The Czech Technical University in Prague
Faculty of Electrical Engineering

# SYSTEM FOR AUTOMATIC AIRCRAFT OBSERVATION AND RECORDING BASED ON ADS-B

2016 Dávid Hriadel

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

# BACHELOR PROJECT ASSIGNMENT

Student: **Dávid Hriadel**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **System for automatic aircraft observation and recording based on ADS-B**

Guidelines:

1. Build an automatic system for aircraft observation and recording that should serve for plane spotters in vicinity of an airport. The system will be capable of determining the position of the observer using GPS, reorienting a camera to smoothly follow a plane selected by the observer, and localizing planes based on receiving ADS-B packets.
2. Survey similar (relevant systems) that are available on the market or were reported in the literature or on the Internet.
3. Optionally you can implement also fine-tracking of the planes based on computer vision and listening to ATC communication.

Bibliography/Sources:

[1] Corke, Peter. Robotics, vision and control: fundamental algorithms in MATLAB. Vol. 73. Springer, 2011.
[2] Articles on ADS-B from Hackaday [online], https://hackaday.com/tag/ads-b/

Bachelor Project Supervisor: Ing. Jiří Zemánek

Valid until the summer semester 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 1, 2016

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

In Prague date ..... .....

**Abstract**

The thesis describes the development of a system for automatic aircraft observation and recording using ADS-B surveillance system. The system process ADS-B messages acquired from aircrafts using suitable receiver and plots them on the map, allowing user to choice the plane he would like to capture through the built-in touchscreen. Furthermore it determines its own position on the Earth with GPS and its magnetic heading required for successful aircraft tracking. After selecting the aircraft desired to watch, servomotors try to smoothly and precisely move the camera so the object is recorder to the video file. The system is powered by batteries and is assembled to the mobile plastic board capable of mounting to a tripod—to allow the system be used on any place.

**Dedication**

I would like to thank my project supervisor, Ing. Jiří Zemánek, for the patient guidance and valuable advices he has provided throughout the project and to other people in the Department of Control Engineering who helped me with any kind of the problem.

# Contents

**Introduction** **3**
Inspiration . . . 3

**1 Hardware implementation** **5**
1.1 ADS-B Receiver . . . 5
1.2 Position and heading determination . . . 10
1.3 Pan/Tilt platform . . . 16
1.4 Computer . . . 17
1.5 Input/Output interface . . . 18
1.6 Camera . . . 19
1.7 Power supply . . . 20
1.8 Final assembly . . . 21

**2 Software implementation** **25**
2.1 Processing ADS-B messages . . . 25
2.2 Processing GPS messages . . . 27
2.3 Processing BNO055 data . . . 27
2.4 Servo control . . . 27
2.5 Graphical User Interface . . . 28
2.6 Tracking algorithm . . . 29

**3 System test** **35**

**Conclusions** **37**

**Bibliography** **39**

**List of Figures** **43**

**List of Tables** **45**

**List of Abbreviations** **47**

**Attachments** **49**

# Introduction

Nowadays, planespotting is one of a many hobbies favoured by aviation enthusiatists. They are coming to airports with desire to watch aircrafts departing and landing. Eventually, they are equipped with cameras and take photos and videos of aircrafts, which later share with the aviation community. In the next section my commentary about current possibilities for plane spotters can be found, including my inspiration for the thesis.

My task is to build a device that enhances spotter's job. The system tracks nearby aircrafts using Automatic Dependent Surveillance - Broadcast (ADS-B) technology, the most precise surveillance system for civil Air Trafic Control (ATC). It gives operator a choice which airplane he would like to capture on the virtual radar screen with displayed aircrafts, automatically points the camera towards the target, and records the video. The quality of capture should be remarkable, so the spotter could later process images and share them with others. There are also more possibilities of application—with the internet connection, air traffic sharing sites could be feeded with the received data, or even the view of a camera could be provided for the whole community by streaming a video directly on the internet.

The work is divided into two major parts. Firstly, possibilities of physical completion must be surveyed. I must decide which is the best solution of individual subproblems for me to make the system correctly functioning. These include receiving ADS-B packets from aircrafts, determining my own position, choosing a computational device, its power supply, and motorized camera holder. Even during a hardware picking, I have to bear in a mind how is the part operated from the software's side. In the end, all components should be assembled onto a mobile platform.

Secondly, the software responsible for the aircraft observation must be implemented. It should be able to quickly process received aircraft's data and data from sensors, plot aircrafts on the map, and smoothly move the camera towards chosen airplane. The precision of aircraft tracking with ADS-B is also analyzed.

Finally, the system has been tested near the Prague's airport. Its performance and results are discussed in the third chapter with possibilities for further improvement.

## Inspiration

My interest in the aviation led me to survey the internet for Do It Yourself (DIY) projects related to the aviation. The first project that caught my attention was the creation of a cheap ADS-B radar at home (project details can be found on `http://www.rtl-sdr.com/adsb-aircraft-radar-with-rtl-sdr/`). That sounds like amazing thing having own aircraft tracker.

The question is what could I do with all positional information about aircrafts. Many people support air traffic sharing sites with their captured data and thanks to them now almost everybody can watch live air traffic around their cities. Most popular ones are `https://www.flightradar24.com/` and `https://flightaware.com/live/`. Watching airplanes fly can be enhanced with listening ATC communication that is also available to everybody on the website `http://www.liveatc.net/`.

Furthermore, there are various nice projects providing live video from airport's runways allowing people at home observating aircrafts during take offs and landings. The best view of Prague's Vaclav Havel airport is on `http://slowtv.playtvak.cz/planespotting-0pr-/planespotting.aspx?c=A150624_164934_planespotting_cat`. The great idea was combination of re-

ceiving ADS-B messages and making video of airplanes, by Simon Aubury—his project Pi Plane was recording flights over his house in Australia and streaming video on the Internet. Details of the project are on the website http://simonaubury.com/the-pi-plane-project1-introduction/ and its design is in the Figure 1.

I have found this project very interesting and immediately I was thinking how can I improve the idea—I do not live by runway, so I want the project to be mobile. Aircrafts are pretty far away thus better camera's zoom is required. I want the device to behave like radar screen—all captured information should be displayed including aircrafts. And the camera should follow the plane smoothly to improve the quality of the video. Also combination of the live view from the runway and precise aircraft tracking sounds like it could take the viewer's experience to the new level. These thoughts encouraged me to do this bachelor thesis.


Figure 1: Simon Aubury's Pi Plane project.

# 1. Hardware implementation

In this chapter, construction of the system is analyzed with considering many aspects during the solution. The system consists from more individual components, connected together in the software. Illustrated in the Figure 1.1, I have divided the problem into logical blocks each having a different task. The brief description of these parts is following:

- a core of the project, some kind of a computer, is required for communicating with sensors and controlling actuators,
- an ADS-B receiver is the essential part for tracking aircrafts,
- to correctly rotate the camera my position and heading have to be known,
- a platform that is capable of moving camera in two axes—called Pan/Tilt,
- a camera itself with lens,
- a way to interact with the system, and
- a power supply to make the system mobile.

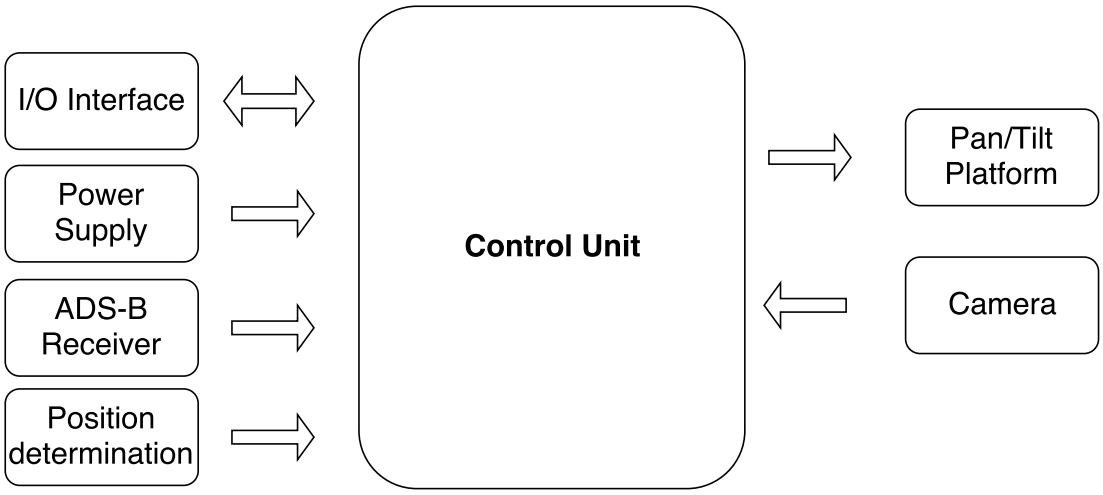



Figure 1.1: The scheme of system's components.

## 1.1 ADS-B Receiver

To begin with, I have to decide how I will receive and decode ADS-B packets and the rest of the system will adjust to such a choice. But firstly let me take a look at a brief history of surveillance systems, resulting in an invention of the ADS-B technology.

### History of surveillance systems

The need of airplane tracking for military and civil purposes resulted in a creation of few technologies that are capable of doing this job. The first one now called Primary Surveillance Radar (PSR), uses technology invented during Second World War. The ground radar's transmitter sends electromagnetic waves that are reflected from objects

and received by radar's receiver. The distance between the radar and the object is calculated using the time-of-flight principle. Radar's antenna also continously rotates, so the object bearing could be known. This system is independent from the object's behavior and can detect anything what reliably reflects electromagnetic waves. Metal airplane's frames have very good reflective properties making such system very efficient. However, both military and civil controllers would like to know more details about aircraft, not only that there is one—for example identification, altitude, velocity. Although some information radars could find out themselves, it is much more complicated. This led to invention of Secondary Surveillance Radar (SSR).

SSR is basically PSR with additional device onboard called transponder that sends additional aircraft information to SSR after ground radar's pulse detection. That means the transponder has to have all electronics for signal modulation and demodulation. There are more ways how could transponder react to a signal detection from the ground's radar. This process is called interrogation and includes Mode A and Mode C interrogation types. Mode A sends 4-digit octal aircraft identification code—known as a Squawk; aimed to recognize aircrafts on the radar screen. Mode C is responsible for transmitting pressure-altitude of the aircraft. Later, Mode S transponder was developed with the 24-bit identification address (but still capable of Mode A and C interrogation), now known as International Civil Aviation Organization (ICAO) address that is unique for every transponder in the world. Furthermore Mode S has brought additional features that make use of the ability to communicate with SSR and other aircrafts, like Traffic Collision Avoidance System (TCAS) to provide more safety for increasing volume of air traffic. These valuable information are taken from the article [33].

Finally, ADS-B has been developed, greatly improving air traffic controlling, surveillance coverage, and safety. It uses modified Mode S transponder that no longer transmits data after ground radar's pulse detection, but does it automatically—'Automatic'. The necessary navigational information, as position and velocity aircraft gathers from Global Positioning System (GPS) and provides to ground stations and other aircrafts—'Surveillance'. From their point of view these data are dependent on functioning of the aircraft's systems and no longer on ground station's equipment—that makes 'Dependent' in the system name. Transponder broadcasts messages twice a second—'Broadcast'— at frequency 1090 MHz. Messages are neither encrypted nor authenticated, thus the content could be read by anybody with a proper receiver. Without this possibility my thesis could not be done. However, ATC ground stations never rely only on a one source of the data. There is always SSR, or PSR, to authentificate position of aircrafts in case of misuse.

Advantages of this system are for instance much less operational cost compared to complicated radar systems and, as I have noticed, improved air traffic controlling. ADS-B provides more accurate aircraft positioning, so the spacings between aircrafts could be smaller resulting in a better traffic flow, shorter approaches, safer parallel landings, and other air traffic enhancements. In last years ADS-B has been implemented by aviation organizations all over the world, as papers from [2] or [3] indicates.

### Receiver choice

Radio receiver that can tune to frequency 1090 MHz with proper Mode S demodulation hardware—Mode S downlink uses Pulse Position Modulation (PPM) [9]—is required to succesfully receive and decode ADS-B messages. There are many options on the market how to obtain such a device.

*Flightradar24* and *FlightAware*, largest applications for live air tracking, send receivers built by them for free to people that can improve their air traffic coverage [11].

That is unfortunately not my case, because Prague is already greatly covered.

A second option is to buy a receiver from a company that specialises in flight tracking. Favourite ones are for example *Kinetic Avionics SBS-3*, or *AirNav RadarBox*. These devices come with a software that already acts as a real time virtual radar screen, which I certainly do not require. The biggest disadvantage of these products is their price—they often starts from 600 €.

A third option is to make use of a device called Software Defined Radio (SDR), popular wide band radio scanner nowadays. The difference from classic radio is that the signal processing has moved from a hardware layer to a software layer. That means the signal modulation and demodulation is not limited by electronics in the receiver, but is implemented in the software [26]. There are many SDR on the market, however, exceptional idea was turning a USB DVB-T receiver with *Realtek RTL2832U* chipset into a SDR. This has been done by finding out, that raw data captured by receiver could be directly accessed and demodulated with modified driver. This dongle—receiver with modified driver—is called RTL-SDR and is the cheapiest, but very sufficient option currently on the market for wide band radio receiving, including ADS-B listening. Thanks to its convenient price around 20 € and a comparable performance with advanced receivers, it has been a choice for my project. Further information and applications could be find on the website of RTL-SDR [26].

My RTL-SDR contains tuner *Rafael Micro R820T* with frequency range 24 MHz–1766 MHz. For the communication with a computer it uses USB and requires also special driver. Reported power consumption is circa 300 mA. In the Figure 1.2 receiver's design is shown.

Figure 1.2: RTL-SDR receiver.[1]

### ADS-B messages receiving and decoding software

Mode S decoder compatible with RTL-SDR is required to acquire ADS-B messages. The choice in this case is simplier, because there is a software specifically designed for

[1] Picture taken from http://www.dozimetry.eu/product.php?id_product=67.

this task—dump1090 [4]. It is minimalistic, robust, and fast decoder with very decent performance. Besides many capabilities of dump1090, like dynamically displaying captured aircraft's data to console or plotting aircrafts on the webserver's map, the function I am using is listening to TCP port, where software serves all decoded data. One possibility is to receive raw, undecoded packets and write program that will take care of all translations to meaningful data. Second one is to receive already decoded data in a special format, which I will discuss later. Now I will try to manually decode ADS-B message for better understanding of the format with instructions from the article published by [29].

### Raw message format

Generally, Mode S message could be either 56 or 112 bits long, but ADS-B has always 112 bits. Figure 1.3 shows the composition of an ADS-B message.

- Downlink Format (DF) [bits 0–4] tells us the type of the message. I am looking for ADS-B message type (DF=17), but it could be for example Mode S transponder reply, or air-to-air message within TCAS.
- Message Subtype (CA) [bits 5–7] has no role in ADS-B messages.
- ICAO [bits 8–31] is the unique transponder idenitification number.
- DATA [bits 32–87] frame contains necessary data.
- PARITY [bits 88–111] bits serve to check if the message passed through a communication channel with no errors.


Figure 1.3: ADS-B message content.

One of messages I have captured with RTL-SDR is

```
0x8D4064422015A672CCA320BB799F.
```

The message is decomposed in the following Table 1.1. With DF=17 the ADS-B type

Table 1.1: Decoded ADS-B message.

| DF | CA | ICAO | TC | DATA | PARITY |
|---|---|---|---|---|---|
| 10001 | 101 | 0x406442 | 00100 | 0x15A672CCA320 | 0xBB799F |
| 17 | 5 | - | 4 | - | - |

of the message is confirmed. A Type Code (TC) indicates what information is in the DATA frame. With TC=4 an aircraft identification—callsign is contained in the packet. Let us have a look, how callsign is coded:

```
HEX: 0x15A672CCA320
BIN: 000101 011010 011001 110010 110011 001010 001100 100000
DEC:   5      26     25     50     51     10     12     32
       E      Z      Y      2      3      J      L      -
```

To translate a decimal value to a character, a special look-up table has to be used:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
A B C D E F G H I J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
32 48 49 50 51 52 53 54 55 56 57
_  0  1  2  3  4  5  6  7  8  9
```

The decoded callsign is **EZY23JL**. In a commercial aviation, callsign is very often same as the flight number, which is composed from ICAO code of the airline **EZY** and the flight identificator **23JL**. It can give us a clue what an airline or a route it is, but either can not. For that purpose I will have to find a way how to obtain additional information in my software, which are not part of ADS-B surveillance. By the way, **EZY** is ICAO code for the easyJet airline.

### BaseStation message format

*BaseStation* is a software developed by Kinetic Avionics, manufacturer of popular Mode S receivers including one I have mentioned before. Kinetic receiver streams data in the BaseStation format and it quite became as standard for various applications because of its popularity, and dump1090 is no exclusion. BaseStation stream from dump1090 could look like this:

```
MSG,1,111,11111,4CA7B6,111111,2016/04/25,21:21:54.474,
2016/04/25,21:21:54.469,RYR34PW ,,,,,,,,,,0
-----------------------------------------------------
MSG,3,111,11111,4BD149,111111,2016/04/25,21:23:08.459,
2016/04/25,21:23:08.458,,40000,,,49.31177,14.31828,,,,,0
-----------------------------------------------------
MSG,4,111,11111,4BD149,111111,2016/04/25,21:23:08.464,
2016/04/25,21:23:08.459,,,435,309,,,0,,,,,0
-----------------------------------------------------
MSG,5,111,11111,3C4DCD,111111,2016/04/25,21:23:08.473,
2016/04/25,21:23:08.461,,6025,,,,,,0,,0,0
-----------------------------------------------------
MSG,6,111,11111,3C4DCD,111111,2016/04/25,21:23:08.627,
2016/04/25,21:23:08.594,,,,,,,,1000,0,0,0,0
-----------------------------------------------------
MSG,7,111,11111,3C4DCD,111111,2016/04/25,21:23:08.687,
2016/04/25,21:23:08.660,,6000,,,,,,,,,0
-----------------------------------------------------
MSG,8,111,11111,4BD149,111111,2016/04/25,21:23:08.859,
2016/04/25,21:23:08.852,,,,,,,,,,,0
```

Every message is one of eight types, depends on what information message brings. Description of each message type is in the table 1.2. Transponder's ICAO address is of course part of each message for aircraft identification and the date and the time when message was generated and received. Note that not only ADS-B messages are received, but also replies to SSR. There are also some '111' fields in messages where should be additional information that BaseStation software is working with, but they are not related with aircrafts. Following aircraft attributes could be obtained from BaseStation messages:

- Mode S hexadecimal identification number—ICAO address,
- callsign (CS),
- Mode C pressure altitude (ALT) - height is calculated relative to the standardized air pressure at the Mean Sea Level (MSL), which is 1013.25 hPa. Resolution 25 ft,

Table 1.2: Description of individual messages in the BaseStation format.

| MSG Type | DF | Surveillance | Description | Content |
|---|---|---|---|---|
| 1 | 17 | ADS-B | Identification | CS, GND |
| 2 | 17 | ADS-B | Surface Position | GS, TRK, LAT, LON, GND |
| 3 | 17 | ADS-B | Airborne Position | ALT, LAT, LON, GND |
| 4 | 17 | ADS-B | Airborne Velocity | GS, TRK, VR, GND |
| 5 | 4 | Mode C | Pressure Altitude | ALT, ALRT, IDENT, GND |
| 6 | 5 | Mode A | Squawk Code | SQK, FLAGS |
| 7 | 16 | Mode S | TCAS | ALT, GND |
| 8 | 11 | Mode S | All Call Reply | GND |

- speed over the ground (GS),
- track of the aircraft (TRK)—not heading; derived from a latitude and a longitude change in a time,
- latitude gathered from onboard GPS (LAT),
- longitude gathered from onboard GPS (LON),
- vertical rate (VR)—resolution 64 ft/s,
- Mode A assigned Squawk code (SQK)—triggered only by SSR interrogation. If aircraft is out of a range from any ground radar's coverage SQK can not be received,
- FLAGS indicating squawk has been changed (ALRT), emergency state has been deployed (EMER), transponder IDENT has been activated (serves for ATC to match aircraft on the SSR radar screen), and ground switch status (GND)—indicating whether aircraft is on the ground or not.

I have two options how to decode received data by Mode S receiver. My choice is definitely BaseStation format, because of much less decoding. Instructions for understanding this format were taken from [7], supported by [35].

## 1.2 Position and heading determination

In this section I will try to describe why and how I can determine system's position and heading required for a successful aircraft targeting.

### Position

The system's position is inevitable to know because of simple reason—angles that camera is required to move are calculated from observer's and target's coordinations (latitude and longitude). Most available technique for getting our location on the Earth, which we use every day, is GPS. I will describe how GPS works in the next subsection, but this information is not required for a correct position receiving and is here only because of my interest in this system.

### GPS principle

GPS is the utility that provides navigational, positional, and timing services for users fully operated by the United States Air Force. It is the fundamental of my application, because both aircrafts use GPS for very precise positioning and the system.

GPS is the constellation of 24 satellites (also called by GPS terminology Space Vehicle (SV)) with four on an one of a six equally-spaced orbital planes transmitting radio signals to civil and military users 24 hours a day. Their arrangement ensures that every user on the planet will always have signal from at least for SVs, which is the minimum for a position determination. Satellite's altitude is approximately 20 200 km with orbital period 12 hrs. In case of a failure there are additional seven back-up SVs on the Earth's orbit. With high-quality GPS receiver the user could be located with a precision better than 3.5 m [13].

GPS SV use two channels for a navigational communication with devices on the ground—L1 modulated with a Pseduo-random Noise (PRN) Course/Acquisition (C/A) and Precise (P) code carried by a wave with the frequency 1575.42 MHz and L2 modulated with just the P code on the frequency 1227.60 MHz. Civilian devices use only L1 channel—modulating binary codes with Binary Phase Shift Keying (BPSK) technique—with the C/A coding while military users use both channels with the P coding that could be even encrypted to Y code. P code is more complex and have some enhancements, hence the navigation for an army is a little bit better and more immune to jamming. However civilian GPS is always being modernized and eventually the precision will be equal. That is happening because of another two signal transmitted that has been currently implemented in new SVs—L2C, which uses L2 channel for civil purposes and L5, which is at the frequency 1176.45 MHz and serves as a civilian Safety-of-Life signal.

The C/A code is unique for all SVs and is as much uncorellated as possible with other PRN codes. These codes must be known to every GPS receiver—it must correctly distinguish different satellites. Firstly, 1023 bit C/A code is transmitted by every SV every milisecond. Secondly, Navigational Messages (NM), low frequency signals modulated to both L1 and L2 carriers carry all inevitable information for receiver to determine position of SVs (orbits data are called Ephemeris), time error corrections (for example caused by ionosphere) and, very important, a precise GPS time of message transmission. If receiver could have synchronized clock with those on SVs, the exact location on the Earth could be known by communication with only three transmitters using trilateration method. Distances from individual SVs would be calculated by simple formula $d = c\Delta t$, where $c$ is the speed of light and $\Delta t$ is the time difference between the message transmission and the reception. Yet boards inside SVs are driven by an extremely precise atomic clocks and it would be nonsense if every GPS receiver had expensive oscillators too, so there is always time shift and the distance calculation have no sense then. The solution is communicating with one more SV that corrects the time measurement inside receiver, as course from [10] says. Mathematically receiver solves following four equations with four unknows to get the absolute position of ground device [32]:

$$
\begin{aligned}
d_1 &= c(t_{r1} - t_{t1} - t_c) = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} \\
d_2 &= c(t_{r2} - t_{t2} - t_c) = \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2} \\
d_3 &= c(t_{r3} - t_{t3} - t_c) = \sqrt{(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2} \\
d_4 &= c(t_{r4} - t_{t4} - t_c) = \sqrt{(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2},
\end{aligned}
$$

where $c$ is the speed of light,
$d_i$ are true distances from SVs,

$t_{ti}$ are GPS times when messages were transmitted (they are part of NM),
$t_{ri}$ are receiver's inaccurate times when signals were received,
$(x_i, y_i, z_i)$ are SV's coordinates in the time of transmission (part of NM),
$i = 1, 2, 3, 4$ represents each (SV),
$(x, y, z)$ are receiver's uknown coordinates, and
$t_c$ is the receiver's clock time delay.

Furthermore GPS could be considered as the exact time provider everywhere on the globe. Still the GPS time is quite different from ours, Coordinated Universal Time (UTC). GPS time was zero on 06.01.1980 and since then it is counting weeks and seconds. By now there is 17 seconds diffrence from UTC time because of corrections of rotational movement of the Earth, as can be seen in the Table 1.3 [19].

Table 1.3: Comparison of current GPS and UTC time.

| | | | | |
|---|---|---|---|---|
| UTC | 2016-05-01 09:39:36 | Sunday | day 122 | timezone UTC+2 |
| GPS | 2016-05-01 09:39:53 | week 1895 | 34793 s | cycle 1 week 0871 day 0 |

### GPS receiver

The receiver I am looking for should fulfil requirement for being low-power, small, and precise. There are more modules commonly used by DIY community. The first category are modules designed to be placed especially on some brand of microcomputers offering also few enhancements. The second category are standalone GPS modules suitable for any kind of computer—that seems as a better idea for my project as I do not exactly know what computer model I will acquire. Interesting receivers are selled by popular DIY manufacturer *Adafruit* with great quality but often for twice as much money as Chinese manufacturer's modules with comparable quality. The one I have found contains chip *u-blox NEO-6M*, versatile high performance position engine with a low power consumption placed on a miniature PCB with a passive ceramic antenna and an Universal Asynchronous Receiver/Transmitter (UART) communication interface. Package, shown in the Figure 1.4, is compatible with both 3.3/5 V level suitable for most microcomputer systems and costs just 15 € [1].

Figure 1.4: GPS module.[2]

Individual messages from GPS receiver follow the National Marine Electronics Association (NMEA) 0183 standard [8]. The message stream from GPS receiver could

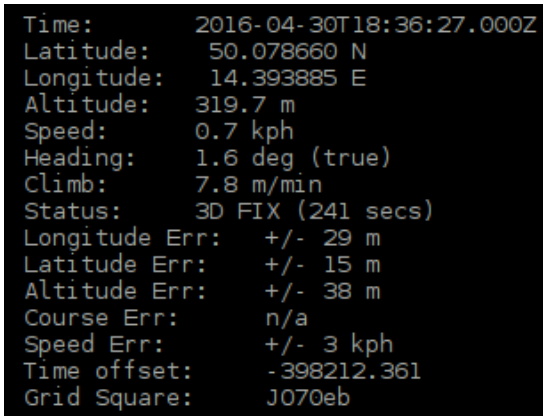
[2]Picture taken from [1].

look like this:

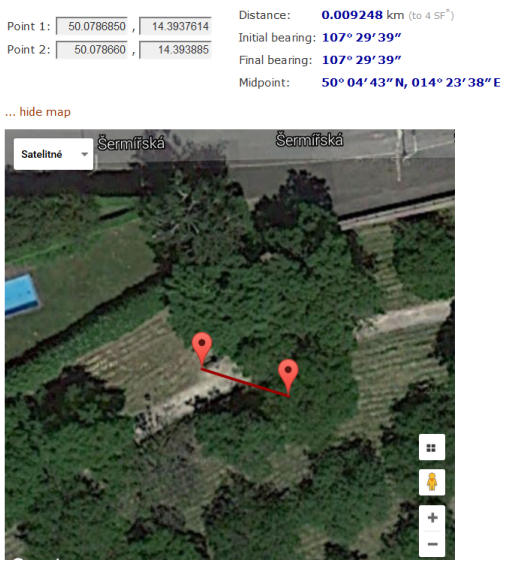
```
$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,61.7,M,55.3,M,,*75
```

For monitoring GPS receivers there is perfect application—GPS Daemon (GPSD) [12]. It translates navigational data in NMEA format from the sensor to more understandable form and outputs them through TCP socket.

I have made some measurements to test the GPS performance. Although manufacturer promises precision up to 3 meters—I guess with high quality antenna that I surely do not have—I expect worse results. In the Figure 1.5 a) is a screen shot of the GPSD test application results. The most important for me is the latitude/longitude and altitude information. The error calculated by GPSD seems very pesimistic, because with such a mistake GPS seems useless. But if I compare my real position when I took the measurement with the result position, as shown in the picture 1.5 b) the distance between these two points is only 9.3 m. Such an offset should not disrupt aircraft tracking and if it do so using more quality active antenna will fix all imperfections. Another interesting value is the altitude. Again the error ±38 m is little bit high, but in fact when I looked to altitude maps I have found the altitude difference is only 1 m.




(a) Captured navigational data




(b) Position precision comparison

Figure 1.5: GPS test results.

From the altitude and position difference could the climb, speed, and heading be calculated. Even when my receiver was at stable position latitude and longitude values were always changing in last two decimal place (fifth decimal place accuracy is 1.1 m and sixth 0.11 m). It lead to misreadings that receiver is moving and climbing.

One more thing the GPS can bring to my system is precise time measurement. A possible application is to put a time trace into videos.

### Heading

Having known the latitude, longitude, and altitude of the observer and the aircraft, a pan and a tilt angle to point the camera towards aircraft could be calculated. The pan angle is the angle between camera's current and initial heading (similar to aircraft's yaw). The tilt angle is the angle indicating how camera's nose is lifted up or down (similar to aircraft's pitch) [21].

If the camera is pointing exactly to the North, necessary pan angle will be the bearing—the angle between the object and the direction toward the North relative to the observer—of the aircraft. For calculating pitch angle the altitude difference and the distance between these two points is required. Nevertheless, camera's initial heading—the angle where camera is pointing relative to the North—would only hardly be the exact geographical north pole. That is why the initial heading of the system must be known and the final pan angle appear to be the difference between desired heading and system's heading. In other words relative bearing of the aircraft—the angle between object's bearing and my heading—is how camera should pan. These angles were nicely explained in [30] on the StackExchange and I illustrated them in the Figure 1.6:

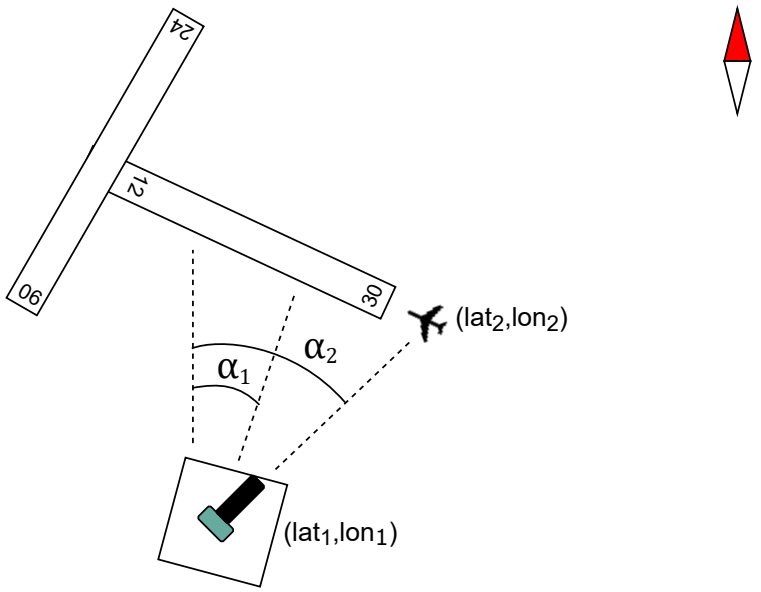



Figure 1.6: Illustration of a heading, bearing, and pan angle.

- aircraft's heading is 300 °,
- original system's heading is $\alpha_1$,
- the bearing of aircraft from the system's view is $\alpha_2$,
- relative bearing is $\alpha_2 - \alpha_1$, which is the same angle as is required from the camera to pan to capture aircraft.

A navigational device that shows direction relative to the magnetic North is a well-known compass. Its electronic variant is called magnetometer. Attention must be payed to distinct heading and magnetic heading.

Earth's rotational axis defines geographic North and South, thus the heading indicates angle relative to the this (true) North. Nextly, Earth's magnetic field could be approximated as the field of a simple bar magnet with the north pole near the true

south pole. However there is an angle discrepancy circa 11.5 ° between the approximated bar and the rotational axis. The angle is called magnetic declination, is different everywhere on the globe and must be applied as a correction when using magnetic sensor for navigation. The value of declination in Prague is circa 4 ° E as the paper generated by National Oceanic and Atmospheric Administration (NOAA) shows in the attachment Figure 3.3.

### Sensor

When I was buying an orientation sensor I was not sure what type of motor control I will use, thus I have obtained 9-axis absolute orientation sensor to have a complete feedback from the camera orientation. Such a sensor contains magnetometer, gyroscope, and acceleromter, so is capable of sensing Earth's magnetic field. The chip is *BNO055* placed on a tiny PCB by Adafruit providing I2C and UART communication interfaces with Python libraries (picture of the sensor is shown in the Figure 1.7. But just for the magnetic field measurements better alternative is a plain magnetometer—digital compass, for example *Honeywell's* popular for DIY projects *HMC5883L*.

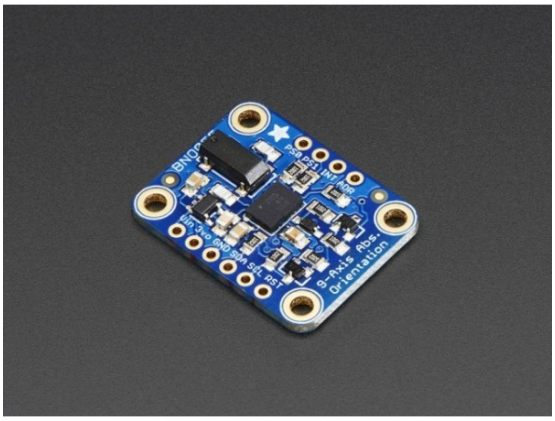


Figure 1.7: BNO055 9-DOF Absolute Orientation IMU Fusion Breakout.[3]

The magnetometer should be level to the Earth's surface for correct heading measurement, though tilt compensation methods exist. It measures the Earth's magnetic field in a three-dimensional vector, as shown in the following Figure 1.8.

Axes $x$ and $y$, planar with Earth's surface indicates heading and axis $z$ is for determining of a direction and a strength of the magnetic field. The magnitude of magnetic flux density $B$ of Earth's magnetic field at any point is simply calculated as

$$|B| = \sqrt{B_x^2 + B_y^2 + B_z^2} \tag{1.1}$$

and gains values from 0.3 to 0.6 G, depending on a position on the Earth (10 000 Gauss (G) = 1 Tesla (T)).

The magnetic heading—angle between the magnetic north (vector $B_{xy}$) and the axis $x$ (direction of the magnetometer), $\theta$ in the Figure 1.8 is calculated depending on

[3]Picture taken from http://www.snailshop.cz/kombinovane-imu/1702-adafruit-9-dof-absolute-orientation-imu-fusion-breakout-bno055.html.

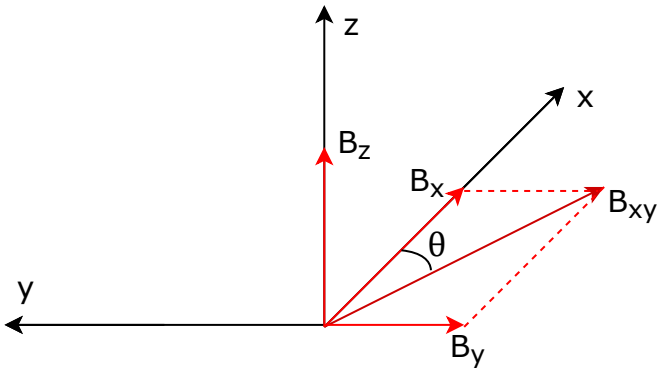



Figure 1.8: Magnetometer's coordinate system.

a sign of $B_x$, respectively $B_y$, and gains value in the interval $\langle 0, 360)$ degrees:

$$\theta = \begin{cases} \tan^{-1}\left(\frac{B_y}{B_x}\right)\frac{180}{\pi}, & B_y \geq 0, B_x \geq 0, \\ 180 + \tan^{-1}\left(\frac{B_y}{B_x}\right)\frac{180}{\pi}, & B_x < 0, \\ 360 + \tan^{-1}\left(\frac{B_y}{B_x}\right)\frac{180}{\pi}, & B_y < 0, B_x \geq 0. \end{cases} \tag{1.2}$$

These knowledge I have taken from the [17].

## 1.3 Pan/Tilt platform

A pan/tilt platform is required to move the camera in two axes discussed in the Section 1.2. Versions on the market suitable for DIY projects are unfortunately only few. Many of them are aimed for very small and light cameras that would definitely not actuate the camera with a lens. Or there are systems with electronic stabilization to hold camera's head stable when the carrier is moving or shaking (for instance drone) for great quality of the video. This is however not my case, because I expect the platform is standing still and such a system would be completely overkill—with both possibilities and costs around 500 €. With an access to the 3D printer I have decided to create such a platform for the project on my own.

The first choice I have to make is which motors I would like to use. Possible options are brushless, stepper, brushed DC, and brushed DC servomotors:

- Brushless and brushed motors are perfect options for continual rotation applications and with a feedback sensor they could be also used for angular positioning. They can not be directly controlled from most microcomputers, therefore control unit is required,
- stepper motors are great for precise angular placement control but also dedicated control unit procuding signals is inevitable,
- brushed DC servomotors allow angular position control without external control unit because necessary electronics is included in themselves and control signals could be produced by any microcomputer. Their disadvantage is lack of ability to perform full rotations.

My choice is brushed DC servomotor because of much simpler control and less cost—motors are driven by Pulse Width Modulation (PWM) signals that directly set angular

position of the rotor [20]. The ability to fully rotate could be omitted, because in most cases I set the camera in a way it needs only a 180 ° view—perpendicular to runway's axis to see the most of the aircraft during landing/take off (similar as in the Figure 1.6). During picking the servo I must bear in mind there might be a lens on the camera that might cause pretty high torque. The model I have chosen is *Hitec HS-5485HB*, with maximum torque 6.4 kg/cm at 6 V, ball bearing, and karbonite gears, shown in the Figure 1.9.

Figure 1.9: Powerful Hitec servo that will actuate the camera.[4]

Next task is to design a 3D model of the pan/tilt specificaly for selected servos. First step is to design the pan/tilt base with mounting holes for later assembly. It is also a holder for the first servo that pans with the rest of the system (Figure 1.10 (a)).

Second step is to design the second servo holder. It is same as the previous one except it lays on the side so the servo's arm tilts the camera (Figure 1.10 (b)). In the bottom side in the center is the shape that perfectly matches pan servo's arm (Figure 1.10 (c)) and from inner side is a hole for screw's head. There is also noticeable hole in the side wall of the holder—its center is in the rotational axis of the tilt servo and ball bearing will be placed there allowing second arm of the camera holder smooth rotation.

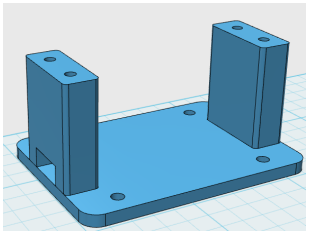
Third step is a creation of the camera holder itself with arms attached to rest of the platform. My first idea was to print this part in one piece but I had troubles with attaching it, so I have decomposed the part into three pieces. One arm of the holder is attached on the servo's arm (Figure 1.10 (d)). Second arm is plugged to the ball bearing (Figure 1.10 (e)) and camera is mounted to the part in the Figure 1.10 (f) that is attached with several screws to its arms. Only this part has to be modified to fit the camera I desire.

The model have been printed with *PLA* and attributes to strengthen the construction because of large forces that may possible lens produce—50 % fill density and shell thickness 0.8 mm. Result is high quality, robust, and strong pan/tilt platform. Major disadvantage is particular dimension and camera's distance from rotational axes. The final construction is shown in the Figure 1.11.

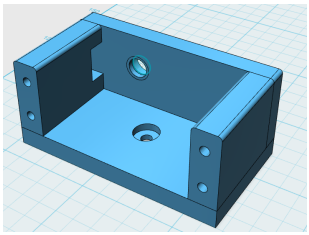
## 1.4 Computer

When it actually comes to choice of a computer, control unit for external components, there is not so much to consider. Most popular low-cost, credit-card sized

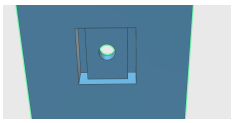
[4] Picture taken from http://www.peckamodel.cz/produkt/rc-modely-a-prislusenstvi/serva-a-prislusenstvi/serva-hitec/1hi3191-servo-hs-54-85-hb.



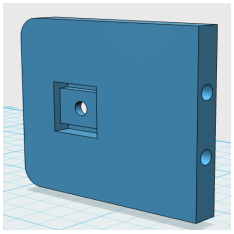
(a) Pan servo holder.



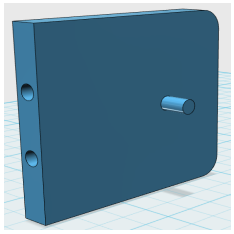
(b) Tilt servo holder.



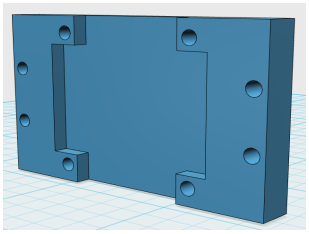
(c) Detail of the shape where is attached servo's arm.



(d) The arm of a camera holder attached to the servo.



(e) The arm of a camera holder attached to the ball bearing.



(f) Final camera holder where is the camera directly attached.

Figure 1.10: Overview of individual components that is pan/tilt made of.

microcomputers for interacting with hardware are *Arduino*, *Raspberry Pi*, *BeagleBone*, and *Intel Edison*. Arduino is mainly aimed for less complex projects, Intel Edison requires Arduino header for interacting with hardware and both does not run operating system. Both BeagleBone and Raspberry Pi runs *Linux*, which I require for correct RTL-SDR functioning. BeagleBone has advantage in having more General Purpose Input/Output (GPIO) ports, analog inputs, support for any kind of hardware, while Raspberry Pi on the other hand has faster quad-core CPU, GPU, larger RAM and is more suitable for software-based projects. Additionaly Raspberry has own interfaces for camera and display, which would be a pity not to make use of. Community support and lots of tutorials are also on the Raspberry's side [14].

The model I am using is *Raspberry Pi 2 Model B*, the newest at the time I was buying one (shown in the Figure 1.12), but now version 3 has been introduced. Probably I would rather buy newer one because of even better performances. Also the memory card has to be bought as a memory storage. I have decided for the 32 GB class 10 micro SDHC card to safely store large videos.

One more important thing is the Raspberry's power consumption. Reported maximum current draw under the stress could be up to 600 mA, but more likely the draw is under 400 mA (Pi's voltage is 5 V) without any peripherals [25].

## 1.5 Input/Output interface

Common ways how to interact with Pi are through Secure Shell (SSH) protocol from another computer, or by mouse and keyboard with monitor (connected through HDMI) if graphical interface is desired. In my case the screen is desired because of displaying aircrafts on the map and view of the camera. Hence SSH is unusable and bringing monitor with the system is also not quite a good solution. However Raspberry released a 7" touchscreen with resolution $840 \times 480$ pixels to create all-in-one, integrated projects—that seems as a best solution for me.

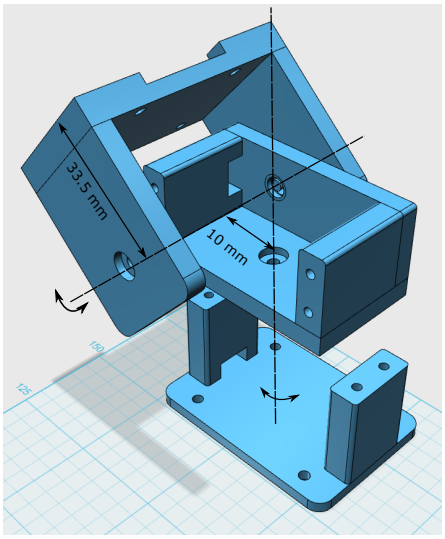



Figure 1.11: Pan/tilt platform designed in the Autodesk 123D application with designated rotational axes.

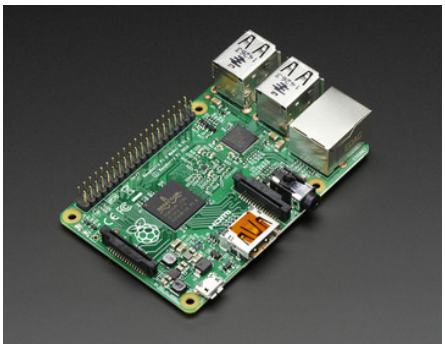


Figure 1.12: Small, inexpensive Linux-based computer Raspberry Pi 2 Model B.[5]

The touchscreen is connected to the adapter board supplied with the display through Display Parallel Interface (DPI). The board converts the signal to compatible one with the Display Serial Interface (DSI) on the computer. Power supply rated for at least 500 mA at 5 V is required for powering the display [16]. Both boards could be mounted to the back of the display using prepared mounting holes, but a display holder is not supplied and must be created or bought. With an advantage of having 3D printer I have again printed the part tailored for my project. The model and display itself are shown in the Figure 1.13. To make touchscreen running on the Raspberry only latest operating system is required.

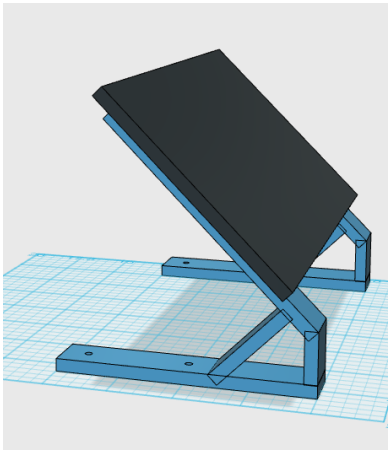
## 1.6 Camera

An image sensor—camera is certainly required to capture flying aircrafts to video. One of the reasons I have chosen Raspberry Pi is the fact that Raspberry Pi Foundation released a camera module especially for theirs computers with decent attributes, low cost, and excellent compatibility. It is connected to the board through Camera Serial Interface (CSI). If someone does not want to use this camera possible option is to capture images with USB web camera.

The camera is equipped with 5-megapixel *Omnivision OV5647* sensor capable of

[5]Picture taken from `https://www.adafruit.com/product/2358`.

(a) The display and its adapter board.



(b) Display holder designed in the Autodesk 123D application.

Figure 1.13: Overview of the Raspberry Pi Official Touchscreen and its designed holder.[6]

1080p30 or 720p60 video modes. However, as I expect aircrafs to be further, the lens is inevitable accessory. The disadvantage of the official module is lack of the possibitlity to replace the lens, but I have found manufacturers in China that produce fully compatible cameras with a lens mount. The one I have ordered is produced by *ArduCam* [5] and comes with the CS mount plus the lens (shown in the Figure 1.14 (a)). Nevertheless, the lens are just with focal length 6 mm and I want better zooming. Reasonable choice, again from China is $5-100$ mm focal length lens aimed for a CCTV application (shown in the Figure 1.14 (b)). Though I must consider servomotor choice because of a torque created by such a large and heavy metal body. The camera board is however very tiny so I have decided to print support for camera mount to release the pressure on the module (shown in the Figure 1.14 (c)). For embedding the camera into applications there are lot of available libraries written in many languages. The power consumption is around 250 mA according to [25].

## 1.7 Power supply

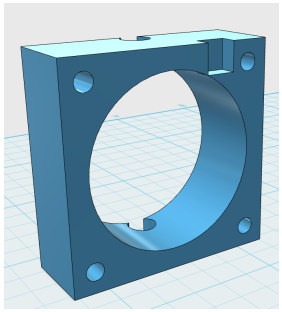
The power source rated at 5 V that could reliably provide 1.5 A is required for powering the computer and its components. My first idea was to acquire a power bank for chraging smartphones—it should provide up to 2.1 A at 5 V and with large capacity it seemed like long-lasting easy-rechargable compatible power source (Pi is powered though the microUSB port). But that also means servos should be powered individualy because of high current spikes they may drain. However when I put the Pi under the load I quickly have found that the power bank is very unreliable power source—the voltage drop at the Pi was almost 1 V. I think it was caused by both resistance of the cable and not great quality of the bank.

Accumulators I have bought are model *eneloop* (AA NiMH, capacity 2000 mAh, voltage 1.2 V), favoured by modeling community, in package of five pieces as shown in the Figure 1.15 (b) (one package for one servo). After I had found that the bank can not be used for powering I tried to make use of these two packages. The DC-DC step-down converter (shown in the Figure 1.15 (a)) is a suitable component to drop the voltage to

[6] Picture of the display taken from https://www.element14.com/community/docs/DOC-78156/l/raspberry-pi-7-touchscreen-display.

(a) ArduCam version of a Raspberry Pi Camera.

(b) Lens with a CS mount and a focal length $5 - 100\,\text{mm}$.



(c) 3D printed support for CS mount on the camera board.

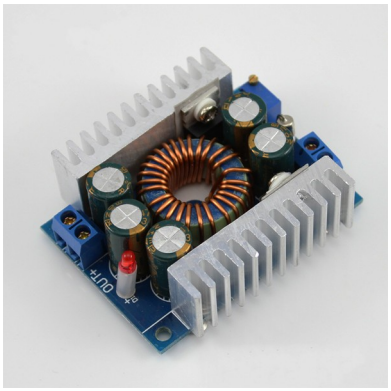
Figure 1.14: Overview of the camera with lens used in my project.[7]

Pi's 5 V. There are two possible connections of battery packs—regulate voltage from first for the Pi and put 6 V from the second for servos, or put both packs to series and convert 12 V to the 5 V. I have decided for the second option because converter conserves power and the current drain at the output is $x$-times smaller ($x = U_{in}/U_{out}$, but there is of course some error) than at the input so accumulators will last more time allowing system work for several hours. The disadvantage is that servomotors will produce slightly smaller torque. The voltage is stable at 5 V even during current spikes because of large capacitors on the board.

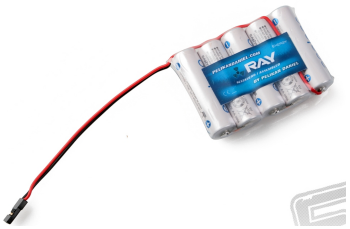
## 1.8 Final assembly

The whole system is assembled to plastic board. To allow stable position during observation I have drilled a hole in the board and put there threaded hole that matches screw on the tripod. Also there is a breadbord on the board for better access to Pi's pins, connection of motor's and Pi's ground, and placement of a switch that turns the power on. Unfortunately the BNO055 sensor uses I2C clock stretching which is buggy in the Raspberry [6], therefore an USB-UART converter has to be used to allow sensor communication with the Pi through the UART interface (Raspberry has only one occupied already by GPS sensor). The complete scheme of the system is illustrated

[7] Picture of the camera taken from http://www.ebay.com/itm/OV5647-Camera-Board-w-CS-mount-Lens-for-Raspberry-Pi-A-B-B-2-Model-B-/281212355128?hash=item4179900238:g:s~0AAOSwqu9U7mu1, picture of the lens taken from http://www.aliexpress.com/item/New-Manual-IRIS-CS-Mount-cctv-lens-F1-8-100mm/2043308150.html.



(a) DC-DC Buck Converter 100 W 12 A 4.5 – 30 V to 0.8 – 30 V Step-down.



(b) The pack of five eneloop batteries providing 6 V.

Figure 1.15: The overview of used accumulators with step-down DC-DC converter.[8]

in the Figure 1.16 and final design of the system is shown in the Figure 1.17. The complete list of used components:

- **plastic board** components are assembled on the board
- 4× **rubber feet** comfortable board placement on the table
- **Raspberry Pi 2 Model B** control unit
- **Raspberry Pi Official Touchscreen** interaction with the user
- **3D printed display holder** holder for the touchscreen
- **breadboard ZY-60** electrical connection of several parts
- **Raspberry Pi Plus Breakout** Pi's GPIO pins on the breadboard
- **Raspberry Pi Ribbon Cable** connection of the breakout and the Pi
- **RTL-SDR with an antenna** reception of ADS-B packets
- **GPS receiver** position of the system determination
- **Adafruit BNO055** magnetic heading determination
- **Arduino USB2Serial Adapter** convert serial signals from BNO055 to USB
- **3D printed pan/tilt** holder for servos and camera
- **ball bearing** smooth rotation of the camera holder's second arm
- 2× **Hitec HS-5485HB servomotor** moving camera in two axes
- **ArduCam Raspberry Pi Camera Module** capturing images
- **CCTV lens** 5 – 100 mm improve image quality

[8] Picture of batteries taken from http://www.pelikandaniel.com/?sec=product&id=38077, picture of the converter taken from http://www.ebay.com/itm/DC-DC-Buck-Converter-100W-12A-4-5-30V-to-0-8-30V-Step-down-Module-Laptop-LED-Car-/261420678182?hash=item3cdde2fc26:g:BRMAAOxyf1dTHtuu.

- 10× **Panasonic eneloop AA accumulator** power source for the system
- **DC-DC Step-down converter** voltage regulation
- **switch** turn on/off the system

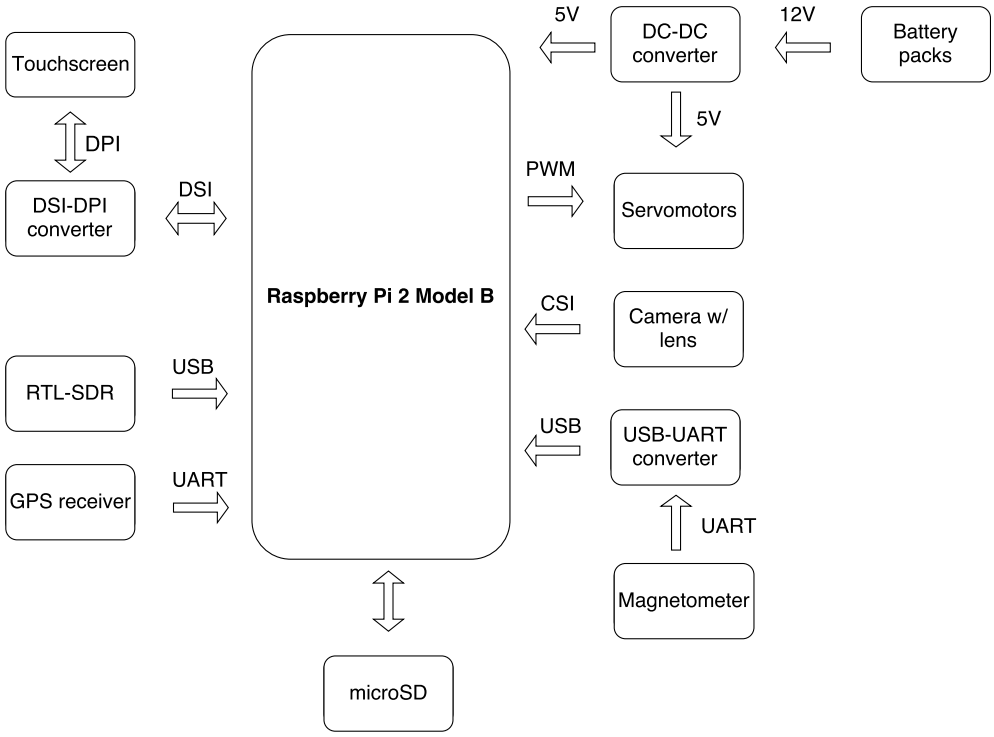
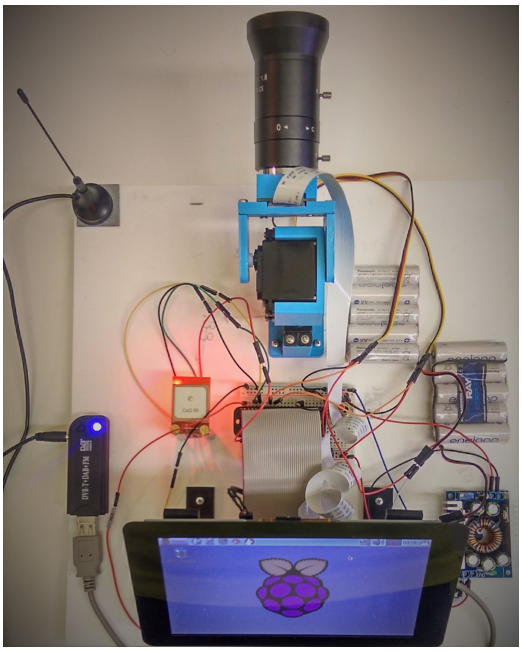



Figure 1.16: The final scheme of system's components and communication interfaces.

(a) System during observation.



(b) Detail on the board with assembled components.

Figure 1.17: Photos of the constructed system.

# 2. Software implementation

In this chapter I will discuss solution of the problem from the software's side. After constructing the system, the program that interacts with individual components and the user simultaneously has to be written in the suitable programming language. The program's task is to get all data from sensors and receivers, allow observer to choose aircraft on the map, and track the selected aircraft while the video is recorded. But firstly I must decide which operating system I would like to use with the Raspberry Pi.

There are few companies developing operating systems for ARM devices like the Raspberry Pi. The most interesting images are officialy supported *Raspbian*, more user-friendly *Ubuntu MATE*, or developer Internet-of-Things packages by *Microsoft* and *Ubuntu Snappy*. As long as the Raspbian is optimized for Raspberry Pi hardware and compatibility is my priority the choice was simple [24].

Same thoughts I can apply for selecting a programming language in which I would like to write the code. Requirements are mainly connectivity with peripherals, support for multithreading, socketing, and Graphical User Interface (GUI). All these specifications fulfil the *Python* programming language. It is a general-purpose, interpreted, interactive, object-oriented, and high-level programming language with an excellent documentation [31].

In the Figure 2.1 is illustrated a simplified scheme of software's components. *gps, BNO055, picamera* are Python packages for establishing communication with individual sensors. Dump1090 and GPSD have been already mentioned as applications (section 1.1 and 1.2) that process Mode S respectively GPS signals. The software will ensure start of these applications using Python *subprocess* module including later resources release. One more decision I have to make is how I will produce PWM signals to control servomechanisms (it could be done either with Python module or with Linux software).

## 2.1 Processing ADS-B messages

For low-level networking Python's module *socket* is available. The connection has to be established with the TCP port number 30003 on the local machine where dump1090 implicitly outputs ADS-B data in the BaseStation format. Processing these messages should be done in an individual thread to allow more tasks work simultaneously (this is done by *threading* Python module). To prevent the socket from overloading while it process messages two threads have to be implemented for correct ADS-B message processing:

First thread receives the data from socket. If there are more ADS-B messages stacked in one large message the thread will do necessary splitting. Also if a message is teared up during the socket transmission the thread will repair the message. Then it passes packets to the second thread. To safely exchange information between threads a FIFO *queue* is very helpful.

Second thread decodes the message and stores the information to the structure available to the whole program. This structure should be thread-safe so multiple thread access will not corrupt the data. Also during the data setting/getting must be clear which aircraft's information are manipulated with. Actually for this purpose there is the unique ICAO transponder address contained in every message that could act as a key in the structure and associated value could be aircraft's data. These data could be holded in a special class. Such a structure is called *dictionary* in Python [23].

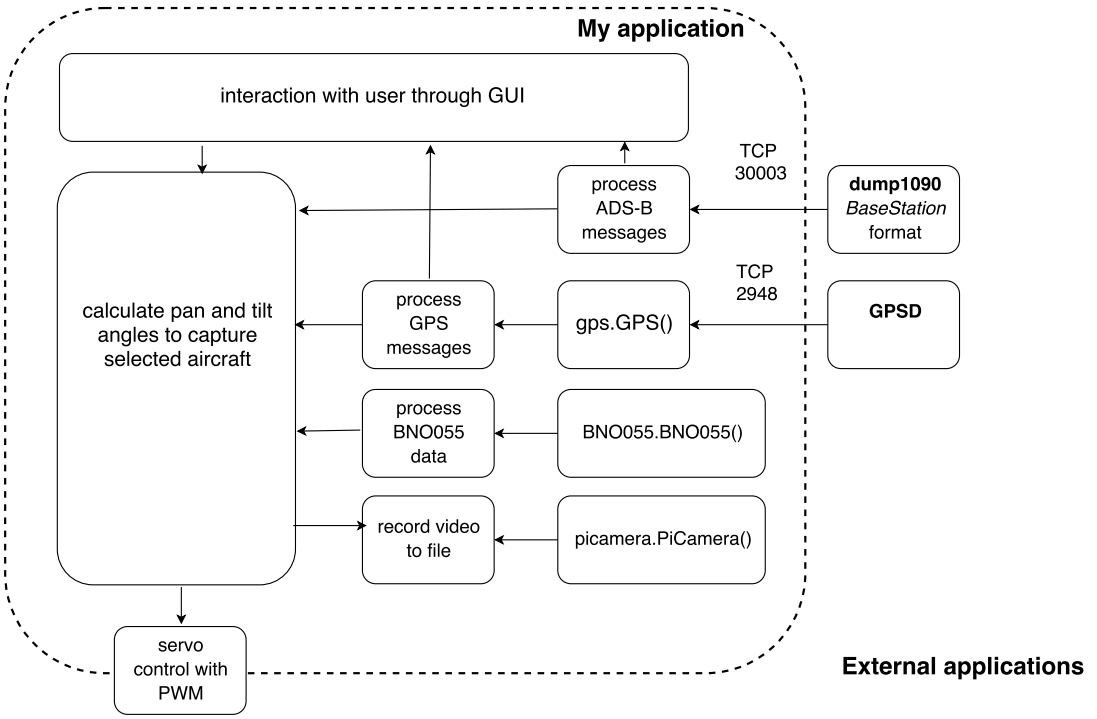



Figure 2.1: Simplified scheme of software subtasks and designated connections with sensors.

### More aicraft information

After implementing previous step I have all necessary positional information for tracking aircrafts. Still the most decisive attribute for spotters is probably the aircraft type or airline. I would like to have these valuable information in my application to draw my attention to interesting flight. Two *SQLite3* databases exist which could provide such information. One is matching callsign (flight number) with airplane's route (*StandingData.sqb*), second one knows which aircraft has assigned tracked transponder ICAO address (*BaseStation.sqb*).

BaseStation database—called after BaseStation software—is the part of the program contributing lot of additional information about the physical air frame. One nice example what I can achieve by getting only one Mode S message:

```
SELECT Registration, ModeSCountry, RegisteredOwners, OperatorFlagCode,
IcaoTypeCode, Type, SerialNo FROM Aircraft WHERE ModeS = '49D092';

OK-NEM|Czech Republic|CSA Czech Airlines|CSA|A319|Airbus A319-112|3406
```

Registration number, or tail number tells us in which country is the aircraft registered (letters before the dash). This immediately confirms next field—**OK** is representing Czech's airplanes. In the next field I can see what airline is the aircraft's operator with its ICAO code and finally the type of the aircraft with its ICAO code and even the manufacturing serial number.

StandingData database is the piece of the Virtual Radar Server software. Its task is to plot aircrafts on the Google Map caught by user's Mode S receiver. One form of the feeding stream for the application is the BaseStation format. To give user more interesting flight information it uses StandingData database to match flight number with the route data and this database is free to download from their website [34].

Example with the captured callsign of the aircraft wearing the ICAO address analyzed paragraph above:

```
SELECT
OperatorName, FromAirportLocation, FromAirportIata, ToAirportLocation,
ToAirportIata FROM RouteView WHERE Callsign = 'CSA2DZ';

Czech Airlines|Prague|PRG|Paris|CDG
```

Operator name was quite obvious in this case from the callsign, but I have obtained the destination and the beggining of the journey. In addition to knowing names of cities there are plenty of information about airports, like the
International Air Transport Association (IATA) code. It could be used to distinguish the airport which is the plane flying from/to in case of there are more in the city.

To query the data from SQLite3 databases Python module *sqlite3* is available.

## 2.2 Processing GPS messages

Indicated in the Figure 2.1, access to the GPS data is acquired through the *gps* module which receives data from the GPSD and translates them into final positional information. There is no necessarity to decode the data—only global availability within the program must be ensured to provide the position for each function. Also I have decided to let the GPS processing run in an individual thread in loop in case the observer's location has been changed during the runtime. When the observer starts capturing an aircraft the latitude, longitude, altitude, and time is taken from the thread.

## 2.3 Processing BNO055 data

To establish connection with the BNO055 sensor and get its reading Adafruit released a Python module. At every start of application the sensor has to be calibrated. After calibration simply magnetic field's vector is read and the heading is calculated from the formula 1.2. That is done with `atan2` function in the *math* package—advantage of `atan2` is that it returns the angle in a quadrant based on signs of arguments thus cases does not have to be used.

## 2.4 Servo control

As I have mentioned, position of the servo is being controlled by width of pulses sent to the servo. However Raspberry lacks the possibility to produce PWM with the hardware and pulses generated with the software are always different because operating system can not guarantee the exact time as we want, causing servo to jitter. The solution is using Direct Memory Access (DMA) for precise timing [15]. I have tried the application *ServoBlaster*, but under the load the width of pulses was slightly changing resulting in servo jitter. Another interesting application that I have tried for producing PWM is *pigpio* [18]. With oscilloscope I have measured very stable pulses with stable frequency 50 Hz. The Python module is available to talk with the pigpio daemon launched in the background—perfect option for me. Still the best way to generate PWM is with external driver unit, but the trick with DMA seems reliable for my application.

## 2.5 Graphical User Interface

GUI's main task is to plot aircrafts on the screen, wait for observer choice, and send signals to control threads. A map where aircrafts will be plotted is required. Powerful tools for geolocation management are Google Maps or OpenStreetMap APIs, but they are mainly aimed for mobile and web services and that is not the case of my application, so I will have to create the map by myself. There are many applications providing GUI libraries for Python, for instance *wxPython* that wraps popular wxWidgets, or advanced *PyQt* that is more than GUI toolkit. Still I will stick to *Tkinter* which is a little bit older package, but has advantage of being included with the Python standard library making it the most convenient and compatible toolkit to work with [22]. To have the code as fast as possible GUI methods should be also run parallel to the rest of the program. Tkinter main window has to be however initialized in the main thread for proper functioning so every other thread is its ancestor, therefore it can act as a control thread—it starts applications and threads and after the window is destroyed it terminates the rest.

Aircrafts will be plotted on the static piece of the map I have cut from the Google Map's website with the center in the Prague's airport. I have noted coordinates of its edges to know where to plot the aircraft. Its disadvantage is that no planes outside the area could be displayed. Algorithm of plotting is simple—every 500 ms is dictionary with aircrafts iterated and those aircrafts that have position inside the map area are plotted as a label with aircraft icon as an image. Additionaly, two labels are displayed over the aircraft—aircraft's ICAO type that must be queried to BaseStation database and the callsign. These information help observer with aircraft identifying. The blue dot represents system's position acquired from the GPS. The typical look of the main window is shown in the Figure 2.2.

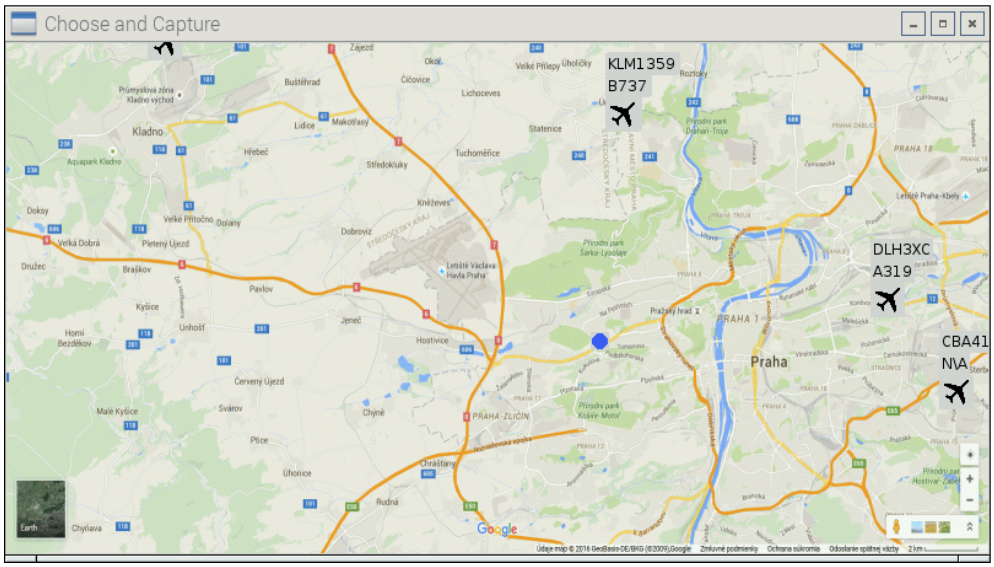



Figure 2.2: The application's main window with displayed aircraft's and observer's position.

Each aircraft's label handles the click on it as a creation of the new window where is every information about the plane. Only one Aircraft information window can be displayed at the time and information are refreshed every 500 ms. Example is shown in the Figure 2.3.

In the new window there are four buttons displayed, but only one, Start preview, is active. After click on the button following happens:

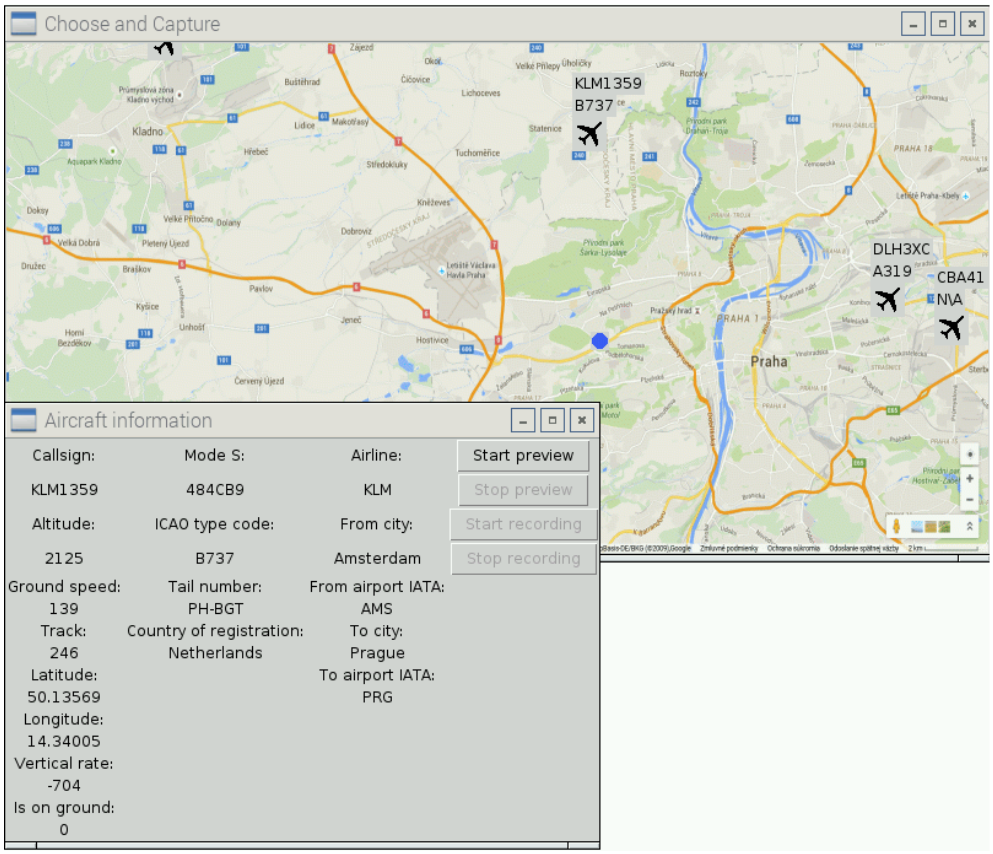



Figure 2.3: The Aircraft information window that is diplayed after the click on aircraft's icon.

- Both GUI loops are stopped,
- the dictionary is flushed to overcome being too large,
- button Start recording becomes active,
- live stream from the camera is displayed on the screen, and
- algorithm responsible for aircraft tracking is started.

### Camera control

A connection with the camera module is established with the object PiCamera from picamera package immediately after pressing the button. The view of the camera is displayed on the screen until the Stop preview button si pressed, therefore it must be run in an individual thread to not take all CPU resources. After successful connection red LED on the camera lights up. PiCamera also flips the view of the camera in case it is rotated and sets the image resolution—720p as a balance between the quality and the video size. The other button, Start recording, tells PiCamera to record the video to a file. The file is named after Mode S address of tracked aircraft and is *h264* format.

## 2.6 Tracking algorithm

The last task is to implement algorithm that from received ADS-B packets and system's position continually calculates the pan and the tilt angle and reorients camera

to smoothly follow the selected plane. To decrease the delay between the time of actual positional message transmission from the aircraft and the time of sending signal to a servo the message processing should be as close as possible to the packet reception from TCP port. And that is in the thread that normally parses data from messages to dictionary. The thread switches to the tracking mode—continues in analyzing the data only if ICAO address in the message is the same as address of tracked aircraft, otherwise packet is not important and is thrown away. Received navigational information (latitude, longitude, altitude) are stored in list structures along with time stamps (the time when message was generated is part of BaseStation message) for later position analyzation. These packets however may be received twice a second and that is not sufficient for smooth aircraft observation. Therefore the state estimation has to be implemented to have aircraft's position continually.

### Pan and tilt angles calculation

The fact that Earth is almost perfect sphere (Earth is slightly ellipsoidal, but ignoring these effects brings very low error to calculations) is no suprise for anybody. To find out the great-circle distance (the shortest distance over the Earth's surface) between two points with known latitudes and longitudes *Haversine* formula could be used:

$$d = 2R\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos\varphi_1\cos\varphi_2\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right), \tag{2.1}$$

where $d$ is the unknown distance,
$R = 6371\,\text{km}$ is the Earth's radius,
$\varphi_{1,2}$ is the latitude of first, respectively second point in radians and
$\lambda_{1,2}$ is the longitude of first, respectively second point in radians.

This formula is very precise—minor errors are caused only by mentioned ellipsoid effects and computing (rounding errors, etc.) [28].

For small distances between aircrafts and receiver occuring in my project the curvature of the Earth is negligible, resulting in the *Equirectangular* approximation. Much higher performance will be traded for slightly decreased accuracy, which is hardly noticeable [28].

By equirectangular approximation is meant shaping distances originally on the arc to straight lines. This forms the new triangle $P_1XP_2$ on the globe, as illustrated in the Figure 2.4.

From this triangle by *Pythagoras* theorem desired distance $d = |P_1P_2|$ could be calculated, but firstly distances $|P_1X|$ and $|XP_2|$ have to be known. To get $|P_1X|$ simple formula is used (variables have the same meaning as in the previous formula 2.1):

$$|P_1X| = R(\varphi_2-\varphi_1). \tag{2.2}$$

To get the size of second side of the triangle, a little bit modified formula brings the answer

$$|XP_2| = R\cos\left(\frac{\varphi_2+\varphi_1}{2}\right)(\lambda_2-\lambda_1). \tag{2.3}$$

The difference in this case is that longitudenal distance between two points depends on the latitude they have. If latitude is different the mean is taken. The fact that these distances are rather small must be kept in mind. Finally, the angle required from camera to tilt to capture chosen aircraft $\beta$ is

$$\beta = \tan^{-1}\left(\frac{h_2-h_1}{d}\right), \tag{2.4}$$

where $h_{1,2}$ is the altitude of first, respectively second point in kilometres.

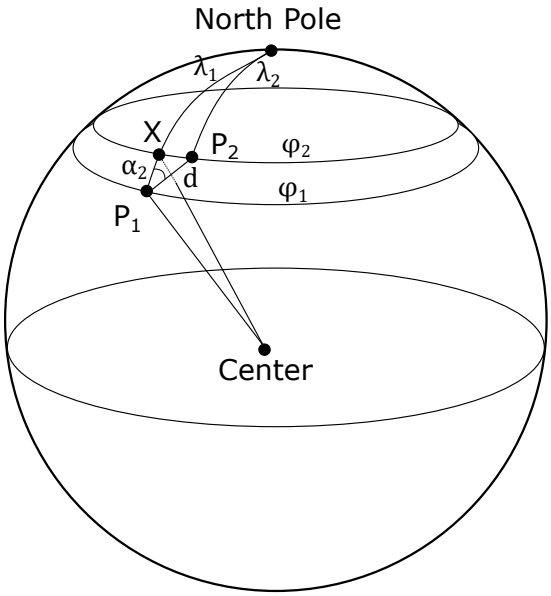



Figure 2.4: Model of the Earth with two points $P_1, P_2$ on the surface and designated relations between them.

The angle required from the camera to pan $\alpha$ is

$$\alpha = \alpha_2 - \theta - \theta_d = \cos^{-1}\left(\frac{|P_1X|}{d}\right) - \theta - \theta_d, \tag{2.5}$$

where $\alpha_2$ is the bearing between system and aircraft,
$\theta$ is system's magnetic heading, and
$\theta_d$ is the magnetic declination in the location of measurement.

A short program in Python will demonstrate the accuracy of equirectangular approximation on a distance between Prague and Košice:

```
import math
latPraha = math.radians(50.08333)
lonPraha = math.radians(14.41666)
latKosice = math.radians(48.7)
lonKosice = math.radians(21.25)
R = 6372.8
dapprox = R*math.sqrt((latKosice-latPraha)**2+
(math.cos(latPraha/2+latKosice/2)*(lonKosice-lonPraha))**2)
dhaversine = 2*R*math.asin(math.sqrt(math.sin(latKosice/2-latPraha/2)**2+
math.cos(latPraha)*math.cos(latKosice)*math.sin(lonKosice/2-lonPraha/2)**2))
print(dapprox)
518.0781837845868
print(dhaversine)
517.8584398669095
print(dapprox-dhaversine)
0.2197439176773
```

The error on a distance around 518 km is only 0.22 km, which is really sufficient for most purposes.

### State estimation

Aircraft's states are attributes that exactly define aircraft's location at the time. It is latitude, longitude, and altitude. To know values of these states between aircraft's broadcasts is important for smooth aircraft observating.

My idea is to make use of that aircraft broadcasts also velocity information—its groundspeed, track, and climbing rate. If I know the last position of aircraft and its velocity, simply by integrating the velocity I get its position at any time. If new position is received, it will be set as the initial condition and integration will be repeated. The estimation can be written as a system of three equations:

$$\dot{\varphi} = \frac{c_{kts} v_{gs} \cos\alpha}{R}, \tag{2.6}$$

$$\dot{\lambda} = \frac{c_{kts} v_{gs} \sin\alpha \cos\varphi}{R}, \tag{2.7}$$

$$\dot{h} = c_{ftps} v_{vr}, \tag{2.8}$$

where $\varphi$ is aircraft's estimated latitude in degrees,
$\lambda$ is aircraft's estimated longitude in degrees,
$h$ is aircraft's estimated altitude in kilometres,
$v_{gs}$ is aircraft's transmitted speed over the ground in knots,
$v_{vr}$ is aircraft's transmitted vertical rate in feets per second,
$c_{kts} = 0.00051444$ is constant converting velocity in knots to kilometres per second,
$c_{ftps} = 0.0003048$ is constant converting velocity in feets per second to kilometres per second,
$\alpha$ is aircraft's transmitted track, and
$R = 6371\,\text{km}$ is the Earth's radius.

For integrating ordinary differential equations in Python there is available subpackage *integrate* from the package *scipy* [27]. The function *ode* finds a solution $y(t)$ for equation $\dot{y} = f(y,t)$ by numeric integration—as this equation could be all three equations 2.6, 2.7, 2.8.

To allow integrating while the new messages are processed the integration should be done in an individual thread. As I have mentioned, every new position acquired sets the initial condition of the integrator. Then the new value of the state after some time, let us say $dt = 0.1\,\text{s}$, is calculated. The thread sleeps for the given time $dt$ and if afterwards no new information is available, the solution of the integrator is considered as the current position of the aircraft resulting in new pan and tilt angles calculation and servo movement. If new velocity information is available, the parameter of the integrated function is changed and integration continues. This algorithm can provide aircraft's position almost in any time I want.

Now using the Python's package *matplotlib* I will plot and compare captured coordinates during the observation with those estimated by myself. In the Figure 2.5 can be seen that estimated values of latitude are very very close to the real ones. That applies also for the longitude estimation as can be seen in the Figure 2.6. The maximum error of the estimation was 10 m, but mostly the error is smaller and should not cause any big discrepancies during the observation (this can be hardly seen in graphs without zooming). However in the Figure 2.7 can be seen that estimated altitude value goes totally off from real values. It is caused by too high resolution of the climbing rate sent from the aircraft—64 ft/s. This approach definitely can not be used for precise altitude estimating. Second idea is to take two last altitude measurements and from the time difference between them the climbing rate will be calculated. The result is shown in the Figure 2.8. The line is much closer to real received altitudes, but there is similar problem as with the climbing rate—the resolution of transmitted altitude is 25 ft (7.62 m) and that looks like not sufficient value for precise altitude estimating. The plane seems to hold on one altitude and after a while it descends to another level, but in reality it is continualy descending. However, such a error from larger distances may have only small effect. To compare, latitude and longitude information have five decimal places with last place's resolution circa 1.1 m.

Finally after implementing this steps I should have everything prepared for the aircraft observating with the constructed system.

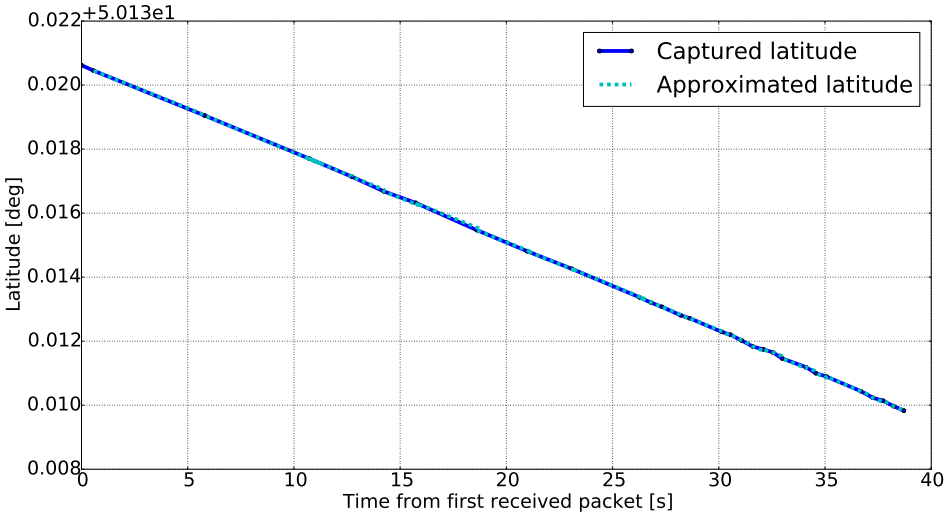



Figure 2.5: Graph indicating latitude growth during the landing with a comparison of the estimated and real value.

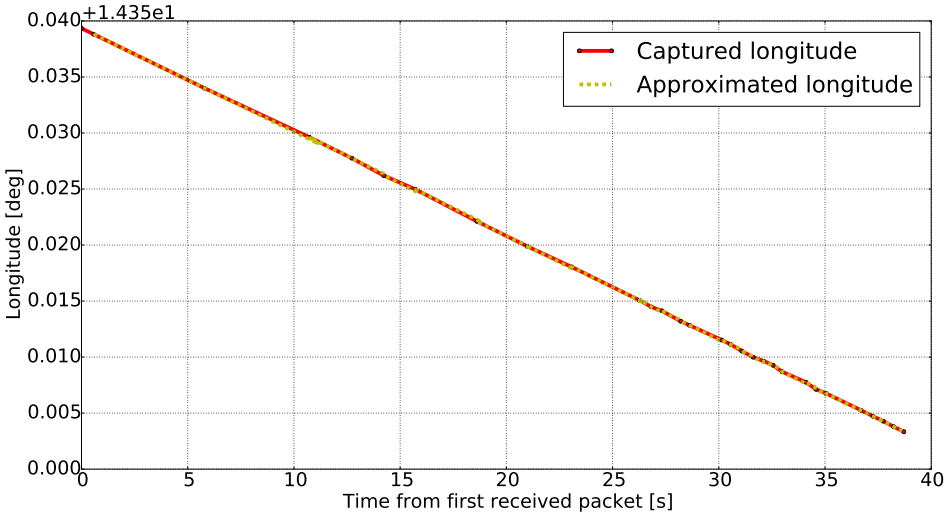



Figure 2.6: Graph indicating longitude growth during the landing with a comparison of the estimated and real value.

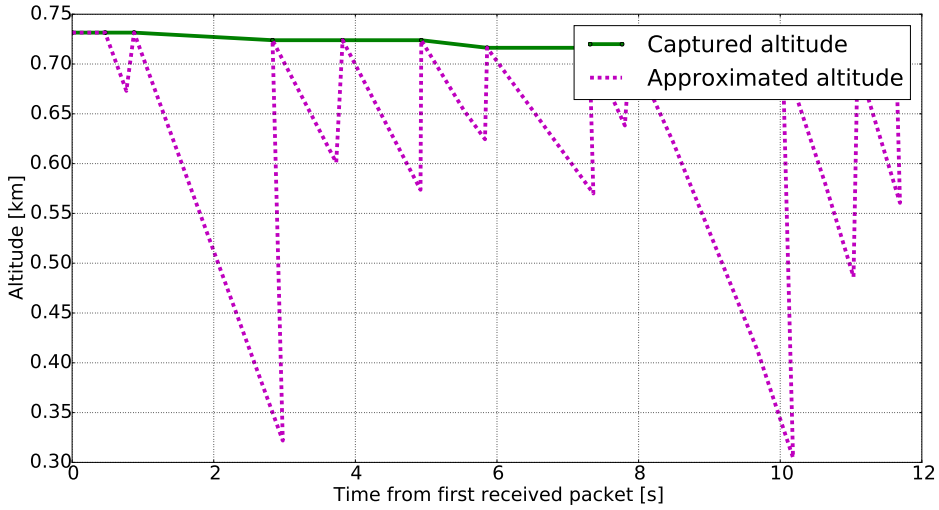



Figure 2.7: Graph indicating altitude growth during the landing with a comparison of the estimated and real value. Climbing rate is taken from the ADS-B message.

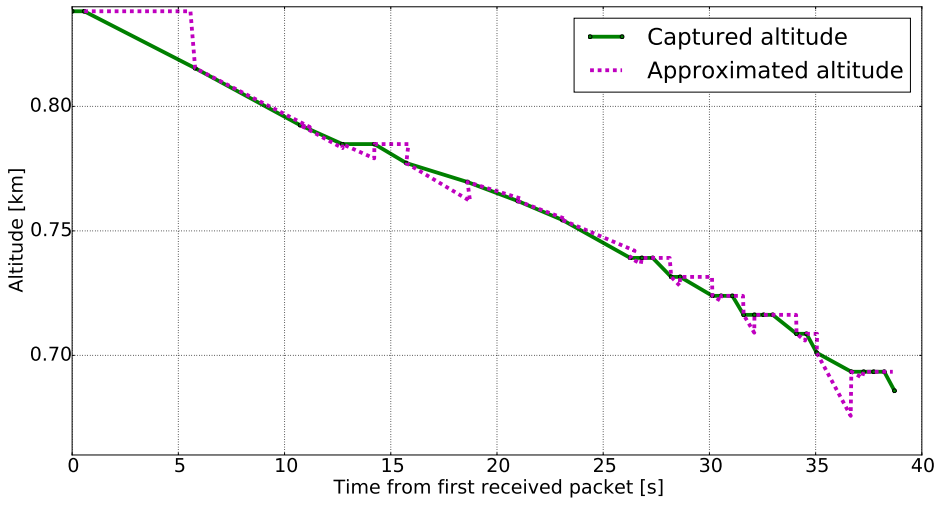



Figure 2.8: Graph indicating altitude growth during the landing with a comparison of the estimated and real value. Climbing rate is calculated as an altitude difference.

# 3. System test

After the final construction I have travelled to the vicinity of Prague's airport to test the functionality of the system (my position is shown in the Figure 3.1).

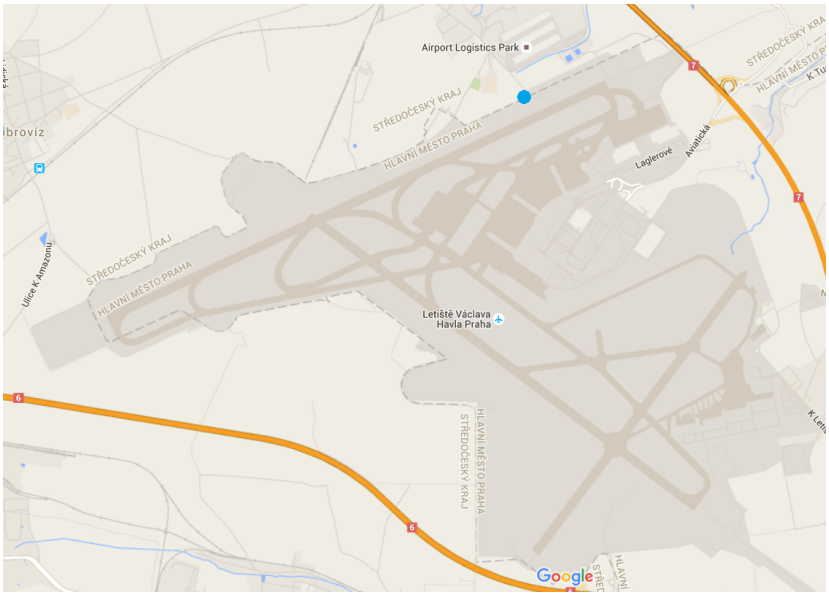



Figure 3.1: My position during the test.

The conclusion is that system worked, but was inaccurate. That means the camera was moving with the plane but was pointing the other direction. Here is the list of factors that may caused it:

- GPS readings were constantly changing,
- magnetometer was also showing different values in the same position. The magnetic heading was determined with the compass inside my smartphone,
- I have not precisely measured marginal angles of servos,
- ADS-B receiver was overloaded because of lots of broadcasts in the area, causing delay.

Although none of this errors was significant, together they may easily caused non-negligible error. Unfortunately, from unknown reason, I was not receiving packets from aircrafts on the ground, therefore I could not capture take offs, which would be simpler to follow. It was definitely not mistake in my code, but rather failure in the dump1090 software. Packets coming from landing aircrafts were correctly received. The power supply was reliably providing the energy for the system for two hours.

The lens were causing some troubles too. Consequence of the high torque produced by them was image jittering. Solution might be applying balance mass on the opposite site of the rotation. Furthermore, giving 6 V to servos would improve their behaviour—resulting in less camera jittering and more fluent moving. As another big lens disadvantage I find lack of ability to automatically focus on the target. The distance of aircraft from my position was always changing and lens were not able to focus on the frame. Ideal position of the camera would be so the distance from aircraft does not change much.

Though I managed to extract one picture of a full aircraft from one of videos I have made. The quality is slightly worse because of mentioned focus problem and weather conditions.


Figure 3.2: Lufthansa's Boeing 737-300 captured by the system.

# Conclusions

To conclude, the task to build a system for automatic aircraft observation and recording based on ADS-B surveillance system has been completed. Both the construction and the application act as I expected, but there is still lots of to improve. System's control unit is a microcomputer Raspberry Pi that communicates with observer through the Raspberry Touchscreen. Software plots captured aircrafts with the RTL-SDR receiver on the map and after a click on the plane displays all available information about it, including those acquired from external databases. Parallel to the main GUI window, software communicates with GPS receiver and magnetometer and tries to determine observer's position and orientation. The platform is so far allowed only to be level to the Earth's surface. After the selection of aircraft that user would like to capture, the program reorients the Raspberry Pi Camera towards the plane by sending signals to servomotors and shows the view of the camera on the screen. If the user likes the image, he can record a video by simple click on the button in the application. The system is powered by series of eneloop batteries with a voltage regulated by DC-DC converter and can operate for more than two hours.

The test has showed that the system was inaccurate (camera was not pointed exactly on the aircraft), but it is more important that the idea could be realized. It just requires better calibration—more precise positional and magnetic readings and servo angular position calibration. The software might also want optimization to be faster because of high density of ADS-B packets transmitted in the area. Moreover, the pan/tilt platform requires upgrade in terms of releasing the high torque caused by large and heavy lens and full 6 V should be provided for servomotors to reduce camera jittering. Now the position of the system must be chosen carefully with respect to possibilities of the lens and that brings limits to the application, which I do not want. The best solution is replacing the lens with those capable of automatic focus and motorized zoom.

Following these recommendations the performance of the system could be greatly improved and camera could nicely follow the plane. I will continue to publish details of the system to inspire and provide my gained knowledge for the whole DIY community on the website `https://dufiblog.wordpress.com/`.

# Bibliography


[1] Advanced Tech.
*GPS Receiver U-blox NEO-6M Module with Ceramic Antenna TTL Interface for raspberry pi 2*. [Online; accessed 1-May-2016]. URL: http://www.aliexpress.com/item/GPS-Receiver-U-blox-NEO-6M-Module-with-Ceramic-Antenna-TTL-Interface-for-raspberry-pi-2/32364699900.html.

[2] Airports Authority of India.
*Automatic Dependent Surveillance–Broadcast (ADS-B) Out Based ATS Surveillance Services*. [Online; accessed 12-May-2016]. 2014. URL: http://www.aai.aero/misc/AIPS_2014_18.pdf.

[3] Airservices Australia.
*Automatic Dependent Surveillance Broadcast*. [Online; accessed 12-May-2016]. 2016. URL: http://www.airservicesaustralia.com/projects/ads-b/.

[4] antirez.
*dump1090*. [Online; accessed 12-May-2016]. 2015. URL: https://github.com/antirez/dump1090.

[5] ArduCam.
*Rev.C OV5647 Camera for Raspberry Pi Improves the Optical Performance*. [Online; accessed 14-May-2016]. 2015. URL: http://www.arducam.com/raspberry-pi-camera-rev-c-improves-optical-performance/.

[6] Avamation mechatronic.
*Raspberry Pi I2C clock-stretching bug*. [Online; accessed 20-May-2016]. 2013. URL: http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html.

[7] BareBones.
*The Definitive Guide to the Kinetic SBS Virtual Radar*. [Online; accessed 13-May-2016]. URL: http://woodair.net/SBS/Article/Barebones42_Socket_Data.htm.

[8] Peter Bennett.
*The NMEA FAQ*. [Online; accessed 1-May-2016]. URL: http://www.kh-gps.de/nmea.faq.

[9] E. Chang et al.
*The Story of Mode S: An Air Traffic Control Data-Link Technology*. [Online; accessed 12-May-2016]. 2000. URL: http://web.mit.edu/6.933/www/Fall2000/mode-s/index.html.

[10] The Pennsylvania State University College of Earth and Mineral Sciences.
*GPS and GNSS for Geospatial Professionals*. [Online; accessed 1-May-2016]. URL: https://www.e-education.psu.edu/geog862/node/1884.

[11] FlightAware.
*FlightFeeder*. [Online; accessed 12-May-2016]. 2016. URL: https://flightaware.com/adsb/flightfeeder/.

[12] gpsd.
*A GPS service daemon*. [Online; accessed 1-May-2016]. URL: http://www.catb.org/gpsd/.

[13] GPS.gov.
*The Global Positiong System*. [Online; accessed 1-May-2016]. URL: http://www.gps.gov/systems/gps/.

[14] Gus.
*Beaglebone Vs Raspberry Pi 2: Choosing The Right Board*. [Online; accessed 14-May-2016]. 2015. URL: https://pimylifeup.com/beaglebone-vs-raspberry-pi/.

[15] Richard Hirst.
*ServoBlaster*. [Online; accessed 23-May-2016]. URL: https://github.com/richardghirst/PiBits/tree/master/ServoBlaster.

[16] Gordon Hollingworth.
*The Eagerly Awaited Raspberry Pi Display*. [Online; accessed 14-May-2016]. 2015. URL: https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/.

[17] Honeywell.
*Compass heading using magnetometers*. [Online; accessed 13-May-2016]. URL: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/Magnetic__Literature_Application_notes-documents/AN203_Compass_Heading_Using_Magnetometers.pdf.

[18] joan2937.
*The pigpio library*. [Online; accessed 23-May-2016]. URL: http://abyz.co.uk/rpi/pigpio/index.html.

[19] LeapSecond.com.
*TAI, UTC, GPS clocks*. [Online; accessed 1-May-2016]. URL: http://www.leapsecond.com/java/gpsclock.htm.

[20] ModMyPi.
*What's The Difference Between DC, Servo & Stepper Motors?* [Online; accessed 14-May-2016]. URL: http://www.modmypi.com/blog/whats-the-difference-between-dc-servo-stepper-motors.

[21] Stephen Mraz.
*What's the Difference Between Pitch, Roll, and Yaw?* [Online; accessed 1-May-2016]. 2014. URL: http://machinedesign.com/engineering-essentials/what-s-difference-between-pitch-roll-and-yaw.

[22] A Kenneth Reitz Project.
*GUI Applications*. [Online; accessed 23-May-2016]. 2016. URL: http://docs.python-guide.org/en/latest/scenarios/gui/.

[23] Python Software Foundation.
*Data structures*. [Online; accessed 16-May-2016]. URL: https://docs.python.org/3.4/tutorial/datastructures.html#dictionaries.

[24] Raspberry Pi Foundation.
*Downloads*. [Online; accessed 15-May-2016]. URL: https://www.raspberrypi.org/downloads/.

[25] Raspberry Pi Foundation.
*What are power requirements?* [Online; accessed 14-May-2016]. URL: https://www.raspberrypi.org/help/faqs/#powerReqs.

[26] RTL-SDR.com.
*About RTL-SDR*. [Online; accessed 12-May-2016]. 2016. URL: http://www.rtl-sdr.com/about-rtl-sdr/.

[27] SciPy.org.
*Integration*. [Online; accessed 24-May-2016]. URL: http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html.

[28] Movable Type Scripts.
*Calculate distance, bearing and more between Latitude/Longitude points*. [Online; accessed 23-May-2016]. URL: http://www.movable-type.co.uk/scripts/latlong.html.

[29] Junzi Sun.
*ADS-B Decoding Guide*. [Online; accessed 12-May-2016]. 2015. URL: http://adsb-decode-guide.readthedocs.io/en/latest/introduction.html.

[30] Tiger963.
*What are the differences between Bearing vs Course vs Direction vs Heading vs Track?* [Online; accessed 1-May-2016]. URL: http://aviation.stackexchange.com/questions/8000/what-are-the-differences-between-bearing-vs-course-vs-direction-vs-heading-vs-tr.

[31] tutorialspoint.com.
*Python - Tutorial*. [Online; accessed 15-May-2016]. URL: http://www.tutorialspoint.com/python/.

[32] The Pennsylvania State University.
*Details of the GPS position calculation*. [Online; accessed 1-May-2016]. URL: https://www.courses.psu.edu/aersp/aersp055_r81/satellites/gps_details.html.

[33] Phil Vabre.
*Air Traffic Services Surveillance Systems, Including An Explanation of Primary and Secondary Radar*. [Online; accessed 9-May-2016]. URL: http://www.airwaysmuseum.com/Surveillance.htm.

[34] Virtual Radar Server.
*Routes and Other Standing Data*. [Online; accessed 16-May-2016]. URL: http://www.virtualradarserver.co.uk/FlightRoutes.aspx.

[35] Christian Wolff.
*Radar Basics*. [Online; accessed 13-May-2016]. URL: http://www.radartutorial.eu/13.ssr/sr24.en.html.

# List of Figures


1 Simon Aubury's Pi Plane project. . . . . 4

1.1 The scheme of system's components. . . . . 5
1.2 RTL-SDR receiver. . . . . 7
1.3 ADS-B message content. . . . . 8
1.4 GPS module. . . . . 12
1.5 GPS test results. . . . . 13
1.6 Illustration of a heading, bearing, and pan angle. . . . . 14
1.7 BNO055 9-DOF Absolute Orientation IMU Fusion Breakout. . . . . 15
1.8 Magnetometer's coordinate system. . . . . 16
1.9 Powerful Hitec servo that will actuate the camera. . . . . 17
1.10 Overview of individual components that is pan/tilt made of. . . . . 18
1.11 Pan/tilt platform designed in the Autodesk 123D application with designated rotational axes. . . . . 19
1.12 Small, inexpensive Linux-based computer Raspberry Pi 2 Model B. . . . 19
1.13 Overview of the Raspberry Pi Official Touchscreen and its designed holder. 20
1.14 Overview of the camera with lens used in my project. . . . . 21
1.15 The overview of used accumulators with step-down DC-DC converter. . 22
1.16 The final scheme of system's components and communication interfaces. 23
1.17 Photos of the constructed system. . . . . 24

2.1 Simplified scheme of software subtasks and designated connections with sensors. . . . . 26
2.2 The application's main window with displayed aircraft's and observer's position. . . . . 28
2.3 The Aircraft information window that is diplayed after the click on aircraft's icon. . . . . 29
2.4 Model of the Earth with two points $P_1, P_2$ on the surface and designated relations between them. . . . . 31
2.5 Graph indicating latitude growth during the landing with a comparison of the estimated and real value. . . . . 33
2.6 Graph indicating longitude growth during the landing with a comparison of the estimated and real value. . . . . 33
2.7 Graph indicating altitude growth during the landing with a comparison of the estimated and real value. Climbing rate is taken from the ADS-B message. . . . . 34
2.8 Graph indicating altitude growth during the landing with a comparison of the estimated and real value. Climbing rate is calculated as an altitude difference. . . . . 34

3.1 My position during the test. . . . . 35
3.2 Lufthansa's Boeing 737-300 captured by the system. . . . . 36
3.3 Declination data in Prague, acquired from NOAA's website. . . . . 49

# List of Tables

1.1 Decoded ADS-B message. . . . . 8
1.2 Description of individual messages in the BaseStation format. . . . . 10
1.3 Comparison of current GPS and UTC time. . . . . 12

# List of Abbreviations

**ADS-B** Automatic Dependent Surveillance - Broadcast . . . 3

**ATC** Air Trafic Control . . . 3

**PSR** Primary Surveillance Radar . . . 5

**SSR** Secondary Surveillance Radar . . . 6

**ICAO** International Civil Aviation Organization . . . 6

**TCAS** Traffic Collision Avoidance System . . . 6

**GPS** Global Positioning System . . . 6

**SDR** Software Defined Radio . . . 7

**MSL** Mean Sea Level . . . 9

**IATA** International Air Transport Association . . . 27

**UTC** Coordinated Universal Time . . . 12

**NMEA** National Marine Electronics Association . . . 12

**GPSD** GPS Daemon . . . 13

**NOAA** National Oceanic and Atmospheric Administration . . . 15

**PWM** Pulse Width Modulation . . . 16

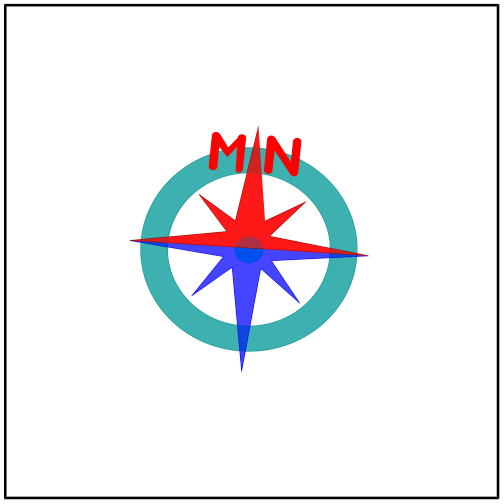
**UART** Universal Asynchronous Receiver/Transmitter . . . 12

**GPIO** General Purpose Input/Output . . . 18

# Attachments

## Declination

| | |
|---|---|
| **Date** | 2016-04-18 |
| **Latitude** | 50.082533° N |
| **Longitude** | 14.345592° E |
| **Elevation** | 0.0 km GPS |
| **Model Used** | WMM2015 |
| **Declination** | 3.67° E changing by 0.13° E per year |
| **Uncertainty** | 0.36° |




Compass shows the approximate bearing of the magnetic north (MN)

Magnetic declination is the angle between true north and the horizontal trace of the local magnetic field. In general, the present day field models such as the IGRF and World Magnetic Model (WMM) are accurate to within 30 minutes of arc for the declination. However, local anomalies exceeding 10 degrees, although rare, do exist.

Document created: 2016-04-18 20:53 UTC

Help: How to interpret results Questions: geomag.models@noaa.gov

Figure 3.3: Declination data in Prague, acquired from NOAA's website.