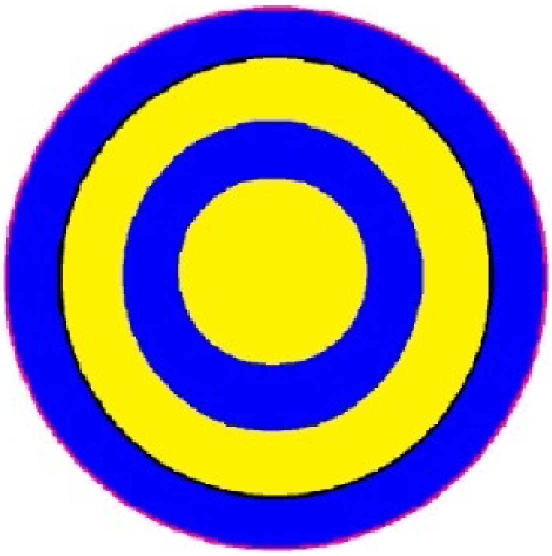
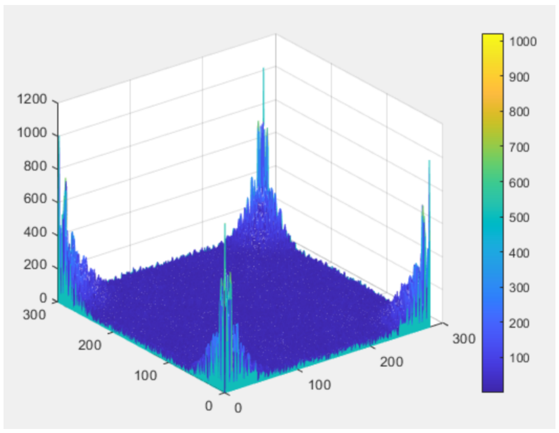

(e) circle colors pattern

Figure 3.26: Tracking patterns used

Figure 3.31 shows some samples that have been tested to obtain a pattern that fulfills the basic tracking features for nanosatellite testings. Although these patterns can be used for other industrial applications, some other considerations should be taken into account. For this main application, the most important feature is symmetry. Since the object will be rotating circularly, the window must be somehow symmetric. Therefore, the first option is to use squares or rectangles as objects to track. Figure (a) is the first studied model. However, while tracking this object with a fitted window (bbox), the trackers lose the tracking (PSR is not enough). The reason why the squares are not a correct candidate is because they do not contain circular symmetry. Other colors or dimensions have been considered. Indeed, it is not the easiest option to find a square pattern to track. (b) and (c) are the second considered option. This pattern could be tracked but has the flaw that in the frequency domain some rotations occur since the tracking window cannot rotate for the proposed trackers. Thus, it is not useful for high speeds since the tracking is easily lost due to the non-symmetry. (c) is the solution to increase the efficiency of the pattern, but still, some other possibilities could be used. (d) and (e) are the best solution found. Concentric circles have circula symmetry and the pattern does not depend on the position at each time step inside the disk. Therefore, the pattern does not rotate and the correlation is faster and more exact.

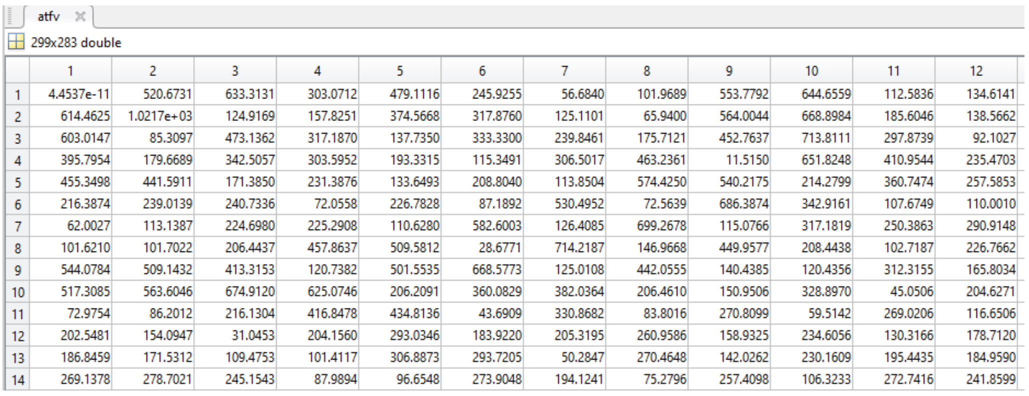

(a) Concentric circles pattern



(b) Concentric circles FFT

Figure 3.27: Circle pattern and FFT

Notice that the same phenomenon seen in Figure 3.31 happens in the pattern. The value at DC or origin is almost zero. And the FFT is the spectral function used at each loop to correlate the image with the newly moved object.



atfv

299x283 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	4.4537e-11	520.6731	633.3131	303.0712	479.1116	245.9255	56.6840	101.9689	553.7792	644.6559	112.5836	134.6141
2	614.4625	1.0217e+03	124.9169	157.8251	374.5668	317.8760	125.1101	65.9400	564.0044	668.8984	185.6046	138.5662
3	603.0147	85.3097	473.1362	317.1870	137.7350	333.3300	239.8461	175.7121	452.7637	713.8111	297.8739	92.1027
4	395.7954	179.6689	342.5057	303.5952	193.3315	115.3491	306.5017	463.2361	11.5150	651.8248	410.9544	235.4703
5	455.3498	441.5911	171.3850	231.3876	133.6493	208.8040	113.8504	574.4250	540.2175	214.2799	360.7474	257.5853
6	216.3874	239.0139	240.7336	72.0558	226.7828	87.1892	530.4952	72.5639	686.3874	342.9161	107.6749	110.0010
7	62.0027	113.1387	224.6980	225.2908	110.6280	582.6003	126.4085	699.2678	115.0766	317.1819	250.3863	290.9148
8	101.6210	101.7022	206.4437	457.8637	509.5812	28.6771	714.2187	146.9668	449.9577	208.4438	102.7187	226.7662
9	544.0784	509.1432	413.3153	120.7382	501.5535	668.5773	125.0108	442.0555	140.4385	120.4356	312.3155	165.8034
10	517.3085	563.6046	674.9120	625.0746	206.2091	360.0829	382.0364	206.4610	150.9506	328.8970	45.0506	204.6271
11	72.9754	86.2012	216.1304	416.8478	434.8136	43.6909	330.8682	83.8016	270.8099	59.5142	269.0206	116.6506
12	202.5481	154.0947	31.0453	204.1560	293.0346	183.9220	205.3195	260.9586	158.9325	234.6056	130.3166	178.7120
13	186.8459	171.5312	109.4753	101.4117	306.8873	293.7205	50.2847	270.4648	142.0262	230.1609	195.4435	184.9590
14	269.1378	278.7021	245.1543	87.9894	96.6548	273.9048	194.1241	75.2796	257.4098	106.3233	272.7416	241.8599

Figure 3.28: 2D FFT of the template pattern

2D FFT shown in figure 3.28 has circular properties. It seems to contain sinusoidal behaviors at 45, 135, 225 and 315 degrees. This frequencies patterns are related with the symmetry of the original image. However,

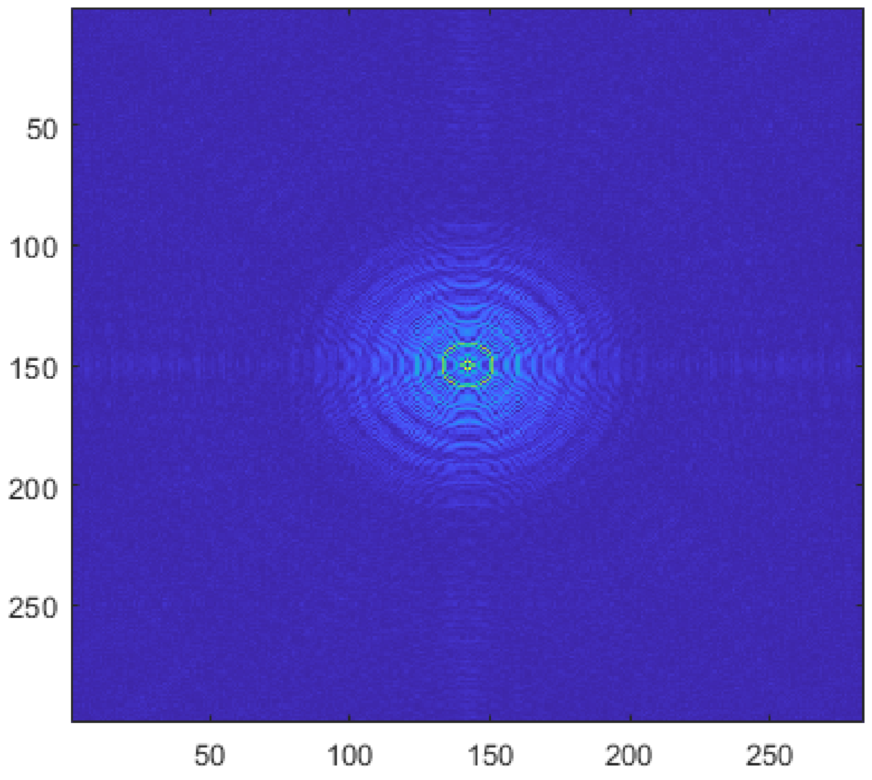


Figure 3.29: FFT of the template pattern

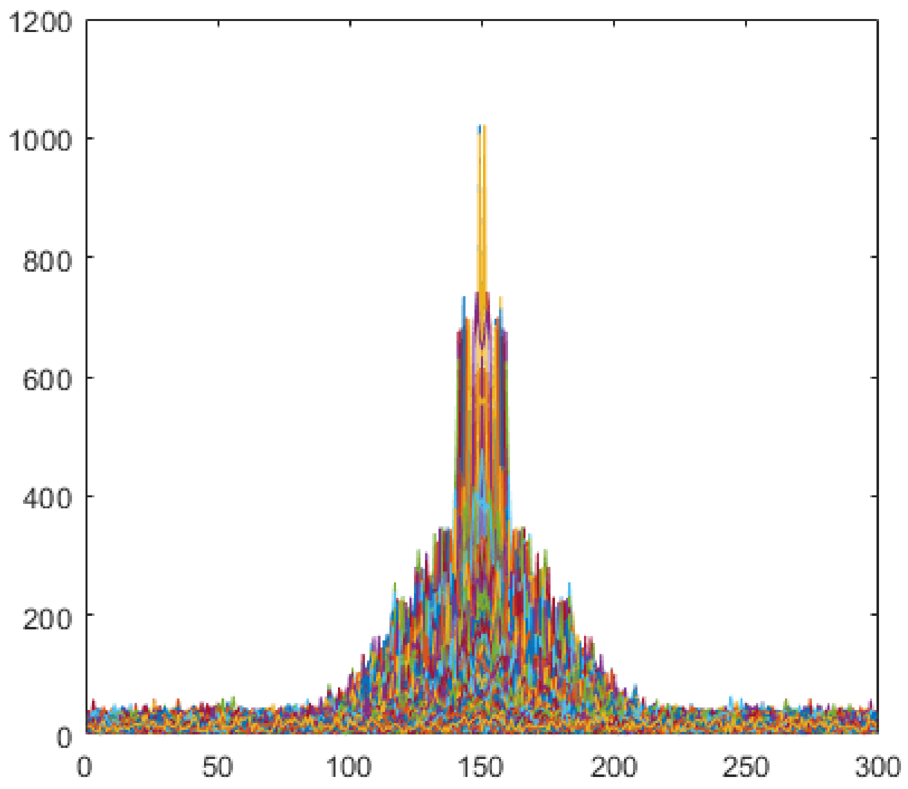


Figure 3.30: First frequencies for the FFT circle

However, how important is to find a template perfectly fitted with the pattern? Indeed, if the Bbox selected is not a perfectly fitted square with the patter the results on tracking can vary significantly.

(a) template inside the pattern

(b) template outside the pattern

Figure 3.31: Circle pattern and FFT

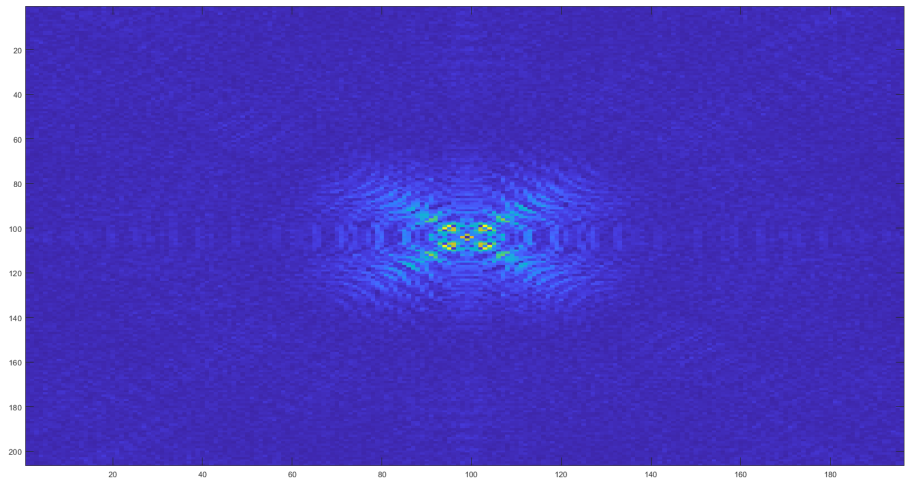


Figure 3.32: 2D FFT of 3.31(a)

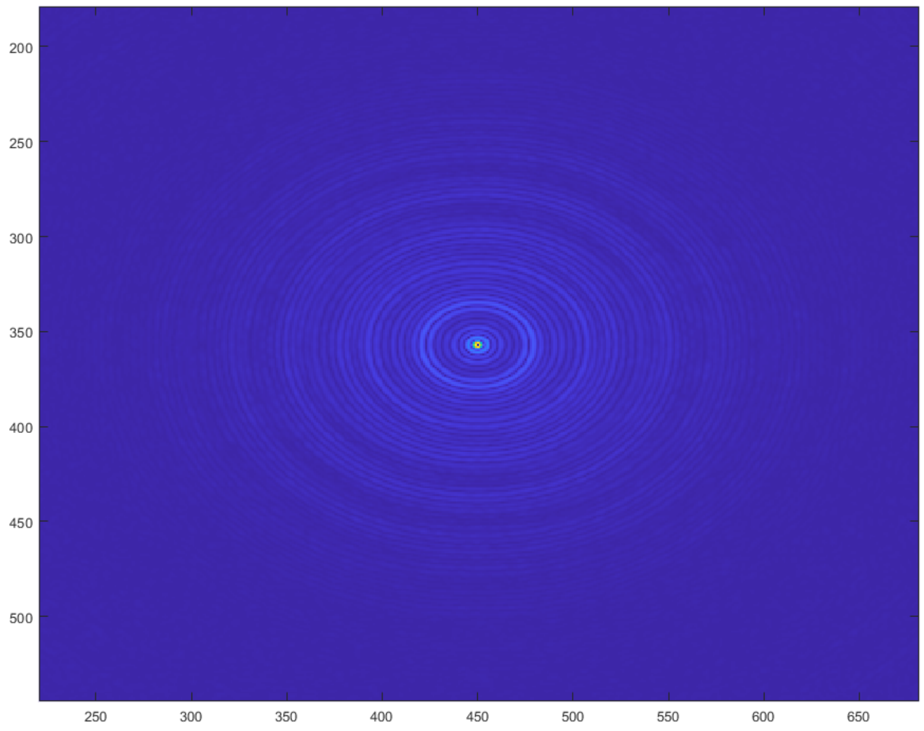
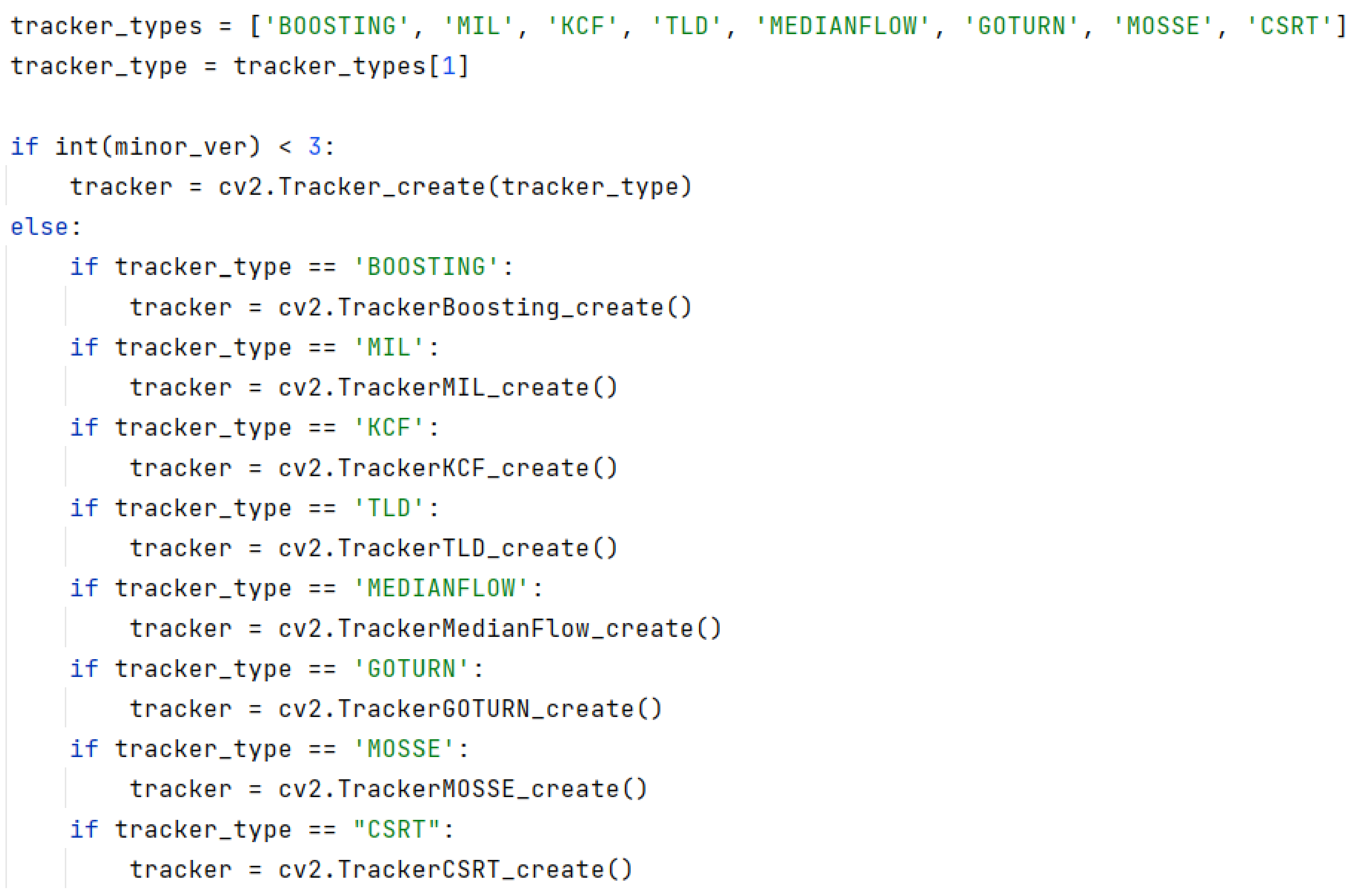


Figure 3.33: 2D FFT of 3.31(b)

It is observed that the template dimensions are important as well as the pattern. If the template is inside the pattern, the FFT will not have perfect symmetry due to the corners. Notice how in 3.31(a) the blue corner is similar to the perfect fitted template in 3.27(a), however, the FFT created new noisy regions in 45, 135, 225 and 315 degrees. This is because the pattern has lost information in the cut. Figure 3.29 showed that only the regions close to the axis have this disturbancies. On the other hand, if the corners are errased from the original image by selecting a template higher than the pattern dimensions, the disturbances are reduced. Figure 3.33 shows a sinc type frequency response in which the pattern in the freqeuncy domain can be distinguished. This is helpful when doing tracking at high velocities, because patterns in the frecuency domain usually helps analisys and complexity of the computations. Indeed, while doing the pattern preparation for the algorithm, it has observed that taking Bbox templates higher than the dimensions of teh patterns allowed the system to operate at higher velocities. Also rectangular Bbox were tried an the results were positive. However, it exist a trade off between the maximum angular velocity achieved and the accuracy of the data measurements. It results that if the pattern is not correctly fitted, the measurements are more noisy. Indeed, it makes sense since the template dimensions is correlated with the template desplacement at each frame. Larger template dimensions allows the template to have less displacements boosting the tracking hook. Small templates, ideally a pixel, will suffer larger displacements that could measure very accurate but with the flaw of the loss of tracking if the sampling time or FPS are not high enough.

Trackers

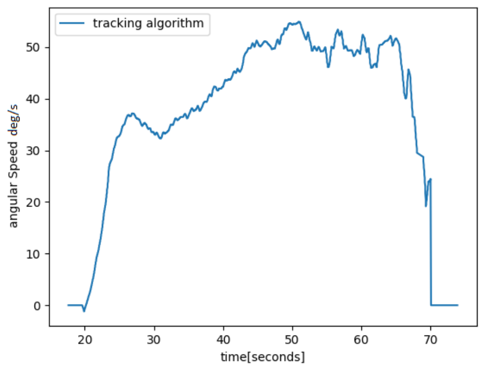
OpenCV library has implemented some tracking algorithm for python: ’BOOSTING’, ’MIL’, ’KCF’, ’TLD’, ’MEDIANFLOW’, ’GOTURN’, ’MOSSE’, ’CSRT’ [17]. Some attempts to get a precise tracking performance have been studied to have accurate mesurements and a correct tracking window of the object in the disk (pattern).



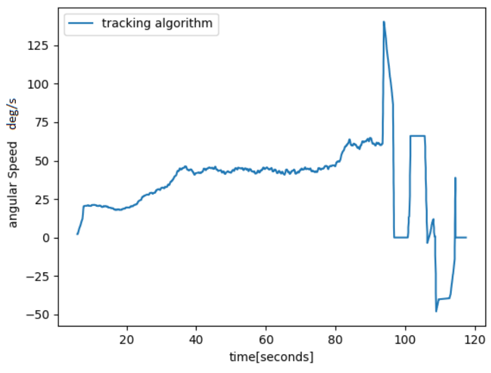
```python
tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
tracker_type = tracker_types[1]

if int(minor_ver) < 3:
    tracker = cv2.Tracker_create(tracker_type)
else:
    if tracker_type == 'BOOSTING':
        tracker = cv2.TrackerBoosting_create()
    if tracker_type == 'MIL':
        tracker = cv2.TrackerMIL_create()
    if tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
    if tracker_type == 'TLD':
        tracker = cv2.TrackerTLD_create()
    if tracker_type == 'MEDIANFLOW':
        tracker = cv2.TrackerMedianFlow_create()
    if tracker_type == 'GOTURN':
        tracker = cv2.TrackerGOTURN_create()
    if tracker_type == 'MOSSE':
        tracker = cv2.TrackerMOSSE_create()
    if tracker_type == "CSRT":
        tracker = cv2.TrackerCSRT_create()
```

Figure 3.34: Trackers list (Python)

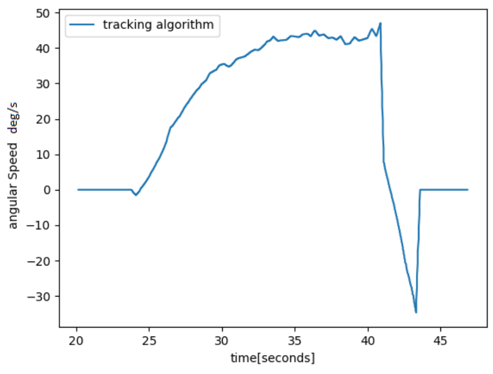
Once the pattern is selected some tests of the algorithm can be performed.



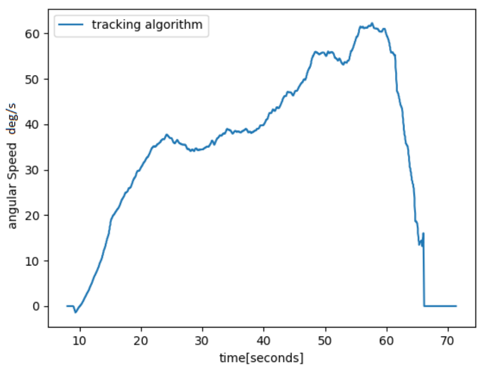
(a) BOOSTING, BlueYellow



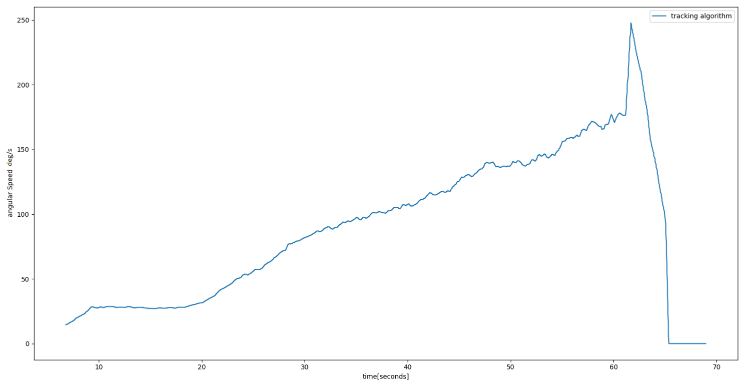
(b) KCF, BlueYellow



(c) MOSSE, BlueYellow



(d) MIL, BlueYellow



(e) CSRT,BlueYellow

Figure 3.35: Tracking lose analisys

It is so important to focus on the pattern that the main reason why the algorithm should work is the dimensions and shape of the pattern. However, trying some patterns it is easily proved that colors do matter. The MOSSE algorithm explained in the past section used intensity preprocessing steps which now can be verified.

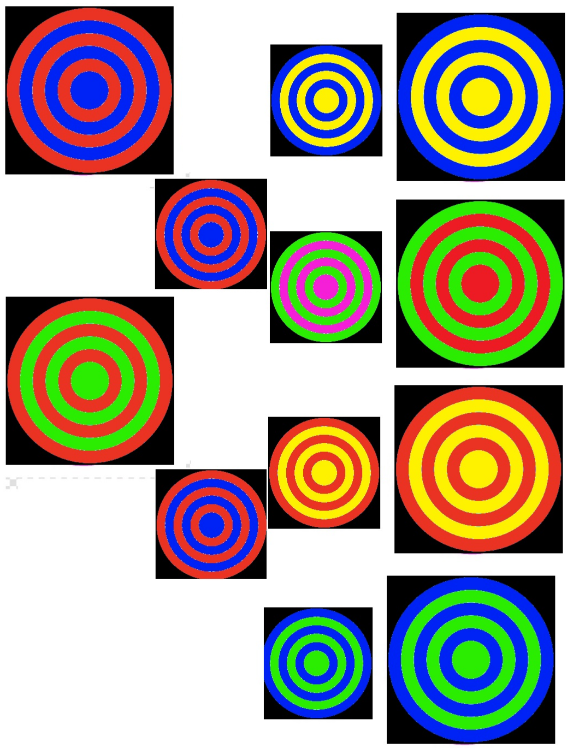


Figure 3.36: Circle patterns colors tested

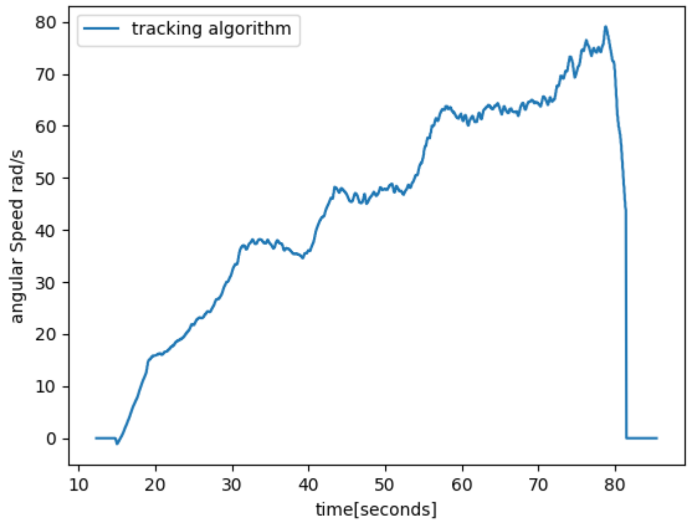
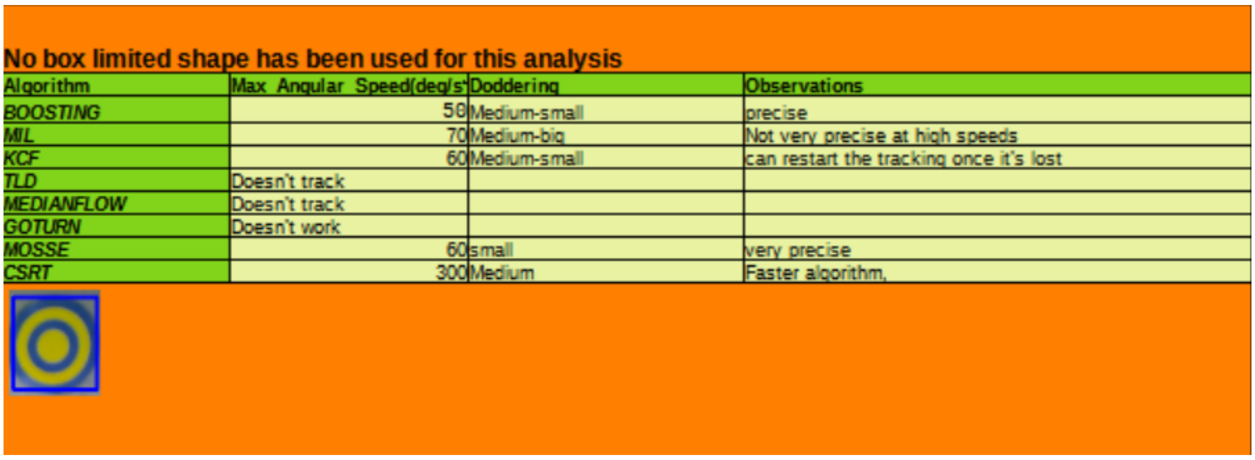


Figure 3.37: BOOSTING RedYellow

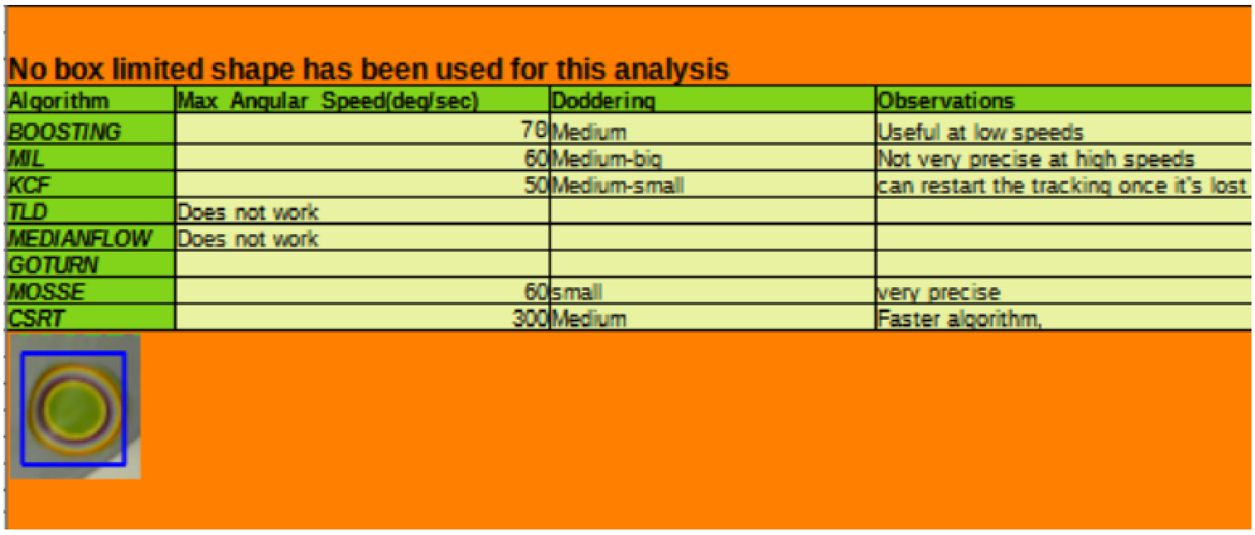
In this last image, the circular pattern with red and yellow colors was used and the tracker could be improved in speed up to 20 degrees/second compared with the BOOSTING Blue Yellow of the Figure 3.38 In conclusion, it seems that colors with high contrast enhance the performance allowing us to find the best correlations in real-time.



No box limited shape has been used for this analysis

Algorithm	Max Angular Speed(deg/s	Doddering	Observations
BOOSTING	50	Medium-small	precise
MIL	70	Medium-big	Not very precise at high speeds
KCF	60	Medium-small	can restart the tracking once it's lost
TLD	Doesn't track		
MEDIANFLOW	Doesn't track		
GOTURN	Doesn't work		
MOSSE	60	small	very precise
CSRT	300	Medium	Faster algorithm,

Figure 3.38: Performance analysis for pattern 1



No box limited shape has been used for this analysis

Algorithm	Max Angular Speed(deg/sec)	Doddering	Observations
BOOSTING	70	Medium	Useful at low speeds
MIL	60	Medium-big	Not very precise at high speeds
KCF	50	Medium-small	can restart the tracking once it's lost
TLD	Does not work		
MEDIANFLOW	Does not work		
GOTURN			
MOSSE	60	small	very precise
CSRT	300	Medium	Faster algorithm,

Figure 3.39: Performance analysis for pattern 2

As results obtained, it has been observed a phenomenon so critical for improving accuracy. Doddering is the result of a not having frame per second enough and also a very sensitive tracker. Doddering can be studiend when the object to track is static. This doddering in the template is translated into noise when the tracking algorithm starts. One way to solve the problem is to increase camera resolution and FPS. Thus, those trackers that do not have too much doddering are better candidates to be used for the algorithm. Summing up, MOSSE and CSRT are the chosen candidates to be used. Furthermore, depending on the specific application, it exists a trade-off between velocity and accuracy. MOSSE tracker shows a really good precision at velocities lower than 50 deg/s. However, it can easily lose track if the speed increases by 60 deg/s. On the other hand, the CSRT tracker showed a unique property that can be useful for critical scenarios. Velocities up to 300 deg/s can be tracked using this tracker. Unfortunately, at those high speeds, the doddering effect is considered to deteriorate the accuracy of the measurements. Notice that the pattern affected the maximum tested speeds but in general it did not affect the general properties of each track as doddering or precision. Using a correct selected pattern the maximum angular velocities reached were increased by 10 deg/s at MIL and KCF for example. This shows us how a correct pattern is not critical to the whole performance of the system but it can enhance considerably the efficiency and capabilities of the whole algorithm.

3.5. Correlation filter trackers limitation and FPS.

Figure 3.40: Camera used: Logitech - P/N: 960-001063

The camera used for the realization of the algorithm and analysis is the Logitech - P/N: 960-001063. It is indeed a simple camera, although it is proved that the algorithm is performed correctly even though the camera hardware properties are not the best. The main feature to highlight of the camera is the Frames Per Second capacity. This one is a 30FPS camera. It is proven that this type of camera allows a proper system analysis. For more precise tracking and accurate samples, some parameters of the camera can be upgraded.

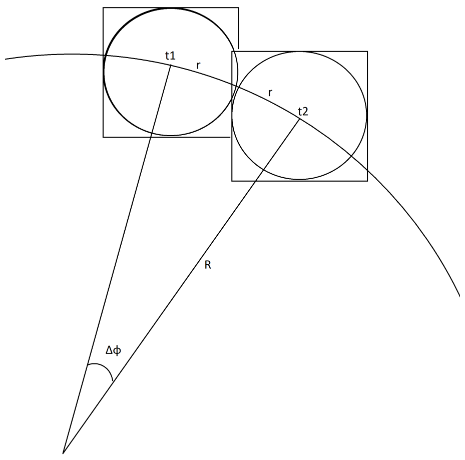


Figure 3.41: Scenario proposed at each sample time

Figure 3.44 shows a similar scenario that happens while tracking in real-time. Imagine the window template is filling the circular pattern. While the object is moving, the tracker will perform FFTs and find the correlation peaks as explained in the past section. Let's focus on the boundary conditions: If the object is not moving, the tracker must find the correlation peak at the center of the window since the template is almost the same as the next frame image. On the other hand, if the object is moving so fast, which is the maximum speed that the object can reach without losing the tracking? Although correlation trackers diverge from the main MOSSE explained. Those other correlation algorithms are just upgrades of this same explained idea. Thus, the Fourier transformation can easily answer this question. Qualitatively, the tracking will be lost when the original object is not found anymore and the next frame. So this happens when the previous template is tangent to the next window to correlate. At the end of this section a simplified model has been explained, however, it is useful to know some parameters' performances.

From the circle we know the length can be expressed as follows:

$$R(\frac{\Delta\theta}{2}) = r \tag{3.43}$$

Where we assume that the diagonal (2r) approximated to the length of the center of two consecutive frames. This can be assumed if the pattern is considerably smaller than the disk. So it starts losing tracking when the circle loses its geometry properties, at half the square.

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{3.44}$$

$$\Delta\theta_{max} = \omega_{max}\Delta t_{min} \tag{3.45}$$

$$FPS = \frac{Num.of frames}{time} = \frac{frames}{t} \tag{3.46}$$

For this camera FPS = 30, Thus,

$$t_2 - t_1 = t = \frac{1}{30} \tag{3.47}$$

For the platform choosen, the ration of circle is 2:11.5

$$\Delta\omega_{max} = \frac{r}{R}FPS = \frac{2}{11.5}30 = 299 deg/sec \tag{3.48}$$

Indeed, the maximum angular velocity reached with the circular blue and yellow pattern was around 300 deg/s as figure 3.42 shows. Notice that at those high velocities the variance of the signal is considerably high. However, the angular measurements can be measured by computing the mean. The peaks observed show how the tracking is lost in that exact frame but recovered in the next frames since the measures are obtained around a constant angular velocity. The tracking used for this max angular velocity was CSRT and can be observed that the algorithm is at the highest capabilities. Figure 3.35 (e) shows that the algorithm can measure accurately at low angular speeds but at higher velocities, the variance is significantly larger.

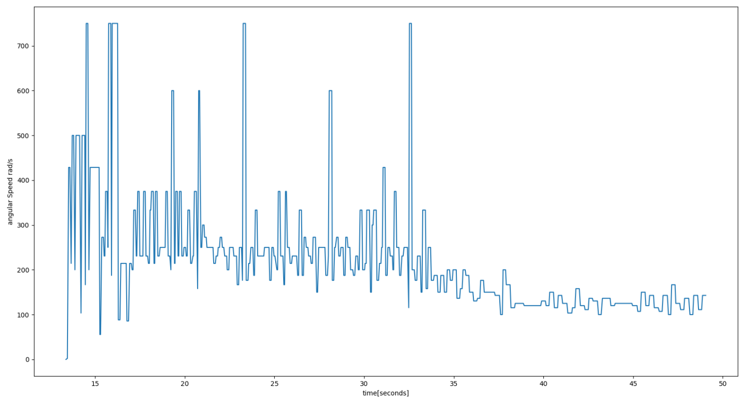


Figure 3.42: Maximum velocities reached

Results:

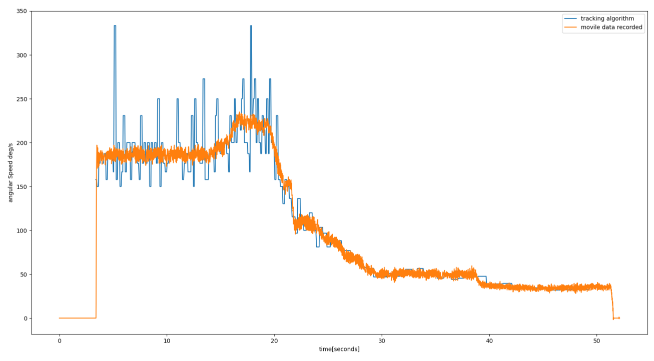


Figure 3.43: Tracking algorithm comparison and testing

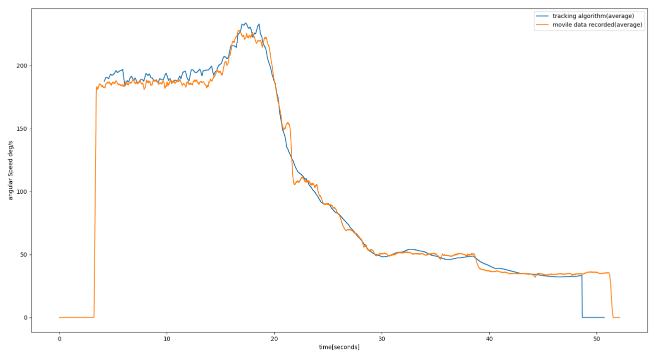
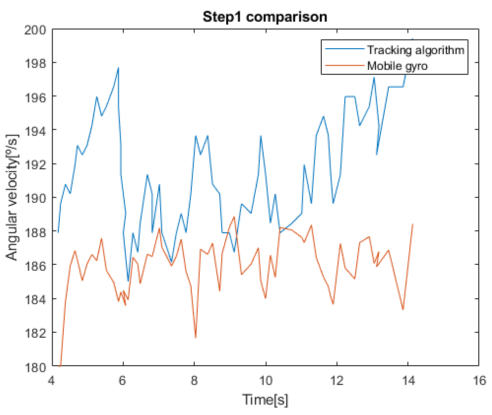


Figure 3.44: Tracking algorithm comparison and testing (moving average)

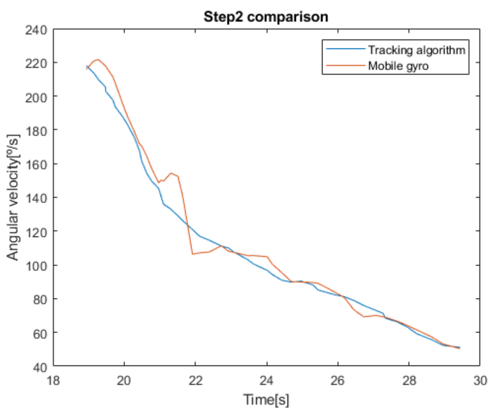
The experiment was done by taking a phone with a gyroscope and the tracker algorithm independently. With a power supply to change the current along the servo of the platform.

Set the phone in the middle of the platform (disk) and calibrate the tracking algorithm. When the system is ready to operate, start at the same time the tracking algorithm and the recording data of the mobile phone gyroscope. The next step is to switch on the power supply and change voltages to obtain different angular velocities to compare. When some data is collected both phone and code can stop. Notice that in the plot the mobile was the last one to stop. After both data set is stored, they can be processed in the same plot and compared as in the case. The last step is to apply a moving average to both data sets to compare the results.

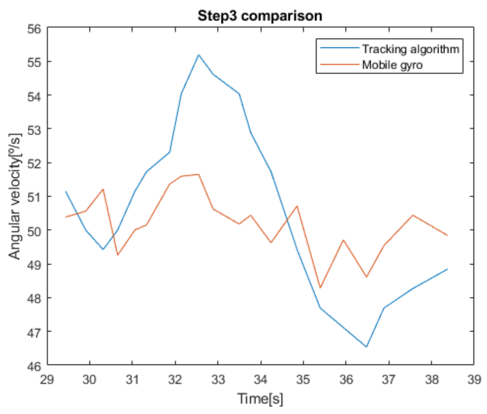
The results obtained show that the algorithm is ready to operate correctly even at larger angular speeds as we can see. Notice that the experiment was done with a cheap camera as explained. Still, the graphs are so close to each other allowing us to obtain good enough quality of the measurements. The data obtained allows to see qualitatively the efficiency of the new system, therewith an quantitative analisys has been performed. The tracking process captures has been splited into 4 steps: step1 at constant high angular velocity, step2 in the deacreasing slope and step 3 and step 4 at 2 different lower angular velocities. See appendix E



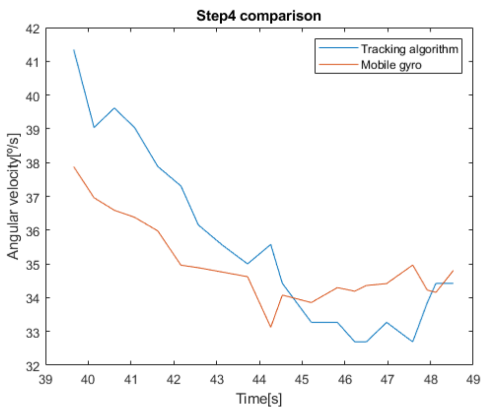
(a) Step 1



(b) Step 2



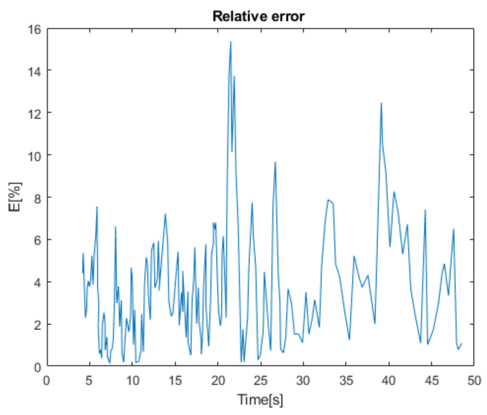
(c) Step 3



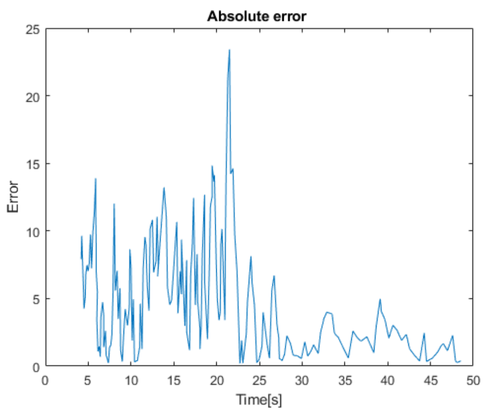
(d) Step 4

Figure 3.45: Overall error analysis

The whole relative and absolute error in the measurement has been computed in figure 3.46

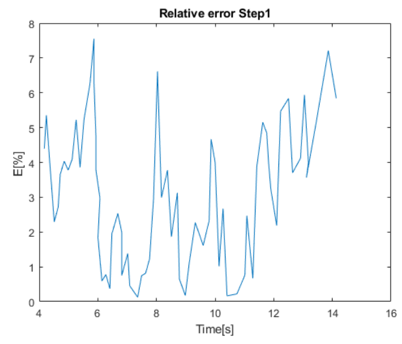


(a) Total relative error

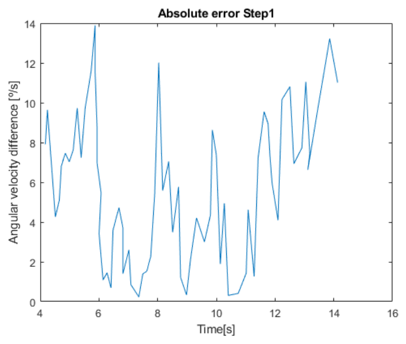


(b) Total absolute error

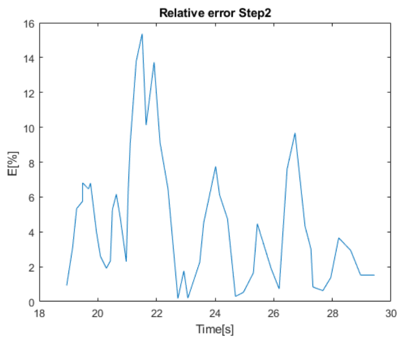
Figure 3.46: Tracking lose analisys



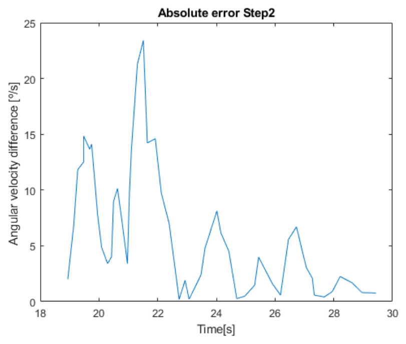
(a) Step 1



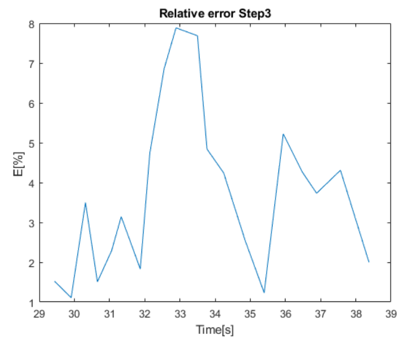
(b) Step 1



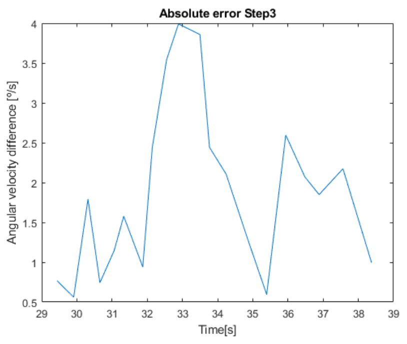
(c) Step 2



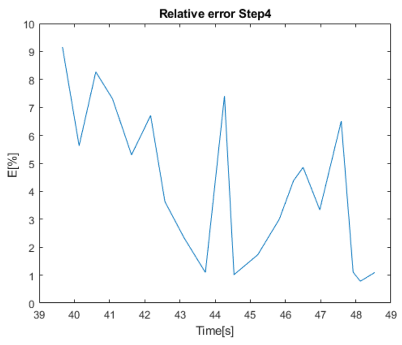
(d) Step 2



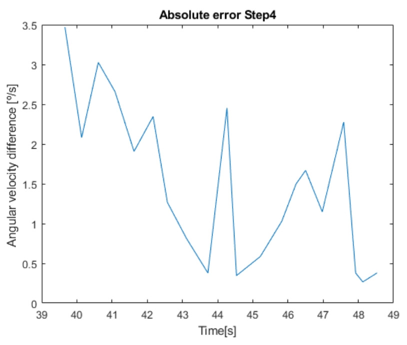
(e) Step 3



(f) Step 3



(g) Step 4



(h) Step 4

Figure 3.47: error analysis by steps

Table 3.1: Experiment results

		Step1	Step2	Step3	Step4	Total
Variance	**Tracking**	10.5225	2.4551e+03	6.8180	6.8670	4.7867e+03
	Mobile	3.1884	2.8751e+03	0.8113	1.4020	4.5789e+03
Covariance		-0.1062	2.6367e+03	1.5354	2.6532	4.6628e+03
Correlation		-0.0183	0.9924	0.6528	0.8551	0.9960
Error	**Relative [%]**	3.1282	4.6434	3.7228	4.2322	3.7734
	Absolute	5.7950	6.2704	1.8740	1.4986	5.0857

Finally,the mean of the errors have been introduced into a datatable in order to sort all the collected data. Variance and covariance has been computed in order to see how are the signal features.

This closer analysis allows to know about the performance for different angular velocities and scenarios. Figure 3.45 shows a closer look into each step of the experiment. Notice the scale of each step. Figure 3.46(a) shows the performance at high angular velocities. Variance of both mobile and tracking are high at those speeds. Notice that the tracking algorithm is barely stable at those starting times. Thus, the tracker would operate more efficiently if the constant angular velocity at that rate would stay longer. The absolute mean error is 5.7 degres/second which is considerably low at those high speeds.It means that the difference between the mobile sampling and the tracking algorithm is aproximatelly 6 º/s in average. The algorithm was testes using CSRT tracker which can operate at high angular velocities. However, the test was carried out knowing the noisy that this tracker can be at angular speeds higher than 100 º/s. But still it allows to study and compare the 2 systems at the worst case scenario. Figure 3.45(d) shows the closest correlation of both systems, which at first sight the tracker algorithm seems to work correctly. The slope case in figure 3.46(b) was done in ordered to see the speed of the algorithm when considering changes. This depends on the data sampling selected on the experiment. Figure 3.43 shows a $Ts = 0.1s$, which can be observed that between times 20 - 22 seconds, the tracking algorithm did not measure correctly the change done in the rotatory platform that the mobile did measure. On the contrary, the mobile phone measured this change incorrectly because the change was not that large in reality. Variance and covariances should not be considered for the slope case scenario because it does not make sense to compute the mean change of a almost constant signal if the signal is not constant at all. However, the error obtained is close to the step1 considering the fast change created in the rotatory platform that makes the values considerably worse.Figures refSF:Step3 and refłSF:Step4 have to property that the angular measurements were slower. thus, the error obtained at lower speeds is lower as well. This means that the system is more accurate when the angular velocities are lower. At a mean angular velocity of 50º/s the variance is 6.8º/s which seems not to change considerably at a mean speed of 30º/s. However, the absolute error is lower at 30º/s which allows the algorithm to be more accurate. Notice that the CSRT tracker is not the best suited for lower angular velocities. MOSSE algorithm works much better and the error decreases considerably at speeds lower that 60º/s. From the covariance point of view 3.1, the study shows a negative tendency of the data distribution for step1. Althought it is difficult to interpret covariance meaning, the other steps tend to have anatural positive tendency. Covariance do not measure the strengh in the linearity, it just gives information about the slope of the tendency of both data samples. However, taking the correlation 3.49

value coefficient, the information of how the strength of the slope is for the dependency of the two variable is avaliable.

$$Corr(x_{Mobile}, x_{Tracking})) = \frac{cov(x_{Mobile}, x_{Tracking}))}{\sqrt{var(x_{Mobile}))}\sqrt{var(x_{Tracking}))}} \tag{3.49}$$

Table 3.1 results show a tendency of measure constant values for both systems. Step1 shows a tendency slope close to zero which means that in average each sample of the mobile correspond with a non linear sample of the tracking algorithm, creating a non tendency or a zero slope line in average. For practical purposes it means that both systems are measuring constants angular velocities. Step2 shows a linear correlation close to 1 which means that both data samples decreases at a close rate as can be observed at 3.46(b). Step3 and Step4 should be considered as a transition step, therefore the data did not have time enough to get stabilized, thats the reason why in step 4 the correlation value is 0.8. As can be observed in 3.46(b) the tendency is to increase for the experimental rotatory platform. However, the correlation is a covariance dependet function which is extremely sensitive to scales. This is the reason why the correlation is that large.

3.6. Code explanation

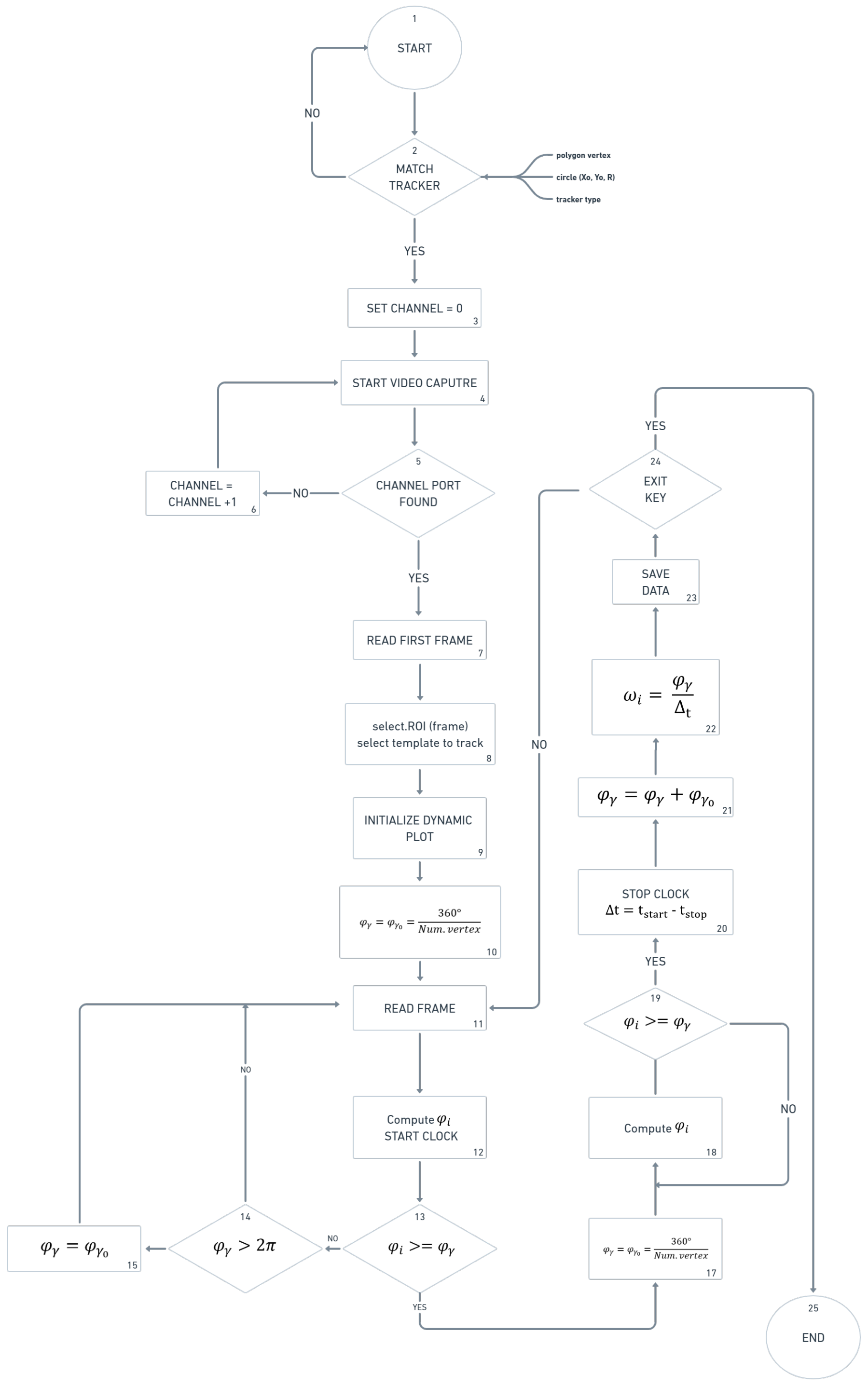

1 START
NO
2 MATCH TRACKER
polygon vertex
circle (Xo, Yo, R)
tracker type
YES
SET CHANNEL = 0 3
START VIDEO CAPUTRE 4
5 CHANNEL PORT FOUND
NO
CHANNEL = CHANNEL +1 6
YES
READ FIRST FRAME 7
select.ROI (frame) select template to track 8
INITIALIZE DYNAMIC PLOT 9
$\varphi_\gamma = \varphi_{\gamma_0} = \frac{360°}{Num.\ vertex}$ 10
READ FRAME 11
NO
Compute φ_i START CLOCK 12
13 $\varphi_i >= \varphi_\gamma$
NO
14 $\varphi_\gamma > 2\pi$
$\varphi_\gamma = \varphi_{\gamma_0}$ 15
YES
$\varphi_\gamma = \varphi_{\gamma_0} = \frac{360°}{Num.\ vertex}$ 17
Compute φ_i 18
19 $\varphi_i >= \varphi_\gamma$
NO
YES
STOP CLOCK $\Delta t = t_{start} - t_{stop}$ 20
$\varphi_\gamma = \varphi_\gamma + \varphi_{\gamma_0}$ 21
$\omega_i = \frac{\varphi_\gamma}{\Delta_t}$ 22
SAVE DATA 23
24 EXIT KEY
NO
YES
25 END

Figure 3.48: Designed tracking algorithm with vertex sampling

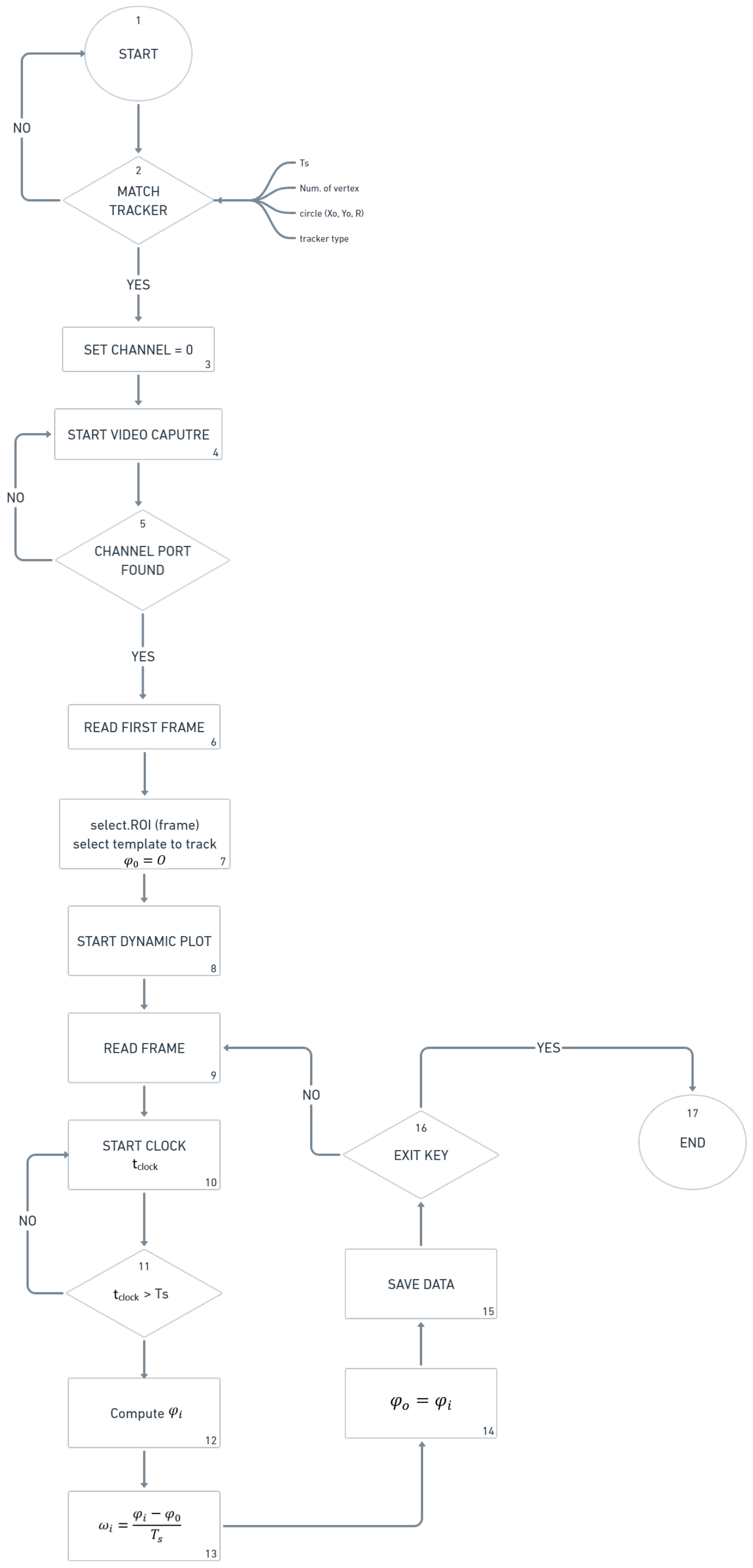


Figure 3.49: Designed tracking algorithm with time sampling

Flow chart 3.48 and 3.49 shows the process of the vertex sampling and time sampling respectively. The vertex version need 3 inputs in order to read the first image the camera receives (7). This first frame is used to select the template(8) by hand and the system will automatically read next frames after template selection. ϕ_γ is the space sampling. Numbre 13 in 3.48 is in charge of determine whereϕhas overcome the input threshold or not. if the current ϕ angle with respect of the original position of the template is lower than the first vertex and the disk did not utterly turned(14) then the system can go directly to read the next frame. Step(15) happens when the updated threshold is higher than the complete turn. Then the threshold is initialized by the default input. If condition (13) is fullfilled, then the system must wait until reach the next vertex(19) to compute the angular velocity(22). Whenever the next vertex is found, the threshold must be updated(21) in order to look for the next vertex in the future frames. Notice that the whole process computes ϕ as the angle with respect of the center of the template selected at the first frame the camera camptures. However, in figure 3.49, the angle ϕ is computed as the angle beetween 2 consecutive frames.So in general phy is smaller than in the vertex version. The diference now remains in the tiem sampling feature(11). Every time the system time exceeds the Ts, the algorithm computes the angular velocity. Notice that in reality the condition makes sense if Ts is in reality the the aggregated value of Ts. Then nTs where n are the samples taken would be the real time overtaking condition to compute ϕ

Algorithm 1: Time sampling tracking algorithm

```
initialization;
Tracker type
Ts
circle x0,y0,R
frame1 = Readfirstframe()
bbox = cv2.selectROI(frame1, False) select template
startclock()
d = dynamPlot.DynamicUpdate() start dynamic plot
while not exit key do
    bboxcenterpreviousFrame = edge2centerSquare(bbox);
    center2boxxvector = define2DVector(circlecenter, bboxcenter)
    center2referencevector = define2DVector(circlecenter, referencepoint)
    isclockwise = checkclockwisedirection()
    phiderivative = abs(phi - prevphi)
    if absphiderivative > 1 and absphiderivative < 359 then
        isStopped = False;
    end
    if isSameTurn then
        if time2sample > agregatedTime then
            sectionangularspeed = (phi - phioldsampled) / (sectiontime)
        else
    end
end
```

Figure 3.6. shows a pseudocode summing up the most important processes in the algorithm and showing the type of functions used. For instance, ϕ is computed as the angle between 2 vectors formed from teh center of the radius to the samples measured. Also clockwise or anticlockwise directions are taken into account. Other conditions as if the turn has been completed or if teh disk is stopped has to be taken into account in order to avoid lap finish problems. Polygon sampling version pseudocode has not been included since the idea is similar but more complex in terms of conditions. In order to obtain all the angular tracking algorithm writen in python, the reader can obtain it at appendix C

CHAPTER 4. FUTURE WORK

Although the improvement up to now have been proved to be useful, it exist some other improvement to consider after all this work done. C++ is a true candidate to embedde the code into a raspberry and control it remotelly. Indeed, since C++ allows to be programed with microcontrolers, the GUI could be controlled externally and the code processed by the hardware in the coils coded in C++. The new communication channel with the GUI and the C++ code coils controllers would need to be designed. Also, some more orbit features could be added into the GUI as automatic previsualization of the orbits in order to know the mission parameters rapidly while doing the tests. Furthermore, the angular tracking algorithm could be added in the raspberry as a new feature fo the system to control the angular velocities at real time with the GUI. On the other hand, angular tracking has many parameters to tune and optimize as has been shown in this work.

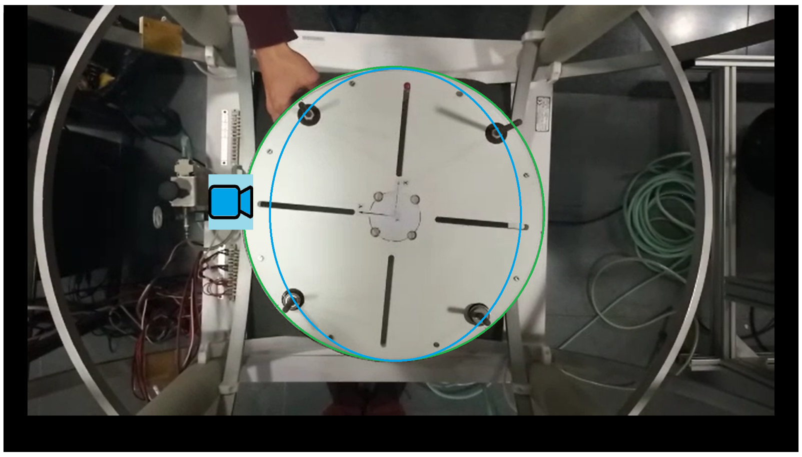


Figure 4.1: Camera not located at center

The system could be suited with 3 cammeras: 1 at the center and the other closer to the center. However, the displaced cameras in reality observes the disk as an ellipse as shown in figure 4.1. Then since the eliptical fit has been studied, it ccould be applied to those non centered cameras in order to compute the angular velocities. Finally the system could be syncronised and sample each camera at Ts/n in order to have n samples in comparison with the current 1 sample each Ts. n uis the numbre of cameras. This would allow the system to have more information in order to improve the measurements.

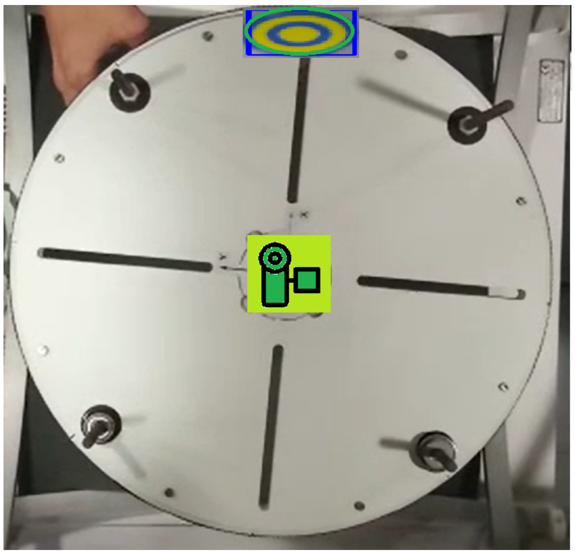


Figure 4.2: Camera located at the center

Another parameter that could be changed is the pattern, since the camera is located at the center, it observes in reality an elipse when the physical pattern is a circle. Therefore, in order to observe a real circle the pattern should be deformed elliptically in the other direction than the elipse observed in Figure 4.2 thus it will work as the inverse transfer fucntion. Finally other parameters as the colors of the patterns or the shapes analised with the FFTs could reveal new better candidates for the system. Also changing the programing language to C would imporve time efficiency of the algorithm which is critical for real time procedures.

CONCLUSIONS

Cubesat testing requirements need Helmholtz coil magnetic field generation in order to create the desired mission orbit conditions. It has been proved that with the aid of a power supply and a device able to switch the current, the system can be automated and correctly set up. The new GUI uses C++ allows the user to set up easily the Helmholtz coil desired environment and also, allows the user not to know how the code is structured so non expert users can use it without fearing on the background code. Furthermore, C++ coding language will allow future Helmholtz coils generation to keep growing due to its adaptative capabilities with embedded systems or SO. Specially with the change of new devices as the power supply or the arduino, with the serial communication set in the C++ code, those future communication will be better linked due to the serial robustness. It has been analized the coils principle and which is its operative range. Indeed, it has been observed how testing procedures generate magnetic fields close to 50 nT applying current magnitudes of the order of 1mA which generates magnetic fields of the order of magnitude of a real small satellite orbit. Also, those magnetic fields have been observed to be stable and constant along all the Cubesat dimensions, which is the first design caracterization of the Helmholtz coils.Orbit propagation and characterization algorithms as SGDP4 and IGRF with the knowledge of the reference frames allow the user to create a file containing the magnetic fields that can be set in Hemlholtz coil equipment in order to simulate the magnetic conditions of the orbit that can be used for testing campaign in order to evaluate the Cubesat sensors and actuators. On the other hand, some actuators such as magnetorquers are in charge of satellite control at the worst case scenarios. Detumbling is one of those scenarios that the testing procedure needs to know the angular velocity of the disk inside the platform as well as the gysocope calibration. It has been proved that the angular tracking algorithm created works for this purpose. Allowing the user to know in real time the angular velocity of any circular rotational movement. Some parameters have been analysed in order to understand the complex system: templates, patterns, Hough auto circle detector, trackers, circular and elliptical fits, frequency responses and so on. Although the algorithm has many parameters, it has been observed that MOSSE tracker at angular velocities close to 60º/s works as correctly and perfectly for detumbling tests that operate at those velocities. Furthermore, MEMS gyroscopes calibrations need to knows constant angular velocities that now can be settled by hand to the calibration procedure using the tracking algorithm. Two algorithm design: time sampling or angular sampling for the best practical purpose depending on the application. Furthermore, the statistical analysis shows that in the worst case scenario using CSRT tracker, in which the disk does not rotate at perfect constant angular velocities changing angular scenarios, the tracker works measuring the angular velocities up to 5º/s of error in average and 0.99 of correlation of the overall process. Indeed, real testing and calibrations do not program those tough changes seen in the experiment. The procedure needs lower angular velocities and constant angular velocities which can be easily tracked and measured by the algorithm.

BIBLIOGRAPHY

[1] Carrara, Valdemir; Barbosa Januzi, Rafael; Hideaki Makita, Daniel; Felipe de Paula Santos, Luis; Shibuya Sato, Lidia. The ITASAT CubeSat Development and Design. In: *Journal of Aerospace Technology and Management* [online]. São José dos Campos, April-June, 2017. DOI:10.5028/jatm.v9i2.614 . xi, 4, 5

[2] Watson, Jeff. *MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments* [online]. United States: Analog Devices. Available: https://www.analog.com/ru/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html . 5

[3] *ISIS Magnetorquer board* [online]. CubeSatShop. Available: https://www.cubesatshop.com/product/isis-magnetorquer-board/ . xi, 10

[4] R. J. Vaccaro and A. S. Zaki. Statistical Modeling of Rate Gyros. In: *IEEE Transactions on Instrumentation and Measurement* [online]. March 2012. DOI: 10.1109/TIM.2011.2171609 . 8

[5] Freescale Semiconductor, Inc. *Allan Variance: Noise Analysis for Gyroscopes* [online]. Telesens, 2015. Available: https://telesens.co/wp-content/uploads/2017/05/AllanVariance5087-1.pdf . 8

[6] *IEEE Standard Specification Format Guide and Test Procedure for Single-Axis Laser Gyros* [online]. IEEE, 2006. DOI: 10.1109/IEEESTD.2006.246241 . xi, 8

[7] Girod, Bernd. *Digital Image Processing* [online]. California: Standford University, 2013. Available: https://web.stanford.edu/class/ee368/Handouts/Lectures/2014_Spring/Combined_Slides/11-Edge-Detection-Combined.pdf . 53, 65

[8] Python. *opencv-python 4.6.0.66* [online]. Python Package Index, 2022. Available: https://pypi.org/project/opencv-python/ .

[9] *Canny Edge Detection* [online]. OpenCV, 2022. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html . 65

[10] Milos Toquica, Hans; Moreno Caderon, Jose Luis. Hough and Watershed Transforms Algorithms Brief Review. [online]. Colombia: Mechanics and mechatronics Department, Engineering Faculty, National University of Colombia, 2017. DOI:10.13140/RG.2.2.35718.98889 . 56

[11] Liu, Shuai; Liu, Dongye; Srivastava, Gautam; Polap, Dawid; Wozniak, Marcin. Overview and methods of correlation filter algorithms in object tracking. In: *Complex & Intelligent Systems* [online]. 2020. DOI: https://doi.org/10.1007/s40747-020-00161-4 . 68

[12] Sidhu, Rumpal. Tutorial on Minimum Output Sum of Squared Error Filter. [online]. 2016. URI: http://hdl.handle.net/10217/173486 . 69

[13] L. Szpak, Zygmunt. *Ellipse Fitting*. [online]. Zygmunt L.Szpark. Computer Vision & Machine Learning, 2015. Available: http://www.users.on.net/ zygmunt.szpak/ellipsefitting.html . 59

[14] *Direct linear least squares fitting of an ellipse*. [online]. Scipython, 2021. Available: https://scipython.com/blog/direct-linear-least-squares-fitting-of-an-ellipse/ . xii, 60

[15] Python. *circle-fit 0.1.3*. [online]. Python Package Index, 2019. Available: https://pypi.org/project/circle-fit/ .

[16] Al-sharadqah, Ali; Chernov, Nikolai. Error analysis for circle fitting algorithms. In: *Electronic Journal of Statistics*. [online]. Birmingham: Department of Mathematics, University of Alabama at Birmingham, 2009. DOI:10.1214/09-EJS419 . 59

[17] AI & Data Science Blog. *A Complete Review of the OpenCV Object Tracking Algorithms*. [online]. Brouton lab. Available: https://broutonlab.com/blog/opencv-object-tracking . 79

[18] Mendoza, Mariela. CubeSat Design Specification. California Polytechnic State University, 2014. . 3

[19] Lovera, Marco. Magnetic satellite detumbling: the b-dot algorithm revisited. In: *2015 American Control Conference (ACC)*. [online]. Chicago: 2015. DOI: 10.1109/ACC.2015.7171005 .

[20] *TRIAD*[online].AHRS: Attitude and Heading Reference Systems, 2019. Available: https://ahrs.readthedocs.io/en/latest/filters/triad.html . 5

[21] *QUEST*. [online]. AHRS: Attitude and Heading Reference Systems, 2019. Available: https://ahrs.readthedocs.io/en/latest/filters/quest.html . 5

[22] Chris Rossouw, Nico. A GPS-based On-board Orbit Propagator for Low Earth-Orbiting CubeSats. [online]. South Africa: Stellenbosch University, 2015. URI: http://hdl.handle.net/10019.1/97908 . 5

[23] *1300 mm Helmholtz Coils. Ferronato BH1300-3-A*. [online]. Serviciencia, 2014. Available: http://www.serviciencia.es/BH1300-i1.htm . xi, 14, 15, 18

[24]

Experiment 67. HALL PROBE MEASUREMENT OF MAGNETIC FIELDS. Department of Physics, University of Illinois at Urbana-Champaign, 2006. Available: https://courses.physics.illinois.edu/phys401/sp2018/Files/Hall%20Probe/E67_Spring09.pdf

[25] Mapping of the Magnetic Field from Helmholtz Coils xi, 18

[26] Magnetic field inside Helmholtz coil arrangementyo

[27] Jovanovic, N; Adiwiluhung Riwanto, Bagus; Niemelä, Patri; Rizwan Mughal, Muhammad; Praks, Jaan. Design of Magnetorquer-Based Attitude Control Subsystem for FORESAIL-1 Satellite. In: *IEEE Journal on Miniaturization for Air and Space Systems* [online]. 2021. DOI:10.1109/JMASS.2021.3093695 10

[28] Gasberg Thomsen, Brian; Nielsen, Jens. CubeSat Sliding Mode Attitude Control. [online]. Aalborg University, 2016. Available: https://projekter.aau.dk/projekter/files/239482913/report_final.pdf xi, 24, 25

[29] Compston, Drew. *International Geomagnetic Reference Field (IGRF) Model*. [online]. MATLAB Central File Exchange, 2022. Available: https://www.mathworks.com/matlabcentral/fileexchange/34388-international-geomagnetic-reference-field–igrf–model/

[30] *International Geomagnetic Reference Field (IGRF)*. [online]. British Geological Survey. Available: http://www.geomag.bgs.ac.uk/research/modelling/IGRF.html 28

[31] Alken, Patrick. *International Geomagnetic Reference Field*. [online]. The International Union of Geodesy and Geophysics, 2019. Available: https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html 5, 26

[32] Gupta, Harsh. Geomagnetic Field, IGRF. In: *Encyclopedia of Solid Earth Geophysics* [online]. Springer, Dordrecht, 2011. DOI: https://doi.org/10.1007/978-90-481-8702-7_111 xi, 26

[33] Análisis de rendimiento del algoritmo SGP4/SDP4 para predicción de posición orbital de satélites artificiales utilizando contadores de hardware Federico J. Díaz1 , Fernando G. Tinetti2,3 , Nicanor B. Casas1 Graciela E. De Luca1 , Sergio M. Martin1 , Daniel A. Giulianelli1 23

[34] Kerrisk, Michael. *termios(3) — Linux manual page*. [online]. 2021.Available: https://www.man7.org/linux/man-pages/man3/termios.3.html 33

[35] Navipedia. *Transformations between ECEF and ENU coordinates*. [online]. European Space Agency, 2022. Available: https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU _coordinates 39

[36] *Erco's FLTK Cheat Page*. [online]. Seriss Corporation. Available: http://seriss.com/people/erco/fltk/ 36, 43

[37] Spitzak, Bill; Sweet, Michael; Earls, Craig. P; Melcher, Matthias; Kaiser, Nicolas; Stott, Ben. *Example 1: Basic FLTK Program*. [online]. FLTK 2.0 Documentation, 2011. Available: https://www.fltk.org/doc-2.0/html/example1.html 36, 42

[38] Space Strategy, First Edition. Jean-Luc Lefebvre. Available: https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html 5

APPENDICES

APPENDIX A. C++ HELMHOLTZ COILS CODE

```
#include "Interface.h"
#include <iostream>
#include <fstream>
#include "Serial.h"
#include "File.h"
#include "start.h"



#include <vector>
#include <cstring>
#include <string.h>
#include <stdio.h>
#include <vector>



#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_JPEG_Image.H>
#include <FL/Enumerations.H>
#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Window.H>
#include <FL/fl_draw.H>
#include <FL/Fl_File_Chooser.H>

#include <cstdio>
#include <functional>

#define WIDTH 650
#define BARWIDTH 200
#define HEIGHT 600
#define BLUE fl_rgb_color(0x60, 0xAE, 0xFF)
#define BLUE_DARK fl_rgb_color(0x09, 0x6D, 0xF9)
#define SEL_BLUE fl_rgb_color(0x2, 0x60, 0xF3)

#define GRAY fl_rgb_color(0x75, 0x75, 0x75)
#define LIGHT_GRAY fl_rgb_color(211, 211, 211)

using cb_t = std::function<void(Fl_Widget *)>;

class MyButton : public Fl_Button {
 private:
    char hint[160];
    Fl_Widget* statusbar;
 public:
     MyButton(int x, int y, int w, int h, const char* l): Fl_Button(x,
 y, w, h, l) {

          strcpy(hint," ");
     }
     void set_hint(Fl_Widget* sb, const char *h) {
         statusbar = sb;
         strcpy(hint,h);
     }
     int handle(int event) {
                 int ret = Fl_Button::handle(event);
         switch (event) {
             case FL_ENTER:
                 color(BLUE_DARK);
```

```
                redraw();
                break;
            case FL_LEAVE:
                color(BLUE);
                redraw();
                break;
        }
        return (ret);
    }
};







class MyBox : public Fl_Box {
  public:
    MyBox(int x, int y, int w, int h, const char *title = NULL): Fl_Box(x, y, w, h, title) {}
    virtual void draw() override {
        fl_rectf(0, h(), w(), 3, LIGHT_GRAY);
        Fl_Box::draw();
    }
};

template <typename T>
class WidgetWrapper : public T {
    cb_t func_ = NULL;

  public:
    WidgetWrapper(int x, int y, int w, int h, const char *title = NULL): T(x, y, w, h, title) {}
    void set_cb(cb_t &&func) {
        func_ = func;
        auto cb = [](Fl_Widget *w, void *data) {
            auto fn = *static_cast<cb_t *>(data);
            fn(w);
        };
        this->callback(cb, &func_);
    }
};





struct inputs{
Fl_Input *new_input1;
Fl_Input *new_input2;
Fl_Input *new_input3;


Fl_Input *new_magnetorquer_input1;
Fl_Input *new_magnetorquer_input2;
Fl_Input *new_magnetorquer_input3;



double Bxx_magnetorquers;
double Byy_magnetorquers;
double Bzz_magnetorquers;

Fl_Button *load_btn;
Fl_Button *load_btn_step1;
Fl_Button *load_btn_step2;
Fl_Button *load_btn_step3;
Fl_Button *load_btn_step4;
Fl_Button *run_dynamic_btn;
Fl_Button *load_btn_dynamic_step1;
Fl_Button *load_btn_dynamic_step2;



};







using namespace std;
void but1_cb( Fl_Widget* o, struct inputs *v1  );
void but2_cb(Fl_Widget* o, struct inputs *v1   );
void but3_cb(Fl_Widget* o, struct inputs *v1 );
void butStep1_cb( Fl_Widget* o, struct inputs *v1 );
void butStep2_cb( Fl_Widget* o, struct inputs *v1  );
void butStep3_cb( Fl_Widget* o, struct inputs *v1  );
void butStep4_cb( Fl_Widget* o, struct inputs *v1  );
void butStep1_dynamic_cb( Fl_Widget* o, struct inputs *v1  );
void butStep2_dynamic_cb( Fl_Widget* o, struct inputs *v1  );



void but4_cb( Fl_Widget* o, void *  );
void but5_cb( Fl_Widget* o, struct inputs *v1  );
void but6_cb( Fl_Widget* o, struct inputs *v1  );
void but7_cb( Fl_Widget* o, struct inputs *v1  );
void Browse_CB( Fl_Widget* o, struct inputs *v1 );
void init_interface();
void init_template();

void get_Bxx_Byy_Bzz_fromStrings(string sBxx,string sByy, string sBzz, double &Bxx, double &Byy,
double &Bzz);
string convertToString(char* a, int size);
int mType;


int main (int argc, char *argv[]) {




//////////////////////////////////////////////////////////////////////////////////////////
////////
        ////////////////////////////////////////////Starting Protocol: mainType=1-----> Static
Orbit Symulator///
        ////////////////////////////////////////////Starting Protocol: mainType=2-----> Dynamic
Orbit Symulator//

//////////////////////////////////////////////////////////////////////////////////////////
////////

                cout<<"intefaceeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"<<endl;
        int mainType;
        init_interface();







//////////////////////////////////////////////////////////////////////////////////////////
///////////
        //////////////////////////////////////////////////////////////7///END OF main
Static_Orbit_Sim/////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
///////////



        return 0;
}




void init_template(){
  auto win = new Fl_Window(WIDTH, HEIGHT, "Flutter-like");
    auto bar = new WidgetWrapper<MyBox>(0, 0, WIDTH, 120, "");
    Fl_Box *box = new Fl_Box(BARWIDTH/2,400,300,300,"HELMHOLTZ");
    //bar->align(FL_ALIGN_LEFT | FL_ALIGN_INSIDE);
    auto txt = new WidgetWrapper<Fl_Box>(
        250, 180, 100, 40, "You have pushed the button this many times:");
    auto count = new WidgetWrapper<Fl_Box>(txt->x(), txt->y() + 40, 100, 40, "0");
    auto but1 = new WidgetWrapper<Fl_Button>(WIDTH - 100, HEIGHT - 100, 60, 60, "@+6plus");
       auto but2 = new WidgetWrapper<Fl_Button>(WIDTH - 200, HEIGHT - 200, 60, 60, "@+6plus");
    win->end();
    win->show();

    // theming
    Fl::background(255, 255, 255);
    Fl::visible_focus(false);

    box->box(FL_UP_BOX);
    box->color(BLUE);
    box->show();

    bar->box(FL_FLAT_BOX);
    bar->labelsize(22);
    bar->labelcolor(FL_WHITE);
    bar->color(BLUE);

    txt->labelsize(18);
    txt->labelfont(FL_TIMES);

    count->labelsize(36);
    count->labelcolor(GRAY);

    but1->color(BLUE);
    but1->selection_color(SEL_BLUE);
    but1->labelcolor(FL_WHITE);
    but1->box(FL_OFLAT_BOX);
    // end theming

    but1->set_cb([=](Fl_Widget *b) {
        auto label = atoi(count->label());
        label += 1;
        char buf[12];
        (void)snprintf(buf, 12, "%d", label);
        count->copy_label(buf);
    });


    but2->color(BLUE);
    but2->selection_color(SEL_BLUE);
    but2->labelcolor(FL_WHITE);
    but2->box(FL_OFLAT_BOX);
    // end theming

    but2->set_cb([=](Fl_Widget *b) {
        auto label = atoi(count->label());
        label += 1;
        char buf[12];
        (void)snprintf(buf, 12, "%d", label);
        count->copy_label(buf);
    });

    Fl::run();

};


void init_interface()
{


        //init_template();
        Fl_Window win( WIDTH, HEIGHT,"Testing" );
        Fl::background(255, 255, 255);
        Fl::visible_focus(false);


        auto barColumn = new WidgetWrapper<MyBox>(0, 0, BARWIDTH, HEIGHT, "Helmholtz");

        barColumn->align(FL_ALIGN_TOP_LEFT| FL_ALIGN_INSIDE);
        //bar->align(FL_ALIGN_LEFT | FL_ALIGN_INSIDE);
        /*
        Fl_Button but1(10, 150, 190, 40, "Static Simulation" );
        Fl_Button but2( 200, 150, 190, 40, "dynamic Simulation" );
        Fl_Button but3( 200, 200, 190, 40, "Load file Simulation" );
        */
        auto btnStatic = new MyButton(0, 200, BARWIDTH, 50, "Static Simulation");
        auto btnDynamic = new MyButton(0, 300, BARWIDTH, 50, "Dynamic Simulation");
        auto btnPropagator = new MyButton(0, 400, BARWIDTH, 50, "Orbit Porpagator");
        auto btnClose = new MyButton(0, 500, BARWIDTH, 50, "Close");
        auto runbtn = new MyButton(BARWIDTH+100, 150, BARWIDTH, 50, "run simulation");
      auto runbtn_step1 = new MyButton(BARWIDTH+100, 25, BARWIDTH, 25, "run simulation step1");
      auto runbtn_step2 = new MyButton(BARWIDTH+100, 75, BARWIDTH, 25, "run simulation step2");
      auto runbtn_step3 = new MyButton(BARWIDTH+100, 125, BARWIDTH, 25, "run simulation step3");
      auto runbtn_step4 = new MyButton(BARWIDTH+100, 175, BARWIDTH, 25, "run simulation step4");
        auto runbtnDynamic = new MyButton(BARWIDTH+100, 250, BARWIDTH, 50, "run simulation");
      auto runbtn_dynamic_step1 = new MyButton(BARWIDTH+100, 25, BARWIDTH, 25, "run dynamic
step1");
      auto runbtn_dynamic_step2 = new MyButton(BARWIDTH+100, 50, BARWIDTH, 25, "run dynamic
step2");
        char buffer[20] = "inoutbuffer111";
        //Fl_Input* input1 = new  Fl_Input(BARWIDTH, 300, 250, 25,"Input:");
        //Fl_Input* input2 = new  Fl_Input(BARWIDTH, 325, 250, 25,"Input:");
        //Fl_Input* input3 = new  Fl_Input(BARWIDTH, 350, 250, 25,"Input:");
        //Fl_Output output(BARWIDTH+100, 375, 250, 25,"Output:");
        //Fl_Widget inputs[3] = {input1,input2, input3};

        struct inputs userInputs;

        userInputs.load_btn = runbtn;
      userInputs.load_btn_step1 = runbtn_step1;
      userInputs.load_btn_step2 = runbtn_step2;
      userInputs.load_btn_step3 = runbtn_step3;
      userInputs.load_btn_step4 = runbtn_step4;


        userInputs.run_dynamic_btn = runbtnDynamic;
      userInputs.load_btn_dynamic_step1 = runbtn_dynamic_step1;
      userInputs.load_btn_dynamic_step2 = runbtn_dynamic_step2;


        userInputs.new_input1 = new Fl_Input(BARWIDTH+100, 300, 250, 25,"Input Bx:");
        userInputs.new_input2 = new Fl_Input(BARWIDTH+100, 325, 250, 25,"Input By:");
        userInputs.new_input3 = new Fl_Input(BARWIDTH+100, 350, 250, 25,"Input Bz:");



        userInputs.new_magnetorquer_input1 = new Fl_Input(BARWIDTH+100, 400, 250, 25,"Voltage
Bx:");
```

```
        userInputs.new_magnetorquer_input2 = new Fl_Input(BARWIDTH+100, 425, 250, 25,"Voltage
By:");
        userInputs.new_magnetorquer_input3 = new Fl_Input(BARWIDTH+100, 450, 250, 25,"Voltage
Bz:");

        char exampleBxx[20] = "50000.0000";
        char exampleByy[20] = "50000.0000";
        char exampleBzz[20] = "-50000.11111";

      char example_magBxx[20] = "10.0000";
        char example_magByy[20] = "10.0000";
        char example_magBzz[20] = "-10.11111";
        //output.hide();
        userInputs.new_input1->hide();
        userInputs.new_input2->hide();
        userInputs.new_input3->hide();
        userInputs.new_input1->value(exampleBxx);
        userInputs.new_input2->value(exampleByy);
        userInputs.new_input3->value(exampleBzz);
        userInputs.new_magnetorquer_input1->hide();
        userInputs.new_magnetorquer_input2->hide();
        userInputs.new_magnetorquer_input3->hide();
        userInputs.new_magnetorquer_input1->value(example_magBxx);
        userInputs.new_magnetorquer_input2->value(example_magByy);
        userInputs.new_magnetorquer_input3->value(example_magBzz);
        userInputs.load_btn->hide();

        userInputs.run_dynamic_btn->hide();
      userInputs.load_btn_dynamic_step1->hide();
      userInputs.load_btn_dynamic_step2->hide();


        //cout << output <<"tyhis is the output1111111"<<endl;
        printf("My Name is %s", buffer);
        cout << "buffer is:"<< buffer[0]<<buffer[1]<< endl;

        /*
        Fl_PNG_Image* startime = new Fl_PNG_Image("helm.png");
         Fl_Box *box = new Fl_Box(100, 100, 100, 100);
        char imagName[20] = "helm.png";
        */






        //box->image(startime);
        //box->show();
        /*
        bar->box(FL_FLAT_BOX);
        bar->labelsize(22);
        bar->labelcolor(FL_WHITE);
        bar->color(BLUE);
        */
        barColumn->box(FL_FLAT_BOX);
        barColumn->labelsize(22);
        barColumn->labelcolor(FL_WHITE);
        barColumn->color(BLUE);

        //btn->color(BLUE);
        //btn->callback((Fl_Callback*)but1_cb,&userInputs);
        /*
        but1.color(BLUE);
        but1.selection_color(SEL_BLUE);
        but1.labelcolor(FL_WHITE);
        but1.box(FL_RFLAT_BOX);
        but1.callback( (Fl_Callback*)but1_cb,&userInputs);

        but2.color(BLUE);
        but2.selection_color(SEL_BLUE);
        but2.labelcolor(FL_WHITE);
        but2.box(FL_RFLAT_BOX);
        but2.callback( but2_cb,  (void* )54321 );

        but3.color(BLUE);
        but3.selection_color(SEL_BLUE);
        but3.labelcolor(FL_WHITE);
        but3.box(FL_RFLAT_BOX);
        char fileName[20] = "";
        but3.callback( Browse_CB,  (void* )54321  );
        */
        btnStatic->color(BLUE);

        //box->image(startime);
        //box->show();
        /*
        bar->box(FL_FLAT_BOX);
        bar->labelsize(22);
        bar->labelcolor(FL_WHITE);
        bar->color(BLUE);
        */
        barColumn->box(FL_FLAT_BOX);
        barColumn->labelsize(22);
        barColumn->labelcolor(FL_WHITE);
        barColumn->color(BLUE);

        //btn->color(BLUE);
        //btn->callback((Fl_Callback*)but1_cb,&userInputs);
        /*
                but1.color(BLUE);
                but1.selection_color(SEL_BLUE);
                but1.labelcolor(FL_WHITE);
                but1.box(FL_RFLAT_BOX);
                but1.callback( (Fl_Callback*)but1_cb,&userInputs);

        but2.color(BLUE);
                but2.selection_color(SEL_BLUE);
                but2.labelcolor(FL_WHITE);
                but2.box(FL_RFLAT_BOX);
        but2.callback( but2_cb,  (void* )54321 );

        but3.color(BLUE);
        but3.selection_color(SEL_BLUE);
        but3.labelcolor(FL_WHITE);
        but3.box(FL_RFLAT_BOX);
        char fileName[20] = "";
        but3.callback( Browse_CB,  (void* )54321

      */
        btnStatic->labelfont(FL_TIMES);
        btnStatic->selection_color(SEL_BLUE);
        btnStatic->labelcolor(FL_WHITE);
        btnStatic->box(FL_RFLAT_BOX);
        btnStatic->callback( (Fl_Callback*)but1_cb,&userInputs);

        btnDynamic->color(BLUE);
        btnDynamic->labelfont(FL_TIMES);
        btnDynamic->selection_color(SEL_BLUE);
        btnDynamic->labelcolor(FL_WHITE);
        btnDynamic->box(FL_RFLAT_BOX);
        btnDynamic->callback( (Fl_Callback*)but2_cb,&userInputs );

        btnPropagator->color(BLUE);
        btnPropagator->selection_color(SEL_BLUE);
        btnPropagator->labelcolor(FL_WHITE);
        btnPropagator->box(FL_RFLAT_BOX);
        btnPropagator->callback( (Fl_Callback*)but2_cb,&userInputs );
        runbtn->hide();
        runbtn->label("@-> Console Static");
        runbtn->color(BLUE);
        runbtn->labelfont(FL_TIMES);
        runbtn->selection_color(SEL_BLUE);
        runbtn->labelcolor(FL_WHITE);
        runbtn->box(FL_RFLAT_BOX);
        runbtn->callback((Fl_Callback*)but3_cb,&userInputs  );

    runbtn_step1->hide();
        runbtn_step1->label("@-> step1");
        runbtn_step1->color(BLUE);
        runbtn_step1->labelfont(FL_TIMES);
        runbtn_step1->selection_color(SEL_BLUE);
        runbtn_step1->labelcolor(FL_WHITE);
        runbtn_step1->box(FL_RFLAT_BOX);
        runbtn_step1->callback((Fl_Callback*)butStep1_cb,&userInputs  );

    runbtn_step2->hide();
    runbtn_step2->label("@->step2");
    runbtn_step2->color(BLUE);
    runbtn_step2->labelfont(FL_TIMES);
    runbtn_step2->selection_color(SEL_BLUE);
    runbtn_step2->labelcolor(FL_WHITE);
    runbtn_step2->box(FL_RFLAT_BOX);
    runbtn_step2->callback((Fl_Callback*)butStep2_cb,&userInputs  );

    runbtn_step3->hide();
    runbtn_step3->label("@->step3");
    runbtn_step3->color(BLUE);
    runbtn_step3->labelfont(FL_TIMES);
    runbtn_step3->selection_color(SEL_BLUE);
    runbtn_step3->labelcolor(FL_WHITE);
    runbtn_step3->box(FL_RFLAT_BOX);
    runbtn_step3->callback((Fl_Callback*)butStep3_cb,&userInputs  );

    runbtn_step4->hide();
    runbtn_step4->label("@->step4");
    runbtn_step4->color(BLUE);
    runbtn_step4->labelfont(FL_TIMES);
    runbtn_step4->selection_color(SEL_BLUE);
    runbtn_step4->labelcolor(FL_WHITE);
    runbtn_step4->box(FL_RFLAT_BOX);
    runbtn_step4->callback((Fl_Callback*)butStep4_cb,&userInputs  );








        runbtnDynamic->hide();
        runbtnDynamic->label("@-> Console Simulation");
        runbtnDynamic->color(BLUE);
        runbtnDynamic->labelfont(FL_TIMES);
        runbtnDynamic->selection_color(SEL_BLUE);
        runbtnDynamic->labelcolor(FL_WHITE);
        runbtnDynamic->box(FL_RFLAT_BOX);
        runbtnDynamic->callback((Fl_Callback*)but5_cb,&userInputs  );




     runbtn_dynamic_step1 -> hide();
        runbtn_dynamic_step1->label("@-> Step1");
        runbtn_dynamic_step1->color(BLUE);
        runbtn_dynamic_step1->labelfont(FL_TIMES);
        runbtn_dynamic_step1->selection_color(SEL_BLUE);
        runbtn_dynamic_step1->labelcolor(FL_WHITE);
        runbtn_dynamic_step1->box(FL_RFLAT_BOX);
        runbtn_dynamic_step1->callback((Fl_Callback*)but6_cb,&userInputs  );



        runbtn_dynamic_step2->hide();
        runbtn_dynamic_step2->label("@->Step2");
        runbtn_dynamic_step2->color(BLUE);
        runbtn_dynamic_step2->labelfont(FL_TIMES);
        runbtn_dynamic_step2->selection_color(SEL_BLUE);
        runbtn_dynamic_step2->labelcolor(FL_WHITE);
        runbtn_dynamic_step2->box(FL_RFLAT_BOX);
        runbtn_dynamic_step2->callback((Fl_Callback*)but7_cb,&userInputs  );

        btnClose->color(BLUE);
        btnClose->labelfont(FL_TIMES);
        btnClose->selection_color(SEL_BLUE);
        btnClose->labelcolor(FL_WHITE);
        btnClose->box(FL_RFLAT_BOX);
        btnClose->callback( but4_cb);
        //Interface::setMainType();
        cout <<"maintype is::::"<<mType<<endl;
        //but.callback( but_set_mainType );
        win.show();
        /*
        char inp1[20] = "";
        char inp2[20] = "";
        char inp3[20] = "";
        strcat(inp1, input1->value());
        strcat(inp2, input2->value());
        strcat(inp3, input3->value());
        cout << inp1 <<"tyhis is the output1111111"<<endl;
        */Fl::run();
};


void but1_cb( Fl_Widget* o, struct inputs *v1 ) {
        string ow;
        v1->new_input1->show();
        v1->new_input2->show();
        v1->new_input3->show();
        v1->new_magnetorquer_input1->show();
        v1->new_magnetorquer_input2->show();
        v1->new_magnetorquer_input3->show();
  v1->load_btn->show();
  v1->load_btn_step1->show();
        char i1[20] = "";
        char i2[20] = "";
        char i3[20] = "";
        Fl_Input* iw1 = v1->new_input1;
        Fl_Input* iw2 = v1->new_input2;
        Fl_Input* iw3 = v1->new_input3;
        //Fl_Input* iw3 = (Fl_Input*)v1;
        //string ow = iw->value();
        strcat( i1,iw1->value());
        strcat( i2,iw2->value());
        strcat( i3,iw3->value());
        //strcat( i3,iw3->value());
        cout << i1 << "outpu1"<<endl;
        cout << i2 << "outpu2"<<endl;
        cout << i3 << "outpu3"<<endl;
        size_t sizexx = std::strlen( i1 );
        size_t sizeyy = std::strlen( i2 );
        size_t sizezz = std::strlen( i3 );
        string stringBxx = convertToString(i1 , sizexx);
        string stringByy = convertToString(i2, sizeyy);
        string stringBzz = convertToString(i3, sizezz);
        cout <<stringBxx<<"string bxx"<<endl;
        //start main2call;
        //main2call.start_static_orbit_simulator();


        //cout << i3 << "outpu1234567"<<endl;

        /*
        Fl_Button* b=(Fl_Button*)o;
        b->label("Good job"); //redraw not necessary
        mType = 1;
        b->resize(BARWIDTH, 200, BARWIDTH, 30); //redraw needed
        b->redraw();




        b->color(BLUE);
        b->selection    v1->new_magnetorquer_input3->show();_color(SEL_BLUE);
        b->labelcolor(FL_WHITE);
        b->box(FL_RFLAT_BOX);
        b->callback( (Fl_Callback*)Browse_CB,v1  );
        */
        Fl_Button* loadbtn=(Fl_Button*)o;
        loadbtn->label("load file"); //redraw not necessary
        mType = 1;
        loadbtn->resize(BARWIDTH+100, 200, BARWIDTH, 30); //redraw needed
        loadbtn->redraw();
  loadbtn->hide();


        loadbtn->color(BLUE);
        loadbtn->selection_color(SEL_BLUE);
        loadbtn->labelcolor(FL_WHITE);
        loadbtn->box(FL_RFLAT_BOX);
        loadbtn->callback( (Fl_Callback*)Browse_CB,v1  );


};


void but3_cb( Fl_Widget* o, struct inputs *v1  ){

        char Bxx_chr[20] = "";
        char Byy_chr[20] = "";
        char Bzz_chr[20] = "";
        char Bxx_mag_chr[20] = "";
        char Byy_mag_chr[20] = "";
        char Bzz_mag_chr[20] = "";
        strcat( Bxx_chr,v1->new_input1->value());
        strcat( Byy_chr,v1->new_input2->value());
        strcat( Bzz_chr,v1->new_input3->value());
        strcat( Bxx_mag_chr,v1->new_magnetorquer_input1->value());
        strcat( Byy_mag_chr,v1->new_magnetorquer_input2->value());
        strcat( Bzz_mag_chr,v1->new_magnetorquer_input3->value());
        double Bxx = atof(Bxx_chr);
        double Byy = atof(Byy_chr);
        double Bzz = atof(Bzz_chr);
        double Bxx_mag = atof(Bxx_mag_chr);
        double Byy_mag = atof(Byy_mag_chr);
        double Bzz_mag = atof(Bzz_mag_chr);


        start main2call;
        main2call.start_static_orbit_simulator( Bxx, Byy, Bzz, Bxx_mag, Byy_mag, Bzz_mag);
```

```
};


void butStep1_cb( Fl_Widget* o, struct inputs *v1 ){

        char Bxx_chr[20] = "";
        char Byy_chr[20] = "";
        char Bzz_chr[20] = "";
        char Bxx_mag_chr[20] = "";
        char Byy_mag_chr[20] = "";
        char Bzz_mag_chr[20] = "";
        strcat( Bxx_chr,v1->new_input1->value());
        strcat( Byy_chr,v1->new_input2->value());
        strcat( Bzz_chr,v1->new_input3->value());
        strcat( Bxx_mag_chr,v1->new_magnetorquer_input1->value());
        strcat( Byy_mag_chr,v1->new_magnetorquer_input2->value());
        strcat( Bzz_mag_chr,v1->new_magnetorquer_input3->value());
        double Bxx = atof(Bxx_chr);
        double Byy = atof(Byy_chr);
        double Bzz = atof(Bzz_chr);
        double Bxx_mag = atof(Bxx_mag_chr);
        double Byy_mag = atof(Byy_mag_chr);
        double Bzz_mag = atof(Bzz_mag_chr);


        start main2call;
        main2call.start_static_orbit_simulator_step1( Bxx, Byy, Bzz, Bxx_mag, Byy_mag, Bzz_mag);
  v1->load_btn_step1->hide();
  v1->load_btn_step2->show();
};


void butStep2_cb( Fl_Widget* o, struct inputs *v1   ){
        start main2call;
        main2call.start_static_orbit_simulator_step2();
  v1->load_btn_step2->hide();
  v1->load_btn_step3->show();
};


void butStep3_cb( Fl_Widget* o, struct inputs *v1 ){
        start main2call;
        main2call.start_static_orbit_simulator_step3();
  v1->load_btn_step3->hide();
  v1->load_btn_step4->show();
};

void butStep4_cb( Fl_Widget* o, struct inputs *v1  ){
        start main2call;
        main2call.start_static_orbit_simulator_step2();
};




void but5_cb( Fl_Widget* o, struct inputs *v1  ){

        char Bxx_chr[20] = "";
        char Byy_chr[20] = "";
        char Bzz_chr[20] = "";
        char Bxx_mag_chr[20] = "";
        char Byy_mag_chr[20] = "";
        char Bzz_mag_chr[20] = "";
        strcat( Bxx_chr,v1->new_input1->value());
        strcat( Byy_chr,v1->new_input2->value());
        strcat( Bzz_chr,v1->new_input3->value());
        strcat( Bxx_mag_chr,v1->new_magnetorquer_input1->value());
        strcat( Byy_mag_chr,v1->new_magnetorquer_input2->value());
        strcat( Bzz_mag_chr,v1->new_magnetorquer_input3->value());
        double Bxx = atof(Bxx_chr);
        double Byy = atof(Byy_chr);
        double Bzz = atof(Bzz_chr);
        double Bxx_mag = atof(Bxx_mag_chr);
        double Byy_mag = atof(Byy_mag_chr);
        double Bzz_mag = atof(Bzz_mag_chr);

        start main2call;
        main2call.start_dynamic_orbit_simulator();


};

void but6_cb( Fl_Widget* o, struct inputs *v1  ){
  start main2call;
  main2call.start_dynamic_orbit_simulator_step1();
  v1->load_btn_dynamic_step1->hide();
  v1->load_btn_dynamic_step2->show();

}

void but7_cb( Fl_Widget* o, struct inputs *v1  ){
  start main2call;
        main2call.start_dynamic_orbit_simulator_step2();
}





void but2_cb(Fl_Widget* o, struct inputs *v1   ) {




        Fl_Button* loadbtn=(Fl_Button*)o;
        loadbtn->label("Load File"); //redraw not necessary
        mType = 1;
        loadbtn->resize(BARWIDTH+100, 200, BARWIDTH, 30); //redraw needed
        loadbtn->redraw();
        loadbtn->color(BLUE);
        loadbtn->selection_color(SEL_BLUE);
        loadbtn->labelcolor(FL_WHITE);
        loadbtn->box(FL_RFLAT_BOX);
        loadbtn->callback( (Fl_Callback*)Browse_CB,v1  );
        v1->run_dynamic_btn->show();
  v1->load_btn_dynamic_step1->show();











        //start main2call;
        //main2call.start_dynamic_orbit_simulator();
};



void but4_cb( Fl_Widget* o, void *  ){

exit(0);

};
```

```
void Browse_CB(Fl_Widget*o,  struct inputs *v1 ) {
        // Create file chooser (if not already)
        static Fl_File_Chooser *cho = 0;
        if ( ! cho ) {
        cho = new Fl_File_Chooser(".", "*", Fl_File_Chooser::MULTI, "Pick Something");
        }
        // Show chooser, wait for user to hit 'OK'
        cho->show();
        while ( cho->visible() ) {
        Fl::wait();
        }
        // Print the items user selected
        for ( int i=0; i< cho->count(); i++ ) {
        printf("FILENAME[%d]: %s\n", i, cho->value(i));

        }
        string fileName =cho->value(0);


    int n = fileName.length();

    // declaring character array
    char char_array[n + 1];

    // copying the contents of the
    // string to char array
    strcpy(char_array, fileName.c_str());

    for (int i = 0; i < n; i++)
        cout << char_array[i];




        //strcat(fileName,cho->value());

        //cout <<"this is fileName"<< char_array<<endl;
        File file_magnetorquers;
        file_magnetorquers.readFile(char_array);
        cout<<"fin"<<endl;

        double BxxmagFile = file_magnetorquers.Bxx[0];
        double ByymagFile = file_magnetorquers.Byy[0];
        double BzzmagFile = file_magnetorquers.Bzz[0];

        v1->Bxx_magnetorquers = BxxmagFile;
        v1->Byy_magnetorquers = ByymagFile;
        v1->Bzz_magnetorquers = BzzmagFile;

        char BxxmagFile_char[16];
        char ByymagFile_char[16];
        char BzzmagFile_char[16];

        sprintf(BxxmagFile_char,"%f",BxxmagFile);
        sprintf(ByymagFile_char,"%f",ByymagFile);
        sprintf(BzzmagFile_char,"%f",BzzmagFile);

        v1->new_magnetorquer_input1->value(BxxmagFile_char);
        v1->new_magnetorquer_input2->value(ByymagFile_char);
        v1->new_magnetorquer_input3->value(BzzmagFile_char);

cout << "BXXBYYBZZMAGNETORQUERS"<<BxxmagFile<<"and"<<ByymagFile<<"and"<<BzzmagFile<<endl;


};
```

```
void get_Bxx_Byy_Bzz_fromStrings(string sBxx,string sByy, string sBzz, double &Bxx, double &Byy,
double &Bzz){

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        //////this function splits the line Bxx;Byy;Bzz and set the values to the
variables/////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

        double B[3];

        double word_doublexx = std::stof(sBxx);
        double word_doubleyy = std::stof(sByy);
        double word_doublezz = std::stof(sBzz);
        B[0] = (double)word_doublexx;
        B[1] = (double)word_doubleyy;
        B[2] = (double)word_doublezz;
        Bxx = B[0];
        Byy = B[1];
        Bzz = B[2];
        //cout << Bxx <<"  "<< Byy<<"  " << Bzz <<"  this is BxxByyBzz inside\n";
}

string convertToString(char* a, int size)
{
    int i;
    string s = "";
    for (i = 0; i < size; i++) {
        s = s + a[i];
    }
    return s;
}
```

```
#include "Serial.h"

HANDLE my;
HANDLE myArduino;
HANDLE myPowerSupply;

//Call this for defaults
void InitSerialPort(void)
{
        // VERY IMPORTANT: Edit this line of code to designate which COM port the ADCS board is
using!!
        int BaudRate = 115200;
#ifdef __WIN32__
        char *port = "\\\\.\\COM12";
#endif
#if defined __linux__ || __APPLE__
        char *port = "/dev/ttyUSB0";
#endif
        my=SerialInit(port,BaudRate);
}


//Call this for higher level control
HANDLE SerialInit(char *ComPortName, int BaudRate)
{
        HANDLE hComm;

#ifdef RPI
        if(wiringPiSetup() == -1) {
                fprintf(stdout, "Unable to start wiringPi: %s\n", strerror (errno));
                return 1;
        }
        hComm = serialOpen(ComSerialRespondPortName,BaudRate);
        if (hComm < 0) {
                fprintf(stderr, "Unable to open serial device: %s\n", strerror (errno));
                return 1;
        }
        return hComm;
#endif

        //Setup for Windows
#ifdef __WIN32__
        hComm = CreateFile(ComPortName,
                                            GENERIC_READ | GENERIC_WRITE,
                                            0, // exclusive access
                                            NULL, // no security
                                            OPEN_EXISTING,
                                            0, // no overlapped I/O
                                            NULL); // null template
        //printf("x0 hComm=0x%08x GetLastError=%d\r\n",hComm,GetLastError());
        bPortReady = SetupComm(hComm, 128, 128); // set buffer sizes
        //printf("x1 PortReady=%d GetLastError=%d\r\n",bPortReady,GetLastError());
        bPortReady = GetCommState(hComm, &dcb);
        // printf("x2 PortReady=%d\r\n",bPortReady);
        dcb.BaudRate = BaudRate;
        dcb.ByteSize = 8;
        dcb.Parity = NOPARITY;
        //dcb.Parity = EVENPARITY;
        dcb.StopBits = ONESTOPBIT;
        dcb.fAbortOnError = TRUE;

        // set XON/XOFF
        dcb.fOutX = FALSE;                    // XON/XOFF off for transmit
        dcb.fInX    = FALSE;                    // XON/XOFF off for receive
        // set RTSCTS
        dcb.fOutxCtsFlow = FALSE;                    // turn on CTS flow control
        dcb.fRtsControl = RTS_CONTROL_DISABLE;    //
        // set DSRDTR
        dcb.fOutxDsrFlow = FALSE;                    // turn on DSR flow control
        //dcb.fDtrControl = DTR_CONTROL_ENABLE;    //
        dcb.fDtrControl = DTR_CONTROL_DISABLE;    //
        //dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;    //
        bPortReady = SetCommState(hComm, &dcb);
        //printf("x3 PortReady=%d\r\n",bPortReady);
        // Communication timeouts are optional
        bPortReady = GetCommTimeouts (hComm, &CommTimeouts);
        //printf("x4 PortReady=%d\r\n",bPortReady);
        CommTimeouts.ReadIntervalTimeout = 5;
        CommTimeouts.ReadTotalTimeoutConstant = 5;
        CommTimeouts.ReadTotalTimeoutMultiplier = 1;
        CommTimeouts.WriteTotalTimeoutConstant = 5;
        CommTimeouts.WriteTotalTimeoutMultiplier = 1;

        bPortReady = SetCommTimeouts (hComm, &CommTimeouts);
        //printf("x5 PortReady=%d\r\n",bPortReady);
#endif

        //On linux you need to open the tty port
#if defined __linux__ || __APPLE__
        printf("Opening Com Port on Linux \n");
        hComm = open(ComPortName,  O_RDWR | O_NOCTTY);
        // Create new termios struc, we call it 'tty' for convention
        struct termios tty;
        memset(&tty, 0, sizeof tty);
        // Read in existing settings, and handle any error
        if(tcgetattr(hComm, &tty) != 0) {
                printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
        }
        tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
        tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most
common)
        tty.c_cflag |= CS8; // 8 bits per byte (most common)
        tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
        tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

        tty.c_lflag &= ~ICANON;
        tty.c_lflag &= ~ECHO; // Disable echo
        tty.c_lflag &= ~ECHOE; // Disable erasure
        tty.c_lflag &= ~ECHONL; // Disable new-line echo
        tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
        tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
        tty.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special
handling of received bytes

        tty.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline
chars)
        tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
        // tty.c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)
        // tty.c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT
ON LINUX)

        tty.c_cc[VTIME] = 10;    // Wait for up to 1s (10 deciseconds), returning as soon as any
data is received.
        tty.c_cc[VMIN] = 0;

        // Set in/out baud rate to be whatever the baudRate variable is
        cfsetispeed(&tty, BaudRate);
        cfsetospeed(&tty, BaudRate);

        // Save tty settings, also checking for error
        if (tcsetattr(hComm, TCSANOW, &tty) != 0) {
                printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
        }
#endif

        return hComm;
}
```

```
char SerialGetc(HANDLE *hComm)
{
	char rxchar;

#ifdef RPI
	if (serialDataAvail(*hComm)) {
		rxchar = serialGetchar(*hComm);
		//fflush(stdout);
	}
	return rxchar;
#else

#ifdef __WIN32__
	bool bReadRC;
	static DWORD iBytesRead;
	do {
		bReadRC = ReadFile(*hComm, &rxchar, 1, &iBytesRead, NULL);
	} while (iBytesRead==0);
	return rxchar;
#endif
#if defined __linux__ || __APPLE__
	// Allocate memory for read buffer, set size according to your needs
	memset(&rxchar, '\0', sizeof(rxchar));
	// Read bytes. The behaviour of read() (e.g. does it block?,
	// how long does it block for?) depends on the configuration
	// settings above, specifically VMIN and VTIME
	int num_bytes = read(*hComm, &rxchar, sizeof(rxchar));
	// n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1
to signal an error.
	if (num_bytes <= 0) {
		//printf("Error reading: %s", strerror(errno));
		rxchar = '\0';
	}
	// Here we assume we received ASCII data, but you might be sending raw bytes (in that
case, don't try and
	// print it to the screen like this!)
	//printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
	//printf("Read %i bytes, rxchar = %c, ASCII = %d ",num_bytes,rxchar,int(rxchar));
	return rxchar;
#endif

#endif

}

void SerialPutc(HANDLE *hComm, char txchar)
{
#ifdef RPI
	serialPutchar(*hComm,txchar);
	fflush(stdout);
	return;
#endif
#ifdef __WIN32__
	BOOL bWriteRC;
	static DWORD iBytesWritten;
	bWriteRC = WriteFile(*hComm, &txchar, 1, &iBytesWritten,NULL);
	return;
#endif
#if defined __linux__ || __APPLE__
	// Write to serial port
	write(*hComm,&txchar,sizeof(txchar));
	return;
#endif
}

void SerialPutString(HANDLE *hComm, char *string,int len) {
	char outchar;
```

```
        outchar = *string++;
        for (int idx = 0;idx<len;idx++){
                SerialPutc(hComm,outchar);
                //printf("%c",outchar);
                outchar = *string++;
        }
}

void SerialPutString(HANDLE *hComm, char *string)
{
        char outchar;
        outchar = *string++;
        //This while loop. Does it always break?
        //I would rather have a for loop
        //I would rather have an input to the code be the length of
        //the string so we just use a for loop
        //The routine above is overloaded with length as an input
        while (outchar!=NULL){
                SerialPutc(hComm,outchar);
                //printf("%c",outchar);
                outchar = *string++;
        }
}

void SerialSendArray(HANDLE *hComm,float number_array[],int num) {
        SerialSendArray(hComm,number_array,num,1);
}

void SerialSendArray(HANDLE *hComm,float number_array[],int num,int echo) {
        union inparser inputvar;
        char outline[20];
        for (int i = 0;i<num;i++) {
                inputvar.floatversion = number_array[i];
                int int_var = inputvar.inversion;
                if (echo) {
                        printf("Sending = %lf %d \n",number_array[i],int_var);
                }
                sprintf(outline,"H:%08x ",int_var);
                if (echo) {
                        printf("Hex = %s \n",outline);
                }
                //This routine uses a while loop until it hits a NULL char
                //SerialPutSrting(hComm,outline);
                //This routine uses a for loop
                SerialPutString(hComm,outline,11); //The 11 here is the number of characters in
the string
                //H: - 2 chars
                //followed by 8 hex chars
                //followed by a space - 1
                //11 total
                //Send a slash r after every number
                SerialPutc(hComm,'\r');
        }
        if (echo) {
                printf("Numbers Sent \n");
        }
}

///////////This is really annoying but when this Serial library was first written, the desktop
////side would send 3 hex numbers at a time and then a \r at the end. The board would then
///respond with 1 hex number at a time with \r at the end. Because of that the SerialPutArray is
///for the desktop to send an array where 3 numbers are followed by \r
///the SerialSendArray is literally the exact same code but it sends 1 number at a time with \r
///at the end. In my opinion, it would be better to send 1 hex number and then \r back and forth
///that way there's no confusion on which routine to use. Problem is that MultiSAT++/HIL is using
//the 3 hex \r format and the RPI Groundstation is using 1 hex \r format. In an effort to not
break
//other people's code I have kept SerialPutArray and SerialSendArray. If we can ever get the
MultiSAT
```

```
//and HIL members in the room together and have a coding party I suggest we change everything to 1
hex \r
//format but for now we will leave this here. CMontalvo 10/13/2020 (This was a Tuesday. Not a
Friday)

void SerialPutArray(HANDLE *hComm,float number_array[],int num) {
        SerialPutArray(hComm,number_array,num,1);
}

void SerialPutArray(HANDLE *hComm,float number_array[],int num,int echo) {
        union inparser inputvar;
        char outline[20];
        int slashr = 0;
        for (int i = 0;i<num;i++) {
                inputvar.floatversion = number_array[i];
                int int_var = inputvar.inversion;
                if (echo) {
                        printf("Sending = %lf %d \n",number_array[i],int_var);
                }
                sprintf(outline,"H:%08x ",int_var);
                if (echo) {
                        printf("Hex = %s \n",outline);
                }
                SerialPutString(hComm,outline);
                slashr++;
                //Send a slash r after every 3rd set of numbers
                if (slashr == 3) {
                        SerialPutc(hComm,'\r');
                        slashr=0;
                }
        }
        if (echo) {
                printf("Numbers Sent \n");
        }
}

//This function will just read everything from the Serial monitor and print it to screen
void SerialGetAll(HANDLE *hComm) {
        char inchar = '\0';
        printf("Waiting for characters \n");
        int i = 0;
        do {
                do {
                        inchar = SerialGetc(hComm);
                        //printf("i = %d inchar = %c chartoint = %d \n",i,inchar,int(inchar));
                } while (inchar == '\0');
                printf("Receiving: i = %d char = %c chartoint = %d \n",i,inchar,int(inchar));
                i++;
        } while ((i<MAXLINE));
        printf("Response received \n");
}

void SerialGetAllPowerSupply(HANDLE *hComm, char currents[50]) {
        char inchar = '\0';
    char inchar_copy = '\0';
    int max_chars_currents = 18;
        printf("Waiting for characters \n");
        int i = 0;
        do {
                do {
                        inchar = SerialGetc(hComm);
                        //printf("i = %d inchar = %c chartoint = %d \n",i,inchar,int(inchar));
                } while (inchar == '\0');
                printf("Receiving: i = %d char = %c chartoint = %d \n",i,inchar,int(inchar));
        currents[i] = inchar;
                i++;
        } while ((i < max_chars_currents));
        printf("Response received \n");
}
```

```
void SerialGetArray(HANDLE *hComm,float number_array[],int num) {
        SerialGetArray(hComm,number_array,num,1);
}

void SerialGetArray(HANDLE *hComm,float number_array[],int num,int echo) {
        union inparser inputvar;
        for (int d = 0;d<num;d++) {
                int i = 0;
                char inLine[MAXLINE];
                char inchar = '\0';
                if (echo) {
                        printf("Waiting for characters \n");
                }
                do {
                        do {
                                inchar = SerialGetc(hComm);
                        } while (inchar == '\0');
                        if (echo) {
                                printf("Receiving: i = %d char = %c chartoint = %d
\n",i,inchar,int(inchar));
                        }
                        inLine[i++] = inchar;
                } while ((inchar != '\r') && (i<MAXLINE));
                if (echo) {
                        printf("Response received \n");
                }

                // Format from Arduino:
                // H:nnnnnnnn

                // Now Convert from ASCII to HEXSTRING to FLOAT
                if (echo) {
                        printf("Converting to Float \n");
                }
                inputvar.inversion = 0;
                for(i=2;i<10;i++){
                        if (echo) {
                                printf("Hex Digit: i = %d char = %c \n",i,inLine[i]);
                        }
                        inputvar.inversion <<= 4;
                        inputvar.inversion |= (inLine[i] <= '9' ? inLine[i] - '0' :
toupper(inLine[i]) - 'A' + 10);
                }
                if (echo) {
                        printf("Integer Received = %d \n",inputvar.inversion);
                        printf(" \n");
                }
                number_array[d] = inputvar.floatversion;
        }
}

void SerialPutHello(HANDLE *hComm,int echo) {
        if (echo) {
                printf("Sending w slash r \n");
        }
        SerialPutc(hComm,'w');
        SerialPutc(hComm,'\r');
        if (echo) {
                printf("Sent \n");
        }
}

int SerialGetHello(HANDLE *hComm,int echo) {
        //Consume w\r\n
        if (echo) {
                printf("Reading the Serial Buffer for w slash r slash n \n");
        }
        char inchar;
        int err = 0;
        for (int i = 0;i<3;i++) {
                inchar = SerialGetc(hComm);
                int val = int(inchar);
                err+=val;
                if (echo) {
                        printf("%d \n",val);
                }
        }
        return err;
}

int SerialListen(HANDLE *hComm) {
        return SerialListen(hComm,1); //default to having echo on
}

int SerialListen(HANDLE *hComm,int echo) {
        //Listen implies that this is a drone/UAV/robot that is simply
        //listening on the airwaves for anyone sending out w \r
        //Listen w\r

        ///////////////THIS WORKS DO NOT TOUCH (RPI ONLY)
        /* char dat;
        if(serialDataAvail(*hComm))
        {
        dat = serialGetchar(*hComm);
        printf("char = %c int(char) = %d \n", dat,int(dat));
}
        return 0;*/
        ///////////////////////////////////////

        int ok = 0;
        char inchar;
        inchar = SerialGetc(hComm);
        int val = int(inchar);
        if ((echo) && (val > 0) && (val < 255)) {
                printf("SerialListen => char = %c int(char) = %d \n", inchar,val);
        }

        if (val == 119) { //That's a w!
                ok += 119;
                //If we received a w we need to read say 10 times and see if we get a \r
                //remember that \r is a 13 in ASCII and \n is 10 in ASCII
                inchar = SerialGetc(hComm);
                val = int(inchar);
                //If we received a 13 or reach max we will break out of this loop
                //There is nothing more we need to do so we will just print val
                //to the screen
                if ((echo) && (val == 13)){
                        printf("Slash R Received!!!! \n");
                }
                if (echo) {
                        printf("Character Received = %c ASCII Code = %d \n",inchar,val);
                }
                //and then increment ok
                ok+=val;
        }
        //either way we shall return ok
        return ok;
}

void SerialDebug(HANDLE *hComm) {
        char inchar;
        inchar = SerialGetc(hComm);
        int val = int(inchar);
        printf("Character Received = %c ASCII Code = %d \n",inchar,val);
```

```
}

void SerialRespond(HANDLE *hComm) {
        //overloaded function just calls the echo on version by default
        SerialRespond(hComm,1);
}

//Responding is very much like SerialPutHello except this is
//board side so this implies that a drone/uav/robot is responding
//to a groundstation computer saying hi.
//the response to hello (w\r) is hello, sir (w\r\n)
void SerialRespond(HANDLE *hComm,int echo) {

        /* THIS WORKS DO NOT TOUCH (RPI ONLY)
        char dat;
        dat = 'w';
        printf("Sending char %c \n",dat);
        serialPutchar(my, dat);
        dat = '\r';
        printf("Sending char %c \n",dat);
        serialPutchar(my, dat);
        dat = '\n';
        printf("Sending char %c \n",dat);
        serialPutchar(my, dat);
        */

        if (echo) {
                printf("Sending w slash r slash n \n");
        }
        SerialPutc(hComm,'w');
        SerialPutc(hComm,'\r');
        SerialPutc(hComm,'\n');
        if (echo) {
                printf("Sent \n");
        }
}
```

```
#ifndef SERIAL_H
#define SERIAL_H

#include <stdio.h>
#include <iostream>

// Flow control flags
#define FC_DTRDSR       0x01
#define FC_RTSCTS       0x02
#define FC_XONXOFF      0x04

// variables used with the com port
#ifdef __WIN32__
//Windows needs some extra stuff
#include <conio.h>
#include <windows.h>
DCB dcb;
COMMTIMEOUTS CommTimeouts;
DWORD iBytesWritten;
DWORD iBytesRead;
bool bPortReady;
bool bWriteRC;
bool bReadRC;
#endif
//Linux just needs to open a file to write to Serial
#if defined __linux__ || __APPLE__
//So this is kind of a hack but basically I change HANDLE to FILE so that
//I can use the exact same function declarations for Linux
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h> // Contains file controls like O_RDWR
#include <errno.h> // Error integer and strerror() function
#include <termios.h> // Contains POSIX terminal control definitions
#include <unistd.h> // write(), read(), close()
#define HANDLE int //this is my hack that I've done so that I can use windows commands on linux
#endif

#ifdef RPI
//sudo apt-get install wiringpi
#include <wiringPi.h>
#include <wiringSerial.h>

//to compile run
//g++ -o run.exe Telemetry.cpp -lwiringPi (this is already in the makefile)
#endif

//Shared my handle for linux/windows
extern HANDLE my;

//This is needed to convert from floats to longs
#ifndef INPARSER_H
#define INPARSER_H
union inparser {
        long inversion;
        float floatversion;
};
#endif

#define MAXFLOATS 10
#define MAXLINE 120


///THE FORMAR FOR SAYING HELLO AND RESPONDING AS WELL AS PUTTING
//AN ARRAY AND READING AN ARRAY REALLY NEEDS TO BE THE SAME
//THERE IS NO REASON REALLY TO HAVE THEM BE DIFFERENT.
//And well unfortunately since sending is w\r and responding is w\r\n
//we have multiple routines for sending and receiving.
```

```
//In addition, the desktop software sends 3 hex numbers followed by \r
//while the board sends 1 hex number followed by a \r. This software was also written
//with cross platform in mind and unfortunately this means that WIN32, Linux, RPI and Arduino
//all have different methods. Again my long term goal is to have the same sending and
//receive format for both board and desktop but since this Serial library is used in multiple
//repos I'm leaving all of these routines for backwards compatibility knowing full well that this
is
//absurdly inefficient and utterly confusing.

//Serial Functions

///////////SERIAL INITIALIZATION/////////////
//Use this if you want to use default ComPortName and BaudRate
void InitSerialPort(void);
//Use this if you want to specify ComPortName and BaudRate
HANDLE SerialInit(char *ComPortName, int BaudRate);

///Get and Receive 1 character
char SerialGetc(HANDLE *hComm);
void SerialPutc(HANDLE *hComm, char txchar);

//This sends a string rather than 1 character
void SerialPutString(HANDLE *hComm, char *string);
//this is an overloaded function that uses a for loop instead of a while loop
void SerialPutString(HANDLE *hComm, char *string,int len);

//This routine reads 1 character and prints to the screen what you received
void SerialDebug(HANDLE *hComm);
//This routine reads everything from serial and prints it to screen. not just 1 character
void SerialGetAll(HANDLE *hComm);
void SerialGetAllPowerSupply(HANDLE *hComm, char currents[50]);

//This is where things get confusing

///Let's assume you have a drone that is constantly listening to a ground station and then
//responding to the groundstation if it receives a "hello" (w\r)
//In this case SerialListen is listening for w\r
int SerialListen(HANDLE *hComm,int echo);
int SerialListen(HANDLE *hComm);
//Serial respond then sends hello, sir (w\r\n)
void SerialRespond(HANDLE *hComm,int echo);
void SerialRespond(HANDLE *hComm);

//Now let's assume you are the ground station and you are saying hello and then
//listening for the response from the drne.
//In this case you need SerialPutHello (w\r)
void SerialPutHello(HANDLE *hComm,int echo);
//and then get the response from the drone or whatever (w\r\n)
int SerialGetHello(HANDLE *hComm,int echo);
//So basically SerialPutHello and SerialRepond are different because SerialPutHello send w\r
//and SerialRespond sends w\r\n which is fucking annoying because if we just sent w\r both ways
//we wouldn't need both of these functions

////Ok then SerialPutArray sends an entire array to a board but uses the format
//put 3 Hex numbers followed by a \r
void SerialPutArray(HANDLE *hComm,float array[],int num);
void SerialPutArray(HANDLE *hComm,float array[],int num,int echo);
//SerialGetArray though assumes the numbers are being received in the format 1 hex number \r
void SerialGetArray(HANDLE *hComm,float array[],int num);
void SerialGetArray(HANDLE *hComm,float array[],int num,int echo);
//Again this is so fucking annoying because since we have 3 hex \r and 1 hex \r we now need
//different routines for a drone

//So this routine here sends an array in the 1 Hex \r format
void SerialSendArray(HANDLE *hComm,float array[],int num);
void SerialSendArray(HANDLE *hComm,float array[],int num,int echo);
//You might be wondering where the read 3 hex \r format function and well I've never needed it
//Right now the MultiSAT++/HIL simulation uses SerialPutArray to send 3 Hex \r to an Arduino.
//The SerialReadArray which read 3 Hex \r is actually currently in a *.ino function. Ugh.
```

```
//The Arduino then sends 1 hex \r using it's own Arduino functions and of course we then just
//use SerialGetArray to decode 1 hex \r format. Big sigh. Cmontalvo 10/13/2020


#endif
```

```
////////////////////////////////////////////////////////////////////
//////////////////////////////////LIBRARIES FOR THE GLOBAL CODE//////
//////////////////////////////////////////////////////////////////////




#include<iostream>

#include "Serial.h"
#include "File.h"
#include "start.h"

#include <string>
#include <math.h>       /* pow */ /* floor */
#include <stdio.h>
#include <time.h>       /* clock_t, clock, CLOCKS_PER_SEC */


#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>

////////////////////////////////////////////////////////////////////
//////////////////////////////////LIBRARIES FOR THE SERIAL CODE//////
//////////////////////////////////////////////////////////////////////

// Linux headers
#include <fcntl.h> // Contains file controls like O_RDWR
#include <errno.h> // Error integer and strerror() function
#include <termios.h> // Contains POSIX terminal control definitions
#include <unistd.h> // write(), read(), close()


using namespace std;

//////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////init all the functions//////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////Static & Dynamic Orbit Simulator/////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////

//Declaration Static Step//
int myPowerSupply_copy;
HANDLE myArduino_copy;
double absCurrents0_init_copy[MAX];
double absCurrents_init_copy[MAX];
char signs0_copy[10] = "";
char signs_copy[10] = "";
int controlTypeArduino_copy = 0;
int controlOutput_copy = 0;
int controlVoltages_copy = 1;
int controlCurrents_copy = 2;
int getControlType_copy = 0;
//Declaration Step Static END//
```

```
//Declaration Dynamic Step//
int myPowerSupply_copy_dynamic;
HANDLE myArduino_copy_dynamic;
double absCurrents0_copy_dynamic[MAX];
int lenB_cut_copy;
char signs0_copy_dynamic[MAX] = "";
char signs_copy_dynamic[555555][MAX];
double absCurrents_copy_dynamic[555555][3];
double dt_copy_dynamic;
int controlCurrents_copy_dynamic = 2;

////Declaration Dynamic Step////

//////////////////////////////////////////////////////////////////////////////
void static_orbit_simulator(double Bxx,double Byy, double Bzz, double Bxx_mag, double Byy_mag,
double Bzz_mag);
void dynamic_orbit_simulator();
void static_orbit_simulator_step1(double Bxx,double Byy, double Bzz, double Bxx_mag, double
Byy_mag, double Bzz_mag);
void static_orbit_simulator_step2();
void static_orbit_simulator_step3();
void static_orbit_simulator_step4();

void dynamic_orbit_simulator_step1();
void init_dynamic_variables_step1(std::vector< double > Bx, std::vector< double > By, std::vector<
double > Bz, double magEarth[3], double camp[3]);
void init_dynamic_variables_step2();




void init_variables(double Bx,double By,double Bz, double x_maglecture, double y_maglecture,
double z_maglecture, char signs0[50], char signs[50], double absCurrents0[3], double
absCurrents[3], string time);
void subtract_matrices(int A1[][3],int B1[][3],int subtracttionAB[][3], int rows, int columns);
void display_array(double A[],int len);
void display_array(char A[100],int len);
void display_array(std::vector< double > A,int len);
void display_array(double A[],int len);
void substract_arrays(double A[], double B[], double res[], int len);
void divide_arrays(double A[], double B[], double res[], int len);
void absolute_value_array(double A[],  double res[], int len);
void get_array_signs(double A[], char signs[], int len);

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////Dynamic Orbit simulator/////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
void unix2julian_conversion(double unix_time, double &JD);
void self_mod(double a,double m, double &res);
void ceci2ecef(double time_JD,  double ceci[3][3]);
void display_3x3matrix(double a[3][3]);
void transpose_matrix(double a[][3],double res[][3], int row,int column);
void multiply_matrices(double a[3][3],double b[3][1], int r1, int c1, int r2, int c2, double
res[3][1]);
void cut_vector(std::vector< double > Bx, std::vector< double > &Bx_res ,int len ,int iniCut);
void set_vectors2matrix(std::vector< double > Bx, std::vector< double > By, std::vector< double >
Bz, double matx[][3], int len);
void init_dynamic_variables(std::vector< double > Bx, std::vector< double > By, std::vector<
double > Bz, double magEarth[3], double camp[3]);
void display_nx3matrix(double a[][3], int len);
void display_nx3matrix(char a[][3], int len);
void repmat3x3(std::vector< double > vector, double matrix[][3], int replen);
void absolute_value_matrix(double A[][3],  double res[][3], int len);
void get_matrix_signs(double matrix[][3], char signs[][3], int len);
void get_row_matrix_nx3(int row, char mat[][3], char res[3]);
```

```
////////////////////////////////////////////////////////////////////////////////////////
///////////serial related functions/////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
void set_string2serialPowerSupply(double values2set[], HANDLE *hComm ,int controlType);
void set_string2serialPowerSupply(char values2set[10], HANDLE *hComm ,int controlType);
void set_string2serialPowerSupply(int values2set[], HANDLE *hComm ,int controlType); //funcion
sobrecargada! para uso de double & int
void set_string_channels2serialArduino(char char2set[50], HANDLE *hComm ,int controlType);
void get_string2serialPowerSupply(HANDLE *hComm, int controlType, double
resCurrentsPowerSupply[3]);
char* passChar(char nameChar[50]);
int open_powerSupply_serial();

//////////////////////////////////////////////////////////
///////////////////////////DECLARE GLOBAL VARIABLES////////
////////////////////////////////////////////////////////////



#define MAX 3

///////////////////////////
////////////////////////////
/////Processing main/////////
//////////////////////////////

start::start()
{
        cout << "lerts'goooo" << endl;

}

/*
void start::set_run_status(int status){
        runStatus = status;

};
*/









void start::start_static_orbit_simulator(double Bxx,double Byy, double Bzz, double Bxx_mag, double
Byy_mag, double Bzz_mag){
static_orbit_simulator( Bxx, Byy, Bzz, Bxx_mag,  Byy_mag, Bzz_mag);
};

void start::start_static_orbit_simulator_step1(double Bxx,double Byy, double Bzz, double Bxx_mag,
double Byy_mag, double Bzz_mag){
static_orbit_simulator_step1( Bxx, Byy, Bzz, Bxx_mag,  Byy_mag, Bzz_mag);
};

void start::start_static_orbit_simulator_step2(){
static_orbit_simulator_step2( );
};


void start::start_static_orbit_simulator_step3(){
static_orbit_simulator_step3( );
};

void start::start_static_orbit_simulator_step4(){
start_static_orbit_simulator_step4( );
};
```

```
void start::start_dynamic_orbit_simulator(){
dynamic_orbit_simulator();
};


void start::start_dynamic_orbit_simulator_step1(){
dynamic_orbit_simulator_step1();
};
void start::start_dynamic_orbit_simulator_step2(){
init_dynamic_variables_step2();
};











void init_dynamic_variables(std::vector< double > Bx, std::vector< double > By, std::vector<
double > Bz, double magEarth[3], double camp[3]){

        double dt = 1;
        double tini = 1377;
        double Nini = round(tini/dt);

        int lenB = Bx.size();
        double magORB_na[3][lenB];

        std::vector< double > Bx_cut;
        std::vector< double > By_cut;
        std::vector< double > Bz_cut;
        cout << "Let's cut vecotr Bx\n";
        cut_vector(Bx,Bx_cut,lenB, Nini);
        cut_vector(By,By_cut,lenB, Nini);
        cut_vector(Bz,Bz_cut,lenB, Nini);

        int lenB_cut = Bx_cut.size();
        double magORB_na_trans[lenB_cut][3];


        cout << "Bx cut size"<< lenB_cut<<endl;

        set_vectors2matrix(Bx_cut,By_cut,Bz_cut,magORB_na_trans,lenB_cut);
        //display_nx3matrix(magORB_na_trans, lenB_cut);


        cout <<"lets cut hereeeeeeeeeeee \n";
//      for (int i = 0; i < lenB_cut ; i++){
//              cout<< "position" << i << "Bx_cut"<<Bx_cut[i]<<endl;
//
//      }

        double magORB_corr[lenB_cut][3];

        for(int i= 0; i < lenB_cut; i++){
                for(int j = 0; j < 3; j++){
                magORB_corr[i][j] = magORB_na_trans[i][j] - magEarth[j];
                }
        }
        //display_nx3matrix(magORB_corr, lenB_cut);

        double magORB_corr0[MAX] = {-magEarth[0], -magEarth[1], -magEarth[2]};
        double current[lenB_cut][3];
        double current0[3];
        for(int i= 0; i<3;i++){
                current0[i] = magORB_corr0[i] / camp[i];

        }

        for(int i= 0; i < lenB_cut; i++){
                for(int j = 0; j < 3; j++){
                        current[i][j] = magORB_corr[i][j] / camp[j];
                }
        }

        for(int i= 0; i<3;i++){
                current0[i] = -1*current0[i];
        }

        for(int i= 0; i < lenB_cut; i++){
                for(int j = 0; j < 3; j++){
                        current[i][j] = -current[i][j];
                }
        }

        //display_nx3matrix(current, lenB_cut);
        cout <<"now\n";
        sleep(2);
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
absCurrents0 = abs(current0);
////////////////////////////////////////////////////////////////////////////////        signs0 =
returnSignsMat(current0);
////////////////////////////////////////////////////////////////////////////////
absCurrents = abs(current);
////////////////////////////////////////////////////////////////////////////////        signs =
returnSignsMat(current);

        double absCurrents0[3];
        double absCurrents[lenB_cut][3];
        absolute_value_matrix(current, absCurrents, lenB_cut);
        absolute_value_array(current0, absCurrents0, 3);
        cout << "lets display the ascurrents\n";
        display_nx3matrix(absCurrents, lenB_cut);
        cout << "lets end display the ascurrents\n";

        char signs0[MAX] = "";
        char signs[lenB_cut][MAX];


        get_matrix_signs(current, signs, lenB_cut);
        get_array_signs(current0, signs0, 3);








        int myPowerSupply = open_powerSupply_serial();

        HANDLE myArduino = SerialInit("/dev/ttyACM0",57600);
        sleep(5);



////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
        //////////////////////////////////////////////////////////////////////////////////// ENABLE
BK CHANNELS AND PREDEFINED VALUES//////////////

////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
///////////////////////////////////////

        int channels_init[MAX] = {1, 1, 1};
//      BKsetOutputEnables([1 1 1]); % Enable [CH1 CH2 CH3]
        double voltages_init[MAX] = {10, 10, 5};
//      BKSetVoltages([10 10 5]);
        double currents_init[MAX] = {1, 1, 1};
//      BKSetCurrents([1 1 1]);% Amperes. Set current of each coil

        int controlOutput = 0;
        int controlVoltages = 1;
        int controlCurrents = 2;

        int channels[10] ;
        int voltages[10] ;
        int currents[10] ;

        channels[0] = channels_init[0];
        channels[1] = channels_init[1];
        channels[2] = channels_init[2];
        voltages[0] = voltages_init[0];
        voltages[1] = voltages_init[1];
        voltages[2] = voltages_init[2];
        currents[0] = currents_init[0];
        currents[1] = currents_init[1];
        currents[2] = currents_init[2];
        char array[10];

        sprintf(array, "%f", 3.123);


        //INIT POWER SUPPLY
        set_string2serialPowerSupply(channels, &myPowerSupply, controlOutput);
        set_string2serialPowerSupply(voltages, &myPowerSupply, controlVoltages);
        set_string2serialPowerSupply(currents, &myPowerSupply, controlCurrents);


        string startStatus;
        cout << "power supply initialized!" << endl;
        cout <<"System start: Do you want to start? enter:Y/N" << endl;
        cin >> startStatus;
        sleep(10);


//      %% INITIALIZE BK AND ARDUINO DRIVERS
//
//              ArduinoAndBkInit();
//
//      %% ENABLE BK CHANNELS AND PREDEFINED VALUES
//
//              BKsetOutputEnables([1 1 1]); % Enable [CH1 CH2 CH3]
//              BKSetVoltages([10 10 5]);
//              BKSetCurrents([1 1 1]);% Amperes. Set current of each coil
//
//
//
//
//
//                      %% INITIALIZE VARIABLES FOR ARDUINO
//                      sign_old = signs0(1);
//              arduinoRelay(signs0(1));
//              % pause(2);
//              BKSetCurrents(absCurrents0);
//              pause(2);
//


        char sign_old[MAX] = "";
        strcat(sign_old, signs0);
        int controlTypeArduino = 0;
        set_string_channels2serialArduino(signs0, &myArduino , controlTypeArduino);//%% SET
Channels




        cout << "lets display signs \n";
        sleep(2);
        display_nx3matrix(signs,lenB_cut);


        double current_meas[lenB_cut] = {0};
    char current_sign[3] = "";
        for(int i = 0; i < lenB_cut; i++){
        clock_t t;

        t = clock();

        double start_time; // clock
                if (strcmp(signs[i], sign_old) != 0){

            get_row_matrix_nx3(i, signs, current_sign);
            cout << "signs in loop!!!!!!!!!!!!!!!!!!!"<<endl;
            display_array(current_sign, 3);

            set_string_channels2serialArduino(current_sign, &myArduino , controlTypeArduino);
            strcpy(sign_old,current_sign);
            cout << "signs_old in loop!!!!!!!!!!!!!!!"<<endl;
            display_array(sign_old, 3);
                        //arduinoRelay(signs(i,:));
                        //sign_old = signs(i,:);
                        cout <<"signs changed\n";


                }


                //% Send current to power supply (only if voltage has changed)
                        //current_now = BKgetMeasuredCurrent();
                int getControlType = 0;
                double received_Currents_PowerSupply[3];
            get_string2serialPowerSupply(&myPowerSupply, getControlType,
received_Currents_PowerSupply);


                if (absCurrents[i][0]!=received_Currents_PowerSupply[0] & absCurrents[i]
[1]!=received_Currents_PowerSupply[1] & absCurrents[i][2]!=received_Currents_PowerSupply[2]){
                        cout << "current changed\n";


            double currents2change[3];
            currents2change[0] = absCurrents[i][0];
            currents2change[1] = absCurrents[i][1];
            currents2change[2] = absCurrents[i][2];
            int controlTypeArduino = 2;
                set_string2serialPowerSupply(currents2change, &myPowerSupply, controlCurrents);
            cout << "current changed\n";

                }

                /*
                % Save measured current
                        currents_meas(i,:) = BKgetMeasuredCurrent();

                */
                t = clock() - t;
                cout <<"paused for:"<< dt-float(t)/CLOCKS_PER_SEC<<"seconds\n";
                sleep(dt - float(t)/CLOCKS_PER_SEC);
        }



}
void get_row_matrix_nx3(int row, char mat[][3], char res[3]){
/*THIS FUNCTIONS SETS A CHAR VECTOR WITH THE VALUES OF THE ROW row IN MATRIX mat: IT IS ONLY VALUD
FOR MATRIX OF SIXE nx3
    res[0] = mat[row][0];
    res[1] = mat[row][1];
    res[2] = mat[row][2];
  */


    for(int i = 0; i < MAX; i++){
        res[i] = mat[row][i];
 }


}
void repmat3x3(std::vector< double > vector, double matrix[][3], int replen){

/////////////////////////////////////////////////////////////////////////////////////////////
        //This function fill the matrix of size replenx3 with the repetitions of the vector
////////

//ie:////////////////////////////////////////////////////////////////////////////////////////
        //vector = [3,4,5] & replen =
7/////////////////////////////////////////////////////////////////
        //matrix = {[3,4,5],[3,4,5],[3,4,5],[3,4,5],[3,4,5],[3,4,5],
[3,4,5]}//////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////

        for(int i = 0; i < replen; i++){
                for(int j=0; j < 3; j++){
                        matrix[i][j] = vector[j];

                }
        }
}

void cut_vector(std::vector< double > Bx, std::vector< double > &Bx_res ,int len ,int iniCut){
        //std::vector< double > Bx_cut;

        for (int i = iniCut; i <= len ; i++){
                Bx_res.push_back(Bx[i]);

        }
        int lenB_cut = Bx_res.size();
        //display_array(Bx_res, lenB_cut);

}
void set_vectors2matrix(std::vector< double > Bx, std::vector< double > By, std::vector< double >
Bz, double matx[][3] , int len){
        double conversion2teslas = pow(10,-9);
        for(int i = 0; i < len; i++){
                for(int j= 0 ; j < 3; j++){
                        switch ( j )
                        {
                        case 0:
                                matx[i][j] = Bx[i]*conversion2teslas;
                                //cout << Bx[i] <<"pos:"<<i<<endl;
                                break;
                        case 1:
                                matx[i][j] = By[i]*conversion2teslas;
                                break;
                        case 2:
                                matx[i][j] = Bz[i]*conversion2teslas;
                                break;
                        }
                }
        }
}

void ceci2ecef(double time_JD, double ceci[3][3]){
        /*
                % Calculate ECI to ECEF rotation quartenion by Julian Date.
                %
                % q = eci2ecef(t)

                % Calculates the ECI->ECEF (z-axis) given a Julian date.
                % Output is within 0 - 2pi boundary.
                % Originally based on eci2ecef.m by raf@control.auc.dk.

                % Rotation of the Earth with respect to the mean equinox is referred to as
                % mean sidereal time (ST). The Earth rotates 360 deg in one sidereal day
                % which is 23h 56m 04.09053s [almanac, b6] in mean solar time.
                % The resulting Earth precession rate is therefore 2*pi/23h 56m 04.09053s
                % Apart from inherent motion of the equinox, due to precession and nutation,
                % ST is a direct measure of the diurnal rotation of the Earth. As a fixpoint
                % for the Earth rotation the Greenwich median transit of the equinox at
                % 0 Jan 1997 is used (JD2450448.5). According to [almanac, b15] the
                % transit takes place at 17h 18m 21.8256s (JD0.7211).
                % The Earth Precession is denoted "omega"
                % The sidereal transit is denoted "s"

        */

//      cout<<"time";
//      printf("%.6f\n", time_JD);

        //% Seconds on a day
        double daysec = 86400.0;
        //% Earth precession [rad/sec]
        double omega = 2*M_PI/86164.09053; //%This value has been correct (KV)
        //% Sidereal epoch (raf's)
        double s = 6.23018356e4/daysec + 2450448.5; //%JD fixpoint
//      cout<<"s";
//      printf("%.10f\n", s);
        //% Time for a revolution
        double T_r = 2*M_PI/omega;


        //% Number of revolutions since epoch
        double N_r = round( ((time_JD-s)*daysec) / T_r);
//      cout<<"heeeeeeeeeeeeeeeeeeeeere"<<endl;
//      cout<<"N_r";
//      printf("%.6f\n", N_r);
//
//      cout<<"T_r";
//      printf("%.6f\n", T_r);
//
//      cout<<"time after round";
//      printf("%.6f\n", time_JD);
//      //cout <<"selfmod"<<psi<<endl;
//      cout <<"omega"<<omega<<endl;
//
//      cout <<"timen"<<time_JD<<endl;
//      cout <<"s"<<s<<endl;
//
//      cout<<"day sec";
//      printf("%.6f\n", daysec);
        //% Time into a revolutio
        double T_n_sec = (time_JD-s)*daysec ;
        double T_n_r = N_r*T_r;
        //double T_n = (time_JD-s)*daysec - N_r*T_r;
//      cout<<"T_n_sec";
//      printf("%f\n",T_n_sec);
//      cout<<"T_n_r";
//      printf("%f\n",T_n_r);

        double T_n;
        T_n = T_n_sec - T_n_r;
        cout<<"T_n";
        printf("%f\n",T_n);
        cout <<"tn"<<T_n<<endl;
        //% Calculate angle with 2pi-boundary

        double psi;
        self_mod( omega*T_n, 2*M_PI , psi);////////////////////////////////////////////double psi
= mod( omega*T_n, 2*pi );

        cout << psi << "this is psi\n";



        ceci[0][0] = cos(psi);
        ceci[0][1] = sin(psi);
        ceci[0][2] = 0.0;
        ceci[1][0] = -sin(psi);
        ceci[1][1] = cos(psi);
        ceci[1][2] = 0.0;
        ceci[2][0] = 0.0;
        ceci[2][1] = 0.0;
        ceci[2][2] = 1;


        cout<<"C inside\n";
        display_3x3matrix(ceci);



        //ceci[3][3] = {{6,6,0},{6, 6, 0}, {0, 0, 1}};

//      Ceci2ecef=[cos(psi),sin(psi), 0;
//        -sin(psi), cos(psi), 0;
//        0, 0, 1];
}
void self_mod(double a,double m, double &res){
        res = a - m*floor(a/m);

}

void unix2julian_conversion(double unix_time, double &JD){
        double JD_0 = 2440587.5000;
        JD = (unix_time/86400.0) + JD_0;


}

char* passChar(char nameChar[50]){
        char newStr[50]= "this";
        strcat(nameChar,newStr);
        cout << "passcahr is :"<< nameChar << endl;
        return nameChar;

}



void get_string2serialPowerSupply(HANDLE *hComm, int controlType, double
resCurrentsPowerSupply[3]){
        //Falta guardar los valores en vez de printear

        char buffer[50] = "MEAS:CURR:ALL?\n";
        switch(controlType) {
        case 0:
                int len = strlen(buffer);
                SerialPutString( hComm, buffer, len);

                /*
                //
                ////////////////////////////////////////////////////////////////////////
                //FALTA GUARDAR LOS VALORES RECIVIDOS EN UNA VARIABLE EN VEZ DE PRINTEAR
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!1
                ///////////////////////////////////////////////////////////////////////7
                //
                */
        char currents_powerSupply[50] = "";
                //SerialGetAll(hComm );
        SerialGetAllPowerSupply(hComm, currents_powerSupply);
                cout << "currents_powerSupply:"<< currents_powerSupply[0]<<currents_powerSupply[1]
<<currents_powerSupply[2]<<endl;
        int len_char_currents = strlen(currents_powerSupply);
        display_array(currents_powerSupply, len_char_currents);
        char current1[5] = "";
        char current2[5] = "";
        char current3[5] = "";

        current1[0] = currents_powerSupply[0];
        current1[1] = currents_powerSupply[1];
        current1[2] = currents_powerSupply[2];
        current1[3] = currents_powerSupply[3];
        current1[4] = currents_powerSupply[4];

        current2[0] = currents_powerSupply[0+6];
        current2[1] = currents_powerSupply[1+6];
        current2[2] = currents_powerSupply[2+6];
        current2[3] = currents_powerSupply[3+6];
        current2[4] = currents_powerSupply[4+6];


        current3[0] = currents_powerSupply[0+12];
        current3[1] = currents_powerSupply[1+12];
        current3[2] = currents_powerSupply[2+12];
        current3[3] = currents_powerSupply[3+12];
        current3[4] = currents_powerSupply[4+12];

        double current1_d = std::atof(current1);
        double current2_d = std::atof(current2);
        double current3_d = std::atof(current3);
        resCurrentsPowerSupply[0] = current1_d;
        resCurrentsPowerSupply[1] = current2_d;
        resCurrentsPowerSupply[2] = current3_d;
        cout << "merged currents"<<endl;
        display_array(resCurrentsPowerSupply,3);

                break;
        }


}


void init_variables(double Bx,double By,double Bz, double x_maglecture, double y_maglecture,
double z_maglecture, char signs0[50], char signs[50], double absCurrents0[MAX], double
absCurrents[MAX],string time){
        cout << "initialiced variables start:" << endl;
        int magField[3] = {0,0,0};
        int arrSize = *(&magField + 1) - magField;
        time = "inf";
        /*
        % time = 100; % seconds
                % Impedances of coils, camp, mag earth offset
                % ALL THE VALUES IN THIS SECTION COULD BE CHANGED DUE TO CALIBRATION OF THE
                % COILS.
                % Not implemented: file to load calibration values
        */
        //Impedances//
        double R_Bx = 12.6*2; // Ohms (Connected on serie)
        double R_By = 11.6*2; // Ohms (Connected on serie)
        double R_Bz = 10.6;   // Ohms (Connected on parallel)
        double R_B[3] = {R_Bx, R_By, R_Bz};
        // MagField/Intensity Relation //
        double x_camp = 200e-6; // Teslas/Ampers
        double y_camp = 200e-6; // Teslas/Ampers
        double z_camp = 200e-6; // Teslas/Ampers
        double camp[3]  = {x_camp, y_camp, z_camp};
        //Offset magnetic field//
//      double x_maglecture = 0;
//      double y_maglecture = 0;
//      double z_maglecture = 0;
        double x_magEarth = x_maglecture*6e-6; //%Teslas
        double y_magEarth = y_maglecture*6e-6;
        double z_magEarth = z_maglecture*6e-6;
        double magEarth[3] = {x_magEarth, y_magEarth, z_magEarth};
        ///////////////////////////////////////////////7
        // Not implemented: file to load offset value
        ////////////////////////////////////////////77/
        //% Calculate voltages: absolute value and sign
//      double Bx=-50e+03;//%nT
//      double By=-50e+03;//%nT
//      double Bz=-70e+03;//%nT
        double conversion2teslas = pow(10,-9);

        double magORB[MAX] = {Bx*conversion2teslas, By*conversion2teslas, Bz*conversion2teslas};
//{Bx, By, Bz}*(10^(-9)    %T
        double magORB_corr[MAX];
        substract_arrays(magORB , magEarth, magORB_corr, arrSize);                             //
magORB_corr = magORB - magEarth;

/*
        //% voltages = magField_corr ./ camp;
        //% absVoltages = abs(voltages);
        //% signs = returnSignsMat(voltages);
        */

        double zeros[MAX] = {0};
//zeros = [0,0,0]
        double magORB_corr0 [MAX];
        double current0[MAX];
        double current[MAX];



        substract_arrays(zeros , magEarth, magORB_corr0, arrSize);
//magORB_corr0 = zeros - magEarth
        divide_arrays(magORB_corr , camp , current, arrSize);
//current =  magORB_corr ./ camp;
        divide_arrays(magORB_corr0 , camp, current0, arrSize);
//current0 = magORB_corr0 ./ camp;
        substract_arrays(zeros , current, current, arrSize);
//% To correct coil polarity current = -current
        substract_arrays(zeros , current0, current0, arrSize);
//current0(1,:)= -current0(1,:);
        ////////////////////////////////////////////
        absolute_value_array(current0, absCurrents0 , arrSize);
//absCurrents0 = abs(current0);
        display_array(current0, arrSize);
        get_array_signs(current0, signs0, arrSize);
//signs0 = returnSignsMat(current0);
```

```
        //printf("%s\n", signs0);

        absolute_value_array(current, absCurrents, arrSize);
//absCurrents = abs(current);
        display_array(absCurrents, arrSize);

//% current = magORB_corr ./ camp;

//% absCurrents = abs(current);
        get_array_signs(current, signs, arrSize);
//signs = returnSignsMat(current);
        //printf("%c\n", signs);
        cout << "values of current:"  << endl;
        display_array(current,3);
        cout << "values of current0:" << endl;
        display_array(current0,3);
        cout << "values of signs0:" << signs0 << endl;
        cout << "values of signs:" << signs << endl;
        cout << "initialiced variables: finished" << endl;


        //char* signs02return = new char[50];
        //strcpy(signs02return, signs0);
}


/////////////////////////77///////////////
//DECLARE ALL FUNCTIONS//
/////////////////////////////////////////7


int open_powerSupply_serial(){


        ////////////////////////////////////////////7
        ///////////////////////////////////////////////INIT serial with power supply
        ///////////777//////////////////////////////

        // Open the serial port. Change device path as needed (currently set to an standard FTDI
USB-UART cable type device)
        int myPowerSupply = open("/dev/ttyUSB0", O_RDWR);

        // Create new termios struct, we call it 'tty' for convention
        struct termios tty;

        // Read in existing settings, and handle any error
        if(tcgetattr(myPowerSupply, &tty) != 0) {
                printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));

        }

        tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
        tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most
common)
        tty.c_cflag &= ~CSIZE; // Clear all bits that set the data size
        tty.c_cflag |= CS8; // 8 bits per byte (most common)
        tty.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
        tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

        tty.c_lflag &= ~ICANON;
        tty.c_lflag &= ~ECHO; // Disable echo
        tty.c_lflag &= ~ECHOE; // Disable erasure
        tty.c_lflag &= ~ECHONL; // Disable new-line echo
        tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
        tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
        tty.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special
handling of received bytes

        tty.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline
chars)
        tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
        // tty.c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)
        // tty.c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT
ON LINUX)

        tty.c_cc[VTIME] = 10;    // Wait for up to 1s (10 deciseconds), returning as soon as any
data is received.
        tty.c_cc[VMIN] = 0;

        // Set in/out baud rate to be 9600
        cfsetispeed(&tty, B38400);
        cfsetospeed(&tty, B38400);

        // Save tty settings, also checking for error
        if (tcsetattr(myPowerSupply, TCSANOW, &tty) != 0) {
                printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));

        }



        return myPowerSupply;



        /////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////finish initialization of SERIAL for the
powersupply//////////////////////







}

void set_string_channels2serialArduino(char char2set[50], HANDLE *hComm ,int controlType){
        int n;
        char buffer[50] = "";
        switch(controlType) {
        case 0:
                n = sprintf(buffer,"<%c%c%c>\n", char2set[0],char2set[1],char2set[2]);

                int len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;
        printf ("%c is a string %d chars long\n",buffer,n);
        cout << "The message sent to Arduino1 via serial is:"<< buffer<< endl;
        }

}


void set_string2serialPowerSupply(char values2set[10], HANDLE *hComm ,int controlType){

        /*
        //controlType = 0  -------> BKSetOutputEnable
        //controlType = 1  -------> BKSetVoltages
        //controlType = 2  -------> BKSetCurrents
        */
        char buffer[50] = "";
        int n;
        int len;
    cout <<"char set serial type"<<endl;
```

```
        switch(controlType){
        case 0:

                n = sprintf(buffer,"APP:OUT %d,%d,%d\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 1:

                n = sprintf(buffer,"APP:VOLT %0.3f,%0.3f,%0.3f \n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 2:

                n = sprintf(buffer,"APP:CURR %0.3f,%0.3f,%0.3f\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;
        }
        cout << "The message sent to powerSuplly via serial is:"<< buffer<< endl;

}

void set_string2serialPowerSupply(int values2set[], HANDLE *hComm ,int controlType){

        /*
        //controlType = 0  -------> BKSetOutputEnable
        //controlType = 1  -------> BKSetVoltages
        //controlType = 2  -------> BKSetCurrents
        */
        char buffer[50];
        int n;
        int len;
        switch(controlType) {
        case 0:

                n = sprintf(buffer,"APP:OUT %d,%d,%d\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 1:

                n = sprintf(buffer,"APP:VOLT %0.3d,%0.3d,%0.3d\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 2:

                n = sprintf(buffer,"APP:CURR %0.3d,%0.3d,%0.3d\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;
        }
        cout << "The message sent to powerSuplly via serial is:"<< buffer<< endl;

}

void set_string2serialPowerSupply(double values2set[], HANDLE *hComm ,int controlType){

        /*
        //controlType = 0  -------> BKSetOutputEnable
        //controlType = 1  -------> BKSetVoltages
        //controlType = 2  -------> BKSetCurrents
        */
        char buffer[50];
        int n;
        int len;

        switch(controlType) {
        case 0:

                n = sprintf(buffer,"APP:OUT %f,%f,%f\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 1:

                n = sprintf(buffer,"APP:VOLT %0.3f,%0.3f,%0.3f\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;

        case 2:

                n = sprintf(buffer,"APP:CURR %0.3f,%0.3f,%0.3f\n", values2set[0], values2set[1],
values2set[2] );
                printf ("%s is a string %d chars long\n",buffer,n);
                len = strlen(buffer);
                SerialPutString( hComm, buffer, len);
                break;
        }
        cout << "The message sent to powerSuplly via serial is:"<< buffer<< endl;

}
void subtract_matrices(int A1[][3],int B1[][3],int subtracttionAB[][3], int rows, int columns){



        int c, d;
    int m = rows;
        int n = columns;
        cout << "rows:"<< m << "columns:" << n << endl;
        for ( c = 0 ; c < m ; c++ )
                for ( d = 0 ; d < n ; d++ )
                        subtracttionAB[c][d] = A1[c][d] + B1[c][d];
}

void transpose_matrix(double a[][3],double res[][3], int row,int column){
        // Computing transpose of the matrix
        for (int i = 0; i < row; ++i)
                for (int j = 0; j < column; ++j) {
                        res[j][i] = a[i][j];
        }
```

```
}

void multiply_matrices(double a[3][3],double b[3][1], int r1, int c1, int r2, int c2, double
res[3][1]){
        // Initializing elements of matrix mult to 0.
        for(int i = 0; i < r1; ++i)
                for(int j = 0; j < c2; ++j)
        {
                res[i][j]=0;
        }

                // Multiplying matrix a and b and storing in array mult.
                for(int i = 0; i < r1; ++i)
                        for(int j = 0; j < c2; ++j)
                        for(int k = 0; k < c1; ++k)
                {
                        res[i][j] += a[i][k] * b[k][j];
                }



}
void display_nx3matrix(double a[][3], int len){

        int row = len;
        int column = 3;
        for (int i = 0; i < row; ++i) {
                for (int j = 0; j < column; ++j) {
                        cout << " " << a[i][j]<< "position:"<< i<< endl;;
                        if (j == column - 1)
                                cout << endl << endl;
                }
        }


}
void display_nx3matrix(char a[][3], int len){

        int row = len;
        int column = 3;
        for (int i = 0; i < row; ++i) {
                for (int j = 0; j < column; ++j) {
                        cout << " " << a[i][j]<< "position:"<< i<< endl;;
                        if (j == column - 1)
                                cout << endl << endl;
                }
        }


}

void display_3x3matrix(double a[3][3]){
        int row = 3;
        int column = 3;
        for (int i = 0; i < row; ++i) {
                for (int j = 0; j < column; ++j) {
                        cout << " " << a[i][j];
                        if (j == column - 1)
                                cout << endl << endl;
                }
        }

}


void display_array(char A[100],int len){
//Displaying output
        for(int i = 0; i < len; ++i)
        {
                cout << A[i] << "  ";
                if(i == len - 1)
                        cout << endl;
        }




}

void display_array(double A[],int len){
        //Displaying output
        for(int i = 0; i < len; ++i)
        {
                cout << A[i] << "  ";
                if(i == len - 1)
                        cout << endl;
        }
}

void display_array(std::vector< double > A,int len){
        //Displaying output
        for(int i = 0; i < len; ++i)
        {
                cout << A[i] << "  ";
                if(i == len - 1)
                        cout << endl;
        }
}


void substract_arrays(double A[], double B[], double res[], int len){


        int n;
        for(n = 0; n < len; n++){
                res[n] = A[n] - B[n];
        }
}


void divide_arrays(double A[], double B[], double res[], int len){


        int n;
        for(n = 0; n < len; n++){
                res[n] = A[n] / B[n];
        }
}
void absolute_value_matrix(double A[][3],  double res[][3], int len){


        for(int i = 0; i < len; i++){
                for(int j = 0; j < 3; j++){
                res[i][j] = abs(A[i][j]);
                }
        }
}



void absolute_value_array(double A[],  double res[], int len){


        int n;
        for(n = 0; n < len; n++){
                res[n] = abs(A[n]);
        }
```

```
}

void get_matrix_signs(double matrix[][3], char signs[][MAX], int len){
/*
THIS FUNCTIONS RETURN A MATRIx SIGS WITH THE RESPECTIV E SIGNS OF THE VALUES IN  MATRIX: VALID FOR
nx3 MATRICES!
i.e: signs = {{"+++"},{-+-},{+-+},{---},etc}

*/



        for(int i = 0; i < len; i++){
                double rowArr[MAX];
                char rowSigns[MAX] = "";
                for(int j = 0; j < 3; j++){
                        rowArr[j] = matrix[i][j];
                }
                get_array_signs(rowArr, rowSigns, MAX);
                for(int k = 0;k < 3; k++){
                        signs[i][k] = rowSigns[k];
                }

        }


}

void get_array_signs(double A[], char signs[MAX], int len){

        char plus[MAX] = "";
        plus[0] = '+';
        char minus[MAX] = "";
        minus[0] = '-';

        int n;
        for(n = 0; n < len; n++){
                if (A[n] > 0)
                        strcat(signs,plus);
//signs = signs + "+";
                else
                        strcat(signs,minus);
//signs = signs + "-";
        }

}
void static_orbit_simulator_step1(double Bxx,double Byy, double Bzz, double Bxx_mag, double
Byy_mag, double Bzz_mag){
        ////////////////////////////////////////////////////////////////////////////Declare
desired MagneticFields
                double Bx = +60e+03;//Bxx;//+60e+03;//%nT
                double By = +50e+03;//Byy;//+50e+03;//%nT
                double Bz = -20e+03;//Bzz;//-20e+03;//%nT

////////////////////////////////////////////////////////////////////////////Declare actual Earth
MagneticField inside the lab with the coils OFF
                double x_maglecture = 0;//Bxx_mag;//0;
                double y_maglecture = 0;//Byy_mag;//0;
                double z_maglecture = 0;//Bzz_mag;//0;
                ////////////////////////////////////////////////////////////////////////////Init
all the currents and variable
                //char signs0[10] = "";
                //char signs[10] = "";
                //double absCurrents0_init[MAX];
                //double absCurrents_init[MAX];
                string time = "";
                init_variables(Bx, By, Bz, x_maglecture, y_maglecture, z_maglecture, signs0_copy,
signs_copy, absCurrents0_init_copy, absCurrents_init_copy, time);
                cout << "signs !!!!!!!!!!!:"<< signs_copy<<endl;
                //printf("%s\n", signs0);
```

```
//* outputs an 'o' character */
                cout << "absCurrents0 is:";
                display_array(absCurrents0_init_copy,MAX);

                cout << "absCurrents is:";
                display_array(absCurrents_init_copy,MAX);








                myPowerSupply_copy = open_powerSupply_serial();









//////////////////////////////////////////////////////////////////////////////7/////////////////
START SERIALS///////////////////////
            //HANDLE myPowerSupply = SerialInit("/dev/ttyUSB0",38400);


        myArduino_copy = SerialInit("/dev/ttyACM0",57600);///dev/ttyACM0
        sleep(5);



////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
        //////////////////////////////////////////////////////////////////////////////// ENABLE
BK CHANNELS AND PREDEFINED VALUES//////////////

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

                int channels_init[MAX] = {1, 1, 1};
//      BKsetOutputEnables([1 1 1]); % Enable [CH1 CH2 CH3]
                double voltages_init[MAX] = {10, 10, 5};
//      BKSetVoltages([10 10 5]);
                double currents_init[MAX] = {1, 1, 1};
//      BKSetCurrents([1 1 1]);% Amperes. Set current of each coil

                controlOutput_copy = 0;
                controlVoltages_copy = 1;
                controlCurrents_copy = 2;


                int channels[10] ;
                int voltages[10] ;
                int currents[10] ;


                channels[0] = channels_init[0];
                channels[1] = channels_init[1];
                channels[2] = channels_init[2];
                voltages[0] = voltages_init[0];
                voltages[1] = voltages_init[1];
                voltages[2] = voltages_init[2];
                currents[0] = currents_init[0];
                currents[1] = currents_init[1];
                currents[2] = currents_init[2];
                char array[10];
                sprintf(array, "%f", 3.123);


                //INIT POWER SUPPLY
                set_string2serialPowerSupply(channels, &myPowerSupply_copy, controlOutput_copy);
                set_string2serialPowerSupply(voltages, &myPowerSupply_copy, controlVoltages_copy);
                set_string2serialPowerSupply(currents, &myPowerSupply_copy, controlCurrents_copy);


}


void static_orbit_simulator_step2(){
        string startStatus;
        cout << "power supply initialized!" << endl;
        cout <<"The system start: Do you want start? enter:Y/N" << endl;
        //cin >> startStatus;

        ///use ecternal function

        sleep(2);

///////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////
        /////////////////////////////////////////////////////////////////////////////%% INITIALIZE
VARIABLES FOR ARDUINO/////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////

  double absCurrents0[10];
  double absCurrents[10];



  absCurrents0[0] = absCurrents0_init_copy[0];
  absCurrents0[1] = absCurrents0_init_copy[1];
  absCurrents0[2] = absCurrents0_init_copy[2];
  absCurrents[0] = absCurrents_init_copy[0];
  absCurrents[1] = absCurrents_init_copy[1];
  absCurrents[2] = absCurrents_init_copy[2];


        controlTypeArduino_copy = 0;
        set_string_channels2serialArduino(signs0_copy, &myArduino_copy ,
controlTypeArduino_copy);//%% SET Channels
        set_string2serialPowerSupply(absCurrents0, &myPowerSupply_copy, controlCurrents_copy);//%%
SET currents VALUE
        sleep(4); ///////////////////////////////////////////////////////////////// set a doze
time to stabilize arduino1

        cout << "PowerSupply correctlly stabilixed" << endl;
  cout << "printing absCurrents0 !!!!!!!!!!"<< absCurrents0_init_copy[0] <<
absCurrents0_init_copy[1] <<absCurrents0_init_copy[2]<< endl;


        set_string_channels2serialArduino(signs_copy, &myArduino_copy, controlTypeArduino_copy);
//% Save currents measured
        set_string2serialPowerSupply(absCurrents, &myPowerSupply_copy, controlCurrents_copy);
        getControlType_copy = 0;
    sleep(5);
    cout << "printing absCurrents !!!!!!!!!!"<< absCurrents[0] << absCurrents[1] <<absCurrents[2]
<< endl;

}

void static_orbit_simulator_step3(){

          string ListeningStatus;
```

```
                cout << "Listening" << endl;
          cout <<"The system will Listen: Do you want get the values? Enter:Y/N" << endl;
          //cin >> ListeningStatus;

          double received_Currents_PowerSupply[3];
                get_string2serialPowerSupply(&myPowerSupply_copy, getControlType_copy,
received_Currents_PowerSupply);
                sleep(10);

}
void static_orbit_simulator_step4(){

        cout << "The system will restart: Do you want to reset to default 0 values? enter:Y/N" <<
endl;
        string closeStatus;
        //cin >> closeStatus;

        double voltages2close[3] = {0, 0, 0};
        double currents2close[3] = {0, 0, 0};
        double signs2close[3] = {1, 1, 1};

        set_string2serialPowerSupply(voltages2close, &myPowerSupply_copy, controlVoltages_copy);
        set_string2serialPowerSupply(currents2close, &myPowerSupply_copy, controlCurrents_copy);
        char signs2closeArduino[MAX];
        get_array_signs(signs2close, signs2closeArduino, MAX);
        set_string_channels2serialArduino(signs2closeArduino, &myArduino_copy,
controlTypeArduino_copy);
}

void static_orbit_simulator(double Bxx,double Byy, double Bzz, double Bxx_mag, double Byy_mag,
double Bzz_mag){
/////////////////////////////////////////////////////////////////////////////Declare desired
MagneticFields
        double Bx = Bxx;//+60e+03;//%nT
        double By = Byy;//+50e+03;//%nT
        double Bz = Bzz;//-20e+03;//%nT
        /////////////////////////////////////////////////////////////////////////////Declare
actual Earth MagneticField inside the lab with the coils OFF
        double x_maglecture = Bxx_mag;//0;
        double y_maglecture = Byy_mag;//0;
        double z_maglecture = Bzz_mag;//0;
        /////////////////////////////////////////////////////////////////////////////Init all the
currents and variable
        char signs0[10] = "";
        char signs[10] = "";
        double absCurrents0_init[MAX];
        double absCurrents_init[MAX];
        string time = "";
        init_variables(Bx, By, Bz, x_maglecture, y_maglecture, z_maglecture, signs0, signs,
absCurrents0_init, absCurrents_init, time);
        cout << "signs !!!!!!!!!!!:"<< signs<<endl;
        //printf("%s\n", signs0);
//* outputs an 'o' character */
        cout << "absCurrents0 is:";
        display_array(absCurrents0_init,MAX);

        cout << "absCurrents is:";
        display_array(absCurrents_init,MAX);








        int myPowerSupply = open_powerSupply_serial();
```

```
///////////////////////////////////////////////////////////////////////7/////////////////
START SERIALS//////////////////////
    //HANDLE myPowerSupply = SerialInit("/dev/ttyUSB0",38400);


        HANDLE myArduino = SerialInit("/dev/ttyACM0",57600);///dev/ttyACM0
        sleep(5);


////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////// ENABLE BK
CHANNELS AND PREDEFINED VALUES///////////////
////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

        int channels_init[MAX] = {1, 1, 1};
//      BKsetOutputEnables([1 1 1]); % Enable [CH1 CH2 CH3]
        double voltages_init[MAX] = {10, 10, 5};
//      BKSetVoltages([10 10 5]);
        double currents_init[MAX] = {1, 1, 1};
//      BKSetCurrents([1 1 1]);% Amperes. Set current of each coil

        int controlOutput = 0;
        int controlVoltages = 1;
        int controlCurrents = 2;

        //char channels[10] = "";
        //char voltages[10] = "";
        //char currents[10] = "";
        //channels[10] = '11';
        //voltages[10] = '10';
        //currents[10] = '11';
        int channels[10] ;
        int voltages[10] ;
        int currents[10] ;
//
/*
        channels[0] = 1;
        channels[1] = 1;
        channels[2] = 1;
        voltages[0] = 1;
        voltages[1] = 1;
        voltages[2] = 1;
        currents[0] = 1;
        currents[1] = 1;
        currents[2] = 1;
        char array[10];
*/

        channels[0] = channels_init[0];
        channels[1] = channels_init[1];
        channels[2] = channels_init[2];
        voltages[0] = voltages_init[0];
        voltages[1] = voltages_init[1];
        voltages[2] = voltages_init[2];
        currents[0] = currents_init[0];
        currents[1] = currents_init[1];
        currents[2] = currents_init[2];
        char array[10];

        sprintf(array, "%f", 3.123);
        //INIT POWER SUPPLY
        set_string2serialPowerSupply(channels, &myPowerSupply, controlOutput);
        set_string2serialPowerSupply(voltages, &myPowerSupply, controlVoltages);
        set_string2serialPowerSupply(currents, &myPowerSupply, controlCurrents);
        string startStatus;
        cout << "power supply initialized!" << endl;
        cout <<"The system start: Do you want start? enter:Y/N" << endl;


        cin >> startStatus;
///use ecternal function

        sleep(2);

///////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////
        /////////////////////////////////////////////////////////////////////%% INITIALIZE
VARIABLES FOR ARDUINO//////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////

    double absCurrents0[10];
    double absCurrents[10];




    absCurrents0[0] = absCurrents0_init[0];
    absCurrents0[1] = absCurrents0_init[1];
    absCurrents0[2] = absCurrents0_init[2];
    absCurrents[0] = absCurrents_init[0];
    absCurrents[1] = absCurrents_init[1];
    absCurrents[2] = absCurrents_init[2];


        int controlTypeArduino = 0;
        set_string_channels2serialArduino(signs0, &myArduino , controlTypeArduino);//%% SET
Channels
        set_string2serialPowerSupply(absCurrents0, &myPowerSupply, controlCurrents);//%% SET
currents VALUE
        sleep(4); ///////////////////////////////////////////////////////////////// set a doze
time to stabilize arduino1

        cout << "PowerSupply correctlly stabilixed" << endl;
    cout << "printing absCurrents0 !!!!!!!!!!"<< absCurrents0[0] << absCurrents0[1]
<<absCurrents0[2]<< endl;


        set_string_channels2serialArduino(signs, &myArduino, controlTypeArduino);       //% Save
currents measured
        set_string2serialPowerSupply(absCurrents, &myPowerSupply, controlCurrents);
        int getControlType = 0;
    sleep(5);
    cout << "printing absCurrents !!!!!!!!!!"<< absCurrents[0] << absCurrents[1] <<absCurrents[2]
<< endl;


    string ListeningStatus;
        cout << "Listening" << endl;
    cout <<"The system will Listen: Do you want get the values? Enter:Y/N" << endl;
    cin >> ListeningStatus;


    double received_Currents_PowerSupply[3];
        get_string2serialPowerSupply(&myPowerSupply, getControlType,
received_Currents_PowerSupply);
```

```
//      string inf_str = "inf";
//      if (!strcmp(time,inf_str)){
//
//              sleep(time);
//
//      }

        //%% Turn off power
        //% Set Voltage and Intensity to 0
        sleep(10);
    cout << "The system will restart: Do you want to reset to default 0 values? enter:Y/N" <<
endl;
    string closeStatus;
    cin >> closeStatus;

        double voltages2close[3] = {0, 0, 0};
        double currents2close[3] = {0, 0, 0};
        double signs2close[3] = {1, 1, 1};


        set_string2serialPowerSupply(voltages2close, &myPowerSupply, controlVoltages);
        set_string2serialPowerSupply(currents2close, &myPowerSupply, controlCurrents);
        char signs2closeArduino[MAX];
        get_array_signs(signs2close, signs2closeArduino, MAX);
        set_string_channels2serialArduino(signs2closeArduino, &myArduino, controlTypeArduino);
        //////////////////////////////////////////////////////////////////////////// PLEASE CLOSE
PORTS
        //% Close ports
        //              BKClose;
        //      ArduinoClose;
        //      % exit();
        //



/////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

//////////////////////////////////////////////////////////////////////////////////END/////////////////
/////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////
        }



void dynamic_orbit_simulator_step1(){
        double dt;
                        dt = 0.25;
                        /*
                        %% Impedances of coils, camp, mag earth offset
                                % ALL THE VALUES IN THIS SECTION COULD BE CHANGED DUE TO
CALIBRATION OF THE
                                % COILS.
                                % Not implemented: file to load calibration values

                                % Impedances
                        */
                        //Impedances//
                        double R_Bx = 12.6*2; // Ohms (Connected on serie)
                        double R_By = 11.6*2; // Ohms (Connected on serie)
                        double R_Bz = 10.6;   // Ohms (Connected on parallel)
                        double R_B[3] = {R_Bx, R_By, R_Bz};
                        // MagField/Intensity Relation //
                        double x_camp = 200e-6; // Teslas/Ampers
                        double y_camp = 200e-6; // Teslas/Ampers
```

```
			double z_camp = 200e-6; // Teslas/Ampers
			double camp[3] = {x_camp, y_camp, z_camp};
			//Offset magnetic field//
			//	double x_maglecture = 0;
			//	double y_maglecture = 0;
			//	double z_maglecture = 0;
			double x_magEarth = 17.538e-6; //%Teslas
			double y_magEarth =-19.05e-6;
			double z_magEarth = -35.53e-6;
			double magEarth[3] = {x_magEarth, y_magEarth, z_magEarth};






			File file;
			char fileName[12] = "";
			strcpy(fileName, "myData.txt");
			file.readFile(fileName);

			double tini = 1377;
			double Nini = round(tini/dt);


			std::vector< double > Bxx;
			std::vector< double > Byy;
			std::vector< double > Bzz;
			Bxx = file.Bxx;
			Byy = file.Byy;
			Bzz = file.Bzz;

			int arrSize = Bzz.size();
			cout <<"size"<<arrSize<<endl;
			display_array(Bzz,arrSize);




			//%% Canvis Edgard

			double unix_time=1584662400.0;  //% Be carefull_ Depending on orbit: %
 unix = 1584662400 Orbits: M6P and Equ
			//unix=1583887278;  % Orbit: R2
			double time;
			std::vector< double > Bx;
			std::vector< double > By;
			std::vector< double > Bz;
			std::vector< double > TIME;






			for (int i = 0; i < 22746; i++){//22746
				if (i==0){
					time = unix_time;
				}
				double vector[3][1] = {{Bxx[i]},{Byy[i]},{Bzz[i]}};

				double JD = 0.0;
				unix2julian_conversion(time, JD );


	//			if (i==0){
	//			cout<<"JD\n";
	//			printf("%.6f", JD);
	//			}
				double C[3][3];
				//double C[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

				ceci2ecef(JD, C);


				if (i==0){
				display_3x3matrix(C);
				}
				double C_trans[3][3];
				int rows_C_trans = 3;
				int columns_C_trans = 3;
				transpose_matrix(C, C_trans, rows_C_trans, columns_C_trans);
	//			if (i==22745){
	//				cout<< JD << "JD\n";
	//				cout<<"C_trans\n";
	//				display_3x3matrix(C_trans);
	//			}



				double BB[3][1];
				multiply_matrices(C_trans, vector, 3, 3, 3, 1, BB);

	//

				if (i>=0){
	//					cout <<"BB is :\n";
	//					cout << BB[0][0]<<" "<<BB[1][0]<<" "<<BB[2][0]<< endl;
	//					cout <<"vector is :\n";
	//					cout << vector[0][0]<<" "<<vector[1][0]<<" "<<vector[2][0]
<< endl;
				}

	//			double BB[3];
	//			BB =
	//
				Bx.push_back(BB[0][0]);
				By.push_back(BB[1][0]);
				Bz.push_back(BB[2][0]);
				TIME.push_back(time);
				time = time + dt;

			}
	//		int len_B  = Bx.size();
	//		cout << "display B please\n";
	//		display_array(Bx, len_B);
	//
	init_dynamic_variables_step1(Bx,By,Bz,magEarth,  camp);


}
void init_dynamic_variables_step1(std::vector< double > Bx, std::vector< double > By, std::vector<
double > Bz, double magEarth[3], double camp[3]){

	  dt_copy_dynamic = 1;
		double tini = 1377;
		double Nini = round(tini/dt_copy_dynamic);

		int lenB = Bx.size();
		double magORB_na[3][lenB];

		std::vector< double > Bx_cut;
		std::vector< double > By_cut;
		std::vector< double > Bz_cut;
		cout << "Let's cut vecotr Bx\n";
		cut_vector(Bx,Bx_cut,lenB, Nini);
		cut_vector(By,By_cut,lenB, Nini);
		cut_vector(Bz,Bz_cut,lenB, Nini);
		int lenB_cut = Bx_cut.size();
		lenB_cut_copy = lenB_cut;
		double magORB_na_trans[lenB_cut][3];


		cout << "Bx cut size"<< lenB_cut<<endl;

		set_vectors2matrix(Bx_cut,By_cut,Bz_cut,magORB_na_trans,lenB_cut);
		//display_nx3matrix(magORB_na_trans, lenB_cut);


		cout <<"lets cut hereeeeeeeeeeee \n";
	//	for (int i = 0; i < lenB_cut ; i++){
	//		cout<< "position" << i << "Bx_cut"<<Bx_cut[i]<<endl;
	//
	//	}

		double magORB_corr[lenB_cut][3];

		for(int i= 0; i < lenB_cut; i++){
			for(int j = 0; j < 3; j++){
			magORB_corr[i][j] = magORB_na_trans[i][j] - magEarth[j];
			}
		}
		//display_nx3matrix(magORB_corr, lenB_cut);

		double magORB_corr0[MAX] = {-magEarth[0], -magEarth[1], -magEarth[2]};
		double current[lenB_cut][3];

		double current0[3];
		for(int i= 0; i<3;i++){
			current0[i] = magORB_corr0[i] / camp[i];

		}

		for(int i= 0; i < lenB_cut; i++){
			for(int j = 0; j < 3; j++){
				current[i][j] = magORB_corr[i][j] / camp[j];
			}
		}

		for(int i= 0; i<3;i++){
			current0[i] = -1*current0[i];
		}

		for(int i= 0; i < lenB_cut; i++){
			for(int j = 0; j < 3; j++){
				current[i][j] = -current[i][j];
			}
		}

		//display_nx3matrix(current, lenB_cut);
		cout <<"now\n";
		sleep(2);
	////////////////////////////////////////////////////////////////////////////////
	////////////////////////////////////////////////////////////////////////////////
absCurrents0 = abs(current0);
	////////////////////////////////////////////////////////////////////////////////
signs0 = returnSignsMat(current0);
	////////////////////////////////////////////////////////////////////////////////
absCurrents = abs(current);
	////////////////////////////////////////////////////////////////////////////////
signs = returnSignsMat(current);

		double absCurrents0_copy_dynamic[3];
		//double absCurrents_copy_dynamic[lenB_cut][3];
		absolute_value_matrix(current, absCurrents_copy_dynamic, lenB_cut);
		absolute_value_array(current0, absCurrents0_copy_dynamic, 3);
		cout << "lets display the ascurrents\n";
```

```
                //display_nx3matrix(&absCurrents_copy_dynamic, lenB_cut);
                cout << "lets end display the ascurrents\n";

                //char signs0_copy_dynamic[MAX] = "";
                //char signs_copy_dynamic[lenB_cut][MAX];


                get_matrix_signs(current, signs_copy_dynamic, lenB_cut);
                get_array_signs(current0, signs0_copy_dynamic, 3);








                myPowerSupply_copy_dynamic = open_powerSupply_serial();

                myArduino_copy_dynamic = SerialInit("/dev/ttyACM0",57600);
                sleep(5);



//////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
                ////////////////////////////////////////////////////////////////////////////////
ENABLE BK CHANNELS AND PREDEFINED VALUES//////////////

//////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

                int channels_init[MAX] = {1, 1, 1};
//      BKsetOutputEnables([1 1 1]); % Enable [CH1 CH2 CH3]
                double voltages_init[MAX] = {10, 10, 5};
//      BKSetVoltages([10 10 5]);
                double currents_init[MAX] = {1, 1, 1};
//      BKSetCurrents([1 1 1]);% Amperes. Set current of each coil

                int controlOutput = 0;
                int controlVoltages = 1;
                int controlCurrents = 2;

                int channels[10] ;
                int voltages[10] ;
                int currents[10] ;

                channels[0] = channels_init[0];
                channels[1] = channels_init[1];
                channels[2] = channels_init[2];
                voltages[0] = voltages_init[0];
                voltages[1] = voltages_init[1];
                voltages[2] = voltages_init[2];
                currents[0] = currents_init[0];
                currents[1] = currents_init[1];
                currents[2] = currents_init[2];
                char array[10];

                sprintf(array, "%f", 3.123);


                //INIT POWER SUPPLY
                set_string2serialPowerSupply(channels, &myPowerSupply_copy_dynamic,
controlOutput);
                set_string2serialPowerSupply(voltages, &myPowerSupply_copy_dynamic,
controlVoltages);
                set_string2serialPowerSupply(currents, &myPowerSupply_copy_dynamic,
controlCurrents);
```

```
                string startStatus;
                cout << "power supply initialized!" << endl;
                cout <<"System start: Do you want to start? enter:Y/N" << endl;
                cin >> startStatus;
                sleep(10);


                };


void init_dynamic_variables_step2(){

                char sign_old_copy_dynamic[MAX] = "";
                strcat(sign_old_copy_dynamic, signs0_copy_dynamic);
                int controlTypeArduino = 0;
                set_string_channels2serialArduino(signs0_copy_dynamic, &myArduino_copy_dynamic ,
controlTypeArduino);//%% SET Channels




                cout << "lets display signs \n";
                sleep(2);
                int lenB_cut = lenB_cut_copy;
                display_nx3matrix(signs_copy_dynamic, lenB_cut);




                double current_meas[lenB_cut] = {0};
          char current_sign[3] = "";
                for(int i = 0; i < lenB_cut; i++){
                clock_t t;

                t = clock();

                double start_time; // clock
                        if (strcmp(signs_copy_dynamic[i], sign_old_copy_dynamic) != 0){

                    get_row_matrix_nx3(i, signs_copy_dynamic, current_sign);
                    cout << "signs in loop!!!!!!!!!!!!!!!!!!!!"<<endl;
                    display_array(current_sign, 3);

                    set_string_channels2serialArduino(current_sign, &myArduino_copy_dynamic ,
controlTypeArduino);
                    strcpy(sign_old_copy_dynamic,current_sign);
                    cout << "signs_old in loop!!!!!!!!!!!!!!!!"<<endl;
                    display_array(sign_old_copy_dynamic, 3);
                                //arduinoRelay(signs(i,:));
                                //sign_old = signs(i,:);
                                cout <<"signs changed\n";


                        }


                        //% Send current to power supply (only if voltage has changed)
                                //current_now = BKgetMeasuredCurrent();
                        int getControlType = 0;
                        double received_Currents_PowerSupply[3];
                  get_string2serialPowerSupply(&myPowerSupply_copy_dynamic, getControlType,
received_Currents_PowerSupply);

                        if (absCurrents_copy_dynamic[i][0]!=received_Currents_PowerSupply[0] &
absCurrents_copy_dynamic[i][1]!=received_Currents_PowerSupply[1] & absCurrents_copy_dynamic[i]
[2]!=received_Currents_PowerSupply[2]){
                                cout << "current changed\n";
                    double currents2change[3];
                    currents2change[0] = absCurrents_copy_dynamic[i][0];
                    currents2change[1] = absCurrents_copy_dynamic[i][1];
                    currents2change[2] = absCurrents_copy_dynamic[i][2];
                    int controlTypeArduino = 2;
                        set_string2serialPowerSupply(currents2change, &myPowerSupply_copy_dynamic,
controlCurrents_copy_dynamic);
                    cout << "current changed\n";

                        }

                        /*
                        % Save measured current
                                currents_meas(i,:) = BKgetMeasuredCurrent();

                        */
                        t = clock() - t;


                        cout <<"paused for:"<< dt_copy_dynamic-
float(t)/CLOCKS_PER_SEC<<"seconds\n";
                        sleep(dt_copy_dynamic - float(t)/CLOCKS_PER_SEC);
                }




};

void dynamic_orbit_simulator(){
double dt;
                dt = 0.25;
                /*
                %% Impedances of coils, camp, mag earth offset
                        % ALL THE VALUES IN THIS SECTION COULD BE CHANGED DUE TO CALIBRATION OF
THE
                        % COILS.
                        % Not implemented: file to load calibration values

                        % Impedances
                */
                //Impedances//
                double R_Bx = 12.6*2; // Ohms (Connected on serie)
                double R_By = 11.6*2; // Ohms (Connected on serie)
                double R_Bz = 10.6;   // Ohms (Connected on parallel)
                double R_B[3] = {R_Bx, R_By, R_Bz};
                // MagField/Intensity Relation //
                double x_camp = 200e-6; // Teslas/Ampers
                double y_camp = 200e-6; // Teslas/Ampers
                double z_camp = 200e-6; // Teslas/Ampers
                double camp[3] = {x_camp, y_camp, z_camp};
                //Offset magnetic field//
                //      double x_maglecture = 0;
                //      double y_maglecture = 0;
                //      double z_maglecture = 0;
                double x_magEarth = 17.538e-6; //%Voltages
                double y_magEarth =-19.05e-6;
                double z_magEarth = -35.53e-6;
                double magEarth[3] = {x_magEarth, y_magEarth, z_magEarth};







                File file;
                char fileName[12] = "";
                strcpy(fileName, "myData.txt");
                file.readFile(fileName);
```

```
		double tini = 1377;
		double Nini = round(tini/dt);


		std::vector< double > Bxx;
		std::vector< double > Byy;
		std::vector< double > Bzz;
		Bxx = file.Bxx;
		Byy = file.Byy;
		Bzz = file.Bzz;

		int arrSize = Bzz.size();
		cout <<"size"<<arrSize<<endl;
		display_array(Bzz,arrSize);




		//%% Canvis Edgard

		double unix_time=1584662400.0;  //% Be carefull_ Depending on orbit: % unix = 
1584662400 Orbits: M6P and Equ
		//unix=1583887278;  % Orbit: R2
		double time;
		std::vector< double > Bx;
		std::vector< double > By;
		std::vector< double > Bz;
		std::vector< double > TIME;





		for (int i = 0; i < 22746; i++){//22746
			if (i==0){
				time = unix_time;
			}
			double vector[3][1] = {{Bxx[i]},{Byy[i]},{Bzz[i]}};

			double JD = 0.0;
			unix2julian_conversion(time, JD );


//			if (i==0){
//			cout<<"JD\n";
//			printf("%.6f", JD);
//			}
			double C[3][3];
			//double C[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

			ceci2ecef(JD, C);


			if (i==0){
			display_3x3matrix(C);
			}
			double C_trans[3][3];
			int rows_C_trans = 3;
			int columns_C_trans = 3;
			transpose_matrix(C, C_trans, rows_C_trans, columns_C_trans);
//			if (i==22745){
//				cout<< JD << "JD\n";
//				cout<<"C_trans\n";
//				display_3x3matrix(C_trans);
//			}
```

```
                        double BB[3][1];
                        multiply_matrices(C_trans, vector, 3, 3, 3, 1, BB);

//

                        if (i>=0){
//                              cout <<"BB is :\n";
//                              cout << BB[0][0]<<" "<<BB[1][0]<<" "<<BB[2][0]<< endl;
//                              cout <<"vector is :\n";
//                              cout << vector[0][0]<<" "<<vector[1][0]<<" "<<vector[2][0]<< endl;
                        }

//                      double BB[3];
//                      BB =
//
                        Bx.push_back(BB[0][0]);
                        By.push_back(BB[1][0]);
                        Bz.push_back(BB[2][0]);
                        TIME.push_back(time);
                        time = time + dt;

                }
//              int len_B  = Bx.size();
//              cout << "display B please\n";
//              display_array(Bx, len_B);
//

                init_dynamic_variables(Bx,By,Bz,magEarth,  camp);



                }
```

```
#include <vector>
#include <string>
#include <cstring>

#ifndef START_Hd
#define START

/////////////////////////////////////////////////////////////////////
///////////////////////////////////LIBRARIES FOR THE GLOBAL CODE//////
///////////////////////////////////////////////////////////////////////




#include<iostream>




#include <string>
#include <math.h>       /* pow */ /* floor */
#include <stdio.h>
#include <time.h>       /* clock_t, clock, CLOCKS_PER_SEC */


#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>

/////////////////////////////////////////////////////////////////////
///////////////////////////////////LIBRARIES FOR THE SERIAL CODE//////
///////////////////////////////////////////////////////////////////////

// Linux headers
#include <fcntl.h> // Contains file controls like O_RDWR
#include <errno.h> // Error integer and strerror() function
#include <termios.h> // Contains POSIX terminal control definitions
#include <unistd.h> // write(), read(), close()


using namespace std;

///////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////init all the functions///////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////Static & Dynamic Orbit Simulator/////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////
///////////////////////////DECLARE GLOBAL VARIABLES///////
////////////////////////////////////////////////////////////
```

```
#define MAX 3

///////////////////////////
////////////////////////////
/////Processing main/////////
//////////////////////////////


class start
{
private:

public:
        //HANDLE *myArduino;
        char signs0[10] = "";
        char signs[10] = "";
        double absCurrents0[3];
        double absCurrents[3];
        int myint;
        int runStatus;
        void get_run_status();
        /*
        void static_orbit_simulator_step1(double Bxx,double Byy, double Bzz, double Bxx_mag,
double Byy_mag, double Bzz_mag);
        void static_orbit_simulator_step2();
        */

        void start_dynamic_orbit_simulator();
        void start_static_orbit_simulator(double Bxx,double Byy, double Bzz, double Bxx_mag,
double Byy_mag, double Bzz_mag);
        void start_static_orbit_simulator_step1(double Bxx,double Byy, double Bzz, double Bxx_mag,
double Byy_mag, double Bzz_mag);
        void start_static_orbit_simulator_step2();
        void start_static_orbit_simulator_step3();
        void start_static_orbit_simulator_step4();


        void start_dynamic_orbit_simulator_step1();
        void start_dynamic_orbit_simulator_step2();


        start();
};






#endif // START_H
```

```
#include "File.h"
#include <iostream>
#include <fstream>
#include <stdexcept>

#include <vector>
#include <cstring>
#include <string.h>
#include <stdio.h>
#include <vector>
using namespace std;

/*void get_Bxx_Byy_Bzz_fromLine(string line, double Bxx, double Byy, double Bzz);*/

void get_Bxx_Byy_Bzz_fromLine(string line, double &Bxx, double &Byy, double &Bzz);


File::File()
{
        cout << "lerts'goooo" << endl;

}

void File::printSomething()
{
        cout << "letsssss" << endl;

}
void File::openFile(char fileName[20]){

        ofstream myfile;
        //myfile.open ("example.txt");
        myfile.open (fileName);
//      myfile << "10838.6602;-1265.53358;21633.67\n";
//      myfile << "10838.6602;-1265.53358;21633.67\n";
//      myfile << "10838.6602;-1265.53358;21633.67\n";


        //myfile.close();
}
void File::readFile(char fileName[20]){

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        ////This function opens the file with format Bxx;Byy;Bzz and fill the atribute Bxx Byy Bzz
the values by pushing  so we fill the 3 vector of magnetic fields//////

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        string line;
        ifstream myfile;
        myfile.open(fileName);
        int count = 0;
        while ( getline (myfile,line) ){
        try{
                if(count != 0){
                cout << line << '\n';
                int n = line.length();
                char char_array[n + 1];
                strcpy(char_array, line.c_str());
                double Bxx;
                double Byy;
                double Bzz;
                get_Bxx_Byy_Bzz_fromLine( line,Bxx, Byy, Bzz);/////////////////////////////cout <<
Bxx <<"  "<< Byy<<"  " << Bzz <<"  this is BxxByyBzz\n";
                setBxx(Bxx);
                setByy(Byy);
                setBzz(Bzz);
                }
                count += 1;

        }
        catch(...){

        }

        }
        myfile.close();
}

void File::setBxx(double bxx){
        Bxx.push_back(bxx);
}
void File::setByy(double byy){
        Byy.push_back(byy);
}
void File::setBzz(double bzz){
        Bzz.push_back(bzz);
}




void get_Bxx_Byy_Bzz_fromLine(string line, double &Bxx, double &Byy, double &Bzz){

/////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        //////this function splits the line Bxx;Byy;Bzz and set the values to the
variables/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        int n = line.length();
        char char_array[n + 1];
        strcpy(char_array, line.c_str());
        double B[3];
        char *split;
        split = strtok(char_array,";");
        string word;
        int i = 0;
        while(split != NULL)
        {
                word=split;
                split=strtok(NULL,";");
                double word_double = std::stof(word);
                B[i] = (double)word_double;
                i += 1;
        }
        Bxx = B[0];
        Byy = B[1];
        Bzz = B[2];
        //cout << Bxx <<"  "<< Byy<<"  " << Bzz <<"  this is BxxByyBzz inside\n";
}
```

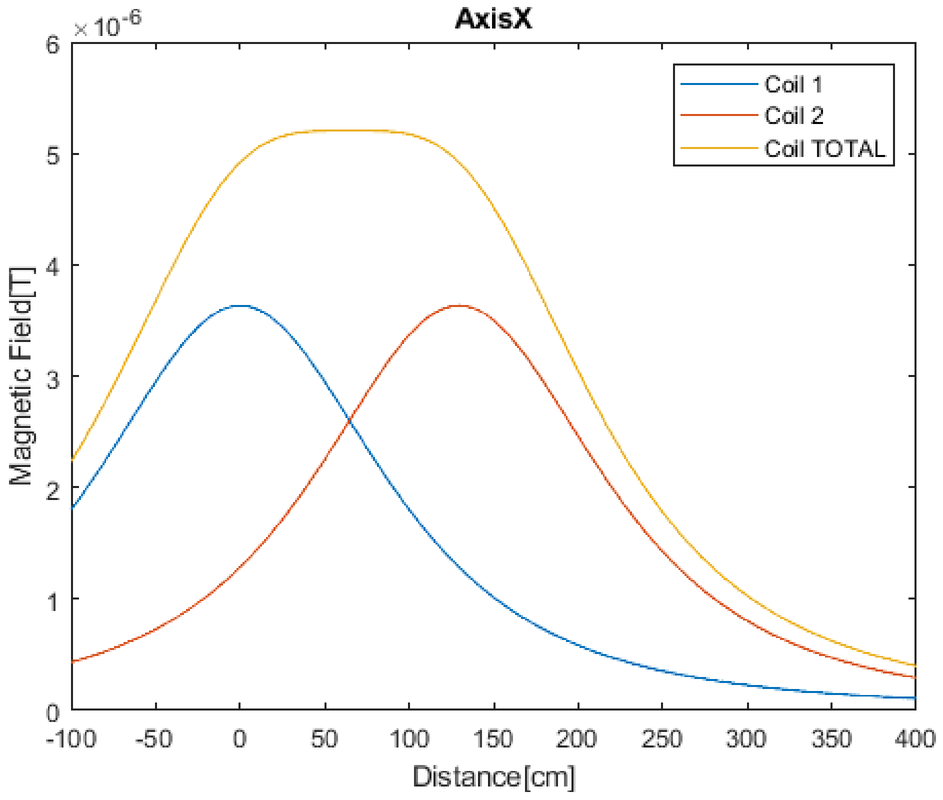
```
#include <vector>
#include <string>
#include <cstring>

#ifndef FILE_Hd
#define FILE_H



class File
{
private:

//      double Bxx[];
//      double Byy[];
//      double Bzz[];
public:

        std::vector< double > Bxx;
        std::vector< double > Byy;
        std::vector< double > Bzz;

        int myint;


//      std::vector< double > Bxx;
//      std::vector< double > Byy;
//      std::vector< double > Bzz;


//      vector<double> Bxx;
//      vector<double> Byy;
//      vector<double> Bzz;


        File();
        /*void get_Bxx_Byy_Bzz_fromLine(string line, double Bxx, double Byy, double Bzz);*/
        //void appendMagneticField(Bxx,Byy,Bzz);
        void openFile(char fileName[20]);
        void readFile(char fileName[20]);

        void setBxx(double bxx);
        void setByy(double byy);
        void setBzz(double bzz);
        void printSomething();
};






#endif // FILE_H
```

APPENDIX B. COILS STUDY

```
clear all;
close all;

diameters = [1.295 1.241 1.187];
N =  [144 138 132];
tau = 4*pi*10^-7;

B_measured =  [5 5 5;
              10 10 10;
              15 15 15;
              20 20 20;
              25 25 25;
              30 30 30;
              35 35 35;
              40 40 40;
              45 45 45;
              50 50 50;
              55 55 55;
              60 60 60;
              65 65 65;
              70 70 70;
              75 75 75;
    ];%nT
B_measured = B_measured*1e-6;


I =   [0.026 0.027 0.027;
       0.051 0.052 0.052;
       0.076 0.077 0.077;
       0.101 0.102 0.102;
       0.126 0.127 0.127;
       0.152 0.152 0.152;
       0.177 0.177 0.177;
       0.202 0.202 0.202;
       0.227 0.227 0.227;
       0.252 0.252 0.252;
       0.277 0.277 0.277;
       0.301 0.301 0.301;
       0.326 0.326 0.326;
       0.352 0.352 0.352;
       0.371 0.371 0.371;
    ];%A

R = diameters/2;
h = [0.617 0.590 0.563]+0.03;%IMPORTANT TO SUM THE STRUCTURE THICKNESS
z = h(1)/2;


Bz1 = zeros(3, 2500);
Bz2 = zeros(3, 2500);
```

```
for axis=1:3
    count = 1;
    for j = -500:2000
        i = j/1000;
        Bz1(axis,count) = (tau*N(axis)*I(axis)*R(axis)^2)/
(2*(R(axis)^2+i^2)^(3/2));
        Bz2(axis,count) = (tau*N(axis)*I(axis)*R(axis)^2)/
(2*(R(axis)^2+(h(axis)-i)^2)^(3/2));
        count = count + 1;

    end
end
Bztot = Bz1 + Bz2;

x = [-500:1:2000];
x = 2* (x/10);
axisVect = ["AxisX", "AxisY", "AxisZ"];
for coil=1:3

    figure(coil+4)
    plot(x, Bz1(coil, :))
    xlabel("Distance[cm]")
    ylabel("Magnetic Field[T]")
    hold on
    plot(x,Bz2(coil, :))
    xlabel("Distance[cm]")
    ylabel("Magnetic Field[T]")
    hold on
    plot(x,Bztot(coil, :))

    legend('Coil 1','Coil 2','Coil TOTAL')
    xlabel("Distance[cm]")
    ylabel("Magnetic Field[T]")
    title(string(axisVect(coil)))
end

B_theory = [];
[I_rows,I_columns] = size(I);
Bz_errorRelativo = zeros(I_rows,I_columns);
Bz_errorAbsoluto = zeros(1,I_columns);

for i=1:I_rows
    for j=1:I_columns
        B_theory(i,j) = (8*tau*N(j)*I(i,j))/(R(j)*125^0.5);
        Bz_errorRelativo(i,j) = ((B_theory(i,j) - B_measured(i,j))/
 B_theory(i,j)) * 100;
    end
end

figure(1)
for i=1:3
    plot(I(:,i), B_measured(:,i))
    hold on
end
```

```
figure(2)
for i=1:3
    plot(I(:,i), B_theory(:,i))
xlabel('Current: Ampers')
ylabel('Bmeasured nT')


    hold on
end


figure(3)
for i=1:3
    plot( I(:,i), Bz_errorRelativo(:,i))
    hold on
end

xlabel('Current: Ampers')
ylabel('Relative Error %')
legend('X-Axis','Y-Axis','Z-Axis')

for i=1:3
    Bz_errorAbsoluto(i) = sum(Bz_errorRelativo(:,i))/(I_rows);
end
```



APPENDIX C. ANGULAR TRACKING ALGORITHM CODE

```python
import readANDwrite
import readANDwrite as readWrite
import Dynamic_Plot as dynamPlot
from numpy.fft import fft, ifft

import scipy as sp
import scipy.interpolate
import matplotlib.pyplot as plt

import sys
import math
# Import essential libraries
import requests
import cv2
import numpy as np
import imutils
#import urllib
import ssl
import time
import math

import circle_fit
import joblib
import constant
from scipy import signal


def circleROI():
    pts = []  # □□□□□

    # □□□□mouse callback function
    def draw_roi(event, x, y, flags, param):
        img2 = img.copy()

        if event == cv2.EVENT_LBUTTONDOWN:  # □□□□□□□□
            pts.append((x, y))

        if event == cv2.EVENT_RBUTTONDOWN:  # □□□□□□□□□□□□□□□
            pts.pop()

        if event == cv2.EVENT_MBUTTONDOWN:  # □□□□□□
            mask = np.zeros(img.shape, np.uint8)
            points = np.array(pts, np.int32)
            points = points.reshape((-1, 1, 2))
            # □□□□
            mask = cv2.polylines(mask, [points], True, (255, 255, 255), 2)
            mask2 = cv2.fillPoly(mask.copy(), [points], (255, 255, 255))  # □□□ ROI
            mask3 = cv2.fillPoly(mask.copy(), [points], (0, 255, 0))  # □□ □□□□□□□□

            show_image = cv2.addWeighted(src1=img, alpha=0.8, src2=mask3, beta=0.2, gamma
=0)

            cv2.imshow("mask", mask2)
            cv2.imshow("show_img", show_image)

            ROI = cv2.bitwise_and(mask2, img)
            cv2.imshow("ROI", ROI)
            cv2.waitKey(0)

        if len(pts) > 0:
            # □pts□□□□□□□□□
            cv2.circle(img2, pts[-1], 3, (0, 0, 255), -1)

        if len(pts) > 1:
            # □□
            for i in range(len(pts) - 1):
                cv2.circle(img2, pts[i], 5, (0, 0, 255), -1)  # x ,y □□□□□□□□□□
                cv2.line(img=img2, pt1=pts[i], pt2=pts[i + 1], color=(255, 0, 0), thickness=2)

        cv2.imshow('image', img2)

    # □□□□□□□□□□□□□□□□□□
    img = cv2.imread("./test_image.jpg")
    img = imutils.resize(img, width=500)
    cv2.namedWindow('image')
    cv2.setMouseCallback('image', draw_roi)
    print("[INFO] □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ROI□□")
    print("[INFO] □‘S’□□□□□□□□□")
    print("[INFO] □ ESC □□")

    while True:
        key = cv2.waitKey(1) & 0xFF
        if key == 27:
            break
        if key == ord("s"):
            saved_data = {
                "ROI": pts
            }
            joblib.dump(value=saved_data, filename="config.pkl")
            print("[INFO] ROI□□□□□□□□.")
            break
    cv2.destroyAllWindows()

def guessCircle(dataPoints,fitType):
    # - xc: x - coordinate ofsolution center(float)
    # - yc: y - coordinateo f solution center(float)
    # - R: Radius of solution(float)
    # - variance or residual(float)
    #dataPoints (n, 2)

    if fitType =="hyper_fit":

        (ccx, ccy, R, variance) = circle_fit.hyper_fit(dataPoints)

    elif(fitType =="least_squares_circle"):

        (ccx, ccy, R, variance) = circle_fit.least_squares_circle(dataPoints)
    else:

        ccx = 0
        ccy = 0
        R = 0
        variance = 0

    #circle_fit.plot_data_circle(ccx,ccy,R)
    #plt.plot(plot2show)
    #plt.show()
    print("plot",(dataPoints[0],dataPoints[1],ccx,ccy,R))

    return(ccx,ccy,R,variance)

def setBboxDimensions(bbox_changed,bbox_reference):
    #this function returns a new bbox bbox= (p1,p2,ax,ay)
    # with the center of the bbox_changed and with the lengths of the bbox_reference
    [cx, cy] = edge2centerSquare(bbox_changed)

    [ax, ay] = [bbox_reference[2],bbox_reference[3]]
    new_bbox = (cx, cy, ax, ay)
    new_bbox = center2edgeSquare(new_bbox)
    return new_bbox

def edge2centerSquare(bbox):
    ################################################################
```

```python
    # bbox has the following format bbox= (p1,p2,ax,ay)
    # where (p1,p2) are the coordinates of the left superior edge
    # where ax is the size of the horizontal side
    # where ay i sthe size of the vertical side
    #################################################################
    cx = bbox[0] + np.floor(bbox[2]/2)
    cy = bbox[1] + np.floor(bbox[3]/2)
    return cx, cy

def center2edgeSquare(bbox):
    [cx, cy] = [bbox[0],bbox[1]]
    [ax, ay] = [bbox[2], bbox[3]]
    new_bbox = (cx-ax/2,cy - ay/2, ax,ay)
    return new_bbox

def getCircleParameters(trackingType, tracker_type):
    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')


    # Set up tracker.
    # Instead of MIL, you can also use


    #tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
    #tracker_type = tracker_types[1]

    if int(minor_ver) < 3:
        tracker = cv2.Tracker_create(tracker_type)
    else:
        if tracker_type == 'BOOSTING':
            tracker = cv2.TrackerBoosting_create()
        if tracker_type == 'MIL':
            tracker = cv2.TrackerMIL_create()
        if tracker_type == 'KCF':
            tracker = cv2.TrackerKCF_create()
        if tracker_type == 'TLD':
            tracker = cv2.TrackerTLD_create()
        if tracker_type == 'MEDIANFLOW':
            tracker = cv2.TrackerMedianFlow_create()
        if tracker_type == 'GOTURN':
            tracker = cv2.TrackerGOTURN_create()
        if tracker_type == 'MOSSE':
            tracker = cv2.TrackerMOSSE_create()
        if tracker_type == "CSRT":
            tracker = cv2.TrackerCSRT_create()

    # Read video
    if(trackingType == "CAPTURE"):
        video = cv2.VideoCapture("C:/Users/User/Documents/Ulises/UPC/TFG/CODE/AngularSpeed/Videos/RotationEdited.mp4")
    elif(trackingType == "CAMERA"):
        channel = 0
        video = cv2.VideoCapture(channel)
    # Exit if video not opened.
    if not video.isOpened():
        print
        "Could not open video"
        sys.exit()

    # Read first frame.
    ok, frame = video.read()
    reference_frame = frame
    #frame = cv2.resize(frame, (540, 380), fx=0, fy=0)
    #frame = cv2.Canny(frame, 100, 200)
    if not ok:
        print
        'Cannot read video file'
        sys.exit()

    # Define an initial bounding box
    #bbox = (287, 23, 86, 320)

    # Uncomment the line below to select a different bounding box
    bbox = cv2.selectROI(frame, False)

     #Initialize tracker with first frame and bounding box
    ok = tracker.init(frame, bbox)

    pointsx = []
    pointsy = []

    # used to record the time when we processed last frame
    prev_frame_time = 0
    prev_frame_angle = 0

    # used to record the time at which we processed current frame
    new_frame_time = 0
    new_frame_angle = 0
    while True:
        c1 = cv2.getTickCount()
        # Read a new frame
        ok, frame = video.read()
        #frame = cv2.Canny(frame, 200, 300)
        if not ok:
            break
        # Start timer
        timer = cv2.getTickCount()
        # Update tracker
        ok, bbox = tracker.update(frame)
        # Calculate Frames per second (FPS)
        fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
        (box_x, box_y) = edge2centerSquare(bbox)
        pointsx.append(box_x)
        pointsy.append(box_y)

        # Draw bounding box
        if ok:
            # Tracking success
            p1 = (int(bbox[0]), int(bbox[1]))
            p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
            cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
        else:
            # Tracking failure
            cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.
FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

        # time when we finish processing for this frame
        new_frame_time = time.time()
        # Calculating the fps

        # fps will be number of frame processed in given time frame
        # since their will be most of time error of 0.001 second
        # we will be subtracting it to get more accurate result

        consecutive_frame_time = new_frame_time - prev_frame_time
        fps = 1 / consecutive_frame_time
        prev_frame_time = new_frame_time

        # converting the fps into integer
        fps = int(fps)

        # converting the fps to string so that we can display it on frame
        # by using putText function
        fps = str(fps)
        # putting the FPS count on the frame
        cv2.putText(frame, fps, (7, 70), cv2.FONT_HERSHEY_SIMPLEX, 3, (100, 255, 0), 3, cv2.LINE_AA)

        c2 = cv2.getTickCount()
        time_taken = (c2 - c1) / cv2.getTickFrequency()

        # Display tracker type on frame
        cv2.putText(frame, tracker_type + " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

        # Display FPS on frame
        cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

        # Display result
        cv2.imshow("Tracking", frame)

        # Exit if ESC pressed
        k = cv2.waitKey(1) & 0xff
        if k == 27: break

    xy = np.zeros((2))
    xyT = transposeVectorsANDmerge(pointsx, pointsy)
    fitType = "hyper_fit"
    (ccx, ccy, R, variance) = guessCircle(xyT, fitType)
    print("fit results", ccx, ccy, R, variance)
    plotDataANDFittedData(pointsx , pointsy, ccx, ccy, R,reference_frame)
    return ccx, ccy, R

def getAngleWithPoints(p1,p2):
    #This fcntion return the angle that forms the 2 point with respect of the horizontal line (the slope angle)
    beta = math.atan2((p1[1]-p2[1]), (p1[0]-p2[0]))
    return beta

def getAngleBetween2vectorsAntiClockWise(vector_2,vector_1):
    x1 = vector_1[0]
    y1 = vector_1[1]
    x2 = vector_2[0]
    y2 = vector_2[1]
    dot = x1 * x2 + y1 * y2  # dot product between [x1, y1] and [x2, y2]
    det = x1 * y2 - y1 * x2  # determinant
    angle = math.atan2(det, dot)  # atan2(y, x) or atan2(sin, cos)
    if angle < 0:
        angle = angle + 2*math.pi
    return 2*math.pi - angle

def getAngleBetween2vectorsClockWise(vector_2,vector_1):
    x1 = vector_1[0]
    y1 = vector_1[1]
    x2 = vector_2[0]
    y2 = vector_2[1]
    dot = x1 * x2 + y1 * y2  # dot product between [x1, y1] and [x2, y2]
    det = x1 * y2 - y1 * x2  # determinant
    angle = math.atan2(det, dot)  # atan2(y, x) or atan2(sin, cos)
    if angle < 0:
        angle = angle + 2*math.pi
    return angle

def GetAngleBetweenPoints( endPt2,  connectingPt, endPt1 ):
    x1 = endPt1[0] - connectingPt[0]; #Vector 1 - x
    y1 = -endPt1[1] - connectingPt[1]; #Vector 1 - y

    x2 = endPt2[0] - connectingPt[0]; #Vector 2 - x
    y2 = endPt2[1] - connectingPt[1]; #Vector 2 - y

    angle = math.atan2(y1, x1) - math.atan2(y2, x2);
    #angle = angle * 360 / (2*math.pi);
    #
    # if angle < 0:]]]
    #     angle += 360
    if angle < 0:
        angle += 2*math.pi


    return angle

def getAngleBetween2vectors(vector_1, vector_2):

    #this function returns the angle formed between 2 vectors in rad!!!!
    # Vector_1 & vector_2 accept the following format, for instance vector_1 = [2,1]

    try:
        unit_vector_1 = vector_1 / np.linalg.norm(vector_1)
        unit_vector_2 = vector_2 / np.linalg.norm(vector_2)
        dot_product = np.dot(unit_vector_1, unit_vector_2)
        angle = np.arccos(dot_product)

    except:
        print(dot_product,angle)
        angle = 0

    return angle

def define2DVector(point1, point2):
    #point1 hs the following shape [p1,p2] and so for point2
    vector= np.zeros(2)
    vector[0] = point1[0] - point2[0]
    vector[1] = point1[1] - point2[1]
    return vector

def isSameTurn(phi,phi_old):
    #compare the phi closing to the lap point. so if phi_old=358 and phi = 2 compares the values with a threshold set to aprox to 300

    phi_difference = abs(phi - phi_old)
    diff_threshold = 300
    if phi_difference > diff_threshold:
        return False
    else:
        return True

def trackCameraAngularSpeed( polygon_vertex, ccx, ccy, R,tracker_type):
    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
    circle_center = [ccx, ccy]

    #txt2save = readWrite.Text("Data1.txt")
    times = []
    angularSpeeds = []

    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
    if __name__ == '__main__':


        # Set up tracker.
        # Instead of MIL, you can also use

        #tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
        #tracker_type = tracker_types[7]

        if int(minor_ver) < 3:
```

```python
            tracker = cv2.Tracker_create(tracker_type)
        else:
            if tracker_type == 'BOOSTING':
                # tracker = cv2.TrackerBoosting_create()
                tracker = cv2.legacy.TrackerBoosting_create()
            if tracker_type == 'MIL':
                tracker = cv2.TrackerMIL_create()
            if tracker_type == 'KCF':
                tracker = cv2.TrackerKCF_create()
            if tracker_type == 'TLD':
                # tracker = cv2.TrackerTLD_create()
                tracker = cv2.legacy.TrackerTLD_create()
            if tracker_type == 'MEDIANFLOW':
                # tracker = cv2.TrackerMedianFlow_create()
                tracker = cv2.legacy.TrackerMedianFlow_create()
            if tracker_type == 'GOTURN':
                # tracker = cv2.TrackerGOTURN_create()
                # tracker = cv2.legacy_TrackerGOTURN_create()
                # tracker = cv2.TrackerGOTURN_create()
                tracker = cv2.TrackerGOTURN.create()
            if tracker_type == 'MOSSE':
                # tracker = cv2.TrackerMOSSE_create()
                tracker = cv2.legacy.TrackerMOSSQE_create()
            if tracker_type == "CSRT":
                tracker = cv2.TrackerCSRT_create()

        # Read video
        channel = 0
        video = cv2.VideoCapture(channel)

        # Exit if video not opened.
        if not video.isOpened():
            print
            "Could not open video"
            sys.exit()

        # Read first frame.
        ok, frame = video.read()
        #frame = cv2.resize(frame, (540, 380), fx=0, fy=0)
        starttime = time.time()
        lasttime = starttime
        laptime = 0
        totaltime = 0
        frames_received = 0

        # frame = cv2.Canny(frame, 100, 200)
        if not ok:
            print
            'Cannot read video file'
            sys.exit()

        # Define an initial bounding box
        # bbox = (287, 23, 86, 320)

        # Uncomment the line below to select a different bounding box
        bbox = cv2.selectROI(frame, False)
        bbox_reference = bbox

        ########### Define the point in the circle to set the start/finish line.
        (reference_x, reference_y) = edge2centerSquare(bbox)
        reference_point = (reference_x, reference_y)

        numTurns = 0

        # Initialize tracker with first frame and bounding box
        ok = tracker.init(frame, bbox)

        # used to record the time when we processed last frame
        prev_frame_time = 0
        prev_frame_angle = 0

        # used to record the time at which we processed current frame
        new_frame_time = 0
        new_frame_angle = 0

        # isGrowing = false
        # isShrinking = false
        aggregated_frames_time = 0
        turn_min_time = 0
        thresholdTime = 1/3 #seconds
        init_section_threshold_time = 0.0
        INITIAL_TIME = 0
        final_section_threshold_time = 0.01
        time2turn = 0
        aggregated_phi = 0

        phi_threshold = (2*math.pi)/polygon_vertex #rad ##### input
        phi_threshold = (360) / polygon_vertex  # deg ##### input
        init_phi_threshold =phi_threshold
        section_angular_speed = 0
        consecutive_frame_time=0.01
        bbox_center_previousFrame = reference_point
        accumulated_phi = 0
        starting_time = 0
        prev_phi = 0
        d = dynamPlot.DynamicUpdate()
        Dynamicplot_x_max = d.on_launch()
        isClockWise = True
        isStopped = True


        while True:
            #time2turn = time.time()
            c1 = cv2.getTickCount()
            # Read a new frame
            ok, frame = video.read()
            # frame = cv2.Canny(frame, 200, 300)
            if not ok:
                break
            # Start timer
            timer = cv2.getTickCount()
            # Update tracker
            ok, bbox = tracker.update(frame)
            bbox = setBboxDimensions(bbox, bbox_reference)
            # Calculate Frames per second (FPS)
            fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)


            (box_x, box_y) = edge2centerSquare(bbox)
            bbox_center = [box_x, box_y]
            center2boxx_vector_previousFrame = define2DVector(circle_center,
bbox_center_previousFrame)
            center2boxx_vector = define2DVector(circle_center, bbox_center)
            center2reference_vector = define2DVector(circle_center, reference_point)

            if(isClockWise):
                phi = getAngleBetween2vectorsClockWise(center2boxx_vector,
center2reference_vector)

            else:
                phi = getAngleBetween2vectorsAntiClockWise(center2boxx_vector,
center2reference_vector)


```

```
            phi = phi * (180/math.pi)
            #phi = GetAngleBetweenPoints(bbox_center, reference_point, circle_center)
            #accumulated_phi = phi
            #phi = getAngleBetween2vectors(center2boxx_vector, center2boxx_vector_previousFrame)
            circle_center = [ccx, ccy]
            box_center = [box_x, box_y]
            #new_frame_angle = getAngleWithPoints(circle_center, box_center)
            #print("phi",phi*(180/math.pi))
            new_frame_angle = phi
            # Draw bounding box
            if ok:
                # Tracking success
                p1 = (int(bbox[0]), int(bbox[1]))
                p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
                cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
            else:
                # Tracking failure
                cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255),
                            2)

            # time when we finish processing for this frame
            new_frame_time = time.time()
            if frames_received == 0:
                INITIAL_TIME = new_frame_time
            #new_frame_time = new_frame_time - INITIAL_TIME



            starting_time = new_frame_time
            turn_min_time = turn_min_time + new_frame_time

            # Calculating the fps
            # fps will be number of frame processed in given time frame
            # since their will be most of time error of 0.001 second
            # we will be subtracting it to get more accurate result

            consecutive_frame_time = new_frame_time - prev_frame_time
            consecutive_frame_beta = abs(new_frame_angle - prev_frame_angle)
            aggregated_frames_time = aggregated_frames_time + new_frame_time

            instant_angular_speed = (consecutive_frame_beta / consecutive_frame_time) * (
                        1 / (360))  # revolution per second



            ################
            #phi in DEGRESS
            ################

            phi_derivative = abs(phi - prev_phi)
            if (phi_derivative > 1 and phi_derivative < 359):  # is stopped condition
                isStopped = False

            if (phi_derivative <= 0):
                isClockWise = False

                print("change to isAntiClockWise", phi)
            else:
                isClockWise = True

                print("change to isClockWise", phi)


            #
```

```python
            #
            # if hasChangedDirection():
            #     if isClockWise:
            #         phi_threshold = phi_threshold + init_phi_threshold
            #     else:
            #         phi_threshold = phi_threshold - init_phi_threshold





            is_same_turn = isSameTurn(phi, prev_phi)
            if isSameTurn(phi, prev_phi) == False:
                numTurns = numTurns + 1
                if phi_threshold == float(360):
                    print("change and threshold 360")
                    phi_threshold = init_phi_threshold

            if phi_threshold > 360:
                phi_threshold = init_phi_threshold


            if (not isStopped): #is stopped condition
                if (phi >= phi_threshold and phi<358 ) or is_same_turn == False:
                    #WARNING THE COMPARATIVE MUST BE IN the same units!!!
                    #try:
                    laptime = round((time.time() - lasttime), 2)
                    #totaltime = round((time.time() - starttime), 2)
                    lasttime = time.time()
                    section_angular_speed = init_phi_threshold / (laptime)
                    init_section_threshold_time =  INITIAL_TIME - consecutive_frame_time
                    if is_same_turn == True:
                        phi_threshold = phi_threshold + init_phi_threshold

                else:
                    init_section_threshold_time = float(init_section_threshold_time) + float(consecutive_frame_time)



            totaltime = round((time.time() - starttime), 2)


            times.append(totaltime)
            angularSpeeds.append(section_angular_speed)
            print("tottime",totaltime,"dynamic_xmax",Dynamicplot_x_max)
            d.on_running(times, angularSpeeds)
            if(totaltime > Dynamicplot_x_max):
                Dynamicplot_x_max = d.window_shifting()




            #print("laptime", laptime, "tottime", totaltime, "phitheres", phi_threshold , "phi", phi,"phi derivative",phi_derivative)


            fps = 1 / consecutive_frame_time
            bbox_center_previousFrame = box_center
            prev_frame_time = new_frame_time
            prev_frame_angle = new_frame_angle
            prev_phi = phi
            # converting the fps into integer
            fps = int(fps)

            # converting the fps to string so that we can display it on frame
            # by using putText function
            fps = str(fps)
            # putting the FPS count on the frame
            cv2.putText(frame, fps, (7, 70), cv2.FONT_HERSHEY_SIMPLEX, 3, (100, 255, 0
), 3, cv2.LINE_AA)


            c2 = cv2.getTickCount()
            time_taken = (c2 - c1) / cv2.getTickFrequency()

            # Display tracker type on frame
            cv2.putText(frame, tracker_type + " Tracker", (100, 20), cv2.
FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

            # Display FPS on frame
            # cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.
FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

            cv2.putText(frame, "Revolutions/sec RPS : " + str(instant_angular_speed), (
100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 170, 50), 2);

            cv2.putText(frame, "AngularSpeed (deg/sec) : " + str(instant_angular_speed
 * 360), (100, 200),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "TargetAngularSpeed (deg/sec) : " + str(
section_angular_speed) , (100, 250),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);


            cv2.putText(frame, "Time to reach the target [s] : " + str(laptime),
                        (100, 300),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "phi(deg) : " + str(phi) ,
                        (100, 350),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "Next Phi Target" + str(phi_threshold),
                        (100, 380),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);

            cv2.putText(frame, "numTurns : " + str(numTurns),
                        (100, 400),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            # cv2.putText(frame, "Angle respect the reference (deg) : " + str(
new_frame_angle * (180 / math.pi)),
            #             (100, 350),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.75,
            #             (50, 200, 50), 2);
            # cv2.putText(frame, "Angle respect the reference (deg) : " + str(
consecutive_frame_beta * (180 / math.pi)), (100, 400),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.75,
            #             (50, 200, 50), 2);
            # Display result

            frames_received = frames_received+1
            cv2.imshow("Tracking", frame)
            # Exit if ESC pressed
            k = cv2.waitKey(1) & 0xff
            if k == 27: break

        return (times, angularSpeeds)
        #txt2save.write_file()
```

```python

def trackCameraAngularSpeedTIME( polygon_vertex, ccx, ccy, R,tracker_type):
    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
    circle_center = [ccx, ccy]

    #txt2save = readWrite.Text("Data1.txt")
    times = []
    angularSpeeds = []

    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
    if __name__ == '__main__':


        # Set up tracker.
        # Instead of MIL, you can also use

        #tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', '
MOSSE', 'CSRT']
        #tracker_type = tracker_types[7]

        if int(minor_ver) < 3:
            tracker = cv2.Tracker_create(tracker_type)
        else:
            if tracker_type == 'BOOSTING':
                #tracker = cv2.TrackerBoosting_create()
                tracker =  cv2.legacy.TrackerBoosting_create()
            if tracker_type == 'MIL':
                tracker = cv2.TrackerMIL_create()
            if tracker_type == 'KCF':
                tracker = cv2.TrackerKCF_create()
            if tracker_type == 'TLD':
                #tracker = cv2.TrackerTLD_create()
                tracker = cv2.legacy.TrackerTLD_create()
            if tracker_type == 'MEDIANFLOW':
                #tracker = cv2.TrackerMedianFlow_create()
                tracker = cv2.legacy.TrackerMedianFlow_create()
            if tracker_type == 'GOTURN':
                #tracker = cv2.TrackerGOTURN_create()
                #tracker = cv2.legacy_TrackerGOTURN_create()
                #tracker = cv2.TrackerGOTURN_create()
                tracker = cv2.TrackerGOTURN.create()

            if tracker_type == 'MOSSE':
                #tracker = cv2.TrackerMOSSE_create()
                tracker = cv2.legacy.TrackerMOSSE_create()
            if tracker_type == "CSRT":
                tracker = cv2.TrackerCSRT_create()

        # Read video
        channel = 0
        video = cv2.VideoCapture(channel)

        # Exit if video not opened.
        if not video.isOpened():
            print
            "Could not open video"
            sys.exit()

        # Read first frame.
        ok, frame = video.read()


        #frame = cv2.resize(frame, (540, 380), fx=0, fy=0)
        starttime = time.time()
        starttime2sample = time.time()
        lasttime = starttime
        laptime = 0
        totaltime = 0
        time2sample = 0
        frames_received = 0

        # frame = cv2.Canny(frame, 100, 200)
        if not ok:
            print
            'Cannot read video file'
            sys.exit()










        # Define an initial bounding box
        # bbox = (287, 23, 86, 320)

        # Uncomment the line below to select a different bounding box


        #reference_circle = detect_shapes(frame)
        #print("detected rad",2*reference_circle[0]-reference_circle[2],
    reference_circle[1]-reference_circle[2], 2*reference_circle[2], 2*reference_circle[2])
        #bbox = (int(np.floor(reference_circle[0]-reference_circle[2])),int( np.floor(
    reference_circle[1]-reference_circle[2])),int( np.floor(2*reference_circle[2])),int( np.
    floor(2*reference_circle[2])))


        #bbox_reference = bbox


        bbox = cv2.selectROI(frame, False)
        #print("heeeeeeeeeeeeeeeeeeeeeeereeeeeee",bbox)
        bbox_reference = bbox





        ########### Define the point in the circle to set the start/finish line.
        (reference_x, reference_y) = edge2centerSquare(bbox)
        reference_point = (reference_x, reference_y)

        numTurns = 0

        # Initialize tracker with first frame and bounding box
        ok = tracker.init(frame, bbox)

        # used to record the time when we processed last frame
        prev_frame_time = 0
        prev_frame_angle = 0

        # used to record the time at which we processed current frame
        new_frame_time = 0
        new_frame_angle = 0

        # isGrowing = false
        # isShrinking = false
        aggregated_frames_time = 0
        turn_min_time = 0
        thresholdTime = 1/3 #seconds
        init_section_threshold_time = 0.0
        INITIAL_TIME = 0
        final_section_threshold_time = 0.01
        time2turn = 0
        aggregated_phi = 0

        phi_threshold = (2*math.pi)/polygon_vertex #rad ##### input
        phi_threshold = (360) / polygon_vertex  # deg ##### input
        init_phi_threshold =phi_threshold
        section_angular_speed = 0
        consecutive_frame_time=0.01
        bbox_center_previousFrame = reference_point
        accumulated_phi = 0
        starting_time = 0
        prev_phi = 0
        phi_old_sampled = 0
        isStopped = True
        isClockwise = True
        discrete_sampling_time = 0.3
        agregatedTime = discrete_sampling_time

        d = dynamPlot.DynamicUpdate()
        Dynamicplot_x_max = d.on_launch()
        loop = 0
        while True:
            #time2turn = time.time()
            c1 = cv2.getTickCount()
            # Read a new frame
            ok, frame = video.read()
            # frame = cv2.Canny(frame, 200, 300)
            if not ok:
                break
            # Start timer
            timer = cv2.getTickCount()
            # Update tracker
            ok, bbox = tracker.update(frame)
            bbox = setBboxDimensions(bbox, bbox_reference)
            # Calculate Frames per second (FPS)
            fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)


            (box_x, box_y) = edge2centerSquare(bbox)
            bbox_center = [box_x, box_y]
            center2boxx_vector_previousFrame = define2DVector(circle_center,
    bbox_center_previousFrame)
            center2boxx_vector = define2DVector(circle_center, bbox_center)
            center2reference_vector = define2DVector(circle_center, reference_point)
            phi = getAngleBetween2vectorsClockWise(center2boxx_vector,
    center2reference_vector)
            phi = phi * (180/math.pi)
            #phi = GetAngleBetweenPoints(bbox_center, reference_point, circle_center)
            #accumulated_phi = phi
            #phi = getAngleBetween2vectors(center2boxx_vector,
    center2boxx_vector_previousFrame)
            circle_center = [ccx, ccy]
            box_center = [box_x, box_y]
            #new_frame_angle = getAngleWithPoints(circle_center, box_center)
            #print("phi",phi*(180/math.pi))
            new_frame_angle = phi
            # Draw bounding box
            if ok:
                # Tracking success
                p1 = (int(bbox[0]), int(bbox[1]))
                p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
                cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
            else:
                # Tracking failure
                cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.
FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255),
                            2)

            # time when we finish processing for this frame
            new_frame_time = time.time()
            if frames_received == 0:
                INITIAL_TIME = new_frame_time
            #new_frame_time = new_frame_time - INITIAL_TIME



            starting_time = new_frame_time
            turn_min_time = turn_min_time + new_frame_time

            # Calculating the fps
            # fps will be number of frame processed in given time frame
            # since their will be most of time error of 0.001 second
            # we will be subtracting it to get more accurate result

            consecutive_frame_time = new_frame_time - prev_frame_time
            consecutive_frame_beta = abs(new_frame_angle - prev_frame_angle)
            aggregated_frames_time = aggregated_frames_time + new_frame_time

            instant_angular_speed = (consecutive_frame_beta / consecutive_frame_time) * (
                        1 / (360))  # revolution per second




            phi_derivative = (phi - prev_phi)
            if (phi_derivative <= 0):
                isClockWise = False

                print("change to isAntiClockWise", phi)
            else:
                isClockWise = True
                print("change to isClockWise", phi)

            abs_phi_derivative = abs(phi_derivative)
            if (abs_phi_derivative > 1 and abs_phi_derivative < 359):  # is stopped condition
                isStopped = False



            ################
            #phi in DEGRESS
            ################
            is_same_turn = isSameTurn(phi, prev_phi)


            if (not isStopped):

                if is_same_turn:
                    if(time2sample >  agregatedTime):
                        #WARNING THE COMPARATIVE MUST BE IN the same units!!!
                        #try:
                        laptime = round((time.time() - lasttime), 2)
                        #totaltime = round((time.time() - starttime), 2)
                        lasttime = time.time()
                        section_angular_speed = (phi - phi_old_sampled) / (laptime)
                        agregatedTime = agregatedTime + discrete_sampling_time
                        phi_old_sampled = phi
                        print("agregated",agregatedTime,"totaltime2sample",time2sample,"phi", phi)
                else:
```

```python

                    if(isClockWise):
                        #phi =  phi - 360
                        phi_old_sampled = phi_old_sampled + 360
                    else:
                        #phi = phi+ (phi_old_sampled-360)
                        phi_old_sampled =  phi_old_sampled - 360
                        print(is_same_turn, "same turn")

            else:
                if (isClockWise):
                    phi_old_sampled = 360
                else:
                    phi_old_sampled = 0
                starttime2sample =time.time()

            totaltime = round((time.time() - starttime), 2)
            time2sample = round((time.time() - starttime2sample), 2)

            times.append(totaltime)
            angularSpeeds.append(section_angular_speed)


            d.on_running(times, angularSpeeds)
            if(totaltime > Dynamicplot_x_max):
                Dynamicplot_x_max = d.window_shifting()



            #print("laptime", laptime, "tottime", totaltime, "phitheres", phi_threshold, "phi", phi,"phi derivative",phi_derivative)


            fps = 1 / consecutive_frame_time
            bbox_center_previousFrame = box_center
            prev_frame_time = new_frame_time
            prev_frame_angle = new_frame_angle
            prev_phi = phi
            # converting the fps into integer
            fps = int(fps)

            # converting the fps to string so that we can display it on frame
            # by using putText function
            fps = str(fps)
            # putting the FPS count on the frame
            cv2.putText(frame, fps, (7, 70), cv2.FONT_HERSHEY_SIMPLEX, 3, (100, 255, 0), 3, cv2.LINE_AA)


            c2 = cv2.getTickCount()
            time_taken = (c2 - c1) / cv2.getTickFrequency()

            # Display tracker type on frame
            cv2.putText(frame, tracker_type + " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

            # Display FPS on frame
            # cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

            cv2.putText(frame, "Revolutions/sec RPS : " + str(instant_angular_speed), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 170, 50), 2);

            cv2.putText(frame, "AngularSpeed (deg/sec) : " + str(instant_angular_speed * 360), (100, 200),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "TargetAngularSpeed (deg/sec) : " + str(section_angular_speed) , (100, 250),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);


            cv2.putText(frame, "Time to reach the target [s] : " + str(laptime),
                        (100, 300),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "phi(deg) : " + str(phi) ,
                        (100, 350),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            cv2.putText(frame, "Next Phi Target" + str(phi_threshold),
                        (100, 380),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);

            cv2.putText(frame, "numTurns : " + str(numTurns),
                        (100, 400),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        (50, 200, 50), 2);
            # cv2.putText(frame, "Angle respect the reference (deg) : " + str(new_frame_angle * (180 / math.pi)),
            #             (100, 350),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.75,
            #             (50, 200, 50), 2);
            # cv2.putText(frame, "Angle respect the reference (deg) : " + str(consecutive_frame_beta * (180 / math.pi)), (100, 400),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.75,
            #             (50, 200, 50), 2);
            # Display result

            frames_received = frames_received+1
            cv2.imshow("Tracking", frame)
            # Exit if ESC pressed
            k = cv2.waitKey(1) & 0xff
            if k == 27: break

        return (times, angularSpeeds)
        #txt2save.write_file()

def add_more_sample_points_to_data(x,y,new_length):
    # # Generate some random data
    # y = (np.random.random(10) - 0.5).cumsum()
    # x = np.arange(y.size)
    #
    # # Interpolate the data using a cubic spline to "new_length" samples
    # new_length = 50
    new_x = np.linspace(x.min(), x.max(), new_length)
    new_y = sp.interpolate.interp1d(x, y, kind='cubic')(new_x)

    # Plot the results
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.plot(x, y, 'bo-')
    plt.title('Using 1D Cubic Spline Interpolation')

    plt.subplot(2, 1, 2)
    plt.plot(new_x, new_y, 'ro-')

    plt.show()
    return new_y

def transposeVectorsANDmerge(vector1,vector2):
```

```python

    vector1length = len(vector1)
    vector3 = np.zeros((vector1length, 2))
    for i in range(vector1length):
        vector3[i, 0] = vector1[i]
        vector3[i, 1] = vector2[i]
    return vector3

def plot_circle(img,circle):
    theta = np.linspace(0, 2 * np.pi, 150)
    radius = circle[2]
    a = radius * np.cos(theta) + circle[0]
    b = radius * np.sin(theta) + circle[1]
    #img, axes = plt.subplots(1)
    plt.plot(a, b)




def plotDataANDFittedData(x, y, ccx, ccy, R,reference_frame):

    theta = np.linspace(0, 2 * np.pi, 150)
    radius = R
    a = radius * np.cos(theta)+ccx
    b = radius * np.sin(theta)+ccy
    figure, axes = plt.subplots(1)
    axes.plot(a, b)
    axes.set_aspect(1)
    plt.plot(x, y, "*", linewidth=1, markersize=2)

    im = axes.imshow(reference_frame)
    axes.annotate('axes fraction',
                xy=(ccx, ccy), xycoords='data',
                xytext=(0.8, 0.95), textcoords='axes fraction',
                arrowprops=dict(facecolor='black', shrink=0.05),
                horizontalalignment='right', verticalalignment='top')

    ###########################
    ##for more info in the plot
    #https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.axes.Axes.annotate.html#
matplotlib.axes.Axes.annotate
    ###########################

    plt.plot(ccx, ccy,"+", linewidth=2, markersize=30)
    plt.show()

def save_file(Filename):
    txt2save = readANDwrite.Text(Filename)
    txt2save.read_file()
    txt2save.print_line(2)
    plotUnits = ["RAD", "DEG"]
    txt2save.plot_file(plotUnits[1],column2read,cutType, numofSamples2cut, time_shift)

def load_file(plotType, plotUnits, fileName, column2read, cutType, numofSamples2cut,
time_shift):
    txt2load = readANDwrite.Text(fileName)
    txt2load.read_file()

    if plotType == "AVERAGE":
        print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
        txt2load.plot_average_file(plotUnits, column2read, cutType, numofSamples2cut,
time_shift)
    else:
        txt2load.plot_file(plotUnits, column2read, cutType, numofSamples2cut,
time_shift)

    return txt2load
```

```python

def load_files(plotType, fileName1, fileName2, column2readFile1, column2readFile2):
    txt2load = readANDwrite.Text(fileName)
    txt2load.read_file()

    plotUnits = ["RAD", "DEG"]
    if plotType == "AVERAGE":
        print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
        txt2load.plot_average_file(plotUnits[0], column2read)
    else:
        txt2load.plot_file(plotUnits[0], column2read)

def get_max_of_values(value1,value2):
    max = value1
    if value1<value2:
        max = value2
    return max

def periodic_corr(x, y):
    """Periodic correlation, implemented using the FFT.

    x and y must be real sequences with the same length.
    """
    return ifft(fft(x) * fft(y).conj()).real

def save_data_in_file(angularSpeeds, times, FileName):

    time.sleep(5)

    txt2save = readANDwrite.Text(FileName)
    for angularSpeed in angularSpeeds:
        txt2save.set_angularSpeed(angularSpeed)
    for t in times:
        txt2save.set_time(t)
    #print(angularSpeeds, times)
    txt2save.write_file()

def detect_shapes(img):
    # convert BGR to RGB to be suitable for showing using matplotlib library
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # make a copy of the original image
    cimg = img.copy()
    # convert image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # apply a blur using the median filter
    img = cv2.medianBlur(img, 5)
    img_heigth = img.shape[0]
    img_width = img.shape[1]
    # finds the circles in the grayscale image using the Hough transform
    imag_diagonal_length = math.sqrt(img.shape[0]**2+img.shape[1]**2)
    circles = cv2.HoughCircles(image=img, method=cv2.HOUGH_GRADIENT, dp=0.05,
                                minDist=imag_diagonal_length, param1=110, param2=39,
minRadius=int(np.floor(img_heigth/4)/4), maxRadius=int(np.floor(img_heigth/8)))
    plt.imshow(img)
    plt.show()
    print("Image_diagonal_length",imag_diagonal_length,"height",img.shape[0],"width",
img.shape[1])
    for co, i in enumerate(circles[0, :], start=1):

        print(co, "and", i)
        plot_circle(img, i)



    #     # draw the outer circle in green
        #cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
    #     # draw the center of the circle in red
    #     cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)
    #
    # # print the number of circles detected
    # print("Number of circles detected:", co)
    # save the image, convert to BGR to save with proper colors
    # cv2.imwrite("coins_circles_detected.png", cimg)
    # show the image
    # Red color in BGR




    #cv2.circle(img, center_circle, radius, color, thickness)
    plt.imshow(cimg)
    plt.show()
    return i

def detect_target(image):


    cap = cv2.VideoCapture(-1)

    # img = cv2.imread('/home/pyarena/python/OpenCV/objectDetection/image1.jpg')
    configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
    weightPath = 'frozen_inference_graph.pb'

    cap.set(3, 640)  #
    cap.set(4, 480)  #

    # classNames = []
    #
    # classFile = 'coco.names'
    # with open(classFile, 'rt') as f:
    #     classNames = f.read().rstrip('\n').split('\n')
    #


    net = cv2.dnn_DetectionModel(weightPath, configPath)
    net.setInputSize(320, 320)
    net.setInputScale(1.0 / 127.5)
    net.setInputMean((127.5, 127.5, 127.5))
    net.setInputSwapRB(True)

    while True:  #
        success, img = cap.read()  #
        classIds, confs, bbox = net.detect(img, confThreshold=0.5)
        print(classIds, bbox)

        if len(classIds) != 0:  #
            for classId, confidence, box in zip(classIds.flatten(), confs.flatten(), bbox):
                cv2.rectangle(img, box, color=(0, 255, 0), thickness=2)
                # cv2.putText(img, classNames[classId - 1].upper(), (box[0] + 10, box[1] + 30), cv2.FONT_HERSHEY_COMPLEX,
                #             1, (0, 255, 0), 2)

        cv2.imshow('output', img)
        cv2.waitKey(1)


##########################################
#Main section
##########################################
tracker_types = ['0        ', '1',   '2'  , '3'   ,     '4'     ,  '5'    ,  '6'    , '7']
tracker_types = ['BOOSTING', 'MIL', 'KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', '
CSRT']
tracker_type = tracker_types[7]
calibration_type = ["CAPTURE","CAMERA"]
#(ccx, ccy, R) = getCircleParameters(calibration_type[1], tracker_type)
#print(ccx, ccy, R)
#(ccx, ccy, R) = ( 316.210044289178, 285.71966217053006, 65.38995024645776)
#(ccx, ccy, R) = (308.7521835339925, 240.50679891472834, 218.77469430950904) #platos +
fronton ball
#(ccx, ccy, R) = (342.80326695024206, 247.0500457462208, 175.7511889181124)#in lab and
camera
#(ccx, ccy, R)= (330.790620646499, 242.1131044484291, 193.85764190881648)
(ccx,ccy,R) = (335.490029240928, 241.71791300211592, 179.51882038995555)

# # time.sleep(4)
polygon_vertex = 10


[times,angularSpeeds] = trackCameraAngularSpeedTIME( polygon_vertex, ccx, ccy, R,
tracker_type)


# ##############################################################
# ##################set off the plt used in the Dynamic Plot##
# ##############################################################
plt.ioff()
#
# print("tracker finished")
#
# fileName2save = "Data2save_01.txt"
# #save_data_in_file(angularSpeeds, times, fileName2save)
# print("tracker finished, lets save the work")
#
# img_platform = cv2.imread("PLATFORM.jpg")
# detect_shapes(img_platform)
#
#
#
#
#
#
# #
# #
# # time.sleep(5)
# # txt2save = readANDwrite.Text("Data2save_07.txt")
# # for angularSpeed in angularSpeeds:
# #     txt2save.set_angularSpeed(angularSpeed)
# # for t in times:
# #     txt2save.set_time(t)
# # #print(angularSpeeds, times)
# # txt2save.write_file()
#
#
#
#
#
#
#
#
# #
#
#
# plotType = ["AVERAGE", "EXACT"]
# plotUnits = ["RAD", "DEG"]
# # data2save = load_file(plotType[0],plotUnits[0],"Data2save.txt",2, "CUT",180, 9.2)
# # movilData = load_file(plotType[0],plotUnits[1],"movilData.txt",3, "NOCUT",100, 9.2)
# # plt.gca().legend(('tracking algorithm(average)','movile data recorded(average)'))
# time_shift = 0
```

```python
# data2save = load_file(plotType[1],plotUnits[0],"Data2save_01.txt",3, "NOCUT",180,
time_shift)
#
# plt.gca().legend(('tracking algorithm','movile data recorded'))
# plt.show()
#
#
#
#
#
#
# plotType = ["AVERAGE", "EXACT"]
# plotUnits = ["RAD", "DEG"]
#
# time_shift = 0
# data2save = load_file(plotType[0],plotUnits[0],"Data2save_01.txt",3, "NOCUT",180,
time_shift)
# plt.gca().legend(('tracking algorithm','movile data recorded'))
# plt.show()
# #
#
#
#
#
#
#
#
#
#

# #
plotType = ["AVERAGE", "EXACT"]
plotUnits = ["RAD", "DEG"]
# data2save = load_file(plotType[0],plotUnits[0],"Data2save.txt",2, "CUT",180, 9.2)
# movilData = load_file(plotType[0],plotUnits[1],"movilData.txt",3, "NOCUT",100, 9.2)
# plt.gca().legend(('tracking algorithm(average)','movile data recorded(average)'))
time_shift = 0
data2save = load_file(plotType[1],plotUnits[0],"Data2save_07.txt",2, "NOCUT",180,
time_shift)
movilData = load_file(plotType[1],plotUnits[1],"movildata1txt.txt",3, "NOCUT",1500,
time_shift)
plt.gca().legend(('tracking algorithm','movile data recorded'))
plt.show()
# #
# #
# #
# #
#
#
#
#
#
# #
#load_files(plotType[0],"movilData.txt","Data2save.txt",3, 2)

angularSpeeds_d2s = data2save.get_angularspeeds()
angularSpeeds_mD = movilData.get_angularspeeds()
angularSpeeds_d2s = readANDwrite.moving_average(angularSpeeds_d2s)
angularSpeeds_mD = readANDwrite.moving_average(angularSpeeds_mD)
# #
# #
# # x = np.arange(len(angularSpeeds_d2s))
# #
# # new_length = len(angularSpeeds_mD)
# # angularSpeeds_d2s = add_more_sample_points_to_data(x,angularSpeeds_d2s,new_length)
# # #angularSpeeds_d2s = readANDwrite.cut_begining_array_num_samples(angularSpeeds_d2s
, 0)
```

```
# # x = np.arange(len(angularSpeeds_d2s))
# # #[angularSpeeds_d2s, angularSpeeds_mD] = readANDwrite.set_same_dimensions_arrays(
angularSpeeds_d2s, angularSpeeds_mD)
# # #corr = signal.correlate(angularSpeeds_d2s, angularSpeeds_mD, mode='same') / len(
angularSpeeds_d2s)
# # corr = np.correlate(angularSpeeds_d2s, angularSpeeds_mD,  "same")
# #
# # #corr = periodic_corr(angularSpeeds_d2s, angularSpeeds_d2s)
# # max_corr = np.max(corr)
# # corr_normalized = corr/max_corr
# # print(max_corr)
# # #max_corr_index = corr.index(max_corr)
# # ind = np.argmax(corr)
# #
# # print("maxcorr",max_corr, ind, len(corr))
# # plt.show()
# # plt.plot(x, angularSpeeds_d2s)
# # plt.plot(x, angularSpeeds_mD)
# # x = np.arange(-np.floor(len(corr)/2), np.floor(len(corr)/2)+1
)##########################################################################+1 out
# # #x = np.arange(0,np.floor(len(corr)))
# # print(len(corr),len(x))
# # plt.plot(x,corr_normalized)
# # plt.show()
# # x_max_corr = x[ind]
# # print(x_max_corr,"xmax corr")
# #
# # # max_time_movilData = movilData.get_timing()[-1]
# # # max_time_data2save = data2save.get_timing()[-1]
# # # max_time = get_max_of_values(max_time_movilData, max_time_data2save)
# # # print("max time", max_time, x_max_corr)
# # # time_shift = -x_max_corr*(max_time+20)*2*2/(len(corr))
# # # print(time_shift)
# # # data2save = load_file(plotType[0],plotUnits[0],"Data2save.txt",2, "CUT",180,
time_shift)
# # # movilData = load_file(plotType[0],plotUnits[1],"movilData.txt",3, "CUT",1500, 0)
# # # plt.show()
# # #
# #

```

```
186 101
```

```python
import matplotlib.pyplot as plt
plt.ion()
class DynamicUpdate():
    #Suppose we know the x range
    min_x = 0
    max_x = 20
    windows_shift = 20# secs

    def on_launch(self):
        #Set up plot
        self.figure, self.ax = plt.subplots()
        self.lines, = self.ax.plot([],[])
        #Autoscale on unknown axis and known lims on the other
        self.ax.set_autoscaley_on(True)
        self.ax.set_xlim(self.min_x, self.max_x)
        #Other stuff
        self.ax.grid()
        return self.max_x


    def window_shifting(self):

        self.min_x = self.max_x
        new_max = self.max_x + self.windows_shift
        self.ax.set_xlim(self.min_x, new_max)
        self.max_x = self.max_x + self.windows_shift
        return new_max



    def on_running(self, xdata, ydata):
        #Update data (with the new _and_ the old points)
        self.lines.set_xdata(xdata)
        self.lines.set_ydata(ydata)
        #Need both of these in order to rescale
        self.ax.relim()
        self.ax.autoscale_view()
        #We need to draw *and* flush
        self.figure.canvas.draw()
        self.figure.canvas.flush_events()

    #Example
    def __call__(self):
        import numpy as np
        import time
        self.on_launch()
        xdata = []
        ydata = []
        for x in np.arange(0,1000,0.5):
            xdata.append(x)
            ydata.append(np.exp(-x**2)+10*np.exp(-(x-7)**2))
            self.on_running(xdata, ydata)
            time.sleep(1)
        return xdata, ydata

#d = DynamicUpdate()
#d()

```

APPENDIX D. CIRCULAR FIT

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def circular_fit(samplesx, samplesy):
    xmean = np.sum(samplesx)/ len(samplesx)
    ymean = np.sum(samplesy)/ len(samplesy)

    samplesx = [i-xmean for i in samplesx]
    samplesy = [i-ymean for i in  samplesy]

    Su = np.sum(samplesx)
    Suu = [i**2 for i in samplesx]
    Suu = np.sum(Suu)
    Suuu = np.sum([i**3 for i in samplesx])
    Svv = np.sum([i**2 for i in samplesy])
    Svvv = np.sum([i ** 3 for i in samplesy])
    Suv = []
    Suvv = []
    Svuu = []
    for i in range(len(samplesx)):
        Suv.append(samplesx[i]*samplesy[i])
        Suvv.append(samplesx[i]*samplesy[i]**2)
        Svuu.append(samplesy[i] * samplesx[i]**2)

    Suv = np.sum(Suv)
    Suvv = np.sum(Suvv)
    Svuu = np.sum(Svuu)

    S = np.array([[Suu, Suv ],[Suv, Svv]])
    Sinv = np.linalg.inv(S)
    SS = np.array([0.5*(Suuu + Suvv), 0.5*(Svvv + Svuu)])

    res = Sinv.dot(SS)
    alpha = (Suu + Svv)/len(samplesx) + res[0]**2 + res[1]**2

    R = np.sqrt(alpha)
    angle = np.linspace(0, 2 * np.pi, 150)
    radius = R
    x = radius * np.cos(angle)  + res[0]
    y = radius * np.sin(angle) + res[1]
    figure, axes = plt.subplots(1)
    axes.plot(x , y )
    axes.set_aspect(1)
    plt.title('Parametric Equation Circle')
    plt.plot(samplesx, samplesy)
    plt.show()

    return res, R

xi = 0.5*np.arange(0,8)
yi = []
for i in range(len(xi)):
    yi.append(0.5*xi[i]**2)
print(xi)
print(yi)
[res, R] = circular_fit(xi, yi)
```

APPENDIX E. ANGULAR TRACKING STUDY

```
function [index] = findClosest(arr1,time1)
closest = arr1(1);

for i=1:length(arr1)
    if(arr1(i)>time1)
        closest = i;
        break
    end
end
index = closest;
end
```

```
Not enough input arguments.

Error in findClosest (line 2)
closest = arr1(1);
```

Published with MATLAB® R2021b

```
clear all;
close all;
% step1_trackingx = interval1_tracking(:,1)
% step1_trackingx = interval1_tracking(:,2)
filenameDavid = 'movildata12.xls';
filenameTrack = 'Datasetonline.xls';
% filename1 = 'tracking11.csv';
% filename2 = 'movile11.csv';
% filename11 = 'tracking2.csv';
% filename22 = 'movile2.csv';
% Ttracking1 = readtable(filename1);
% Tmovile1 = readtable(filename2);
% Ttracking2 = readtable(filename11);
% Tmovile2 = readtable(filename22);
TDavid= readtable(filenameDavid);
Ttracking = readtable(filenameTrack);
%
% trackX_int1 = Ttracking1.x;
% trackY_int1 = Ttracking1.y;
% aver_track1 = mean(trackY_int1)
% VAR_track1 = var(trackY_int1)
%
%
% movileX_int1 = Tmovile1.x;
% movileY_int1 = Tmovile1.y;
% aver_movile1 = mean(movileY_int1)
% VAR_movile1 = var(movileY_int1)
%
%
%
% cross = cov(trackY_int1, movileY_int1)
%
%
%
% trackX_int2 = Ttracking2.x;
% trackY_int2 = Ttracking2.y;
% aver_track2 = mean(trackY_int2)
% VAR_track2 = var(trackY_int2)
%
%
% movileX_int2 = Tmovile2.x;
% movileY_int2 = Tmovile2.y;
% aver_movile2 = mean(movileY_int2)
% VAR_movile2 = var(movileY_int2)

%
% cross = cov(trackY_int2, movileY_int2)


tmovil = [];
wmovil = [];
t = TDavid.time;
```

```
w = TDavid.wz_rad_s_;
for i=1:length(t)
    tcell = cell2mat(t(i));
    tdata = strrep(tcell, ',', '.');
    t1 = str2double(tdata);
    tmovil = [tmovil t1];

    wcell = cell2mat(w(i));
    wdata = strrep(wcell, ',', '.');
    w1 = str2double(wdata);
    w1 = -w1*(180/pi);
    wmovil = [wmovil w1];
end


wmovilavr = movmean(wmovil, 101);
plot(tmovil,wmovilavr)





ttracking = []
wtracking = []
tt = Ttracking.time;
wt = Ttracking.wz_rad_s_;
for i=1:length(tt)
    tcell = cell2mat(tt(i));
    tdata = strrep(tcell, ',', '.');
    tt1 = str2double(tdata);
    ttracking = [ttracking tt1];

    wtt= cell2mat(wt(i));
    wdata = strrep(wtt, ',', '.');
    w1 = str2double(wdata);
    wtracking = [wtracking w1];
end

hold on
% wmovilavr = movmean(wmovil, 101);
plot(ttracking,wtracking)

wmovile_sel = [];

for i=1:length(wtracking)
    t=ttracking(i);
    time_close = findClosest(tmovil,t);
    wmovile_sel = [wmovile_sel wmovilavr(time_close)];
end


% ttracking(length(ttracking)) = [];
% ttracking(length(ttracking)) = [];
```

```
plot(ttracking, wmovile_sel)


relError =[]
err = (wmovile_sel - wtracking)./wmovile_sel *100;
figure(5)
relativeError =  mean(abs(err))
plot(ttracking, abs(err))
  xlabel("Time[s]")
    ylabel("E[%]")
    title("Relative error ")
figure(6)
absoluteError = mean(abs(wmovile_sel - wtracking));
plot(ttracking,abs(wmovile_sel - wtracking))
  xlabel("Time[s]")
    ylabel("Error")
    title("Absolute error ")



%%%%STEP1

step1time = ttracking(1:1:69);
step1tracking = wtracking(1:69);
step1movile= wmovile_sel(1:69);
figure(10)
plot(step1time,step1tracking)
hold on
plot(step1time,step1movile)
 legend('Tracking algorithm','Mobile gyro')
    xlabel("Time[s]")
    ylabel("Angular velocity[º/s]")
    title("Step1 comparison")

relError =[]
err = (step1movile - step1tracking)./step1movile *100;

figure(111)
plot(step1time, abs(err))
    xlabel("Time[s]")
    ylabel("E[%] ")
    title("Relative error Step1")
    figure(112)
plot(step1time,abs(step1movile - step1tracking))
    xlabel("Time[s]")
    ylabel("Angular velocity difference [º/s]")
    title("Absolute error Step1")
relativeErrorStep1 = mean(abs(err));
absoluteErrorStep1 = mean(abs(step1movile - step1tracking));
```

```
%%STEP2



step2time = ttracking(98:1:145);
step2tracking = wtracking(98:1:145);
step2movile= wmovile_sel(98:1:145);
figure(11)
plot(step2time,step2tracking)
hold on
plot(step2time,step2movile)
 legend('Tracking algorithm','Mobile gyro')
    xlabel("Time[s]")
    ylabel("Angular velocity[º/s]")
    title("Step2 comparison")


relError =[];
err = (step2movile - step2tracking)./step2movile *100;

figure(113)
plot(step2time, abs(err))
    xlabel("Time[s]")
    ylabel("E[%] ")
    title("Relative error Step2")
    figure(114)
plot(step2time,abs(step2movile - step2tracking))
    xlabel("Time[s]")
    ylabel("Angular velocity difference [º/s] ")
    title("Absolute error Step2")

relativeErrorStep2 = mean(abs(err));
absoluteErrorStep2 = mean(abs(step2movile - step2tracking));



%STEP3



step3time = ttracking(145:1:164);
step3tracking = wtracking(145:1:164);
step3movile= wmovile_sel(145:1:164);
figure(12)
plot(step3time,step3tracking)
hold on
plot(step3time,step3movile)
 legend('Tracking algorithm','Mobile gyro')
    xlabel("Time[s]")
    ylabel("Angular velocity[º/s]")
    title("Step3 comparison")
relError =[];
```

```
err = (step3movile - step3tracking)./step3movile *100;

figure(115)
plot(step3time, abs(err))
    xlabel("Time[s]")
    ylabel("E[%] ")
    title("Relative error Step3")
    figure(116)
plot(step3time,abs(step3movile - step3tracking))
    xlabel("Time[s]")
    ylabel("Angular velocity difference [º/s]")
    title("Absolute error Step3")

relativeErrorStep3 = mean(abs(err));
absoluteErrorStep3 = mean(abs(step3movile - step3tracking));


%STEP4$%%%%%%%%%%



step4time = ttracking(168:end);
step4tracking = wtracking(168:end);
step4movile= wmovile_sel(168:end);
figure(13)
plot(step4time,step4tracking)
hold on
plot(step4time,step4movile)
 legend('Tracking algorithm','Mobile gyro')
    xlabel("Time[s]")
    ylabel("Angular velocity[º/s]")
    title("Step4 comparison")

relError =[];
err = (step4movile - step4tracking)./step4movile *100;

figure(117)
plot(step4time, abs(err))
    xlabel("Time[s]")
    ylabel("E[%] ")
    title("Relative error Step4")
    figure(118)
plot(step4time,abs(step4movile - step4tracking))
    xlabel("Time[s]")
    ylabel("Angular velocity difference [º/s] ")
    title("Absolute error Step4")


relativeErrorStep4 = mean(abs(err));
absoluteErrorStep4 = mean(abs(step4movile - step4tracking));



display("table analysis")
```

```
variancetrack = var(wtracking);
variancemob = var(wmovile_sel);
cross = cov(wtracking, wmovile_sel);
absoluteError;
relativeError;
corrsteps = cross/(sqrt(variancetrack)*sqrt(variancemob));

variancetrack1 = var(step1tracking);
variancemob1 = var(step1movile);
cross1 = cov(step1tracking, step1movile);
relativeErrorStep1;
absoluteErrorStep1;
corrstep1 = cross1/(sqrt(variancetrack1)*sqrt(variancemob1));

variancetrack2 = var(step2tracking);
variancemob2 = var(step2movile);
cross2 = cov(step2tracking, step2movile);
relativeErrorStep2;
absoluteErrorStep2;
corrstep2 = cross2/(sqrt(variancetrack2)*sqrt(variancemob2));

variancetrack3 = var(step3tracking);
variancemob3 = var(step3movile);
cross3 = cov(step3tracking, step3movile);
relativeErrorStep3;
absoluteErrorStep3;
corrstep3 = cross3/(sqrt(variancetrack3)*sqrt(variancemob3));

variancetrack4 = var(step4tracking);
variancemob4 = var(step4movile);
cross4 = cov(step4tracking, step4movile);
relativeErrorStep4;
absoluteErrorStep4;
corrstep4 = cross4/(sqrt(variancetrack4)*sqrt(variancemob4));
```

```
Warning: Column headers from the file were modified to make them valid MATLAB
identifiers before creating variable names for the table. The original column
headers are saved in the VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as
table variable names.
Warning: Column headers from the file were modified to make them valid MATLAB
identifiers before creating variable names for the table. The original column
headers are saved in the VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as
table variable names.

ttracking =

     []


wtracking =

     []
```