TÉCNICO LISBOA

Development of an Instrumentation System for Small Rockets

Guilherme Clavel Perestrelo Caiado

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Agostinho Rui Alves da Fonseca,
Prof. Paulo Jorge Soares Gil

Examination Committee

Chairperson: Prof. João Manuel Lage de Miranda Lemos
Supervisor: Prof. Agostinho Rui Alves da Fonseca
Member of the Committee: Prof. Luís Alberto Gonçalves de Sousa

December 2016

Acknowledgments

The author would like to thank the flowing people:

the Aeroclube de Torres Vedras for their assistance and runway, particularly Captain João Francisco and Carolina;

Diogo Henriques and Duarte Donas-Boto for their assistance and equipment;

Aerotec for their equipment;

professors Agostinho Fonseca and Paulo Gil for their knowledge and input;

all my teacher for all they gave me;

my friends (without whom studying would have bee much easier, but hardly any fun);

and last, but not least, my family, particularly my parents and brothers, for standing with me along this, suddenly, hard path and letting me know the delay was a damn problem, but nothing I couldn't overcome, I love you all.

Abstract

The objective of this thesis is to develop a compact instrumentation system for a model rocket, and a telemetry base station, using off-the-shelf components. First the actual parameters to be recorded are discussed and chosen. Four data types were deemed important: time, atmospheric data (pressure and temperature), translational data (position, velocity, and acceleration) and orientation data (attitude, angular velocity, and magnetic field orientation). Second, the system is designed and the component selected, all while analysing the advantages and limitations of each part, culminating in the selection of the test rocket. Third, the next logical step is to create the software to interface with the system and components, this includes the on board micro-controller programming as well as the telemetry station computer executable. Fourth, the whole system is ground tested, to calibrate the sensors and validate adequate system function before the start of airborne testing. Finally, the system is flight tested, first only during decent, to validate the recovery system, and then in the chosen test rocket. It was concluded that the instrumentation system achieved all the proposed requirements, the data received in the flight-test closely matches the simulation data, with the differences being more likely due to imperfections in engine construction than due to system error.

Keywords: Instrumentation, rocket, telemetry, compact, off-the-shelf

Resumo

O objectivo desta tese é o desenvolvimento de um sistema de instrumentação para um "*model rocket*", e uma estação base de telemetria utilizando componentes *off-the-shelf*. Primeiro foram escolhidos os parâmetros a serem medidos. Quatro tipos de dados foram considerados importantes: tempo, dados atmosféricos (pressão e temperatura), dados de translação (posição, velocidade e aceleração), e dados de orientação espacial (atitude, velocidade angular, e orientação magnética). De seguida o sistema foi projectado e os componentes seleccionados, sempre analizados as vantagens e limitações de cada componente. O próximo passo lógico foi o desenvolvimento do software de controlo do microcontrolador e também o executável da estação de telemetria. Iniciaram-se depois os testes de solo a fim de calibrar os instrumentos, bem como confirmar o correcto funcionamento do sistema. Por ultimo realizaram-se os ensaios em voo, primero só durante a descida, com o objectivo te validar o sistema de recuperação, e finalmente os ensaios no rocket de teste. Conclui-se que o sistema de instrumentação cumpriu todos os objectivos delineados, os dados recebidos durante os ensaios em voo ficaram muito próximos das simulações, as diferenças existentes sendo devidas a imperfeições de construção dos motores.

Palavras-Chave: Instrumentação, telemetria, foguete, compacto, *off-the-shelf*

Contents

List of Tables xi

List of Figures xi

Acronyms xvii

1 Introduction 1
- 1.1 Brief History . . . 1
- 1.2 Research Question . . . 2
- 1.3 Significance . . . 2
- 1.4 Motivation . . . 2
- 1.5 State-of-the-art . . . 2
- 1.6 Overview of the Work . . . 3

2 Preliminary Work 5
- 2.1 Rockets . . . 5
- 2.2 Parameters to Measure . . . 9

3 Acquired and developed components 15
- 3.1 Microprocessor/Base Platform . . . 15
- 3.2 Remote Communication . . . 17
- 3.3 Data Recording . . . 19
- 3.4 Inertial, magnetic and atmospheric data . . . 20
- 3.5 GNSS receiver: Time, Position and Velocity . . . 22
- 3.6 Electrical Power . . . 24
- 3.7 Rocket, engine and platform . . . 26
- 3.8 Other equipment (Ground Segment) . . . 28
- 3.9 Assembly . . . 28

4 Algorithm and Code development 31
- 4.1 Rocket Segment: Arduino sketch . . . 31
- 4.2 Ground Segment: Telemetry Station C++ code . . . 34
- 4.3 Python Binary to CSV code . . . 35
- 4.4 Matlab Data Handling Code . . . 36

5 Ground Testing 37
- 5.1 Weighing and CG . . . 37
- 5.2 Attitude Test . . . 38
- 5.3 Pressure and Temperature Test . . . 41

5.4 GPS 43

6 Launch Procedure **53**
6.1 Launch site and conditions 53
6.2 Instrumentation system setup 54
6.3 Rocket launch setup 54

7 Flight Testing **55**
7.1 Static Fall 55
7.2 Aircraft Delivered Fall 57
7.3 Flight Test 59

8 Conclusions **75**
8.1 Achievements 76
8.2 Future Work 76

A Arduino code, Rocket Segment **81**
A.1 Arduino 81

B Matlab code **89**
B.1 Rocket Altitude Simulation 89
B.2 CSV File Importer, Ground Segment 91

C Python code, Ground Segment **93**
C.1 .bin to .csv 93

D C++ code, Ground Segment **95**

E Drawings **101**

List of Tables

2.1 Rocket Engine Comparison 7
2.2 Parameters to Measure 10

3.1 Arduino Comparison 16
3.2 XBee Specifications 17
3.3 XBee Setup 19
3.4 Altimu Characteristics 21
3.5 GPS Options 23
3.6 NMEA Sentences and UBX PVT 23
3.7 UBX PVT structure, U/I=unsigned/signed followed by the number of bytes 24
3.8 uBlox Configuration 24
3.9 uBlox Dynamic Platform Models 24
3.10 Current Draw 25
3.11 Battery Comparison 25
3.12 Current Budget 26
3.13 Instrument Mass 26
3.14 Rocket Comparison 27
3.15 Rocket Engine Comparison 27
3.16 Mass Budget 28
3.17 Connections 28
3.18 Simulation Results 30

4.1 Arduino output message 33
4.2 CSV file structure 36

5.1 Simulation Results 38
5.2 Attitude values 39
5.3 Position values 43
5.4 Velocity (m/s) 44

6.1 Launch site and clear distance 53
6.2 Launch checklist 54

7.1 Launch conditions 60

List of Figures

2.1 Model Rocket Design 5
2.2 Model Rocket Stability 6
2.3 Model Rocket Motor 7
2.4 Model Rocket Flight 9

3.1 Arduino Fio 15
3.2 XBee S2B 17
3.3 Range Test Location 18
3.4 SD Card Reader 20
3.5 Pololu Altimu-10 20
3.6 u-blox MAX-M8Q breakout with Maruwa SL1252R antenna with SAW/LNA 22
3.7 Lithium Polymer Battery 26
3.8 ESTES Maxi Alpha3 27
3.9 System diagram 29
3.10 Simulations 30

4.1 Application views 34

5.1 Attitude Test Setup 39
5.2 Acceleration, angular velocity and magnetic field, GT01 40
5.3 Attitude, GT02 41
5.4 Temperature 42
5.5 Pressure 42
5.6 Position, GT01 44
5.7 Velocity Static Test 45
5.8 Full planar trajectory GT01 45
5.9 Planar position and velocity, for position accuracy, GT01 45
5.10 Satellite coverage and planar position, GT01 46
5.11 Barometric and GPS altitude, GT01 46
5.12 Velocity, GT01, theoretical test value in red 47
5.13 Velocity components, stretch A, GT01 47
5.14 Velocity components, stretch B, GT01 48
5.15 Difference in velocity magnitude, stretch A, GT01 49
5.16 Difference in velocity magnitude, stretch B, GT01 49
5.17 Position GT01 50
5.18 Satellite coverage, GT01 50
5.19 Position for test section, GT01 50
5.20 Normalized absolute velocity, GT01 51

5.21 Velocity, GT01 52

7.1 Pressure, 2016/05/04 GT02 56
7.2 Pressure and acceleration on drops, GT02 56
7.3 Vertical velocity and altitude 58
7.4 Planar trajectory 59
7.5 Sample period, Flight Tests 60
7.6 Acceleration, Flight Test 1 61
7.7 Acceleration, Flight Test 2 62
7.8 Acceleration, Flight Test 3 62
7.9 Acceleration, Flight Test 4 62
7.10 Angular velocity, Flight Tests 63
7.11 Magnetic field, Flight Tests 64
7.12 Pitch, Roll, Yaw, Flight Tests 65
7.13 Pressure, Flight Tests 66
7.14 Temperature, Flight Tests 67
7.15 Satellites used in solution, Flight Tests 68
7.16 Altitude, Flight Tests 69
7.17 Velocity, Flight Tests 70
7.18 Velocity, Flight Test 2 71
7.19 Velocity, Flight Test 3 71
7.20 Velocity, Flight Test 4 71
7.21 Google maps overlay, Flight Test 72
7.22 Difference between time in telemetry data and local data 73
7.23 Comparison of acceleration during powered phase 74

List of Algorithms

4.1 Read GPS 32
4.2 Main Loop 34
4.3 Make Vectors 35

Acronyms

AHRS Attitude and Heading Reference System. 11, 12, 22, 33, 38, 73

BBR Battery Backup RAM. 24

CAM Computer-aided Manufacture. 28

CEP Circular Error Probable. 42

CFG Configuration. 24

CG Centre of Gravity. 6, 8, 29, 37, 38

CP Centre of Pressure. 6, 8, 29

CSV Comma Separated Value. 31, 35

DRMS Distance Root Mean Squared. 42

FIFO First In, First Out. 74

FTDI Future Technology Devices International. 28

GNSS Global Navigation Satellite System. 3, 10, 11, 16, 22–24, 28, 31, 55, 66

GPS Global Positioning System. 10, 11, 18, 22–24, 26, 28, 31–33, 35, 42, 45–47, 52, 56, 65, 66, 73

I^{2}C Inter-Integrated Circuit. 16, 21, 31

IMU Inertial Measurement Unit. 3, 21

LOS Line of Sight. 17, 19

MARG Magnetic, Accelerometer and Rate Gyro. 3, 21

MISO Master Input, Slave Output. 19, 28

MOSI Master Output, Slave Input. 19, 28

NAR National Association of Rocketry. 6, 7

NASA National Aeronautics and Space Administration. 1

NAV Navigation. 23, 24

NMEA National Marine Electronics Association. 22, 23

NSRP NASA Sounding Rocket Program. 1

PVT Position, Velocity and Time. 23, 24, 31

R&D research and development. 17

SCL Serial Clock. 21, 28

SCLK Serial Clock. 19, 28

SDA Serial Data. 21, 28

SM Static Margin. 6, 29

SPI Serial Peripheral Interface Bus. 16, 19, 22, 31

SS Slave Select. 19, 28

Tripoli Tripoli Rocketry Association.. 6, 7

UART Universal Asynchronous Receiver Transmitter. 16, 22, 23

UBX uBlox. 22–24, 31

USB Universal Serial Bus. 28, 53, 56

Chapter 1

Introduction

Taking measurements is an integral part of the scientific method and by extension engineering. This need to measure parameters must be addressed by using instruments, equipment specifically designed to be able to quantify a given magnitude. The process of applying instruments to the system to be studied or to a vehicle that will house and deliver them to the final destination is called instrumentation.

In the aerospace field instrumentation is of particular importance, it's an integral part of both manned and unmanned vehicles, to ascertain their flight conditions, position and heading. The nature of flying, its weight requirements, altitude and dynamics achieved provide an additional challenge when instrumenting this type of systems.

The objective here taken on is to assemble a compact, light and commercial off-the-shelf (COTS) system for a small model rocket. This is a simple enough objective but a demanding one, as will be shown, the small size, lift-off weight and dynamic nature of a rocket provide a considerable challenge.

1.1 Brief History

Although the origins of rockets can be traced back to 100 AD, in the Chinese empire, modern rocketry began whit a Russian schoolteacher, Konstantin Tsiolkovsky, who proposed rocket power space exploration, and first suggested the use of liquid propellants. A few years later Robert H. Goddard started practical experiments with rocketry. The objective was to reach heights unachievable by meteorological balloons, he was the first to build and test liquid powered rockets. [1]

Hermann Oberth, a German scientist, wrote a book about rocket travel that inspired numerous rocket clubs around the world, the advent of model rocketry. One of such groups, the *Verein fur Raumschiffahrt* (Society for Space Travel), went on to develop numerous rockets, and during World War II the V2 (under the direction of Wernher von Braun). After the allied victory, German scientist and technology were divided between the Soviet Union and the United States, which lead to the development of intercontinental ballistic missiles, and from there the space race and exploration as we know it. [1]

Alongside space exploration, other types of rockets remained, and from the numerous rocket societies came civilian rocket clubs, that is model rocketry, and from the National Aeronautics and Space Administration (NASA) suborbital test rockets came the NASA Sounding Rocket Program (NSRP), established in 1959 [2]. These are the more accessible members of the rocket family, used both for recreational use (model high power rockets) and for scientific experiments (sounding rockets), sounding rockets have been instrumental in many research fields [3].

1.2 Research Question

The objective of this thesis is to develop a light, compact, reliable and replaceable COTS instrument system capable of being used in a small rocket and providing the data deemed required, while surviving the mission. It is also important to state the procedure required for a launch, and the requirements a rocket must fulfil to be able to carry the system.

1.3 Significance

Sounding rockets have been used in research for quite some time, and have had a considerable impact in the scientific community [3], currently there is still a significant amount of research using sounding rockets [2], recent works like [4–9].

Model rockets however remain used exclusively by enthusiast, there are several reasons for this, the limited payload capability, comparably low range, the inherently short span of the mission, and (most of all) the relative cost compared to meteorological balloons, that can achieve significant altitude, carry increased payload, have a longer mission life and lower risk.

Despite this however, model rockets have one unique ability, due to the obvious similarities between sounding and model rockets, the latter are a natural stepping stone to the former, and a perfect place to develop rocket related technology. With this in mind, and given the lack of known use of model rockets for this relatively low cost type of research, it is interest to attempt to instrument even the smaller rockets to extract valuable data from inexpensive rockets, which in turn could be used in developing larger rockets.

Locking at the issue in educational oriented mindset, it is clear that instrumented rocked could be used to attract the interest of young students, and so to foster and develop a scientific and engineering approach to field testing, much like [10].

Finally, if correctly miniaturized and constructed, relatively inexpensive PCB boards of all these instruments (or a part of them) could be marketed not only to the amateur rocket community, but also to all the remote control circle, as well as any applications where a small, cheap and reliable form of telemetry/instrumentation could be of use.

1.4 Motivation

One of the ulterior motives of this thesis is to be able to foment the development of scientific work at Técnico around rocketry, with the possibility of using high power model rockets to test technology, hopefully amounting to a fully-fledged sounding rocket in the (likely distant) future.

1.5 State-of-the-art

The most developed area of small instruments for airborne vehicles lies in small remote controlled and autonomous air vehicle, "drones", which use the sensors for control, and high altitude ballooning, which have looser weight restrictions and sample rate requirements.

Currently most systems available for model rockets are simple altimeters, most of which only display the maximum altitude achieved and possible activate the ejection charge at a given altitude or at the apogee.

There are a few exceptions however, notably the Altus Metrum System [11] which records data from a 9-axis Inertial Measurement Unit (IMU), a Global Navigation Satellite System (GNSS) receiver, and

atmospheric data, but has a cost of around 400 $US.

Arguably the best board soon to hit the market is the Apollo from Carbon Origins [12], which has all the previous sensors, in addition to UV and IR sensors, SD card logging, Bluetooth, Wi-Fi and an OLED screen, a great board, but with no possibility of telemetry and a hefty price of over 200 $US (by current estimates), and of course not yet in the market.

All the previous boards have also the disadvantage of being proprietary and requiring the full board to be bought and replaced in its entirety, and they are locked in a specific configuration with no option to upgrade sensors.

1.6 Overview of the Work

In this thesis, a compact, light, modular, COTS and replaceable instrumentation system for a small rocket is developed. It is able to record in an SD card and telegraph using Xbee, time data from the processor internal clock, 9-axis Magnetic, Accelerometer and Rate Gyro (MARG) data, barometric and temperature data, and GNSS time, position, and velocity.

In comparison with the systems discussed in Section 1.5, this instrumentation system is able to record all the major data to study the flight of the rocket (IMU, GNSS, barometric data), it is sampled at over 40 Hz, which matches most remote controlled/autonomous vehicle requirements, it theoretically works at high altitude ballooning heights, it is extremely light and compact while remaining modular and independently sourced (unlike the referred rocket systems), and is able to send telemetry data at long distance (which could further be increased by attaching a different antenna to the Xbee).

The design process first starts with an overview of rocket flight and the parameters to be measured. Proceeding then to equipment selection and testing, keeping in mind the quantities to be measured, the functional requirements and the compact, light and modular/replaceable/COTS design philosophy, and finally the choice of test rocket. The following stage is to develop adequate code for both the instrument system and ground tracking station. The next step is ground testing the equipment to obtaining relevant sensor data, such as offsets and bias, for calibration, and to verify the correct functioning of the system to clear it for airborne use. The conclusion is to flight test the equipment, first only during a drop using to validate the recovery system and system function in a more controlled environment, and finally to launch the system from the test rocket, confirming that the system fully accomplishes the design goals.

Chapter 2

Preliminary Work

In this chapter the basics of rocket design and flight will be touched upon, as well as the parameters to measure.

2.1 Rockets

Unlike most other forms of flight, rockets rely purely on thrust to counteract the force of gravity. In rocket flight, aerodynamic considerations are restricted to the design of the overall low drag shape of the rockets, lifting surfaces only appearing in the fins; this is particularly apparent in full size rockets where there is no aerodynamic stabilization at all, stability is achieved by thrust vectoring (such as engine gimbals).

2.1.1 Model Rocket Design

Model rocket designs normally follow the same structure, there are exceptions like plane/rocket hybrids, but generally they all keep to the design described in fig. 2.1.

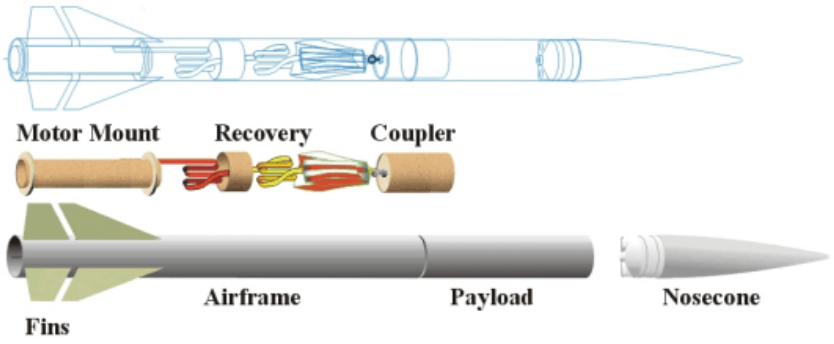


Figure 2.1: Typical model rocket design [13]

Smaller rockets normally consist only of a short airframe where the motor mount and recovery system are housed, and a nose cone where (in the larger of these) there may be some payload. Launch lugs or buttons are outfitted to the exterior of the airframe in order to maintain stability during launch. Larger rockets may also have an intermediate section between the nose-cone and the recovery system where some payload is placed. To achieve greater heights rockets may have stages, just like their equivalent

in full size rockets, stages consist on propulsion and a surrounding airframe, that is a motor mount and an outer shell, usually with additional fins for stabilization [14] [15].

Similarly to model planes, these rockets are built with lightweight materials, in the smaller ones construction is normally out of tough cardboards, balsa wood and plastics; larger rockets utilize high end aerospace plastics, glass or carbon fibbers, and some may utilize aluminium or other light alloys [14] [15].

Stability

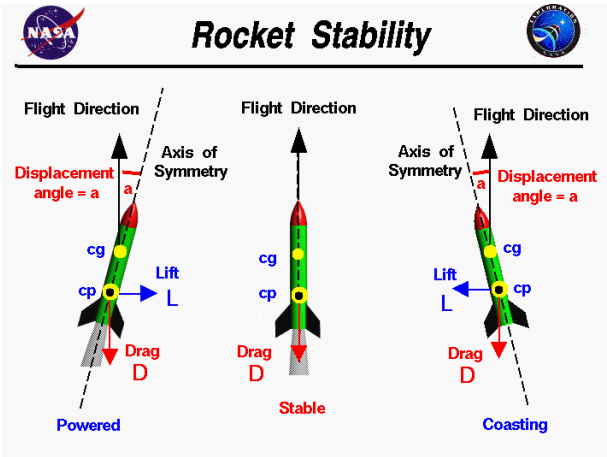


Figure 2.2: Model rocket stability [16]

Model rockets rely on aerodynamics for stability, it can be described as a balance between the position of two points, the Centre of Gravity (CG) and the Centre of Pressure (CP). The CP is the point at which aerodynamic forces can be applied as only two forces (lift and drag), i.e. a null aerodynamic moment. The CG is the point at which the weight of the rocket can be applied [17].

As can be seen in fig. 2.2, if the CP is located below the CG both lift and drag forces generate a restorative moment about the CG, i.e. the displacement angle will tend towards zero. [14]

CG position does not move as much in model rockets as in full sized ones since the weight of propellant represents normally bellow 20% of total mass (as opposed to 80%), however the CP does tend to shift depending on winds, as such it is advisable to have a stability margin also called Static Margin (SM) (the distance between CG and CP) of at least 2 calibers i.e. 2 diameters [15].

2.1.2 Motors

Model rocket motors denomination generally obeys a certain rule consisting on a letter, a first number and a second number. Though not mandatory this standard is universally accepted and respected, by both manufacturers and regulatory organizations such as Tripoli and National Association of Rocketry (NAR)

The letter represents the total impulse range [15] (commonly measured in newton-seconds [N.s]), with each letter representing twice the impulse of the one preceding it, a chart of these denominations with notes on important values are included in table 2.1 [18].

The first number refers to the average thrust (measured in newton [N])[15]. Higher average thrust results in faster acceleration and a shorter burn time, which can allow for more mass to be lifted at launch, sacrificing a higher altitude as the powered phase will be much shorter and so drag and gravity will dominate the rocket dynamics much sooner.

Table 2.1: Rocket Engine Comparison

Class	Total Impulse [N.s]	Comments
A	1.26–2.50	Smaller motors exist, normally named after a fraction of A
B	2.51–5.00	
C	5.01–10.0	
D	10.01–20.0	Largest motor shipped without certification in the EU
E	20.01–40.0	
F	40.01–80.0	
G	80.01–160	Largest commercial model rocket motor
H	160.01–320	Level 1 Tripoli/NAR certification
I	320–640	
J	640–1,280	Level 2 Tripoli/NAR
K	1,280–2,560	
L	2,560–5,120	
M	5,120–10,200	Level 3 Tripoli/NAR
N	10,200–20,500	
O	20,500–41,000	
P	41,000–81,900	FAA permit required in US
Q	81,900–164,000	
R	164,000–328,000	
S	328,000–655,000	Largest amateur rocket

The last number represents the delay, in seconds [15], between the end of thrust and the ignition of the ejection charge. This value can be 0, if there is no ejection charge, as in the case of a first stage motor. Another possibility is that instead of a number the letter P, which means that there is no delay charge but there is an ejection charge which can be triggered by a plug, normally used in rockets with remote triggering, such a safeguard for gliders, or rockets with electronic altimeters.

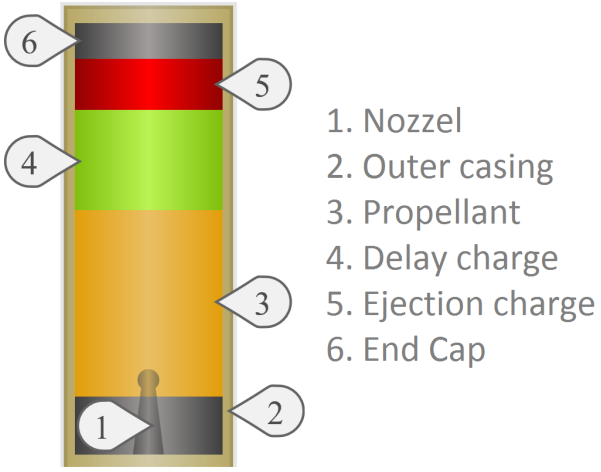


Figure 2.3: Model rocket motor [19]

Structure

Model rocket engines consist on six parts as seen in fig. 2.3. The nozzle, represented in (1) is normally made from clay and is designed to maximized thrust. The outer casing (2) is generally made of cardboard or rolled paper, its diameter is normally dependent on the total impulse, going up to 152 mm, though most motors are smaller than 29 mm, with a typical length of 8 cm. For more powerful engines, the casing

must be a hard plastic. The rocket propellant (3) used for smaller motors (up to E) is commonly black powder, more powerful motors use composite propellants, such as ammonium perchlorate, potassium nitrate and aluminium powder. The delay charge (4) is a slow burning chemical that burns evenly and releases smoke, both for easier tracking and aesthetic reasons. The ejection charge (5) creates a gas explosion powerful enough to force the nose cone off and so deploying the recovery system. The end cap (6) is simply a piece of clay that pierces the outer shell and directs the ejection charge upwards. [15]

2.1.3 Recovery methods

Model rockets are usually made to be reusable, this requires that the rocket survives the descent from the apogee to the touchdown. The denominations for the most common recovery methods follow [15].

Featherweight and tumble

These methods are only suitable for the lightest rockets, because they rely on letting the rocket simply flutter downwards after ejection of the engine. The difference between featherweight and tumble lies on the fact that featherweight rockets are not modified and fall in ballistic trajectories, and tumble recovery is applied to slightly heavier rockets, where a ballistic trajectory is dangerous, and so the ejection charge is used to move the motor further backward without ejecting causing the CG to move behind the CP and the now unstable rocket to tumble down, hence the name.

Nose-blown, streamer and parachute

These methods use the ejection charge to expel the nose, leaving the rocket connected by a shock cord usually made of resistant rubber or Kevlar. Nose blown recovery relies only on the high drag of the rocket in this shape to slow down the landing, streamer recovery deploys wide and long strips of fabric (usually plastic) attached to the shock cord to generate drag by flutter and slow the touchdown, finally parachute recovery expels a parachute that fills with air and generates the most drag. Both streamers and parachutes require wadding to protect and hold them in place during the explosion of the ejection charge, this is normally made from any non-flammable fibbers.

Glide

These are usually remote controlled rockets that possess two large fins, and so can glide down and land as a model air plane.

Retrospin

In this case the ejection charge deploys a system by which a tab on each fin is deployed, causing the rocket to enter a spin that will slow it's decent, much like an autogyro.

2.1.4 Phases of Flight

Model rockets normally have three stages of flight, powered ascent, coasting flight and slow descent. These can be seen in fig. 2.4.

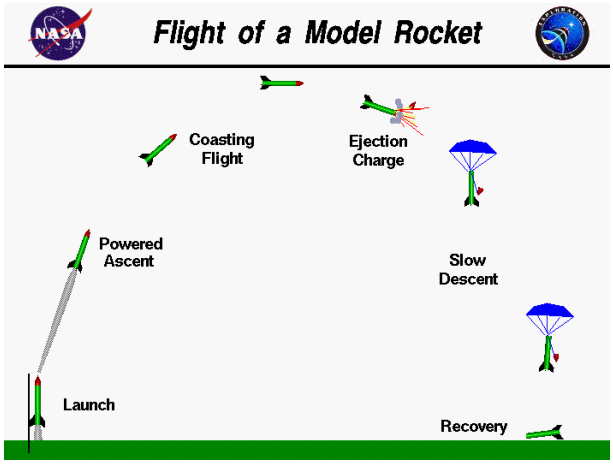


Figure 2.4: Typical model rocket flight [16]

Powered ascent

Before the proper powered ascent even begins, the launch must be given some consideration, full sized rockets can rely on engine gimbals for stability, that is a movable engine nozzle, model rockets however rely on aerodynamics, fins mostly. Since these are highly dependent on velocity, during the first few seconds of ignition (where velocity is close to zero) the rocket must be stabilized by a launch rod, it also guarantees that the direction of the thrust is vertical during launch, thus avoiding a stray rocket [15] [14].

The main characteristic of the powered flight phase is that the dominant force is thrust, which is an order of magnitude above any other force acting during this stage applying a large acceleration. For most model rockets this phase is relatively short, most motors burn only for a few seconds, it is during this time however that the rocket experience the most acceleration, reaching its top speed at the end of the powered flight. [15] [14]

Coasting flight

From this point forward only three forces act on the rocket, lift, drag and gravity, thus it begins to slow down. During this stage the delayed charge of the motor is slowly burning. All the kinetic energy here is either being transformed in potential energy resulting in height gain or being dissipated by drag forces, it is during this stage when the rocket would preferably reach its apogee, though some charge delays will not provide enough time [15].

Slow descent

After the delay charge reaches the end it explodes and ejects the parachute and the wadding, if it deploys correctly, from this point on the rocket experiences a dramatic increase in drag. It will now slowly fall to the ground, during this time the main care is to track the rocket and hope it does not land in an inaccessible area [15].

2.2 Parameters to Measure

The main goal when designing an instrumentation system is to measure a set of parameters and their evolution in time [20]. Therefore, the first decision to be made is which physical quantities to measure and the expected range. A summary of relevant information would be table 2.2.

Table 2.2: Parameters to Measure [3, 21]

Parameters	Proj. Range	Test Range	Accuracy
Time	Launch time +15 min	Launch time +2 min	0.001 s
Horizontal Position (Lat./Lon.)	Ø 400 m from launch	Ø 100 m from launch	Ø 3 m
Static Pressure	30/110 kPa	90/110 kPa	0.1 kPa
Altitude (GNSS, barometric)	At launch + 10km	At launch 300 m	3 m
Linear Velocity, (N,E,D).	$\pm$ 300 m s^{-1}	$\pm$ 50 m s^{-1}	3 m s^{-1}
Linear Acceleration, 3 ax.	$\pm$ 15 g	$\pm$ 9 g	0.05 g
Angular Velocity, 3 ax.	$\pm$ 360 °s^{-1}	$\pm$ 360 °s^{-1}	1 °s^{-1}
Attitude (Pitch,Roll,Yaw)	full range	full range	3 °
Static Temperature	-20/50 °C	0/40 °C	1 °C

2.2.1 Time

Time is likely the most important physical quantity to measure, without it there is no reference frame, and the rest of the data cannot be adequately analysed and compared to external data, moreover without a adequate time frame it becomes impossible to obtain related properties through computational methods, e.g. integrating acceleration to obtain velocity of differentiating position to do the same.

Time measurements in this project will be secured by two different yet complementary sources, GNSS time and the CPU internal clock. The GNSS time will provide the date and time accurate to the 100 ns [22], this however is limited by the frequency at which the receiver can be queried, in between GNSS data time will be marked by the internal clock of the CPU, with the provided precision, which will likely only need to be of a millisecond. The expected time frame will be the date and time at the time of launch and no more than 2 minutes after, the more precise estimate depends on the calculated time of flight, as for larger rockets, 15 min would be a conservative estimate.

2.2.2 Horizontal Position

Position is another key player in obtaining any rigid body motion information. With accurate and frequent enough position updates a solid estimate of velocity can be obtained.

Position is normally defined in Cartesian coordinates, and consists of three components, one for each dimension. Here however position is being divided into two different sections, the current one referring to planar position, in spherical coordinates, and altitude. The choice for obtaining the current physical property was a GNSS receiver, which outputs latitude, longitude and altitude.

The expected range in this subset of position is preferably not very high, the flight path of the rocket as explained in Section 2.1, will preferably be mostly vertical, in a small parabola, so it should not exceed a radius of 50 meters around the launch site, the more accurate range will be obtained after all information on rocket, payload and motors is known. In any case this will not present a problem as Global Positioning System (GPS) (the chosen GNSS) has nearly global coverage, including the totality of Portuguese territory [23]. The update rate and precision will again be tied to the GPS receiver, if no differential correction is used an error of more than 3 meters is to be expected [23], the update rate will be of around 10 Hz, which is what can be expected from most small and light GPS receivers, and should be enough to accurately determine trajectory. As for maximum expected ranges, larger rockets may drift up to 400 m before landing [24].

2.2.3 Altitude

Altitude is the most important measurement of position for the instrumentation system, the dominant direction in rocket flight is the vertical one. Measurements of altitude will be obtained by two methods, GPS receiver, and barometric altitude.

For the test rocket the altitude range should not exceed 300 m above launch height, the update rates and expected precision for barometric and GNSS altitudes are significantly different. The range on larger rockets may go to several kilometres, the record being 115 km [21].

Barometric altitude has traditionally been used in flight since the inception of altitude measurement. It is very dependent on meteorological phenomena. Calculating altitude relative to sea level by such method is likely unacceptably erroneous but variations in altitude can be measure by this method. Atmospheric pressure can be expected to vary linearly in our case, it can still experience vast changes not due to the altitude [25].

The update rate is of around 10 Hz, enough to adequately determine trajectory, and sufficient to be able to calculate vertical velocity if needed.

GPS altitude can be accurate to roughly 3 meters which can be improved by differential correction, and can be depended upon to calculate absolute altitude, yet again the update rate limitations are apparent as 10 Hz from a small module is the maximum normally available [26]. It must be noted that commercial GPS in the US are limited to 18 km altitude [23].

2.2.4 Linear Velocity, 3 axes

Linear velocity in aerodynamics is always an important factor, this velocity however is normally calculated in relative wind speed for airplanes (since they depend on this quantity for lift) [20], given that a static and dynamic pressure reading is out of the question for such a small device (at least for its first iteration) the system must rely on GPS velocity, which presents another problem since most small GPS receivers only provide a scalar velocity and a ground track.

The dominating component of velocity will be the vertical one, yet all will be measured. The top velocity for our test would likely be around the 40 $\mathrm{m\,s^{-1}}$, faster rockets may reach near supersonic speeds [15] [3].

As with position (planar and altitude), the velocity will be determined at least 10 Hz to obtain useful trajectory data. GNSS receivers will provide this data on the world reference frame (North, East, Down). US GPS receivers are limited to a maximum velocity of 1000 kt.

2.2.5 Linear Acceleration, 3 axes

Acceleration is one of the main measurement to be made to successfully understand the dynamics of a particular motion, it allows some insight into the forces in play, and it also presents an important gateway into attitude information from an inertial standpoint [20].

Acceleration is measured by three accelerometers, one for each axis. It can be expected that most acceleration will occur in the axial direction of the rocket, the other values will likely be due to wind, vibration and the slight parabola.

Observing the data in Section 2.1, it can be expected that acceleration peaks will be in two moments, launch and parachute opening, in both cases the expected acceleration will be of around 8-9 g, about 90 $\mathrm{m\,s^{-2}}$, thus the range will be of $\pm$9 g, that is approximately $\pm$ 50 $\mathrm{m\,s^{-2}}$.[15]

The update rate will be again as high as possible, higher than 10 Hz ideally, close to 25 Hz for an Attitude and Heading Reference System (AHRS) [27]. As for the precision required anything above 0.05 g would suffice. Larger rockets experience more acceleration which could reach an excess of 10 g. The reference frame for acceleration output will be the body frame.

2.2.6 Attitude

Attitude is the orientation of a rigid body. In a rocket, this parameter proves to be particularly important when considering the relative position of the rocket with regards to its velocity vector, to determine several aerodynamic properties [20].

Attitude will be measured mainly through a combination of magnetometers, three on a per axis basis, accelerometers, again three, and the integration of the angular velocity data from the three gyroscopes.

Attitude is relative to a frame of reference, and the actual data range can encompass any position. The precision would be acceptable if in the range of around 2-3°. The actual range and precision to be monitored however would be the magnetometers sensitivity and saturation, as well as the influence of internal magnetic fields, without further analysis it is hard to pinpoint a range and accuracy, though for all commercially available equipment the range far exceeds normal magnetic fields, and the accuracy is acceptable, especially considering the correction by gyroscope integration and acceleration determination.

As for sample rate, attitude should be of at least 25Hz to obtain adequate orientation data for the AHRS, including rapid variations.

Magnetic Data, 3 axes

Magnetic data will be used for North/South orientation, and the three-dimensional value will be recorded, though due to the difficulty in calibrating magnetometers in such magnetically dense situations only a normalized value will likely be useful and no attempt will be made to obtain precise magnetic readings, that being said, with some data treatment this could be done after the fact. An update rate of more than 10 Hz would be preferred, 25 Hz for AHRS. The reference frame for magnetometer output will be the body frame.

2.2.7 Angular Velocity

Another important physical quantity to measure is angular velocity, not only does it provide the rate of rotation for each axis, but it is the main source of information regarding moments applied to the rocket, which can be critical when assessing stability. It will also provide one of the main components of an AHRS.[20]

Angular velocity is measured by a set of three gyroscopes, one per axis. It can be expected that the main component of angular velocity will be rotation about the axial direction of the rocket, as a stabilizing spin is to be expected.

Estimating the rate of rotation of the rocket can be a tricky calculation, however this rotation will likely not be very fast, no more than 360 degrees per second certainly, the same would apply to larger rockets. A precision of around a $1°s^{-1}$ would be acceptable, with an update rate of preferably more than 25 Hz for AHRS. The reference frame for angular velocity output will be the body frame.

2.2.8 Static Pressure

Pressure is an additional physical quantity that provides information which may be useful for analysis of the atmosphere or other calculations, it mostly figures on the list because this information is already obtained when calculating the barometric altitude.

Static pressure is preferred, but the actual type pressure measured depends on the placement of the sensor, if there is flow on the sensor the measure pressure will no longer be static. Measuring dynamic pressures impedes or seriously hampers the determination of altitude and atmospheric conditions.

Pressure is measured by a barometer and will likely range from 110 - 90 kPa, a very conservative estimate that ranges from 1 km below sea level and 1 km above in a standard atmosphere [25], barometers normally have much wider ranges. As for sample rate, the highest possible, 5 Hz at minimum, though what truly defines the sample rate is the necessity of altitude information, for which 10 Hz would be preferable if the trajectory data is to be usable in rocket analysis. Larger rockets could go as low as 30 kPa depending on altitude.

2.2.9 Static Temperature

Temperature again provides interesting atmospheric information, and can be used to more accurately estimate altitude based on pressure. It may be interesting reading both for atmospheric and instrument temperature for higher rockets, where the it may be as low as $-20°C$ [25].

Static temperature is desired, preferably ignoring changes by greenhouse or joule effects, to acquire atmospheric data, however instrument temperature may also be interesting. The determined temperature (static or not) depends on final assembly.

For our case, it is featured merely because most barometric altimeters already read it. A 5 Hz sample rate should suffice for atmospheric analysis. As for precision, anything in the range of the $1°C$ would be acceptable.

Chapter 3

Acquired and developed components

To fulfil the aforementioned goals a series of sensors and electronic equipment are required as well as a model rocket to flight test the instrumentation. In addition to specific requirements, particular to each sensor, the general objective is to have a light and small instrumentation system, which is made from easily replaceable COTS parts, the system also needs to survive the rocket environment. The first two requirements (common in any airborne activities) insure that the instruments can be used in a compact rocket with a low power motor (or indeed any other system with similar or laxer restrictions), the third assures that should anything go awry the instruments can easily be replaced.

3.1 Microprocessor/Base Platform

The microprocessor is the brain of the system; it assures the correct functioning of every part of the instrumentation system. To receive data from multiple sensors, record it and transmit it wirelessly, a microprocessor must be used, in lieu of a simpler short circuited sensor-emitter system. The base platform chosen used was the Arduino Fio is shown in fig. 3.1.

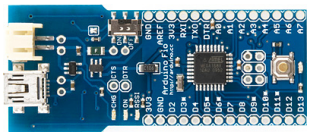


Figure 3.1: Arduino Fio [28]

A custom-built PCB board would provide some benefits, being small, light and resistant, before building one though a prototype should be tested. The best (and more commonly available) system for electronic prototyping is an Arduino platform. A custom board prevents replacement of components, using breakouts allows for the substitution of specific sensors and not an entire board if there is a problem with a specific breakout or the need to upgrade a sensor.

An Arduino is an open source micro-controller board that provides a simple way to interface with sensors and actuators. It guarantees compatibility and community support while trading off compactness and specificity.

Using Arduino comes with several advantages, tried and tested software, components guaranteed to be compatible, and above all, a vast on-line community where assistance and group problem-solving is available. Ease of access is also a major point in favour of Arduino since many on-line retailers offer board and components, including some national distributors in Portugal. It also offers great versatility as there are many boards and an expansive ecosystem of breakouts.

Table 3.1: Arduino Comparison [28]

Model	Fio	Micro	Mini	Nano	Pro	Pro Mini
Processor	ATmega328P	ATmega32U4	ATmega328P	ATmega328P	ATmega168	ATmega328P
Operating/In Tension	3.3 V / 3.7-7 V	5 V / 7-12 V	5 V / 7-9 V	5 V / 7-9 V	3.3 V / 3.35-12 V	3.3 V / 3.35-12 V
CPU Speed	8 MHz	16 MHz	16 MHz	16 MHz	8 MHz	8 MHz
Analog I/O	8/0	12/0	8/0	8/0	6/0	6/0
Digital IO/PWM	14/6	20/7	14/6	14/6	14/6	14/6
SRAM	2 KB	2.5 KB	2 KB	2 KB	1 KB	1 KB
Flash	32 KB	32 KB	32 KB	32 KB	16 KB	16 KB
Dimensions	28x65 mm	18x48 mm	18x30 mm	18x45 mm	52x53 mm	18x33 mm
Mass	9 g	13 g	-	5 g	10 g	2 g
XBee slot	Yes	No	No	No	No	No
LiPoly In	Yes	No	No	No	Yes	No

Out of the many Arduino boards available, given our main goals of a light, small and yet versatile platform, a few options present themselves, which can be seen in table 3.1.

There are several criteria for choosing the board, those based on physical properties (mass and dimension), those based on computational capacity (SRAM and processor speed), energetic considerations (operating and input tension) and Input/Output protocols and available ports.

Physical properties are a defining characteristic, extremely important for this design, however all the boards presented in table 3.1 were chosen because their mass and dimensions conformed to the expected requirements, with possible exception of the Pro for smaller rockets.

Some criteria are rendered mostly neutral, particularly connection options. As will be seen in sections 3.2 and 3.5, the required connections are Inter-Integrated Circuit (I^2C) protocol for the Altimu-10, Universal Asynchronous Receiver Transmitter (UART) for the uBlox Max-M8Q GNSS receiver and Serial Peripheral Interface Bus (SPI) for the Adafruit SD, all boards feature these connections. The only connection that proves differentiating is the XBee S2B, which utilizes Serial communication, but for that connection it requires a shield or an independent breakout, which adds weight and occupies space, in this regard the Arduino FIO takes the lead since it features a dedicated XBee slot.

SRAM and Flash, also called Programem are also important factors, SRAM mostly, given the large number of variables that must be handled while running the program. This all but excludes both the Pro and the Pro Mini from consideration, since they feature a much smaller SRAM, barely enough for the required sensor libraries.

As can be seen in the table, all Arduinos feature Atmel processors, particularly, most platforms on the list feature the ATmega328P. As shown in [29] this processor's maximum clock speed is directly connected to the required operating tension, the clock speed can only reach 8 MHz at 3.3 V, and 16 MHz at 5 V. For now, it can be stated that a higher clock speed is always preferable, not only because computations perform faster but also because effective baud rate is determined by subdivisions of the clock [30, 31], and since the internal clock of the XBee is a 16 MHz clock it would be beneficial to have a matching one in the Arduino when working with both, yet this also leads to an increased power consumption.

Finally, tanking in to account the decision taken on section 3.6 about batteries, the choice of Lithium Polymer cell which functions at 3.7 V forces us to either use a step up circuit, which would add weight and increase power losses requiring a larger battery, or to choose a 3.3 V platform, which limits processor speed to 8 MHz, but also lowers power drain. This decision would tend mostly towards the Fio, which also conveniently provides a charging circuit for this types of battery.

Taking all stated above it is clear that the best fit for the project is the Arduino Fio, which can be seen in fig. 3.1.

3.2 Remote Communication

Real time ground tracking enables us to perform preliminary data analysis and obtain a closer estimate of the rocket's landing spot, but to do so a transmission unit of some sort must be used. Since this project revolved around availability, and using "off the shelf" components a XBee radio unit seemed the perfect fit. The used modules are a pair XBee Pro S2B seen in figure fig. 3.2.

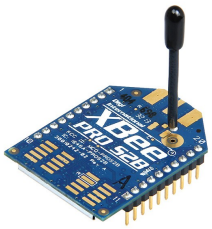


Figure 3.2: XBee S2B [30]

By using of the shelve components the project is somewhat limited in versatility, this however is negligible in face of the assurance given by using tried and tested components, which cuts on research and development (R&D) cost and time, and particularly in the ranged communication department where quality construction is paramount for good efficiency in transmission. That being said, the XBee radio allows for significant customization, not only in transmission rates and protocols, but also in antennae, thus allowing for ample room to increase the heights and distances travelled

Table 3.2: XBee Specifications [30]

Specification	S2B
Indoor Range	60 m
LOS Range	1500 m
Transmission Power	10 mW(+10 dBm)
Data Throughput	35000 $\mathrm{bit\,s^{-1}}$
Serial Interface Data Rate	1200 - 10^6 $\mathrm{bit\,s^{-1}}$
Receiver Sensitivity	-102 dBm
Sully Voltage	2.7 - 3.6 V
Operating Current (Transmit)	117 mA
Operating Current (Receive)	47 mA
Operating Frequency	ISM 2.4 GHz
Operating Temperature	-40 to 85 °C

The XBee being used is an S2B radio, with a standard antenna, the main specifications are in table 3.2. Two characteristics will be particularly significant, Line of Sight (LOS) range and the operating current in transmission (the latter value will be utilized in section 3.6).

As for the LOS range, the value provided by the manufacturer is 1500 m, which will be more than sufficient for most rockets, and larger, higher reaching ones could be outfitted with an active antenna supported by the XBee. In order to confirm the provided value a range test was performed.

3.2.1 Range Test

Requirements

The range test was performed by using two XBee S2B, one connected to a PC with a XBee Explorer and one placed on a XBee shield fitted to an Arduino Uno powered by a portable power bank. A mobile phone was used to maintain contact between the two operators. The XBee setup can be found in section 3.2.2, with the only difference being the baud rate, which was 9600.

A serial reading program must be used to interface with the static XBee, i.e. listen to its serial port, the preferred one was XCTU, provided freely by the manufacturer.

The Arduino Uno had a simple counter programmed into it, which would print a number every second, this print translates to a serial Tx output that was broadcaster by the XBee

Two people were required for the test one to move the Arduino and one to check the output. A field of view of 1500 m was an obvious requirement.

The conditions for passing the test was the systems being able to maintain a fluid connection at the longest distance, and the ability to quickly recover the connection when a short interruption affected the system.

Procedure

A static base station is set up; this position marks the origin. The person with the Arduino first connects the system and both verify the correct and fluid transmission of data.

The person with the Arduino begins moving away, stopping at known distances, previously decided or marked in the moment by GPS. Disconnects at certain distances provide an opportunity to evaluate reconnect capabilities.

The responsible for monitoring the PC must pay attention to the number progression, as well as the frequency of arrival, to check if there are any connection problems.

In our test the chosen location was the Cruz-Quebrada shoreline, in Lisbon Portugal, mostly due to proximity, no automotive traffic and relatively limited pedestrian use. This are and the station positions can be observed in fig. 3.3.

Figure 3.3: Range Test Location

Results

The test was a success, the XBee remained connected until the furthest point available for testing, although it failed when tested in point 8, but this was 2.1 km away, beyond the expected range. There was a signal loss after point 7, due to the edification present, as it was impossible to maintain LOS. At point 5 there was again a brief disconnect again due to the loss of direct sight of the origin.

The periodical disconnect showed a small delay when reconnecting. Short interruptions in the connection due to obstacles (e.g. large groups of people) resulted in large transmission in lump of the buffer, with usually no loss of information, though that depended on the distance to the source and the duration of interruption, i.e. whether there was saturation of the 100 Byte buffer. Bigger cut outs (a small concave area around point 5) resulted in the saturation of the buffer and the loss of some characters.

3.2.2 XBee Setup

Table 3.3: XBee Setup

Setting	Receiver	Emitter
Firmware	ZigBee Coordinator AT	ZigBee Router AT
PAN ID	55	55
Destination Address High	13A200	13A200
Destination Address Low	40B0EA43	40B4E295
Baud Rate	57600	57600
Parity	No	No
Stop Bit	1	1
DIO 7 Configuration	CTS flow control	CTS flow control
Power level	Highest	Highest
Power Mode	Boost Mode Enabled	Boost Mode Enabled

The setup was based on [31], the main settings are on table 3.3. The Pan ID is optional since this is a point to point network not a mesh, the Destination Address Low is the Serial Address specific to each radio, being that the destination for the receiver is the serial of the emitter and vice versa. Another possible firmware choice would be the Emitter as an End-Point AT.

3.3 Data Recording

Recording data locally provides several advantages, normally it is the fastest and more reliable way of storing information and assures some redundancy when coupled with telemetry.

For our current design, and given our precepts of small, light and readily available, the most obvious solution is a microSD card. There are many other technologies but few that are smaller and lighter, and none are nearly as widespread as microSD cards. The added bonus in our case is that most Arduino users came to the same conclusion so there are many modules available that connect to microSD cards, as well as function libraries to connect to them.

The micro SD cards are connected to the Arduino using a SPI, it consists of four pins Serial Clock (SCLK) (output from master), Master Output, Slave Input (MOSI) (output from master), Master Input, Slave Output (MISO) (output from slave) and Slave Select (SS) (active low, output from master) [32]. Which correspond respectively to digital pins 13, 12, 11 and a selectable one, in our case 10, on the Arduino Fio[28].

The chosen microSD breakout was the one from Adafruit pictured in fig. 3.4, mainly due to the trust deposited in the manufacturer, and the ability to power it both from 3.3 V and 5 V sources.

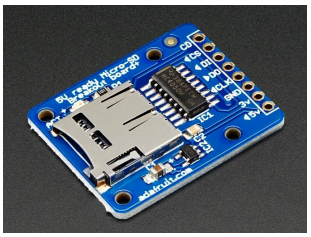


Figure 3.4: SD Card Reader [33]

The most important factor when using microSD cards is not the breakout board but the card itself. This can be a problem as microSD cards have vastly different characteristics depending on maker and model, this not only refers to expected and documented characteristics, e.g. as read/write rates and capacity, which can be previously consulted, but also to more normally unimportant features as power consumption and write block times, that given our constraints might lead to some problems.

The SD current input is not only dependent on the card manufacturer but also on whether there is a write, which means it works in peaks. The writes themselves only occur when the buffer is full, and the current draw is usually related to the size of the buffer, since the buffer size is often related to the SD card size it seems most prudent to acquire a small size microSD card. On this fact most users agree [34], however not all cards are either documented or have been tested, as such there is no guarantee on power demand, it can be safely stated however that it should not exceed 150 mA.

The card selected is a Lexar 8 GB SDHC, chosen because it was the smallest available at a local retailer.

Another fact to consider when using microSD cards is the libraries available for use, the Arduino come with a default library available the SD.h library, which functions appropriately yet uses a large amount of SRAM, and does not achieve very good write speeds. This is likely due to it being a wrapper around an old fat16 library, and so operates on older slower protocols. For those reasons the library used is a new library called SdFat.h [35] created by the original creator of the fat 16 library, which was found to achieve greater write speeds while utilizing a smaller amount of memory.

3.4 Inertial, magnetic and atmospheric data

All this functions are in one category since the chosen board, Pololu Altimu-10 v4 seen in fig. 3.5, compounds the sensors responsible for gathering the said data onto one sole breakout. This guarantees optimal miniaturization, and since the board is fairly common it is in agreement with the smaller, lighter and readily available design principle.

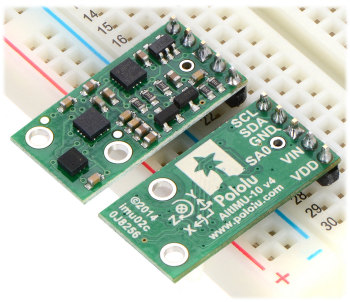


Figure 3.5: Pololu Altimu-10 [36]

The Altimu10 has three gyroscopes, three accelerometers, three magnetometers, all in a per axis

basis, and a pressure and temperature sensor. The digital barometer and thermometer corresponds to the LP25H, the gyroscopes to the L3GD20H and the accelerometers and magnetometers to the LSM303D.

The accelerometer and gyroscopes, are responsible for inertial data, registering all the forces and accelerations experienced by the rocket during flight. Inertial data can also be combined with magnetic data to provide spatial orientation. Several tests were performed to insure the specifications met the requirements imposed, set in section 2.2, this tests are described in section 5.2, they are obviously dependent on the AHRS algorithm used.

The pressure and temperature sensor can provide atmospheric data, which can be useful for a number of reasons, if correctly calibrated it can also be used to calculate altitudes.

More information on these sensors is represented in table 3.4.

Table 3.4: Altimu Characteristics [36–39]

Characteristic	Altimu-10 v4
Dimensions	25×13×3 mm
Mass	2 g
Supply Current	6 mA
Operating Voltage	2.5 - 5.5 V
I^2C Output	
Gyroscope	16 bit per axis
Accelerometer	16 bit per axis
Magnetometer	16 bit per axis
Barometer	24 bit
Sensitivity Range	
Gyroscope	±245, ±500 ± 2000 $°s^{-1}$
Accelerometer	±2, ±4, ±6, ±8 ±16 g
Magnetometer	±2, ±4, ±8, ±12 G
Barometer	260-1260 mbar

The Altimu-10 communicates with the Arduino using I^2C interface, which means all sensors share the same two pins, the Serial Data (SDA) for data and Serial Clock (SCL) for the clock, respectively pins A4 and A5 in the Fio, each sensor is then distinguished by their address. This means that although every sensor is located on the same board they are effectively treated as individual sensors by the software. [40]

3.4.1 Output considerations

The 16-bit output from the accelerometer is left aligned on the register and so the first four bit are always zero, as such the values should be right shifted 4 bits (divided by 16). It's worth noting also that "g" is not a SI unit, but it's assumed that the g used corresponds to the standard g defined in [41] as 9.80665 $m\,s^{-2}$.

As for output ranges, in coherence with section 2.2, the output for the accelerometer will be ± 16 g, the gyroscope ± 2000 $°s^{-1}$, the barometer will feature in its only range and the magnetometer will function in the default ± 4 G unless it saturates.

The magnetometer must be calibrated, to avoid bias. There is an Arduino sketch included in the library for this use, while running the software the sensor must be fully rotated about all three axes.

To interpret the data into attitude some software must be used, IMU or preferably MARG software will interpret the data, for example [42] here used or [27, 43, 44] , there are several other algorithms that can be used or even new ones that can be developed.

The axis definition is precisely the opposite of those displayed on the Pololu board fig. 3.5, as the pins were soldered towards the flat end of the board (like the board on the left of the figure), still in the AHRS program the axis are coherent with the standard Euler angles, that is positive pitch nose up, positive roll right wing down and positive yaw clockwise.

3.5 GNSS receiver: Time, Position and Velocity

Tracking the velocity and position of the rocket is essential to study its trajectory and behaviour Currently the best way to track both is using a GNSS receiver. These are standard in many applications, including in many Arduino uses, which guarantees good compatibility. If possible, and to increase the quality of the data, differential correction will be used. In our case the GNSS used will be GPS. The module chosen was a u-blox MAX-M8Q breakout with Maruwa SL1252R antenna with SAW/LNA from Uputronics seen in fig. 3.6.

Figure 3.6: u-blox MAX-M8Q breakout with Maruwa SL1252R antenna with SAW/LNA [45]

GNSS receivers are amongst the larger and heavier components, yet they are still small and light enough to be practical, particularly considering the important information they acquire.

The only foreseeable limitations are with update rates, since GNSS models are normally capped at 10Hz, and with sentence parsing since GPS are normally connected with UART. More detail on the sentences received and the manner of their reception can be found in section 3.5.1 and chapter 4.

More information on the Adafruit Ultimate GPS (the first tested option) and the uBlox module can be observed in table 3.5.

3.5.1 Communication Protocol

GNSS receivers output predefined sentences, most of these follow the National Marine Electronics Association (NMEA) standards, called NMEA sentences, although there are some proprietary strings such as the uBlox (UBX) protocol used by our chosen module. The relevant NMEA sentences equated and the Position, Velocity and Time (PVT) UBX sentence used can be seen in table 3.6.

Using a UART connection GNSS sentences are transmitted in continuous form, which generates problems when receiving data from other sensors simultaneously. Meaning that while the GPS sentence is being received not much other processing can be done. One method to overcome this problem is using an interrupt routine connected to the Rx UART port.

After receiving the sentence, it must be parsed, it is in this area where the UBX protocol shines, since it uses a byte based structure (sending bytes) unlike NMEA which uses a character based one (sending characters). The UBX Protocol use of byte structures i.e. four bytes per 32-bit integer, and not

Table 3.5: GPS Options [26, 45–47]

Characteristic	Ultimate GPS	M8Q
Dimensions	25.5×35×6.5 mm	30×22×1.6 mm
Antenna	15×15×4 mm	Ø13 ×20 mm
Mass	8.5 g	10 g
Supply Current	25 mA (track) 20mA (nav.)	22 mA (track) 5mA (nav.)
Input Voltage	3 - 5.5 V	3 V
Satellites	22 (track) 66 (search)	56 Channels
Acquisition Sensitivity	-145 dBm	(-)
Tracking Sensitivity	-165 dBm	(-)
Start time	34 s	20 s
Max Update rate	10 Hz (5 Hz pos.)	10 Hz
Position accuracy	<3 m	<3 m
Velocity accuracy	0.1 m/s	0.1 m/s
Protocol	NMEA 0183	NMEA and UBX
Output	UART	UART, SPI and TWI

Table 3.6: NMEA Sentences and UBX PVT [26, 48]

NMEA Sentence	Time	Lat.& Long.	Altitude	Velocity	Example
GGA	Yes	Yes	Yes	No	$GPGGA,123519,4807.038,N,01131.000,E, 1,08,0.9,545.4,M,46.9,M,,*47
RMC	Yes	Yes	No	Yes	$GPRMC,123519,A,4807.038,N,01131.000,E, 022.4,084.4,230394,003.1,W*6A
GLL	No	Yes	No	No	$GPGLL,4916.45,N,12311.12,W,225444,A,*1D
VTG	No	No	No	Yes	$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K*48
UBX PVT Sentence	Yes	Yes	Yes	Yes	0xB5 0x62 (...) CksmA CksmB (100 bytes total)

in characters i.e. 1 byte per character is more economical, and has the added advantage of the sentence having always the same length (100 B), which also facilitates checksum calculation and parsing.

The format of the PVT sentence and relevant data is in table 3.7 [26]. Multi-byte data is sent in Little Endian (which is useful for parsing as the Arduino memory uses the same Endian). The checksum is calculated from the Class field (included) to the end of the Payload using an 8-bit Fletcher Algorithm.

3.5.2 GNSS Setup

The settings chosen for the GNSS receiver are displayed in table 3.8, the update rate is the maximum possible, so is the baud rate, (which is required by the connection, more in section 3.9). The only setting not included in the table is the dynamic platform model, the relevant ones are shown in table 3.9, the mode chosen will be dependent on testing done in chapter 5. If deciding based only on the dynamics of the system, the single possible mode would be Airborne <4g but as these accelerations occur briefly it may be best to sacrifice precision in these short seconds for more accuracy at other times.

To setup the GPS it must be connected to the uCenter PC program and have a CR1216 coin cell connected.

The setup is made through the UBX Configuration (CFG) class sentences, namely: RATE (for the update rate), PRT (for the port and baud rate), MSG (for the output sentence selection), NAV5 (for the Dynamic Platform Models) and CFG (to save the configuration to the Battery Backup RAM (BBR)).

Table 3.7: UBX PVT structure, U/I=unsigned/signed followed by the number of bytes [26]

Structure		Format	Unit	Description
Header	0	0xB5	-	First sync character
	1	0x62	-	Second sync character
Class	0	0x01	-	Sentence class, NAV
ID	0	0x07	-	Sentence ID, PVT
Length	0-1	92	byte	Length of payload
Payload	6	U1	month	Month
	7	U1	day	Day
	8	U1	hour	Hour
	9	U1	min	Minute
	10	U1	s	Second
	16-19	I4	ns	Fraction of second
	20	U1	-	3: 3D fix
	23	U1	-	Number of satellites used in solution
	24-27	I4	1e-7 deg	Longitude
	28-31	I4	1e-7 deg	Latitude
	32-35	I4	mm	Height
	48-51	I4	$\mathrm{mm\,s^{-1}}$	NED north velocity
	52-55	I4	$\mathrm{mm\,s^{-1}}$	NED east velocity
	56-59	I4	$\mathrm{mm\,s^{-1}}$	NED down velocity
Checksum	0	CK_A	-	First byte of the checksum
	1	CK_B	-	Second byte of the checksum

Table 3.8: uBlox Configuration

Setting	
Update rate	10 Hz
Baud rate	57600
Active sentence	NAV PVT

Table 3.9: uBlox Dynamic Platform Models [26]

Platform	Max Alt. (m)	Max. hor. vel. (m/s)	Max. vert. vel. (m/s)	Sanity check type	Max. pos. dev.
Portable	12000	310	50	Alt. & Vel.	Medium
Airborne $<$1g	50000	100	100	Alt.	Large
Airborne $<$2g	50000	250	100	Alt.	Large
Airborne $<$4g	50000	500	100	Alt.	Large

3.6 Electrical Power

To provide energy to all systems a power source is require, given the nature of the project the obvious choice would be a chemical battery. Several pieces of information are required to decide which battery to utilize, the physical characteristics of the battery (its dimension and mass), the nominal voltage it provides, the maximum continuous draw current and the battery capacity. Another important fact is the temperature of operation, as in higher rocket flights temperatures can easily be below freezing. The chosen battery technology was Lithium Polymer batteries.

First, to have an estimate of the draw current, the requirements for all subsystems must be added. This data is displayed in table 3.10. It must be noted however that all these figures are for maximum current draw, so the actual value will likely be inferior to the requirement.

In the case of the Arduino it must be noted that the estimate was obtain in part by considering the

Table 3.10: Current Draw

Component	Fio	XBee	microSD	Altimu-10	UBlox Max-M8Q	Total
Current draw	65 mA	117 mA	150 mA	10 mA	22 mA	367 mA

processor and in part by measurements by the community.

The SD card reader is also to be observed, as the draw is mostly dependent on the card and the values vary greatly depending on the manufacturer, what's more, most manufacturers do not provide the value for the current requirements. It is also important to note that the current draw of the SD card is not flat, that is, once the buffer is full and a write must take place there is a peak current draw, that is the value displayed on the table. There is a direct correlation between the amount of drawn current and the size of the SD card, due to the larger buffer on larger cards, this means that since the program does not require a large amount of storage space it would be preferable to use a smaller card, the smallest one available locally was an 8 GB Lexar card. In any case the value used in table 3.10 is a majorant of the estimates.

Table 3.11: Battery Comparison [49]

Battery	Li-Poly	9V-Li (LA522)	Coin-cell (CR2032)
Volume	6 cm^3	21.4 cm^3	1 cm^3
Mass	11 g	33.9 g	3g
Nominal Voltage	3.7 V	9 V	3 V
Capacity	400 mA h	750 mA h	210 mA h
Continuous Discharge Rate	400 mA	1000 mA	0.2 mA
Operating temp.	-20/60 °C	-40/60 °C	-

Based on that information there were several options available, coin-cell stacks, 9V batteries and Lithium Polymer. All major data relevant to the decision is displayed in table 3.11.

The inclusion of the coin-cell stack was due to the lack of availability of the data regarding current draw at the time, since with that information it was abundantly obvious the coin-cells could not deliver the required power.

Regarding the 9 V battery, Lithium was the technology chosen given the higher energy density. During testing it was noted that the voltage rapidly decreased as the battery was drained. This, coupled with the conversion losses as Arduinos either work at 5 V or 3.3 V, resulted in poor efficiency, adding that most components operate at 3.3 V led us to choose a 3.3 V Arduino to improve efficiency. With this fact in mind, and considering the heavy weight led to the dismissal of this battery.

Lithium-Ion Polymer batteries (fig. 3.7) was the winning bid. LiPo or LiPoly as they are commonly known, is a battery technology that is commonplace, particularly in remote control aerial vehicles and wearable tech due to their high specific energy 100–265 $W\,h\,kg^{-1}$ and energy density 250–730 $W\,h\,l^{-1}$ [50]. One possible limitation is the output tension that would have to forcibly be 3.7 V, but since this tension also proves to be the one at which Arduinos are most energy efficient this would already be the favoured voltage. One disadvantage of using this type of battery is the requirement of a charge circuit, since these batteries are volatile and can explode[50].

Li Poly batteries come in a wide array of sizes, capacities, and draw currents, the chosen one was the smallest and lightest that would fit our needs. Given the relatively short time of flight of most model rockets, the capacity is not much of an issue, with that said the discharge current in usually decided based on the capacity, taking the information from table 3.10 a choice can now be made by selecting the closest battery that matches our current needs, that would be a 400 mA h battery with a 1C discharge

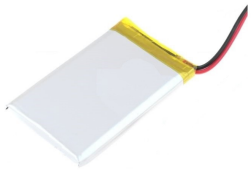


Figure 3.7: Lithium Polymer Battery [49]

rate (400 mA). The current budget can now be calculated, table 3.12.

Table 3.12: Current Budget [49]

	Current [mA]	
Total Component	367	
Battery Supply	400	
Margin	33	8.25%

3.7 Rocket, engine and platform

After the battery selection, an estimate of total mass of the instrumentation system can be made, which is shown in table 3.13. Based on this estimate and on the approximate knowledge of the total instrumentation volume the next step is to choose a model rocket, another option would be to build one, but this would require several more design decisions, and would defeat the objective of a general-purpose instrumentation system.

Table 3.13: Instrument Mass [28, 30, 36, 45, 49]

Component	Fio	XBee	microSD	Altimu-10	GPS	Battery	Total
Mass	9 g	10 g	5 g	2 g	10 g	11 g	50 g

3.7.1 Rocket

Given the number of options in the marked the first limitation auto-imposed was to select a ESTES rockets, since these were the most common and readily available, and the manufacturer was one with renown being one of the largest, if not the largest, on the market.

Another restriction imposed was the most powerful engine that could be bought without the need for official certification, this information was discussed in chapter 2, the restriction is not a legal one in Portugal but supplier imposed, in any case the propeller must be limited to D category engines.

Given the restrictions above, a few options presented themselves, which are displayed in table 3.14.

After careful analysis, the perfected option was the ESTES Maxi Alpha 3, seen in fig. 3.8. First its diameter was one of the widest, at 66 mm, guaranteeing adequate space for the instrumentation system, it was one of the longest at 84.5 cm, which (as seen in chapter 2) provides some assurances regarding stability, and its weight was not excessive, at 187 g, though it was one of the heaviest.

Table 3.14: Rocket Comparison [18]

Rocket	Maxi Alpha 3	Red Flare	Firebolt	Top Shot	Big Daddy
Mass	187 g	184.3 g	192.8 g	150.3 g	150.3 g
Diameter	66 mm	66 mm	42 mm	42 mm	76 mm
Length	84.5 cm	62 cm	92.7 cm	94 cm	48.3 cm
Engines	D12-3 - E9-6	D12-5 - E9-6	D12-5 - E12-6	D12-5 - E12-6	C11-3 - E9-6

Figure 3.8: ESTES Maxi Alpha 3 [18]

The platform used for launch will be the one produced by Estes, the Porta Pad II, which is compatible with the selected rocket, indeed there is a practical set which includes both.

3.7.2 Engine

Having chosen the rocket, it must now be decided which motor will be used to power it. The basis for decision is having a sufficiently powerful engine to lift the whole assembly. There again reigns the self-imposed limitation of ESTES engines, for the same aforementioned reasons, though given the standard denomination of engines makes this less of a factor in the present case. A few options are showcased in table 3.15.

Table 3.15: Rocket Engine Comparison, [18]

Engines	C11-3	D12-3	D12-5	E9-4
Total Impulse (N s)	10	10	20	30
Charge Delay (s)	3	3	5	4
Max. Lift Mass (g)	170	396	283	425
Max. Thrust (N)	22.1	32.9	32.9	25
Thrust Duration (s)	0.8	1.6	1.6	2.8
Initial Mass (g)	33.3	42.2	43.1	56.7
Propeller Mass (g)	11	24.93	24.93	35.80

Let us briefly restate the engine denominations previously addressed in chapter 2. The first letter represents the total impulse range (commonly measured in newton-seconds (N s), with each letter representing twice the impulse of the one preceding it. The first number refers to the average thrust (measured in newton (N)), the last number represents the delay, in seconds (s), between the end of thrust and the ignition of the ejection charge.

As can be seen the best fit for our needs is the D12-3 engine, although the D12-5 would likely provide a higher peak altitude, the D12-3 guarantees more of a margin in Lift Mass. this can be observed in the mass budget displayed in table 3.16.

Table 3.16: Mass Budget

Component	Mass [g]	
Rocket	187	
Engine	42.2	
Payload	50	
Total	280	
Max. Lift	396	
Margin	116	30%

3.8 Other equipment (Ground Segment)

Beside the instruments and the rocket other equipment required includes a laptop computer, to use in the field, a Future Technology Devices International (FTDI) cable to communicate and program the Arduino and the GPS, and a Universal Serial Bus (USB) XBee Explorer to connect the computer to the receiver XBee and a microSD card reader.

3.9 Assembly

The assembly of the instruments can be done in several different manners; larger rockets often have dedicated equipment bays that can and should be used when possible; smaller rockets may require the nose to be altered so it can include the instruments.

There are a few concerns when assembling the system. First the sensors must be held in place to avoid disconnections due to vibration. Second the Altimu v4 must be placed in such a way so that one of its axis is aligned with the axis of symmetry of the rocket, and it must be securely fixed so as not to lose its alignment. Third, if telemetry is a great concern, the XBee must be close enough to the surface to be able to efficiently transmit data. Fourth the GPS must preferably point towards the sky.

One possible adaptation (which was selected) was to cut aerospace eXtruded PolyStyrene (XPS) to fit the sensors and the inside of the rocket's nosecone using Computer-aided Design (CAD) modelling and Computer-aided Manufacture (CAM) Computer Numerical Control (CNC) for construction, appendix E. Other options would be an electronic bay in larger rockets, a light 3D printed nosecone and simply "stuffing" a nosecone or body tube while taking care to maintain the Altimu 10 Z-axis aligned with the rocket symmetry axis.

The actual wiring is described in table 3.17, not included are the power connections, which are between the sensor's ground and the Arduino ground and the sensor's Vin and the 3V port.

Table 3.17: Connections

Sensor	Port	Pins
Pololu	SDA	A4 (SDA)
	SCL	A5 (SCL)
microSD	CS SS	D10
	DI MISO	D11
	DO MOSI	D12
	CLK SCLK	D13
GNSS	Tx	D0 (Rx)

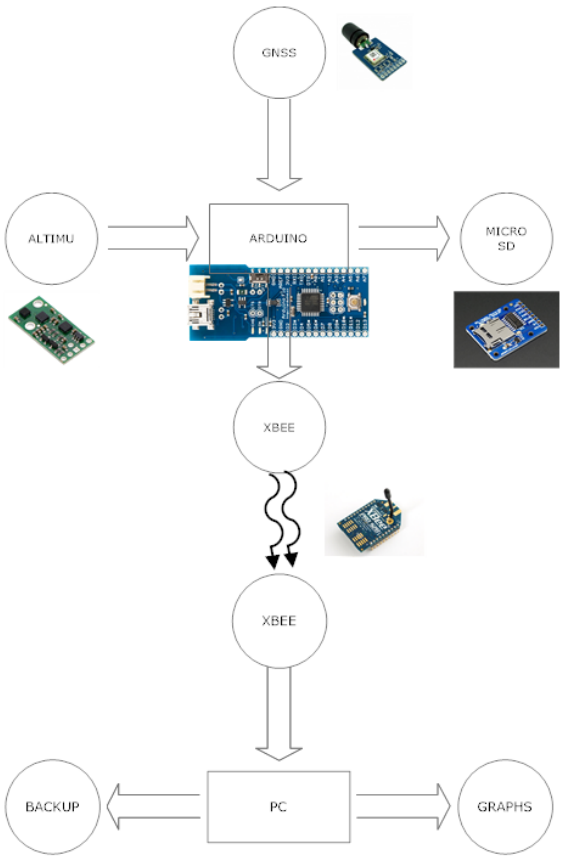


Figure 3.9: System diagram

3.9.1 Design Verification

To verify the stability of the rocket and to calculate its estimated maximum altitude achieved three software were used, Rocksim a proprietary software from Apogee Rockets[51], OpenRocket, a tried and tested rocket design tool [52], and as self-developed Matlab script that calculated height based on the eq. (3.1) and using Newton's method to solve it. The full code is available in appendix B.1.

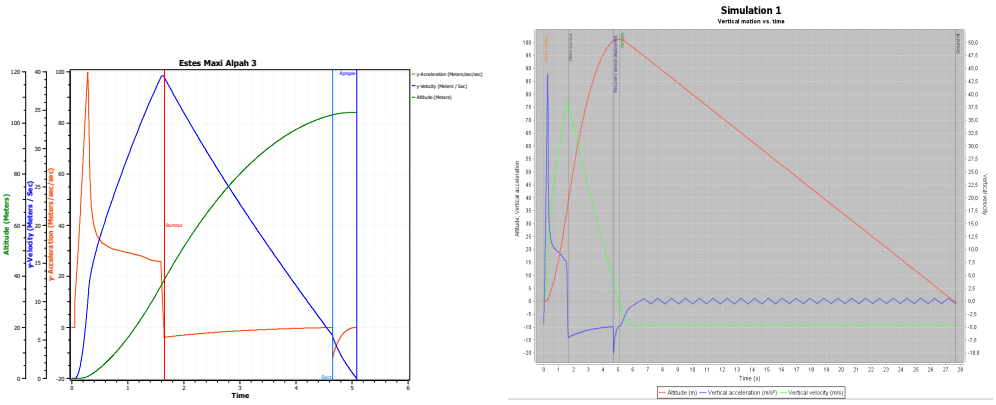
eq. (3.1) is a one-dimensional approximation of rocket flight, only the vertical coordinate is considered, the acceleration of gravity (g) is assumed constant (valid when altitude is much smaller that the radius of the Earth [20]), thrust ($T(t)$) is dependent on time and extracted from the rocket motor's curve, as a two step function, where propellant loss, and therefore mass ($m(t)$), is only dependent on impulse consumed. The drag coefficient is dependent only on the cross-section area of the rocket (A), the air density (ρ) which is assumed to be that of a sea level standard atmosphere, and C_d which is the drag coefficient of the rocket, estimated as 0.4 [14].

$$\frac{d^2x}{dt^2} = \frac{T(t)}{m(t)} - g - k\frac{dx}{dt} \tag{3.1}$$

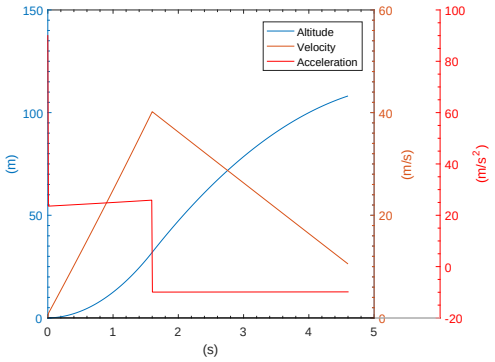
where,

$$k = \frac{1}{2}A\rho C_d \tag{3.2}$$

The results were the following: first, that the rocket remained stable, which means that the static margin is greater than two calibres; second, the expected values for some variables during flight, important



(a) RockSim (b) OpenRocket



(c) Matlab

Figure 3.10: Simulations

quantities are displayed in table 3.18, the behaviour during flight in figs. 3.10a to 3.10c, both Rocksim and Matlab simulations only go up to chute deployment, the former due to it being a free test version and the latter due to the end of the validity of the equation modelled, OpenRocket goes until touchdown occurs.

Table 3.18: Simulation Results

	Rocksim	OpenRocket	Matlab
CP	65.6 mm	62.7 mm	-
mass w/o motor	258.2 g	260 g	-
CG w/o motor	42.6 cm	42.9 cm	-
SM w/o motor	3.48 cal.	3.01 cal.	-
mass w/ motor	300 g	302 g	300 g
CG w/ motor	47.07 cm	47.4 cm	-
SM w/ motor	2.81 cal.	2.32 cal.	-
Max. Altitude	104 m	101 m	108 m
Max. Vert. Vel.	39.52 m/s	39 m/s	40.18 m/s
Max. Vert. Acc.	90.03 m/s^2	87.7 m/s^2	90.19 m/s^2

Chapter 4

Algorithm and Code development

This will constitute a brief explanation the main parts of the algorithm of the Rocket Segment Arduino sketch, the Ground Segment C++ Telemetry station, Python binary to Comma Separated Value (CSV) conversion and Matlab data handling code, and specify some restrictions.

4.1 Rocket Segment: Arduino sketch

In order to create and compile the code the Arduino IDE software was used, specifically in the mode matching the platform used, the Fio. The main Sketch is included in appendix A.

The major problems when developing the code in the Arduino are the limited computational capacity and limited memory. The former means not many calculations can be performed while keeping update rate high, this is limiting mostly due to AHRS determination which takes up most resources, and communications, which take the remainder, to help with communications only hardware based protocols are used, so the single dedicated UART pins are used Rx for GPS and Tx for XBee. The latter (memory) limits the system twofold, first by limiting the number of libraries that can be used, and second by limiting variables to avoid stack overflow. One thing that helps with both is GPS reception and parsing, and output sentence management using memory unions and structures.

The libraries used are SDFat [35], SPI[53], Wire[54], NeoHWSerial[55], L3G[56], LSM303[57] and LPS[58]. The non-native fat library was used because it functions better with modern microSD cards (allowing faster write speeds) while also being lighter than the old libraries. NeoHWSerial was used to allow interrupt routines triggered by incoming data on the Rx pin, useful for receiving GPS sentences as will be shown. The remaining libraries are standard for the sensors (L3G,LSM303,LPS) and for communication protocols (Wire for I^2C and SPI for SPI).

There are three major activities in the code, reading the GPS UBX sentence, calculating attitude and outputting the data. The determination of attitude follows the algorithm in [42, 59] and will not be discussed further, the reception of the UBX sentence is included in section 4.1.1, the data output is addressed presently.

The Arduino outputs the sentence displayed in table 4.1, this sentence was structured in such a way to have all data self-contained in 32 bit intervals to avoid padding in 32 bit (or 64 bit) systems, making the data-stream readable by the union of a byte array and a structure in a little-endian system, such as most computers currently in use.

Algorithm 4.1 Read GPS sentence and verify checksum, signal print if valid

Require: $byte$
  if $i > 1$ **then**
    if $i = 99$ **then**
      return $GPS.PVT(i) \leftarrow byte$
      $i \leftarrow 0$
      if $GPS.PVT(98) = Ck_a$ **and** $GPS.PVT(99) = Ck_b$ **then**
        return $prt \leftarrow$ **true**
        $Ck_a \leftarrow 0$
        $Ck_b \leftarrow 0$
      else
        $Ck_a \leftarrow 0$
        $Ck_b \leftarrow 0$
      end if
    else
      return $GPS.PVT(i) \leftarrow byte$
      $i \leftarrow i + 1$
      if $i \neq 98$ **then**
        $Ck_a = Ck_a + byte$
        $Ck_b = Ck_b + Ck_a$
      end if
    end if
  else if $i = 0$ **and** $byte = 0\text{xB5}$ **then**
    return $GPS.PVT(i) \leftarrow byte$
    $i \leftarrow i + 1$
  else if $i = 1$ **and** $byte = 0\text{x62}$ **then**
    return $GPS.PVT(i) \leftarrow byte$
    $i \leftarrow i + 1$
  else
    $i \leftarrow 0$
  end if

Table 4.1: Arduino output message

Bytes	quantity	description	format
1	G	header 1	char
2	C	header 2	char
3-8	Gyr[3]	angular velocity (x,y,z)	3 x int16
9-14	Acc[3]	acceleration (x,y,z)	3 x int16
15-20	Mag[3]	magnetic field (x,y,z)	3 x int16
21-24	roll	roll	float
25-28	pitch	pitch	float
29-32	yaw	yaw	float
33-36	t	time	int32
37-40	sample period	sample period	float
41-44	pressure	pressure	float
45-48	temperature	temperature	float
49	hour	hour	int8
50	min	minute	int8
51	sec	second	int8
52	SV	number of satellites used	int8
53-56	nano	nanosecond	int32
57-60	lon	longitude	int32
61-64	lat	latitude	int32
65-68	hGEO	altitude	int32
69-72	velN	velocity North	int32
73-76	velE	velocity East	int32
77-80	velD	velocity Down	int32

4.1.1 Receiving the GNSS UBX NAV PVT sentence

Please refer to section 3.5.1 for the PVT sentence structure. To simplify parsing, and taking advantage of the equal endianness of the GPS and the Arduino, there is a union (GPS) between the incoming byte array PVT and the byte structure gps that follows the PVT sentence structure displayed in table 3.7.

To deal with reading the PVT sentence the code runs with an interrupt connected to the Rx UART pin, thus guaranteeing that every character coming through is read. This interrupt code can be seen in algorithm 4.1.

First the header sync characters, 0xB5 and 0x62, are verified and the byte counter i is started, if the header sync characters do not match i is reset to 0 and awaits a correct start.

If the start is successful, all bytes are read and saved, and the checksum, Ck_a and Ck_b, is calculated following an 8-bit fletcher algorithm.

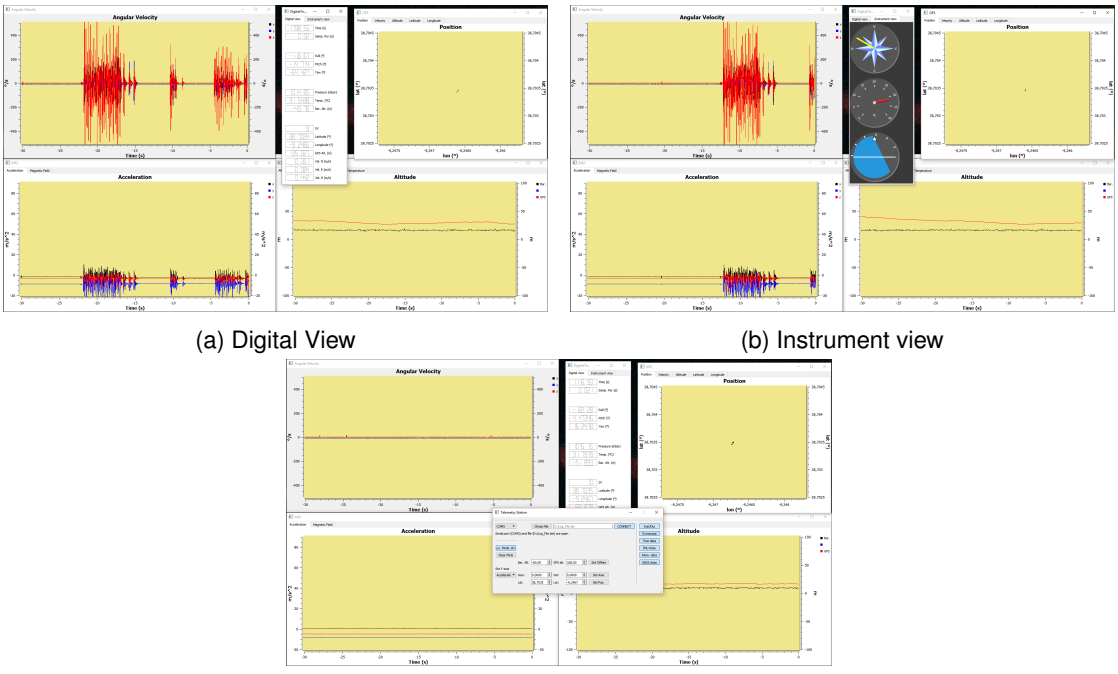
For the 99th character checksum calculation is terminated and for the 100th the checksum calculated values are compared to their expected values and if they both match the $print$ flag is set to **true**.

4.1.2 Main loop

The main loop, seen in algorithm 4.2, runs at 50 Hz if possible. Every 20 ms it reads data from the Pololu Altimu sensors, process it using the AHRS algorithm [42, 59] to calculate attitude, if the $print$ flag is **true** then it updates the GPS part of the output sentence and sets the flag to **false**. Every 20 ms (more if there is a SD write block) the system prints a stream of 80 bytes from the structure in table 4.1 to both the XBee and the SD card.

Algorithm 4.2 Main Loop

Require: Libraries and variable setup
  loop
    $dt = time - time_{old}$
    if $dt >= 20$ms **then**
      $time_{old} \leftarrow time$
      Read Sensors
      RUN AHRS ALGORITHM
      print Sensors
      if $print$ is **true then**
        print gps
        $print \leftarrow$ **false**
      end if
    end if
    if $time > LOG_{time}$ **then**
      EXIT LOOP
    end if
  end loop



(a) Digital View

(b) Instrument view

(c) Menu open

Figure 4.1: Application views

4.2 Ground Segment: Telemetry Station C++ code

A telemetry base station or ground tracking station allows for additional redundancy by recording data remotely and makes possible to visualize this data in real-time.

The code was developed in QT [60] a multi-platform IDE for C++ programing, currently the code has been deployed as a WIN32 application, but using the base code it can easily be deployed for other platforms.

The Telemetry station program has three main parts that coincide with the three different threads used, reception and logging of the message, conversion of data and handling of plots vectors, and plotting and UI.

The reception follows a similar strategy to the one described in section 4.1.1, but without use of a

checksum. The function then logs the sentence into a binary file and passes it on to the vector creator thread.

The vector creator thread is described in section 4.2.1.

The main thread displays the graphs and options of the UI, allowing the user to choose which plot (or widget) to display and certain details such as altitude offset, the scale of each graph, the COM port used and the log file name and directory.

To illustrate the implementation, the vector creator function will be analysed, as the full code would unduly extend this section. Some screens taken on a Windows machine can be seen in fig. 4.1.

4.2.1 Example: Make Vectors

The vector creator thread houses an object that converts the structure passed by the reception thread object to the correct units and then manages the data vectors so that they display approximately the previous 30 seconds of data, which signifies a 1500 dimension vectors. This object provides the pointers to the vector required by each graph object in the main thread.

A simplified algorithm is displayed in algorithm 4.3, the corresponding code is included in appendix D

Some key aspects of the implementation are related to the correct management of memory to avoid overwrites when appending vectors. This is solved by using the correct class for the vector, in this case the qVector format.

Another important aspect revolves around maintaining a correct time vector between t=0 and t=-30 s (30 seconds before), this is guaranteed efficiently by updating existing memory locations when required.

Algorithm 4.3 Make Vectors

```
Require: Header File
  INPUTS: message and or clear vector
  if message received then
    apply conversion factors
    if vectors size < 1500 then
      append vector
    else
      delete vector(0)
      append vector
    end if
  end if
  if clear vector received then
    vector resize to 0
  end if
  return vector
```

4.3 Python Binary to CSV code

Python was used to convert binary to CSV due to the built-in structure casting and CSV file printing, though many other programs could be used. The script converts a binary file in the format displayed on table 4.1 to a CSV file following the structure in table 4.2, this allows for easier post processing. It is included in appendix C.1.

Table 4.2: CSV file structure

Field number	quantity	conversion factor	final unit
1-2	header		
3-5	angular velocity (x,y,z)	70*0.001	º/s
6-8	acceleration (x,y,z)	16*0.244*0.001	g
9-11	magnetic field (x,y,z)	0.16*0.001	G
12	roll		rad
13	pitch		rad
14	yaw		rad
15	time	0.001	s
16	sample period	-	s
17	pressure	-	mbar
18	temperature	-	°C
19	hour	3600	s
10	minute	60	s
21	second		s
22	number of satellites used	-	-
23	nanosecond	10^{-9}	s
24	longitude	-	º
25	latitude	-	º
26	altitude rel. to geode	0.001	m
27	velocity North	0.001	m/s
28	velocity East	0.001	m/s
29	velocity Down	0.001	m/s

4.4 Matlab Data Handling Code

Matlab was used for most figure plotting and data handling due to the inherent ease to plot in this language.

The Matlab code takes the inputs from a comma separated value file (.csv) in the format displayed in table 4.2, extracts the data and applies the correct factor to obtain the measured parameters in the desired unit. It obtains also two time vectors, one for the GPS data and another (with more inputs) for the sensor data. The code is included in appendix B.2. The values for hour, minute, second and nanosecond are added to obtain the time vector from the GPS data.

This script was used for the post processing and analysis of the data; other software could be used depending on user preference. It is included in appendix B.2.

Chapter 5

Ground Testing

Ground test include all tests of the system which did not involve any flight-time. The objectives of these tests were to determine the system's physical characteristics, to calibrate the sensors, and to validate the system for airborne use.

5.1 Weighing and CG

The objective of this test is to determine the mass of the instrumentation system, and the new centre of gravity of the rocket. The objective is twofold, knowing the information in and of itself (comparing it to the simulated models), and devising a substitute mass to use in the first flight tests, thus minimizing the risk of system loss.

5.1.1 Requirements

The entire rocket assembled with recovery system, motor and payload is required, as well as a precision digital scale. For the replacement mass an appropriate set of masses is required. A long wire is needed to determine the CG.

5.1.2 Procedure

Mass

Place the system on the scale and measure its mass, repeat zeroing the scale after the system is lifted to guarantee precision, switch then to the substitute masses and remove or add mass until the weight of the mass is close enough to that of the system. Repeat the measurements of mass and system in succession to verify the results.

Determination of CG

Suspend the rocket on a wire with the system on the nosecone and engine on the back, move the wire along the length of the rocket until both side are levelled.

5.1.3 Results

The mass of the instrumentation system was found to be 48.00 g, very close to the estimate 50 g. The weight of the replacement mass, which consisted of four discreet masses, two 20 g, one 5 g and one

2 g, and a piece of extruded polystyrene foam also weighted 48.00 g, after careful removal of excess mass from the foam block. The foam block was used to carry the other weights and to insure a secure fit with the instrument housing, thus avoiding vibrations.

The Rocket CG was found to be at 47.5 cm from the tip, very close to those obtained in the design software, table 5.1.

Table 5.1: Simulation Results

	Rocksim	OpenRocket	Measured
CG w/o motor	42.6 cm	42.9 cm	42.5 cm
CG w/ motor	47.07 cm	47.4 cm	47.5 cm

5.2 Attitude Test

This test aims to finds the static offsets of the accelerometers and the gyroscopes, it also serves to test the accuracy of the AHRS algorithm.

It's important to note the values between the standard g, g_0 = 9.80665 m/s^2= 1 g and the g acceleration in Lisbon, g_{LX} = 9.80131 m/s^2 = 0.999455 g.

The relation between the acceleration of gravity in the Earth frame $(0, 0, g)$ and in our AHRS axis, **a** is given by Equation 5.1 which is the conversion of the Z component of the general frame to a Tait–Bryan rotation, following [61],

$$a_x = -\sin(\theta).g \tag{5.1}$$

$$a_y = \cos(\theta).\sin(\varphi).g \tag{5.2}$$

$$a_z = \cos(\theta).\cos(\varphi).g \tag{5.3}$$

where θ is roll and φ is pitch.

It must also be noted that the AHRS axis are a 180 degree rotation around the Pololu accelerometer/magnetometer/gyroscope axis **m**, which means,

$$a_x = m_x \tag{5.4}$$

$$a_y = -m_y \tag{5.5}$$

$$a_z = -m_z \tag{5.6}$$

5.2.1 Requirements

The first requirement for the test is all the systems components and a flat platform where to attach the Pololu. A second platform that is can be tilted in a controlled fashion to the desired angle is also required, and an inclinometer to measure the slope, in this case the inclinometer used was a LUCAS Anglestar DP 60. The setup can be view in figure 5.1.

5.2.2 Procedure

Place the platform with the instruments on the tilt platform flat and let the program run for about 10 minutes to assure a stable state. Then tilt the platform to the desired angle, rotating the base platform

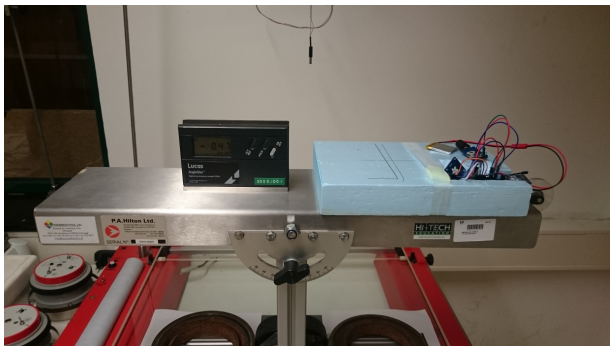


Figure 5.1: Attitude test setup

$\pm$90 degrees over the tilt platform plane to measure roll.

5.2.3 Results

Static offsets

The first test, with a static platform, allows for the calculation of the static offset of both the accelerometer and gyroscope as the expected values for each of them are known, the gyroscope should be at **0**, and the accelerometer should be measuring g_{LX} = 0.9995 g with θ = -0.50° and φ = -0.25°, which means following eq. (5.1) and eq. (5.4).

$$a = (0.0088, 0.0044, -0.99941)$$

As can be seen in figs. 5.2a to 5.2c, 5.4a and 5.5a there is a small stabilization time until the values remain constant, this appears to be mostly due to the influence of temperature on the readings (sensor warm-up), and stabilizes at the same time as the temperature does.

We calculate the values based on the average readings during the last half of the test.

The values for the gyroscope offsets are,

$$gyr_{off} = (0.8279, 11.6876, 2.0109) \text{ °/s}$$

as for the accelerometers, subtracting the expected values,

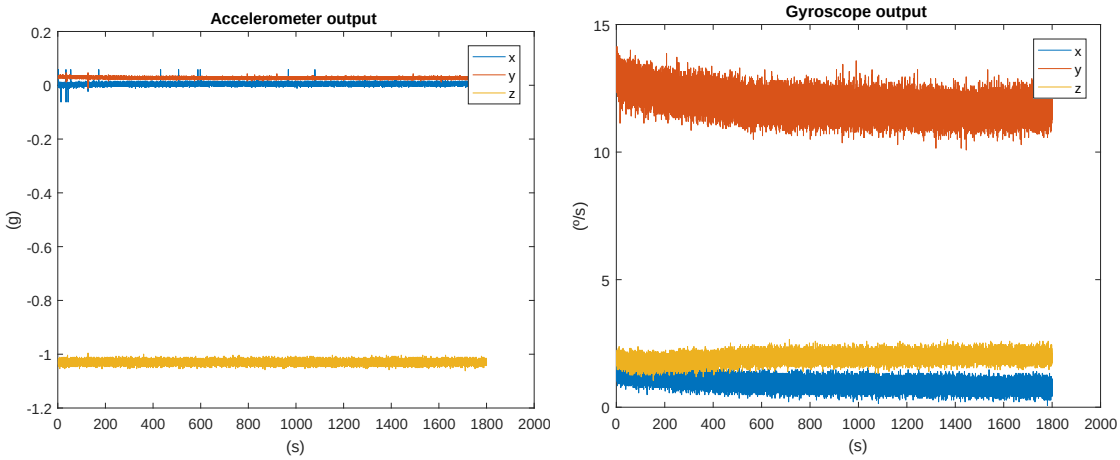
$$acc_{off} = (0.0135, 0.0234, -0.0303) \text{ g}$$

Attitude angles

Table 5.2: Attitude values

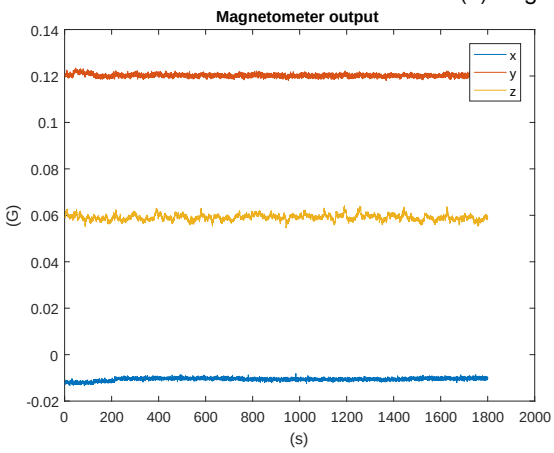
θ	φ	θ_m	σ_θ	φ_m	σ_φ	ψ_m	σ_ϕ
45°	-	45.57°	0.061	-1.02°	0.059	131.97°	0.287
-45°	-	-45.41°	0.070	1.98°	0.053	-41.21°	0.142
-	-45°	-0.48°	0.057	-45.27°	0.058	-134.20°	0.157
-	45°	0.34°	0.047	44.02°	0.057	35.83°	0.075

Using the algorithm discussed in [27], the attitude can be calculated. In figs. 5.3a to 5.3c we can observe the values obtained for the respective angles. The actual slope being measured with the in-



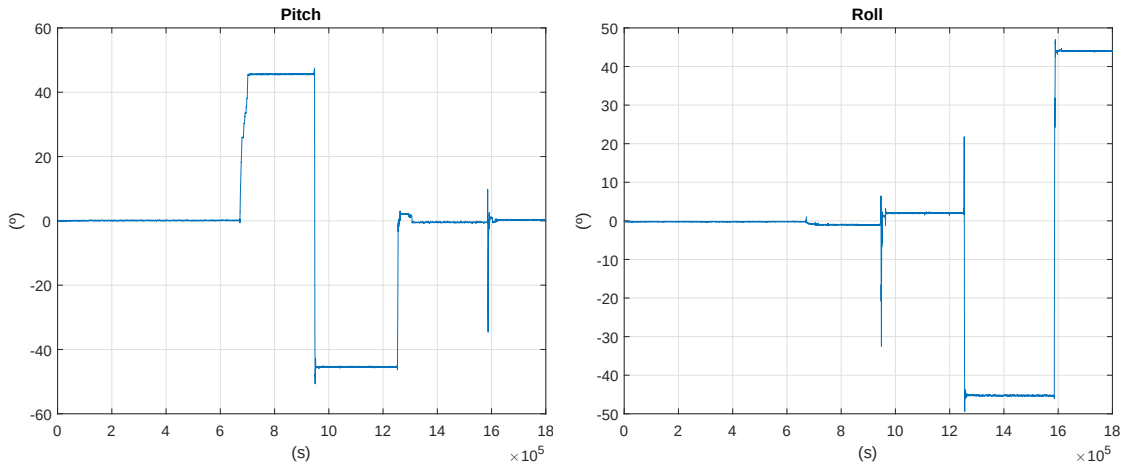
(a) Acceleration

(b) Angular velocity



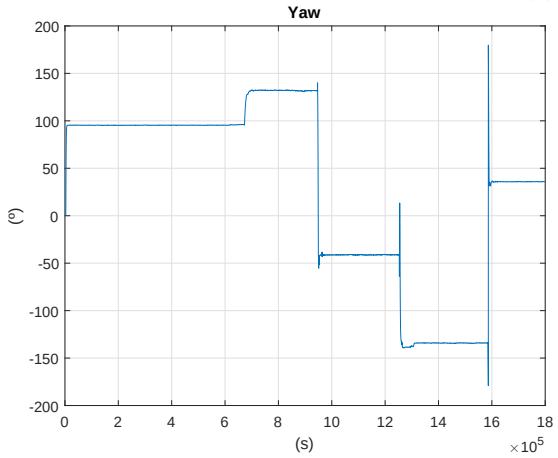
(c) Magnetic field

Figure 5.2: Acceleration, angular velocity and magnetic field, GT01



(a) Pitch

(b) Roll



(c) Yaw

Figure 5.3: Attitude, GT02

clinometer and the values obtained by the system are present in table 5.2. The system results here displayed are a mean of the values obtained during the relevant sample period.

It's worth noting there is no theoretical expected value for the transverse measurement or yaw due to, respectively, a small convex bend of the surface of the inclining board and the strong magnetic currents in the lab, which make defining south/north with precision a complex task. It is still clear though, that the difference between each measurement's yaw is approximately $45\,°$ form the closest one, as expected.

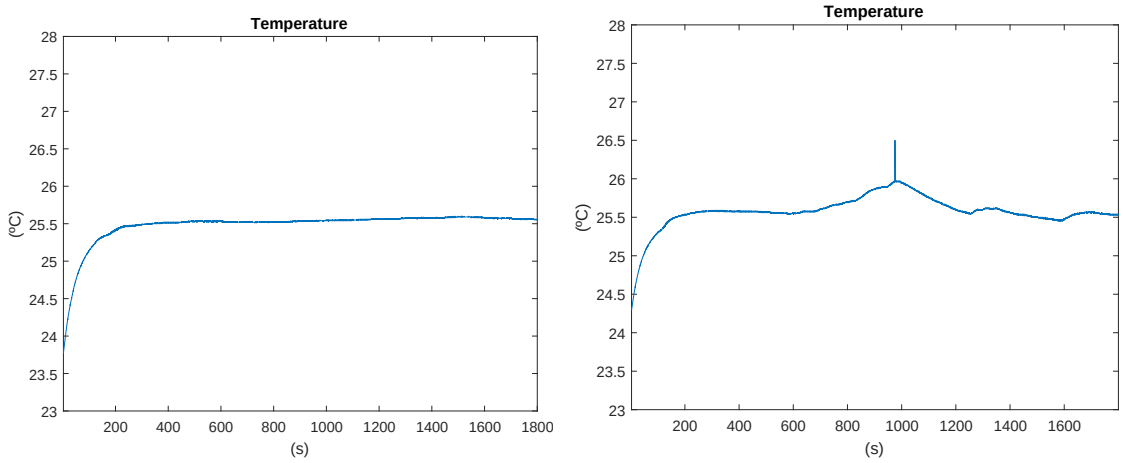
It can be observed that the biggest error measured inferior to $1\,°$, which means the system is more than precise enough for our goals.

5.3 Pressure and Temperature Test

This test aims to finds the static offsets of the barometer and thermometers.

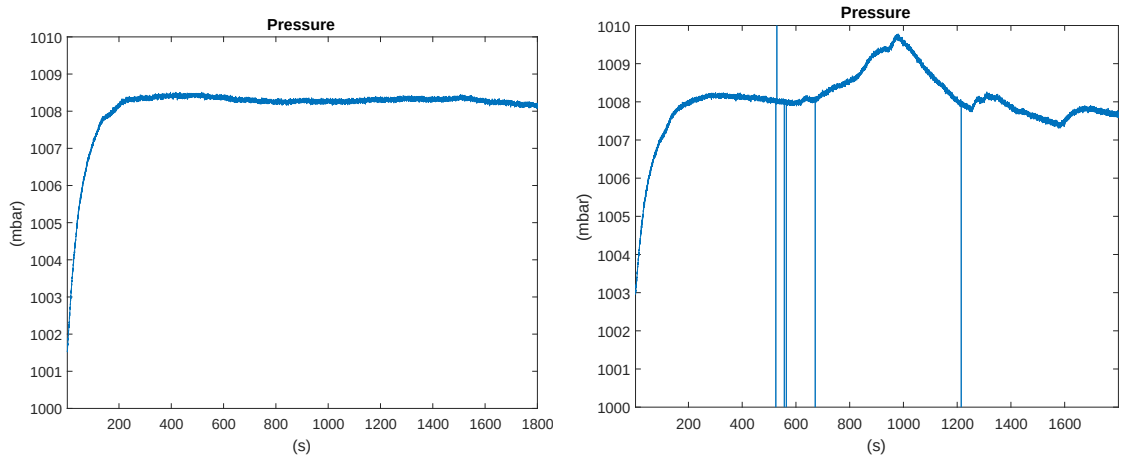
5.3.1 Requirements

The first requirement for the test is all the systems components. An external barometer and thermometer to compare the results to are needed. A location with stable pressure and temperature is preferable, the presented data relates to a closed room in a basement which guarantees barometric and thermal stability.



(a) GT01 (b) GT02

Figure 5.4: Temperature



(a) GT01 (b) GT02

Figure 5.5: Pressure

5.3.2 Procedure

Turn on the system and measure both the temperature and the pressure while recording the data from the external sensors. Take note when doors/windows are opened as it may influence both temperature and pressure.

5.3.3 Results

The temperature was known to be a stable 22°C, we can see in fig. 5.4a that after an initial warm up the temperature stabilizes around 25.5°C. On the following test, fig. 5.4b we can observe the temperature to be keeping within the same values.

Pressure follows a similar evolution, an initial warm-up and then a steady state value of about 1008 mbar, fig. 5.5a, in the following test we see a similar result, fig. 5.5b, and a clear peek, that coincides with the time a door stood open, notice how it matches a peak in temperature at that exact time. It's clear that both pressure and temperature readings are stable, pressure has no visible offset and temperature seems to have a 3.5 °C offset, probably due to sensor heating.

5.4 GPS

This test aims to verify the accuracy of the GPS, first in a static environment and then in a moving one.

5.4.1 Requirements

The required equipment consists of: the instrumentation system, a transport platform, a relatively clear space for the static test and a vehicle for the dynamic test.

For the static test to compare with differential GPS results: Septentrio PolaRx2e, recording data in the internal memory; NovAtel L1/L2/L-Band GPSAntenna Model 600-LB; NovAtel cable; 12 Volts battery as a power supply; Tripod for antenna assembly. Additional the permanent GPS station of the *Faculdade de Ciências da Universidade de Lisboa* was used, which includes a Leica RS500 receiver and a Leica AT504 antenna with choke ring. Trimble Total Control 2.73 software is needed to compute the differential correction

5.4.2 Procedure

Setup the GPS on the pc following section 3.5.2, select the desired dynamic platform model, in this case, static for the static test and dynamic for the portable test. Set up the instruments and start logging.

For the static test place the tripod in the desired place (preferably a clear high point) and leave the instrument taped to the differential antenna. For the dynamic test drive through the intended course.

5.4.3 Results

Static

The instrumentation system was 106.5 mm west of the differential antenna, this is already included in the position results. The differential software presented horizontal standard deviation of 0.5 mm and vertical standard deviation of 0.7 mm, altitude is relative to WGS84 ellipsoid.

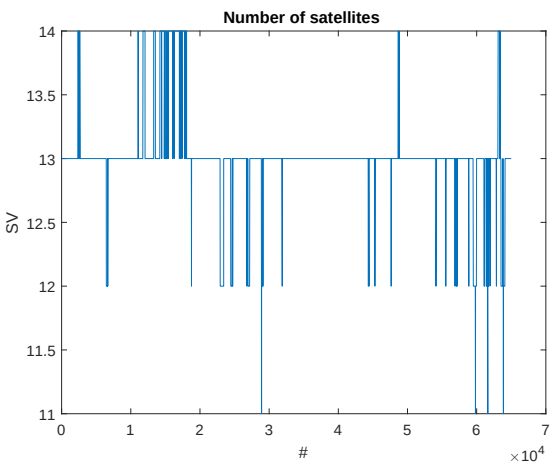
The number of satellites used in the solution, fig. 5.6a, was more than sufficient for a 3D position, with this in mind, and knowing there was no differential correction for the instrumentation system, an accuracy close to the maximum 3 m is to be expected.

In fig. 5.6b the altitude accuracy can be observed, the place in question was at 142.209 meters, we can observe the average measured altitude was 145.5, with an average error of -3.29 meters, and the further border of the 2RMS value was at -7.59 meters from the actual position [62], thus altitude error was at maximum 7.59 meters.

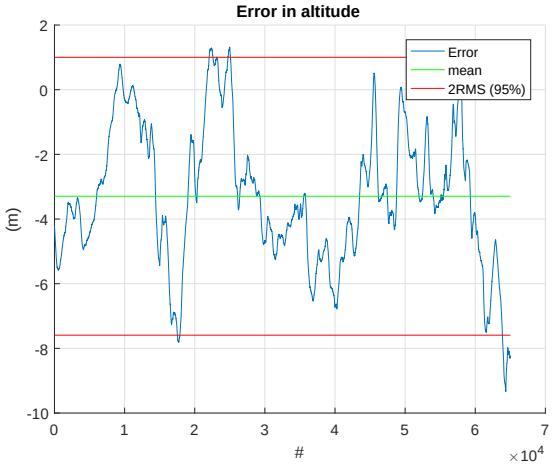
Regarding the horizontal position, fig. 5.6c, in fig. 5.6d the error is presented (after converting to Universal Transverse Mercator [63]), the 50% Circular Error Probable (CEP) [62] is approximately 1.5 m in radius while the Distance Root Mean Squared (DRMS) (approx. 95-97% CEP [62]) radius is inferior to 4 m, nearly as good as can be expected. The absolute deviations are also small as displayed on table 5.3.

Table 5.3: Position values

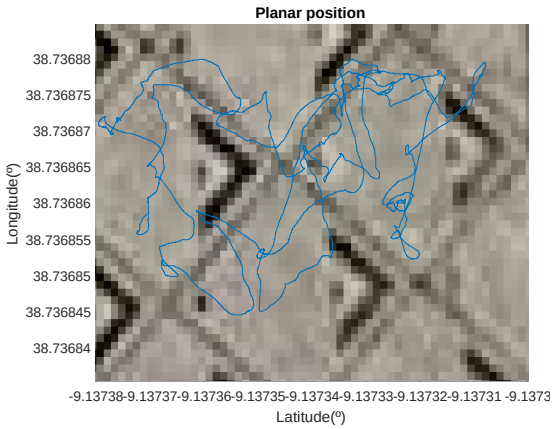
	Longitude	Latitude	δ lon. in UTM	δ lat. in UTM
Real	38.7368605°	-9.1373417°	-	-
Measured	38.7368666°	-9.1373418°	-	-
Difference	0.000060°	-0.0000020°	-0.6672 m	0.1725 m



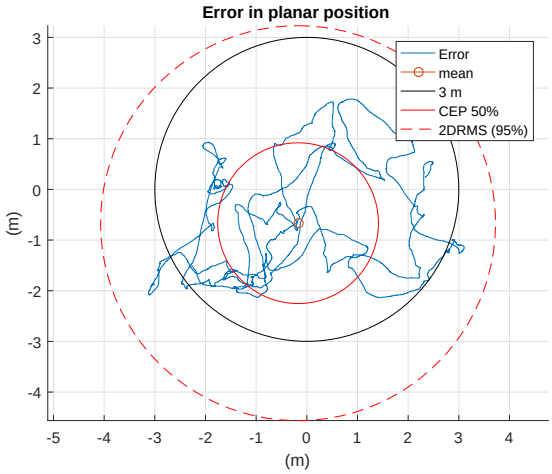
(a) Satellites in view



(b) Error in altitude, z



(c) Planar position, (Lat,Lon)



(d) Error in planar position

Figure 5.6: Position, GT01

Velocity is always close to zero, fig. 5.7, with amplitude inferior to 0.05 m/s and a distribution roughly equal above and below the actual value, near 0 m/s average for all values, table 5.4.

Table 5.4: Velocity (m/s)

μVel. North	σVel. North	μVel. East	σVel. East	μVel. Down	σVel. Down
-0.00001	0.0045	0.00035	0.0072	0.00451	0.0124

Dynamic

The first test done utilized the Portable dynamic mode and took the path displayed in fig. 5.8.

In fig. 5.9, the path taken and velocity on a particular area is displayed, the location was chosen for its scarce traffic, and it can clearly be observed the position is accurate enough to be used for vehicle navigation.

The testing then proceeded to a highway test, divided in two parts, the first one going out, fig. 5.10c, and the second returning, fig. 5.10d. The satellites connected in both cases are in figs. 5.10a and 5.10b, the number of satellites used in the solution is between 10 and 12, which is far more than needed and should provide excellent results.

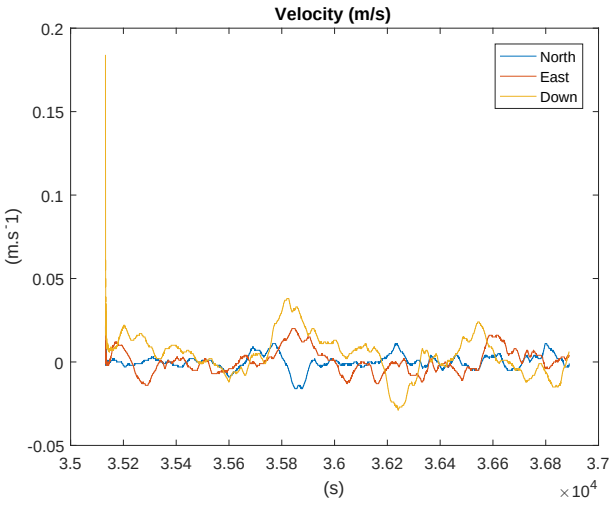


Figure 5.7: Velocity, Static Test

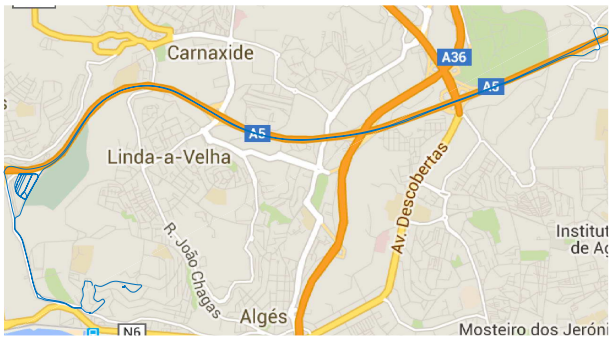


Figure 5.8: Full planar trajectory GT01

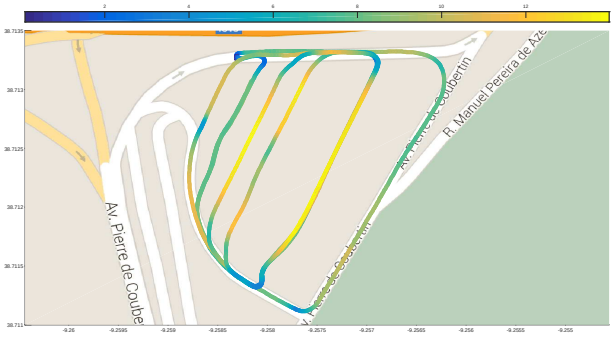
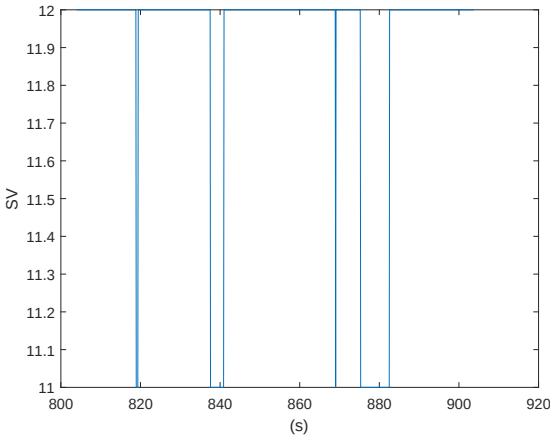


Figure 5.9: Planar position and velocity, for position accuracy, GT01

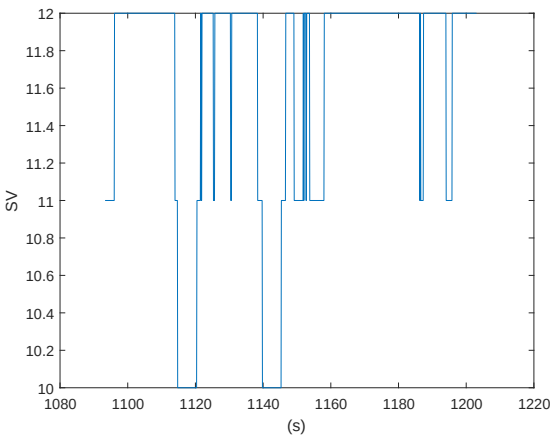
To estimate the accuracy of the altitude the GPS value is compared to the barometric one, figs. 5.11a and 5.11b. It's clear that despite a static offset the overall variation is much the same, so the vertical position is probably very accurate, discounting the aforementioned static offset.

The objective was to achieve approximately 27.78 m/s (100 km/h) in the first lap, and 33.33 m/s (120 km/h) on the second. As evidenced by figs. 5.12a and 5.12b the results were somewhat shy of the intended value, this due to the inherent imprecision of the driver, of the limiter or cruise control and the car speedometer (which is normally 10% [64]), added to the various road conditions, turns, climbs that slightly alter the value. Still, in both cases the results were appropriately close to the expected value, the mean values obtained were, respectively, 26.58 m/s and 31.56 m/s

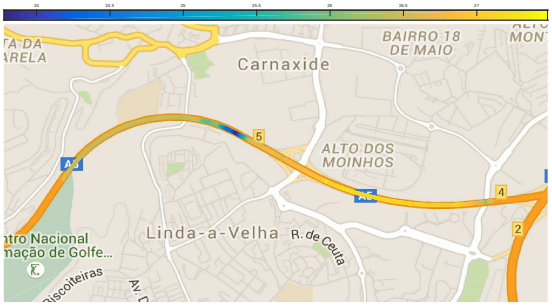
Then, in order validate the values of the velocity, an estimate of it was calculated by using finite



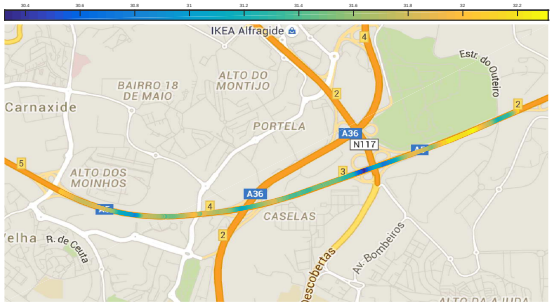
(a) Satellite coverage, stretch A



(b) Satellite coverage, stretch B

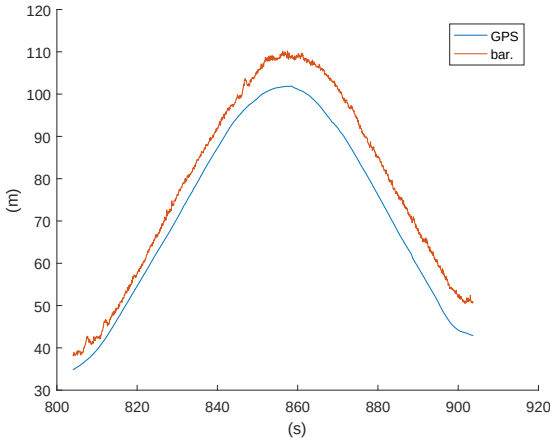


(c) Planar position, stretch A

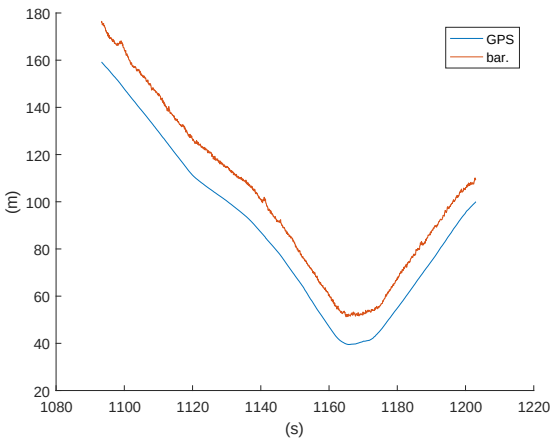


(d) Planar position, stretch B

Figure 5.10: Satellite coverage and planar position, GT01



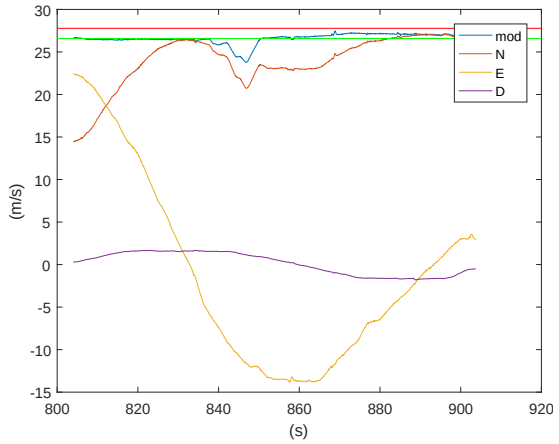
(a) Altitude, stretch A



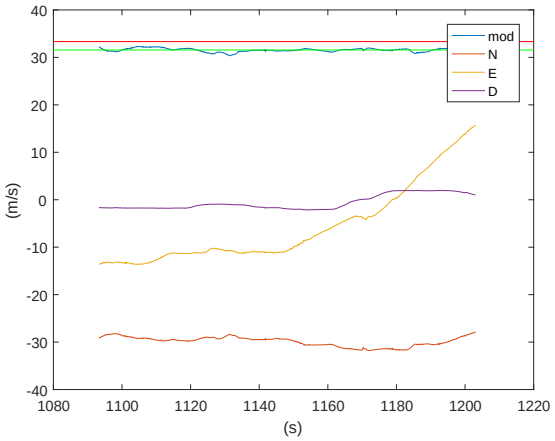
(b) Altitude, stretch B

Figure 5.11: Barometric and GPS altitude, GT01

differences from Universal Transverse Mercator (UTM) values of position, figs. 5.13a to 5.13c and 5.14a to 5.14c, this is possible because the GPS calculates these values with different methods, signal time for position and Doppler effect for velocity, [23]. It's clear that the variation of the values is very similar, though for obvious reasons (namely the precision of position being at maximum 3 m) the differentiated value presents more noise.

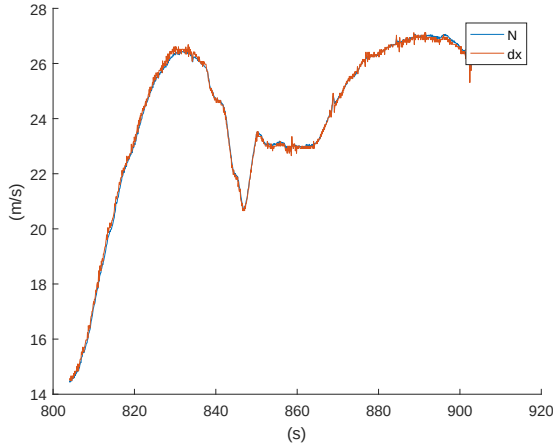


(a) Velocity, stretch A

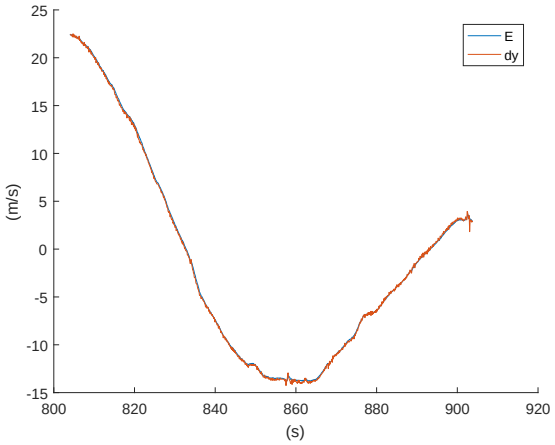


(b) Velocity, stretch B

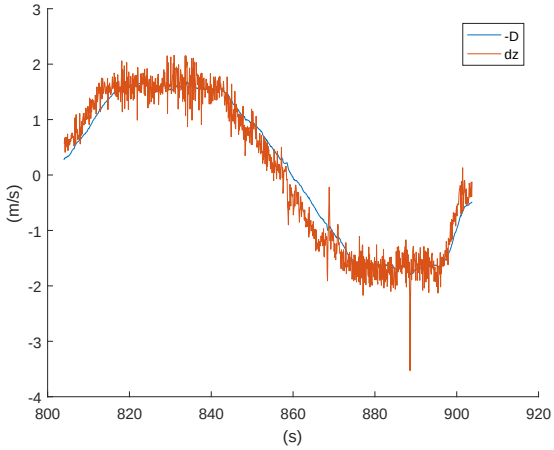
Figure 5.12: Velocity, GT01, theoretical test value in red



(a) X component, positive east

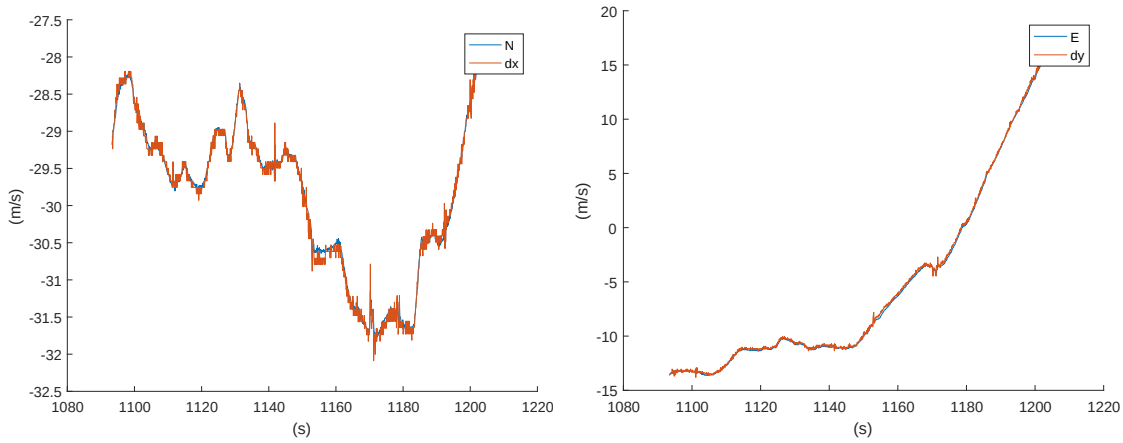


(b) Y component, positive north



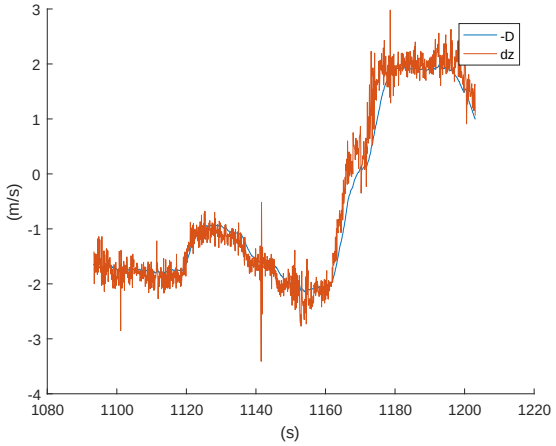
(c) Z component, positive up

Figure 5.13: Velocity components, stretch A, GT01



(a) X component, positive east

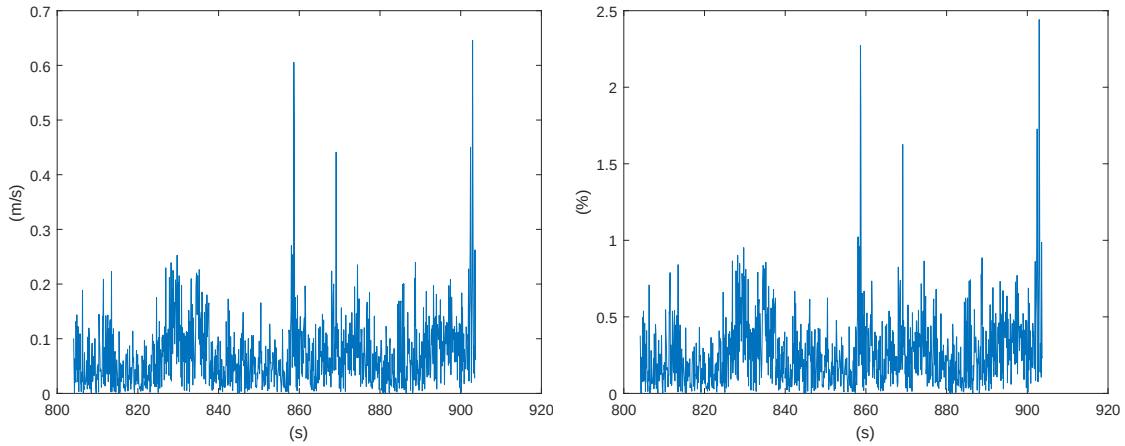
(b) Y component, positive north



(c) Z component, positive up

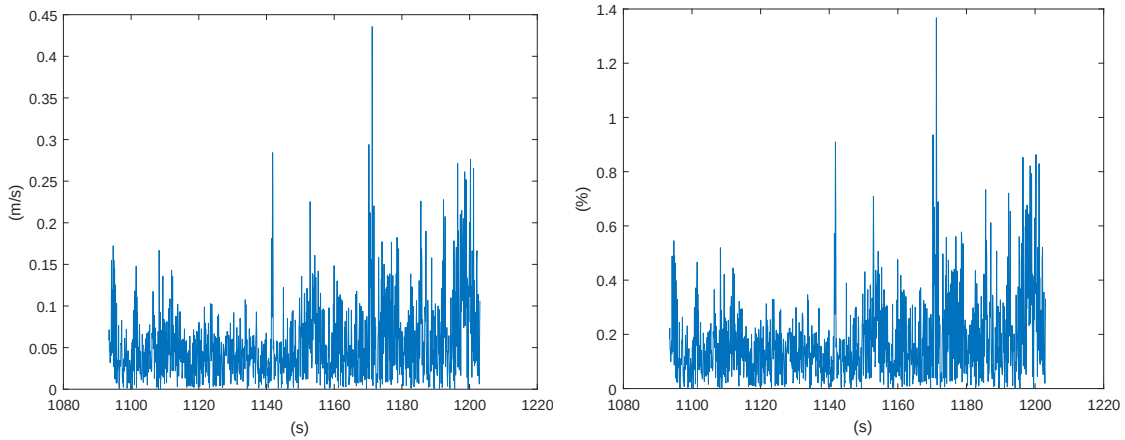
Figure 5.14: Velocity components, stretch B, GT01

The absolute difference between both predictions is presented in figs. 5.15a and 5.16a and the relative difference in figs. 5.15b and 5.16b, both are small and within the values expected (i.e. the absolute is inferior to 3 m/s).



(a) Absolute difference (b) Relative difference

Figure 5.15: Difference in velocity magnitude, stretch A, GT01



(a) Absolute difference (b) Relative difference

Figure 5.16: Difference in velocity magnitude, stretch B, GT01

Test with GPS on Airborne <4g mode

A test was made using the less filtered GPS mode, the Airborne <4g dynamic mode. This test was made mostly on the highway as seen in fig. 5.17.

Given the reduced use of filtering it's paramount to use as many satellites as possible, this will not be a problem with a rocket launch since it will normally be performed in an open field where the receiver has plenty of time to link to as many satellites as possible and once in flight it should remain the same (or improve) due to the altitude gain. Looking at fig. 5.18a it's clear the initial 4500 samples should be disregarded, as well as a few locations based on the x coordinate, between 4.785 10^5 and 4.85 10^5, where due to a tunnel and high buildings a lower number of satellites are in view. The result is displayed in fig. 5.18b.

Notice that for most of the trajectory the x position (longitude) could be said to be unique, i.e. longitude (y) and altitude (z) can be displayed as functions of x, figs. 5.19a and 5.19a.

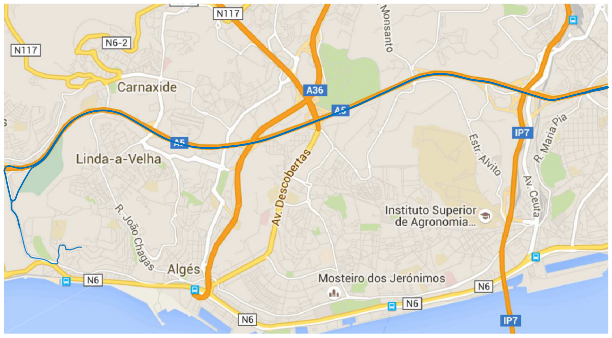
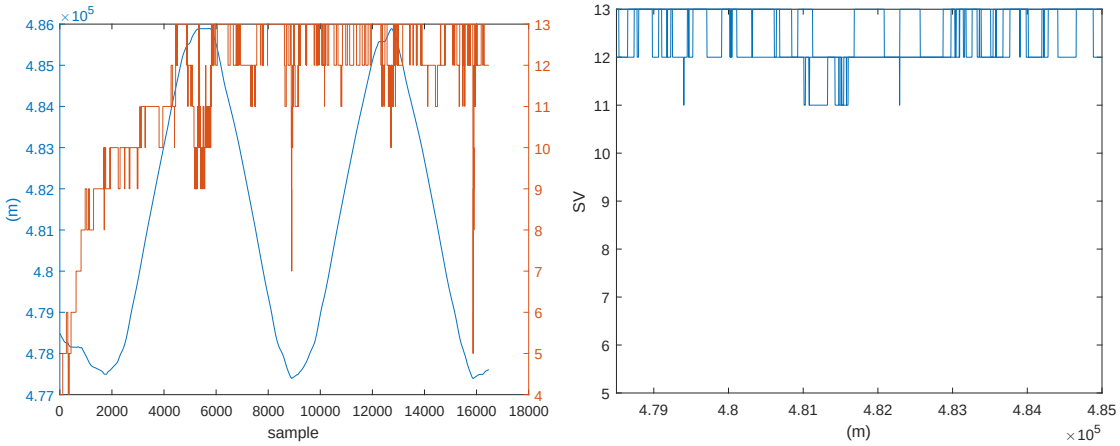


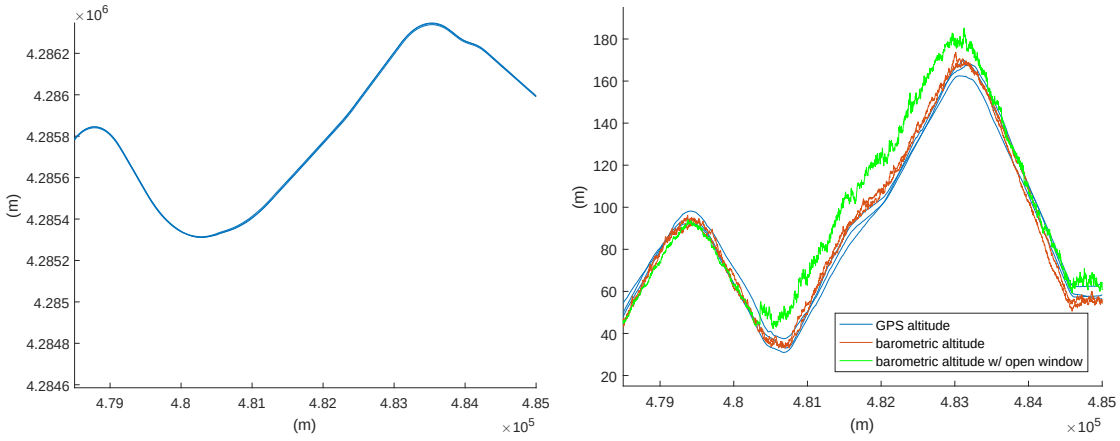
Figure 5.17: Position GT01



(a) Satellite coverage and X position of samples

(b) Satellite coverage on test section

Figure 5.18: Satellite coverage, GT01

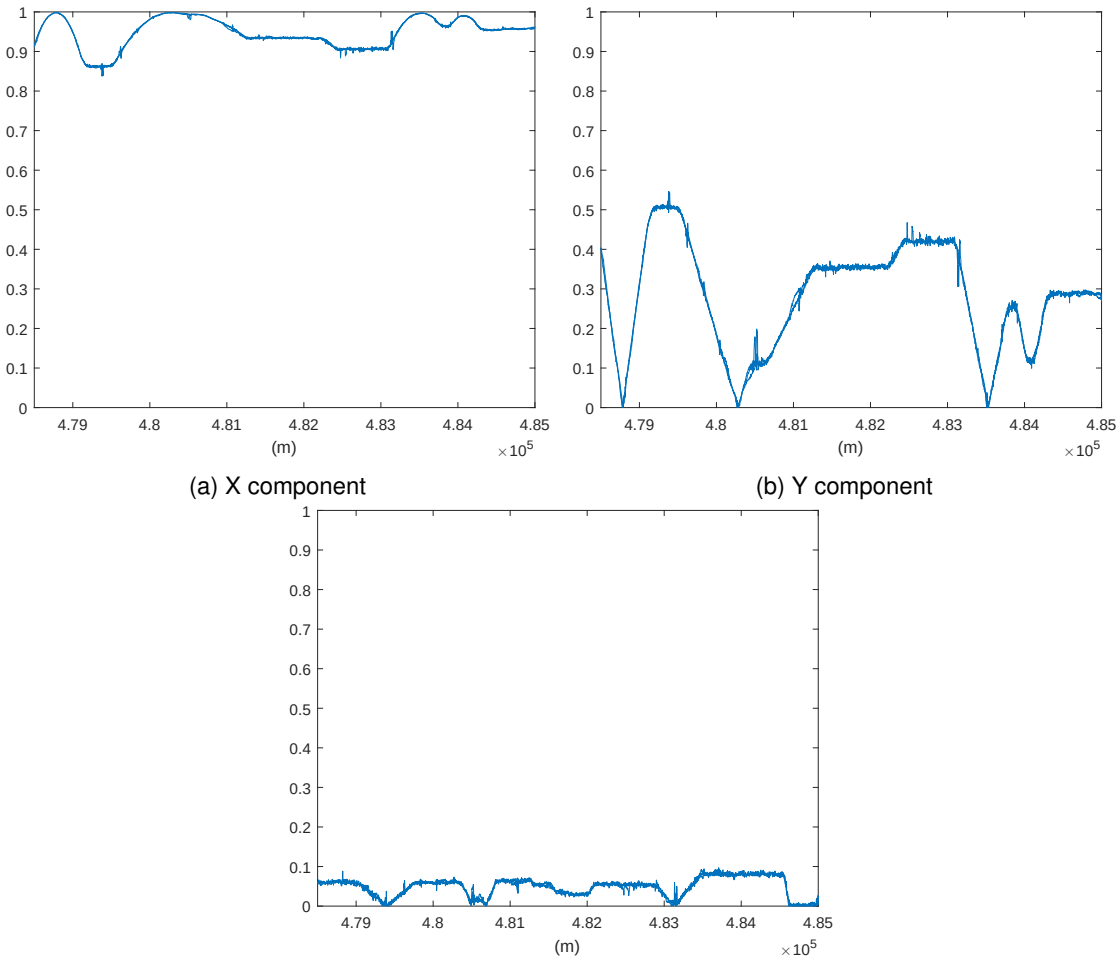


(a) Northing

(b) Altitude

Figure 5.19: Position for test section, GT01

Given that the car is confined to the surface (it's not airborne), as such the normalized velocity (for velocities not 0) should only be a function of the orientation of the vehicle, and as such a function of the surface and, based on the first statements a function of the position. This allows a verification of the coherence of the velocity when compared to the position (normalized to the velocity modulus), seen in figs. 5.20a to 5.20c. As we can see the three runs seem to overlap clearly.



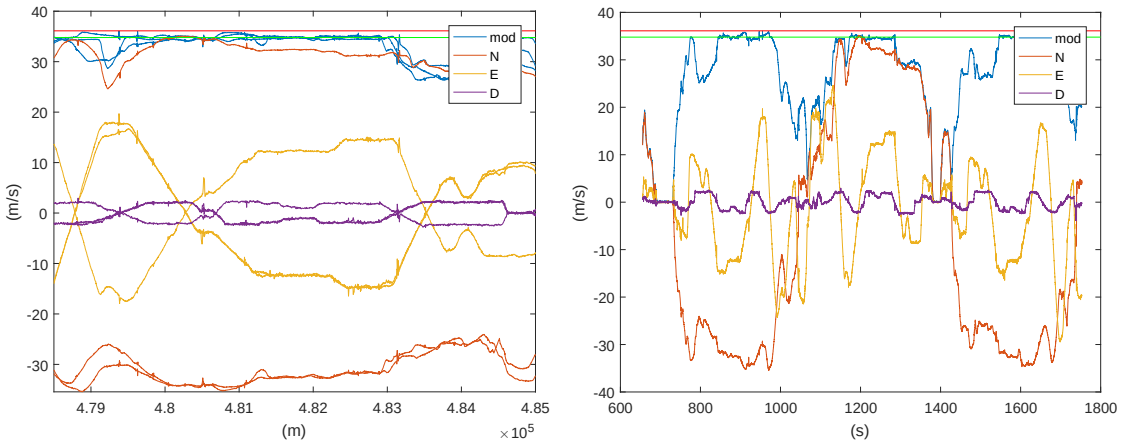
(a) X component

(b) Y component

(c) Z component

Figure 5.20: Normalized absolute velocity, GT01

Because the trajectory is made in both directions, a mirrored velocity about the x-axis is to be expected when considering direction, seen in fig. 5.21a. The velocity modulus should be nearing 36.11 m/s (130 km/h), as it does, figs. 5.21a and 5.21b, the mean value for the flat zones at near maximum velocity being 34.78 m/s.



(a) Velocity for test section

(b) Velocity in time

Figure 5.21: Velocity, GT01

Chapter 6

Launch Procedure

In this section a simple summary of launch safety practices is presented, it also includes information on how to set up the instrumentation system. For complete information please consult [24].

6.1 Launch site and conditions

Optimally the launch site should be flat and clear of any obstacles. The dimensions to be respected can be found in Table 6.1, where clear distance is the separation required between the launch platform and people, and Launch site represents the minimum diameter of the launch site.

The ground immediate adjacent to the launch platform must be clear of flammable materials, as dry grass. The platform itself should have an appropriate blast deflector to protect the platform from the flames. The launch platform should have a sufficiently rigid rod or rail that guarantees that the rocket is launched in the appropriate direction, no rocket should be launch at an angle superior to $20°$ to the vertical direction

Table 6.1: Launch site and clear distance[24]

Motor	Clear distance	Launch site
A	5-9 m	31 m
B	5-9 m	62 m
C	5-9 m	122 m
D	5-9 m	153 m
E	9 m	305 m
F	9 m	305 m
G	9 m	305 m
Two Gs	9 m	305 m
H	30 m	457 m
I	30 m	457 m
J	30 m	457 m
K	61 m	457 m
L	91 m	457 m
M	152 m	457 m
N	305 m	610 m
O	457 m	915 m
P-S	610 m	915 m

Launch should not be tried during winds above 30 km/h, and clouds should be avoided, depending on the rocket flight height.

6.2 Instrumentation system setup

There are two important steps when starting up the instrumentation system. First the calibration of the magnetometer in relation to its current setting. Second the setup and acquisition of satellites by the GPS receiver, it is preferable to let it run until it has sufficient satellites (no less than 4), it could either be done when connected to the battery or to a PC.

The GPS must be set up in the PC uCenter program and then connected to the system.

6.3 Rocket launch setup

This checklist is based on the one in the ESTES manual [18, 65]

Table 6.2: Launch checklist[18, 65]

13	Verify the weather conditions.
12	Connect the GPS to the PC and input the desired settings trough the uCenter app.
11	Connect the GPS to the system and turn the system on.
10	Connect the Xbee to the PC and start the terminal app.
9	Verify the correct reception of data and the GPS connection by monitoring the Xbee.
8	Restart the instruments and let them run in a flat position for a few minutes.
7	Pack the flame-resistant wadding and parachute.
6	Place the nosecone on the rocket.
5	Verify the engine is secure.
4	Install the igniter in the motor
3	Place rocket on the launchpad and attach micro-clips, be certain the safety key is not in the launch controller.
2	Verify area clearance
1	Audible countdown
0	Launch

Chapter 7

Flight Testing

This section includes all test which required the system to be airborne.

7.1 Static Fall

The objective of this test is to verify that the recovery system is functioning correctly, and if the telemetry system effectively records a fall.

7.1.1 Requirements

The requirements include a spot with level difference of approximately 4.5 m, the assembled rocket, including telemetry and recovery systems (but not the motor), field test equipment, consisting of a Xbee explorer USB and a PC are also required. A weight with the same mass as the telemetry system must also be provided. A lifeline to secure the system.

7.1.2 Procedure

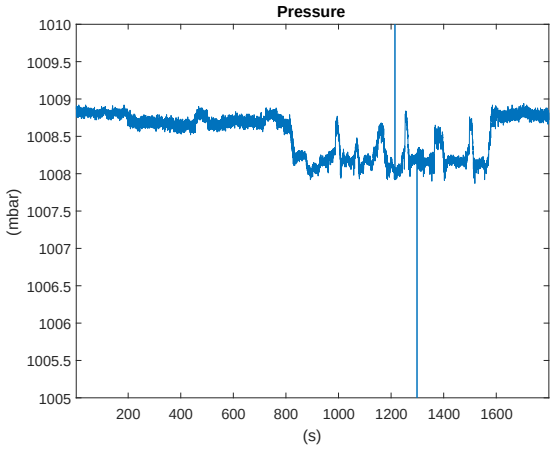
The system is dropped with the weight in the nose but not the instrumentation, parachute extended and the free drop is recorded. A second drop is then made with the instruments in place of the weight and the lifeline attached.

7.1.3 Results

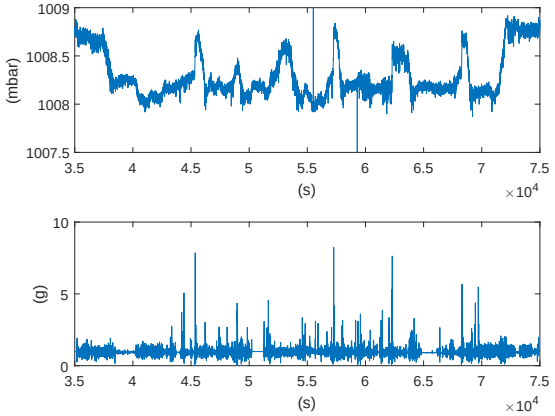
Looking at pressure in fig. 7.1a it is clear there was a time the system remained at a higher altitude, marked by the lower pressure. In fig. 7.1b three detailed rapid changes in pressure are visible, those are the three drops.

Taking a closer look at the drops it is possible to calculate the duration of the fall by looking at acceleration and barometric data. The start of the drop is signalled by a window of no force applied while at low pressure, and the end by a large spike in acceleration, due to the impact or stress of the supporting cable. The fact that in the final moments the acceleration measured is close to 1 g is proof that terminal velocity was achieved and the rocket is in static equilibrium (which in this case means registering 1 g, gravity being counteracted by the drag of the parachute).

In figs. 7.2a to 7.2c the falls are analysed in detail considering the previous information, from where we can ascertain that this 4.5 m drop took on average 1.1 s, which adds up to a terminal velocity of 4.15 m/s, as expected, fig. 3.10b.

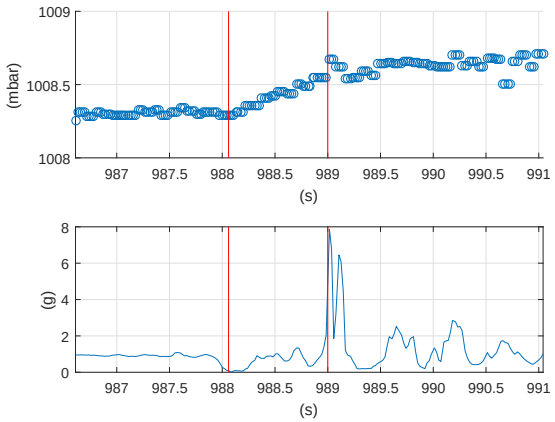


(a) Pressure from 16/05/04 GT02

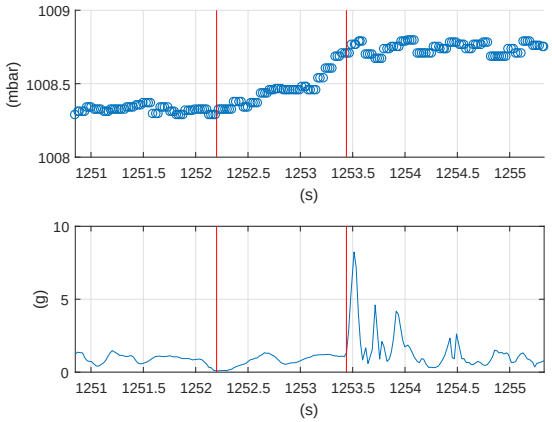


(b) Detail Pressure and acceleration from the drops 16/05/04 GT02

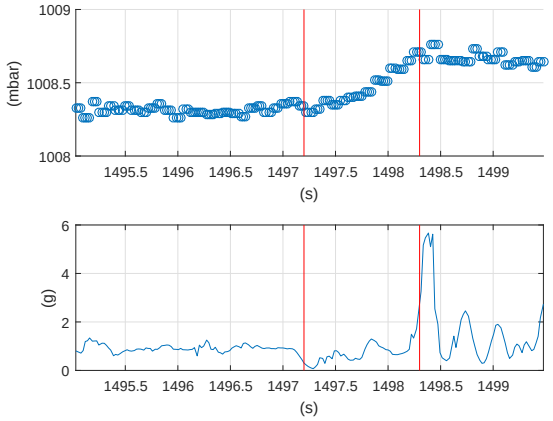
Figure 7.1: Pressure, 2016/05/04 GT02



(a) Drop 1



(b) Drop 2



(c) Drop 3

Figure 7.2: Pressure and acceleration on drops, GT02

Drop 1, fig. 7.2a, was likely cut short by impact as demonstrated by two successive high accelerations. Drop 2 fig. 7.2b, was likewise ended by an impact, but as no high peak follows, it was likely at full extension or close to that. Drop 3, fig. 7.2b, ended simply by cable tension, evident by the smaller terminal acceleration, which is also flatter.

7.2 Aircraft Delivered Fall

This test aims to ascertain the efficacy the recovery system of the flight-test rocket, and to validate the telemetry system in conditions that better resemble an actual rocket flight.

7.2.1 Requirements

An airplane is required, with a window or door that can be opened mid-flight, preferably the test should be conducted over soft ground, to minimize danger from a strong impact. The body of the rocket was not used to avoid damage to this fragile part.

7.2.2 Procedure

The instrumentation is set up and dropped from an airplane at a low altitude, circa 100m.

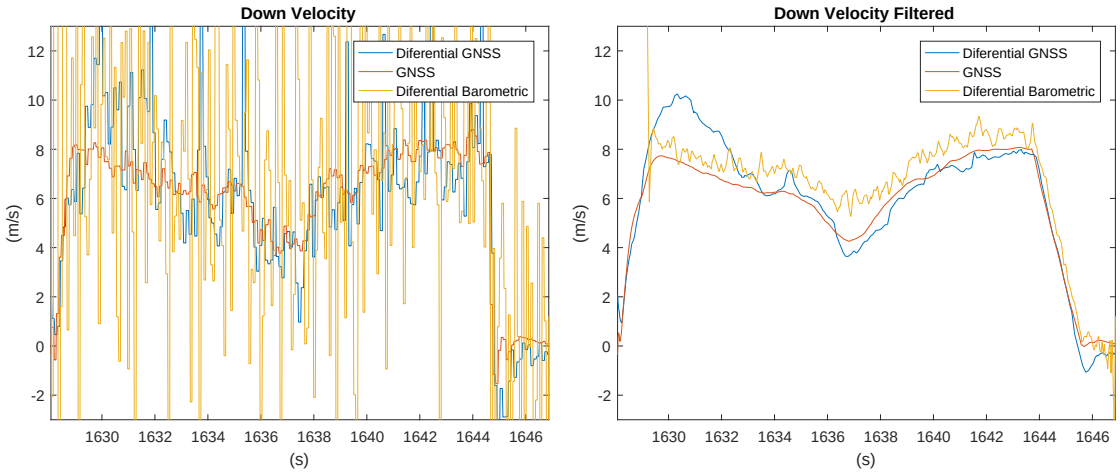
7.2.3 Results

The parachute ruptured, due to operator mistake, the inside of the chute was pointed towards the incoming airflow, causing it to fill with air at the aircrafts airspeed, too much air for the parachute which immediately ruptured, still the chute functioned as a streamer and adequately slowed the rocket, culminating in a safe and successful landing.

As can be seen in fig. 7.3c the drop was from about 105 meters, the initial difference between GNSS and barometric altitude are due to the airflow velocity leaving the aircraft.

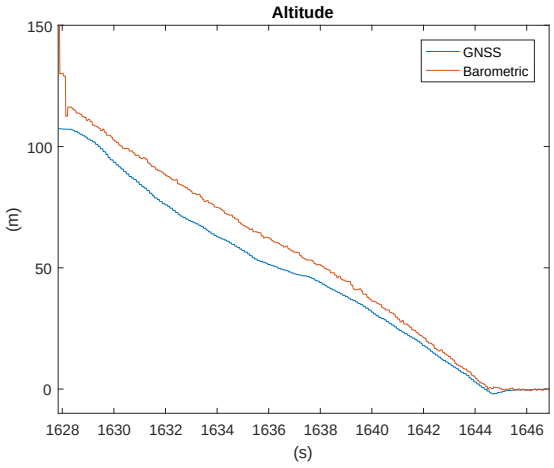
The decent velocity averaged about 7 m/s fig. 7.3b, 2 m/s faster than the expected 5 m/s, yet still slow enough for a safe landing. To validate the down velocity, it was calculated by differentiating GNSS and Barometric altitude, and then passed through a moving average filter with a 0.1 span [66], we can clearly see that the evolution is coherent, and so are the values.

In fig. 7.4 the aircraft trajectory can be seen, as well as the path taken during the decent, it confirms the accuracy of the GNSS position, as the recovery was made at the designated spot, slightly eastward of the north end model aircraft runway (the smaller tarmac runway).



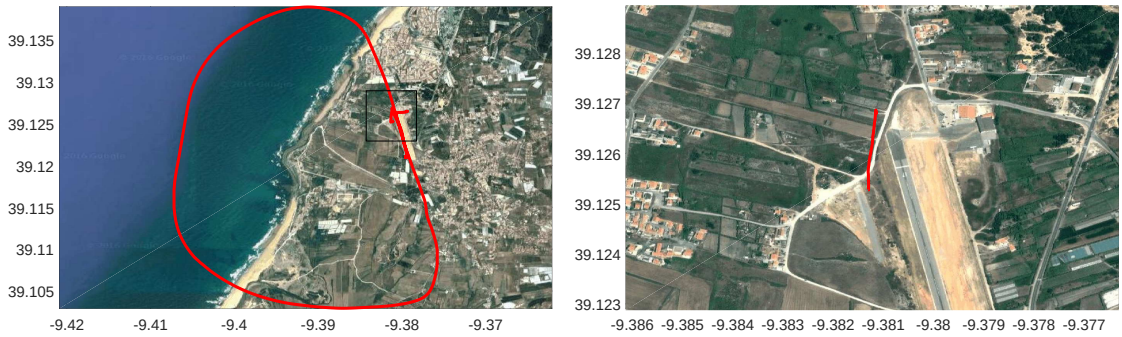
(a) Down Velocity

(b) Filtered Down Velocity



(c) Altitude

Figure 7.3: Vertical velocity and altitude



(a) Aircraft Trajectory

(b) Parachute Drop

Figure 7.4: Planar trajectory

7.3 Flight Test

This final test must measure the efficacy and efficiency of the instrumentation system during a rocket flight. Four flight tests were made with the test rocket, and a fifth on with an Intruder (a much more powerful rocket). This final test failed due to delays in launch caused by engine problems which lead to the system cut-off time to be reached, and it could not be restarted due to safety concerns. This last test still proved useful since, although no data was recorded, the recovery system failed, and so the fall from above 700 m proved the housing was extremely protective, as no component was damaged on impact.

7.3.1 Requirements

The requirements include a field with a 300 m radius, the assembled rocket, motor and recovery systems, the telemetry system. The ground segment equipment, consisting in an Xbee explorer USB and a PC are also required.

7.3.2 Procedure

The system is initialized after a GPS lock. The launch is performed per the recommendations in chapter 6.

7.3.3 Results

Flight tests 2-4 were performed sequentially in the space of 1h, flight test 1 was performed on a different day. For the first and second tests, there was mild wind, and stable, for the last two tests there were strong winds significantly variable at altitude.

During flight test 3 the shock cord was ruptured, as such the body of the rocket tumbled down unrestrained, there were mild damages, easily repairable, but they required a hole to be made in the side of the rocket.

The recorded data was compared to the OpenRockets simulation for the launch conditions, table 7.1.

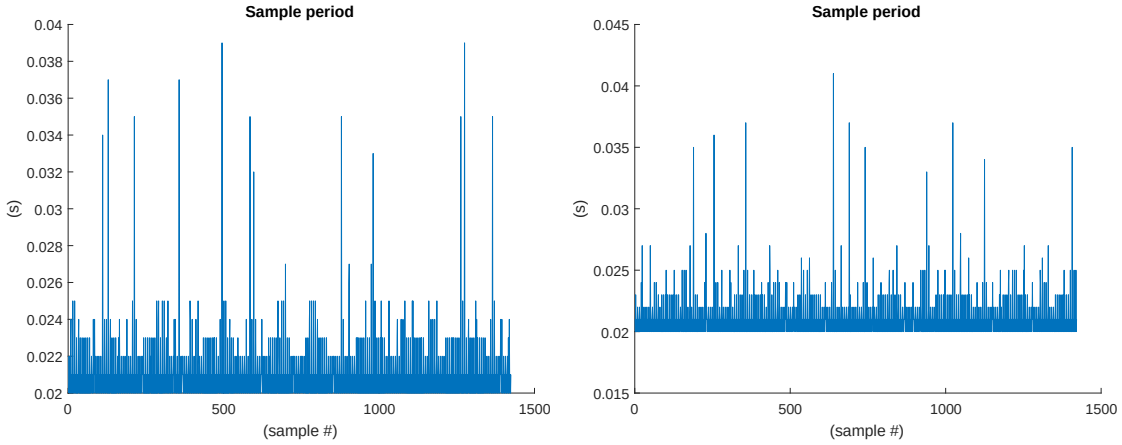
Sample rate

As can be observed in fig. 7.5, the sample rates during flight were always within the limits and expected values, as such the effective sample rate was between 40-50 Hz.

Table 7.1: Launch conditions

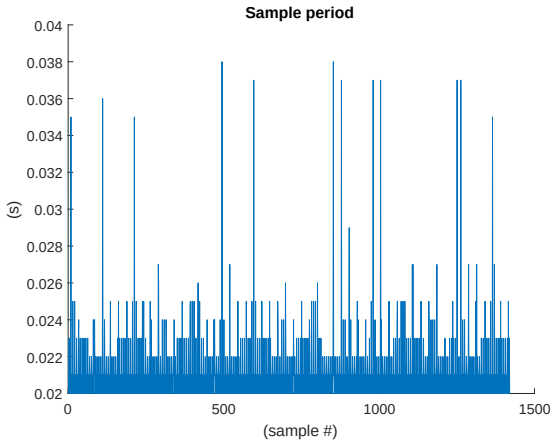
FT	Wind vel.	Wind dir.	Launch Ang.	Temp.
1	5 m/s	320°	15°	35 °C
2	7.5 m/s	225°	5°	25 °C
3	10 m/s	225°	15°	25 °C
4	10 m/s	225°	5°	25 °C

Peaks in sample period are due to SD card page writes and hard writes, which briefly lock the program cycle.

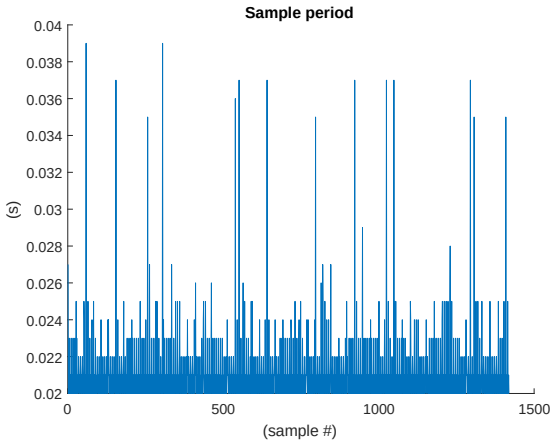


(a) FT1

(b) FT2



(c) FT3



(d) FT4

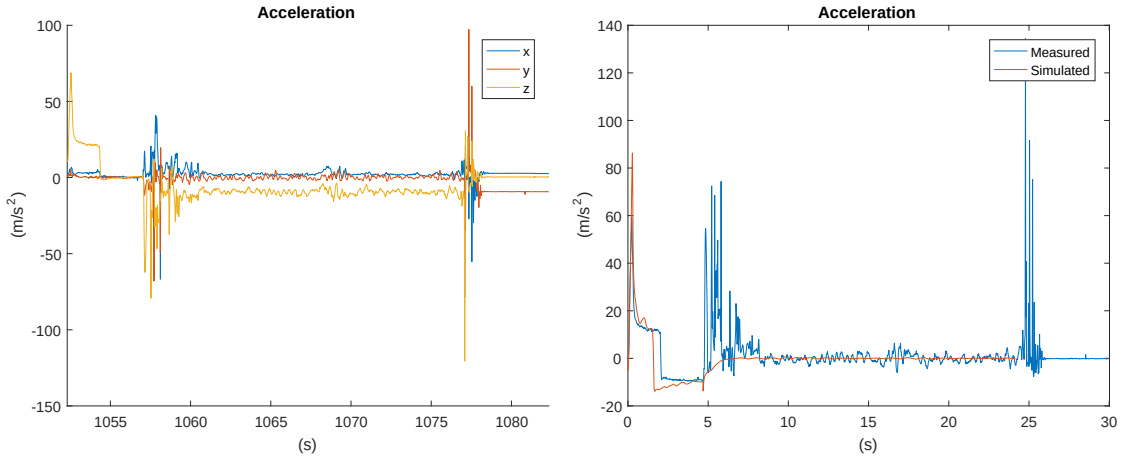
Figure 7.5: Sample period, Flight Tests

Acceleration

Accelerations, fig. 7.6, were always dominant on the Z axis, the values peak at the start of the powered phase (the launch), at chute deployment (ejection charge) and when impacting the ground.

For comparison with simulation results 1 g must be subtracted from measured acceleration since the inertial sensor measures 1 g at rest (due to gravity). In the simulations there are no peaks in ejection and landing, in the former case because the simulation treats the rocket as a rigid body, as such the nose ejection is not represented, the latter because simulation does not include ground impact as it ends at touchdown.

The measured acceleration can be used to easily separate the three phases of rocket flight, at launch it features accelerations several times that of gravity, during the costing phase it should show a value close to zero (since the only force at work is gravity, and the rocket is "free-falling" at this stage, mimicking 0 g) and the recovery phase, where the acceleration measured should be close to 1 g after stabilization, which signifies the drag force equals gravity for a constant speed decent.



(a) Acceleration

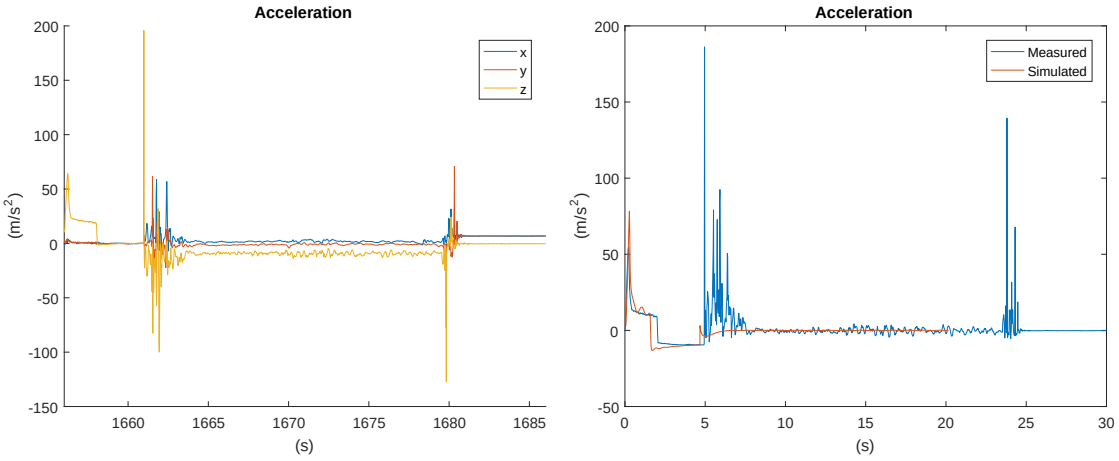
(b) Measured and simulated vert. acc.

Figure 7.6: Acceleration, Flight Test 1

For the first flight test it can be observed in fig. 7.6b that the rocket engine actually had much lower acceleration peak, yet a longer burn time, and as such a similar total impulse, as observed in very close maximum velocities and altitudes.

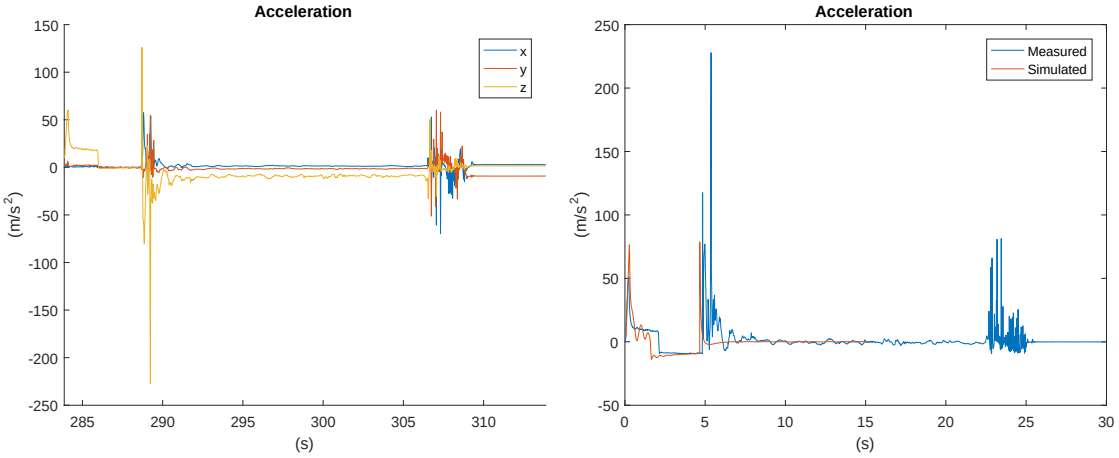
For the next flight-test the parachute and recovery connection used was different (heavier) one, approximatively 20g, as such acceleration should be proportionately reduced, as is indeed the case. The acceleration for the three other flight-tests was similar with differences due mostly to motor construction.

All motors appeared to have longer burn times and lower maximum thrust fig. 7.23, more similar to D11 engines than D12 [18].



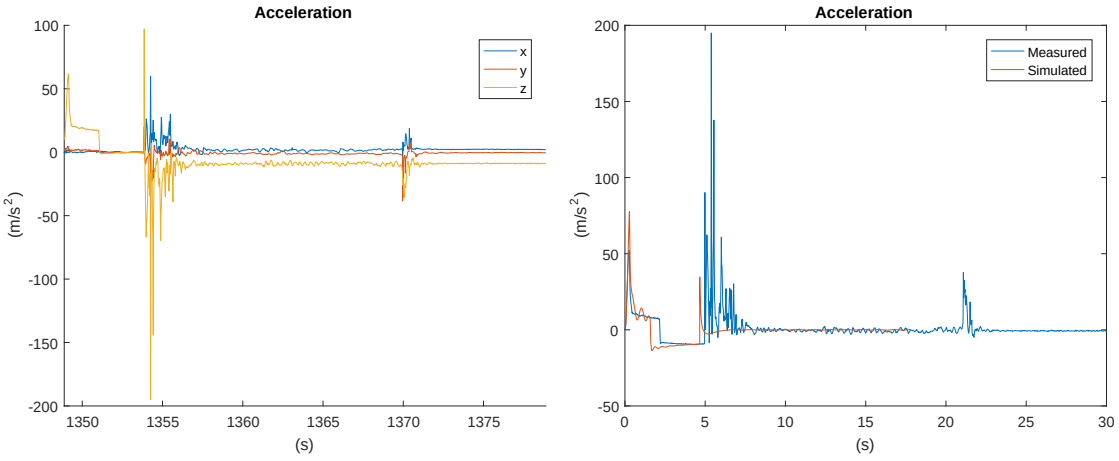
(a) Acceleration (b) Measured and simulated vert. acc.

Figure 7.7: Acceleration, Flight Test 2



(a) Acceleration (b) Measured and simulated vert. acc.

Figure 7.8: Acceleration, Flight Test 3



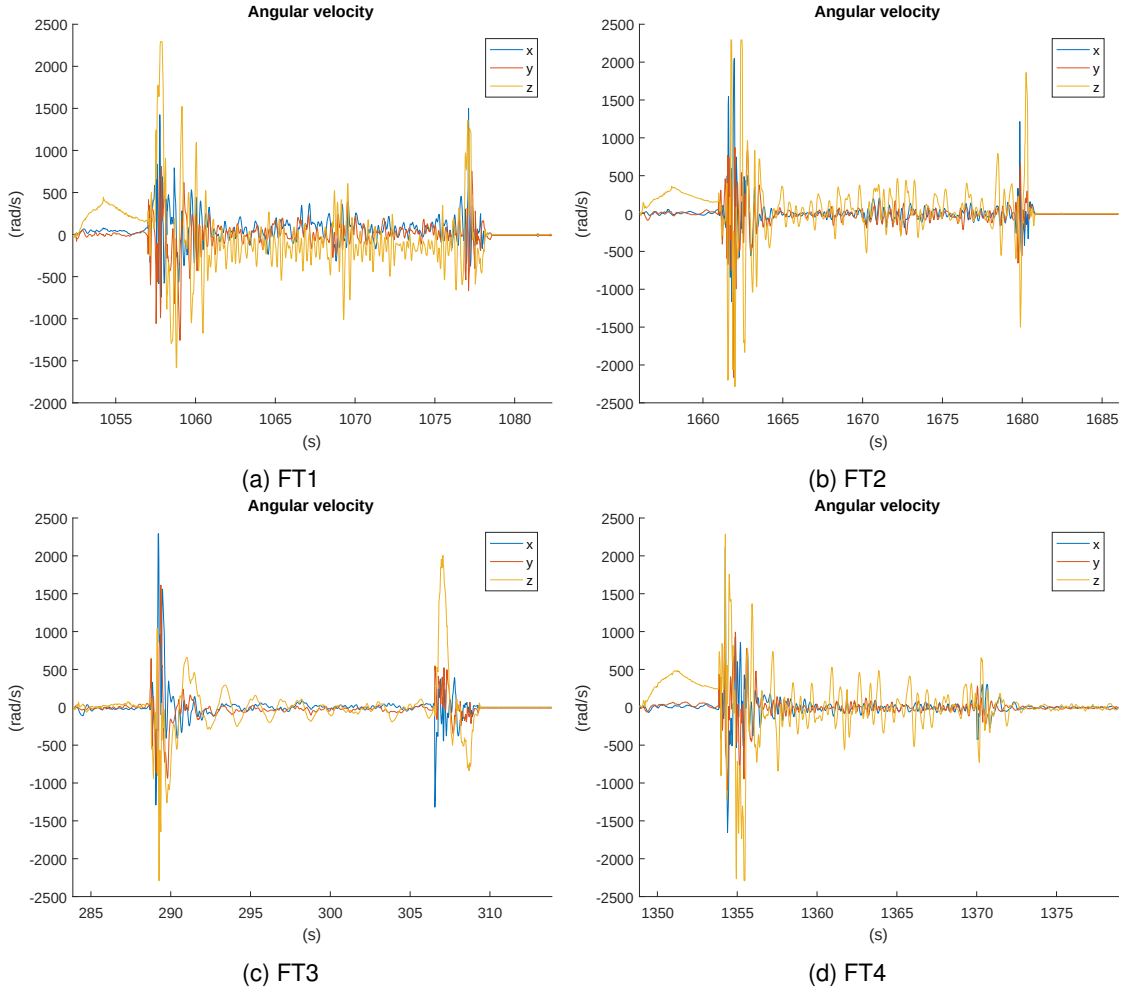
(a) Acceleration (b) Measured and simulated vert. acc.

Figure 7.9: Acceleration, Flight Test 4

Angular Velocity

There is a clear dominance of the z component of angular velocity, consistent with the stabilizing spin usual in aerodynamic stabilized rockets. This spin increases during the powered phase of flight and then decreases as drag and gravity become the dominant forces during the coasting phase.

During the recovery phase there is no clear spin, the movement of the nosecone is erratic, there are however two clear peaks, one at the ejection charge deployment and one at landing, afterwards angular acceleration is zero.



(a) FT1

(b) FT2

(c) FT3

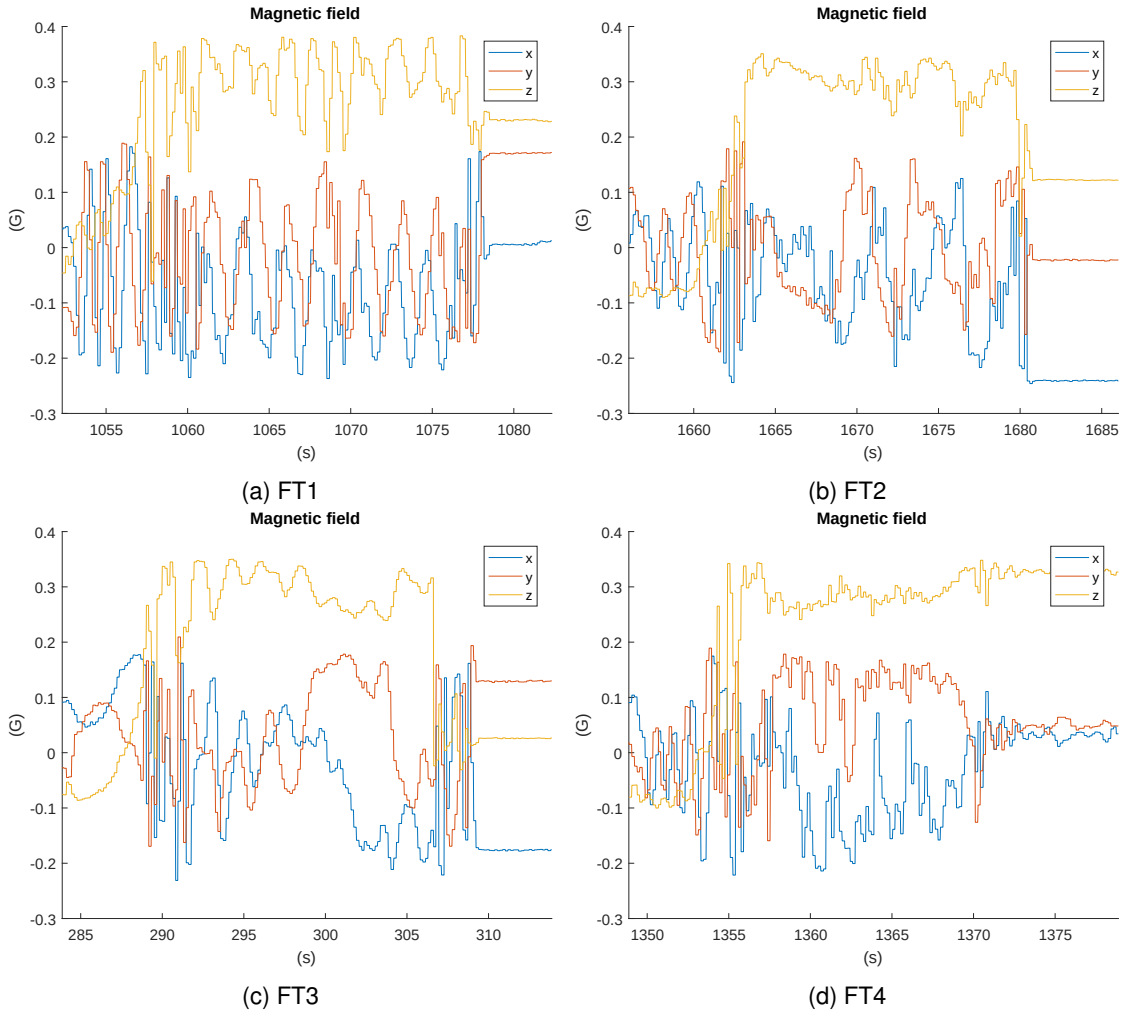
(d) FT4

Figure 7.10: Angular velocity, Flight Tests

Flight tests 1,2 and 4 present the expected results, a pyramid like dominance of Z angular velocity during the powered and coasting phase, a noise like distribution after chute deployment and 0 value after touchdown, all accompanied by the expected peaks.

Flight test 3 was unique as there was no stabilizing spin which, as will be seen, caused the rocket to take a much higher pitch angle after leaving the launching rod. This launch (coincidentally) also saw the shock-cord disconnected, thus there is much reduced noise like wobbling during recovery, as the nosecone was able to sit directly below the parachute without impacting the body of the rocket, causing most angular velocity to be about the Z axis, consistent with successive twisting and untwisting of the shroud lines.

Not much can be said about the magnetic field, it varies coherently with yaw, in a bi-dimensional approximation.



(a) FT1

(b) FT2

(c) FT3

(d) FT4

Figure 7.11: Magnetic field, Flight Tests

Attitude

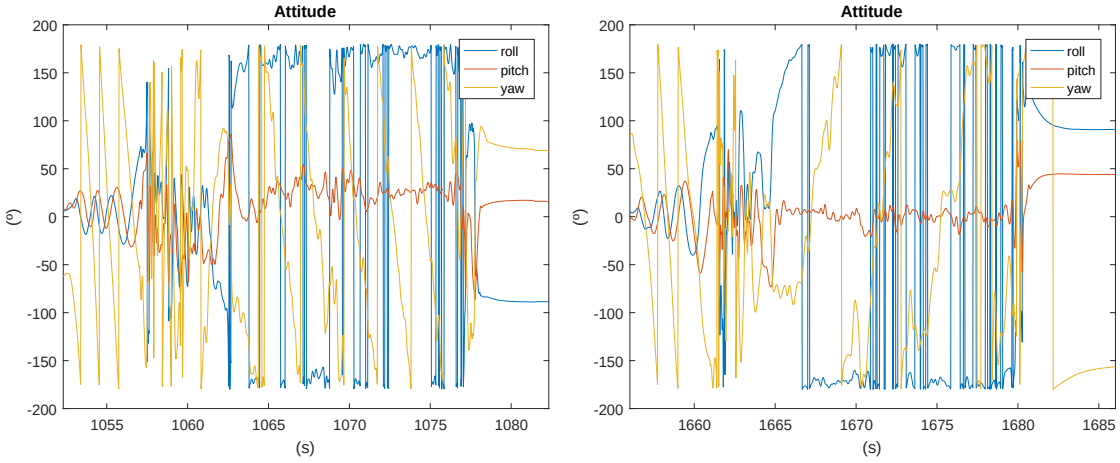
During the powered and costing phase the spin clearly translates to several revolutions in yaw, and a near phase opposed variation between pitch and roll, increasing in magnitude as the stabilizers direct the rocket upwind, seen in fig. 7.12a. These variations of roll and yaw also indicate a small precession, expected in not actively stabilized rocket not perfectly balanced launched from a flexible rod. [14]

During the coasting phase, the AHRS resilience on accelerometer for attitude determination means that at this stage the 0 g like state limits precision significantly for pitch and roll, there is no accelerometer based correction. Yaw remains unaffected due to it being calculated based on the magnetometer reading.

During the recovery phase there should be again a clear spin visible, albeit a slower one. As seen in the angular velocity, there is a near erratic movement when it comes to pitch and roll, but now clearly at around $\pm 180°$, the nose pointing roughly to ground.

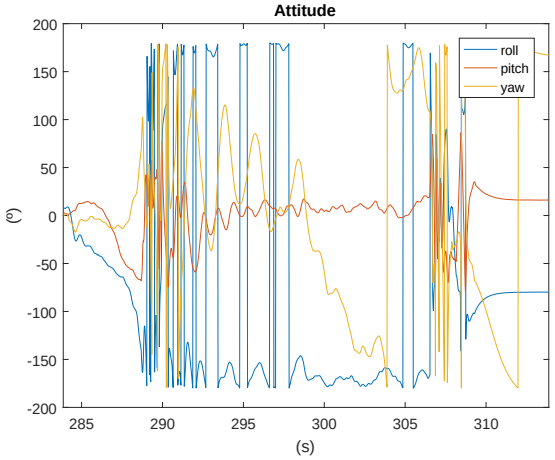
Again, flight tests 1, 2 and 4 presented the expected behaviour, erratic close to $0°$ pitch and close to $180°$ roll and variable yaw. The initial launch angle somewhat increases during flight as the rocket directs itself upwind thanks to stabilization.

The outlier is again flight test 3, since there is no stabilizing spin the rocket has no phase like alternation between pitch and roll, instead both values increase separately, much faster than before, yet still upwind, this is due to the smaller angular momentum which make the fin induced upwind rotation much

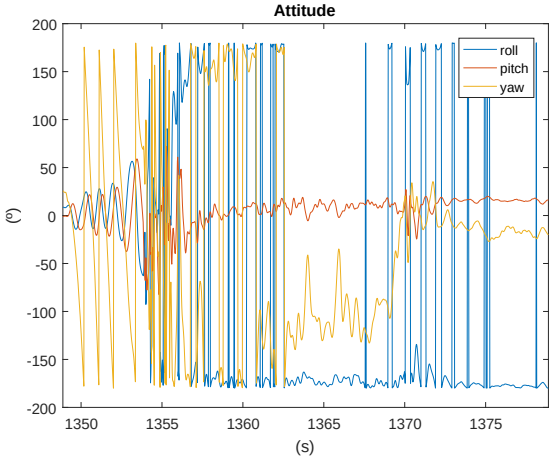


(a) FT1

(b) FT2



(c) FT3



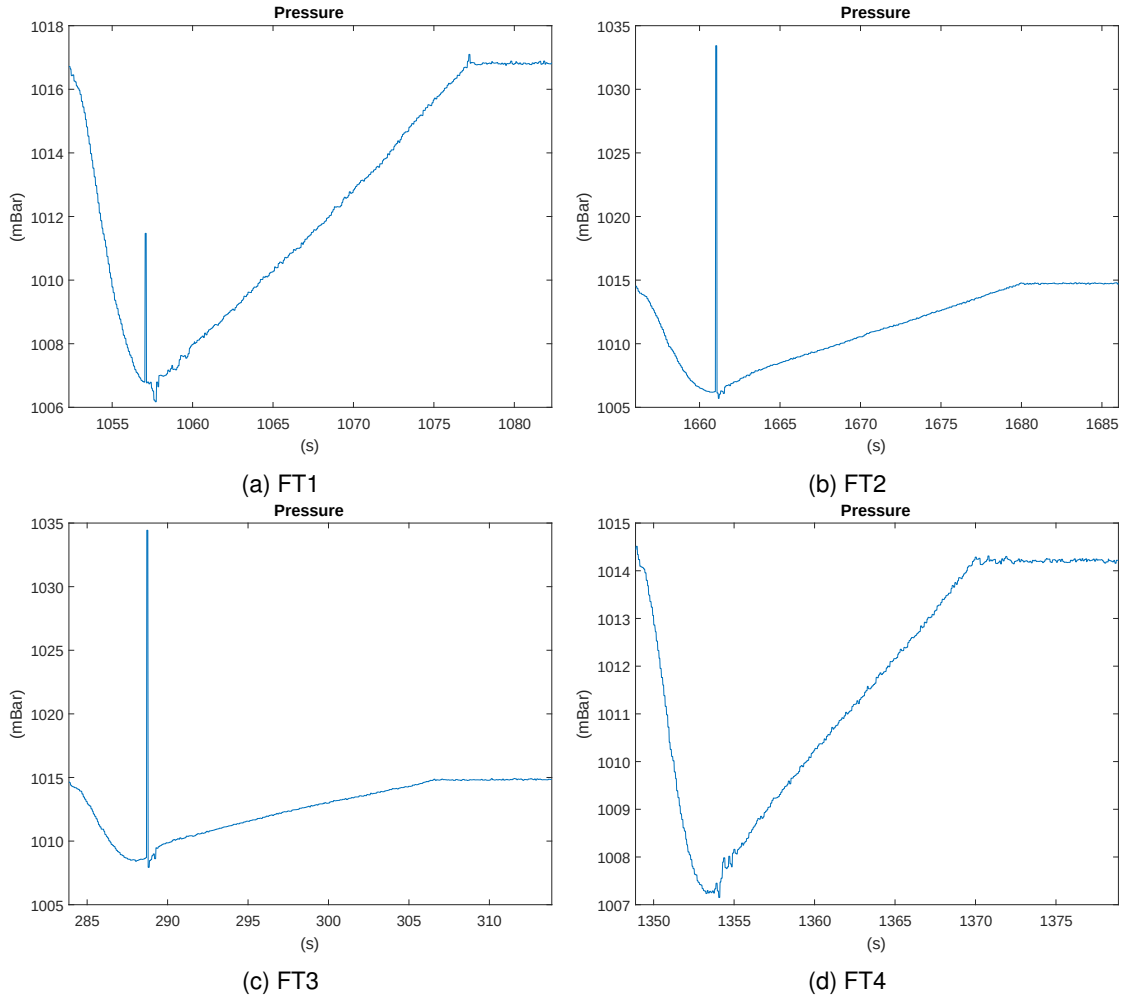
(d) FT4

Figure 7.12: Pitch, Roll, Yaw, Flight Tests

more pronounced. While in other flight test the angle to the vertical never surpassed $30°$ until velocity was near 0, on the third flight test very quickly this angle rose to nearly $75°$.

Pressure and temperature

Pressure (fig. 7.13) is expected to lower during both powered and costing phases with an inflection in-between as acceleration reaches 0 and velocity begins to decline. During the decent it increases again, approximately reaching the same value it had before launch. During recovery, there should be an altitude unrelated increase in pressure due to the more exposed intake being hit by the wind, which also causes increased variation.



(a) FT1 (b) FT2

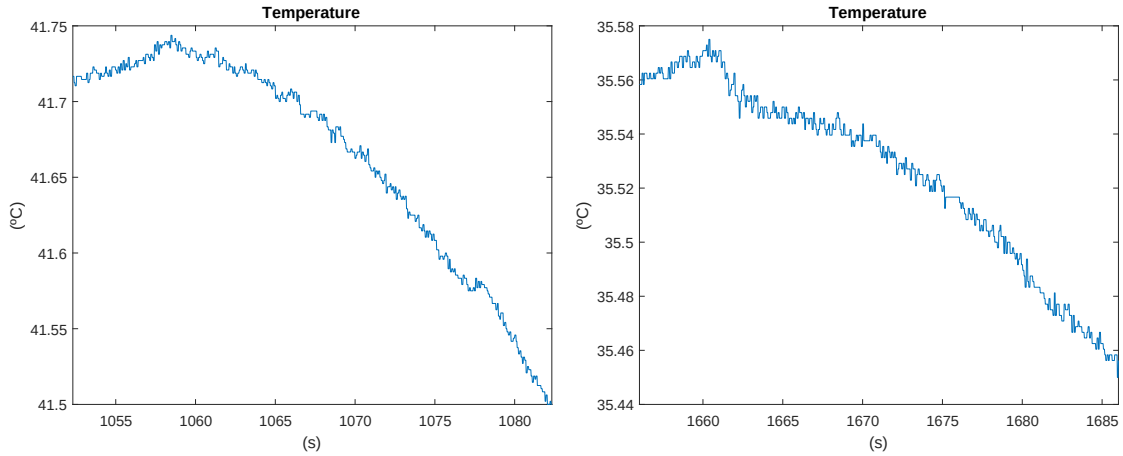
(c) FT3 (d) FT4

Figure 7.13: Pressure, Flight Tests

In flight tests 1-3 there is a clear peak during the ejection charge, caused by the pressure wave of the explosion. The differing values of the pressure spike might be due to less wadding in the rocket or simply manufacture differences in the ejection charge. There were no peaks in the last test due to the hole on the side of the rocket and the partial coverage of the intake due to repairs.

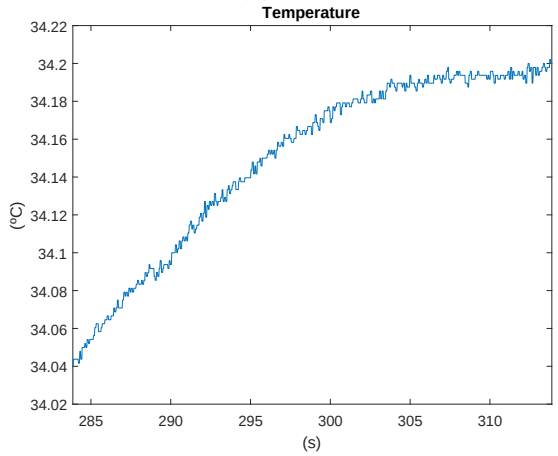
Temperature, fig. 7.14, remains relatively stable, varying at most 0.25 °C, the temperature is noticeably higher than the ambient one (35 or 25 °C). The decrease in temperature after ejection is caused by increased airflow in the nosecone, gradually meeting the ambient temperature. This high value is due to both solar radiation and joule effect in a confined space.

Flight tests 1,2 and 4 show an increase or maintenance of temperature until recovery and then a sharp decrease a due to exposure to the elements, this does not happen in flight test 3 because the whole system was opened for reset between flight test and the temperature increases the whole time influenced by the effects mentioned in the previous paragraph.

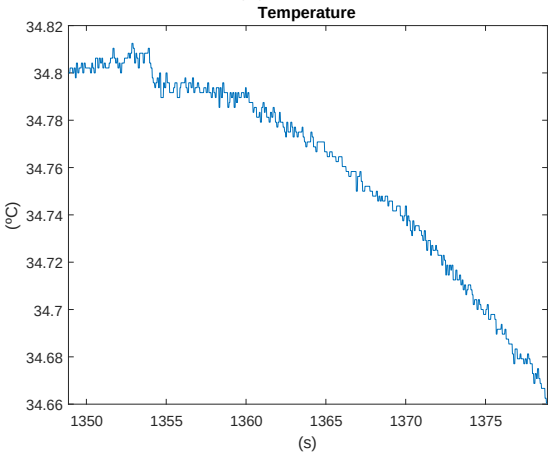


(a) FT1

(b) FT2



(c) FT3



(d) FT4

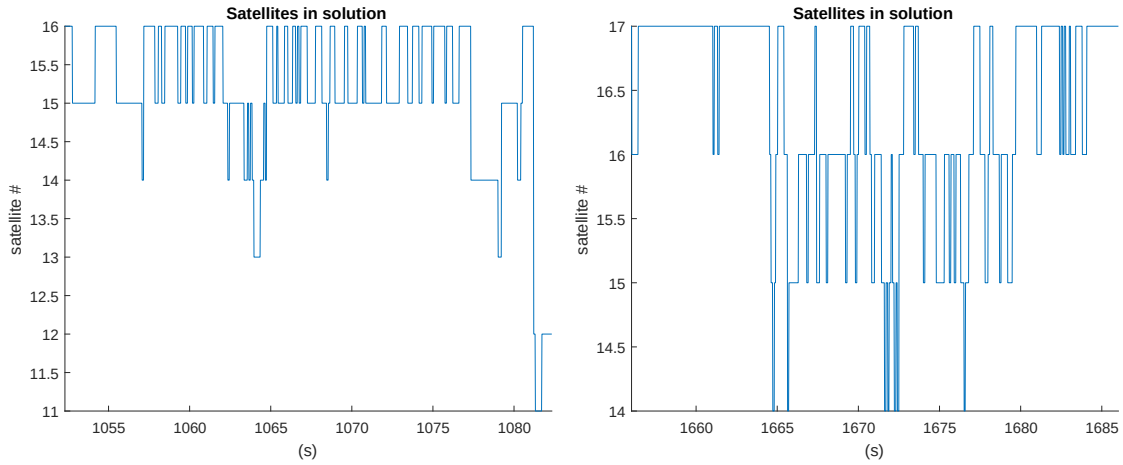
Figure 7.14: Temperature, Flight Tests

Number of satellites

During most of the flight 15 satellites were used to compute the GPS data, as such the results should be very accurate, fig. 7.15.

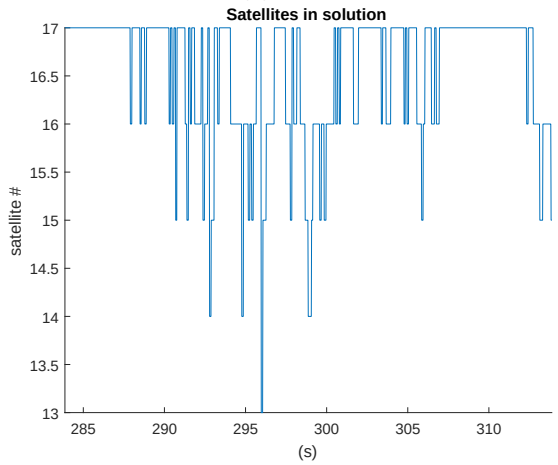
In flight test 1, shortly after landing, most satellites were lost, due to an impact between the GPS antenna and the walls of the nosecone, this caused additional errors afterwards, as seen in the altitude figures. To prevent this from ever happening again an additional protection and support casing was added to the GPS receiver. The other flight tests had more satellites and no disconnections.

During flight the number of satellites never dropped below 12, which indicates GPS measurements should be very accurate.

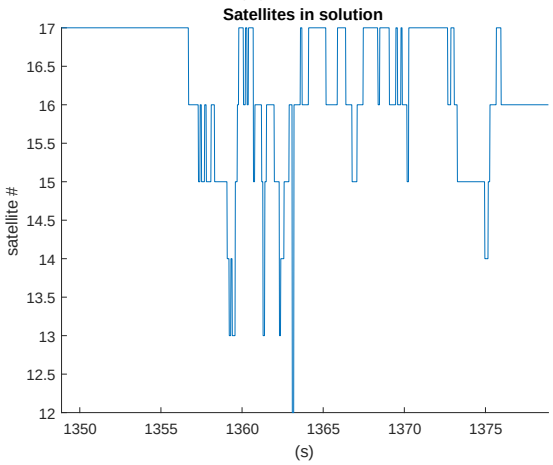


(a) FT1

(b) FT2



(c) FT3



(d) FT4

Figure 7.15: Satellites used in solution, Flight Tests

Altitude

Altitude, fig. 7.16, behaves opposite to pressure (as expected), it increases during the powered and coasting phases with an inflection in between, as explained in the pressure section, then it slowly descends towards the starting value as the chute take takes over during recovery.

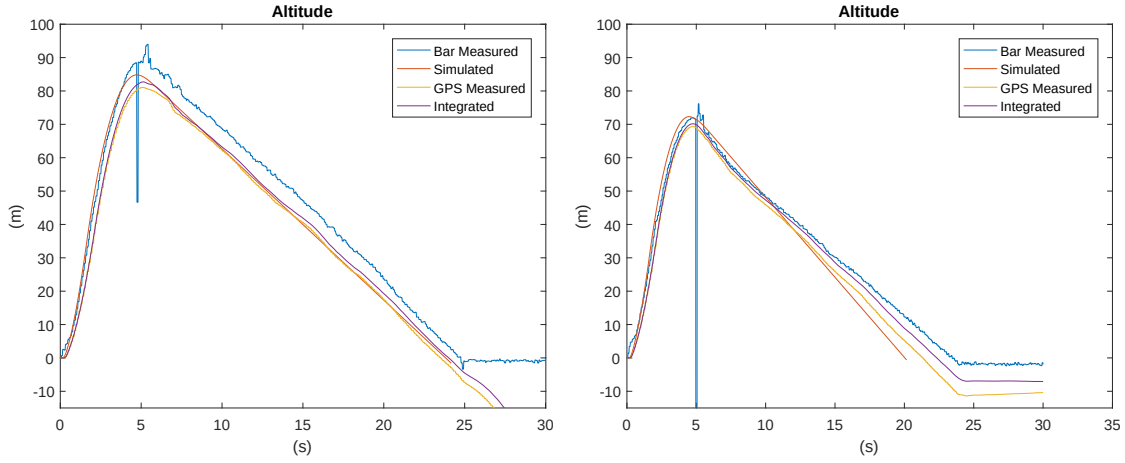
There are two methods here used to measure altitude, barometric calculations based on pressure and temperature, and GNSS altitude, calculated via pseudo distances and times. An additional altitude is presented by GNSS vertical velocity integration in post processing, since this velocity is derived from Doppler effect calculations it should provide a third independent form of altitude measurement [23].

Due to the Joule and greenhouse effects on the temperature sensors, local weather readings were used for barometric calculations. table 7.1

All the calculated altitudes and the respective simulations are close together during ascent, for flight tests 2-4 the descent differs greatly due to the use of a new nylon parachute, which is more effective, and also due to the winds at altitude which were more intense.

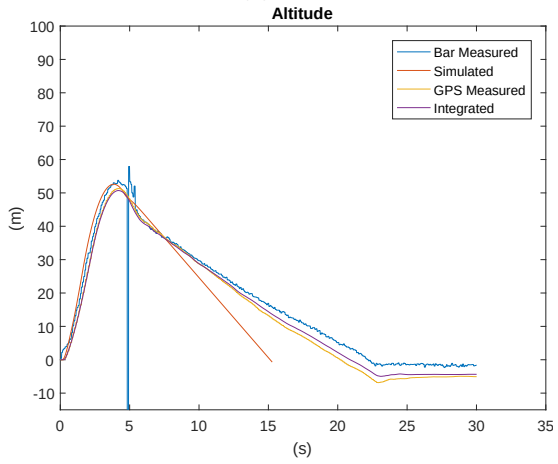
In flight test 1, after the loss of satellite the altitude continued to descend until it reached a relatively stable yet illogical value. In this test the static port was larger than in subsequent tests which may account for the increased difference in barometric altitude after ejection.

GPS slightly overestimates landing altitude, this is most likely due to having surpassed the $4g$ limits of the internal filters, which means that integrated vertical velocity, submitted to no filters, is likely a better estimate for short flights.

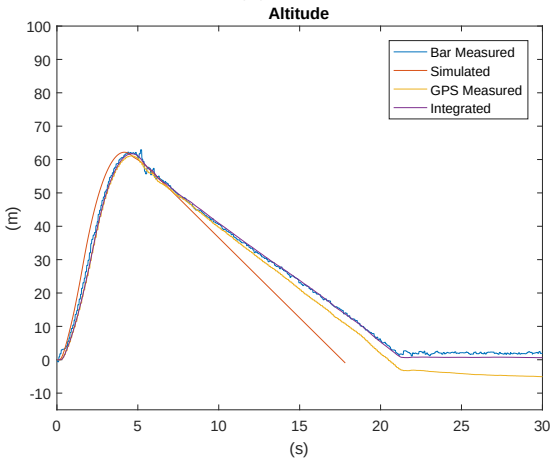


(a) FT1

(b) FT2



(c) FT3



(d) FT4

Figure 7.16: Altitude, Flight Tests

Velocity

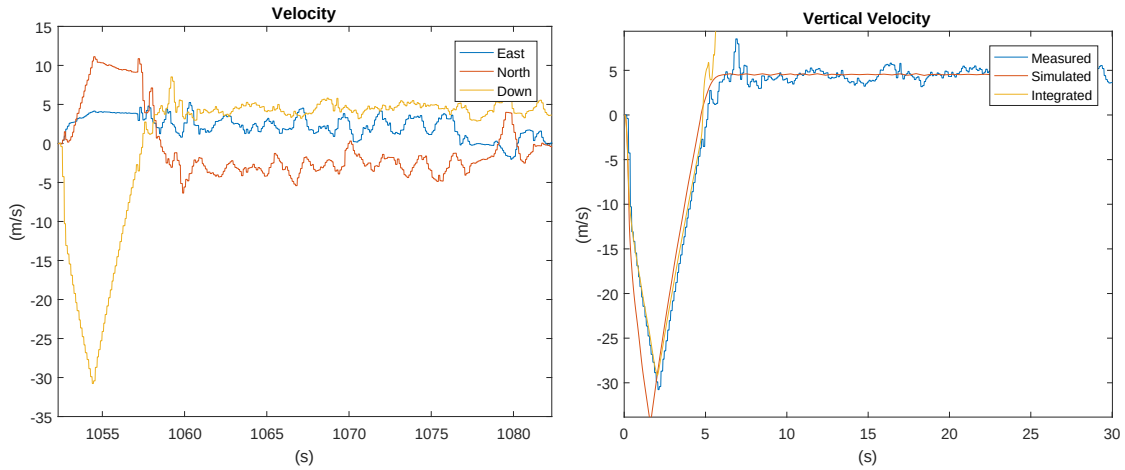
Velocity was clearly dominated by the vertical component, which increased consistently during the powered phase, and then gradually reached zero. Decent velocity during the recovery phase was approximately 5 m/s, as designed.

Vertical velocity during the powered phase increases steadily until it peaks and marks the start of the coasting stage. During decent it should remain stable at 5 m/s, or the designed decent velocity.

In post processing, during the powered phase another measurement vertical velocity calculated by integrating acceleration taking attitude into account is compared to the measured one, in every case they closely matched.

Vertical velocities were in accordance with the simulations, always underestimated, this is due to the smaller acceleration peak measure, which again is likely due to engine variation in construction, all coherent with the recorded values.

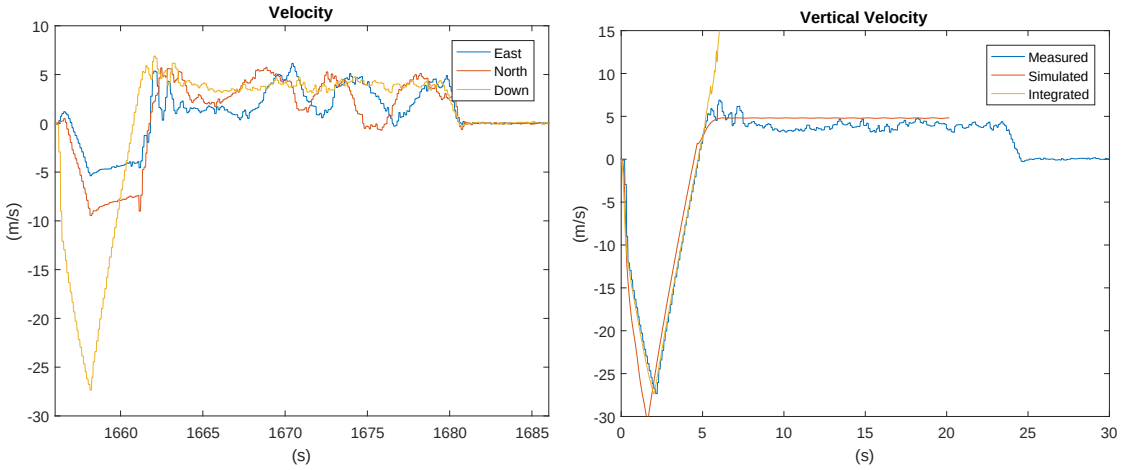
For flight test 1, during all flight, North velocity was dominant in the plane, consistent with a rocket launched upwind when wind direction is 320°. East velocity was nearly constant at about 2.5 m/s. All other flights also presentment coherent values for horizontal velocity during the recovery phase compared to the prevailing wind.



(a) Velocity

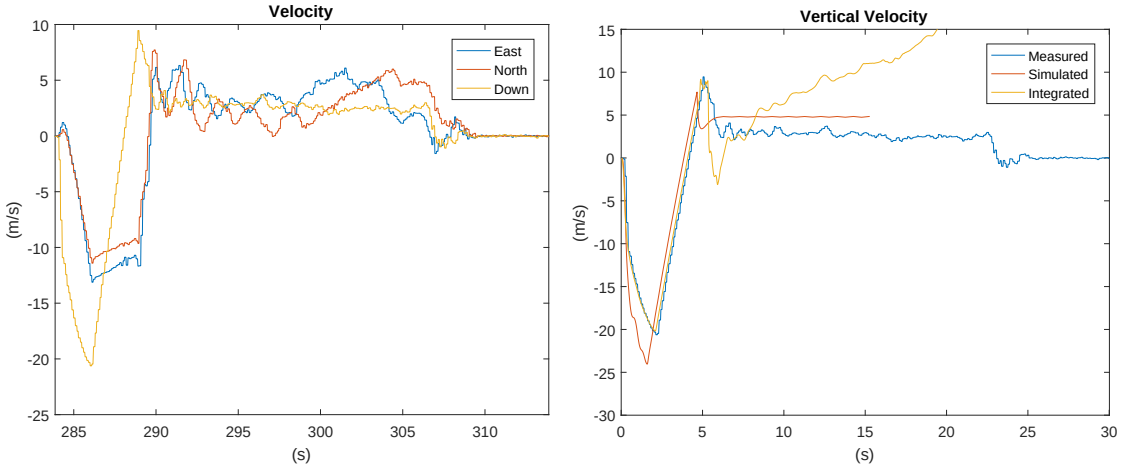
(b) Comparison

Figure 7.17: Velocity, Flight Tests



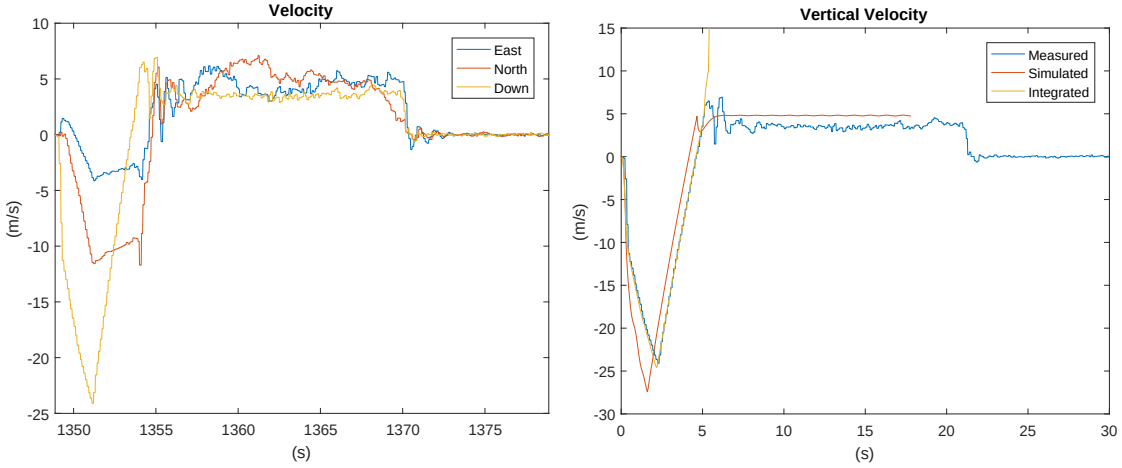
(a) Velocity (b) Comparison

Figure 7.18: Velocity, Flight Test 2



(a) Velocity (b) Comparison

Figure 7.19: Velocity, Flight Test 3



(a) Velocity (b) Comparison

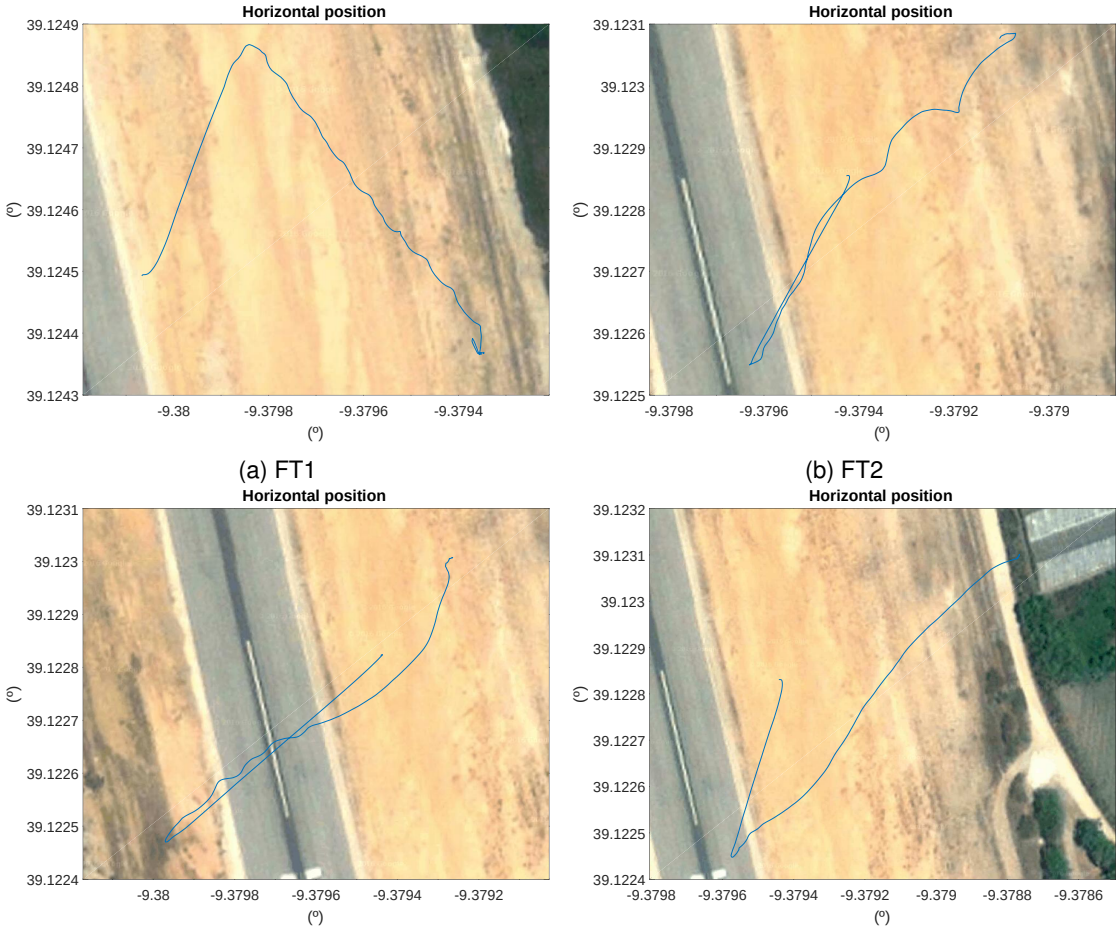
Figure 7.20: Velocity, Flight Test 4

Horizontal position

All trajectories are mostly vertical; the horizontal displacement is due to the rockets being directed up-wind of the rocket launch and the wind effect after parachute deployment.

In the flight test the position determination is mostly correct as the launch and recovery sites were very close to the ones marked in the Google maps overlay fig. 7.21.

In flight tests 2-4 it the trajectory clearly mirrors the recorded 225° wind direction. The only slight incoherence is in the powered phase of flight test 4, probably due to the reduced launch angle and the stronger wind at the time, resulting in less rocket velocity wind alignment, causing a lesser drift towards the south-west. Flight test 1 matches a 320° wind.



(a) FT1

(b) FT2

(c) FT3

(d) FT4

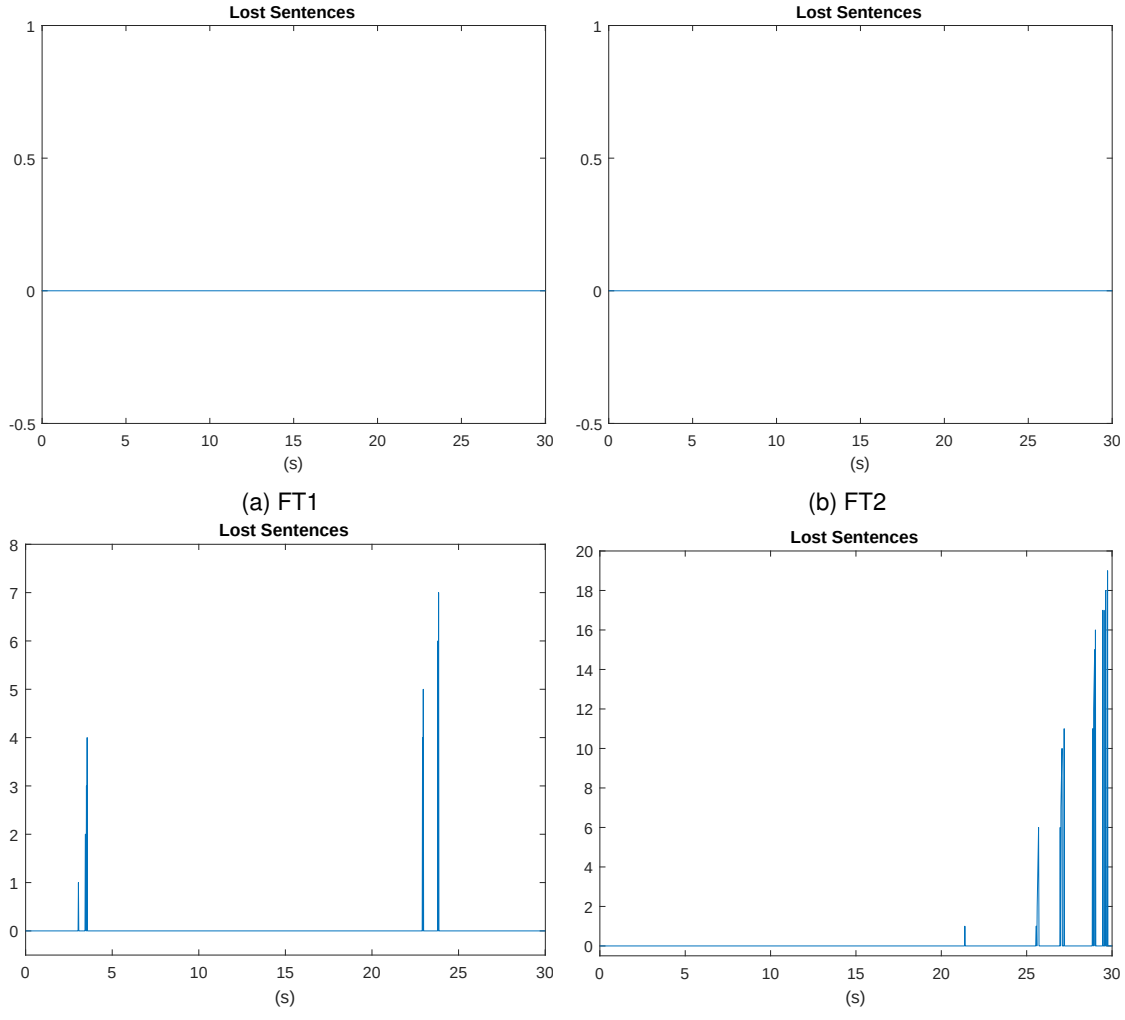
Figure 7.21: Google maps overlay, Flight Test

Telemetry data reception

To verify the correct reception of telemetry data, both csv files are compared, for the interval of flight. As displayed in figure fig. 7.22.

For flight test one and two we can see that all the values were received correctly. Flight test three, where recovery partially failed, presents a 4 sentence loss when the nose dive starts, probably due to having the full length of the rocket between the Xbees, the latter losses are probably due to people blocking the antenna while moving for recovery, the cumulative total is 9 sentence loss. Flight test four showed significant loss (19 sentences), mostly towards the end, likely due to the dense reeds where the rocket landed. It's worth noting that due to some water drops during flight tests 3 and 4 the telemetry station was covered and in a box, which would further disrupt reception.

Overall, in the more than 1400 samples taken for each flight less than 2% were lost, excluding those after touchdown the number falls below 1%.



(a) FT1 (b) FT2

(c) FT3 (d) FT4

Figure 7.22: Difference between time in telemetry data and local data

Overview

Analysing both altitude and vertical velocity for the 4 flight tests, we can get a good overview of the efficiency of the instrumentation system. Velocity is obtained from GNSS and integration of acceleration,

altitude is extracted from GNSS, barometer and integration of GNSS velocity. These calculations were performed in the powered and coasting phase, until the apogee is reached or the ejection charge is active.

The deviations are relative small, never exceeding 5 m for altitude and 1 m/s in velocity.

For the actual analysis of a specific rocket it should be noted that these derived measurements have little use, precision of the system is exclusively dependent on the instrument precision, there is no point in integrating and accumulating error, or differentiating and expanding noise when an accurate sensor exists to take these measurements, however care should be taken (as described in chapter 5) to calibrate sensors before flight-testing to better control error and so to improve performance.

As can be seen in figs. 7.17b, 7.18b, 7.19b and 7.20b, the recorded data matched the OpenRocket simulations for the local conditions (launch angle and wind). The differences observed seem to stem from inaccuracies when building the engines, this is apparent in fig. 7.23, where powered stage accelerations are compared, these profiles (lower but more prolonged) match D11 engines closer than D12 engines, they also seem to be in the lower end of the D segment (lower total impulse). All of this shows why the rocket achieved similar heights (longer burn but lower impulse) but lower velocities (lower peak thrust).

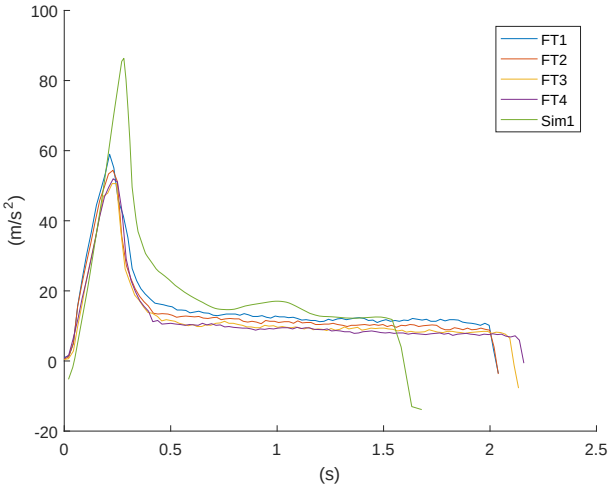


Figure 7.23: Comparison of acceleration during powered phase

Chapter 8

Conclusions

The parameters chosen adequately characterize the rocket, from them it is possible to review all the rockets' flight and reconstruct it.

The instrumentation system developed works in accordance to the design directives, it is light, compact, COTS and modular. The housing constructed for it is steady and works well with the chosen rocket. This was further proved by its survival to direct impact and falls from several meters.

The instruments meet the specifications required, that were defined in order to document the rocket flight. The on-board recording and telemetry work as intended and offer redundancy. The Arduino proved to be a small reliable and adaptive platform, although its limited memory and processing power were effectively a major system limitation, offset by its low power consumption and reliance on light lithium polymer batteries. Telemetry was guaranteed by a Xbee, which worked as expected with excellent communication efficacy and efficiency at the tested ranges. The Pololu proved more than able to measure all the required data, in a very compact package. Choosing a uBlox GPS proved essential in system optimization as the byte base structure it uses permitted much simpler and more efficient parsing. The microSD card and its board proved reliable as no data was lost or corrupted, although its lock time hampers sample rate.

The housing developed proved durable, and so did the rocket, surviving all test flights with only minor damaged, all repairable in the field.

The software performed as expected, in the Arduino it pushes it to the limits and allows for real-time AHRS calculations, local data recording and telemetry, while remaining the 40-50 Hz sample interval, the need to shut down at set times to ensure no data corruption is limiting but can easily be offset by adequate planing while boosting reliability. The ground tracking station is able to simultaneously record the raw data stream, interpret it, apply the correct factors, form the required 30 s data vectors and display it, all while keeping the interface responsive, the UI is not the most attractive but it is simple, unobtrusive and works well on the field.

The ground tests were effective in calibrating the sensors, and validating the system for airborne use. These tests verified the sensors were accurate enough four our instrumentation goals.

The flight test very closely documented the rocket flight, they showed the system performed well in its design environment and so validate the whole design objective.

In essence, the system is able to record the dynamics of the rocket, its acceleration, angular velocity, attitude, linear velocity and position (including altitude), in addition it gathers pressure, thermal and magnetic data from it's environment, sends it all through an antenna and stores it in a micro SD card. The telemetry base station records and presents in real time all the data it receives in the form of graphs of the most important parameters and an artificial horizon and compass for attitude.

8.1 Achievements

The instrumentation system size and weight permit it to work in a varied number of environments, and the modular COTS design allows for easy replacement of any damaged parts.

The quality of the used instruments and the swiftness of the Arduino sketch make the system able to adequately model highly dynamic systems, such as the rocket. The sample rate borders the Arduino processing power, meaning it could not be improved without substantial hardware changes.

The measured parameters are useful for both real-time and post processing analysis of a rocket flight.

The ground tracking station makes for much simpler recovery of the rocket after touchdown.

8.2 Future Work

Using a more powerful microprocessor it should be possible to improve the data rate. To achieve this one recommendation would be to use circular buffers for serial communications and the First In, First Out (FIFO) buffers in the Pololu sensors.

Based in this instrumentation system it would be fairly easy to add control to a rocket, or indeed any aerial vehicle, such as an airplane or a quad-copter This would be a good place to start development of a full-sized sounding rocket.

The ground tracking station software could be enhanced into a more user-friendly application. A mobile app and Wi-Fi connectivity cold be added.

Bibliography

[1] T. Benson. (Apr. 26, 2016). Brief history of rockets, NASA, [Online]. Available: http://www.grc.nasa.gov/WWW/k-12/TRC/Rockets/history_of_rockets.html.

[2] NASA Sounding Rockets Program Office, *Nasa sounding rocket program handbook*, NASA, Goddard Space Flight Centrel, Jun. 2005.

[3] G. Seibert, *The History of Sounding Rockets and Their Contribution to European Space Research*. Noordwijk, Netherlands: ESA Publications division, 2006.

[4] J. Prades, A. Ghiotto, E. Kerhervé, and K. Wu, "Broadband sounding rocket antenna for dual-band telemetric and payload data transmission", *IEEE Antennas and Wireless Propagation Letters*, vol. 15, pp. 540–543, 2016, ISSN: 1536-1225. DOI: 10.1109/LAWP.2015.2457338.

[5] P. c. Tsai, H. p. Lin, D. b. Lin, and C. j. Chang, "Design and implement of transceiver system for hybrid rocket with coplanar wraparound antenna at 433MHz", *Antennas Propagation IET Microwaves*, vol. 8, no. 13, pp. 1080–1090, Oct. 2014, ISSN: 1751-8725. DOI: 10.1049/iet-map.2013.0522.

[6] M. E. Eckart, J. S. Adams, S. R. Bandler, S. E. Busch, J. A. Chervenak, A. J. Ewin, F. M. Finkbeiner, R. L. Kelley, C. A. Kilbourne, F. S. Porter, J. P. Porst, J. E. Sadleir, S. J. Smith, E. J. Wassell, and E. Figueroa-Feliciano, "Development of tes microcalorimeter arrays for the micro-x sounding rocket experiment", *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, p. 2 101 705, Jun. 2013, ISSN: 1051-8223. DOI: 10.1109/TASC.2013.2244631.

[7] K. Chiang, M. Psiaki, S. Powell, R. Miceli, and B. O'Hanlon, "GPS-based attitude determination for a spinning rocket", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 4, pp. 2654–2663, Oct. 2014, ISSN: 0018-9251. DOI: 10.1109/TAES.2014.120822.

[8] R. Varaprasad, R. Gnanasambanthan, J. V. N. Ravikumar, C. J. R. Pamireddy, C. Ramsenthil, K. Haribabu, and V. S. Rao, "Ground-based real-time auto commanding system for isro advanced technology vehicles", *IEEE Aerospace and Electronic Systems Magazine*, vol. 29, no. 5, pp. 18–24, May 2014, ISSN: 0885-8985. DOI: 10.1109/MAES.2014.120207.

[9] S. Patra and E. A. Spencer, "Plasma impedance probe: Simulations and comparison to sounding rocket mission data", *IEEE Transactions on Plasma Science*, vol. 41, no. 1, pp. 220–231, Jan. 2013, ISSN: 0093-3813. DOI: 10.1109/TPS.2012.2225112.

[10] European Space Agency. (Oct. 17, 2016). Cansat, ESA, [Online]. Available: http://www.esa.int/Education/CanSat.

[11] B. Garbee, K. Packard, B. Finch, and A. Towns, *The altus metrum system*, Altus, 2016.

[12] B. Benchoff. (Apr. 27, 2016). Apollo, the everything board, HackaDay, [Online]. Available: http://hackaday.com/2014/09/24/apollo-the-everything-board/.

[13] F. Rockets! (Oct. 14, 2016). Basic components of a sport rocket, [Online]. Available: http://www.flyrockets.com/work.asp.

[14] USDOD, Ed., *Design of aerodynamically stabilized free rockets*, MIL-HDBK-762(MI), 1990.

[15] Centuri, *Model rocket design manual*, Centuri, 1971.

[16] T. Benson, Ed. (Oct. 14, 2016). Welcome to the beginner's guide to rockets, National Aeronautics and Space Administration, [Online]. Available: https://spaceflightsystems.grc.nasa.gov/education/rocket/bgmr.html.

[17] F. M. White, *Fluid Mechanics*, 7th ed. New York, NY, USA: McGraw-Hill, 2011.

[18] ESTES, *Catalog*. ESTES, 2016. [Online]. Available: http://www.estesrockets.com/media//catalogs/Estes_2016_Catalog.pdf (visited on 10/13/2016).

[19] Iomesus. (Oct. 14, 2016). Model rocket engine, Wikimedia, [Online]. Available: https://commons.wikimedia.org/wiki/File:Model-rocket-engine.svg.

[20] A.Pool and D.Bosman, Eds., *AGARD-AG-160*, *Basic principles of flight test intrumentation engeneering*, 2nd ed., Advisory group for aerospace research and development, North Atlantic Treaty Organization, 1994.

[21] C. S. eXploration Team. (Oct. 14, 2016). Official altitude press release, Civilian Space eXploration Team, [Online]. Available: http://www.ddeville.com/derek/CSXT.htm.

[22] P. H. Dana and B. M. Penrod, "The role of GPS in precise time and frequency dissemination", *GPS World*, 1990.

[23] NAVSTAR GPS, *Global positioning system standard positioning service performance standard*, US Department Of Defence, 2008.

[24] Tripoli Rocketry Association, *Tripoli Rocketry Association Safe Launch Practices*. Tripoli Rocketry Association, 2007.

[25] "The international civil aviation organization standard atmosphere", *Nature*, vol. 179, no. 4554, pp. 299–300, Feb. 1957. DOI: 10.1038/179299a0. [Online]. Available: http://dx.doi.org/10.1038/179299a0.

[26] ublox, *U-blox 8 / u-blox m8 receiver description*, ublox, Feb. 26, 2016.

[27] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm", in *2011 IEEE International Conference on Rehabilitation Robotics*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2011. DOI: 10.1109/icorr.2011.5975346. [Online]. Available: http://dx.doi.org/10.1109/ICORR.2011.5975346.

[28] Arduino/Genuino. (Apr. 26, 2016). Arduino fio, Arduino Genuino, [Online]. Available: http://www.arduino.cc/en/Main/ArduinoBoardFio.

[29] Atmel Corporation, *Atmega48A/pa/88A/pa/168A/pa/328/p*, Datasheet, Atmel Corporation, 2014.

[30] Digi International, *Xbee/xbee-pro zb RF modules*, Digi International, 2012.

[31] R. Faludi, *Building Wireless Sensor Networks*, 1st ed, B. Jepson, Ed. Sebastopol, CA, USA: O'Reilly, 2010.

[32] L. E. Frenzel, "Serial peripheral interface (SPI)", in *Handbook of Serial Communications Interfaces*, Elsevier BV, 2016, pp. 143–145. DOI: 10.1016/b978-0-12-800629-0.00035-8. [Online]. Available: http://dx.doi.org/10.1016/b978-0-12-800629-0.00035-8.

[33] Adafruit. (Oct. 14, 2016). Microsd card breakout board+, [Online]. Available: https://www.adafruit.com/product/254.

[34] Arduino Forum. (May 2, 2016). Which microsd cards to use for low current and low power consumption?, [Online]. Available: http://forum.arduino.cc/index.php?topic=261021.0.

[35] greiman. (May 2, 2016). Sdfat, [Online]. Available: https://github.com/greiman/SdFat.

[36] Pololu. (Apr. 26, 2016). Altimu-10 v4 Gyro, accelerometer, compass, and altimeter (l3gd20H, LSm303d, and lps25H carrier), [Online]. Available: https://www.pololu.com/product/2470.

[37] STMicroelectronics, *L3gd20H, MEMS motion sensor: Three-axis digital output gyroscope datasheet*, STMicroelectronics, 2013.

[38] STMicroelectronics., *Lps25H, MEMS pressure sensor: 260-1260 hPa absolutedigital output barometer datasheet*, STMicroelectronics, 2013.

[39] STMicroelectronics, *LSm303d ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module datasheet*, STMicroelectronics, 2013.

[40] NXP, *Um10204, i2C-bus specification and user manual*, NXP, 2007.

[41] Bureau International des Poids et Mesures, *The International System of Units (SI)*, 8th ed. Bureau International des Poids et Mesures, 2006.

[42] W. Premerlani and P. Bizard, "Direction cosine matrix IMU: Theory", 2009.

[43] J. S. Hu and K. C. Sun, "A robust orientation estimation algorithm using marg sensors", *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 3, pp. 815–822, Mar. 2015, ISSN: 0018-9456. DOI: 10.1109/TIM.2014.2359815.

[44] Y. Tian, H. Wei, and J. Tan, "An adaptive-gain complementary filter for real-time human motion tracking with marg sensors in free-living environments", *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 21, no. 2, pp. 254–264, Mar. 2013, ISSN: 1534-4320. DOI: 10.1109/TNSRE.2012.2205706.

[45] Uputronics. (Oct. 14, 2016). U-blox max-m8q breakout with maruwa sl1252r antenna with SAW/LNA, Uputronics, [Online]. Available: https://store.uputronics.com/?route=product/product%5C&product_id=83 (visited on 10/13/2016).

[46] Delano, *Fgpmmopa6H GPS standalone module datasheet*, GlobalTop Technology, 2011.

[47] Adafruit. (Apr. 26, 2016). Adafruit ultimate gps breakout - 66 channel w/10 hz updates - version 3. Accesed 12 October 2015, Adafruit, [Online]. Available: http://www.adafruit.com/products/746.

[48] NMEA. (Apr. 26, 2016). Nmea 0183 protocol, National Marine Electronics Association, [Online]. Available: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp.

[49] PTROBOTICS. (Oct. 14, 2016). Polymer lithium ion battery - 3.7v 400mah, [Online]. Available: http://www.ptrobotics.com/lipo/3233-polymer-lithium-ion-battery-3-7v-400mah.html.

[50] W. A. van Schalkwijk and B. Scrosati, Eds., *Advances in Lithium-Ion Batteries*. Springer Science + Business Media, 2002. DOI: 10.1007/b113788. [Online]. Available: http://dx.doi.org/10.1007/b113788.

[51] Apogee. (May 1, 2016). Rocksim, [Online]. Available: https://www.apogeerockets.com/Rocket_Software.

[52] S. Niskanen, *Openrocket technical documentation*, May 10, 2013.

[53] Arduino/ Genuino. (May 2, 2016). Spi library, [Online]. Available: https://www.arduino.cc/en/reference/SPI.

[54] Arduino/Genuino. (May 2, 2016). Wire library, [Online]. Available: http://www.arduino.cc/en/reference/wire.

[55] SlashDevin. (May 2, 2016). Neohwserial, [Online]. Available: https://github.com/SlashDevin/NeoHWSerial.

[56] Pololu. (May 2, 2016). Arduino library for pololu l3g boards, [Online]. Available: https://github.com/pololu/l3g-arduino.

[57] Pololu. (May 2, 2016). Arduino library for pololu LSm303 boards, [Online]. Available: https://github.com/pololu/lsm303-arduino.

[58] Pololu. (May 2, 2016). Arduino library for pololu lps pressure sensor boards, [Online]. Available: https://github.com/pololu/lps-arduino.

[59] kevin-pololu. (Oct. 14, 2016). Mimu-9-ahrs-arduino, Pololu, [Online]. Available: https://github.com/pololu/minimu-9-ahrs-arduino.

[60] Q. Company. (Oct. 14, 2016). Qt, [Online]. Available: https://www.qt.io/.

[61] R. M. Howard, "Dynamics of flight: Stability and control, 3rd ed", *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 4, pp. 839–840, Jul. 1997. DOI: 10.2514/2.4126. [Online]. Available: http://dx.doi.org/10.2514/2.4126.

[62] G. Y. Chin., "Two-dimensional measures of accuracy in navigational systems", US Departmant of Transportation, Final Report, 1987.

[63] J. P. Snyder, *Map projections–A working manual*. US Government Printing Office, 1987, vol. 1395.

[64] UN Economic Comission for Europe, Ed., *UN Vehicle Regulations - 1958 Agreement*, UN Economic Comission for Europe, 1958.

[65] ESTES, *Model rocketry technical manual*, ESTES, 1993.

[66] MathWorks. (Oct. 14, 2016). Smooth, MathWorks, [Online]. Available: https://www.mathworks.com/help/curvefit/smooth.html.

Appendix A

Arduino code, Rocket Segment

A.1 Arduino

```
// 16 g ACC!!!!!!!!!!!!!!!
   // X axis pointing forward
   // Y axis pointing to the left
   // and Z axis pointing up.
// Positive pitch : nose down
// Positive roll : right wing down
// Positive yaw : counterclockwise
int SENSOR_SIGN[9] = {1,-1,-1,-1,1,1,1,-1,-1}; //Correct directions x,y,z - gyro,
    accelerometer, magnetometer

// tested with Arduino Uno with ATmega328 and Arduino Duemilanove with ATMega168

#include <NeoHWSerial.h>
#include <SPI.h>
#include <SdFat.h>
#include <Wire.h>
#include <LPS.h>

// LSM303 accelerometer: 16 g sensitivity !!!!!!!!!!!!!!!!!
// 11.712  mg/digit; 1 g = 85 !!!!!!!!!!!!!!!!!!!!!!!!!!!!
#define GRAVITY 85  //this equivalent to 1G in the raw data coming from the accelerometer
    !!!!!!!!!!!!!!!!!

#define ToRad(x) ((x)*0.01745329252)  // *pi/180
#define ToDeg(x) ((x)*57.2957795131)  // *180/pi

// L3G4200D gyro: 2000 dps full scale
// 70 mdps/digit; 1 dps = 0.07
#define Gyro_Gain_X 0.07 //X axis Gyro gain
#define Gyro_Gain_Y 0.07 //Y axis Gyro gain
#define Gyro_Gain_Z 0.07 //Z axis Gyro gain
#define Gyro_Scaled_X(x) ((x)*ToRad(Gyro_Gain_X)) //Return the scaled ADC raw data of the
    gyro in radians for second
#define Gyro_Scaled_Y(x) ((x)*ToRad(Gyro_Gain_Y)) //Return the scaled ADC raw data of the
    gyro in radians for second
#define Gyro_Scaled_Z(x) ((x)*ToRad(Gyro_Gain_Z)) //Return the scaled ADC raw data of the
    gyro in radians for second

// LSM303 magnetometer calibration constants; use the Calibrate example from
```

```
// the Pololu LSM303 library to find the right values for your board
#define M_X_MIN -421
#define M_Y_MIN -639
#define M_Z_MIN -238
#define M_X_MAX 424
#define M_Y_MAX 295
#define M_Z_MAX 472

#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

//For debugging purposes
//OUTPUTMODE=1 will print the corrected data,
//OUTPUTMODE=0 will print uncorrected data of the gyros (with drift)
#define OUTPUTMODE 1

//#define PRINT_DCM 0     //Will print the whole direction cosine matrix
#define PRINT_ANALOGS 0 //Will print the analog raw data
#define PRINT_EULER 1   //Will print the Euler angles Roll, Pitch and Yaw

#define STATUS_LED 13

float G_Dt=0.02;    // Integration time (DCM algorithm)  We will run the integration loop at
    50Hz if possible

long timer=0;   //general purpuse timer
long timer_old;
long timer24=0; //Second timer used to print values
int AN[6]; //array that stores the gyro and accelerometer data
int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of the sensors

int gyro_x;
int gyro_y;
int gyro_z;
int accel_x;
int accel_y;
int accel_z;
int magnetom_x;
int magnetom_y;
int magnetom_z;
float c_magnetom_x;
float c_magnetom_y;
float c_magnetom_z;
float MAG_Heading;

float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a vector
float Gyro_Vector[3]= {0,0,0};//Store the gyros turn rate in a vector
float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
float Omega_P[3]= {0,0,0};//Omega Proportional correction
float Omega_I[3]= {0,0,0};//Omega Integrator
float Omega[3]= {0,0,0};

// Euler angles
float roll;
float pitch;
float yaw;
```

```
93 float errorRollPitch[3]= {0,0,0};
   float errorYaw[3]= {0,0,0};
95
   unsigned int counter=0;
97 byte gyro_sat=0;

99 float DCM_Matrix[3][3]= {
     {  1,0,0  }
101  ,{  0,1,0  }
     ,{  0,0,1  }
103 };
   float Update_Matrix[3][3]={{0,1,2},{3,4,5},{6,7,8}}; //Gyros here
105


107 float Temporary_Matrix[3][3]={
     {  0,0,0  }
109  ,{  0,0,0  }
     ,{  0,0,0  }
111 };

113 LPS ps;

115 ////////////SD VARS
    const uint8_t chipSelect = 10;
117
    // Log file base name.  Must be six characters or less.
119 #define FILE_BASE_NAME "Data"
    //------------------------------------------------------------------------------
121 // File system object.
    SdFat sd;
123
    // Log file.
125 SdFile file;

127 // Time in micros for next data record.
    uint32_t logTime;
129
    // Error messages stored in flash.
131 #define error(msg) sd.errorHalt(F(msg))


133
    float P[2];
135
    ////////////GPS WORKING VARS
137 volatile boolean prt;

139 //uint8_t b; not needed, as it is passed in as an argument to GPS_Read
    volatile uint8_t  i; // size reduced because packet size is < 256.
141 volatile uint16_t drp = 0; // ok, dropped count *could* be > 255.


143
    uint8_t cka;
145 uint8_t ckb;

147 ////////////GPS OUTPUT
    long GDT[8];
149 uint32_t AC[8];

151 struct gps_t {
```

```
153   uint8_t H1;
      uint8_t H2;
155   uint8_t Class;
      uint8_t ID;
157   uint16_t Length;
      uint32_t iTow;
159   uint16_t Year;
      uint8_t Month;
161   uint8_t Day;
      uint8_t Hour;
163   uint8_t Min;
      uint8_t Sec;
165   uint8_t valid;
      uint32_t tAcc;
167   long nano;
      uint8_t fixType;
169   uint8_t flags;
      uint8_t reserved1;
171   uint8_t numSV;
      long lon;
173   long lat;
      long height;
175   long hMSL;
      uint32_t hAcc;
177   uint32_t vAcc;
      long velN;
179   long velE;
      long velD;
181   long gSpeed;
      long headMot;
183   uint32_t sAcc;
      uint32_t headAcc;
185   uint16_t pDOP;
      uint8_t reserved2;
187   uint8_t reserved2a;
      uint8_t reserved2b;
189   uint8_t reserved2c;
      uint8_t reserved2d;
191   uint8_t reserved2e;
      long headVeh;
193   uint8_t reserved3;
      uint8_t reserved3a;
195   uint8_t reserved3b;
      uint8_t reserved3c;
197   uint8_t ck_a;
      uint8_t ck_b;
199 };

201 union gps_u {
      uint8_t PVT[100];
203   struct gps_t gps;
    };
205

    union gps_u GPS;
207


209 //////TEST
```

```
struct msg_t {
  uint8_t H1;
  uint8_t H2;
  int AN[9];
  float roll;
  float pitch;
  float yaw;
  long t;
  float dt;
  float P[2];
  uint8_t Hour;
  uint8_t Min;
  uint8_t Sec;
  uint8_t numSV;
  long nano;
  long lon;
  long lat;
  long hMSL;
  long velN;
  long velE;
  long velD;
};

union msg_u {
  uint8_t strm[80];
  struct msg_t msg;
};

union msg_u MSG;


void setup()
{
  NeoSerial.begin(57600);
  NeoSerial.println("MIMUTEST 16");
  pinMode (STATUS_LED,OUTPUT);  // Status LED

  I2C_Init();

  NeoSerial.println("Pololu MinIMU-9 + Arduino AHRS    16 SYNC");

  digitalWrite(STATUS_LED,LOW);
  delay(1500);

  Accel_Init();
  Compass_Init();
  Gyro_Init();
   ps.init();
  ps.enableDefault();
  delay(100);
  delay(20);

  //Must be calibrated for each module.
  AN_OFFSET[0]=-4;
  AN_OFFSET[1]=-145;
  AN_OFFSET[2]=-24;
  AN_OFFSET[3]=0;
  AN_OFFSET[4]=-1;
  AN_OFFSET[5]=1;
```

```
  //NeoSerial.println("Offset:");
  for(int y=0; y<6; y++)
    NeoSerial.println(AN_OFFSET[y]);

  delay(2000);
  digitalWrite(STATUS_LED,HIGH);

  const uint8_t BASE_NAME_SIZE = sizeof(FILE_BASE_NAME) - 1;
  char fileName[13] = FILE_BASE_NAME "00.bin";
  delay(1000);///ALT

  // Initialize the SD card at SPI_FULL_SPEED for better performance.
  if (!sd.begin(chipSelect, SPI_FULL_SPEED)) {
    sd.initErrorHalt();
  }

  // Find an unused file name.
  if (BASE_NAME_SIZE > 6) {
    error("FILE_BASE_NAME too long");
  }
  while (sd.exists(fileName)) {
    if (fileName[BASE_NAME_SIZE + 1] != '9') {
      fileName[BASE_NAME_SIZE + 1]++;
    } else if (fileName[BASE_NAME_SIZE] != '9') {
      fileName[BASE_NAME_SIZE + 1] = '0';
      fileName[BASE_NAME_SIZE]++;
    } else {
      error("Can't create file name");
    }
  }
  if (!file.open(fileName, O_CREAT | O_WRITE | O_EXCL)) {
    error("file.open");
  }

  NeoSerial.attachInterrupt( GPS_Read );

  NeoSerial.print(F("Logging to: "));
  NeoSerial.println(fileName);


  timer=millis();
  delay(20);
  counter=0;
}

void GPS_Read( uint8_t b ) {
  //  while (NeoSerial.available() > 0 && rec == false) {
  //    b = NeoSerial.read();     //READ BYTE

  if (i > 1) {           //SEE IF WE ARE RECIEVING
    if (i == 99) {       //SEE IF WE ARE ENDING
      GPS.PVT[i] = b;
      i = 0;             //RESET BYTE COUNTER
      if (GPS.PVT[98] == cka && GPS.PVT[99] == ckb) {
        //          rec = true;       //SET RECIEVED FLAG IF CHECKSUM IS OK
        cka = 0;
        ckb = 0;
        prt = true;
```

```
      } else {
        cka = 0;
        ckb = 0;
      }
    } else {
      GPS.PVT[i] = b;         //ADD BYTE
      if (i != 98) {     //CALCULATE CHECKSUM
        cka = cka + b;
        ckb = ckb + cka;
      }
      i++;                //PROGRESS COUNTER
    }

  } else if ((i == 0) && (b == 0xB5)) { //CHECK START BYTE
    GPS.PVT[i] = b;           //ADD BYTE
    i++;                  //PROGRESS COUNTER

  } else if ((i == 1 && b == 0x62)) {     //CHECK SECOND BYTE
    GPS.PVT[i] = b;         //ADD BYTE
    i++;                //PROGRESS COUNTER
  } else {
    i = 0;              //RESET BYTE COUNTER
    drp++;
  }
  //  }
}

void loop() //Main Loop
{
  if((millis()-timer)>=20)  // Main loop runs at 50Hz
  {
    //NeoSerial.println(millis()-timer);
    counter++;
    timer_old = timer;
    timer=millis();
    if (timer>timer_old)
      G_Dt = (timer-timer_old)/1000.0;     // Real time of loop run. We use this on the DCM
    algorithm (gyro integration time)
    else
      G_Dt = 0;

    // *** DCM algorithm
    // Data adquisition
    Read_Gyro();   // This read gyro data
    Read_Accel();      // Read I2C accelerometer
    Read_Compass();     // Read I2C magnetometer
    MSG.msg.P[0] = ps.readPressureMillibars();
    MSG.msg.P[1] = ps.readTemperatureC();


    if (counter > 5)  // Read compass data at 10Hz... (5 loop runs)
      {
      counter=0;
      Compass_Heading(); // Calculate magnetic heading

      file.sync();


      }
```

```
387
      // Calculations...
389   Matrix_update();
      Normalize();
391   Drift_correction();
      Euler_angles();
393   // ***
       MSG.msg.H1='G';
395  MSG.msg.H2='C';
     MSG.msg.t=timer;
397  MSG.msg.dt=G_Dt;
     MSG.msg.roll=roll;
399  MSG.msg.pitch=pitch;
     MSG.msg.yaw=yaw;
401   if (prt) {
        MSG.msg.Hour=GPS.gps.Hour;
403     MSG.msg.Min=GPS.gps.Min;
        MSG.msg.Sec=GPS.gps.Sec;
405     MSG.msg.nano=GPS.gps.nano;
        MSG.msg.numSV=GPS.gps.numSV;
407     MSG.msg.lat=GPS.gps.lat;
        MSG.msg.lon=GPS.gps.lon;
409     MSG.msg.hMSL=GPS.gps.height;
        MSG.msg.velN=GPS.gps.velN;
411     MSG.msg.velE=GPS.gps.velE;
        MSG.msg.velD=GPS.gps.velD;
413     file.write(MSG.strm,sizeof(MSG.strm));
        NeoSerial.write(MSG.strm,80);
415     prt=false;
      }else{
417     file.write(MSG.strm,sizeof(MSG.strm));
        NeoSerial.write(MSG.strm,80);
419     }
  }
421 if (millis() > 1800000) {
    File_Close();
423   while (1) {}
  }
425}

427void File_Close() {
  file.close();
429 NeoSerial.println(F("Done"));
}
```

Code/Ard/MIMUTEST_16G.ino

Appendix B

Matlab code

B.1 Rocket Altitude Simulation

```
close all;
clear all;
clc;

%Constantes
g=9.81;

mr=0.252;
l=0.845;
d=0.066;
a=0.033*0.033*3.14;
bt=1.6;
det=3;

mm=0.042;

mp=0.006;

m=mr+mm+mp;

%m=0.344;

Cd=0.4;
rho=1.225;

k=0.5*a*rho*Cd;


%Condicoes iniciais
x(1)=0;
dx(1)=0;



%Step de Integracao
dt=0.01;

%Thrust
T(1)=30;
```

```
n=round(bt/dt);

for j=2:n
    T(j)=10;
end

for kk=n+1:(det+bt)/dt+1

            T(kk)=0;
end


% It=dt*32+dt*(n-1)*12;




for i=1:(det+bt)/dt+1

    ddx(i)=T(i)/m-g-k*dx(i)/m;

    dx(i+1)=dx(i)+ddx(i)*dt;

    x(i+1)=x(i)+dt*dx(i);

    if i>=n

    end

    m=m-0.021*T(i)/sum(T);
%      I=T(i)*dt;

%      m(i+1)=m(i)-I/It*;


end

ddx(i+1)=0/m-g-k*dx(i+1)/m;

fprintf('Massa: %.2f \n',m)

fprintf('Altitude maxima: %.2f \n',x(i))

fprintf('Tempo de voo: %.2f \n',(i)*dt)

fprintf('Velocidade maxima: %.2f \n',max(dx))

fprintf('Accelarcao maxima: %.2f \n',max(ddx))
% plot(0:dt:(i)*dt,x)
% xlabel('(s)')
% ylabel('m')
% hold on
% plot(dx)
% hold on
% plot(ddx)
% hold on


ylabels{1}='(m)';
```

```
 ylabels{2}='(m/s)';
 ylabels{3}='(m/s^2)';
  [ax,hlines] = plotyyy(0:dt:(i)*dt,x,0:dt:(i)*dt,dx,0:dt:(i)*dt,ddx,ylabels);
 xlabel(ax(1),'(s)');
 legend(hlines, 'Altitude','Velocity','Acceleration')
 print('Matlab_sim1','-depsc')

%plot(0:dt:(i-1)*dt,T)
```

Code/Rocket.m

B.2 CSV File Importer, Ground Segment

```
function
    [GyFin,AcFin,AgFin,MgFin,rollFin,pitchFin,yawFin,TFin,dtFin,PFin,TmpFin,TG0Fin,TGFin,LaFin,LoFin,VNFin,VE
    = READCSV(g16,FILENAME)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


%gLX=9.80131; %em Lisboa
g0=9.80665; %standard

    if g16
        FAcc=16*.244*10^-3; %factor do accelerometro 16 devido ao shift e .244 da doc para 8g
    else
        FAcc=16*.732*10^-3; %factor do accelerometro 16 devido ao shift e .732 da doc para 16g
    end

FGyr=70*10^-3; %70 devido a 2000dps
FMag=0.080*10^-3; %0.080 devido a 2gauss

deg=180/pi;

%AN_OFFSET=csvread(File,0,0,[0 0 5 0]).';

M=csvread( FILENAME,0,2);

Gyr=M(:,1:3);
GyFin=FGyr*Gyr;%deg/s

Acc=M(:,4:6);
AcFin=FAcc*Acc;%g
AgFin=(FAcc*g0)*Acc;%m/s2

Mag=M(:,7:9);
MgFin=FMag*Mag;%G

roll=M(:,10);
rollFin=roll*deg;

pitch=M(:,11);
pitchFin=pitch*deg;

yaw=M(:,12);
yawFin=yaw*deg;

```

```
T=M(:,13);
TFin=T*0.001; %s

dt=M(:,14);
dtFin=dt;


PT=M(:,15:16);
PFin=PT(:,1);%mbar
TmpFin=PT(:,2);%Celsius

%nd=1:length(M(:,1));

n=0;

GPS(length(M(:,1)),1)=0;

for i=1:length(M(:,1))

   % if M(i,20)>9
        n=n+1;
        GPS(n,1:11)=M(i,17:27);
        GPS(n,12)=i;
        GPS(n,13)=GPS(n,1)*3600+GPS(n,2)*60+GPS(n,3)+GPS(n,5)*10^-9;
        if n>1
            GPS(n,14)=M(i,13)-M(GPS(n-1,12),13);
            GPS(n,15)=GPS(n,14)-GPS(n-1,14);
        end
    %end
end

TG0Fin=GPS(:,13);
TGFin=GPS(:,14);%s
LaFin=GPS(:,7)*10^-7;%deg
LoFin=GPS(:,6)*10^-7;%deg
VNFin=GPS(:,9)*10^-3;%m/s
VEFin=GPS(:,10)*10^-3;%m/s
HFin=GPS(:,8)*10^-3;%m
VDFin=GPS(:,11)*10^-3;%m/s
SVFin=GPS(:,4);

end
```

Code/READCSV.m

Appendix C

Python code, Ground Segment

C.1 .bin to .csv

```
##from visual import *
##from visual.graph import *
##import serial
##import string
import math
import numpy as np
from struct import *
from collections import namedtuple
import csv

from numpy import linalg as LA

from time import time

DATA=namedtuple('DATA','H1 H2 gx gy gz ax ay az mx my mz pitch roll yaw t dt P T h m s SV n
    lon lat alt vN vE vD')
#dat=DATA(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
ff = open("Log_File.bin", "rb")
f = open("File_TEST"+str(time())+".csv", 'w')
file_writer = csv.writer(f)
e=0
while(e==0):
    try:
        data=ff.read(80)
##        print calcsize('<bbhhhhhhhhhhffffffffbbbblllllll')
        LINE=(unpack_from('bbhhhhhhhhhhffffffffbbbblllllll',data))
        dat=DATA._make(LINE)
        file_writer.writerow(LINE)
        print dat.t
    except:
        print 'error'
        e=1
f.close
```

Code/FILEREAD.py

Appendix D

C++ code, Ground Segment

```
#ifndef MAKEVECTORS_H
#define MAKEVECTORS_H

#include <QObject>
#include <QVector>
#include "obser.h"

class MakeVectors : public QObject
{
    Q_OBJECT
public:
    explicit MakeVectors(QObject *parent = 0);


signals:
    sGyr(QVector<double>*,QVector<double>*,QVector<double>*,QVector<double>*);
    sAcc(QVector<double>*,QVector<double>*,QVector<double>*,QVector<double>*);
    sMag(QVector<double>*,QVector<double>*,QVector<double>*,QVector<double>*);
    sVel(QVector<double>*,QVector<double>*,QVector<double>*,QVector<double>*);
    sPrs(QVector<double>*,QVector<double>*);
    sTmp(QVector<double>*,QVector<double>*);
    sdt(QVector<double>*,QVector<double>*);
    slat(QVector<double>*,QVector<double>*);
    slon(QVector<double>*,QVector<double>*);
    saltg(QVector<double>*,QVector<double>*);
    saltB(QVector<double>*,QVector<double>*);
    salt(QVector<double>*,QVector<double>*,QVector<double>*,QVector<double>*);
    sTim(QVector<double>*,QVector<double>*);
    sPos(QVector<double>*,QVector<double>*);


public slots:
    void RecMsg(msg_t msg);
    void Start();
    void clear();
    void set_off(double altb,double altg){altb_off=altb;altg_off=altg;}
    void setAcc16(bool acc);

private:
    double altb_off=0;
    double altg_off=0;
    double CAcc=256;
```

```
43
    void make(double t, QVector<double> *t_b);
45  void timealign(QVector<double> *t_o, QVector<double> *t_b);
    QVector<double> Tim;
47  QVector<double> Tie;
    QVector<double> Acc;
49  QVector<double> Acc1;
    QVector<double> Acc2;
51  QVector<double> Gyr;
    QVector<double> Gyr1;
53  QVector<double> Gyr2;
    QVector<double> Mag;
55  QVector<double> Mag1;
    QVector<double> Mag2;
57  QVector<double> Vel;
    QVector<double> Vel1;
59  QVector<double> Vel2;
    QVector<double> Prs;
61
    QVector<double> altB;
63
    QVector<double> Tmp;
65  QVector<double> dt;
    QVector<double> lat;
67  QVector<double> lon;
    QVector<double> alt;
69 };

71 #endif // MAKEVECTORS_H
```

Code/UniTest/makevectors.h

```
1  #include "makevectors.h"
   #include <math.h>
3
   MakeVectors::MakeVectors(QObject *parent) : QObject(parent)
5  {

7  }

9  void MakeVectors::Start(){
       Tim.reserve(1520);
11     Tie.reserve(1520);
       Acc.reserve(1520);
13     Acc1.reserve(1520);
       Acc2.reserve(1520);
15     Gyr.reserve(1520);
       Gyr1.reserve(1520);
17     Gyr2.reserve(1520);
       Prs.reserve(1520);
19
       altB.reserve(1520);
21
       Mag.reserve(1520);
23     Mag1.reserve(1520);
       Mag2.reserve(1520);
25     Vel.reserve(1520);
       Vel1.reserve(1520);
27     Vel2.reserve(1520);
```

```
    Tmp.reserve(1520);
    dt.reserve(1520);
    lat.reserve(1520);
    lon.reserve(1520);
    alt.reserve(1520);
    Tie.append(0);
}

void MakeVectors::setAcc16(bool acc)
{
    if(acc)
    {
        CAcc=85;

    }else {
        CAcc=256;

    }
    qDebug() << QString::number(CAcc,'g',20);
}

void MakeVectors::RecMsg(msg_t msg)
{
    make(msg.t/(double)1000,&Tie);
    timealign(&Tim,&Tie);

    emit sTim(&Tim,&Tie);

    make((double)msg.AN[0]*0.07,&Gyr);
    make((double)msg.AN[1]*0.07,&Gyr1);
    make((double)msg.AN[2]*0.07,&Gyr2);


    emit sGyr(&Tim,&Gyr,&Gyr1,&Gyr2);

    make(9.81*msg.AN[3]/CAcc,&Acc);
    make(9.81*msg.AN[4]/CAcc,&Acc1);
    make(9.81*msg.AN[5]/CAcc,&Acc2);
    emit sAcc(&Tim,&Acc,&Acc1,&Acc2);



    make((double)msg.P[0],&Prs);
    emit sPrs(&Tim,&Prs);

    double altb_p=(1-qPow((double)msg.P[0]/1013.25,0.190284))*145366.45;
    double altb_f=altb_p-altb_off;
    make(altb_f,&altB);
    emit saltB(&Tim,&altB);


    make((double)msg.AN[6]*0.016,&Mag);
    make((double)msg.AN[7]*0.016,&Mag1);
    make((double)msg.AN[8]*0.016,&Mag2);
    emit sMag(&Tim,&Mag,&Mag1,&Mag2);

    make((double)msg.velN*0.001,&Vel);
    make((double)msg.velE*0.001,&Vel1);
    make((double)msg.velD*0.001,&Vel2);
```

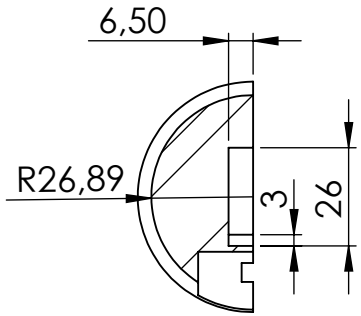
```
87      emit sVel(&Tim,&Vel,&Vel1,&Vel2);

89      make((double)msg.P[1],&Tmp);
        emit sTmp(&Tim,&Tmp);
91
        make((double)msg.dt,&dt);
93      emit sdt(&Tim,&dt);

95      make((double)msg.lat*0.0000001,&lat);
        emit slat(&Tim,&lat);
97
        make((double)msg.lon*0.0000001,&lon);
99      emit slon(&Tim,&lon);


101
//      qDebug()<<QString::number((double)(msg.lon*0.0000001),'g',20);
103
        emit sPos(&lon,&lat);
105
        double altg_p=(double)msg.hMSL*0.001;
107     double altg_f=altg_p-altg_off;
        make(altg_f,&alt);
109     emit saltg(&Tim,&alt);

111     emit salt(&Tim,&altB,&alt,&alt);
}
113
void MakeVectors::make(double t,QVector<double>* t_b)
115 {
        if(t_b->size()>=1500)
117     {
            t_b->removeFirst();
119         t_b->append(t);
        }else{
121         t_b->append(t);
        }
123 }

125 void MakeVectors::timealign(QVector<double> *t_o, QVector<double> *t_b)
{
127
        for(int i=0;i<t_b->size();i++)
129     {
            if(i < t_o->size())
131         {
                t_o->replace(i,t_b->value(i)-t_b->last());
133         }else{
                t_o->append(t_b->value(i)-t_b->last());
135         }

137     }
}
139
void MakeVectors::clear()
141 {
        Tim.resize(0);
143     Tie.resize(0);
        Acc.resize(0);
145     Acc1.resize(0);
```

```
    Acc2.resize(0);
    Gyr.resize(0);
    Gyr1.resize(0);
    Gyr2.resize(0);
    Prs.resize(0);
    altB.resize(0);

    Mag.resize(0);
    Mag1.resize(0);
    Mag2.resize(0);
    Vel.resize(0);
    Vel1.resize(0);
    Vel2.resize(0);
    Tmp.resize(0);
    dt.resize(0);
    lat.resize(0);
    lon.resize(0);
    alt.resize(0);
    Tie.resize(0);

    qDebug()<<"size"<<Tim.size();
}
```

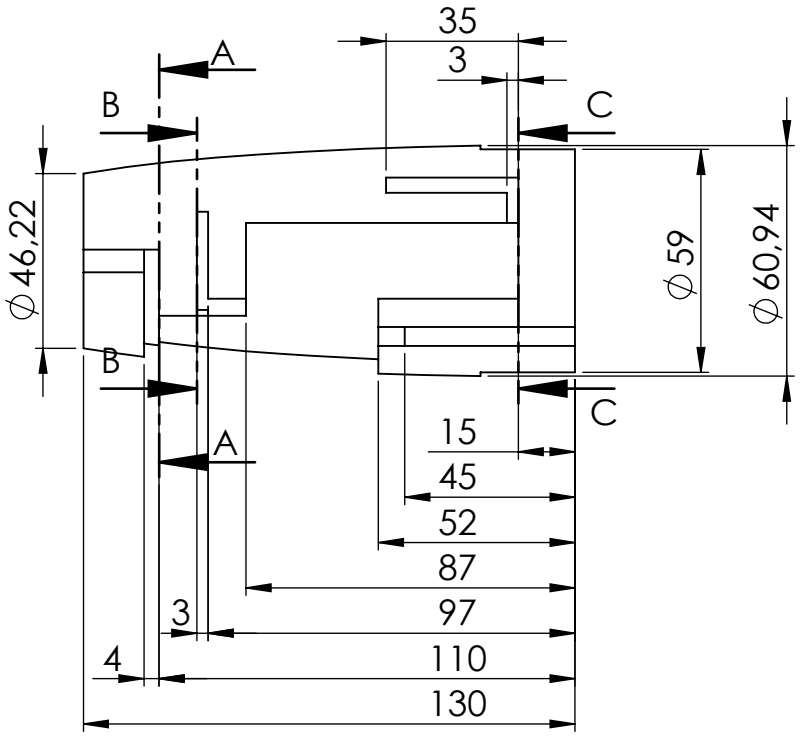
Code/UniTest/makevectors.cpp

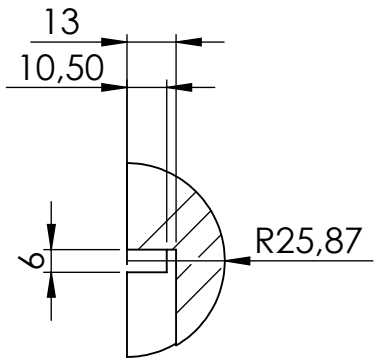
Appendix E

Drawings

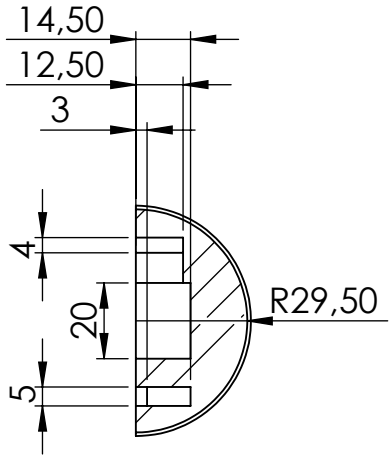


SECTION B-B





SECTION A-A



SECTION C-C

Escala 1:2		Rúbrica	Data
	Desenhou		14/11/2016
	Verificou		14/11/2016
Material: XPS	Projectou		
	Fabricou		
Tolerância Geral:			
Acabamentos:	Notas:		

Small Rocket Instrumentation System

Nome:Guilherme Caiado n.º 70127
Curso: MEAer

Nosecone Housing

Folha	Desenho n.º	Folha	Peso:	Revisto:
A4	M0XL01X-001	1	5.49	