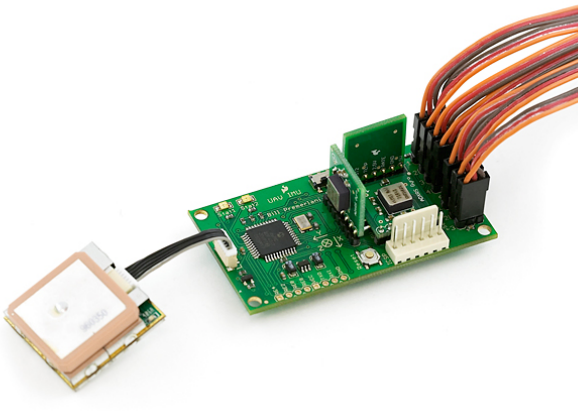
University of Seville
Navigation Aids Systems Course

# GPS and Inertial data acquisition, wrangling and fusion. Brief Study of Airborne Software Systems Certification.



Carlos Caravaca Gallego
Almudena Toronjo Ruiz

March 5, 2021

Escuela Técnica Superior de
INGENIERÍA DE SEVILLA

# Contents

**1 Introduction 2**

**2 Methodology 3**
2.1 GPS Data Acquisition . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2.2 Inertial Data Acquisition . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Data Wrangling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.4 Sensor Data Fusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

**3 Theoretical Analysis and Problem Formulation 14**
3.1 Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
3.2 Ada Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

**4 Data Acquisition and Wrangling 17**

**5 Data Fusion 18**

**6 Conclusion 21**

**7 Futures Lines of Research 22**

**References 23**

# 1 Introduction

The objective of this project is the procurement, handling and integration of both inertial and GPS data from the electronic components provided by the lecturers. Since this topic has already been widely examined by former projects, our focus will be on the study of the considerations in the algorithm certification. The subject of certification is often overlooked within the undergraduate curriculum, with the main emphasis put towards systems design. Nevertheless, not only is systems certification a paramount step of the process of systems design, but we consider it to be the starting point of the development of on board algorithms, given the fact that a design unable to succeed in the applicable tests required for approval is absolutely useless, hence the importance of the research of this matter.

For this purpose, the project has been divided into the presentation of the research methodology applicable to each task, where certification considerations will be mostly displayed, a theoretical analysis of the sensor filtering problem and the Kalman Filter solution [1], and subsequent sections where the actual data acquisition, wrangling and fusion will be performed employing the electronic component that we have been assigned. Finally, conclusions will be drawn from our study and future lines of research will be presented in a synthesized manner.

# 2 Methodology

In this section, the research methodology will be laid out and the technologies chosen for each purpose will be justified. The definition of the methods used to accomplish every task that has been set out constitutes a matter of chief importance in order to rationalize the validity of the results obtained.

## 2.1 GPS Data Acquisition

The acquisition of the GPS data will be carried out by the electronic component that we have been assigned for this project; namely, the UAV v2 Development Platform by SparkFun [2].

### *2.1.1 UAV v2 Development Platform*

The EM406 GPS-UAV Development Platform Version 2.9 is a board that has the accelerometers, the gyros, and the A/D reference voltage all operating ratiometrically and tied to a single regulated 3.3 volt power supply. This completely eliminates variations in the 5 volt supply to the board as a source of accelerometer and gyro drift.

Therefore, delving into key parts of the hardware [3]:

- **EM406 SiRF III GPS receiver**

  The EM406 initializes very quickly. After that, the EM406 reverts to factory settings, that is, 4,800 baudrate and NMEA for the communications protocol.

  Once per second, the GPS reports useful information for navigation including longitude, latitude, altitude, and direction and speed over ground. Vertical speed is available via the binary interface.

  Communications with the GPS can be carried out either through NMEA standard format or a binary format.

- **dsPIC30F4011**

  The CPU is a Microchip dsPIC30F4011. The dsPIC30F4011is ideally suited for use in a three axis GPS-UAV, with plenty of processing power and features, including:

  - 16 bit architecture, with 40 bit accumulators.
  - C compiler available from Microchip.
  - Clock speed up to 120 MHz.
  - Two USARTS. One of them is connected to the GPS, the other can be used for debugging.

  - Vectored, prioritized interrupts simplify interrupt processing. Each interrupt source is matched to a dedicated service routine for just that resource. Each source can be assigned one of seven levels of interrupt priority.

  - Free running PWM generator makes it possible to control 3 servos after initial setup by simply writing to three 16 bit duty cycle registers.

  - Free running buffered A/D converter samples 7 analog input channels and generates an interrupt when a set of samples is ready.

  - 16 bit capture inputs measure the pulse widths of the 4 radio channels and generates a separate interrupt for each channel.

  - Mapping of program space into data space. The combination of the dsPIC architecture and the support provided by the C compiler makes it easy to construct lookup tables.

- **Oscillator**

  The CPU can be run at 8, 16, 32, 64, or 120 MHz.

- **Gyros and 3 axis accelerometer**

  There are three STMicroelectronics LISY300AL gyros, one for each axis, and a three axis MMA7260 accelerometer. Sensitivity of the gyros is approximately 3.3 millivolts per degree per second, with a maximum range of plus or minus 300 degrees/second. Sensitivity of the accelerometers, which are set on the 6 g range, is approximately 200 millivolts per g, with a maximum range of plus or minus 6 g. The board uses a DCM firmware show that is able to achieve wide dynamic range and good accuracy at the same time.

  For each axis (X ,Y and Z), the accelerometer measures acceleration along that axis, and the gyro measures rotation around that axis (the gyros produce positive change in output voltage when they are rotated in the direction indicated on their breakout boards). However, it should be noted that what each accelerometer actually measures is the gravity minus acceleration.

  The outputs of the gyros and accelerometers are analog voltages. Zero rotation and zero acceleration correspond approximately to half voltage. Gyro supply voltages, accelerometer supply voltage, and the analog-to-digital converter reference voltage are all tied to an onboard regulated 3.3 volt supply. The gyros, accelerometers, and A/D are all ratiometric, so effects ofvariations in the 5 volt main supply to the board are totally eliminated.

- **Filtering and sampling rate**

  The accelerometers and gyros have built in analog first order RC low pass filters for noise reduction and anti-aliasing.

- **Radio/servo input/output pulses**

  The interface between the GPS-UAV and the servos and the RC receiver is via 7 PWM I/O connection points on the board. There are 4 PWM inputs from the RC receiver labeled 1, 2, 3, and 4 on the board, collectively labeled “Radio”.

  There are 3 PWM servo outputs labeled 1, 2, and 3 on the board, collectively labeled “Servos”. Signals on these connections are standard RC 5 volt TTL servo pulses.

  PWM inputs are through interrupt-on-change pins. The dsPIC30F4011 makes the interface rather simple. Each input pin can be assigned its own separate interrupt routine, and each pin can use a capture function to measure pulse width. PWM outputs are through dsPIC30F4011 PWM generators. The 16 bit architecture makes it particularly simple to generate the waveforms needed by the servos.

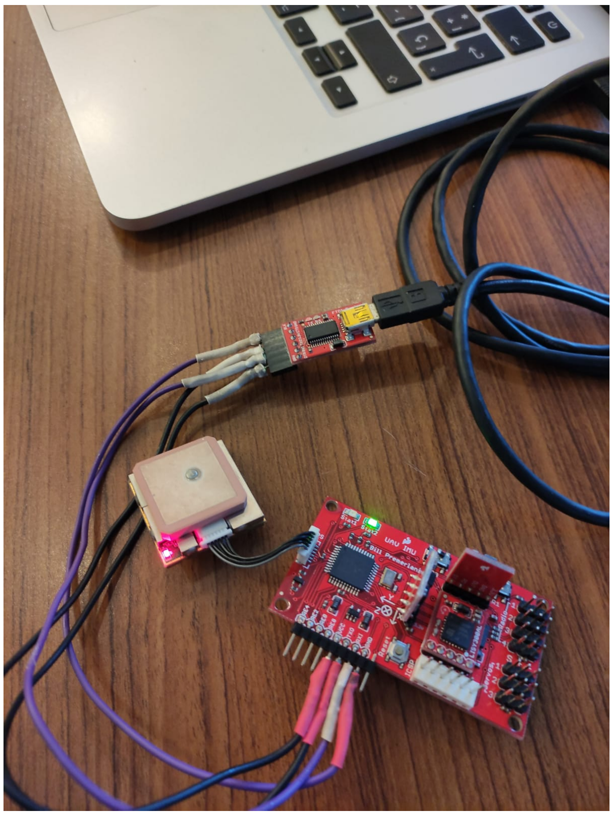


Figure 2.1: Serial Interface Connection

- **ICSP header**

  The GPS-UAV includes an ICSP header which is used for programming and debugging (VPP, VCC, GND, PGD, PGC and N/C)

- **Spare pins**

  The board also provides access to the unused CPU pins: 4 digital I/Os and a spare USART, (USART1).

|  |  |
|---|---|
| GND | ground |
| RXI | U1ARX/RC14 |
| VCC | Regulated 5 V |
| RE8 | RE8 |
| RE0 | RE0 |
| RE2 | RE2 |
| RE4 | RE4 |

- **Accessories**

  A basic breakout board for the USB to serial IC is needed for the interface with the computer. The pinout of this board matches the FTDI cable to work with official Arduino and cloned 3.3V Arduino boards. It can also be used for general serial applications [4]. In order to perform the hookup of the boards, documentation had to be consulted to make sure the appropriate connections were made [5]. This connection is displayed in figure 2.1.

### *2.1.2 NMEA 0183*

| *Message* | **Function** |
|---|---|
| GSV | Number of SVs in view, PRN, elevation, azimuth, and SNR |
| GSA | GPS DOP and active satellites |
| VTG | Actual track made good and speed over ground |
| RMC | Position, Velocity, and Time |
| GNS | GNS Fix data |
| GGA | Time, position, and fix related data |

Table 2.1: NMEA sentences overview.

NMEA 0183 is both an electrical and data specification for communication between marine electronic devices such as gyrocompass, autopilot and GPS receivers amongst many other types of instruments. It has been defined by, and is controlled by, the U.S.-based National Marine Electronics Association [6].

NMEA uses a simple ASCII, serial communications protocol that defines how data is transmitted in a *sentence* from one *talker* to multiple *listeners* at a given time.

Current GPS receiver communication is defined within this specification. The great majority of computer programs that provide real time position information expect data received from satellites to be in NMEA format. This data includes the complete PVT (position, velocity, time) solution computed by the GPS receiver. The objective of NMEA is to send a line of data called a sentence that is totally self contained. There are standard sentences for each device category and there is also the ability to define proprietary sentences for use by the individual company. All of the standard sentences have a two letter prefix that defines the device that uses that sentence type (For gps receivers the prefix is GP, for Galileo the prefix is GA, etc...) followed by a three letter sequence that defines the

| *Field* | **Example Value** |
|---|---|
| Message ID | $GPGGA |
| UTC of position fix | 172814.0 |
| Latitude | 3723.46587704 |
| Direction of Latitude | N |
| Longitude | 12202.26957864 |
| Direction of Longitude | W |
| GPS Quality indicator | 1: GPS fix |
| Number of SVs in use | 0-24 |
| HDOP | 1.2 |
| Orthometric height | 18.9 |
| Unit of measure for orthometric height | M (meters) |
| Geoid separation | -25.6 |
| Unit of measure for geoid separation | M (meters) |
| Checksum | *4F |

Table 2.2: GGA sentence description.

sentence contents [7].

| *Field* | **Example Value** |
|---|---|
| Message ID | $GPRMC |
| UTC of position fix | 172814.0 |
| Status | A (Active) / V (Void) |
| Latitude | 3723.46587704 |
| Direction of Latitude | N |
| Longitude | 12202.26957864 |
| Direction of Longitude | W |
| Speed over ground | 22.4 (m/s) |
| Track angle | 84.0 (degrees) |
| Date | 221297 |
| Magnetic variation | 3.9 (Degrees) |
| Checksum | *6B |

Table 2.3: RMC sentence description.

Table 2.1 summarizes the set of NMEA messages supported by the receiver that will either be useful for our project or it is considered relevant to mention [8].

Additionally, the contents of the sentences *RMC* and *GGA* will be detailed in subsequent tables since they will be the ones used for the project. Most of the information will be retrieved from *GGA* sentence whereas *RMC* sentence will be used to scale the velocity components (see Table 2.2 and 2.3).

### *2.1.3 Python*

The acquisition of NMEA sentences could have been carried out through CuteCom application [9] which includes a graphical interface to facilitate the understanding of data being received. However, python has been chosen to obtain this information in order to test the pySerial library [10] that provides an easy method to establish a serial interface connection. Listing 1 displays the acquisition of this information using a 1 Hz frequency and 4800 as baudrate. The standard output of this program has been piped to a text file to be parsed in the following sections.

```python
import serial
import time

class serialInterface():

  tty = None
  baud = None
  serial = None

  def __init__(self, tty, baud = 4800):
    self.tty = tty
    self.baud = baud
    self.serial = serial.Serial(self.tty, self.baud)

  def read(self):
    in_bytes = self.serial.inWaiting()
    time.sleep(1)
    if in_bytes > 0:
      return self.serial.read(in_bytes)
    else:
      return ""

interface = serialInterface('/dev/ttyUSB1',4800)

while True:
  print(interface.read())
```

Listing 1: gps_acquisition.py

## 2.2 Inertial Data Acquisition

The inertial data will be obtained through coordination with other group of students who are currently researching inertial data manipulation for this year's project. For this purpose, the attainment of data must be performed working side by side and performing the measurements for the same test.

The group of Javier Beltrán and Arturo Lammers is developing a calibration and data acquisition project from an Inertial Measurement Unit, i.e., the 9DOF Razor IMU [11]. This IMU incorporates three sensors (triple-axis gyro, triple-axis accelerometer and triple-axis magnetometer) to give nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328 and output over a serial interface. This enables the 9DOF Razor to be used as a very powerful control mechanism for UAVs, autonomous vehicles and image stabilization systems.

An example of the data provided is displayed in Table 2.4. Additionally, the values of the gyros and the magnetometers are also provided in a similar format.

| Timestamp (ns) | accX (m/s$^2$) | accY (m/s$^2$) | accZ (m/s$^2$) |
|---|---|---|---|
| 6.81E+13 | -0.0104 | 0.1904 | 9.9066 |
| 6.81E+13 | -0.2194 | 0.5006 | 9.8176 |
| 6.81E+13 | 0.0448 | 0.1305 | 9.9859 |

Table 2.4: Example of Inertial Data

## 2.3 Data Wrangling

The data wrangling will be performed through Python. Python is the most widely used programming language for data scientists today [12]. With over 70,000 Python libraries, the possibilities within this language seem endless. Python also allows a programmer to create CSV output to easily read data in a spreadsheet, which is a feature that will be used to parse the information from the GPS.

The inputs are the raw data from the GPS and the desired output is data (position and velocity) in a format that can be interpreted by MATLAB and the Ada programming language to carry out the filtering.

```python
import utm
import csv
import numpy as np

#Define function to convert latitude and longitude to local XYZ
    coordinates
def latlon_to_xyz(latitude,longitude):
  #Latitude in format DDMM.MMMMM
  #Longitude in format DDDMM.MMMMM
  lat_sign = 1 if latitude[1] == "N" else -1
  lon_sign = 1 if longitude[1] == "E" else -1
  lat_deg_min    = [int(latitude[0][:2]),float(latitude[0][2:])]
  lon_deg_min    = [int(longitude[0][:3]),float(longitude[0][3:])]
  lat  = ((float(lat_deg_min[0])
              + float(lat_deg_min[1])/60.0)*lat_sign)
  lon = ((float(lon_deg_min[0])
              + float(lon_deg_min[1])/60.0)*lon_sign)
  return (utm.from_latlon(lat, lon)[0], utm.from_latlon(lat, lon)[1])

state = {} #Initialize state as an empty python dictionary
GPS_rate = 1 #Establish GPS_rate in Hz
with open('GPS.txt', 'r') as f:
  sentences = f.readlines()
  for line in sentences:
    #Retrieve position and velocity from GGA sentences
    if line[3:6] == 'GGA':
      gga  = line.replace("\r\n","").split(",")
      timestamp = float(gga[1])
      lat = (str(gga[2]),str(gga[3]))
      lon = (str(gga[4]),str(gga[5]))
      x, y = latlon_to_xyz(lat,lon)
      z = float(gga[9])
      position = [x,y,z]
      #Try-except to avoid errors when initialising the system
      try:
        velocity = [(x - y)/GPS_rate for x, y in
                        zip(position, state[timestamp-GPS_rate][0:3])]
      except:
        velocity = [0,0,0]
      state[timestamp] = position + velocity

#Routine to correct velocity using RMC sentences
with open('GPS.txt', 'r') as f:
  sentences = f.readlines()
  for line in sentences:
    if line[3:6] == 'RMC':
      rmc = line.replace("\r\n","").split(",")
      speed_rmc = float(rmc[7])*0.51444 #Knots to m/s
      timestamp = float(rmc[1])
      velocity_gga = np.array(state[timestamp][3:6])
      speed_gga = np.linalg.norm(velocity_gga)
      correction_factor = speed_rmc/speed_gga
      state[timestamp][3:6] = [component * correction_factor
                                for component in velocity_gga]

#Convert dictionary to list to it can be written in CSV format
state_list = []
for key in state:
  state[key].insert(0,key)
  state_list.append(state[key])

#Output results to CSV format
with open("gps_out.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(state_list)
```

Listing 2: gps_parsing.py

Listing 2 shows the Python snippet that performs the parsing of the NMEA sentence type *GGA*. The module *csv* from the Python standard library [13] is used to easily output the data to Comma-Separated Values format [14]. A function is defined to help transform latitude and longitude strings into relative $x$ and $y$ coordinates. For this purpose, the *utm* library [15] was imported and used to apply the Universal Transverse Mercator, which is a system for assigning coordinates to locations on the surface of the Earth. It divides earth into 60 zones and projects each to the plane as a basis for its coordinates. Specifying a location means specifying the zone and the x, y coordinate in that plane [16]. The velocity is computed as the difference between the actual position and the last position divided by the GPS rate, for it is a much simpler approach than trying to project the velocity vector onto the UTM coordinates chosen. To minimize the error derived from this approach, the velocity calculated through the aforementioned method is corrected with the speed over ground measured by the GPS by parsing the *RMC* sentence.

On the other hand, the inertial data supplied by our peers is already in a format that is suitable to be implemented in the filter. Thus, no further data wrangling is needed to proceed with the analysis.

### 2.4 Sensor Data Fusion

The actual Kalman Filter implementation will be done through MATLAB for simplicity, but the basis for the implementation through the Ada programming language will be laid out and the structure of this approach will be constructed. Ada is an international technical standard, defined both by ISO (International Organization for Standardization) and the IEC (International Electrotechnical Commission) [17], commonly known as Ada 2012.

### *2.4.1 Ada Programming Language*

#### 2.4.1.1 History

The Ada programming language (Ada for short) owes its name to Ada Lovelace (1815–1852), who has been credited as the first computer programmer in history [18]. Its original development was carried out by a team under the supervision of a French computer scientist named Jean Ichbiah from Bull SAS (a French computer company headquartered in Paris) under contract to the United States Department of Defense (DoD) from 1977 to 1983 to replace over 450 programming languages used by the DoD at that time [19].

In the 1970s the US Department of Defense (DoD) was worried about the number of different programming languages being used for its embedded computer system projects, none of which supported safe modular programming and many of which were obsolete or hardware-dependent. In 1975, the High Order Language Working Group (HOLWG) [20] was formed with the intent to reduce this number by finding or creating a programming language suitable for the department's requirements. After many iterations the eventual programming language was named Ada.

The language became an ANSI standard in 1983 [21], and after translation in French it also became an ISO standard in 1987 [22]. This version of the language is commonly known as Ada 83, from the date of its adoption by ANSI, but is sometimes referred to also as Ada 87, from the date of its adoption by ISO.

Ada 95, the joint ISO/ANSI standard [23] was published in February 1995, making Ada 95 the first ISO standard object-oriented programming language. The US Air Force funded the development of the GNAT Compiler to help with the standard revision and future acceptance. Presently, the GNAT Compiler is part of the GNU Compiler Collection.

Efforts have been put into improving and updating the technical content of the Ada language. At the Ada-Europe 2012 conference, the Ada Resource Association (ARA) and Ada-Europe announced the completion of the design of the latest version of the Ada language and the submission of the reference manual to the ISO for approval, which was published in December 2012 [17].

#### 2.4.1.2 Features

The Ada programming language was designed from its inception to be used in applications where safety and security are of the utmost importance. Its feature set and programming paradigms, by design, allow software developers to develop applications more effectively and efficiently. It encourages a "think first, code later" principle which produces more readable, reliable, and maintainable software.

The SPARK programming language [24] is a formally verifiable subset of the Ada language which allows developers to mathematically prove program correctness through static means. This is accomplished by eliminating the features of the language that introduce ambiguity and non-deterministic behavior. The language put together with a verification toolset and a design methodology ensures the development and deployment of low-defect software for high reliability applications [25].

Ada is a structured, statically typed, imperative, and object-oriented high-level programming language, extended from Pascal and other languages. It has built-in language support for design by contract (DbC), extremely strong typing, explicit concurrency, tasks, synchronous message passing, protected objects, and non-determinism. Ada improves code safety and maintainability by using the compiler to find errors in favor of runtime errors.

The syntax of Ada minimizes choices of ways to perform basic operations, and prefers English keywords (such as "or else" and "and then") to symbols (such as "||" and "&&"). Ada uses the basic arithmetical operators "+", "-", "*", and "/", but avoids using other symbols. Code blocks are delimited by words such as "declare", "begin", and "end", where the "end" (in most cases) is followed by the identifier of the block it closes (e.g., if ... end if, loop ... end loop). In the case of conditional blocks, this avoids a dangling else that could pair with the wrong nested if-expression in other languages like C or Java.

Ada is designed for developing very large software systems. Ada packages can be compiled separately. Ada package specifications (the package interface) can also be compiled separately without the implementation to check for consistency. This makes it possible to detect problems early during the design phase, before implementation starts.

These features make the Ada programming language (and the rest of the AdaCore Technologies) an ideal tool to approach the design with compliance to the primary document by which the certification authorities such as FAA, EASA and Transport Canada approve all commercial software-based aerospace system [26]; namely, the DO-178C (which was to be expected since the reason Ada was created in the first place was to be able to comply with this type of standard).

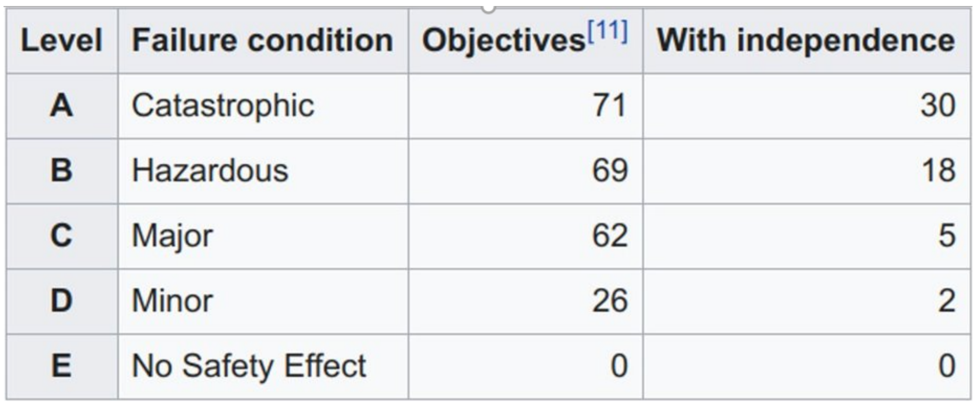
### *2.4.2 DO-178C*

DO-178C [27], Software Considerations in Airborne Systems and Equipment Certification is published by RTCA (Radio Technical Commission for Aeronautics), in a joint effort with EUROCAE (European Organisation for Civil Aviation Equipment), and replaces DO-178B [28]. The new document is called DO-178C/ED-12C and was completed in November 2011 and approved by the RTCA in December 2011.

The FAA published a document [29] on July 2013, making DO-178C a recognized "acceptable means, but not the only means, for showing compliance with the applicable airworthiness regulations for the software aspects of airborne systems and equipment certification."

The document attributes to every system a software level, also known as the Design Assurance Level (DAL), which is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system. The failure conditions are categorized by their effects on the aircraft, crew, and passengers:

- **A - Catastrophic**: Failure may cause deaths, usually with loss of the airplane.
- **B - Hazardous**: Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
- **C - Major**: Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries).
- **D - Minor**: Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change.
- **E - No Effect**: Failure has no impact on safety, aircraft operation, or crew workload.



| Level | Failure condition | Objectives[11] | With independence |
|---|---|---|---|
| A | Catastrophic | 71 | 30 |
| B | Hazardous | 69 | 18 |
| C | Major | 62 | 5 |
| D | Minor | 26 | 2 |
| E | No Safety Effect | 0 | 0 |

Figure 2.2: Number of objectives to be satisfied depending on the DAL

Depending on the DAL, the document establishes a number of objectives to be satisfied by the systems. The number of objectives to be satisfied (some with independence) is determined by the software level A-E. The phrase "with independence" refers to a separation of responsibilities where the objectivity of the verification and validation processes is ensured by virtue of their "independence" from the software development team. For objectives that must be satisfied with independence, the person verifying the item shall not be the person who authored the item and this separation must be clearly documented. Figure 2.1 displays the number of objectives the systems have to meet depending on the DAL that has been attributed to them.

DO-178 requires a high traceabiliy, which is a connection (called a trace) between the certification artifacts. For example, a Low Level Requirement (LLR) traces up to a High Level Requirement (HLR). A traceability analysis is then used to ensure that each requirement is fulfilled by the source code, that each requirement is tested, that each line of source code has a purpose (is connected to a requirement), and so forth. Traceability ensures the system is complete. The rigor and detail of the certification artifacts is related to the software level.

Certifying that a system complies with this standard is a challenging task (very often the certification of the software constitutes the largest portion of the budget in a software development project), especially for the verification activities, but appropriate usage of qualified tools and specialized runtime libraries can significantly simplify the effort. For this purpose, the utilisation of the technologies offered by AdaCore is very convenient [30], since it helps to cover not only the core DO-178C document but also the technology supplements such as Model-Based Development and Verification (DO-331 / ED-218) [31], Object-Oriented Technology and Related Techniques (DO-332 / ED-217) [32], and Formal Methods (DO-333 / ED-216) [33].

All this information justifies the choice of the Ada programming language for the development of the filtering algorithm, since the design of this software system would already be oriented towards certification, which we consider that it should be the main goal of any engineering design. Although the basis for the Ada implementation will be laid out, the considerations for certifying the software will not be discussed in this project. This line of research is left open for anyone wishing to continue with it, but beware that the extension of it may be more appropriate for a doctoral thesis than it is for a course project.

# 3 Theoretical Analysis and Problem Formulation

## 3.1 Kalman Filter

Kalman filtering (linear quadratic estimation) is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone [34].

Therefore, the Kalman filter is a set of mathematical equations that provide an efficient recursive solution through the least-squares method. The objective of this solution is to calculate a linear, unbiased, and optimal estimator of the state of the system at the moment of time $t$ based on the information available at time $t_{-1}$ and update, with the additional information available at time $t$, these estimates [35].

The filter is a mathematical procedure that operates through a mechanism of prediction and correction. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty [36].

Likewise, the algorithm is recursive. It can run in real-time, using only the present input measurements and the previously calculated state and its uncertainty matrix. Consequently, the filter can predict the system state in the past, present, and future, even when the modeled system's precise nature is unknown.

According to the linear systems theory, the linear system dynamics can be represented by a state-space model, where a set of first-order differential equations express deviations from a reference trajectory.

The equations of the algorithm are as follows [37]:

State-space model

$$x_{k+1} = \Phi_{k+1,k} x_k + w_k \tag{3.1}$$

$$z_k = H_k x_k + v_k \tag{3.2}$$

Where $w_k$ and $v_k$ are white Gaussian noises (without autocorrelation), independent, of null mean and with covariance matrices $Q_k$ and $R_k$ respectively.

Likewise, White noise is the one that tends to zero with the evolution of the system. It is a random noise that appears as a result of errors in the measurements or the prediction made by the dynamic model of the system variables. It is called noise or uncertainty because it involves small variations on actual results.

Q and R indicate the dispersion degree of the differences between the predicted and measured values, respectively, around their mean, from one iteration to another. When this difference is kept constant and in a reasonable environment of the mean, the covariance will maintain a stable value, but when one of these differences is very different from the trend that was followed, the filter assumes that the value of the current iteration has a lot of error or uncertainty. However, if it is not an isolated case, from the moment that differences begin again to follow this new trend, the filter will adjust to these new data to return a correct estimate.

Initialization (k=0)

$$\hat{x_0} = E[x_0] \tag{3.3}$$

$$P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T] \tag{3.4}$$

Calculation for k = 1,2...

Calculation of the state estimated spread at the beginning:

$$\hat{x_k}^- = \Phi_{k,k-1} \hat{x_{k-1}} \tag{3.5}$$

Calculation of the spread of the covariance at the beginning:

$$P_k^- = \Phi_{k,k-1} P_{k-1} \Phi_{k,k-1} + Q_{k-1} \tag{3.6}$$

Kalman profit matrix calculation:

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{3.7}$$

Estimated post status update:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k \hat{x}_k^-) \tag{3.8}$$

Update of the estimate covariance:

$$P_k = (I - K_k H_k) P_k^- \tag{3.9}$$

## 3.2 Ada Implementation

The basic structure of the general Kalman Filter implementation in Ada is displayed in Listing 3. It is designed as an Ada generic package, which is an application-independent approach of a specific update method. The package with the dimensions of the state vector is instantiated and the model is supplied data in procedure calls before the actual prediction and update steps are carried out. These definitions may also be hidden alternatively by placing them in the private part of the package specification. In this design, the measurement model is defined as:

$$z = Hx + v \tag{3.10}$$

where $x$ is the error state, $v$ is white gaussian measurement noise and $H$ is the measurement vector, whereas the residual $r$ is computed as:

$$r = z - Hx \tag{3.11}$$

Most of the functionalities and characteristics of the Ada programming language used in the structure defined in Listing 3 are explained in the following paragraphs, to facilitate the comprehension of the code to those unfamiliar with Ada syntax and properties, such as specific packages, generics and modular programming.

```ada
with Ada.Numerics.Real_Arrays;
use Ada.Numerics.Real_Arrays;
generic
  type Real is digits <>;  -- used to define any floating-point type
  N : Positive; -- N represents the dimension of the state
  M : positive; -- M represents the number of measurements
package generalKalmanFilter is
  type State is Real_vector (1..N);
  type stateMatrix is Real_Matrix (1..N, 1..M);
  type Measurements is Real_Matrix (1..N, 1..M);
  procedure defineStateMatrix (F : in stateMatrix);
  procedure defineNoise (Q : in stateMatrix);
  procedure defineMeasurements (h : in Measurements);
  procedure defineMeasurementsNoise (r : in Real);
  procedure predictionStep (X : in out State;
                P : in out stateMatrix);
  procedure updateStep (z : in Real;
              X : in out State;
              P : in out stateMatrix;
              R : out Real);
end generalKalmanFilter;
```

Listing 3: generalKalmanFilter.adb

The *Ada.Numerics.Real_ Arrays* package provides support for vectors and matrices. It includes common matrix operations such as inverse, determinant and eigenvalues. Vectors and matrices can be declared using the *Real_ Vector* and *Real_ Vector* types, respectively [38].

Generics are used for metaprogramming in Ada. They are useful for abstract algorithms that share common properties with each other [39]. A generic is declared by using the keyword *generics*.

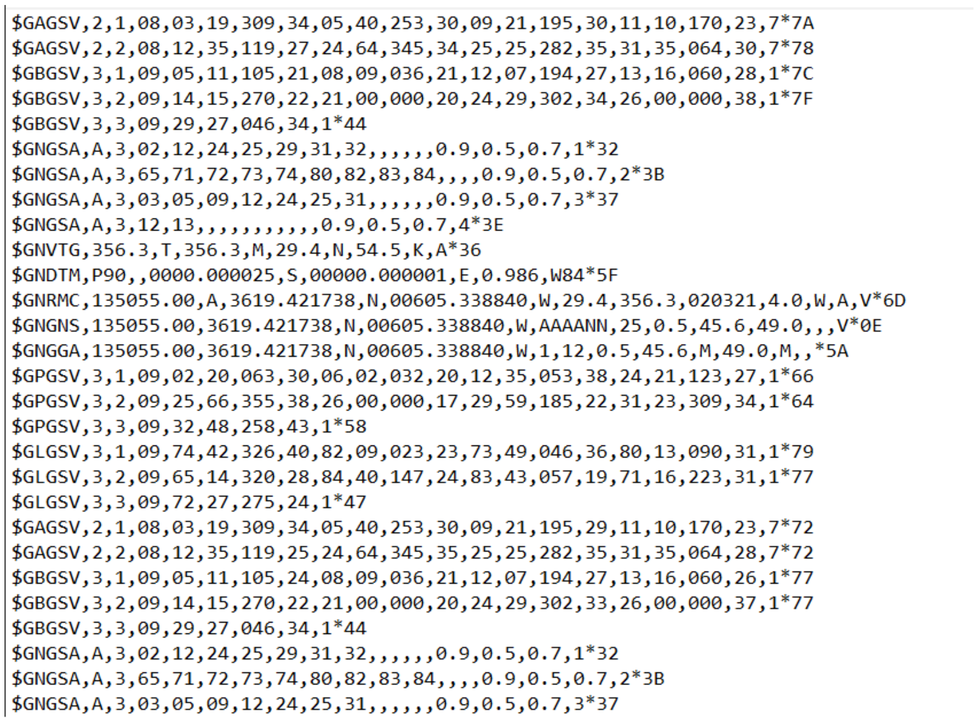
The structure of the main program is performed through *packages*. Ada packages let code be modular, separating programs into semantically significant units. Additionally the separation of a package specification from its body can reduce compilation time. While the *with* clause indicates a dependency, the prefix of the name of the package is no longer needed since a *use* clause has been included. Accessing entities from a package uses the dot notation, *Package.Entity*. Finally, a *with* clause can only appear in the prelude of a compilation unit, that is, before the reserved word, such as *procedure*, that marks the beginning of the unit. It is not allowed anywhere else. This rule is only needed for methodological reasons, since the person reading the code should be able to see immediately which units the code depends on [40].

In statically typed languages, a type is mainly (but not only) a compile time construct. It is a construct to enforce invariants about the behavior of a program. Invariants are unchangeable properties that hold for all variables of a given type. Enforcing them ensures, for example, that variables of a data type never have invalid values [41].

# 4 Data Acquisition and Wrangling

| Timestamp (ns) | X (m) | Y (m) | Z (m) | Vx (m/s) | Vy (m/s) | Vz (m/s) |
|---|---|---|---|---|---|---|
| 6.81E+13 | 761270.48 | 4024265.13 | 48.9 | -2.2 | 20.0 | 0.2 |
| 6.82E+13 | 761268.30 | 4024285.20 | 49.2 | -2.1 | 19.7 | 0.3 |
| 6.83E+13 | 761266.20 | 4024305.33 | 49.2 | -2.0 | 19.5 | 0 |

Table 4.1: Example of Inertial Data



```
$GAGSV,2,1,08,03,19,309,34,05,40,253,30,09,21,195,30,11,10,170,23,7*7A
$GAGSV,2,2,08,12,35,119,27,24,64,345,34,25,25,282,35,31,35,064,30,7*78
$GBGSV,3,1,09,05,11,105,21,08,09,036,21,12,07,194,27,13,16,060,28,1*7C
$GBGSV,3,2,09,14,15,270,22,21,00,000,20,24,29,302,34,26,00,000,38,1*7F
$GBGSV,3,3,09,29,27,046,34,1*44
$GNGSA,A,3,02,12,24,25,29,31,32,,,,,,0.9,0.5,0.7,1*32
$GNGSA,A,3,65,71,72,73,74,80,82,83,84,,,,0.9,0.5,0.7,2*3B
$GNGSA,A,3,03,05,09,12,24,25,31,,,,,,0.9,0.5,0.7,3*37
$GNGSA,A,3,12,13,,,,,,,,,,,0.9,0.5,0.7,4*3E
$GNVTG,356.3,T,356.3,M,29.4,N,54.5,K,A*36
$GNDTM,P90,,0000.000025,S,00000.000001,E,0.986,W84*5F
$GNRMC,135055.00,A,3619.421738,N,00605.338840,W,29.4,356.3,020321,4.0,W,A,V*6D
$GNGNS,135055.00,3619.421738,N,00605.338840,W,AAAANN,25,0.5,45.6,49.0,,,V*0E
$GNGGA,135055.00,3619.421738,N,00605.338840,W,1,12,0.5,45.6,M,49.0,M,,*5A
$GPGSV,3,1,09,02,20,063,30,06,02,032,20,12,35,053,38,24,21,123,27,1*66
$GPGSV,3,2,09,25,66,355,38,26,00,000,17,29,59,185,22,31,23,309,34,1*64
$GPGSV,3,3,09,32,48,258,43,1*58
$GLGSV,3,1,09,74,42,326,40,82,09,023,23,73,49,046,36,80,13,090,31,1*79
$GLGSV,3,2,09,65,14,320,28,84,40,147,24,83,43,057,19,71,16,223,31,1*77
$GLGSV,3,3,09,72,27,275,24,1*47
$GAGSV,2,1,08,03,19,309,34,05,40,253,30,09,21,195,29,11,10,170,23,7*72
$GAGSV,2,2,08,12,35,119,25,24,64,345,35,25,25,282,35,31,35,064,28,7*72
$GBGSV,3,1,09,05,11,105,24,08,09,036,21,12,07,194,27,13,16,060,26,1*77
$GBGSV,3,2,09,14,15,270,22,21,00,000,20,24,29,302,33,26,00,000,37,1*77
$GBGSV,3,3,09,29,27,046,34,1*44
$GNGSA,A,3,02,12,24,25,29,31,32,,,,,,0.9,0.5,0.7,1*32
$GNGSA,A,3,65,71,72,73,74,80,82,83,84,,,,0.9,0.5,0.7,2*3B
$GNGSA,A,3,03,05,09,12,24,25,31,,,,,,0.9,0.5,0.7,3*37
```

Figure 4.1: Example of GPS raw output.

Figure 4.1 displays an extract of the raw output provided by the GPS receiver. The file is in *.txt* format as specified in Section 2. The process of data wrangling transforms these data into the format showed in Table 4.1.

On the other hand, the inertial data does not need wrangling since the format is already adequate to our application and the acquisition is discussed in our peers' project. Therefore, all the data acquisition and adjustment to the corresponding format is complete at this point so we can proceed to the actual fusion of the data and visualization.

## 5 Data Fusion

GPS and INS navigation systems are complimentary. On the one hand, the GPS gives a high precision but discrete measurement in time, whose error is limited and low bandwidth. On the other hand, the INS provides a continuous estimate over time, where its error grows with it and with high bandwidth. Therefore, for the integration of both systems, a Kalman filter is used to minimize the final errors in the estimation of the position and obtain results that are practically exact to the real one.

The following code implements the function available in Matlab to apply a Kalman filtering on the data fusion. Likewise, it is important to note that this application of the Kalman filter is off-line, which means that it has been done after data collection.

```
%KALMAN FILTER USING MATLAB FUNCTION
clc; format long;

%INICIALITATION. DATA COLLECTION
%Integration interval
t = 0.02;
X0 = zeros(6,1);
P0 = 400*eye(6);
A = [1 0 0 t 0 0; 0 1 0 0 t 0; 0 0 1 0 0 t; 0 0 0 1 0 0; 0 0 0 0 1 0; 0
    0 0 0 0 1];
C = eye(6);

Solution = [];
V_IMU=[];
P_IMU=[];
MATRIX_GPS=[];

Q = eye(6);
R = eye(6);

H = dsp.KalmanFilter('StateTransitionMatrix',A, 'ControlInputPort',false
    ,'MeasurementMatrix',C, 'ProcessNoiseCovariance',Q, '
    MeasurementNoiseCovariance',R,'InitialStateEstimate',X0, '
    InitialErrorCovarianceEstimate',P0);

j=1;
k=0;

for i=1:2801

Time_IMU = IMU_ACC(i,1);
```

```
Yaw = IMU_GYRO(i,2);
Pitch = IMU_GYRO(i,3);
Roll = IMU_GYRO(i,4);
Ax_IMU = IMU_ACC(i,2);
Ay_IMU = IMU_ACC(i,3);
Az_IMU = IMU_ACC(i,4);

x_GPS = GPS(j,2);
y_GPS = GPS(j,3);
z_GPS = GPS(j,4);

Vx_GPS = GPS(j,5);
Vy_GPS = GPS(j,6);
Vz_GPS = GPS(j,7);

MATRIX_GPS=[MATRIX_GPS;x_GPS y_GPS z_GPS Vx_GPS Vy_GPS Vz_GPS];

Arot_IMU = rotation_IMU (Pitch, Roll, Yaw, Ax_IMU, Ay_IMU, Az_IMU);

Vel_IMU = quadv(@(x)descom(x,Arot_IMU),0,t);
Pos_IMU = quadv(@(x)descom(x,Vel_IMU),0,t);

Posx_IMU=Pos_IMU(1)*100;
Posy_IMU=Pos_IMU(2)*100;

V_IMU=[V_IMU; Vel_IMU];
P_IMU=[P_IMU; Pos_IMU];

%DATA FUSION
x_FUS = x_GPS + Posx_IMU;
y_FUS = y_GPS + Posy_IMU;
z_FUS = z_GPS + Pos_IMU(3);

Vx_FUS = Vx_GPS + Vel_IMU(1);
Vy_FUS = Vy_GPS + Vel_IMU(2);
Vz_FUS = Vz_GPS + Vel_IMU(3);

%STATES VECTOR
Med_FUS = [x_FUS; y_FUS; z_FUS; Vx_FUS; Vy_FUS; Vz_FUS];
Kalman = step(H, Med_FUS);
Solution = [Solution; Kalman(1) Kalman(2) Kalman(3) Kalman(4) Kalman(5)
    Kalman(6)];

k=k+1;

if k==50
    k=0;
    j=j+1;
end

end

display(Solution)
```

Listing 4: KalmanFilterMatlab.m

```
function AR = Arotation_IMU (Pitch,Roll,Yaw,Acel_x,Acel_y,Acel_z)

Rz_yaw = [cos(Yaw) -sin(Yaw) 0; sin(Yaw) cos(Yaw) 0; 0 0 1];
Ry_roll = [cos(Roll) 0 sin(Roll); 0 1 0; -sin(Roll) 0 cos(Roll)];
```

```
Rx_pitch = [1 0 0; 0 cos(Pitch) -sin(Pitch); 0 sin(Pitch) cos(Pitch)];

M = Rz_yaw*Ry_roll*Rx_pitch;
AR = M*[Acel_x; Acel_y; Acel_z];
end
```

Listing 5: Arotation.m

First of all, the movement is linear, so the new position will be based on the previous one. Due to the frequencies of each device are different, the faster one, the IMU, is used to interpolate the positions and speeds between fifty GPS measurement inputs. Every second and every 20 milliseconds a GPS input and data from the IMU will be received, respectively.

To calculate the position, first of all, the geographic coordinates (Latitude, Longitude, Altitude), which is how the GPS position data is received must be converted to cartesian coordinates. Afterwards, the axis rotation and integration of the values provided by the IMU will be determined.[42]

Accelerations provided by the IMU are expressed in the IMU's axes. However, all the measurements must be expressed on the same axis. Therefore, to be processed by the Kalman filters at the same time as the GPS's measurement, they have to be expressed in the geographical axes from the earth, this is achieved by using the function $ArotationIMU.m$

Likewise, it should be noted that the pitch angle ($\theta$) is the one that indicates rotation about the X axis called, the roll angle ($\varphi$) indicates rotation about the Y axis, and the yaw angle ($\Psi$) indicates rotation around the Z axis.

Once the accelerations have been rotated to geographic axes, it is necessary to integrate the values obtained once to obtain velocities, and twice to obtain position values. Matlab's function quadv has been used for this purpose, which allows working with vectors.

On the other hand, since the sensors provide the same type of information about movement, it is necessary to merge both signals in order to enter the filter as a single source for measuring positions and speeds. To this aim, the data provided by both sensors have been expressed in the same coordinate system and the same units (rad, m and s).

Therefore, measurements of position and speed that pass through the filter are a GPS's measurements and IMU's measurements composition.

$$Position = (x_{IMU} + x_{GPS}, y_{IMU} + y_{GPS}, z_{IMU} + z_{GPS}) \tag{5.1}$$

$$Speed = (Vx_{IMU} + Vx_{GPS}, Vy_{IMU} + Vy_{GPS}, Vz_{IMU} + Vz_{GPS}) \tag{5.2}$$

Below are some of the results obtained after executing the code.

To sum up, Kalman filter is not able to eliminate errors in the calibration of the sensors or in the modeling of the dynamic system that imply large variations on the real results.[42] This is what happened in this case, the measurements taken by the IMU were not entirely reliable, so the measurements with the greatest weight were those of the GPS. Nevertheless, the general results are adequate since they effectively follow the values obtained from the data fusion.

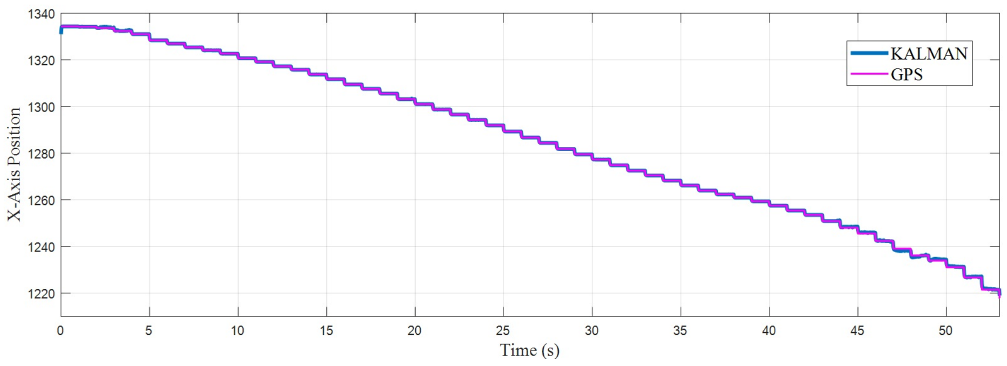


Figure 5.1: X-Axis Position.

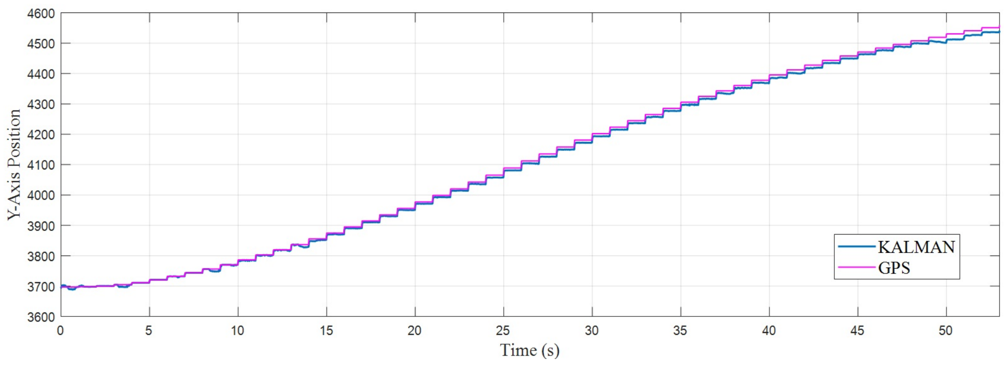


Figure 5.2: Y-Axis Position.

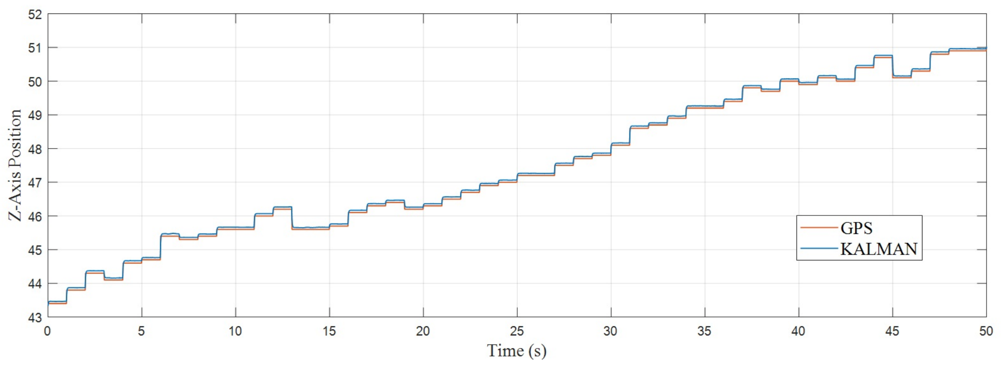


Figure 5.3: Z-Axis Position.

Therefore, it is important to highlight the idea that a Kalman filter is not capable of deciding whether the measure or the model is correct or wrong. Its only task is to estimate, within the available data, an optimal solution free of random noise.

## 6 Conclusion

The airborne software systems certification requirements have been briefly studied and applied to the case study of the Kalman Filter, one of the most widely used algorithms in filtering and control theory. The basis for an Ada based certification-oriented approach have been laid out in a generic form opening the way for further investigation related to this topic.

On the other hand, the data manipulation required for carrying out the filtering and the Kalman algorithm itself have been developed using different technologies, such as Python and Matlab, which are de-facto standards in the development of industry avionics systems.

This project also allowed us to expand our knowledge in topics seen during this Navigation Aids Systems Course, such as the NMEA 0183 standard and to introduce ourselves to the complex world of the electronic components, where we started facing a black box and ended up extracting useful information from the datasheets and manual, which is an undoubtedly indispensable tool.

It is also important to note that the generic approach used to establish the basis for a Kalman Filter implementation through the Ada programming language has nonetheless a number of limitations and important trade-offs. For example, even though the covariance matrices are defined symmetric, extra matrix procedures may need to be defined if one were to try to save space and computation time by taking advantage of this situation. We must familiarise ourselves with this kind of solution compromises since it will be our jobs to face them.

It is acknowledged that the depth of each topic researched in our project is inversely proportional to the variety of topics investigated, which is why a list of futures lines of research has been constructed in order to promote further deepening in each of the aspects that define a project of this magnitude.

Finally, it is important to point out how essential it is for any design to be focused towards compliance with the applicable standards. This orientation must be carried out first of all to avoid iterations, which are inevitable if this is not taken into account. Regulations are of the upmost importance in our industry and being knowledgeable about them and able to comply with them is an invaluable skill that should be developed as much as possible.

## 7 Futures Lines of Research

- One possible future line of research would be the design of any number of the procedures defined within the basic structure of the Ada implementation, if not all of them.

- Another interesting future line of research is the study of the considerations needed to analyze the certification process of the algorithm design. AdaCore tools and technologies (including AdaCore Technologies for DO-178C / ED-12C book [30]) make this process easier but they are not open-source [43].

- Data visualization was in this case out of the scope of our project. However, since data is already in a format whose handling is straightforward, the development of a Python application to visualize the coordinates obtained from the Filter fusion algorithm could be an interesting extension of our project.

- Another line of research is the case of applying Kalman filtering in real-time. While data is collected, the filter returns its estimate to be able to automatically correct the state. Nevertheless, it would be necessary to implement, acquire, process and filter in a programming language suitable for the platform where it will be executed. In addition to the difficulty of implementing all matrix operations through coding. Furthermore, unforeseen situations can occur in real-time.

- From the final results, it can be concluded that the higher the quality of the data collection process is and the better calibrated the devices are, the more reliable the data extracted from them will be. Therefore, it would be interesting to improve the calibration and capacity of the devices.

## References

[1] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: `10.1115/1.3662552`. eprint: `https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35\_1.pdf`. URL: `https://doi.org/10.1115/1.3662552`.

[2] *UAV v2 Development Platform by SparkFun.* URL: `https://www.sparkfun.com/products/retired/9038`.

[3] *UAV v2 Development Platform User Manual.* URL: `https://www.sparkfun.com/datasheets/GPS/GPSUAV2Manual_v29.pdf`.

[4] *FTDI Basic Breakout.* URL: `https://www.sparkfun.com/products/9873`.

[5] *Hookup Guide.* URL: `https://learn.sparkfun.com/tutorials/sparkfun-usb-to-serial-uart-boards-hookup-guide`.

[6] *NMEA 0183.* URL: `https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard`.

[7] *NMEA FAQ.* URL: `https://web.archive.org/web/20140215150802/http://www.kh-gps.de/nmea.faq`.

[8] *NMEA Senteces Table.* URL: `https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_MessageOverview.html`.

[9] *CuteCom.* URL: `http://cutecom.sourceforge.net/`.

[10] *pySerial Library.* URL: `https://pypi.org/project/pyserial/`.

[11] *9 Degrees of Freedom - Razor IMU.* URL: `https://www.sparkfun.com/products/retired/10736`.

[12] *PYPL PopularitY of Programming Language.* URL: `https://pypl.github.io/PYPL.html`.

[13] *CSV File Reading and Writing.* URL: `https://docs.python.org/3/library/csv.html`.

[14] *CSV format.* URL: `https://tools.ietf.org/html/rfc2048`.

[15] *UTM library.* URL: `https://pypi.org/project/utm/0.4.2/`.

[16] John P. Snyder. *Map projections: A working manual.* U.S. Government Printing Office, 1987.

[17] *ISO/IEC 8652:2012 Information technology — Programming languages — Ada.* Standard. International Organization for Standardization/ International Electrotechnical Commission, Dec. 2012.

[18] J. Fuegi and J. Francis. “Lovelace Babbage and the creation of the 1843 ’notes”’. In: *IEEE Annals of the History of Computing* 25.4 (2003), pp. 16–26. DOI: 10.1109/MAHC.2003.1253887.

[19] *The Ada Programming Language.* URL: https://web.archive.org/web/20160522063844/http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html.

[20] *The DoD High Order Language Working Group.* URL: http://archive.adaic.com/pol-hist/history/holwg-93/holwg-93.htm.

[21] *ANSI/MIL-STD 1815A.* URL: https://web.archive.org/web/20040625113309/http://archive.adaic.com/standards/83lrm/html/Welcome.html.

[22] *ISO 8652:1987 Programming languages — Ada.* Standard. International Organization for Standardization, June 1987.

[23] *ISO/IEC 8652:1995 Information technology — Programming languages — Ada.* Standard. International Organization for Standardization/ International Electrotechnical Commission, Feb. 1995.

[24] John W. McCormick and Peter C. Chapin. *Building High Integrity Applications with SPARK.* Sept. 2015. ISBN: 9781107656840.

[25] *Learn AdaCore.* URL: https://learn.adacore.com/about.html.

[26] *Who’s Using Ada?* URL: https://www2.seas.gwu.edu/~mfeldman/ada-project-summary.html#Banking_and_Financial_Systems.

[27] *Buy DO-178C.* URL: https://my.rtca.org/nc__store?search=do-178c.

[28] *DO-178B and DO-178C Differences.* URL: https://www.patmos-eng.com/do-254-training-do-178c-training/178b-178c-differences/.

[29] *AC 20-115C.* URL: https://web.archive.org/web/20140903075843/http://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-115C.pdf.

[30] Frédéric Pothon and Quentin Ochem. *AdaCore Technologies for DO-178C / ED-12C.* Mar. 2017.

[31] *D0-331 Guidance for Model-Based Development and Verification.* URL: https://www.sae.org/learn/content/c2008/.

[32] *D0-332 Object-Oriented Technology and Related Techniques.* URL: https://standards.globalspec.com/std/1461703/rtca-do-332.

[33] *D0-333 Formal Methods.* URL: https://www.rtca.org/training/do-333-formal-methods-do-178c-supplement/.

[34] Mohsen Kavehrad and Reza Aminikashani. *Visible Light Communication Based Indoor Localization.* 2019.

[35] *El filtro de Kalman.* URL: https://www.bccr.fi.cr/investigaciones-economicas/DocMetodosCuantitativos/Filtro_de_Kalman.pdf.

[36] Carlos Caravaca Gallego. *Design of Extended Kalman Filters for Estimation of UAV Flight Parameters.* Istanbul Technical University, 2020.

[37] Eric Pula Moreno Juan Carlos Fernandez-Portillo Vicho and Miguel Martínez De la Hidalga Martínez. *FILTRO DE KALMAN. INTEGRACIÓN TIGHT INS/GPS.* Universidad de Sevilla, 2012.

[38] *Ada Standard library.* URL: `https://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/standard_library_routines.html`.

[39] *Generics in Ada.* URL: `https://en.wikibooks.org/wiki/Ada_Programming/Generics`.

[40] *Ada packages.* URL: `https://www.adaic.org/resources/add_content/standards/05aarm/html/AA-7-1.html`.

[41] *Ada types.* URL: `https://www.radford.edu/~nokie/classes/320/fundamentals/fundTypes.html`.

[42] Maria Esther Aranda Romasanta. *Estudio y aplicación del Filtro de Kalman en fusión de sensores en UAVs.* Universidad de Sevilla, 2017.

[43] *AdaCore Tools.* URL: `https://www.adacore.com/gnatpro/toolsuite`.