

DIPLOMARBEIT

BreakdownDrone

2016/17 von:

Image processing

Fabian PIRIBAUER

5AHIF

Programming of the drone

Florian UNGERSBOECK

5AHIF

Betreuer / Betreuerin:

MMag. Dr. Michael Stifter

Wiener Neustadt, am 3rd April, 2017

Abgabevermerk:

Übernommen von:

Contents

Diplomarbeit Dokumentation	iv
Diploma Thesis Documentation	vi
1 Introduction	1
1.1 Overview (Florian)	1
1.2 Goal (Florian)	1
1.3 Definition (Florian)	1
1.4 History (Florian)	2
1.5 Commercial use of Drones (Fabian)	2
1.6 Legal Position (Florian)	3
1.7 Related Work (Fabian)	4
1.8 Social Aspect (Florian)	4
2 Hardware (Florian)	6
2.1 Drone	6
2.1.1 Autopilot System	7
2.1.2 GPS Module	7
2.1.3 Telemetry Module	7
2.1.4 Motors	8
2.1.5 Battery Pack	8
2.2 Custom Build Control Module	9
2.2.1 Raspberry Pi 3	9
2.2.2 PiCam	10
2.2.3 Motor to Release the Triangle	10
2.3 Breakdown Triangle	10
2.3.1 Release Mechanism	12
2.4 Communication between Drone and Raspberry	12
2.4.1 Architecture	12
2.4.2 Host Controller	13
2.5 Alternative Hardware	13
2.5.1 Drone	13
2.5.2 Raspberry Pi	14
3 Software (Fabian)	15
3.1 Programming Language	15
3.1.1 Background	15
3.1.2 Why Python2.7?	15
3.2 MAVLink (Florian)	15

3.2.1	Message System	16
3.3	DroneKit (Florian)	16
3.3.1	API Features	17
3.3.2	Connection to a Vehicle	17
3.3.3	Vehicle Modes	17
3.3.4	Vehicle Attributes	18
3.3.5	Vehicle Movement	18
3.4	Communication	20
3.5	Camera Stream	20
4	Navigation (Fabian)	21
4.1	Global Positioning System (Fabian)	21
4.1.1	Calculating the Distance between two GPS Coordinates	21
4.1.2	Offsetting a GPS Coordinate	22
4.2	Usage of Map Data	23
4.2.1	OpenStreetMap	23
4.2.2	Reverse Geocoding	24
5	Computer Vision (Fabian)	26
5.1	Misconception of Computer Vision	26
5.2	OpenCV	26
5.3	Image Data Format	27
5.4	Smoothing	27
5.4.1	How does Smoothing Work?	27
5.4.2	Convolution Matrix	28
5.4.3	Mean Filter	28
5.4.4	Gaussian Blur	28
5.4.5	Implementation in OpenCV	29
5.5	Edge Detection	29
5.5.1	Canny Edge Detector:	29
5.5.2	Implementation in OpenCV	31
5.6	Line Detection	32
5.6.1	Hough Line Transform	32
5.6.2	Implementation in OpenCV	32
5.7	Relative Position to Landmarking	34
5.7.1	Angle Calculation	34
5.8	Distance Estimation	35
5.8.1	Camera and Focal Length	35
5.8.2	Field of View Calculation	35
5.8.3	Convert Pixels to Actual Meters	36
5.8.4	Get Shortest Distance in Pixels	37
6	Flight(Florian)	38
6.1	Autonomous vs. Manual Mission	38
6.2	Pre-Flight	38
6.3	Takeoff	38
6.4	Flying the Path	39
6.4.1	Heading for a Node	39
6.4.2	Calculating the Distance to a Node	39
6.5	Landing and Steering	40

6.6	Releasing the Breakdown Triangle	41
6.7	Way back to the Starting Point	42
6.8	Performance Analysis	43
6.8.1	Why Performance Analysis?	43
6.8.2	Different Methods of Analysis	43
6.8.3	Tested Device	44
6.8.4	Indicators Analyzed	44
6.8.5	Performance during Flight	45
6.8.6	Discussion of Results	45
6.8.7	Methods of increasing Performance	45
7	Conclusion	47
7.1	What was Planned?	47
7.2	Benchmarking	47
7.2.1	Flight Accuracy	47
7.2.2	Landing Accuracy	47
7.3	Testing	48
7.3.1	Testing of Flight Accuracy	48
7.3.2	Testing of the Landing Accuracy	48
7.4	Results	49
7.5	Further Development	49
7.5.1	Obstacle Avoidance	49
7.5.2	Hardware Upgrade	50
7.5.3	Triangle	50
	Acronyms	51
	Bibliography	52

Diplomarbeit Dokumentation

Namen der Verfasser/innen	<div> <div>Name 1</div> <div>Name 2</div> <div>Name 3</div> <div>Name 4</div> <div>Name 5</div> </div> <div> Dieses PDF ausfüllen und als PDF drucken. Das gedruckte PDF in den Ordner pdf speichern. </div>
Jahrgang Schuljahr	5AHIF 2015 / 16
Thema der Diplomarbeit	Entwurf eines Versuchstandes für Kreispumpen
Kooperationspartner	Irgendeine Firma wenn vorhanden

Aufgabenstellung	Hier steht die Aufgabenstellung in einigen Sätzen.
------------------	--

Realisierung	Beschreibung der Realisierung.
--------------	--------------------------------

Ergebnisse	Beschreibung der Ergebnisse.
------------	------------------------------

Typische Grafik, Foto
etc. (mit Erläuterung)

Beschreibung des unten eingefügten Bildes. Unten
klicken um das Bild zu wählen. Das Bild muss als
PDF vorliegen (z.B. mit GIMP als PDF speichern).

1850 x 1950
1850
1750
1650

Teilnahme an
Wettbewerben,
Auszeichnungen

1. Preis in Irgendeinem Wettbewerb

Möglichkeiten der
Einsichtnahme in die
Arbeit

HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

Approbation

Prüfer

Abteilungsvorstand

(Datum, Unterschrift)

Mag. Max Mustermann

AV Mag. Max Mustermann


Diploma Thesis Documentation

Authors	Name 1 Name 2 Name 3 Name 4 Name 5
Form	5AHIF
Academic Year	2015 / 16
Topic	Titel englisch
Co-operation partners	Irgendeine Firma wenn vorhanden

Assignment of tasks	Hier steht die Aufgabenstellung in englisch in einigen Sätzen.
---------------------	--

Realization	Beschreibung der Realisierung auf englisch.
-------------	---

Results	Beschreibung der Ergebnisse auf englisch.
---------	---

	COLLEGE OF ENGINEERING WIENER NEUSTADT
	Department: Informatik Educational Focus: Maschineningenieurwesen

Illustrative graph, photo (incl. explanation)	Englische Beschreibung des unten eingefügten Bildes. Unten klicken um das Bild zu wählen.
	<div> <div>1850 x 1950</div> <div>1850</div> <div>1750</div> <div>1650</div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>

Participation in competitions, Awards	1. Preis in Irgendeinem Wettbewerb
--	------------------------------------

Accessibility of diploma thesis	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt
---------------------------------	--

Approval (Date, Sign)	Examiner Mag. Max Mustermann	Head of Department AV Mag. Max Mustermann
--	---	--

Abstract

The rapid growth of automobiles has lead to an increasing amount of breakdowns and unfortunately brings its death toll with it. The amount of resulting secondary accidents must not be ignored. The BreakdownDrone is an thesis about whether it is feasible to use unmanned aerial vehicles in order to make those slightly less likely by fully autonomously placing a breakdown triangle. A quadcopter is used for the mission and is supported by a companion computer, a Raspberry Pi. For finding the flight path, the Raspberry is equipped with a Pi Camera Module v2 which is used in conjunction with the Open Computer Vision library. By applying a number of filter, edge detection and line detection algorithms it is possible to extract the road markings from the camera image and determine angle, as well as the distance to the line. Furthermore the precision of the vehicle's GPS stabilized navigation system is called into question.

Kurzfassung

Chapter 1

Introduction

1.1 Overview (Florian)

During the last 100 years the usage of cars, trucks and motorcycles has grown very rapidly. With an increasing amount of vehicles on the streets results in more accidents. This is why there were many approaches in order to protect people involved in an accident. One of them is the breakdown triangle. The problem with the breakdown triangle is that not only it is very dangerous to set it up while the traffic is flowing fluently, but also the distance between the vehicle and the triangle has to be correct to make sure everyone is able to recognize a dangerous situation in time.

1.2 Goal (Florian)

The drone is supposed to deploy a breakdown triangle autonomously by flying along the roadside, as soon as the source distance between the drone and the crashed vehicle is long enough the breakdown triangle gets put down in the correct orientation to the street near the edge of the road.

1.3 Definition (Florian)

A drone is what most people call an **Unmanned Aerial Vehicle (UAV)** in their everyday language [1]. It flies without a pilot on board of the vehicle. It can either be controlled by a pilot using a remote control or the drone flies autonomously (controlled by an onboard computer) which means it can fly without any interaction between a human and the aircraft [2].

UAVs can vary greatly for example in its size and the number of motors/propellers attached, but in commercial use the most common vehicle is a quadcopter. Four propellers are attached to motors which are driven by the flight controller, the heart of every drone. The flight controller uses a **Global Positioning System (GPS)** module to maintain a stable flight. In a remote controlled mission a telemetry is required in order to transmit collected data to the controller, so the pilot can determine what state the vehicle currently is in. Additionally the telemetry receives the control instructions from the pilot via the remote control and forwards commands to the flight controller.[3] During an autonomous mission the drone moves by commands from a running program on the flight controller or even an external controller attached to the **UAV**.

1.4 History (Florian)

One of the first unmanned air raids happened in August 22, 1849 where Austrians, who already possessed parts of Italy, attacked Venice. The Austrians launched about 200 balloons with explosives attached to them. Some of the bombs exploded like planned (by a timed fuse) but some actually got blown back over the Austrian border because the wind suddenly changed [4].

The interest in Unmanned Aerial Vehicles for military use has grown since the bumbling Austrian experiment. During World War 1 Archibald Montgomery Low invented the first remote controlled aircraft [5]. The US military had great hope in **Unmanned Aerial System (UAS)** and developed the Hewitt-Sperry Automatic Airplane. When the first world war ended in 1918 the development of **UAVs** was paused, but in 1935 the English film and television actor Reginald Denny became interested in radio controlled model airplanes and opened a model plane shop. [6]

During the Second World War the Nazi German Luftwaffe developed and used various **UAVs**, one of them being the so called flying bomb V1 (Vergeltungswaffe 1). It was a radio controlled bomb with about 1 ton of explosives on board, mostly used to attack London and Belgium in 1944 [7]. After World War II jet engines were developed and used for drones by the Australian Government Aircraft Factories [8].

The US Air Force started their **UAV** program (today known as “Red Wagon”) after the Soviet Union shot down one of their U-2 spy planes during the cold war in 1960 [9]. During the War of Attrition (1967-1970) in the Middle East Egyptians first used a drone with a camera attached to it to gain awareness about the enemy movements. Just three years later the US military confirmed their usage of drones during the Vietnam war [10] in order to not unnecessarily lose more pilots during high risk reconnaissance missions. Israel's military developed a drone with great success during the 1982 Lebanon War. Israel managed to take down all Syrian air defense without the loss of a single pilot [11]. During the 1980s and 1990s drones became increasingly more significant for the US military.

The European Union started a project in 2002 called “CAPECON”. Its objective was to advance **UAV** technology in military and commercial use, thus improving reliability, safety and performance. The project was set aside in 2005 [12]. Until 2012 the US Airforce deployed almost 7500 drones which is 30% of all US aircrafts [13].

Recent drones also became increasingly more important in commercial use. Big companies are planning to use them for their daily business. One example is Amazon Prime Air, where Amazon tries to deliver their orders within 30 minutes or less [14].

1.5 Commercial use of Drones (Fabian)

Although drones have been around for much longer, only in the last few years the open market picked up on them. As the technology advances, building a business out of drones becomes much more imaginable. Also, with change of the law in favor of commercial **UAVs** entrepreneurs are more tempted towards using them. The federal government of the United States of America has already made it easier to get a certificate for commercial missions, trying to shift companies to the new trend[15]. Some may argue that this will lead to a dramatic loss of privacy, since almost every drone uses a camera for collision detection reasons, but also for data collection. But there are some great uses for **UAVs** in the industry.

Possible Usages of Drones

Object Delivery: Since drones are so flexible, it's just a matter of time when they will be used to transport small objects like letter from A to B. The company Amazon has announced to use drones in the near future to deliver packages [14]. The only problem is that the current 'sense and avoid' technology is not sophisticated enough to ensure the safety of the those near the UAV and therefore it is still in development.

Agriculture: One noticeable advantage of drones is their capability of creating maps of huge areas with little effort. They are faster and cheaper than other solutions like helicopters or satellite imagery. Using a near infrared camera you can render a map which displays the vegetation index of the crops [16]. This could help a farmer to quickly discover droughts or comparing fields that were differently fertilized.

Photography: Drones give professional photographers a whole new perspective that can be used. It's easier to make pictures of a large amount of people for example at a wedding.

Media: As with photography, the media would be able to replace expensive news helicopters with small drones providing a live media feed.

Mining: Drones can be used to generate height maps and 3D models of quarries. Those models can be used to calculate the volume of certain piles which usually needs a few work hours of estimating the volume.

Safety and Security: Drones are already used in the military for border control and various other tasks. Shouldn't we consider better applications to improve our overall safety? This is what this thesis is about. We try to find dangerous tasks that pose no problems to drones and let them do it instead. In our case it's to safely position a breakdown triangle. An even more life critical mission would be to find avalanche-buried people. But there is certainly more which drones can be used for, especially for emergency services.

1.6 Legal Position (Florian)

In many countries like Austria the flight of unregistered drones is prohibited outdoors. The organization "AustroControll" allows the usage of drones in public areas. There are a few classifications which are differentiated in Austria.

class one (within visual contact)

These kind of UAVs, with a maximum of 150kg, are only allowed to fly within visual contact and a maximum height of 150 meters. The drone has to be marked accordingly. Operators of the vehicle gets an approval after a technical and operational examination.

class two (without visual contact)

A drone pilot of a class two vehicle does not have to have sight contact with the drone but the permission for flying the drone is the same as the allowance of civil aviations. The Pilot's license is also required.

In general class one vehicle gets classified by their danger which potential means they get differentiated by their weight and their field of usage.

	field of usage			
	I: untilled	II: unsettled	III: settled	IV: densely settled
mass with max. 5kg	A	A	B	C
mass with max. 25kg	A	B	C	D
mass with max. 150kg	B	C	D	D

In order to get the class A approval needed for the Breakdown Drone following documents need to be sent to AustroControll:

- detailed description of the UAV
- declaration of reliability
- certificate of insurance
- photographic identification of the pilot

Additional requirements needed for class B approval are:

- proof that the drone matches the building regulations
- noise report
- operational safety analysis
- declaration of qualification of the pilot(s)

For type C and D, in addition to all of the points mentioned before:

- pilots license and a confirmation of medical fitness
- confirmation of knowledge of air traffic law

1.7 Related Work (Fabian)

In the last couple of years research about UAVs became increasingly more popular. Besides military and commercial use, there have been some studies which focus on civil applications, like traffic control or search and rescue missions. In [17] a multipurpose UAV has been built for search and rescue operations in the event of a avalanche. Like this study, where prevention of follow up accidents is the main goal, such technology can save lives and it's vitally important to encourage this kind of research.

1.8 Social Aspect (Florian)

A drone flying above our heads could mean anything. It could mean it is from a news channel making a new documentation about our wildlife, but it could as well be a drone used to monitor a specific citizen. This uncertainty is what makes most people very skeptical about commercially used drones especially the older generations.

Various UAVs can be used and are partly already used for very dangerous missions like

taking a look around and inside a contaminated atomic power plant. Natural catastrophes for example would be another occasion where the usage of drones shows their big advantages over humans or other ground machines. They are very accurate, easy to use and above all that they are very fast when it really counts. Imagine someone is partially trapped under a house after a big earthquake, where search groups could take days and perhaps even weeks to find someone, drones would only take a few minutes for finding someone and then reporting the exact location of the trapped person back to the search group. Other positive usages could be delivering important medicine to people needing them fast, like hikers being bit by a venomous animal that need antidotes before it is too late. These are just a few things mentioned that can really help society.

On the other hand there are always gonna be people that use such technologies for illegal activities. No one wants to be observed at any given time, especially if you do not even know for sure and just live with the guess of getting looked at right now. Without knowing for sure what this specific drone is doing flying right over peoples heads, everyone will certainly be skeptic. On the other hand it can be a key moment in a police investigation when searching for an criminal. This uncertainty is what makes this huge controversy around this subject.

Another motive is the military use of drones in modern times. Can a soldier ethnically agree to potentially kill hundreds of humans by remotely flying a drone and dropping off a bomb? This can be a huge ethical issue even if it can mean saving thousands of people.

Chapter 2

Hardware (Florian)

2.1 Drone

The drone itself is a commercial drone called the 3DR Iris+. It is put together by the many parts listed here. A drone or basically any **UAV** system is able to roll around three axis in



Figure 2.1: In this figure the 3DR Iris+ is shown with custom made breakdown triangle on a custom made control module attached to the main frame of the drone.

a 3 dimensional room. In a plot the three axis can be described as x, y and z axis. The positional control of a **UAV** system is usually converted to pitch, yaw and roll which are described in radians. Small **UAVs** usually have two control mechanisms: remote control (or radio control) and autopilot control mode. Controlling a drone via remote control means a human pilot sending radio signals through a remote control by changing the states of several throttles. Whereas autopilot means the flight controller of a **UAV** can keep the vehicle in a desired state, for example some kind of automatic positioning system [18]. Hence the Breakdown drone only flies by autopilot the remote controlled option will not be described any further.

2.1.1 Autopilot System

An autopilot consists of hardware and software to be able to guide a **UAV**. Its main purpose is to pilot the **UAV** during take-off, trajectory following and landing. The autopilot is a key component in the flight control system, because it has to communicate with a ground station (in the Breakdown Drone case the Raspberry Pi 3) which is responsible for telling the autopilot which mode to use. At the same time it receives broadcasts from the **GPS** satellites for position determination and therefore it sends control signals to all motors (and servos in a aircraft UAV system) [18]. The Pixhawk is a autopilot not only designed for quadcopters but also for helicopters, cars, boats or any other autonomous vehicle.

It is build by **3D Robotics (3DR)** and developed as an open-source project by many engineers around the globe. Following build-in modules get the whole system running:

- **Processor**

- 32-bit ARM Cortex M4 core with FPU
- 168 MHz/256 KB RAM/2 MB Flash
- 32-bit failsafe co-processor

- **Sensors**

- MPU6000 as main accel and gyro
- ST Micro 16-bit gyroscope
- ST Micro 14-bit accelerometer/compass (magnetometer)
- MEAS barometer

2.1.2 GPS Module

The so called GPS-BR-0008 module allows the flight controller to connect to the **GPS** satellites for precise navigation outdoors. It is connected directly to the Pixhawk controller.

GPS is a constellation of 27 satellites (24 of them operating and 3 in case one fails) that send radio signals that receivers can capture. Initially **GPS** was developed as a way of navigation for and from the US military, but soon they decided to make it available for everyone. Every single one of these 760 kg heavy satellites circles our planet at about a height of 20200 km. Their orbits are arranged in a way so that at any given time everywhere on earth there are at least 4 satellites in reach. The **GPS** modules job is it to get a connection to at least four of these satellites and measure the distance to each one of these. Now it is possible to determine a exact location using the mathematical method of trilateration [19]. To simplify trilateration a little bit one can look at a two dimensional projection of the earth as in figure 2.3 on the left. However with only three satellites a receiver can only calculate the position of itself, which is exactly at the point of intersection of the three radians from the satellites. With a fourth satellite it is possible to determine the height of the receiver as well. Adding additional satellites to the system allows a much more accurate determination of location.

2.1.3 Telemetry Module

The 433MHz Telemetry module is used for wireless data exchange between the aircraft and any ground controller. A ground controller can either be the remote control or any PC or Android Phone/Tablet.

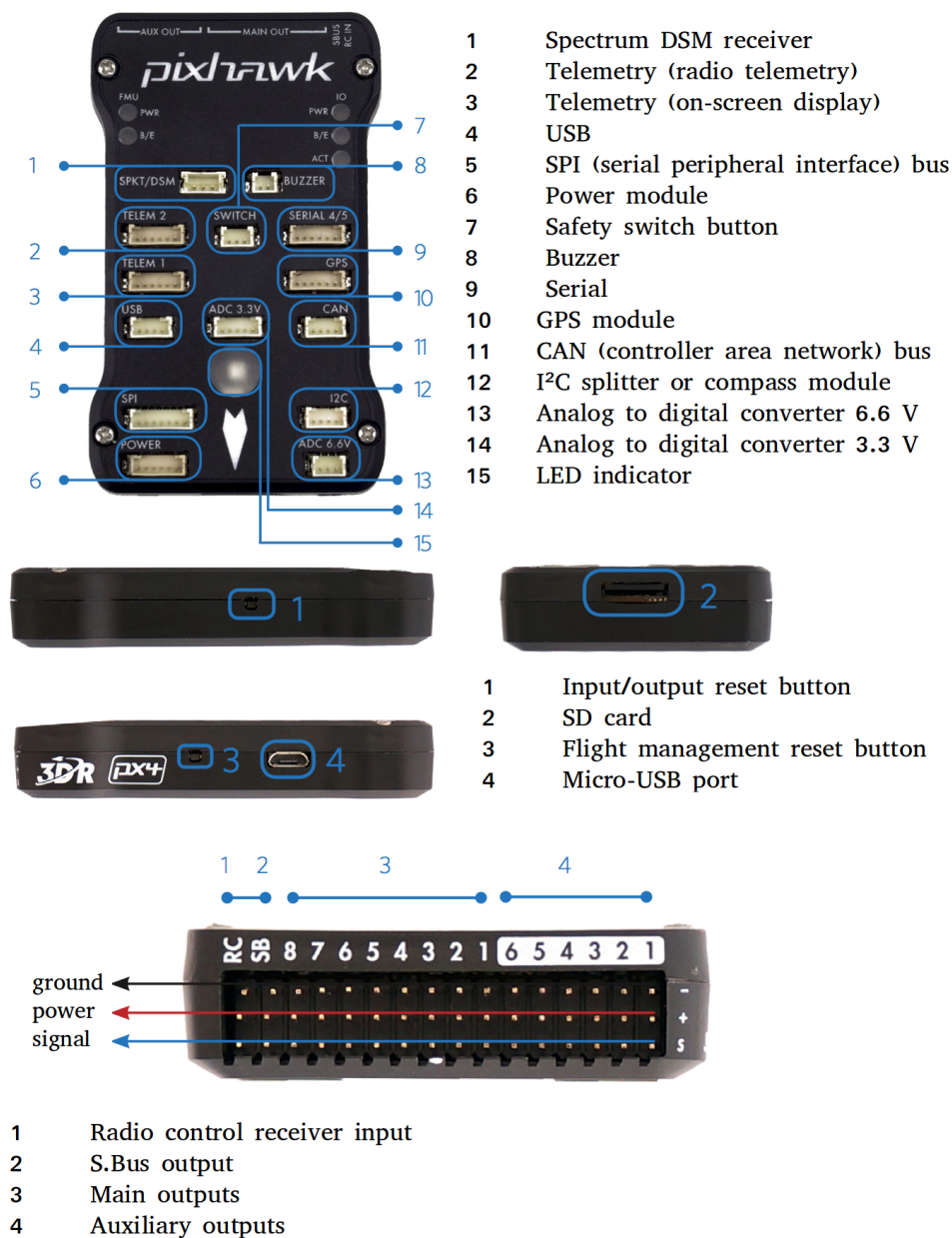


Figure 2.2: In this figure the main I/O ports of the Pixhawk autopilot is shown.

2.1.4 Motors

Standard motors for the Iris+ are the T-Motor MN2213. These allow the drone to carry an extra weigh of about 500 g. With a left-hand threaded attachment on two of the four motors and a right-hand threaded attachment on the other two motors it allows the propellers to self-tighten.

2.1.5 Battery Pack

Mainly used for power supply is the iris+ battery pack. It is a lithium polymer battery with a capacity of 5100 mAh. This capacity allows the drone to fly for up to 20 minutes

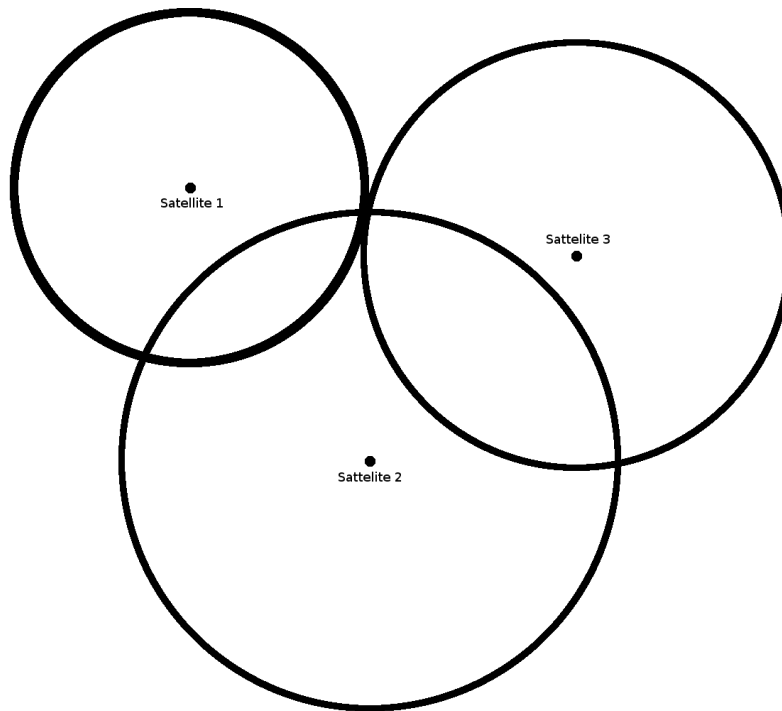


Figure 2.3: This two dimensional figure shows how three satellites are required be able to tell the exact location of a device which is represented by the intersection fo the three radii.

with one charge. The XT60 connector transmits the power to the flight controller and the motors.

2.2 Custom Build Control Module

The control module contains all the important parts to actually deliver and put down the breakdown triangle to its target location.

2.2.1 Raspberry Pi 3

The Raspberry Pi 3 is a one-board computer developed by the Raspberry Pi Foundation. The most remarkable property about the Pi is its size, because it is only as small as a credit card, but it has a quad core processor built-in.

A brief hardware overview:

- **SoC:** Broadcom BCM2837
- **CPU:** 4x ARM Cortex-A53, 1.2GHz
- **GPU:** Broadcom VideoCore IV
- **RAM:** 1GB LPDDR2 (900MHz)
- **Networking:** 100 Mbit Ethernet, 2.4GHz 802.11n wireless

- **General-purpose input/output (GPIO):** 40-pin header

On the Breakdown Drone, the Pi is the processing unit to relieve the flight controller. The unit communicates with the flight controller via USB to get all the data it needs to calculate the approximate path the drone has to take in order to reach its destination. The Raspberry Pi is connected via serial bus to a camera module used for image processing. The implemented algorithm are able to detect the end of the street. The main task is to release the breakdown triangle at the final point, therefore a motor controlled by the Pi is needed as well. The whole system is powered by an 5600 mAh powerbank.

2.2.2 PiCam

The Pi Camera Module v2 is a small and light sensor that is attached to the Raspberry Pi. It only weighs 3 g and has a still resolution of 8 Megapixels. Additionally to its photo resolution the camera is capable of producing videos in 1080p30, 720p60 or 640 x 480p60/90. The sensor has a horizontal field of view of about 62.2 degrees and 48.8 degrees for its vertical field of view. It has a fixed focal length of 3.04 mm, a focal ratio of 2.0 and is not able to automatically focus objects. [20]

2.2.3 Motor to Release the Triangle

The 40 **GPIO** pins on the Model 3 of the Raspberry pi are used, as the name already indicates for reading input to connected sensors or outputting data to different electrical parts such as LEDs, motors etc. These 40 pins get divided into two 5V power inputs, two 3.3V power inputs, eight grounded pins and a total of 28 pins used for in- and output. Some of the IO pins have a special purpose, but are not needed to control a simple DC motor. To control a DC (direct current) motor a few parts are needed [21].

- The Raspberry Pi itself
- a breadboard
- a motor driver chip like the L293D
- cables to connect everything together
- a DC motor
- and a holder for four AA batteries

The L293D is a motor driver which allows a controlled movement of any DC motor in both directions (clockwise and counterclockwise). Its IC (integrated circuit) supports 16-pins and therefore 2 DC motors to be controlled simultaneously. Built-in the L293D is the concept of a H-bridge which allows a voltage applied to the motor in one direction or the other. With roughly 20mA being the maximum of power provided by the main processor, another source of power supply is needed, that's where the batteries come in place. To actually release the triangle, a SG-5010 Servo is used mainly for its light weight and high accuracy for its low cost. If all the parts are put together correctly like in Figure 2.4 the servo can be controlled by a simple python script.

2.3 Breakdown Triangle

The triangle had to be build completely from scratch because commercially available breakdown triangles are not only impractical to attach to the drone but also way too heavy. In

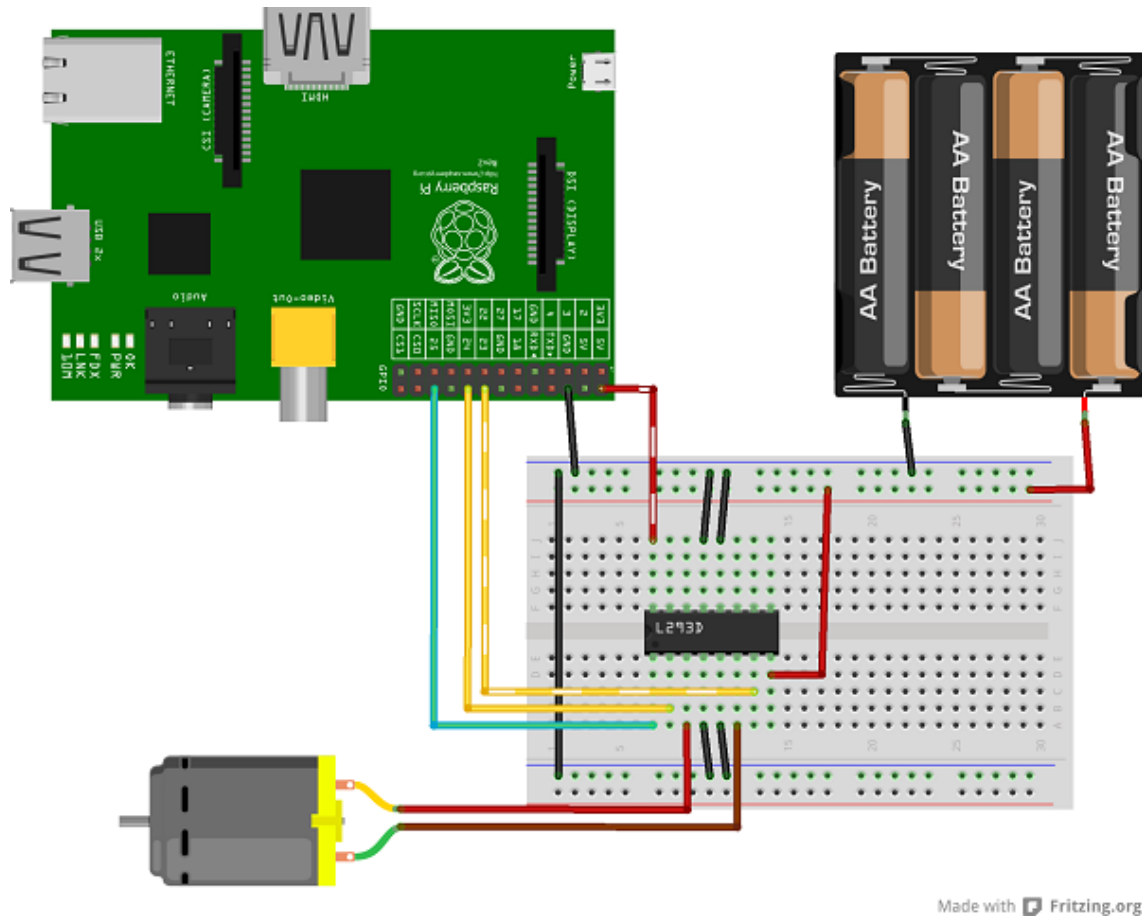


Figure 2.4: Assembly of the circuit used to power the motor [21].

the figure above there are several components highlighted:

The **reflective area** is stuck to a very light weight but stable synthetic material, it is used to warn drivers who approach the crash scene.

Attached to the reflective triangle is a **PVC Pipe**. Its purpose is not only to somehow pin the triangle to the drone but at the same time keep a safe distance from the triangle to the propellers. On the very top of the PVC Pipe is a loop of wire to fix the whole construction to the vehicle.

Last but not least the **wooden part** inside the PVC Pipe. It is stuck through the bottom of the triangle and glued to the PVC Pipe so neither the triangle nor the wood is able to wobble around. A nice side effect of this part also is the fact that it is lowering the center of mass by a lot so when the drone flies away from the spot the triangle gets put down, the triangle doesn't move from its intended place. In fact it doesn't move at all.

This construction has only a weight of about **70 g** which is a very significant improvement over a triangle found in a store with a weight of at least 2 kg. While being so light it also stands heavy winds coming from the drone.

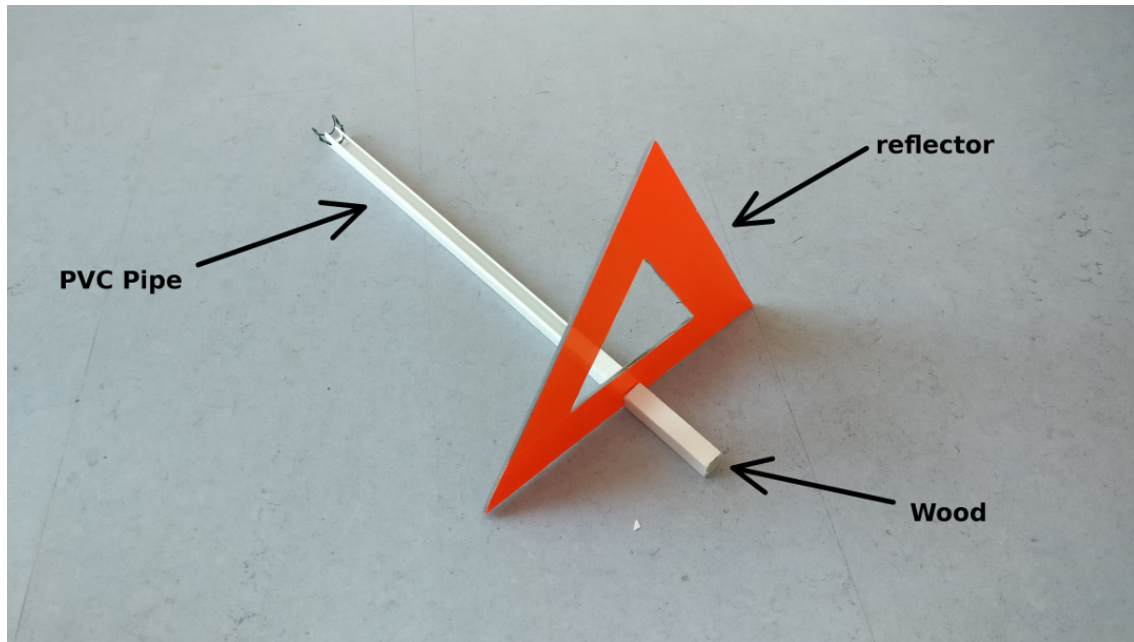


Figure 2.5: The self build triangle used for the Breakdown Drone.

2.3.1 Release Mechanism

The release mechanism is a DC motor that holds down the breakdown triangle with a metal bar exactly at the indentation of the PVC pipe.

2.4 Communication between Drone and Raspberry

The USB (Universal Serial Bus) communication between the Pixhawk controller and the Raspberry Pi gets controlled by a so called Host-Controller, which in this case is build onto the board of the Raspberry Pi. Only the host is allowed to read or send data to or from the other device.

2.4.1 Architecture

The components of a USB host controller seen in figure 2.6:

- Client Driver Software: is a software module used to exchange data between the operating system and the USB device.
- USB Driver: is a system software Bus Driver that is able to abstract the details of a particular host controller driver for a specific operating system.
- Host Controller Driver: provides the software layer between the host controller hardware and the USB driver.
- Host Controller: is the specific hardware implementation. One for the Full- (USB 2.0) and one for the Low-speed (USB 1.1) host controller.
- USB Device: hardware item that performs a specific task for the user.

With two different host controller and their cohesive drivers it is possible to not only allow the usage of USB 2.0 devices but also the older USB 1.1 standard, therefore USB is able to fully support backwards compatibility.

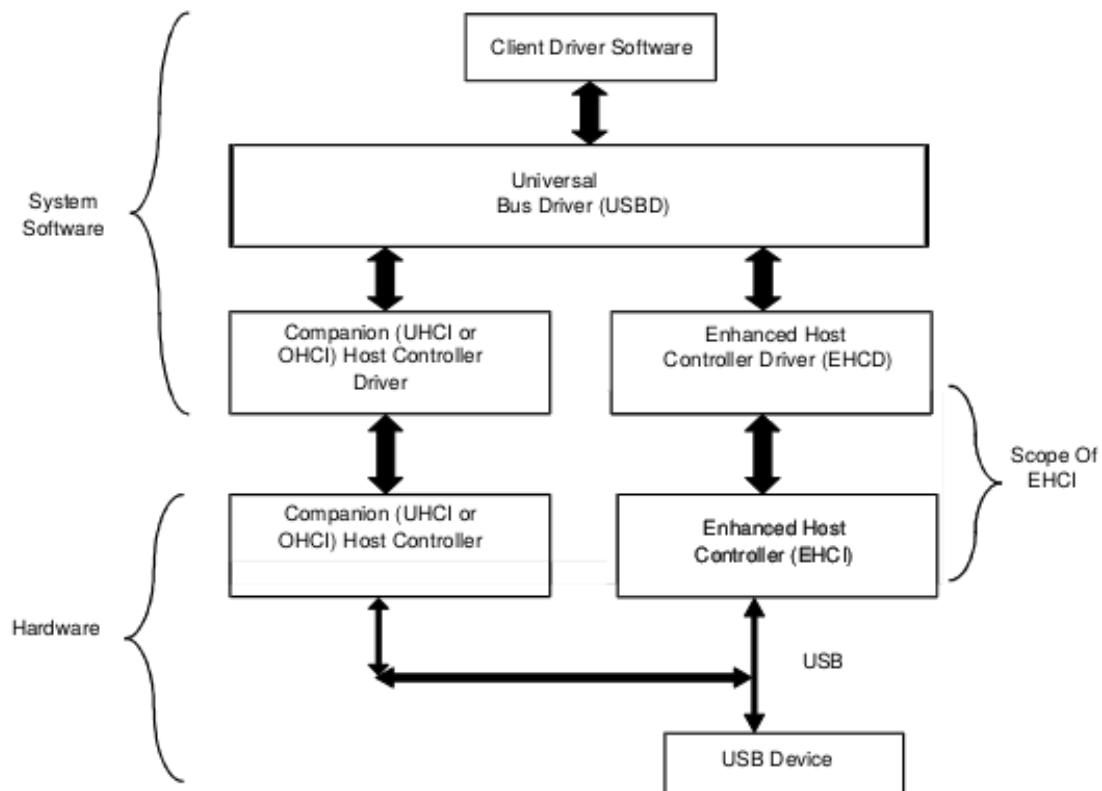


Figure 2.6: Illustration of a Block diagram in a host system.[22]

2.4.2 Host Controller

Internally the host controller addresses any connected device with a 7 bit long identifier. Therefore it has a theoretic maximum of 127 devices connected. If there is a new device detected on a port the controller enables this port and sends the connected device a reset. Thereby the device gets address 0 and the controller is then able to assign a unique address to the device.

2.5 Alternative Hardware

2.5.1 Drone

It is possible to completely build a drone from scratch. However a lot of time and effort will be put into building and programming a self made drone. If someone still decides to do so he will have to do a lot of research beforehand in order to not miss out on any of the drones features needed. Some crucial factors are for example the overall weight of the drone, whether or not the drone will be able to carry enough additional weight if so desired and last but not least the drones stability during the flight has to be given at any time. Furthermore if a builder does not intend to program autonomous missions, a way of controlling the drone will also be required. The main options for steering a UAV are either by a smartphone app or via remote control.

There are also possibilities to buy already assembled drones. The offers of the different companies vary greatly. One of the most well known companies in commercial drone industry is dji (Dà-Jiāng Innovations Science and Technology Co.). Their most recent drones are the so called “mavic pro” and the “phantom 4”. Advantages of these two drones are their high build quality, very accurate sensors and a very good customer service. But their disadvantages are not to be ignored either. The dji drones are not fully programmable and their price is not really low either with a price of 1199€ for the mavic pro and 1699€ for the phantom 4 (price from the dji online store on February 2017). Dji also released a fully programmable drone, the so called “Matrice 100” but it is coming in at a price of 3.599€ (price from the dji online store in February 2017). Since the 3DR iris+ came in at a lower price of xxx€ and is fully open-source it was the obvious choice for the Breakdown Drone with the only problem being that the iris+ has been discontinued [23].

2.5.2 Raspberry Pi

There are certainly countless other single-boarded computers on the market than the Raspberry Pi series, this section is going to discuss why the Raspberry Pi 3 was used over its competitors.

The probably most known competitor is the Banana Pi M64. It is roughly the same size as the Raspberry and has in most parts the same I/O (Input/Output) devices built-in. Its main differences in terms of specifications are the 2GB of RAM compared to the 1GB of the Raspberry Pi and the Banana Pi has an internal storage capacity of 8GB. In terms of I/O ports it is missing some ports like the **Cameral Serial Interface (CSI)** which means for the Banana Pi a USB Camera would be necessary, however if a USB camera is used it has to be taken into consideration that the Banana Pi only has 2 USB ports in total compared to the 4 ports on the Raspberry Pi [24].

Another very well known option is the Cubieboard 4. It scores with its 2GHz octa core CPU, 2GB of RAM and 8GB of internal flash storage compared to the specifications of the Raspberry Pi 3. Just like the Banana Pi the Cubieboard also lacks a **CSI** on board, which has to be coped with an USB camera as well. But not only the **CSI** is missing but the Cubieboard is also missing **GPIO** pins entirely, therefore another way of controlling a motor for the breakdown triangle is needed [25].

On paper all the other options of the Raspberry Pi might seem nice, but what all of them are missing is the huge community behind the Raspberry Pi. There are numerous people and open source projects that are able to help a developer developing a new project or technology. This is the reason the Breakdown Drone uses a Raspberry Pi 3 over all its competitors.

Chapter 3

Software (Fabian)

3.1 Programming Language

Python2.7 was used as the primary programming language for the entire project. Python is a multi purpose programming language which has great built-in support for collections. It's an interpreted language and is supported on many different platforms (Windows, Linux, OSX, BSD, Solaris, etc.). With the language being weakly typed and having a lot of syntactic sugar, it makes the code easier to read and needs generally less code than its competitors like Java or C++ for the same operation.

3.1.1 Background

Python was created in the year 1989 by Guido van Rossum, who named the programming language after the British comedian Monty Python. It was originally only a hobby project that should create a scripting language that could replace the at its time dominant programming language, ABC. Due to its dynamic nature it got quite popular and in 2000, Python 2.0 was released.[26]

Eight years later in 2008 Python 3.0 was released and meant to replace the old 2.x versions. Since the new major version is not backwards compatible, many programmers stuck with 2.x. Therefore the 2.x branch was still updated and in 2010 the final version 2.7 was released. Python2.7 is still frequently used and will be officially supported until 2020.[26]

3.1.2 Why Python2.7?

Even with Python3 being released in December 2008 many libraries have never been ported from 2.x to 3.x, making them incompatible and compelling developers to use the legacy version in order to have access to essential external modules. With the Dronekit-API being a KO criterion for this project and it only being available in Python2, using Python3 was never considerable.

3.2 MAVLink (Florian)

MAVLink (Micro Air Vehicle Communication Protocol) is a header-only message marshaling library for micro air vehicles. It was first released by Lorenz Meier in 2009.

3.2.1 Message System

The MAVLink Protocol is designed to exchange data between a vehicle and a GCS (Ground Control Station) via a serial communication channel (USB). The Raspberry Pi of the Breakdown Drone works as a GCS.

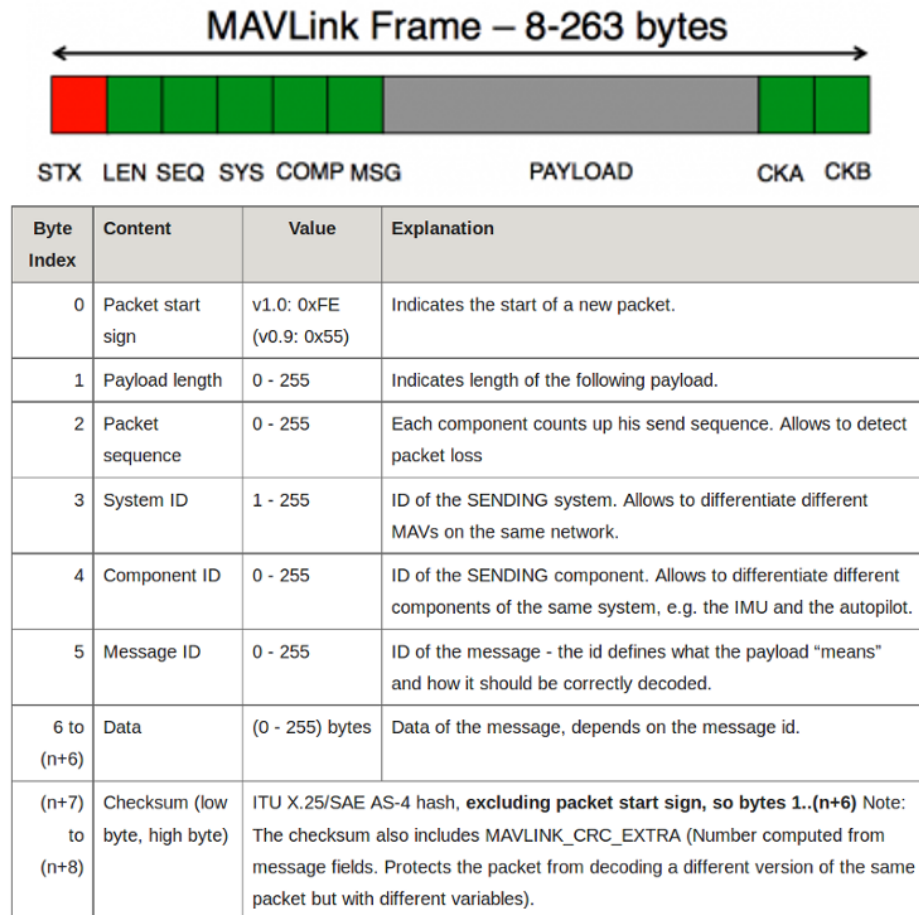


Figure 3.1: This figure describes the structure of all MAVLink messages [27].

An example message is sent by the GCS to the UAV. Immediately after the message was sent the GCS starts a timer. As soon as the UAV gets the message from the GCS it executes it and responses with an acknowledgement message. The GCS waits for a certain amount of time for the acknowledgement message to arrive, if it doesn't within this timespan, the initial message will be sent again. An autopilot system (like the Pixhawk autopilot) will take the frame (shown in 3.1) apart and execute its payload. Since the Acknowledgement package will not have to contain any data it will only have a length of 8 bytes. A normal packages size can range anywhere from 9 to 263 bytes per package.

3.3 DroneKit (Florian)

DroneKit is a open-source and community driven library that allows developers to be able to create and run applications on various different so called companion computers. It takes advantage of a low latency connection (USB) between the companion computer and the

UAV to enhance the system, by adding greater intelligence to the vehicle or enable the usage of computationally intensive tasks such as the calculation of the flight path (seen in chapter 4) or the processing of computer vision (seen in chapter 5). Initially there were three different methods of developing with the DroneKit API - via Cloud, Java or Python. However DroneKit-Cloud has been shut down on March 31st, 2016 because the developers do not have the resources required to further develop their product anymore. DroneKit-Java is mainly used to be able to develop custom Android applications. The Python API has found great usability in systems with a second dedicated computer for providing the vehicle with the flight information it needs. [28]

DroneKit uses the MAVLink Protocol (Micro Air Vehicle Communication Protocol) to communicate between the companion computer and the drone. (as previously described in 3.2)

3.3.1 API Features

The main features the API is able to provide to a developer are:

- connection from the companion computer to the vehicle.
- the ability to get and set vehicle states/telemetry- and parameter information.
- monitoring of vehicle state changes.
- a way to guide a vehicle to a specified location.
- sending custom messages to control the vehicle movement or other hardware (gimbal).
- create and manage various waypoint missions.
- alternation of RC channel settings.

3.3.2 Connection to a Vehicle

The connect function (seen in listing 3.1) returns a vehicle object that can will further be used from the companion computer to communicate with the vehicle. Its Parameters are: `'/dev/ttyACM0'` which describes the serial interface the USB cable is connected to, `wait_ready` is set to true so the program freezes until the API has established a safe connection to the vehicle, and the baud rate describes symbols or pulses get transferred per second, this information is important so the autopilot system knows at which rate it has to refresh its input stream in order to receive the data correctly.

Listing 3.1: Code snippet for connecting to a vehicle that is linked via USB cable.

```
1 vehicle = connect('/dev/ttyACM0', wait_ready=True, baud=57600)
```

3.3.3 Vehicle Modes

The 3DR Iris+ comes with various different vehicle mode out of the box. The most important ones are:

- RTL (return to launch)
- POSHOLD (holds current position)
- LAND
- STABILIZE
- AUTO
- GUIDED
- ALT_HOLD (holds current altitude)

- BRAKE
- LOITER
- SPORT (enables higher speeds at the loss of altitude)

Every single one of these modes are set- and gettable by calling the vehicles “mode” attribute (seen in listing 3.2).

Listing 3.2: Sets the vehicles mode to “GUIDED”. This can be exchanged for any of the other modes mentioned above.

```
1 #setting the vehicle mode to GUIDED
2 vehicle.mode = VehicleMode("GUIDED")
3 #printing current vehicle mode
4 print vehicle.mode.name
```

3.3.4 Vehicle Attributes

A lot of the vehicles attributes can be accessed via the vehicle object generated in 3.3.2.

Listing 3.3: Shows the code required to access the most important vehicle attributes.

```
1 print "Autopilot Firmware version: %s" % vehicle.version
2 print "Autopilot capabilities (supports ftp): %s" % vehicle.capabilities.ftp
3 print "Global Location: %s" % vehicle.location.global_frame
4 print "Global Location (relative altitude): %s" % vehicle.location.
  ↳ global_relative_frame
5 print "Local Location: %s" % vehicle.location.local_frame
6 print "Attitude: %s" % vehicle.attitude
7 print "Velocity: %s" % vehicle.velocity
8 print "GPS: %s" % vehicle.gps_0
9 print "Groundspeed: %s" % vehicle.groundspeed
10 print "Airspeed: %s" % vehicle.airspeed
11 print "Gimbal status: %s" % vehicle.gimbal
12 print "Battery: %s" % vehicle.battery
13 print "EKF OK?: %s" % vehicle.ekf_ok
14 print "Last Heartbeat: %s" % vehicle.last_heartbeat
15 print "Rangefinder: %s" % vehicle.rangefinder
16 print "Rangefinder distance: %s" % vehicle.rangefinder.distance
17 print "Rangefinder voltage: %s" % vehicle.rangefinder.voltage
18 print "Heading: %s" % vehicle.heading
19 print "Is Armable?: %s" % vehicle.is_armable
20 print "System status: %s" % vehicle.system_status.state
21 print "Mode: %s" % vehicle.mode.name # settable
22 print "Armed: %s" % vehicle.armed # settable
```

3.3.5 Vehicle Movement

The probably most important part of the DroneKit API is the movement of the vehicle itself.

Arm

On a drone the vehicle will first have to start spinning the motors with the propellers attached to them (also called “arming the vehicle”). This can be done by trying to arm the vehicle by sending the command to arm as long as it hasn’t armed yet.

Listing 3.4: Shows the code required to wait for a vehicle to be armable and then arms the vehicle


```

1 print "Basic pre-arm checks"
2 #wait for the autopilot to be ready.
3 while not vehicle.is_armable:
4     print " Waiting for vehicle to initialise..."
5     sleep(1)
6
7 print "Arming motors"
8 # Copter should arm in GUIDED mode
9 vehicle.mode = VehicleMode("GUIDED")
10
11 # Confirm vehicle armed before attempting to take off
12 while not vehicle.armed:
13     vehicle.armed = True
14     print " Waiting for arming..."
15     sleep(1)

```

Takeoff

After the vehicle is armed it is possible to take off to a certain altitude. The program waits for the UAV to reach a certain altitude to eliminate a possible error in the sensors measurement an inaccuracy of 5% is included.

Listing 3.5: Takes off and waits for the vehicle to have reached the target altitude with an error of 5%.

```

1 print "Taking off!"
2 vehicle.simple_takeoff(10)
3
4 while True:
5     #prints current altitude.
6     print " Altitude: ", vehicle.location.global_relative_frame.alt
7     #stops current loop if the target altitude is reached.
8
9     if vehicle.location.global_relative_frame.alt>=10*0.95:
10         print "Reached target altitude"
11         break
12     sleep(1)

```

In Air Movement

After arm and takeoff the vehicle should already be in mode “GUIDED“. Then the speed is set to a realistic velocity always measured in $\frac{m}{s}$. Setting the speed doesn’t do anything until a function is called that actually changes the position of the vehicle. Such a function can either be the approach of a coordinate or a specified movement in either the pitch, yaw or roll axis to the relative frame of the UAV. For the flight to a coordinate a LocationGlobal object has to be created it concludes the latitude, longitude and altitude of the specified node.

Listing 3.6: Code snippet for flying to a coordinate.

```

1 vehicle.airspeed = 3
2 loc = LocationGlobal(latitude, longitude, altitude)
3 vehicle.simple_goto(loc) #initiates the movement in air.
4 #calculates the remaining distance from current position to node
5 remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
6 #loops as long as the distance to the node is 1 meter or less
7 while not remainingDistance <= 1:
8     print "Distance to target: ", remainingDistance

```

```

9     sleep(0.1)
10    remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
11    print "reached node!"

```

The second method of moving the vehicle in air is the relative movement of the frame. There are certain methods of doing that but some of them are not yet implemented in the DroneKit API. Most commonly movement gets described as velocity on a specific axis (pitch, yaw, roll). Therefore the change of position can be calculated via vectors.

Listing 3.7: Code snippet for flying to a coordinate.

```

1 msg = vehicle.message_factory.set_position_target_local_ned_encode(
2     0,          # time boot ms (not used)
3     0, 0,      # target system, target component
4     mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, # frame
5     0b0000111111000111, # type mask (only speeds enabled)
6     0, 0, 0,   # x, y, z positions (not used)
7     velocity_x, velocity_y, velocity_z, # x, y, z velocity in m/s
8     0, 0, 0,   # x, y, z acceleration (not supported yet, ignored in GCS Mavlink)
9     0, 0)      # yaw, yaw rate (not supported yet, ignored in GCS Mavlink)
10
11 # send command to vehicle on 1 Hz cycle
12 for x in range(0,duration):
13     print x
14     vehicle.send_mavlink(msg)
15     sleep(1)

```

3.4 Communication

As the Raspberry Pi is most of the time not in range of a wireless network, alternatives are necessary to communicate with the drone during flight.

As a solution the integrated WiFi module of the Raspberry Pi 3 is turned into an Access Point and a simple DHCP server is installed for dynamic ip allocation, using *hostapd* and *isc-dhcp-server* respectively.

This allows remotely controlling the Raspberry Pi via SSH, which makes executing new programs easier and debugging more comfortable.

3.5 Camera Stream

Furthermore, a simple webstream application is implemented that serves the processed images of the camera in real time. It's a python script that utilizes Flask [29] as a webserver. With this it can be monitored how the image processing algorithm interprets the images during a flight. Flask helps finding faults in the street detection program. Since it acts like a usual website, the stream can be accessed from *any* device connected to the Raspberrys Wifi.

Chapter 4

Navigation (Fabian)

One of the most challenging tasks when it comes to developing software for autonomous vehicles, is to find an optimal path on which your vehicle will move. Therefore the developer has to decide which input sensors, cameras or other utilities will be used for a robust identification of obstacles. When dealing with self-driving cars or other **Unmanned Ground Vehicles (UGH)** there's usually more than enough space for integrating such devices. Drones on the other hand restrict the developer by its physical limitations. The quadcopter used in this work has a maximum payload of about 500 g, which makes it hard to use heavy 3D depth cameras. Furthermore, the computational resources on board a drone are, in comparison to usual desktop computers, meager, making traditionally used mapping algorithms unusable, as they are computationally expensive.

For navigating outdoor, **GPS** is a reliable method and it's used to stabilize and assist the drone used in this project.

4.1 Global Positioning System (Fabian)

GPS is a satellite-based navigation system. **GPS** can be used in all weather conditions as long as the receiver is in sight of four or more satellites [30, pp.1–3].

World Geodetic System 1984 (WGS84) is the coordinate system used in conjunction with **GPS** for position determination. **WGS84** maps the **GPS** data to a *latitude* and a *longitude*; the coordinates are measured in degrees.

Latitude specifies the north-south coordinate of the position. The Equator's position has a latitude of 0° , the South Pole has a latitude of -90° and the North Pole has a latitude of $+90^\circ$. The distance between one degree latitude is roughly 111 km.

Longitude specifies the west-east coordinate of the position. The 0° longitude passes through the prime meridian (Greenwich) [31]. Longitudes range from -180° to $+180^\circ$. Negative longitudes are west of the prime meridian and positive longitudes are east of it. The distance between one degree longitude depends on the latitude. The closer the latitude is to the poles, the smaller the distance between the longitudes gets.

4.1.1 Calculating the Distance between two GPS Coordinates

The distance in meters between two coordinates is needed multiple times, for example to determine when the **UAV** is far enough away from the starting point.

There are various approaches for calculating the distance between two points. To determine the best method for the given use-case, a set of approximations were compared:

Equirectangular Approximation

Equirectangular means that the longitudes and latitudes are mapped as vertical straight lines and horizontal straight lines, respectively. Then the Pythagorean theorem can be used to calculate the distance between two points:

$$\begin{aligned}x &= \Delta\lambda \cdot \cos \bar{\varphi} \\y &= \Delta\varphi \\d &= R \cdot \sqrt{x^2 + y^2}\end{aligned}$$

where:

- λ ... longitudes in radians
- φ ... latitudes in radians
- $\bar{\varphi}$... arithmetic mean of the latitudes
- R ... Earth radius $\approx 6.371 \cdot 10^6 m$
- d ... distance between the two points in meters

The equirectangular approximation doesn't consider the ellipsoidal shape of the earth and therefore is only accurate for $d \ll R$.

Haversine Formula

A more precise calculation can be achieved by assuming a spherical Earth. The haversine formula [32, p.159]

$$\begin{aligned}a &= \sin^2(\Delta\varphi/2) + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \sin^2(\Delta\lambda/2) \\c &= 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a}) \\d &= R \cdot c\end{aligned}$$

where:

- a ... an intermediate result
- c ... the angle between the two points in radians

is a quite good approximation as long as the latitudes are not close to the poles.

Vincenty Formulae

If the Earth is treated as an ellipsoid, an even better precision can be achieved. The Vincenty formulae is such an approximation and has a precision of about 0.5 mm [33, pp.88–93]. Computationally, it's a lot more extensive compared to the previous approaches.

4.1.2 Offsetting a GPS Coordinate

Moving a coordinate in a specific cardinal direction relative for a certain amount of meters is needed to determine GPS coordinates relative to the UAV. For calculation an approach similar to the equirectangular approximation is used:

$$\begin{aligned}\varphi &= \frac{d}{R} \cdot \cos \theta + \varphi_0 \\ \lambda &= \frac{d}{R} \cdot \frac{\sin \theta}{\cos \varphi_0} + \lambda_0\end{aligned}$$

where:

- φ_0 ... latitude of the original coordinate
- λ_0 ... longitude of the original coordinate
- θ ... cardinal direction
- d ... distance in meters

Same as the compass rose, the cardinal direction is split into 360° clockwise, starting north with 0°.

4.2 Usage of Map Data

An offline map is used to identify the initial position and generate a path that leads to the final position, where the breakdown triangle should be placed. This path consists of a list of **GPS** coordinates which all lie within the bounds of the street. It is assumed that the space above the street is empty regarding any obstacles which could endanger the vehicle.

4.2.1 OpenStreetMap

Open Street Map (OSM) is a crowd-sourced project which provides volunteered geographic information of the whole world. It's data is published under the Open Database License, granting free access to anyone.

Why OSM?

Popular map services (*e.g.*, *Google Maps*) prohibit mass downloading of their data or using it in a derived work, but **OSM** allows developers to store their geographic data and modify it. Since the Raspberry Pi is not connected to the Internet during flight, a offline geographic data is mandatory.

Data Model

A **OSM** data file logically consists of three different element types which try to model the physical world:

- **Node:** This element represents a physical point in space. It's defined by its latitude and longitude.
- **Way:** An ordered list of nodes. A way could be a street, river or any other structure that can be represented as a polygonal chain or as a closed polygon (*e.g.*, *a parks border*).
- **Relation:** Relations model the geographical relationship between two objects (*e.g.*, *all bus stop nodes of a bus line*). There aren't any relations that hold relevant data for highways, so this element type is neglected in this study.

Additionally every element may have multiple tags which describe the purpose and type of the according element. A tag consists of a key and a value, which are presented in the form of "key"="value".

The key is used to describe a feature and the value gives detailed information about the feature. The most important key that can be defined for ways is the *highway* key. It specifies that the corresponding way is some kind of road, street or path. The value for this key names the type of the street. For example the tag "highway"="motorway" would describe that the according road is a motorway, which implies that cars may use this way.

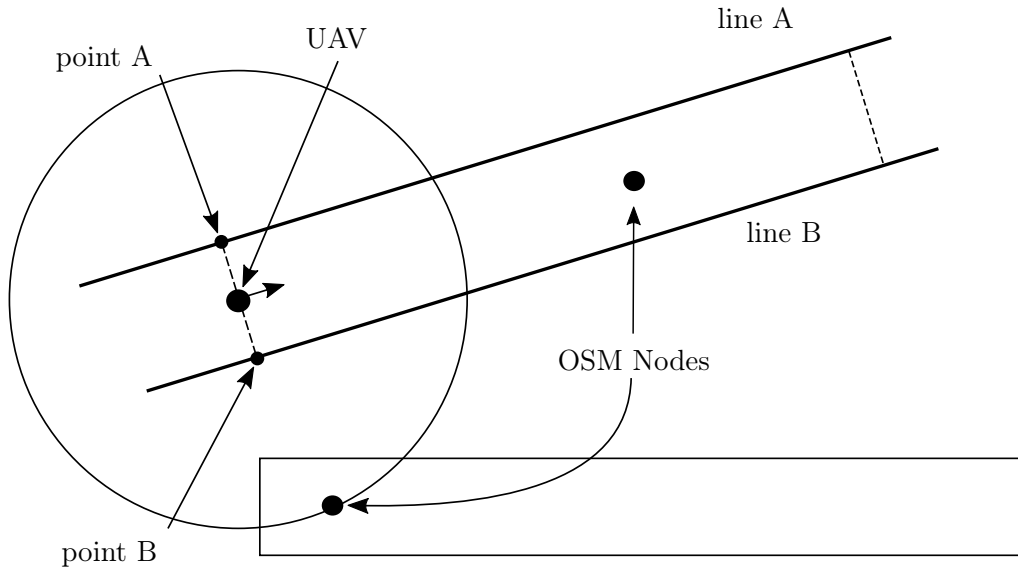


Figure 4.1: Illustrating the problematic of reverse geocoding using a circular lookup. The lines A and B approximate the bounds of the street and together with the dotted lines they build the rectangle used for filtering the OSM Nodes

This information is useful if one wants to implement different behavior for certain types of streets.

Data Format

A map of the whole world can be downloaded at the [OSM](#) website [34]. The map data is available in two formats: human readable [Extensible Markup Language \(XML\)](#) and [Protocolbuffer Binary Format \(PBF\)](#). The PBF is a compressed alternative to the XML format and needs far less disk space. The map data of Austria is about 500 MB large using PBF versus 800 MB using a bzip2 compressed XML format. Since there is little storage available, the PBF map data is used. The raw map data still contains relatively much irrelevant data and therefore only the nodes and ways that represent actual streets are extracted from the original data. For stripping the irrelevant data, the open sourced command-line tool *Osmosis* [35] is used. The filtered map of Austria is then only about 70 MB large, which is a great improvement.

4.2.2 Reverse Geocoding

Geocoding means converting a textual description of a location (the description could be the name of the country, city and street) to a coordinate. Reverse geocoding is the opposite process that gets a readable location using latitude and longitude as input. In most applications figuring out the country or city is sufficient and relatively easy. Using OpenStreetMap that would mean comparing all nodes with the given position and determining the nearest node. The node would then contain information about the country and city associated to it. Finding the nearest street is more difficult because it might be that a node of another street is closer to the position than the next node of the actual street as shown in figure 4.1. Therefore an algorithm was developed that doesn't use a circular lookup but restricts the set of possible nodes to a certain direction.

Approximating the Bounds of the Street

Instead of searching in a circle around the initial position, a rectangle is modeled that should approximate the street. To determine if the node is inside the rectangle shown in 4.1 following statements have to be true:

1. The coordinates of the node have to be between the lines A and B
2. The node is in front of the UAV and not behind
3. The nodes distance does not exceed a certain value

Defining the Lines A and B

The lines A and B are defined as a linear equation derived from the formula used for offsetting a GPS coordinate:

$$\varphi \cdot \sin \theta - \lambda \cdot \cos \theta \cdot \cos \varphi = \varphi_0 \cdot \sin \theta - \lambda_0 \cdot \cos \theta \cdot \cos \varphi_0$$

where:

- (φ_0, λ_0) ... starting point of the line
- θ ... the orientation of the line
- (φ, λ) ... any point on the line

Based on this equation the lines are calculated as in the following code snippet:

```
1 pointA = move_coord(self.coord,self.tolerance,self.angle + 90.0)
2 pointB = move_coord(self.coord,self.tolerance,self.angle - 90.0)
3
4 lineA = create_line(pointA,self.angle)
5 lineB = create_line(pointB,self.angle)
6
7 upper_bound = max(lineA,lineB)
8 lower_bound = min(lineA,lineB)
```

Description of variables and functions:

- `self.coord`: Tuple that holds the coordinate of the UAV measured in degrees.
- `self.angle`: Cardinal direction of the UAV measured in degrees.
- `self.tolerance`: Distance from the UAV to point A or B in meters
- `move_coord(coordinate,distance,cardinal_direction)`: offsets a coordinate as described in 4.1.2.
- `create_line(coordinate,cardinal_direction)`: calculates the right side of the former equation returning a float value.

With the bounds set up it can be checked if a coordinate X is between the two lines using the expression `lower_bound <= create_line(X,self.angle) <= upper_bound`.

Chapter 5

Computer Vision (Fabian)

Computer vision refers to extract semantical information from an image using a computer program. It tries to identify physical objects just like a human would do. It's a large field that can be used for various tasks like face detection or counting how many people are in a given image. In this work computer vision is used to identify the outer road marking of a street so that the **UAV** can navigate accordingly. [36, pp.2–3]

5.1 Misconception of Computer Vision

Because Computer Vision tries to mimic the human eye in certain aspects, it is often understood as an attempt to completely imitate human vision. This is indeed a misunderstanding since such implementations have to be limited to certain tasks, thus restrict the amount of error possible. Identifying a car in an image is quite intuitive to humans as years of experience have influenced ones visual perception, but a computer only sees a grid of numbers in those images, so one assigned task is already hard enough. [36, pp.1–4]

There is another misconception about the uses of computer vision. Most people think of surveillance, face recognition, unmanned flying vehicles or other uses on the web. Very few know of its other purposes in aerial photography, or manufacturing. Almost any mass-produced Item has been scanned for errors using some sort of computer vision. [36, pp.1–4]

Since it's already used in so many fields and has still an increase in demand, there are several programming libraries available.

5.2 OpenCV

OpenCV is, as the name implies, an open source computer vision library written in C/C++. The Python module which provides an interface of the native C/C++ library and practically has the same functionality, is used for image processing. There are implemented more than 500 functions in OpenCV used for a wide variety of different tasks. [36, pp.1–2]

The library abstracts the actual implementation of the different algorithms and at the same time still gives developers the opportunity to tinker with different settings, making it still possible to adjust them to the given task. Additionally, one doesn't have to know all the implementation details, making it possible to only focus on the mathematical challenges which are demanding already.

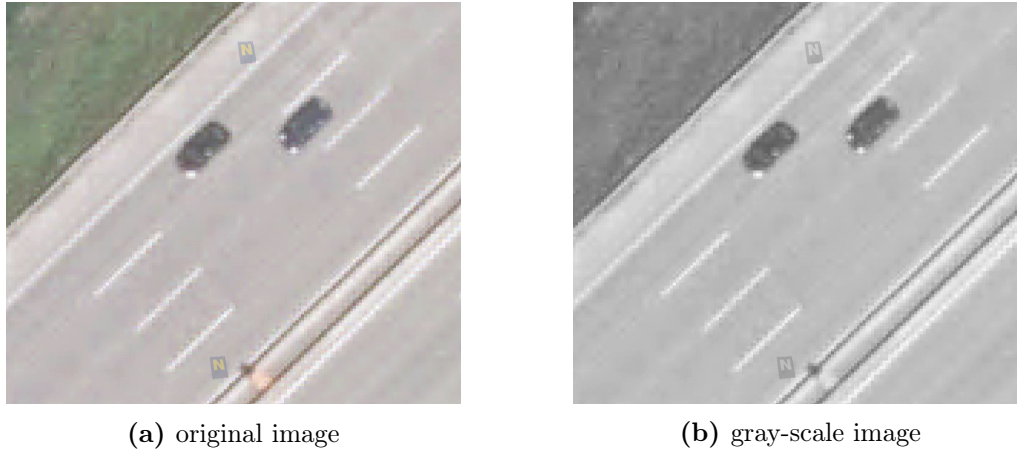


Figure 5.1: On the left side the original image is displayed. The right image shows how the initial one looks like after it has been converted to gray-scale.

5.3 Image Data Format

The images are generally saved as RGB bytes in memory, where each color is a 2-dimensional array and each color value can range from 0 to 255 which corresponds to 1 byte of data. The Pi Camera used has a resolution of 8 Mega-pixels, meaning that one captured image uses roughly 24 MB of RAM. Before applying any additional image processing algorithms to it, the image is converted to gray-scale (see 5.1), meaning that it only consists of one byte array using 8 MB of RAM. Therefore only one third of the memory space is needed, which makes applying the following algorithms easier and more efficient in terms of memory and CPU usage.

5.4 Smoothing

Camera images have practically always some amount of noise. Noise in this context is defined as any unwanted disturbance that interferes with the desired image. Such disturbances be caused by electrostatic or electromagnetic coupling with nearby power lines, radio transmitters or other electromagnetic signals in general. [37, pp.5–6]

Therefore the image is smoothed before any feature detection algorithms (e.g. line detector) are applied. Skipping this step would increase the likelihood of falsely detected features drastically.

5.4.1 How does Smoothing Work?

There are various different approaches for blurring an image, but they generally use the same principal. The idea is to normalize the image. Normalizing is understood as bringing the different intensity levels of the pixels closer and therefore eliminate intensities that differ greatly from their neighboring pixels. How many neighbors are included is decided by the kernel size.

5.4.2 Convolution Matrix

The convolution matrix, also called kernel, is used for calculating the new value of a pixel when smoothing or for other filter operations. The different smoothing algorithms only use different values for the kernel. 3x3 is the most common size for a convolution matrix, since it's sufficient to consider all direct neighbors of one pixel. In the following example it is shown how a 3x3 kernel is applied to a pixel of an arbitrary image:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9)$$

Where the 3x3 matrix consisting of the letters a to i represents a part of an image and the matrix denoted by the numbers 1 to 9, a fictive kernel. The right hands side equation result would be the new value for the middle pixel e , after the kernel was applied. [38, pp.233-234]

5.4.3 Mean Filter

As the name suggests, the mean filter takes the mean value of the pixels inside the kernel for calculating the new intensity value and thus normalizing the image. Therefore a 3x3 convolution matrix for a mean filter looks like:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

The perk of this method is its straight forward nature. The downside is that a single discordant value can distort the mean and create unnatural intensity levels, that are not present in the original image.[38, pp.150–152]

5.4.4 Gaussian Blur

Under utilization of the 2-D Gaussian function

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

the kernel can be generated. The original image is blurred to a certain degree depending on the standard deviation σ . The larger σ gets, the greater will the smoothing effect be. As σ grows larger a bigger kernel size is needed to properly reflect the Gaussian distribution. [38, pp.156–159]

The following 5x5 convolution matrix is an example for the Gaussian Blur using $\sigma = 1$:

$$\frac{1}{115} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

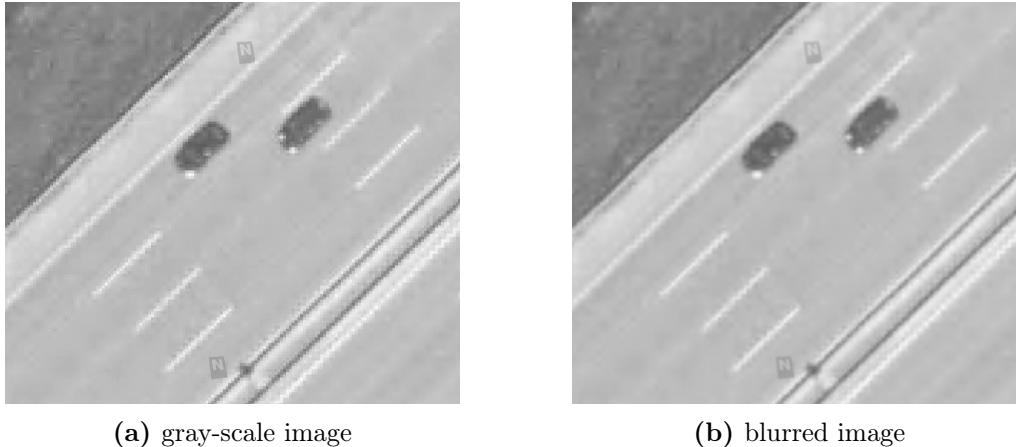


Figure 5.2: Using a standard derivation σ of 1.0 and a `ch:navigationkernel` size of 5x5 yields this blurred image as result.

5.4.5 Implementation in OpenCV

For applying the Gaussian Blur to a gray-scale image, the OpenCV library provides the function `cv2.GaussianBlur`. The following snippet show how the method is invoked:

```
1 #Read image from camera
2 img = self.video.read()
3 #Convert image to gray-scale
4 imggray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
5
6 ksize = (5,5)
7 sigma = 1.0
8 imgblur = cv2.GaussianBlur(imggray, ksize, sigma)
```

Description of variables and functions:

- `self.video.read()`: Reads the current frame from the PiCam
- `ksize`: Kernel size used for the Gaussian Blur. Note that the matrix not necessarily has to have the same width as height. But the convolution matrix needs a center, thus making width and height odd numbers.
- `sigma`: Standard derivation of the distribution function.
- `imgblur`: The image after the Filter has been applied.

5.5 Edge Detection

Edge detection is a technique that uses gray-scale images to determine the boundaries between two regions of different intensity values [38, p.183]. With the utilization of such algorithms it is possible to filter the road markings and thus detect them.

5.5.1 Canny Edge Detector:

The Canny Edge Detection Algorithm was one of the first feature detector used in this project. It is divided into 5 different steps:

1. Gaussian smoothing to reduce image noise
2. Find the gradients of the image

3. Suppress non-maximum values
4. Use two thresholds for determining potential edges
5. Suppress edges that are not attached to strong edges

Smoothing:

Firstly the input image is smoothed using the Gaussian Blur discussed previously. It's important to reduce noise since if one didn't, the noise could be mistaken for edges. [39, p.2]

Find Gradients:

The next step is to determine where the intensity of the images changes the most, those points have a large gradient magnitude. In order to calculate the magnitude of the gradients the Sobel-operator has to be applied first. The Sobel algorithm returns the gradient in the x- and y-direction by applying the following kernels, respectively: [39, p.3]

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Now the overall magnitude of the gradient can be determined by using the Pythagorean theorem

$$|G| = \sqrt{G_x^2 + G_y^2}$$

or a simpler Euclidean distance function like the Manhattan distance

$$|G| = |G_x| + |G_y|$$

to reduce the computational complexity. [39, p.3]

Non-maximum Supression:

The next step is to remove gradients that lay next to each other, so that only one of them remains. Therefore the local gradient with the highest intensity should be kept and the rest deleted. The gradients that are compared are determined by their direction

$$\theta = \arctan \frac{|G_y|}{|G_x|}$$

which is rounded to the nearest multiple of 45°. Then the gradient is compared to the nearest gradients that have the same or the opposite θ value. If the gradients magnitude is not a local maximum, it will be erased from the image. [39, p.4]

Double Thresholding:

The remaining pixels can still be result of noise or other color variations. Canny's edge detector uses a high and low threshold value, as opposed to a single value. If a pixel has a greater intensity than the high threshold it is labeled as *strong*, if it's between high and low value as *weak* and all other pixels are *suppressed*. [39, p.5]

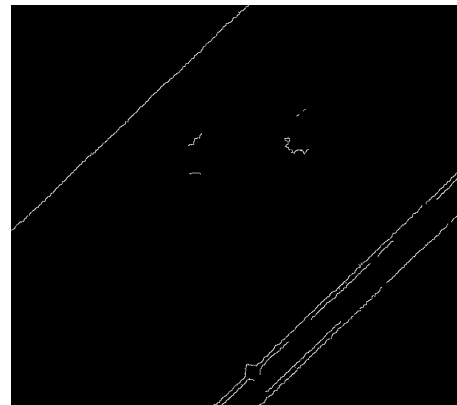
Create Final Edge Image:

The strong edges are definitely part of the final image and therefore be immediately included. The weak edges are only considered as final edges if they are connected to at least one strong edge. The reasoning for this method is that most edges labeled as weak and not connected to a strong edge, are probably caused by some sort of noise. Those which are although connected to at least one, are likely to be part of the same edge and should be included, so unintended gaps are prevented. [39, pp.5–6]

5.5.2 Implementation in OpenCV



(a) Image filtered using the Gaussian Blur. The kernel size and σ are automatically calculated by OpenCV.



(b) Image after the Canny's Edge Detector has been applied.

Figure 5.3: This figure illustrates the Canny's Edge Detector by using the `cv2.Canny` function provided by the OpenCV library.

Figure 5.3 illustrates what the an blurred image looks like after the Canny's Edge Detector has been applied. The following code has been used to generate the graphics:

```
1 #Get current frame after Gaussian Blur has been applied
2 imgblur = self.get_blurred_img((5,5),1)
3
4 low_threshold = 50
5 high_threshold = 150
6 imgcanny = cv2.Canny(imgblur, low_threshold, high_threshold)
```

Description of variables and functions:

- `self.get_blurred_img(ksize,sigma)`: Reads the current frame from the PiCam and apply the Gaussian Blur using a kernel size of 5x5 and $\sigma = 1$.
- `low_threshold`: The low threshold value used in double thresholding
- `high_threshold`: The high threshold value used in double thresholding
- `imgcanny`: The bitmap after the Canny's Edge Detector has been applied.

Although it is part of the edge detection algorithm to apply the Gaussian Blur, one can still manually lay a filter on top of it before passing it onto the edge detector. The reason being that OpenCV does not give fine control about the smoothing mechanism used in the `cv2.Canny`. The output is mainly influenced by the parameters `low_threshold`

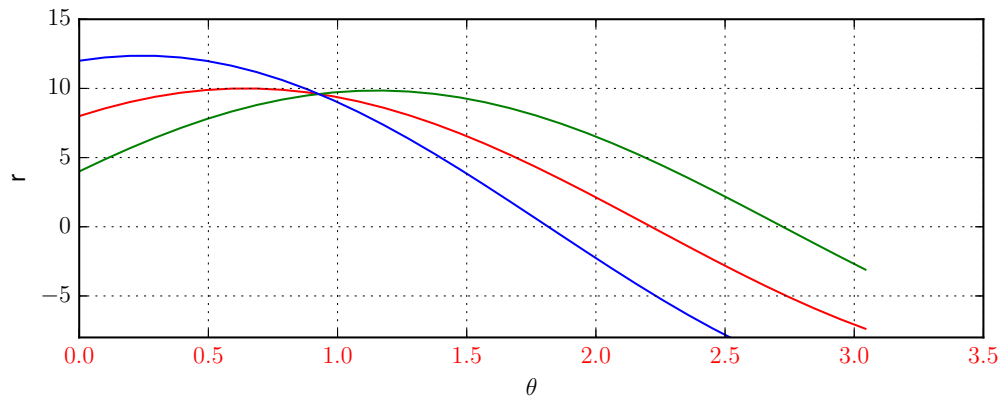


Figure 5.4: An example graph showing the line families for the points (8,6), (4,9) and (12,3), colored in red, green and blue, respectively. Only lines with $r > 0$ and $0 \leq \theta \leq 2\pi$ are considered.

`↔` and `high_threshold`. Those are the numbers used for the double thresholding step as described previously in 5.5.1.

5.6 Line Detection

With the edges being singled out it's still unclear what should be considered as a line and thus as a border marking. One of the most used algorithms for simple line detection is the *Hough Line Transform* [38, p.215].

5.6.1 Hough Line Transform

The algorithm makes use of the Polar coordinate system instead of the Cartesian one. The parametric form of the linear equation

$$r = x \cdot \cos \theta + y \cdot \sin \theta$$

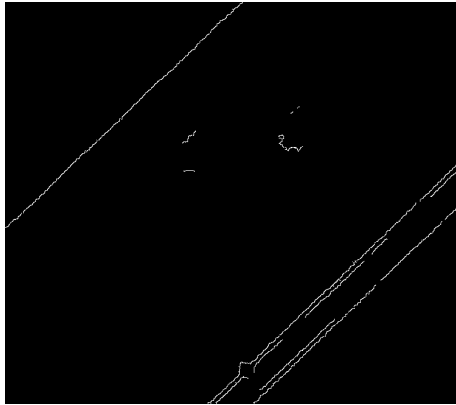
defines the family of lines for a known point (x, y) , meaning that every pair (r, θ) that satisfies said equation passes through the known point. The sinusoidal graph in 5.4 displays the line families for three different points. With this representation the lines shared by the points can be relatively easily recognized by looking at the intersections of the graph.

In the next phase the Hough Line Transform algorithm counts how many points of the total amount of points have the same line in common by this method and then applies a threshold removing all lines that don't pass through at least the amount of points specified by the threshold.

The simplicity of Hough Line Transform makes it quite easy to implement but also leads to no fine control over the result as the threshold is the only input parameter supplied. Thus it becomes apparent why feature detectors like the Canny's Edge Detector have to be applied prior to the actual line detector.

5.6.2 Implementation in OpenCV

Image 5.5b shows how the original image looks like when the lines obtained from the *Hough Line Transform* are drawn onto it. At first, the original image was read, the Canny's Edge Detector applied and at last the line detection:



(a) Image used as the base for the line transform.



(b) Image after the Hough Line Transform using a threshold of 200 has been applied.

Figure 5.5: The above two images nicely display how the Hough Line Transform works in combination with another feature detector.

```

1 img = self_video.read()
2 imggray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
3 imgblur = cv2.GaussianBlur(imggray,(5,5),2)
4 imgcanny = cv2.Canny(imgblur,50,150)
5
6 lines = cv2.HoughLines(imgcanny,1,math.pi/180,200)
7 if lines is not None:
8     for line in lines:
9         for rho,theta in line:
10             x0 = rho*math.cos(theta)
11             y0 = rho*math.sin(theta)
12
13             #Make lines longer than the image so they fill the whole picture
14             x1 = int(x0 + 1000*(-b))
15             y1 = int(y0 + 1000*(a))
16             x2 = int(x0 - 1000*(-b))
17             y2 = int(y0 - 1000*(a))
18
19             #draw the line
20             cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

```

Description of variables and functions:

- `cv2.HoughLines(imgcanny,1,math.pi/180,threshold)`: The OpenCV call for applying the Hough Line Transform. The constants 1 and `math.pi/180` specify that the line should start at the origin point. The threshold are minimal amount of points that have shared by a line to be accepted.
- `lines`: A list holding all lines detected by the algorithm.
- `rho`: The r value of a line of the lines set.
- `theta`: The polar angle of the line.
- `x0`: The x-coordinate as in the Cartesian system.
- `y0`: The y-coordinate as in the Cartesian system.
- `cv2.line(img,p1,p2,color),thickness`: Draw the line on the original image, using a color of (0,0,255), which is RGB an corresponds to red.

This is the first algorithm of the chain that actually doesn't return another binary

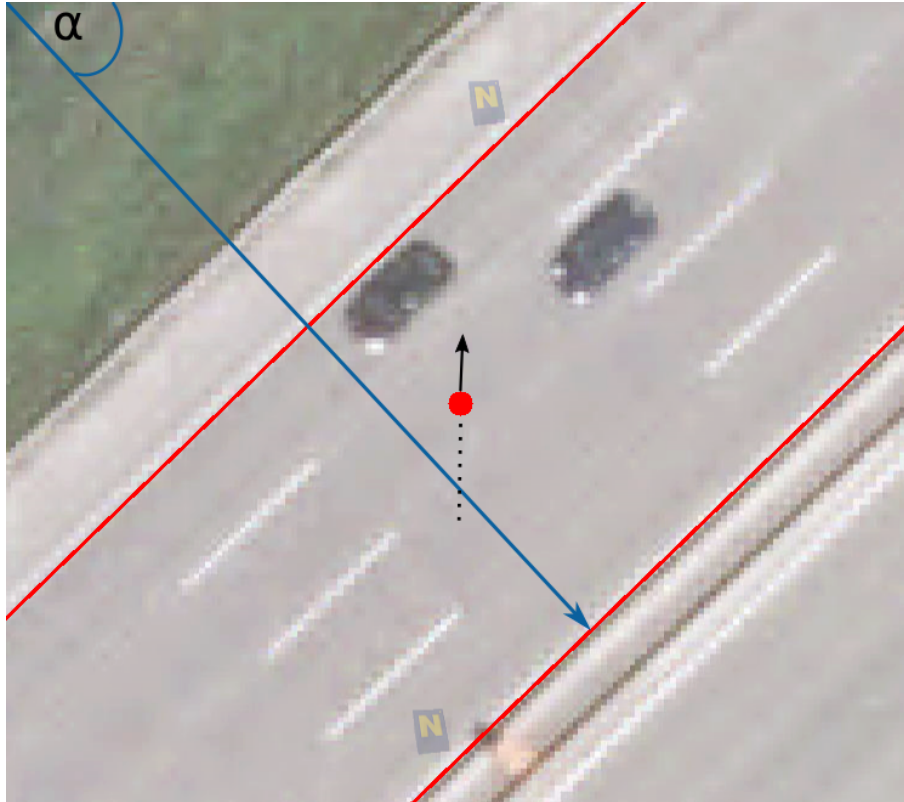


Figure 5.6: ©Land Niederösterreich, NÖ Atlas

An image showing an applied Hough Line Transform. The red dot symbolizes the position of the UAV (center of the image), the red lines are the ones detected by the Hough Line Transform and the cyan ones are their polar vector.

image but instead it returns a list of lines saved in the polar form (r, θ) . This is the last algorithm in the image processing chain and after that only the logical lines that were obtained are used.

5.7 Relative Position to Landmarking

After the line has been located, there still two unknown variables: the distance d to the line and the angle α between the vehicle and the landmarking as shown in 5.6. Depending on those values the UAV has to counter-steer in order to stay on the target. The program generally attempts to keep a relative angle to the line of less than 5° to the line, trying to minimize the total error and the need for counter-steering.

For the orientation α it is safe to assume that the PiCam lays directly in the yaw axis and thus the center of the image can be seen as current location of the UAV. The error caused is acceptable since the angle will eventually reach an equilibrium where it is parallel to the landmarking.

5.7.1 Angle Calculation

When using the `cv2.HoughLineTransform` functions the line is returned as (r, θ) . But one has to be careful because in OpenCV, the origin $(0,0)$ of a image usually starts at the top-left hand corner and *not* on the bottom-left one. In this particular case there is virtually

no difference, since the angle θ increases clock-wise. That means that in this context the drone has an orientation of -180° .

Figure 5.6 shows that the angle between the red street line and the UAV is defined as $90^\circ - \alpha$.

5.8 Distance Estimation

The next step is to roughly calculate the distance to the line. The problematic when making such an estimation are the unknown variables like height of the UAV or the angle of the camera. On one hand those values can fluctuate during flight and on the other hand the sensors responsible for getting the actual height of the vehicle, may not necessarily be accurate.

5.8.1 Camera and Focal Length

To understand the mathematics behind cameras, one has to roughly understand the different properties that its lens or other parts can have:

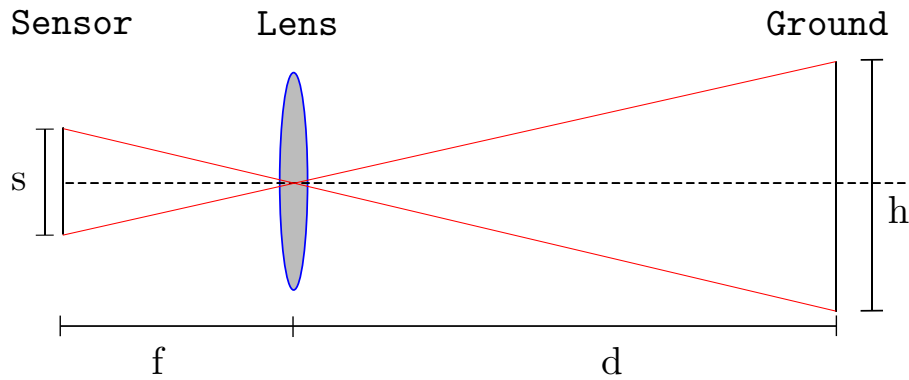


Figure 5.7: This figure illustrates how the lens bends to light in order to get a bigger field of view. The distance between the sensors and the lens is the focal length f , s is the the width of the sensor, d is the distance to the ground and h is the field of view.

Figure 5.7 displays how the light is travels through the lens and onto the sensor. The length between those two is called *focal length* [40, p.115] and is 3.04 mm for the PiCam, the sensor has an image area of $3.68 \cdot 2.76 \text{ mm}^2$. The camera has an horizontal field of view of about 62.2° [20]. The field of view (or sometimes called angle of view) is everything that is seen by the camera.

Therefore a larger sensor or a smaller focal length increases the over all field of view but on the loss of color fidelity since the sensor can only capture so many pixels.

5.8.2 Field of View Calculation

In photography the field of view is the field of vision of the camera, the parts of the real world that it actually can see. Its value is usually given in degrees since the actual size depends on the distance between camera and ground.

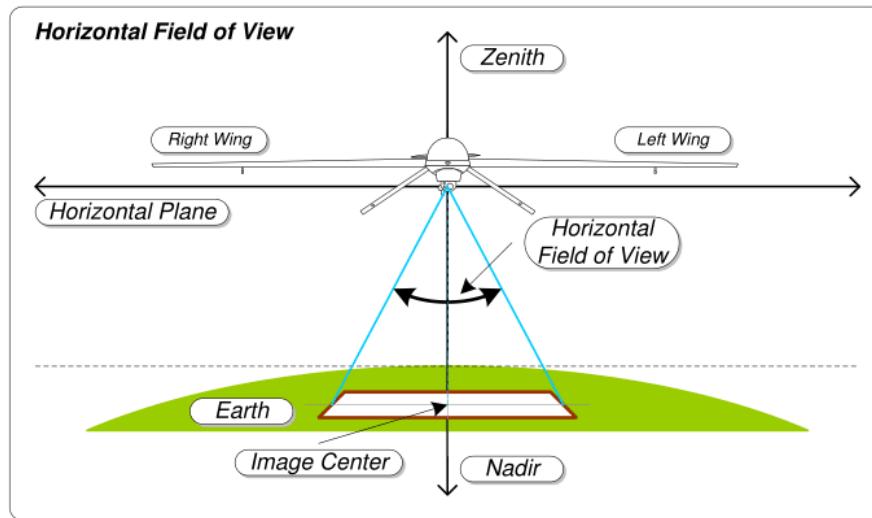


Figure 5.8: A schematic showing what the horizontal field of view of the camera is. The UAV shown in this figure does not correspond to the drone used in this project. Zenith is called everything above the UAS and Nadir everything beneath it. The image was taken from Motion Imagery Standards Board [41, p.66].

Horizontal Field of View

When taking figure 5.8 as reference the horizontal field of view can be calculated in meters by using some simply geometry:

$$h = 2 \cdot d \cdot \tan \frac{FOV_H}{2}$$

where:

h ... horizontal field of view in meters

d ... distance to the ground in meter

FOV_H ... horizontal field of view in degrees, the PiCam v2 has a horizontal field of view of about 62.2°

Vertical Field of View

The same applies to the vertical field of view, the only difference being that the FOV_V has an angle of 48.8°. The actual size of the vertical pixels and not only the horizontal ones is needed, since landmarking will not always be parallel and thus for calculating the shortest path, the vertical ones are needed as well.

5.8.3 Convert Pixels to Actual Meters

With the essential groundwork being done, the size that one pixel represents can be determined by using the resolution of the sensor. As mentioned before the PiCam v2 has 3280 width pixels and 2464 height pixels. Yet again, by usage of some trivial math the distance in meters between the pixels can be calculated.

$$width = \frac{h}{3280} \qquad height = \frac{v}{2464}$$

where:

h ... width of the horizontal field of view in meters
 v ... height of the vertical field of view in meters
 $width$... width of one pixel in meters
 $height$... height of one pixel in meters

5.8.4 Get Shortest Distance in Pixels

Now with the actual dimensions of a pixel known, the shortest distance from the UAV to the line can be determined. The general formula for calculating the shortest between a point and a line, that is given in the general equation

$$a \cdot x + b \cdot y + c = 0$$

can be written in this form [42, p.15]:

$$d = \frac{|a \cdot x_0 + b \cdot y_0 + c|}{\sqrt{a^2 + b^2}}$$

where:

x_0 ... x coordinate of the UAV in the image
 y_0 ... y coordinate of the UAV in the image
 d ... distance in pixels

Using the parametric equation from 5.6.1, we can substitute the constants and variables:

$$d = |\sin\theta \cdot y_0 + \cos\theta \cdot x_0 - r|$$

Note that the denominator of the original fraction is 1, as it equates to the Pythagorean Identity $\sin^2 \theta + \cos^2 \theta = 1$.

The next step is to determine the width in actual meters. Since the horizontal field of view is not the same size as the vertical one, the orientation of the drone to the line does matter. It has to be kept in mind that the orientation of the UAV is not α in this context but $90^\circ - \alpha$. Thus when using sine and cosine they can be swapped out to achieve the offset of 90° : $\sin 90^\circ - \alpha = \cos \alpha$ and $\cos 90^\circ - \alpha = \sin \alpha$.

Bearing this in mind the actual distance l can be worked out:

$$l = \cos \alpha \cdot d \cdot height + \sin \alpha \cdot d \cdot width$$

First, the amount of vertical and horizontal pixels is calculated using cosine and sine, respectively. With the two groups being separated the height and width per pixel that was determined previously, can be used to get the overall distance.

Chapter 6

Flight(Florian)

6.1 Autonomous vs. Manual Mission

A fully autonomous mission not only ensures the usability for every possible group of users but it also guarantees a easy to use product. The breakdown drone is designed to help people in a car accident to setup the breakdown triangle in a very easy and fast way so the affected people do not have to worry about the triangle and simply can start working on fixing their problems if possible. If however the breakdown drone were to be controlled manually or semi manually it still takes time and effort a car accident casualty probably doesn't have. Drones controlled manually does mean a well trained pilot is required. So in order to use the breakdown drone, at least one passenger has to be trained to not only fly the drone but also feel comfortable in challenging situations like flying near fast moving cars on a highway. In a semi manual operation there could be some kind of app involved where the driver is able to set a location and/or a flight path for the drone in order to avoid obstacles easily. Disadvantages of a fully autonomous drone are certainly the avoidance of obstacles such as trees or other cars/trucks. Another very difficult aspect of an autonomous drone are the moral issues such as: who is at fault if the drone crashes and causes another accident? But with all that in mind it is still very beneficial to have a fully featured and simple to setup drone in an emergency situation.

6.2 Pre-Flight

Due to the physical connection of the Raspberry Pi to the drone via USB cable, it is possible to communicate over the serial interface ACM0. In Raspbian all the serial interfaces are found under the /dev directory. To ensure a stable connection the baudrate is set to 57600. This means there is a maximum data transfer rate of 57.6 kbit/s.

The vehicle then has to wait for a strong GPS signal in order to get the exact location. Afterwards the location is read and additionally the cardinal direction which is used to determine in which direction from its starting position it is supposed to head.

6.3 Takeoff

At this point it is still possible that the vehicle is not quite initialized yet. If thats the case the program simply waits for the vehicle to continue.

The vehicle now needs to be set into a GUIDED mode so the drone knows it is getting its flight commands from the serial interface. Now the drone is ready to arm its motors. This can take some time, hence the program has to wait for the vehicle to finish.

Taking off is now possible. The default height is 10 meters above the ground.

Listing 6.1: Code snippet for taking off.

```

1 print "Taking off!"
2 vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude, aTargetAltitude
    = 10
3
4 while True:
5     #prints current altitude.
6     print " Altitude: ", vehicle.location.global_relative_frame.alt
7     #stops current loop if the target altitude is reached.
8
9     if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
10         print "Reached target altitude"
11         break
12     sleep(1)

```

6.4 Flying the Path

The previously calculated path of nodes is stored in a python list of tuples. The tuple contains two elements, the latitude and the longitude of the node.

6.4.1 Heading for a Node

It is now possible to fly to all nodes sequentially. Before the vehicle can head for a node it has to be converted to a **LocationGlobal** object so the goto function knows where the flight point is.

Listing 6.2: Code snippet that shows the approach of the calculated nodes.

```

1 #flying the waypoints
2 for node in path:
3     #converting the latitude, longitude and altitude to a LocationGlobal object
4     loc = LocationGlobal(node[0], node[1], 10)
5     print loc     #prints the next node
6     #initiates the movement in air.
7     vehicle.simple_goto(loc, airspeed=3 , groundspeed=3)
8     #calculating the remaining distance from current position to node
9     remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
10    #checks if the current position is 1 meter or less towards the node.
11    while not remainingDistance <= 1:
12        print "Distance to target: ", remainingDistance
13        sleep(0.1)
14        remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
15    print "reached node!"
16
17 sleep(5)

```

6.4.2 Calculating the Distance to a Node

Now the Raspberry Pi constantly needs to check whether or not the node has actually been reached. If the distance from the vehicle is less than one meter away from the location of

the node it is considered as if the exact location is reached.

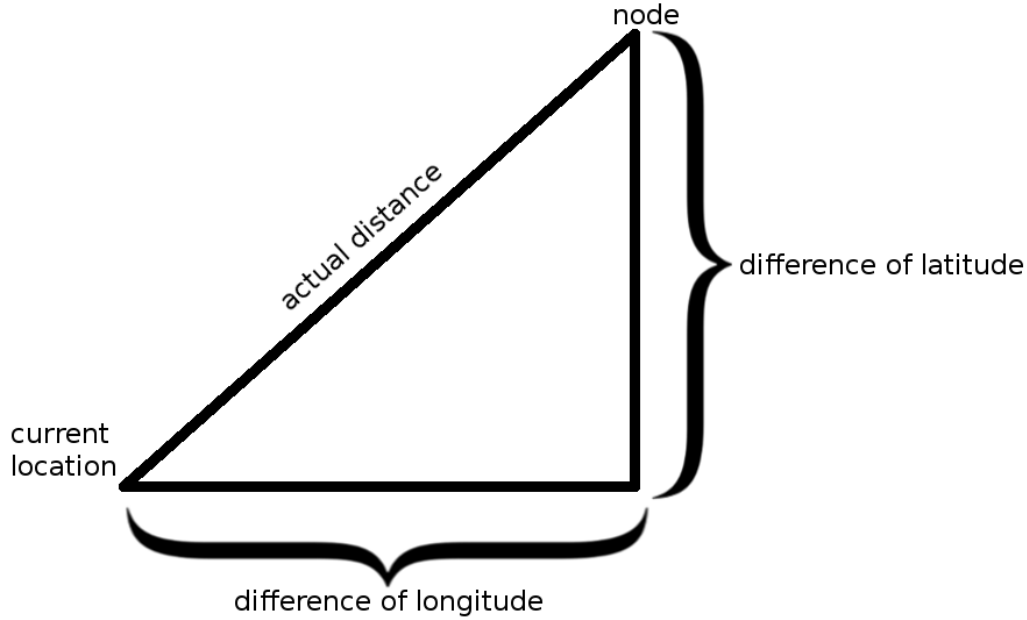


Figure 6.1: Shows the information needed to calculate the distance between the current location and the node.

$$node = (48.013551, 16.320265)$$

$$location = (48.009288, 16.313751)$$

$$\Delta latitude = node.lat - location.lat = 0.004263$$

$$\Delta longitude = node.lon - location.lon = 0.006514$$

$$\sqrt{\Delta latitude^2 + \Delta longitude^2} = 0.007785$$

$$fac = 40075017/360^\circ = 1.113195 \cdot 10^5$$

$$0.007785 \cdot 1.113195 \cdot 10^5 = 866.61m$$

6.5 Landing and Steering

While landing the drone is a task that can be achieved by using the vehicle mode “LAND” steering the vehicle towards the road side/line is a far more complicated task. The “LAND” mode can be called by DroneKit API but the steering to the sides has to be made using the MAVLink message factory shown below.

Listing 6.3: The “LAND” mode gets called and after that, the vehicle steers to whatever direction the function tells it to.

```

1 #landing
2 print "landing"
3 vehicle.mode = VehicleMode("LAND")
4
5 while vehicle.location.global_relative_frame.alt > 0.5:
6     sleep(0.2)
7     direction = Camera.calcwheretogo();
8     send_ned_velocity(direction[0], direction[1], direction[2], direction[3])
9     print vehicle.location.global_relative_frame.alt

```

The function “send_ned_velocity” steers in a certain direction for a certain amount of time.

Listing 6.4: This function tells the drone to follow the specified vectors for a specified amount of time.

```

1 def send_ned_velocity(velocity_x, velocity_y, velocity_z, duration):
2     #Move vehicle in direction based on specified velocity vectors for a certain
    amount of time.
3     msg = vehicle.message_factory.set_position_target_local_ned_encode(
4         0,          # time boot ms (not used)
5         0, 0,       # target system, target component
6         mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, # frame
7         0b0000111111000111, # type mask (only speeds enabled)
8         0, 0, 0, # x, y, z positions (not used)
9         velocity_x, velocity_y, velocity_z, # x, y, z velocity in m/s
10        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS Mavlink)
11        0, 0)    # yaw, yaw rate (not supported yet, ignored in GCS Mavlink)
12
13     # send command to vehicle on 1 Hz cycle
14     for x in range(0,duration):
15         print x
16         vehicle.send_mavlink(msg)
17         sleep(1)

```

6.6 Releasing the Breakdown Triangle

As soon as the drone has landed safely the breakdown triangle can be dropped from its mount. Therefore the Raspberry Pi needs to control the DC-motor attached to the custom control module. The program needs to access the GPIO pins turn them to “HIGH” and after a brief amount of time it needs to turn them off again.

Listing 6.5: This code turns the DC-motor attached to the GPIO pins of the Raspberry Pi on and after 4 seconds off again.

```

1 import RPi.GPIO as GPIO
2 from time import sleep
3
4 GPIO.setmode(GPIO.BOARD)
5
6 #defines to which ports the motor is connected
7 Motor1A = 16
8 Motor1B = 18
9 Motor1E = 8

```

```

10
11 #sets all the GPIO pins to read data
12 GPIO.setup(Motor1A,GPIO.OUT)
13 GPIO.setup(Motor1B,GPIO.OUT)
14 GPIO.setup(Motor1E,GPIO.OUT)
15
16 #turns the motor on
17 print "Turning motor on"
18 GPIO.output(Motor1A,GPIO.HIGH)
19 GPIO.output(Motor1B,GPIO.LOW)
20 GPIO.output(Motor1E,GPIO.HIGH)
21
22 #waits for the motor to have opened
23 sleep(4)
24
25 #stopps motor again
26 print "Stopping motor"
27 GPIO.output(Motor1E,GPIO.LOW)

```

6.7 Way back to the Starting Point

After successfully unloading the breakdown triangle the drone can now start its way back to the vehicle it initially came from. Therefore the drone has to take off again and fly the list of nodes in the reversed order it came. After doing so the Breakdown Drone can land again.

Listing 6.6: Arms the drone and takes off, then flies the waypoints in reverse order and lands again.

```

1 arm_and_takeoff(10)
2
3 node_len = len(path)
4
5 while node_len > 0:
6     loc = LocationGlobal(path[node_len-1][0],path[node_len-1][1], 10)
7     print loc
8     vehicle.simple_goto(loc)
9
10    remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
11    while not remainingDistance <= 1: #checks if the current position is 1 meter or
        less towards the node.
12        print "Distance to target: ", remainingDistance
13        sleep(0.1)
14        remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
15    print "reached node!"
16    node_len = node_len-1
17
18 loc = LocationGlobal(current_latitude, current_longitude, 10)
19 print loc
20 vehicle.simple_goto(loc, airspeed=3, groundspeed=3)
21
22 remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
23 while not remainingDistance <= 1: #checks if the current position is 1 meter or less
        towards the node.
24    print "Distance to target: ", remainingDistance
25    sleep(0.1)
26    remainingDistance=get_distance(vehicle.location.global_relative_frame, loc)
27 print "reached node!"
28

```



```
29 print "landing"  
30 vehicle.mode = VehicleMode("LAND")
```

6.8 Performance Analysis

In computer science the performance analysis is a key element of software development. The methods of program analysis allows developers to better understand more complex programs. Interpreters and Compilers can use certain tools of performance analysis to optimize the runtime as well.

6.8.1 Why Performance Analysis?

In robotic environments controllers often have very limited resources available. Therefore it is very important to write code which uses as little CPU runtime and internal memory as possible. The main factors that have to be considered while looking at performance are not only CPU runtime and memory usage but also

6.8.2 Different Methods of Analysis

The three main analysis tools are: static, dynamic and hybrid analysis.

The so called 'static analysis tools' never modify the binary image of the application but instead use techniques like source code instrumentation or sampling to obtain analytic data. As soon as the results are recorded they can now be analyzed to determine any bottlenecks that may exist in an analyzed program. Static analysis tools however are incapable of modifying a running program, therefore any statistical data a developer wishes to determine has to be specified before the application is run. Since these tools report their results asynchronously it is not possible to get notified of any performance issues while the application is running and the performance is recorded. Meaning a developer can only get their desired results after an entire run has completed, so static analytic tools can not provide any real time feedback to diagnose the performance. In addition to that, said analysis tool can cause unintended side affects because they require either a dedicated data collection routines in a set of code or the use of external sampling routines. This can cause massive overhead while gathering statistical data especially in embedded robotics systems. The external code can additionally change the behavior of said analyzed program, therefore it introduces performance problems that did not exist prior to analysis, this can even cause a partially false analysis. Although they can cause so much trouble providing statistics, they still gather mostly very useful data for a developer in the real world.[43]

Because static analysis tools view the binary image of said program as a 'black-box' that should not be modified under any circumstances, dynamic analysis tools rely on modification of the binary image of the application. These modifications are typically inserted during runtime of the application so a highly accurate statistic can be gathered in real time. This enables insight into program performance that would not be possible for static analytic tools. Dynamic analysis tools get divided into two different types: binary instrumentation and probing. Tools that use binary instrumentation are able to inject customized analysis routines into arbitrary locations within an application binary to record wide variety of performance data. Probing tools however need support routines enabled by shared libraries! to gather information about the systems internal status, this enables the analysis of multiple levels of abstraction. Because binary instrumentation tools are able to inject

analysis routines into the application binary, they are able to change the structure of the application they are supposed to profile. Here as well programs tend to run a little bit slower during the analysis process caused by the insertion of performance monitoring routines. In general it is recommended for developers to only analyze only those code segments that are more likely to cause performance issues so the rest of the application does not get affected by the profiler.[43]

Developers who use hybrid profiling methods try to blend in parts of both static and hybrid analysis methods. This allows to only implement the most effective features both methods offer. Therefore these profiler are often capable of providing a level of utter utility that is impossible to be achieved by any single purpose analysis mechanism. Due to the wide variety of mixture of both static and dynamic analysis methods it is not really possible to determine a 'typical' implementation of a hybrid profiler. It is to be mentioned that if a hybrid profiler needs more runtime than a single-purpose profiler it is of no avail. This massive overhead comes from the fact that certain types of static and dynamic instrumentation mechanisms are unable to run in parallel mode. If that happens the effective overhead of both static and dynamic methods add up together to the runtime of the profiled application. So hybrid analytic tools should only be used by developers if they need information that cannot be provided by single purpose profilers.[43]

6.8.3 Tested Device

Since the Raspberry Pi 3 is the main driver for the Breakdown Drone it is also the main tested device. With its ARM Cortex-A53, 1.2 GHz Quad core CPU and 1 GB of RAM it should be able to perform the tasks required quite easily.

6.8.4 Indicators Analyzed

The two main indicators for performance on the Breakdown Drones Raspberry Pi are CPU usage and memory usage. Both of these can be gathered by the Linux/Unix tool 'ps'. Therefore a batch script (displayed in 6.7) was written to gather data every second. This batch script was executed at the begin of the application and ran in a sub process of the main application at the end of the main application the subprocess was killed as well.

Listing 6.7: Bash script that writes the performance into a csv file every second.

```

1 #!/bin/bash -e
2 echo "performance tracking script"
3 NAME="$(date '+%Y-%m-%d--%H-%M-%S').csv"
4 touch $NAME
5 echo "date;time">$NAME
6 while true; do
7     DATE=$(date '+%Y-%m-%d;%H:%M:%S')
8     CPU=`ps -p $1 -o \%cpu | tail -n +2`
9     MEM=`ps -p $1 -o \%mem | tail -n +2`
10    echo "$DATE;$CPU;$MEM">>$NAME
11    sleep 1
12 done

```

6.8.5 Performance during Flight

In figure 6.2 one can see the CPU performance on a graph over time. The performance was measured every single second. Clearly noticeable is the first spike in performance, it is at about 7% CPU usage overall. This happens during setup and path calculation. The next low of CPU usage is directly after setup, this is because the Raspberry Pi has to wait for the vehicle to initialize in order to safely lift off the ground. From this point on the CPU usage hovers at about 2-2.5% during flight.

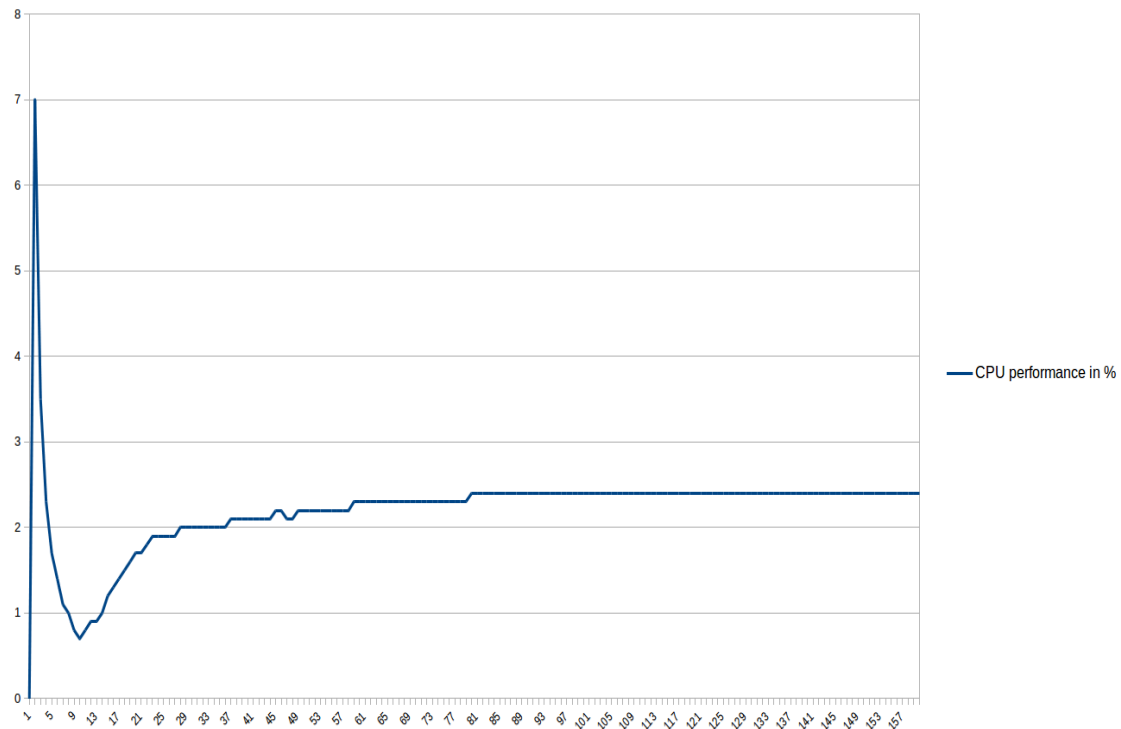


Figure 6.2: Shows the CPU usage over time during takeoff, flight and landing in percent.

The memory usage stays at a consistent 0.1% (as shown in 6.3) of the total amount of memory. This is because every calculation happens in real time and none of the data needs to be stored temporarily.

6.8.6 Discussion of Results

Since the CPU performance during regular flight is always below 3% it is possible to say that it won't effect the Linux operating system running in the back and whatever it has to deal with. The memory is not taking much of the total memory as well, so in conclusion it is safe to say the Raspberry Pi can handle the performance needed for flying the Breakdown Drone easily.

6.8.7 Methods of increasing Performance

The first spike in CPU usage is when the vehicle is initializing, this could also be done manually but since the Breakdown Drone is supposed to work without the interference of a user this was not an option. the 2.3% could also be slightly reduced by outsourcing some

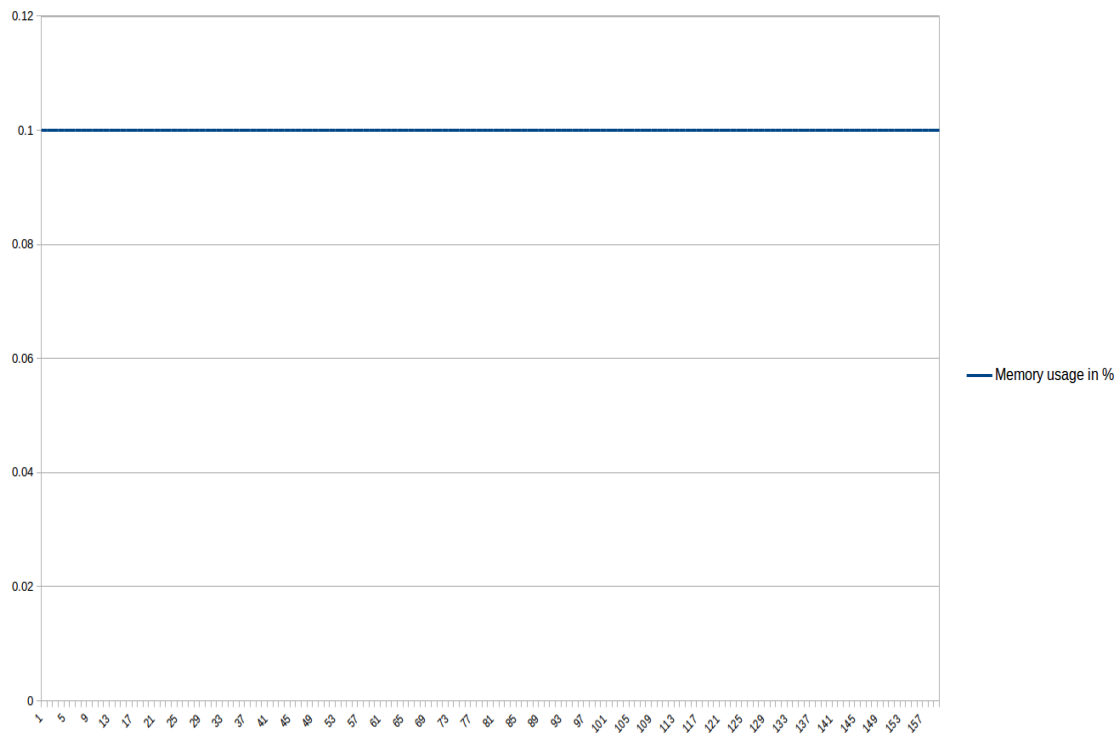


Figure 6.3: Shows the Memory usage over time during takeoff, flight and landing in percent.

of the distance calculations to the flight controller itself. Though it has been determined that the flight controller suffers from the additional CPU load from time to time.

Chapter 7

Conclusion

7.1 What was Planned?

The Breakdown Drone was supposed to be a helping tool in a very rough situation like a car accident. It needed to be not only safe but very fail-safe as well. Probably the two main problems during development were a safe flight and a safe landing. For the flight a path finding had to be implemented to fly the drone approximately along the road the accident is. Especially tricky was the landing because GPS is not always accurate enough to be able to rely on GPS only. Therefore a system had to be developed that can help out when it comes to precisely landing near the roadside. The use of a camera was the obvious choice. With an open-source library like open-cv it is possible to detect either a roadside or a road marking, based on this data the system should be able to judge what the right approach is for the landing approach.

7.2 Benchmarking

7.2.1 Flight Accuracy

To test the drones accuracy under different circumstances like different wind levels or even snowfall three shapes were drawn on a map, a square, a triangle and a straight line. The coordinates of every shape were determined and used manually as the drones waypoints instead of generating them based on the offline card material. After setting up a program for each shape it could be started at any given time. During the testflights the Raspberry Pi on the drone would record the actual location of the drone by writing the GPS coordinates into a CSV-file every 0.5 seconds. After a complete flight the coordinates from the waypoints were portrayed on a map and a straight line was drawn between them. The previously recorded coordinates could now be displayed on the map as well. It also had to be recorded with how much wind strength the testflight was made. It can be read from the map how much the drone was off from the different shaped lines with a certain amount of wind strength.

7.2.2 Landing Accuracy

One of the most important, if not the most important tasks of the Breakdown Drone is to accurately land on the side of the road. It is not only dangerous for the drone but also the bypassing cars and their occupants. The landing accuracy can simply be determined by measuring how far the drone is off a specified target. To put it simpler, several testflights are made telling the drone to land exactly on a straight line after each landing has been

done the distance between this line and the center of the drone is measured and recorded in a CSV-file. This method can now be applied in different scenarios, like before with various different weather conditions.

7.3 Testing

7.3.1 Testing of Flight Accuracy

Testing the flight accuracy turned out to be a little problematic because of the drone behaving oddly. What is meant by that is that the 3DR Iris+ was crashing at seemingly random occasions which it never did before. Seemingly random means it was never quite clear as of why this is the case. Sometimes it would crash right after takeoff not even making it to its target altitude, sometimes it would crash after reaching the targeted altitude and sometimes when a node was reached. At this time firmware version 3.4 was on the Pixhawk autopilot. It seemed like this software version was not stable enough for the Breakdown Drone so it was downgraded to an apparently more stable version 3.2. The problem of crashing while taking off was resolved with this downgrade. However the Breakdown Drone would still crash after reaching target altitude or after reaching the first node. As a first method of debugging the drone was started multiple times under different wind strengths. This however made no difference what so ever. A second guess was that the `simple_goto` function of the DroneKit API was now working properly. So to eliminate this reason of failure a function was implemented that would send a MAVLink message directly without the intervention of the API. But the result was still the same, the Breakdown Drone still crashed. The third possible reason of failure was the height sensor of the 3DR Iris+. To tell whether or not this was the reason for the drone to crash the drones measured altitude was printed onto the console of the Raspberry Pi. During flight one could clearly see the target altitude was printed on the console, however the drone was losing altitude rapidly. Because of these tests one can assume that the altitude sensor was the point of failure. So neither the build in function from the API could detect a drop in height nor the self written function could steer against this drop. For the Breakdown Drone to get back in the air this sensor had to be replaced, however since these tests were made in February of 2017 and the drone was already discontinued by the developers in July 2016, there was no support for the drone anymore.

7.3.2 Testing of the Landing Accuracy

Since the landing of the drone works by slowly descending at a certain velocity and waiting for the graph of the accelerometer to spike, it was not affected by the failure of the height sensor. For the data to be recorded the drone flew to an approximate altitude of about 10 meters off the ground. Then the program with the landing adjustment was started and after the drone had landed it was measured how far off the drone was to the line that was headed for. To collect meaningful data this process was repeated 25 times and after each time the steering function was adjusted. Whether or not the drone was to the left from the line or to the right was not significant to the test since the only interesting factor is the deviation.

One can see in graph 7.1 that after about 11 tries and a deviation 8 to 11 cm was reached and it was not possible to get any nearer to the line. This can be because of factors like the wind blowing the drone to the side just a little bit right before it lands. Because the drone is already so near to the ground can neither detect the line anymore nor steer to adjust to it.

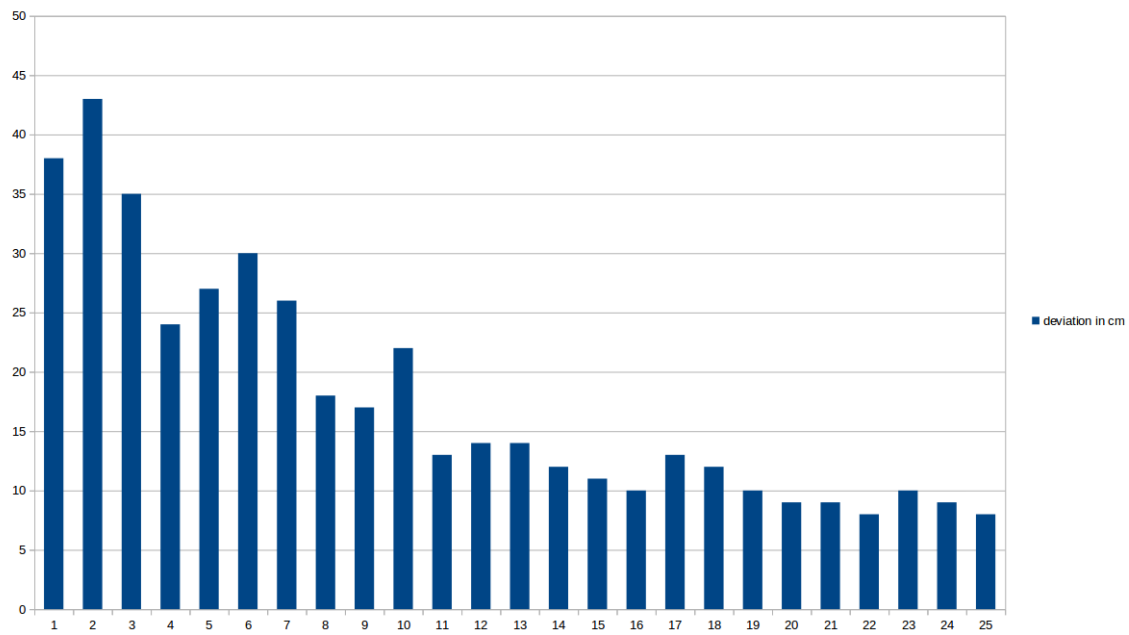


Figure 7.1: This graph shows the deviation of the drones middle point to the line the drone was supposed to head for in the process of landing the vehicle as near as possible to the line.

7.4 Results

The landing did work with an average error of about 9 cm which for a task like this is more than accurate enough. Since the height sensor did stop working during development of the Breakdown Drone it was sadly not possible to fully test its capabilities. However it was fully implemented and should work on any other drone with the Pixhawk autopilot as the flight controller. Because of the more or less success of this project it can safely be said that a further development in this field can be recommended.

7.5 Further Development

The key features of the Breakdown Drone are implemented however that does not mean it is in its final state. There are several areas that can still be expanded or improved.

7.5.1 Obstacle Avoidance

Obstacle avoidance is a very important topic when it comes to movement of autonomous vehicles. It can range from an autonomous driving car that needs to be able to dodge people suddenly jumping on a street to a drone needing to evade an upcoming tree. This is especially important when the life of a human being could be on the line. For such a important task very good sensors have to be used. The probably best sensor for a cheap price is the Microsoft Kinect. The Kinect however has to be looked at carefully because the IR sensor might not work outdoors. If this is the case, a different, preferably a sensor with a viewing angle of 360° has to be purchased and this is going to cost a lot of money.

7.5.2 Hardware Upgrade

Hardware upgrade not only means the replacement of the hight sensor of the 3DR Iris+, which of course is the most problematic component of the drone. Different drones in general should be taken into consideration when looking to develop the Breakdown Drone further. If possible a drone with higher payload capacity should be acquired. This can mean a better camera can be put on the UAV, an additional infrared sensor or even a better computer then the Raspberry Pi could be added to the drone if needed.

7.5.3 Triangle

If a different drone is purchased one may look at a different breakdown triangle as well. The version for this drone was designed to be light weight so the 3DR Iris+ is still able to carry it alongside the weight of the custom control module, while still withstanding the strong winds from the Breakdown Drone while it is flying away. A heavier triangle means better stability and a higher surface and reflection area, therefore easier to recognize for other bypassing drivers.

Acronyms

3DR 3D Robotics. 6, 13

CSI Cameral Serial Interface. 13

GPIO General-purpose input/output. 9, 13

GPS Global Positioning System. 1, 6, 7, 16–19

OSM Open Street Map. 18, 19

PBF Protocolbuffer Binary Format. 19

UAS Unmanned Aerial System. 2

UAV Unmanned Aerial Vehicle. 1–4, 6, 16, 17, 19–21

UGH Unmanned Ground Vehicles. 16

WGS84 World Geodetic System 1984. 16

XML Extensible Markup Language. 19

Bibliography

- [1] *Unmanned aircraft systems (uas)*. [Online]. Available: http://www.icao.int/Meetings/UAS/Documents/Circular%20328_en.pdf (visited on 11/19/2016).
- [2] [Online]. Available: <http://dspace.bracu.ac.bd/xmlui/bitstream/handle/10361/4203/Drone-Thesis%5C%20Paper.pdf> (visited on 11/19/2016).
- [3] *Telemetry, summary of concept and rationale*. [Online]. Available: <http://adsabs.harvard.edu/abs/1987STIN...8913455> (visited on 11/19/2016).
- [4] [Online]. Available: http://www.ctie.monash.edu.au/hargrave/rpav_home.html (visited on 11/19/2016).
- [5] *Remote piloted aerial vehicles*. [Online]. Available: http://www.sf-encyclopedia.com/entry/low_a_m (visited on 11/19/2016).
- [6] *Reginald denny (1891-1967); the dennyplane*. [Online]. Available: <http://www.ctie.monash.edu.au/hargrave/dennyplane.html> (visited on 11/19/2016).
- [7] [Online]. Available: <http://www.historyplace.com/worldwar2/timeline/v1.htm> (visited on 11/19/2016).
- [8] [Online]. Available: <https://www.flightglobal.com/pdfarchive/view/1952/1952%5C%20-%5C%201148.html> (visited on 11/19/2016).
- [9] [Online]. Available: <http://www.dtic.mil/dtic/tr/fulltext/u2/a525674.pdf> (visited on 11/19/2016).
- [10] [Online]. Available: <http://motherboard.vice.com/read/vintage-videos-show-the-us-air-forces-first-armed-drone-dropping-bombs> (visited on 11/19/2016).
- [11] [Online]. Available: <http://www.wsj.com/articles/SB126325146524725387> (visited on 11/19/2016).
- [12] [Online]. Available: http://cordis.europa.eu/programme/rcn/4902_en.html (visited on 11/19/2016).
- [13] [Online]. Available: <https://www.wired.com/2012/01/drone-report/> (visited on 11/19/2016).
- [14] [Online]. Available: <https://amazon.com/primeair> (visited on 11/19/2016).
- [15] D. of Transportation(DOT), *Rulemaking RIN 2120-AJ60: Operation and Certification of Small Unmanned Aircraft Systems*, 2015.
- [16] J. Weier and D. Herring, *Measuring vegetation(NDVI & EVI)*, 2000. [Online]. Available: <https://earthobservatory.nasa.gov/Features/MeasuringVegetation/> (visited on 01/03/2017).

- [17] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, "Multipurpose UAV for search and rescue operations in mountain avalanche events", *Geomatics, Natural Hazards and Risk*, pp. 1–16, 2016.
- [18] H. Chao, Y. Cao, and Y. Chen, "Autopilots for small unmanned aerial vehicles: A survey", *International Journal of Control, Automation and Systems*, vol. 8, no. 1, pp. 36–44, 2010.
- [19] [Online]. Available: http://space.skyrocket.de/doc_sdat/navstar.htm.
- [20] *Camera module - raspberry pi documentation*, 2016. [Online]. Available: <https://github.com/raspberrypi/documentation/blob/master/hardware/camera/README.md> (visited on 03/18/2017).
- [21] J. Barnett, *Controlling dc motors using python with a raspberry pi*, 2014. [Online]. Available: <https://business.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051> (visited on 01/22/2017).
- [22] I. Corporation, *Enhanced host controller interface specification for universal serial bus; revision 2.0*, 2002.
- [23] [Online]. Available: <http://www.dji.com/products/drones> (visited on 02/26/2017).
- [24] [Online]. Available: <http://www.banana-pi.org/m64.html> (visited on 02/26/2017).
- [25] [Online]. Available: http://www.pollin.de/shop/dt/MjY3NzkyOTk-/Bauelemente_Bauteile/Entwicklerboards/Cubie_Board/Cubieboard_3_Cubietruck_Kit_A20_2_GB_8_GB_WLAN_BT_SATA.html (visited on 02/26/2017).
- [26] S. Wood, *A brief history of python*, 2015. [Online]. Available: <https://www.packtpub.com/books/content/brief-history-python> (visited on 03/18/2017).
- [27] [Online]. Available: <http://qgroundcontrol.org/mavlink/start>.
- [28] [Online]. Available: <http://python.dronekit.io>.
- [29] A. Ronacher, *Flask, a microframework for python based on werkzeug*. [Online]. Available: <http://flask.pocoo.org/> (visited on 11/24/2016).
- [30] A. El-Rabbany, *Introduction to GPS: The global positioning system*. Artech House, 2002.
- [31] S. Malys, J. H. Seago, N. K. Pavlis, P. K. Seidelmann, and G. H. Kaplan, "Why the greenwich meridian moved", *Journal of Geodesy*, vol. 89, no. 12, pp. 1263–1272, 2015, ISSN: 1432-1394. DOI: [10.1007/s00190-015-0844-y](https://doi.org/10.1007/s00190-015-0844-y). [Online]. Available: <http://dx.doi.org/10.1007/s00190-015-0844-y>.
- [32] R. W. Sinnott, "Virtues of the haversine", *Sky and Telescope*, vol. 68, no. 2, pp. 159+, 1984.
- [33] T. Vincenty, "Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations", *Survey review*, vol. 23, no. 176, pp. 88–93, 1975.
- [34] *Planet OSM*. [Online]. Available: <http://planet.openstreetmap.org/> (visited on 01/31/2017).
- [35] *Openstreetmap/osmosis*. [Online]. Available: <https://github.com/openstreetmap/osmosis> (visited on 01/31/2017).
- [36] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

- [37] C. D. Motchenbacher and J. A. Connelly, *Low-noise electronic system design*. Wiley New York, 1993, pp. 5–6.
- [38] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, “Hypermedia image processing reference”, *Department of Artificial Intelligence, University of Edinburgh*, 1994.
- [39] J. Stewart, *Canny edge detection*, 2009. [Online]. Available: <http://watkins.cs.queensu.ca/~jstewart/457/notes/24-canny.pdf> (visited on 03/18/2017).
- [40] T. Dobbert, *Matchmoving: The Invisible Art of Camera Tracking*. Sybex, 2005, ISBN: 0782144039,9780782144031.
- [41] “UAS Datalink Local Set”, Motion Imagery Standards Board, Standard, Oct. 2014.
- [42] B. Spain, *Analytical conics*. Courier Corporation, 2007.
- [43] J. Thiel, *An overview of software performance analysis tools and techniques: From gprof to dtrace*.