BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

DESIGN OF QUADCOPTER CONTROL SYSTEM WITH VISUAL GUIDANCE
NÁVRH ŘÍZENÍ KVADROKOPTÉRY S VIZUÁLNÍM NAVÁDĚNÍM

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Ondřej Pokorný**
AUTOR PRÁCE

SUPERVISOR — **Ing. Ondřej Andrš, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Bachelor's Thesis Assignment

Institut:	Institute of Solid Mechanics, Mechatronics and Biomechanics
Student:	**Ondřej Pokorný**
Degree programm:	Applied Sciences in Engineering
Branch:	Mechatronics
Supervisor:	**Ing. Ondřej Andrš, Ph.D.**
Academic year:	2017/18

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Bachelor's Thesis:

Design of quadcopter contol sytem with visual guidance

Brief description:

The aim of the thesis is to design a quadcopter control system with visual guidance towards a target. The solution will be based on the use of computer vision in the field of multirotor platforms. The goal is to integrate commercially available units into a comprehensive control system for the flying platform. At the core of the system will be the Raspberry microcomputer and the Arduino microcontroller with custom software.

Bachelor's Thesis goals:

1. Perform a study of the current issue, including requirements and specifications.
2. Perform the design of the automatic control unit.
3. Implement the proposed solution.
4. Verify and test the implemented solution.

Recommended bibliography:

UPTON, Eben a Gareth HALFACREE. Raspberry Pi: uživatelská příručka. 2., aktualizované vydání. Přeložil Jakub GONER. Brno: Computer Press, 2016. ISBN 978-80-251-4819-8.

MONK, Simon. Raspberry Pi cookbook. Beijing: O'Reilly, 2014. ISBN 1449365221.

NORRIS, Donald. Build your own quadcopter: power up your designs with the Parallax Elev-8. New York: McGraw-Hill Education, 2014. ISBN 0071822283.

SCARPINO, Matthew. Motors for makers: a guide to steppers, servos, and other electrical machines. Indianapolis, Indiana: Que, 2015. ISBN 0134032837.

Students are required to submit the thesis within the deadlines stated in the schedule of the academic year 2017/18.

In Brno, 26. 10. 2017

L. S.

prof. Ing. Jindřich Petruška, CSc. Director of the Institute	doc. Ing. Jaroslav Katolický, Ph.D. FME dean

Abstrakt

Tato práce pojednává o návrhu a implementaci systému řízení kvadrokoptéry s vizuálním naváděním k vytištěné značce. Systém se skládá výhradně z komerčně dostupného hardwaru a open source nebo uživatelského softwaru. Jednotkami jsou například mikropočítač Raspberry Pi 3 Model B, mikrokontrolér Arduino Nano, letový kontrolér Omnibus F3 atd.

V první části je nastíněna struktura navrhnutého systému a popsány vlastnosti a funkce jednotlivých komponent. Následuje přehled použitých druhů komunikací a jejich verzí specifických pro létající platformy. Nakonec je popsána architektura uživatelského softwaru společně s fungováním jednotlivých částí a důvody pro jejich přítomnost v kódu.

Druhá část se zaměřuje na použití knihovny pro augmentovanou realitu ArUco za účelem odhadování polohy, dále se zaměřuje na opatření zavedená pro kompenzaci nedostatků spjatých s použitím tohoto systému. Tato část také obsahuje popis vývoje řídícího algoritmu a následného testování implementovaného řešení. Na závěr jsou navženy možné další kroky ve vývoji.

Summary

This thesis deals with the design and comprehensive implementation of a quadcopter control system with visual guidance towards a printed marker. The system consists exclusively of low-cost, commercially available hardware and open-source or custom software. The units used are, for example, microcomputer Raspberry Pi 3 Model B, microcontroller Arduino Nano, flight controller Omnibus F3, etc.

In the first part, the structure of the system is outlined and the properties and functions of the components described. Following is an overview of the communications used and their versions specific to flying platforms. Finally, the architecture of the custom software is described together with the inner workings of the single parts and the reasons for their presence in the code.

The second part details the use of the ArUco augmented reality library for pose estimation, including the measures introduced to compensate for the inherent flaws of this system. This part also contains a description of the control algorithm development and of the subsequent testing of the implemented solution, as well as suggested further steps.

Klíčová slova

kvadrokoptéra, vizuální navádění, bezpilotní prostředky, bakalářská práce, VUT v Brně.

Keywords

quadcopter, visual guidance, UAV, bachelor's thesis, BUT

POKORNÝ, O. *Návrh řízení kvadrokoptéry s vizuálním naváděním.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018. 31 s. Vedoucí bakalářské práce Ing. Ondřej Andrš, Ph.D..

I hereby declare that I have written the bachelor's thesis on my own according to advice of my supervisor Ing. Ondřej Andrš, Ph.D. and that all sources are listed in the bibliography

Ondřej Pokorný

I'd like to thank my family for their moral and material support, my thesis leader and prof. Tomáš Březina for their advice and the municipality of Borač for providing me with the testing area.

Ondřej Pokorný

Contents

1. Introduction

Unmanned Aerial Vehicles (UAVs) have recently become a major focus of research for they allow for a variety of applications, ranging from payload transportation and search-and-rescue missions to aerial photography and infrastructure inspections. Specifically, rotary-wing UAVs (also known as multirotors) are gaining massive popularity; since they are capable of vertical take-off and landing as well as hover and slow movements, they are suitable for flying in confined spaces or executing precise operations. On the other hand, unlike fixed-wing UAVs, multirotors lack any natural stabilization elements and require Inertial Measurement Units (IMUs) in conjunction with considerable computational resources to keep the aircraft stable. The advancements in miniaturization of these technologies and their increased affordability are the main reasons behind the recent expansion of multirotors.

1.1. Goals and Motivations

Despite the significant progress in terms of computational power, localization of UAVs remains a substantial challenge, particularly in GPS-denied environments. It is vital to autonomous flight that the vehicle can obtain its own position in respect to its surroundings. Many possible applications of multirotors include environments that obstruct the GPS transmission and so, additional sensory input is needed. Traditional sensors used for localization in robotics, such as laser scanners, are often unsuitable for UAVs due to the payload and power consumption constraints inherent to flying platforms. Therefore, cameras are often used, being a cheap, lightweight means of acquiring information-rich data.

The goal of this thesis is to design and implement a quadrotor control system based around the aforementioned visual input. The solution should consist of low-cost, commercially available units and open-source or custom software. The sensing, image processing and computation should be done entirely on-board, as external systems and ground stations pose serious limitations in real-life usage of UAVs. The aircraft should be capable of hovering indoors with the use of basic control principles.

1.2. Existing Solutions

The task of estimating the aircraft's pose (i.e. position and orientation) is typically solved by utilizing an array of infrared cameras and emitters distributed around the testing area and placing reflective markers on the tracked object. Such systems offer up to sub-20μm accuracy [1] and are frequently used as so-called ground truth systems, providing a reference when assessing the accuracy of the tested method, as can be seen in [2]. This solution, however, does not fulfill the requirement of on-board sensing and tends to be rather expensive.

One of the major issues of on-board localization via cameras is obtaining the absolute scale of the viewed scene. In [3], homography is estimated between consecutive images and Simultaneous Localization And Mapping (SLAM) is employed to estimate the attitude and velocity. The data is then fused with measurements from an inertial unit through an Extended Kalman Filter (EKF). Alas, low-cost commercially available IMUs only

provide angular velocity and acceleration, not translational. Moreover, the experiment was carried out in an outdoor environment in a large field, allowing for accuracy within 2 meters, rendering it completely inapplicable for the goal mentioned above.

A similar approach was taken in [4] where the accuracy was greatly increased with stereo vision, using two front-facing and two downward-facing cameras, achieving an average error of 11.0cm in hover. Unfortunately, dual cameras not only result in larger platform and higher power demands, but also increase the computing power needed for image processing. Furthermore, SLAM and EKF add a layer of complexity that is beyond the scope of this thesis.

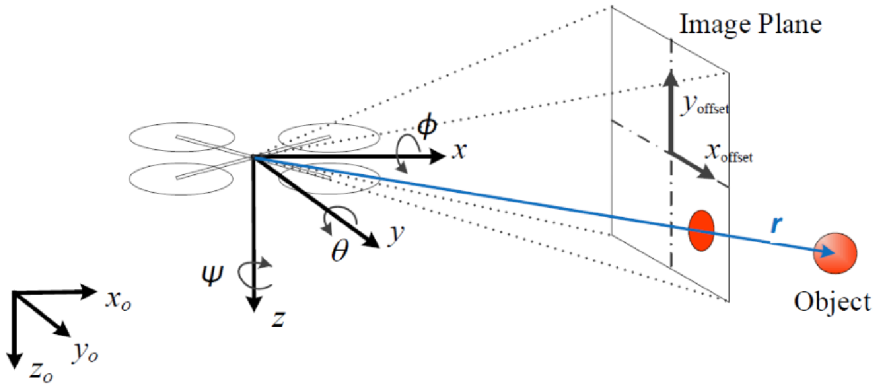

Figure 1.1: A figure from [2] showing the image plane, which has to remain unaffected by the quadcopter's roll and pitch movements (© 2014 IEEE)

In many scenarios (e.g. following or escorting), it is sufficient to know the exact position of the UAV in relation to a specific object. Given that the geometry of the object is known *a priori*, the scaling factor no longer presents a problem. As shown in [2], it is possible to realize such an object-tracking system using only a Raspberry Pi microcomputer, Raspberry Pi native camera sensor and the OpenCV library. Nevertheless, the nature of the object used (a bright colored sphere) introduces the need for a camera gimbal to keep the camera's attitude constant. To avoid this need and reduce the payload the quadrotor has to carry, printed markers (originally used for augmented reality) can be utilized as presented in [5].

1.3. Proposed Solution

The quadcopter itself will be assembled from hobby parts in a configuration that is commonly used (and has been proven optimal for the chosen frame size) to ensure reliable performance. To reduce the complexity of the task, an off-the-shelf Flight Controller (FC) with an embedded IMU and open-source firmware will be employed, providing attitude stabilization. Additionally, ultrasound range finder will be connected to the FC to aid the custom position control algorithm by using the already-present altitude control feature of the FC's firmware. The FC then, based on the stick commands and the data from its sensors, outputs control signals to the Electronic Speed Control modules (ESCs) that drive the quadcopter's motors.

The position sensing will be executed through a downward-facing camera and a printed marker from an open-source augmented reality library. The marker will be placed on the

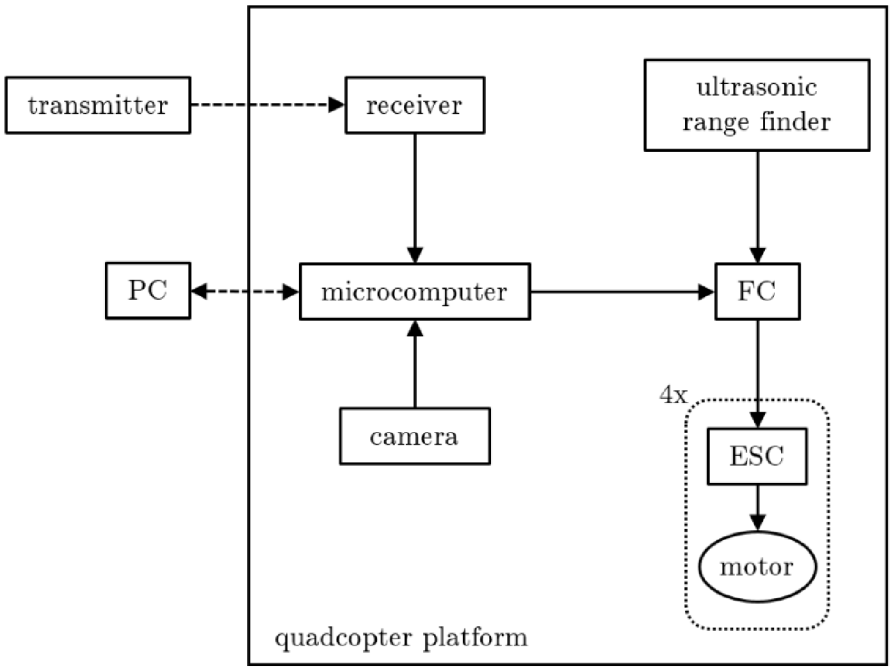

Figure 1.2: A block diagram of the proposed solution

ground beneath the aircraft. The image processing and position control computations will be performed on a microcomputer (attached to the flying platform), using the aforesaid library and custom software. The microcomputer will also be receiving stick commands from a hand-held radio controller (generally referred to as a "transmitter") to enable manual takeover at any moment as well as initial take-off and final landing.

Lastly, the microcomputer will be connected to a laptop (PC) via Wi-Fi, transmitting relevant data for debugging and receiving commands for activating the software along with custom software updates. The entire proposed system is illustrated in fig. 1.2.

2. Platform

The development of the flying platform began in the spring of 2017. The first version (pictured in figure 2.1) was built upon a pre-assembled quadrotor in a V-tail configuration; the rear arms are inverted and tilted upwards around the longitudinal axis of the body. Such construction leaves sufficient space for all the necessary components as the top rear is protected from the propellers. Unfortunately, the quadcopter proved itself unsuitable for hovering due to the fact that the flight controller was mounted without any dampening. The vibrations from the motors that propagated through the frame to the accelerometer in the FC's IMU resulted in invalid readings, as the sum of the accelerations in all axes becomes greater than the gravitational acceleration and it becomes impossible to determine the "downwards" direction that is used for attitude calculation.

Figure 2.1: The first version of the quadrotor platform

The design of the frame and the circuit board did not allow for any after-market dampening elements to be implemented. Also, the whole system consisted of atypical and proprietary solutions and only the Emax RS2205 2300KV brushless DC motors and Emax Lightning 20A ESCs were possible to transplant into a new frame.

The latest version can be seen in fig. 2.2 with the units labeled. All the components installed are covered in chapter 2.1 with the exception of the Power Distribution Board (PCB), ESCs and motors. The propellers used are HQprop 5x4.3x3 V1S Durable, the frame size is 250mm (motor axis to motor axis diagonally) and the system is powered by a 4-cell 1500mAh Lithium-Polymer battery.

2.1. Hardware

As outlined in chapter 1.3, at the core of the platform is an Omnibus F3 flight controller. The FC ensures stability and attitude control based on the stick input. Since the FC's firmware (INAV version 1.9) has a surface-following feature, an ultrasonic rangefinder (US-100) was connected via an Arduino Nano microcontroller board. The board accommodates initiating and receiving distance measurements from the rangefinder and sending the readouts to the FC upon request, using the I^2C bus. Accounting for the fact that Arduino Nano uses 5V logic levels and the Omnibus F3 uses 3.3V logic, a bi-directional logic level converter had to be inserted between those two devices.

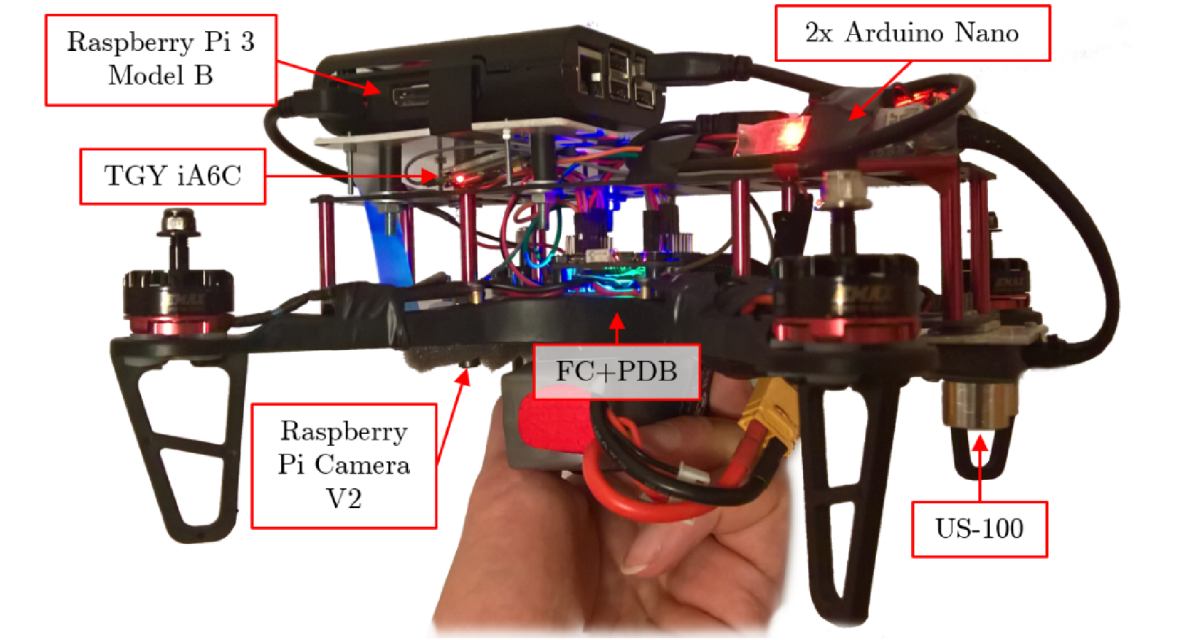

Figure 2.2: The current version of the quadrotor platform

The stick input into the FC comes from the Raspberry Pi 3 Model B via the MSP serial interface. Essentially, the microcomputer acts as a pilot, processing the visual data from the Raspberry Pi camera module, performing closed-loop position control computations, and then, depending on whether the aircraft is in automatic or manual mode, either outputting the results of the algorithm or just forwarding the stick commands coming from the operator. Moreover, the part of the custom software present on the microcomputer that is responsible for image processing and position control sends data through a ROS network (described in chapter 2.1.2) to the part of the software that receives and outputs stick commands. The ROS master of that network is located on a laptop connected to the microcomputer's wireless network, making it possible to log the data and later analyze it.

In the test scenario, the operator uses the TGY i6S radio controller for manual maneuvering. The stick input is received by the TGY iA6C radio receiver and turned into a PPM signal. As the PPM signal is very time-sensitive, another Arduino Nano had to be dedicated to the task of decoding it and forwarding the values via USB to the Raspberry Pi, since the microcomputer is unsuitable for precise tasks requiring μs accuracy. The early version of the platform used a single Arduino Nano for receiving the PPM signal and data from the microcomputer and generating output PPM signal leading to the FC. That solution, however, suffered from data fluctuations and output noise due to the fact that at any given moment, only one interrupt process can be running on the Arduino's MCU, thus creating inaccuracies in the time sensitive readings and signal generation.

The entire implemented system is captured in figure 2.3, including the specific components and communication interfaces used. The following sub-chapters briefly describe the units and give insight into the principles of the communications used, especially the versions which are utilized in the field of UAVs.

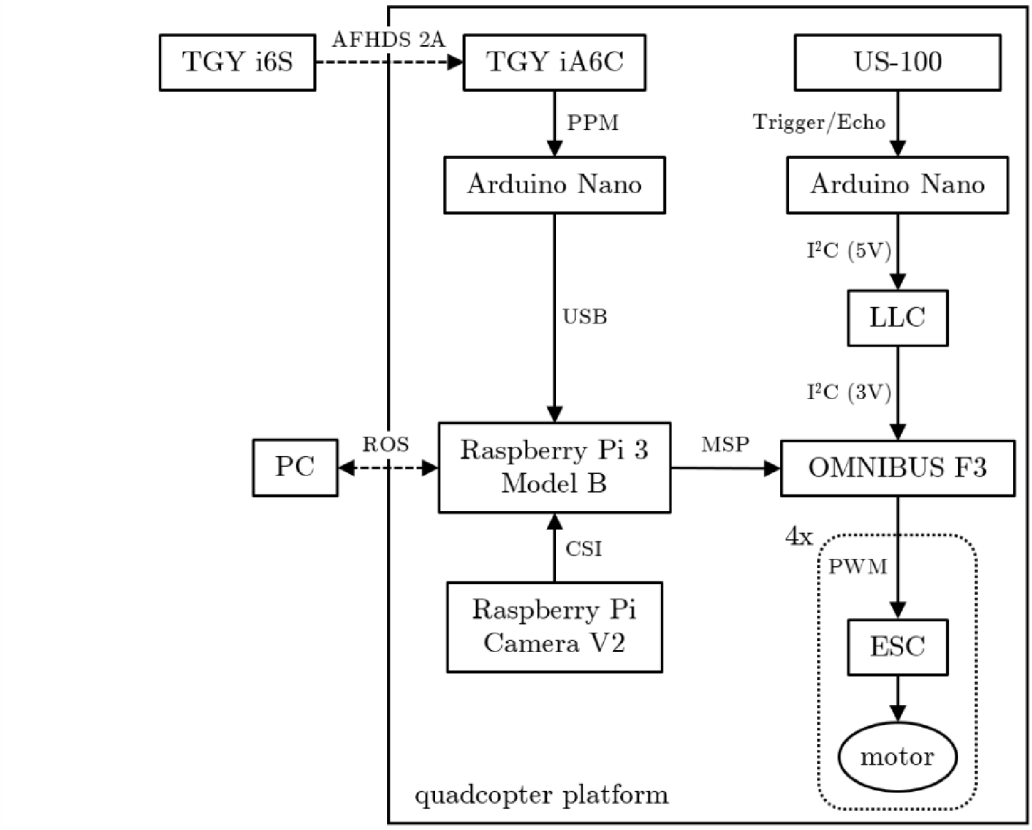

Figure 2.3: A block diagram of the implemented solution

2.1.1. Unit descriptions

Arduino Nano

Arduino Nano (fig. 2.4) is a small single-board microcontroller based on the Atmel ATmega328. Among its features are an integrated miniUSB port, 8 analog pins, 22 digital pins (6 of which are PWM enabled), 32kB of flash memory, 2kB SRAM, 1kB EEPROM and a clock speed of 16MHz. The Arduino Nano accepts an input voltage between 5 and 12 Volts. It is worth noting that all Arduino software and Arduino hardware designs are distributed as open-source, thus many manufacturers can produce Arduino boards legally, making it a cheap, universal and ubiquitous tool for prototyping and Do-It-Yourself (DIY) projects.

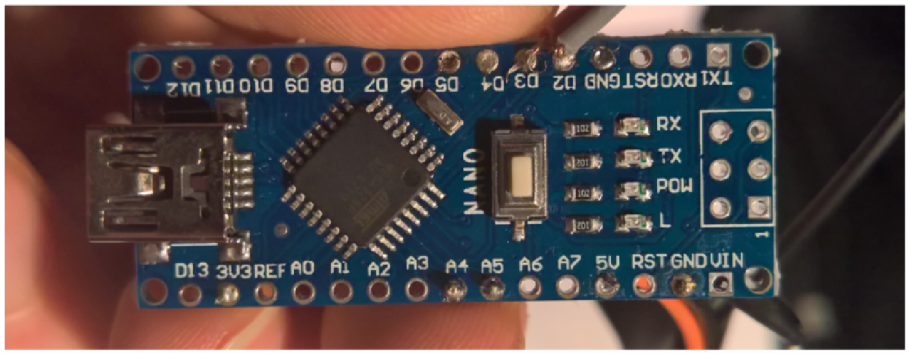

Figure 2.4: An Arduino Nano V3 board manufactured by Geekcreit (Tong De Limited)

LLC - Logic Level Converter

A bi-directional logic level converter (also called a logic level shifter) is a circuit that safely steps down signals from higher voltage to lower voltage and vice versa.

OMNIBUS F3

Omnibus F3 (fig. 2.5) is a popular flight controller based on the STM32 F303 Micro-Controller Unit and the MPU6000 Inertial Measurement Unit (3-axis gyroscope, 3-axis accelerometer). It features a BMP280 barometer unit, an On-Screen Display chip, a microSD card slot (for blackbox data storage), 8 PWM outputs, 3 UARTs (Universal Asynchronous Receiver-Transmitter interfaces) and SPI, PPM and I^2C connections, and many more flying platform-specific peripherals and connectivities. The board can be powered by voltage of up to 21V (the voltage of a fully charged 5 cell Lithium-Polymer battery).

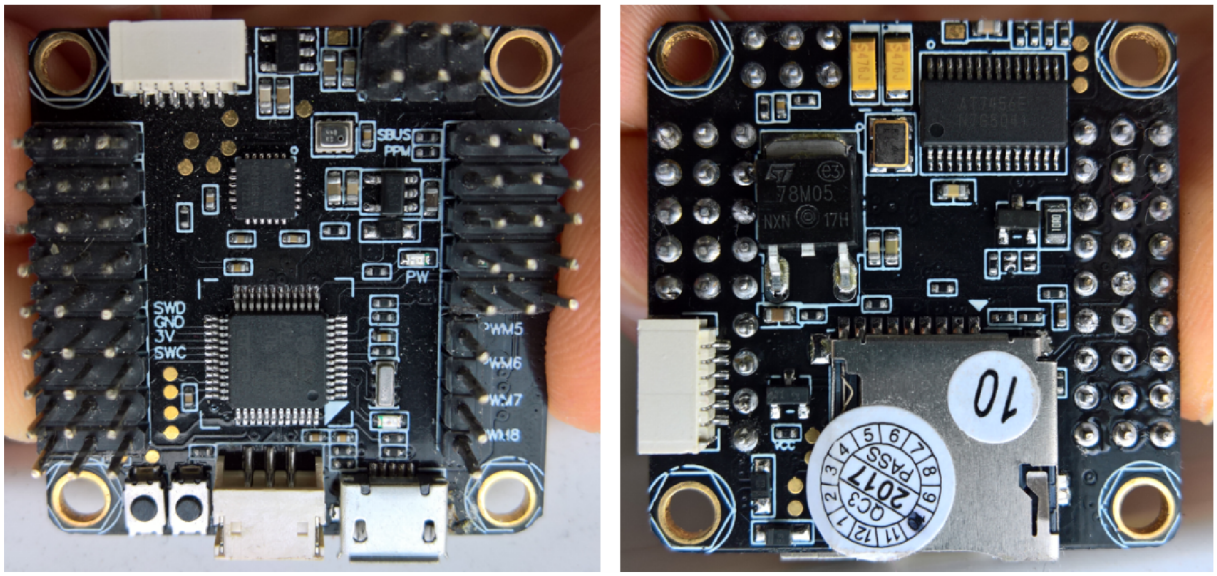

Figure 2.5: An Omnibus F3 flight controller board with all header pins soldered.

Raspberry Pi 3 Model B

Raspberry Pi is a series of microcomputers developed by the Raspberry Pi Foundation to facilitate education in the field of computer science in schools and developing countries. Since the initial product launch in 2012, Raspberry Pis have outgrown their original purpose and found application in such fields as robotics, as well as prototyping and DIY projects. As of March 2018, 19 million units have reportedly been sold [6]. The predominant operating system of choice is Linux.

The Model B of the third generation (fig. 2.6) was the first one from the series to introduce a built-in Wi-Fi chip, along with Bluetooth Low Energy. At the heart of this model is a Quad Core Broadcom BCM2837 64-bit ARMv8 processor accompanied by 1GB RAM. It offers full connectivity, including HDMI, 4 USB ports, an Ethernet port, 4-pole jack (audio and video) port, 40-pin extended GPIO, CSI and DSI ports, a microSD card slot and a microUSB port (power only).

Figure 2.6: The Raspberry Pi 3 Model B [7]

Raspberry Pi Camera V2

The Raspberry Pi Camera module V2 is a second generation of the official camera module for Raspberry Pi microcomputers. It has an 8-megapixel sensor Sony IMX219 and is capable of capturing still images together with videos of resolution up to 1920×1080 pixels and a frame rate of 30 FPS.

TGY i6S

TGY i6S is a 2.4GHz radio controller by Turnigy. It uses the AFHDS 2A protocol described in chapter 2.1.2. Among its highlights are two internal dual omnidirectional antennas (oriented at a 90° angle to each other), a backlit LCD touchscreen, a microUSB port for connecting to PC (firmware updates and simulator controller) and a variety of position, momentary and self-centering proportional wheel switches.

TGY iA6C

TGY iA6C is a 2.4GHz radio receiver by Turnigy. It also uses the AFHDS 2A protocol and has dual omnidirectional antennas. Its output signal is either encoded as PPM (see chapter 2.1.2) or S.BUS/I.BUS (digital protocols). Unlike many other radio receivers, the TGY iA6C has a built-in voltage sensor to report the battery voltage back to the transmitter.

US-100

US-100 (fig. 2.7) is an ultrasonic rangefinder offering up to 1mm accuracy and a range of distances from 2 to 450cm. The sensor's operating voltage is 5V, its detection cone angle is 15° and it can communicate either in a trigger/echo mode (as detailed in chapter 2.1.2) or in a serial mode which also outputs temperature to further increase accuracy of the readings (speed of sound is temperature-dependent). The mode is selected via a jumper on the backside.

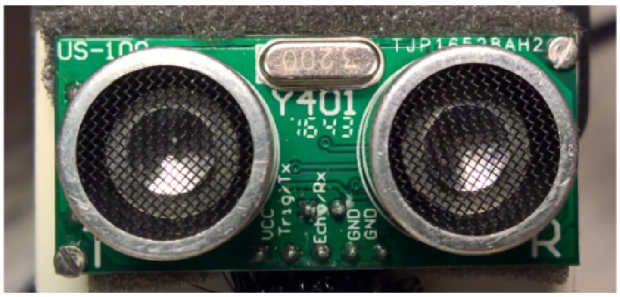

Figure 2.7: The US-100 mounted on the flying platform.

2.1.2. Communications overview

AFHDS 2A

The AFHDS 2A (Automatic Frequency Hopping Digital System) is a second generation of the data transmission protocol developed by FLYSKY RC model technology Co.,LTD and is used in their and their partners' 2.4GHz radio control systems.

CSI

The Camera Serial Interface (CSI) is the standard for connecting camera modules to processors, originally developed for the mobile phone industry by the Mobile Industry Processor Interface (MIPI) Alliance.

I²C

I²C (Inter-Integrated Circuit) bus is a popular asynchronous serial interface that allows for communication between multiple "slaves" and even multiple "masters" using only two wires: serial data line (commonly noted as SDA) and serial clock line (SCL). To make this possible, each device connected to the bus has to be addressable by its own unique address. Also, every byte transmitted is followed by an Acknowledge (ACK) or Not Acknowledge (NACK) bit from the receiving side, confirming (or not) that the byte was successfully received and the transmission can continue.

MSP

The MultiWii Serial Protocol is a widespread standard for communication with flight controllers over UART. MSP was originally part of the MultiWii flight controller firmware. The data is transmitted over MSP in frames, which are structured as shown in figure 2.8.

The header contains a preamble ("$M") and information about the direction of the transmission in a form of a "<" (to FC) or ">" (from FC) character. The next byte contains the length of the payload. The 5th byte is the message identifier, defining the type of message (based on a pre-defined list of message types). What follows is the payload itself ended with a checksum byte (security measure). At the time of writing, there are about 38 message types documented, ranging from raw IMU data requests and stick commands to control algorithm adjustments and GPS data requests.

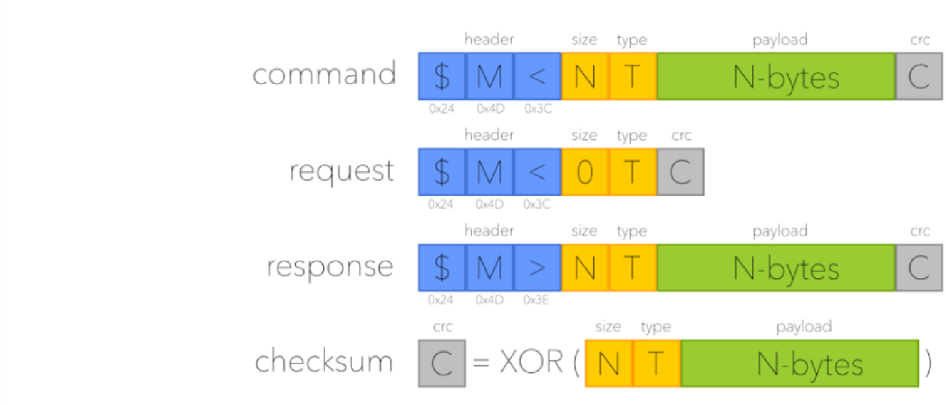

Figure 2.8: MultiWii Serial Protocol frame format [8]

PPM

The Pulse Position Modulation is a signal modulation technique that uses pulses of a fixed width and the information transferred is encoded as time delays between the pulses. Its main use is in optical communication, although this description will focus on its specific use for communication of radio control (RC) receivers with flight controllers or other microcontrollers. The signal in this case has a form depicted in figure 2.9.

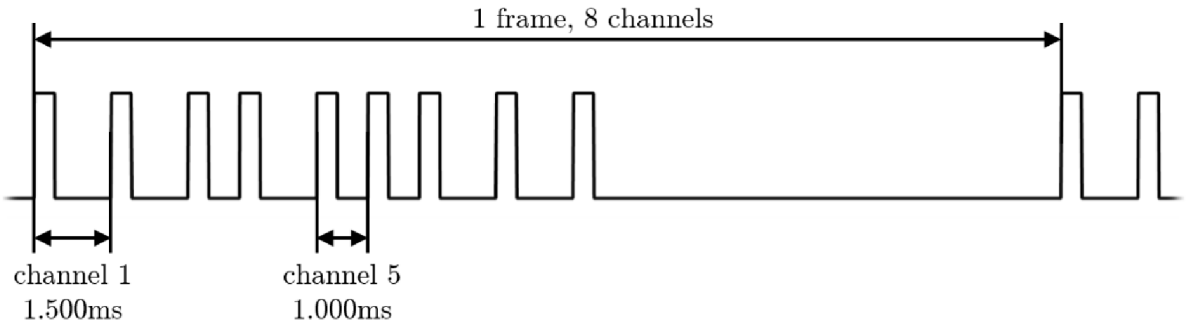

Figure 2.9: PPM signal outputted by an RC receiver

Every frame (usually 20ms long, varies by manufacturer) can contain up to 8 channels separated by fixed-width pulses (in this instance 0.4ms long). The value of every channel is the time delay between the pulses' rising edges and ranges from 1ms (0%) to 2ms (100%). In the example noted in fig. 2.9, the first channel has a value of 50%, while the fifth channel has a value of 0%. This percentage usually represents the amount of stick deflection of the transmitting radio controller.

PWM

The Pulse Width Modulation is a signal modulation technique that encodes the information transferred into the pulse length. The typical form of PWM signal, used in, for example, switched-mode power supplies or motor drivers, transfers a single value as a percentage of the time the signal is high. However, PWM signals are also often used as control signals, having a fixed-length time frame of 20ms and a pulse width ranging from 1ms (0%) to 2ms (100%). The PWM used specifically on the flying platform to transfer values from the FC to the ESCs follows a OneShot125 protocol, which uses pulses 125μs

to 250μs long, allowing for refresh rates of up to 4kHz. It also does not have a constant time period, as it waits for the next update from the FC and only then sends the value. This FC-ESC synchronization results in a reduced control delay. Such kind of signal is visualized in fig. 2.10.

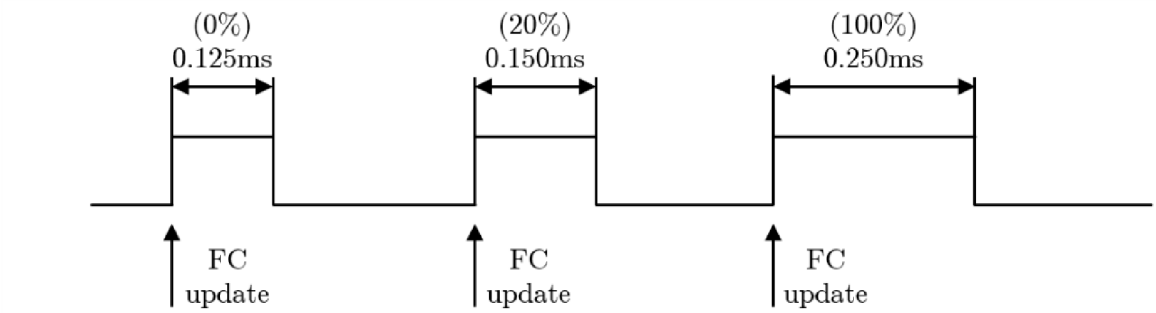

Figure 2.10: An example of the PWM signal under the OneShot125 protocol

ROS

ROS stands for Robotic Operating System, but rather than an operating system, it is a framework and a set of tools and libraries. ROS is completely open-source and lightweight and has been implemented in many programming languages. The system presented in this thesis takes advantage of the fact that ROS can connect any two processes (called nodes) that are on the same network. The ROS communication infrastructure offers several types of communication between the nodes: services (synchronous request-response communication), topics (asynchronous streaming) and storing data on a Parameter Server. The communication style used in this case are topics.

Topics are a unidirectional way for nodes to exchange messages anonymously and independently of each other in a publisher/subscriber fashion. During its initialization, every node has to register with the ROS master using a unique node name. ROS master then keeps track of publishers and subscribers to topics and enables the nodes to allocate each other. Firstly, a node notifies the master that it wants to publish messages to a certain topic. Once another node subscribes to that topic, the data stream starts flowing peer-to-peer. Every topic can have multiple publishers and/or multiple subscribers. The specific use of ROS topics on the flying platform is explained in greater depth in chapter 2.2.

Trigger/Echo

Many ultrasonic rangefinders, including the US-100, HC-SR04 and SRF05, use a simple interface consisting of an input "trigger" pin and an output "echo" pin. To start a distance measurement, a 5V pulse at least 10μs long has to be applied to the trigger pin. The rangefinder then emits eight 40kHz pulses and measures the time it takes for the acoustic waves to deflect from any objects in its way and then return back to the sensor. Lastly, the rangefinder holds the echo pin in a HIGH state proportionally to the time recorded. Given that the speed of sound is about $340m/s$ and that the sound wave goes across the distance 2 times, the value calculated from the measurement in figure 2.11 is

$$distance = time \cdot velocity = \frac{503.89 \cdot 10^{-6}}{2} \cdot 340 = 0.0857m = 8.57cm \qquad (2.1)$$

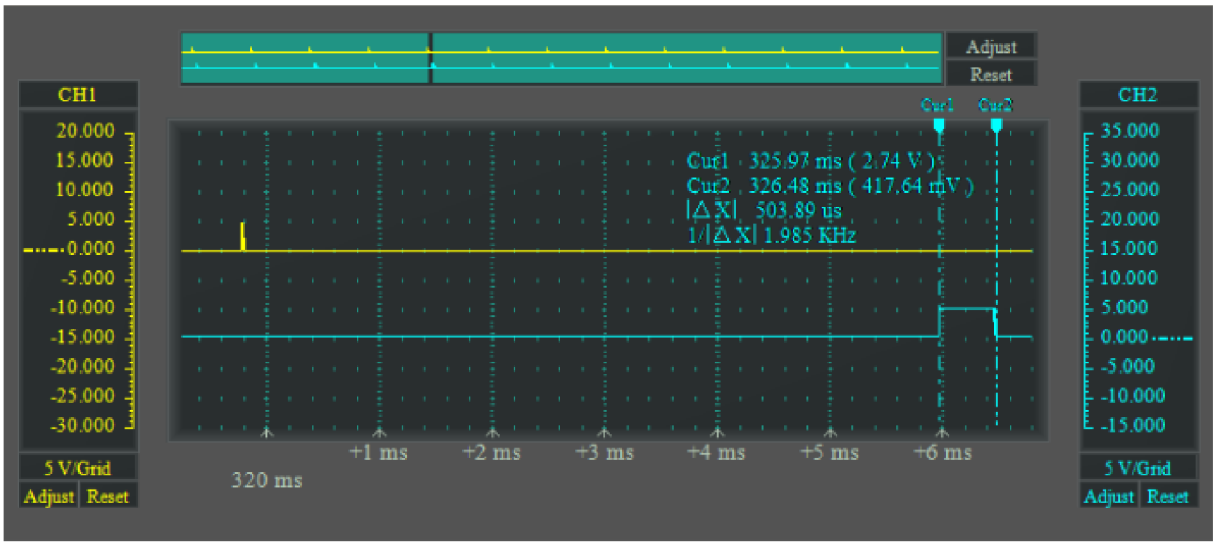

Figure 2.11: An oscilloscope screen shot showing the trigger (CH1) and echo (CH2) signal

2.2. Software Architecture

Both Arduino Nanos, as well as the Raspberry Pi, are running a custom software written in C for Arduinos and Python for the Raspberry Pi. In addition, a laptop is connected to the microcomputer's Wi-Fi and is used to run the ROS master (roscore) and record data for later analysis. The programming language of choice, Python, unlike for example C++, is an interpreted programming language offering reduced complexity, garbage collection, high-level native data types without the need to declare them and overall user-friendly syntax, making it the first choice for many program and web-app developers. The broad range of libraries and scientific tools together with no need for compiling are also the reasons for this language being used for scientific research, development and rapid prototyping. The Raspberry Pi runs two Python scripts in particular; the first script is responsible for image processing and position control, the second script switches between the operator's stick input and the position regulator's output. The software structure is laid out in figure 2.12.

The "image_and_control.py" script is executing a loop with a more-or-less constant frequency of 10Hz maintained by the rospy library's "rate.sleep()" feature, which holds the script idle in case the loop finishes earlier than in the desired time period. The frequency was chosen for reasons detailed in chapter 3.1. At the beginning of the loop, image data is acquired from the Raspberry Pi camera module to be processed by the ArUco augmented reality library (a sub-library of the CV2 library). A set of commands from the ArUco library is executed, first detecting potential ArUco markers of the ChArUco board, then refining the candidates, interpolating ChArUco corners and finally estimating the board's pose (position and attitude). Next, given that the library yields a valid result, the position control algorithm calculates the stick command needed to eliminate the control error (the algorithm is described in depth in chapter 3).

Inside of this script is present a ROS node (via the rospy library) which publishes the position data, the stick commands and an information about whether the position control algorithm is receiving useful data. All this information is published as a single message into a ROS topic. The topic has 2 subscribers in total; a node running in the second

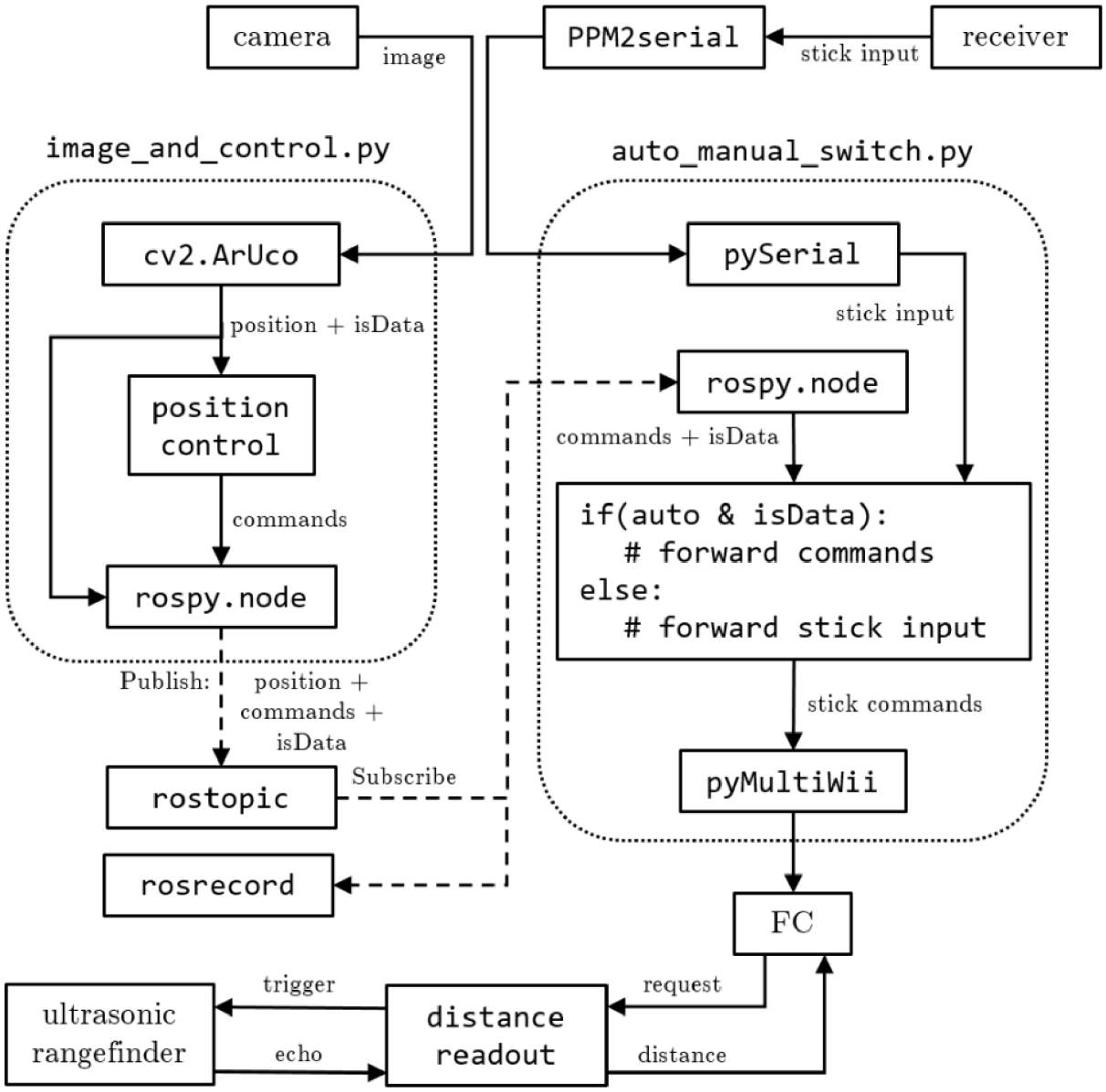

Figure 2.12: A block diagram showing the custom software architecture

Python script and a node running in the laptop terminal (this node performs recording of the messages for debugging). As noted in chapter 2.1.2, the ROS master running in another terminal on the laptop acts as a middleman only in the initial phase when the publishing node needs to be connected to the subscribing nodes. Besides creating a means of intercepting the data flow, the use of ROS topics also solves a problem of the first and second script using loops running with different frequencies.

After decoding the stick input values from the PPM signal, the Arduino Nano sends them to the "image_and_control.py" script thanks to the pySerial library. One of the PPM channels contains a value representing a position of one of the radio controller's position switches. This switch is used by the operator to switch between "auto" (as in automatic) and manual mode of flight. The manual mode is engaged by the operator only prior to landing and in emergency situations. If the switch is in the auto position AND the position control algorithm is being provided with position data, the loop in this script forwards the control commands to the FC via the pyMultiWii library. If either of the

conditions is not fulfilled, the script forwards the stick input (manual mode). That makes it possible to manually navigate the quadrotor towards the ChArUco board and for the position controller to take over only when it has sufficient position data. In the same way, if the quadrotor happens to lose the target, the script starts forwarding operator's inputs.

Moreover, to ensure seamless transition between the stick input and the position controller's command and to prevent the aircraft from sudden and exaggerated movements, a measure displayed in fig. 2.13 was introduced. If the control command significantly varies from the previous value, the previous value will be slightly changed towards the command value. Given that the switching loop runs with a frequency of 400Hz and *alfa=0.5*, the complete transition would take 0.4s in an extreme case of a 100 points difference.

```
if abs(xC-aileron)>20:
    aileron = aileron+np.sign(xC-aileron)*alfa
else:
    aileron = xC
if abs(yC-elevator)>20:
    elevator = elevator+np.sign(yC-elevator)*alfa
else:
    elevator = yC
```

Figure 2.13: A transition measure preventing sudden changes in stick commands

The remaining custom software is the code deployed on the second Arduino Nano; about every 76ms, distance measurement is triggered and read out. The distance is then sent to the FC when requested. Additionally, the built-in LED on the Arduino board lights up when the distance measured is less than 100cm, providing the user with a simple way of troubleshooting the platform.

3. Position Sensing And Control

For the position sensing and control, a Cartesian coordinate system with the origin in the center of the target was used, as illustrated in figure 3.1. The Y axis is oriented in the forward direction (*pitch+*) of the flying platform and the position in this axis is controlled by the elevator stick command, while the X axis is oriented in the right direction of the quadcopter (*roll+*) and the position in this axis is controlled by the aileron stick command. As described in previous chapters, the control in the Z axis is done by the flight controller's firmware's built-in surface following feature.

The following chapters focus on describing the ArUco position estimation process and its properties and caveats that were discovered during testing and subsequently compensated for, as well as on describing the development of the position control algorithm and the results achieved with the solution implemented.

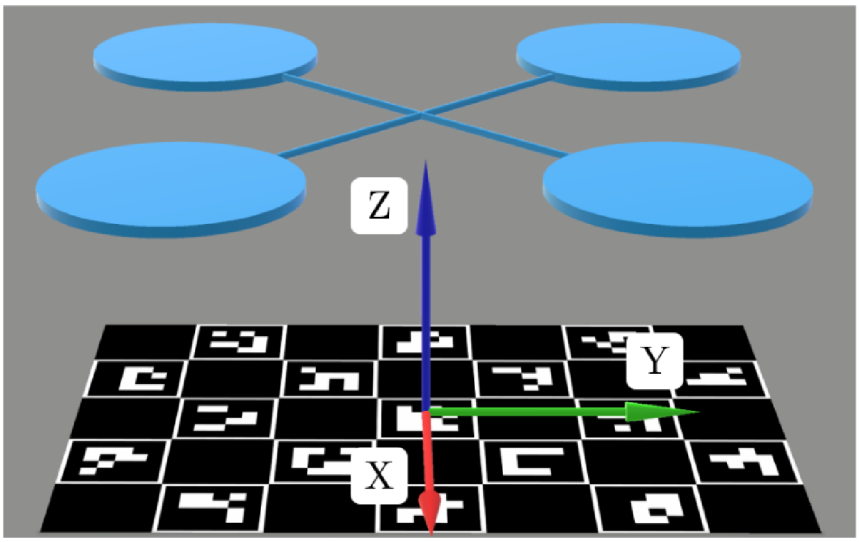

Figure 3.1: An illustration of the world coordinate system

3.1. ArUco pose estimation

ArUco is a popular open-source augmented reality library based on [9] and [10] and is now a part of the OpenCV library as a module. It is used to generate special binary square fiducial markers (up to 1024 unique markers) and detect them in images. To estimate the pose of a camera with respect to ArUco markers, the algorithm has to be provided with calibration parameters that describe the properties of the camera's optics. OpenCV offers an interactive calibration tool, making it fairly easy to obtain the camera's optical parameters using either dual circles pattern or a ChArUco board. ChArUco is a chessboard with ArUco markers interspersed inside and provides more stable and reliable edge recognition than a bare ArUco marker grid.

After markers are detected and refined, the corners of the ChArUco board can be interpolated and the board's pose estimated. The resulting translation and rotation vectors represent the position and attitude of the board in respect to the camera and can be used for projection on the board in real time as shown in fig 3.2.

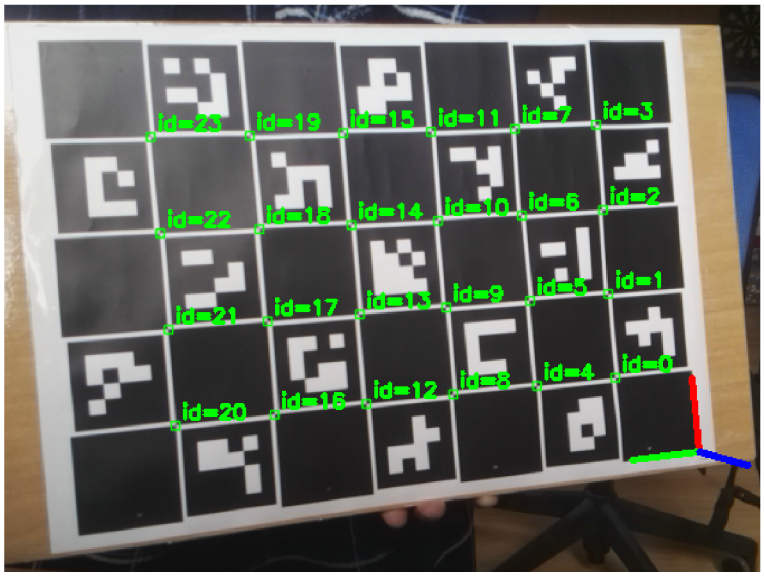

Figure 3.2: A projection of axes and corner IDs on the detected ChArUco board

When assessing the accuracy of the pose estimation, the Z axis measurements appeared to suffer from an error that was not constant, unlike the measurements in X and Y directions which have only an insignificant offset. The Z axis position, which corresponds to the distance from the ChArUco board, deviates from the ground truth in a more-or-less linear fashion, as depicted in figure 3.3. By applying linear regression to the relation between the measured position and the ground truth, a simple equation can be obtained for retrieving a more accurate Z position value.

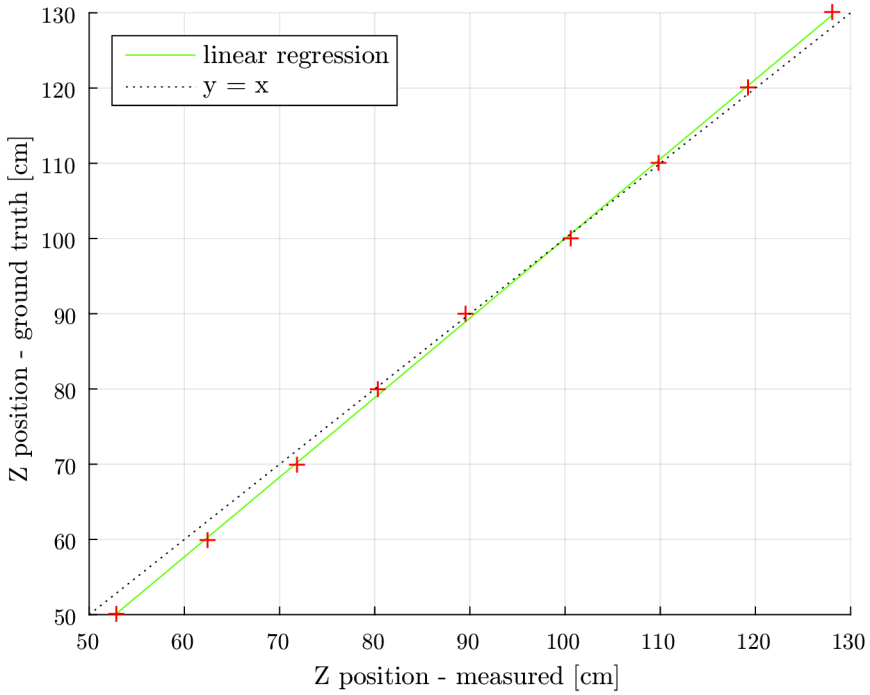

Figure 3.3: Graph showing the relation between the real and the measured Z position

This relation is different for every combination of markers and had to be measured every time a different size or scale of the ChArUco board was chosen during the experimentation phase. In the case of the board used for recording the final results, the particular form of the equation is:

$$Z_{real} = 1.0577485523 \cdot Z_{meas} - 5.7732550258cm \tag{3.1}$$

Despite the final Z axis value not being used in the control loop, the primary Z axis value, which represents the distance of the marker from the camera in the coordinate system originating from the camera, is important for the subsequent calculations.

Since the position control requires information about the camera's position in the world coordinate system, the data gained in the aforementioned steps have to be transformed from their camera-centered coordinate system. First, the rotation vector is converted into a rotation matrix R via the Rodrigues function. Then, the position C of the camera in world coordinates is calculated from the R and the vector of translations T as follows (notation ^{T} stands for transpose):

$$C = -R^T T \tag{3.2}$$

To determine the optimal frequency for the position sensing and control loop, a series of measurements was carried out recording the performance of the Python loop running on the Raspberry Pi microcomputer with various degrees of processing load. The results are summarized in figure 3.4. The speed of the marker detection and ChArUco pose estimation greatly varies and depends on many factors, such as the number of markers in the view, the distance of the board from the camera, the angle of the board, etc. Considering that the position control algorithm takes a negligible processing time in comparison to the image processing algorithm, a frequency of 10Hz was chosen for this loop. As explained in chapter 2.2, if the iteration of the loop finishes earlier than in 100ms, the rospy library will wait for the rest of the time period before allowing the loop to continue. If the iteration takes longer than 100ms to finish, a new iteration starts right away.

processes running in Python:	**Frames Per Second**
image capture	26.4
image capture + marker detection	11.1-13.6
image capture + marker detection + ChArUco pose estimation	10.2-13.6

Figure 3.4: Performance of a Python loop running on the Raspberry Pi microcomputer

Unfortunately, such a low-frequency video system is more prone to data outages, as one invalid image results in at least 100ms of no data being inputted into the position controller. Invalid result of the marker detection can be caused by motion blur or simply by the ChArUco board temporarily exiting the field of view of the camera. For these reasons, the *isData* parameter had to be designed not merely to hold a binary value, but to rise from 1 to a certain level instead. While this parameter is greater then 1, the last output of the position control algorithm is sent as a command. The maximum value of the parameter is 5, adding a buffer of up to 400ms for the ArUco library (the value increases by 1 on every iteration). The usefulness of this measure is demonstrated in figure 3.5, where the regions highlighted in red represent frames with no position data.

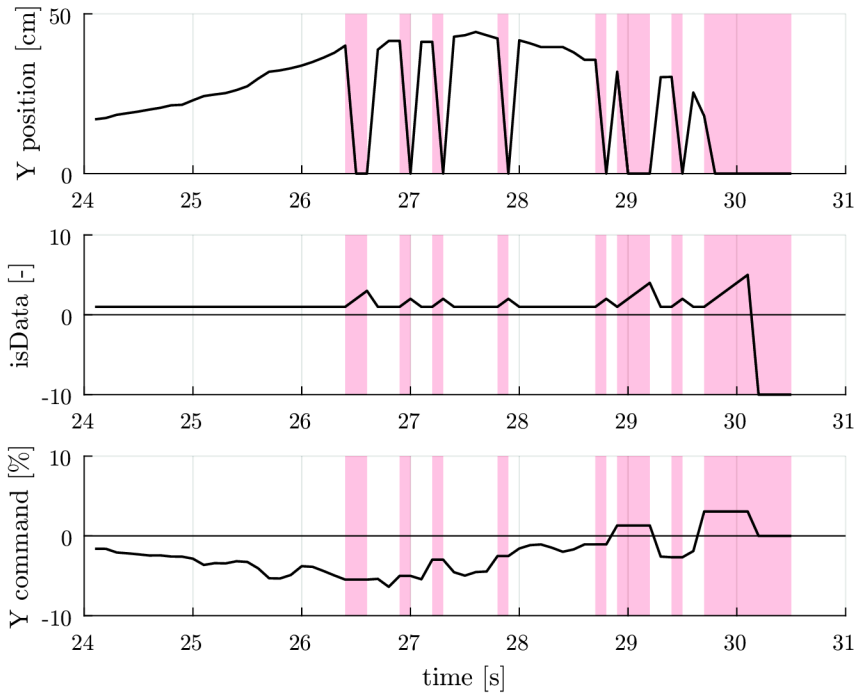

Figure 3.5: A flight recording showcasing the ability to minimize the position error despite position data outages (marked red)

Furthermore, the data acquired from the ArUco library are accompanied by noise-like inconsistencies that grow stronger with greater distance of the camera from the board. Taking into account the relatively small field of view of the Raspberry Pi camera module (60°x40°) and the resulting need to maintain a sufficient distance from the marker to keep it in view while moving around, filtering is inevitable.

The filter of choice is a median filter as a simple yet effective means of reducing noise. Specifically, a median of the last 4 values is used as the input into the position control algorithm. The same filter and number of samples is applied to the derivatives (velocities). However, the existence of an array of the last 4 values causes issues upon detecting the position and velocity for the first time after a period of no markers being detected, especially if the previous values vastly differ from the current values. The median filter yields the ”old” measurements (or values close to them) temporarily and as a consequence, a spike in velocity is typically seen on the output, often three to four times larger than the actual velocity.

To prevent the position control algorithm from reacting to this false readout, the *isData* parameter does not start at 0, but starts at -10 instead. Therefore, the position control does not engage until about a full second after the first marker recognition, ensuring that the position and velocity data are stable and reliable. An example of such behavior is shown in figure 3.6. Combined with the transition measure described in chapter 2.2, the system is capable of a smooth progression from the manual navigation towards the target to the automatic navigation when the ChArUco board is detected.

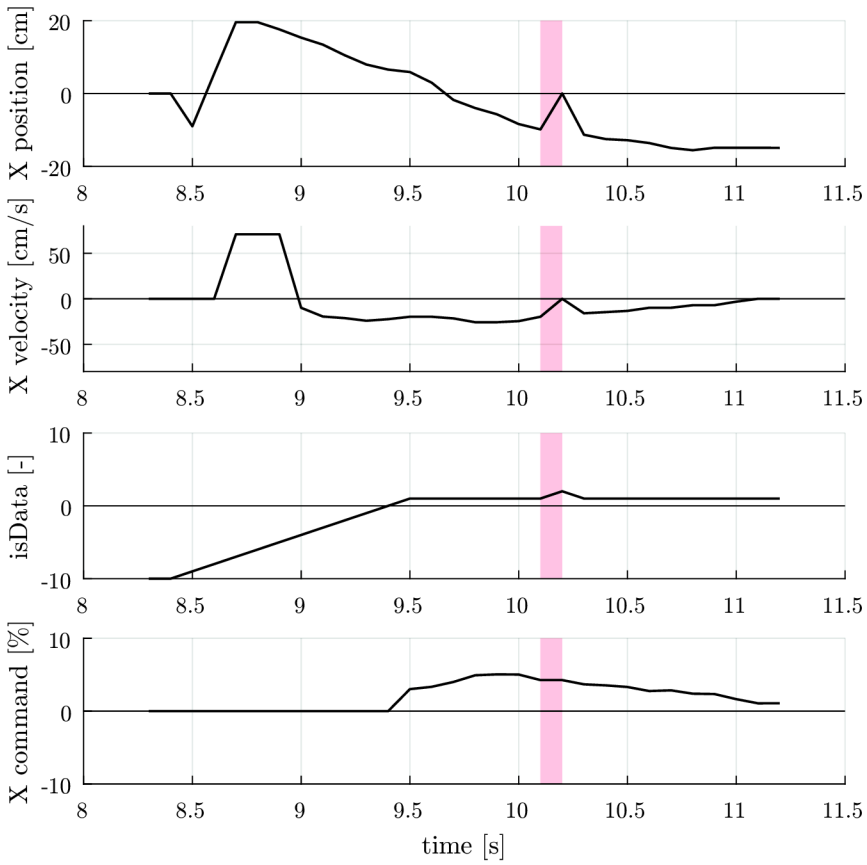

Figure 3.6: A flight recording showing the false peak in velocity and the position control algorithm engaging only after the data readouts have settled

3.2. Position control algorithm

One of the most basic concepts in control theory is a closed-loop negative feedback control system. As can be seen in figure 3.7, the measured process variable (output of the plant recorded by the sensor) is subtracted from the setpoint, which is the desired output of the whole system. The result of this subtraction is the error signal, based on which the controller decides about the value of the manipulated variable. The plant represents the dynamic system or process to be controlled. The output of the system, which is measured by the sensor and fed back, depends on the action signal from the controller and the dynamic properties of the plant.

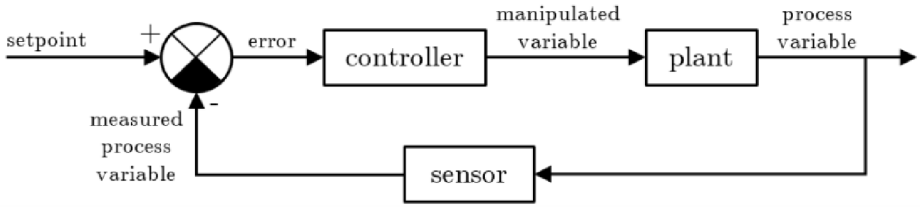

Figure 3.7: A closed-loop negative feedback control system

In the case of this thesis, the setpoint is position [0,0], the controller is represented by the position control algorithm, the manipulated variables are the stick commands, the

plant is the quadrotor with its actual position being the processing variable and the sensor consists of the camera and the ArUco library.

The stick command translates directly into the tilt angle of the aircraft around the given axis. As is illustrated in figure 3.8, the tilt results in the thrust force F being distributed among the the X and Z axes (2D space is used for simplicity). Assuming that the tilt angle is very limited, small-angle approximation can be used to substitute $sin(\alpha)$ with α. The small angle also allows for approximating the vertical force as $F_z \approx F$. Since the quadcopter is supposed to be static along the Z axis, the thrust can be expressed as $F = mg$, where m is the mass of the platform and g is gravitational acceleration. It is also assumed that the effect of air resistance is negligible, as the platform performs only relatively slow movements. Similar assumptions were made for example in [11].

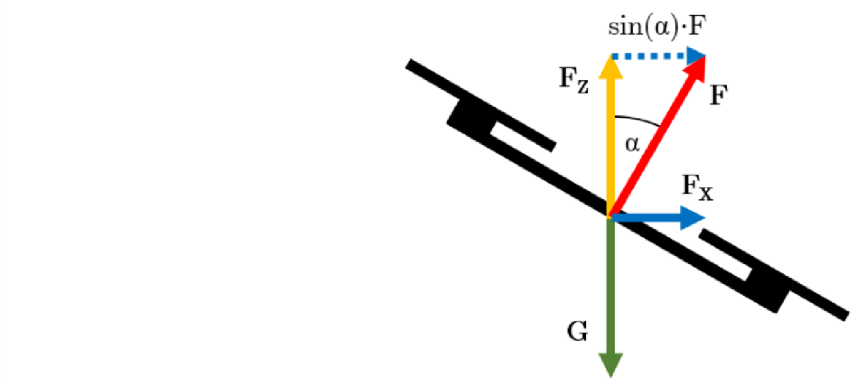

Figure 3.8: An illustration of the force distribution resulting from the platform tilting

To obtain the transfer function describing the relation between the stick command (input) and the resulting position of the aircraft (output), the following series of steps has to be realized, in which X_c is the stick command in the X axis (roll), a_x stands for acceleration in the X axis and k is a constant for converting the stick command into tilt angle in radians:

$$\alpha = k \cdot X_c, \tag{3.3}$$

$$F_x = \alpha \cdot mg, \tag{3.4}$$

$$a_x = \frac{F_x}{m} = \alpha \cdot g = k \cdot X_c \cdot g. \tag{3.5}$$

Because acceleration is a second derivative of position, Laplace transformation yields

$$a_x(s) = s^2 \cdot x, \tag{3.6}$$

where the parameter s expresses derivation. By substituting a_x with the result from equation 3.5

$$k \cdot X_c \cdot g = s^2 \cdot x, \tag{3.7}$$

the transfer function Q of the quadcopter system can be calculated:

$$Q(s) = \frac{x}{X_c} = \frac{k \cdot g}{s^2}. \tag{3.8}$$

3.2.1. Proportional control

The simplest type of controller is a proportional controller. The error is only multiplied by a constant P (as in proportional) and then directly inputted into the plant. Position control using only P gain was simulated and the result displayed in figure 3.9.

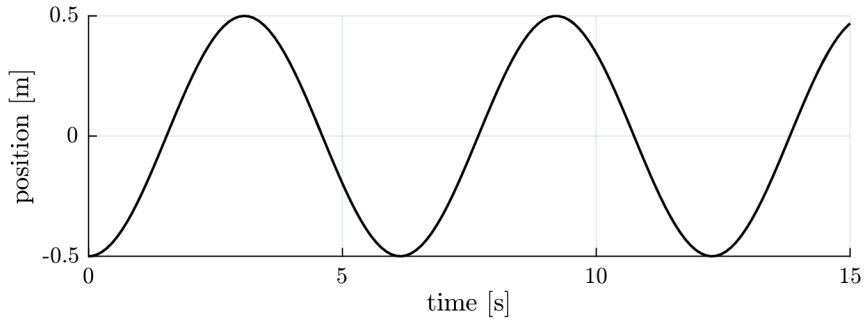

Figure 3.9: Position data from the simulation of proportional control

It is apparent that such type of controller is unsuitable, as the the model oscillates with the amplitude equal to the initial position without any dampening. The reason is that the quadcopter transfer function is a second-order system as it contains two integrating elements (s^2). To be able to control the system, the controller must have a derivative term.

3.2.2. Introducing derivative term

To implement a derivative into the controller in an easy-to-comprehend fashion, an inner loop velocity control concept was chosen. The first proportional term together with the position error determines the setpoint for the inner loop, which acts essentially as a velocity controller. The system laid out in figure 3.10 describes the final version of the controller used (including the median filters), although slightly more complicated systems were tested in chapter 3.2.3.

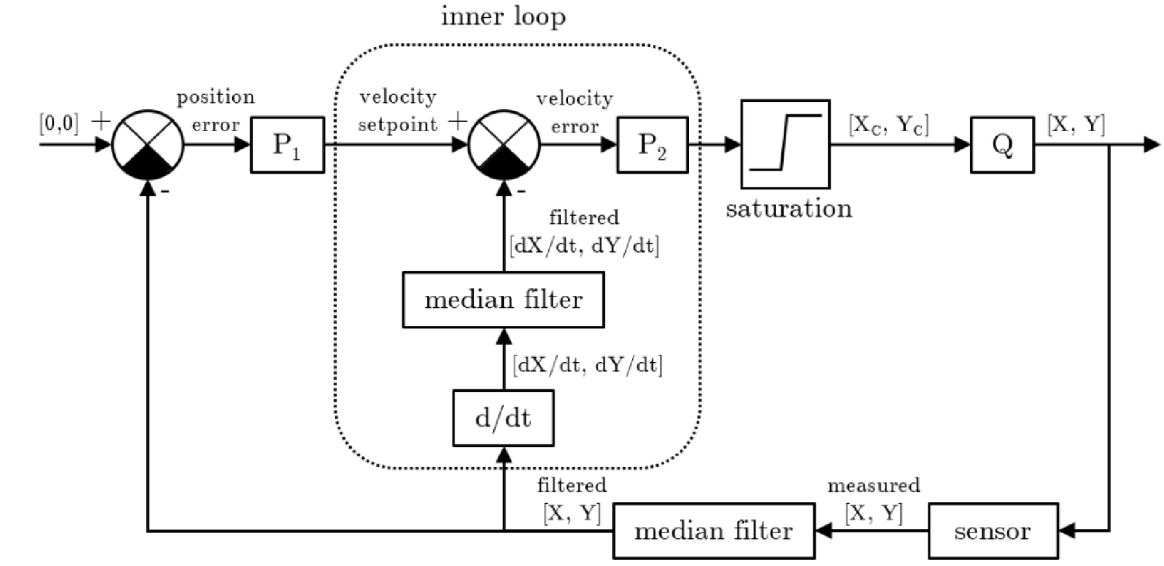

Figure 3.10: Position control system with inner loop velocity control

Based on the current difference of the desired position and the perceived position, the desired velocity is established via the P_1 gain. After filtering the position data, calculating the derivative and filtering it again, the velocity error is multiplied by the P_2 gain. Before the stick commands are sent to the plant, they are passed through a saturation function, limiting the controlled value to a safe level.

The behavior of the controller was verified in a simulation with the P_1 gain set to 0.5, meaning that when the platform is 30cm from the origin of the coordinate system, the setpoint for the velocity controller will be 15cm/s. The data plotted in figure 3.11 were recorded with the P_2 gain holding a value of 1.4. The saturation function was limiting the output to 10% to ensure that the tilt angle won't surpass a threshold of the maximum angle for small-angle approximation.

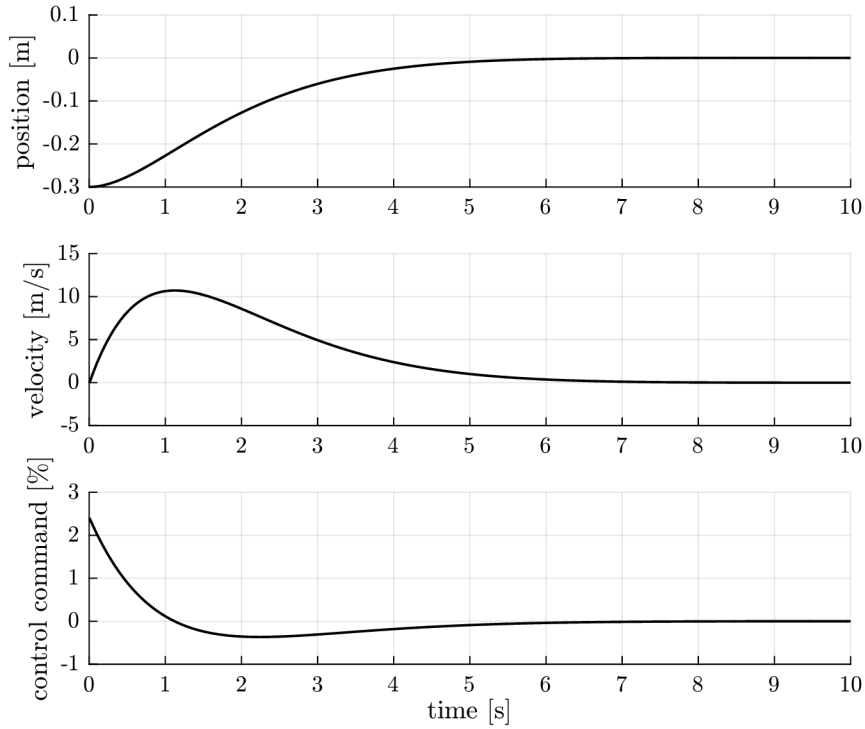

Figure 3.11: Data from the simulation of the position control with inner loop velocity control

3.2.3. Testing

The tests were carried out in a local gymnasium and took over 60 hours. The first control algorithm that was used was an exact implementation of the controller modeled in chapter 3.2.2 with the exception of the P_2 gain, which was initially lowered before the basic functionality was tested on the real system.

In every flight, the target was slowly approached in manual mode (i.e. controlled by the operator) and once the system obtained position data, the control was automatically switched from the operator to the position controller. The direction of approach was semi-arbitrary to capture possible differences in flight characteristics related to the direction of the flight of the aircraft.

In many of the trials, the quadrotor seemed to get in the proximity of the ChArUco board with successful velocity reduction, but over time, instead of getting even closer to the target and eliminating the control error, it started retreating, essentially resulting in lost of the visual of the board. An example of such flight can be viewed in figure 3.12.

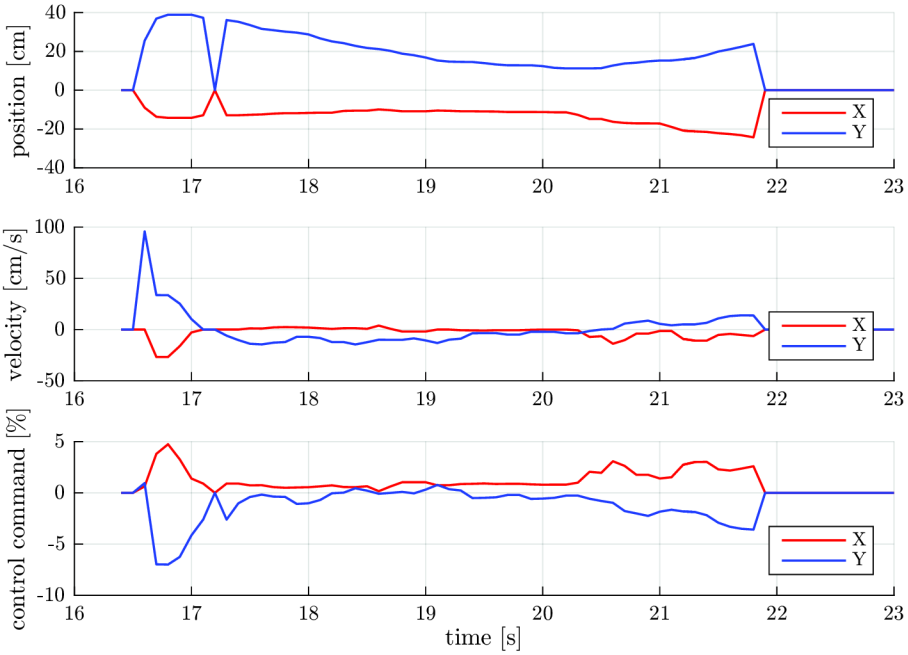

Figure 3.12: An example of the quadrotor retreating after getting in the proximity of the ChArUco board

Assuming that the cause of this behavior was an imperfect calibration of the accelerometer or other constant influence, an integral term was added in parallel with the P_2 gain to help the controller reduce the error when the result yielded by the proportional term was not large enough to counter the disturbance force acting on the aircraft. The I term was implemented at a very low value at first, about 1%, but when raised to a level that would have the desired effect, large overshoots started to occur, as the quadrotor would not start slowing down soon enough. Figure 3.13 shows a recording of a flight in which the velocity does not start decreasing until after the origin of the coordinate system is surpassed, due to the integral component being too prominent.

Several other implementations of the integral term were proposed and tested, including a system that turned the integrator on only once the quadrotor reached a position close to [0,0], the idea being that the initial approach would be controlled only by the proportional terms to ensure a timely velocity reduction. Once the absolute value of the position error decreases beyond a certain level, the integral term is engaged and the integrator is enabled. In theory, this approach should hold the platform in the close vicinity of the target. However, it didn't solve the original problem shown in figure 3.12.

The reason why the P_2 gain was not increased to a degree that would cause an elimination of the position error was the fact that higher numbers led to sudden and exaggerated control commands with the effect of reaching a velocity too great to be lowered before losing the target from view, or too great for the camera to be able to provide useful images consistently.

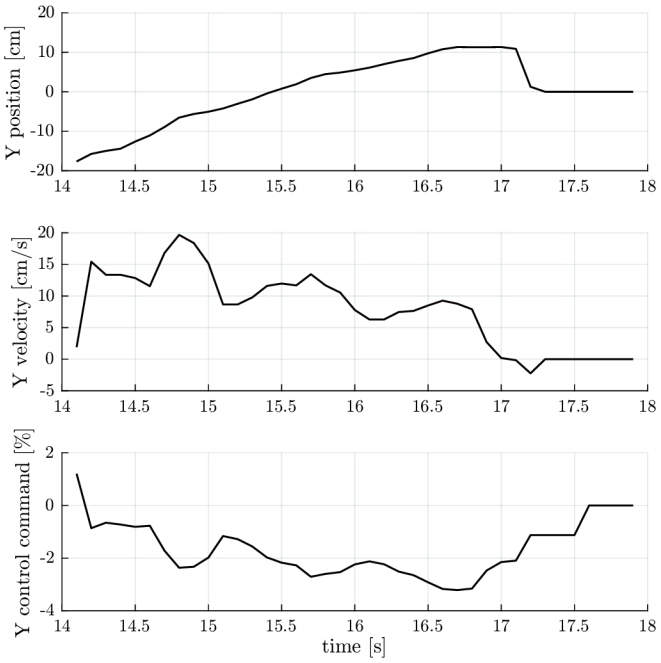

Figure 3.13: A flight recording displaying the unwanted effect of the integral component

Consequently, the whole system present on the flying platform was revised and the transition measure introduced (described in chapter 2.2) and the *isData* parameter was redesigned (described in chapter 3.1) to help the controller mitigate the effect of data outages. Furthermore, the size of the ChArUco board was increased and the number of markers raised. That expanded the area in which the platform can move without losing visual of the board and also improved accuracy of the position estimation.

Altogether, these advancements made it possible to increase the proportional terms to an extent that resulted in successful position error elimination. Figure 3.14 offers the data from the flight that used these new values and is documented on a video recording attached to this thesis. The enlarged board is depicted in figure 3.1 and the measurements shown in figure 3.3 were carried out for this board.

As can be seen, the system oscillates in proximity of the origin of the world coordinate system. This is due to several reasons. Firstly, the frequency of the control loop at 10 Hz is far from ideal for such a dynamic system. The more aggressive tune of the controller has the effect of faster movements and the loop is too slow to be able to control such fast processes. Secondly, the median filters cause a delay of up to 0.3s for the position measurements and up to 0.6s for the velocity measurements. The outcome is the controller acting on values that are “old” and not as relevant. Thirdly, the surface-following feature of the FC’s firmware is not reliable for longer periods of time as the altitude starts drifting slightly. To compensate, the operator has to make adjustments in the set altitude.

Unfortunately, the change to the manual input in altitude is often larger than would be appropriate and causes the quadcopter to significantly change the magnitude of the thrust force, thus significantly changing the magnitudes of the horizontal components of the force and introducing substantial disturbances into the system, which usually leads to increasing the amplitude of the oscillations despite their originally stable nature.

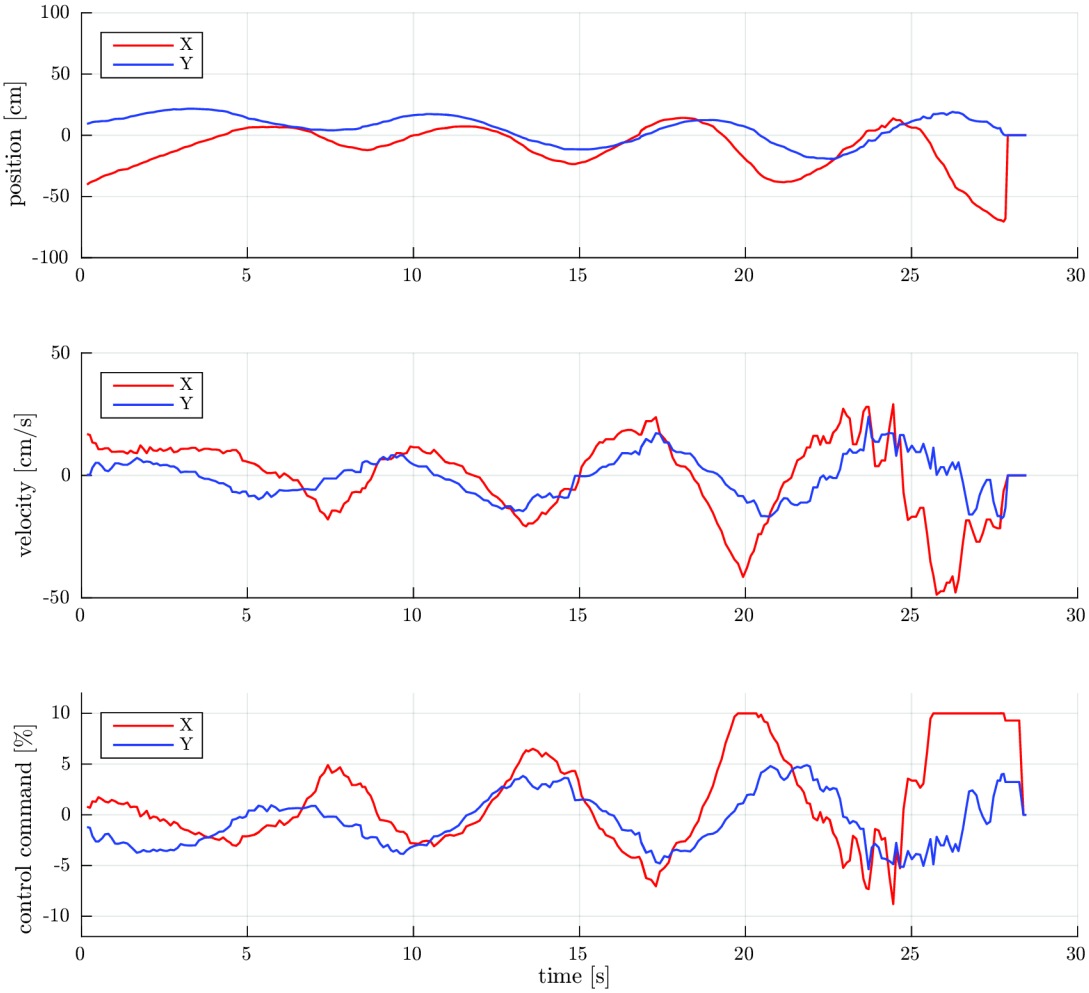

Figure 3.14: Data from the flight recorded in the video attached to this thesis

Lastly, towards the end of the flight, the multirotor ascended to an altitude in which the measurements become unviably inaccurate and subsequently, the derivation becomes unusable due to the higher noise in the position signal. Since the control signal directly depends on the velocity, the controller becomes unable to act accordingly and causes the platform to fly out of range of the camera.

4. Conclusion

The goal of the thesis was to design and implement a quadcopter control system with visual guidance, using low-cost, commercially available hardware and open-source or custom software, with the computations executed entirely on-board and with the capability to hover indoors using basic control principles.

Based on the research of existing systems, a solution was proposed with a description of the units' functions and implemented in a custom-built flying platform. All the units consist of relatively cheap and widely-available components (such as the single-board microcomputer Raspberry Pi 3 Model B or the microcontroller board Arduino Nano) and contain only open-source software or software written by the author of this thesis. The components are described in depth in chapter 2.1.1, while the communications between them are detailed in chapter 2.1.2, including technicalities concerning the versions of signals specific to UAV platforms (as opposed to general description).

The structure and function of the custom software is outlined in chapter 2.2. Its architecture and the use of the Robotic Operating System allows for the position sensing and control loop to run on a different frequency than the loop forwarding the stick commands to the Flight Controller. At the same time, all data can be recorded during flight by a laptop connected to the flying platform's wireless network.

For position sensing, augmented reality library ArUco was chosen. After calibrating the camera and choosing an appropriate marker option, the properties, performance and accuracy of the pose estimation were examined and tested and compensation elements for inaccuracies and issues related to the image processing were designed (see chapter 3.1).

Lastly, a model of the quadcopter system was inferred and multiple control systems were designed, tested in simulation and applied to the flying platform. Throughout the extensive testing of the system in and indoor environment, many measurement were obtained and many steps towards improving the system were carried out. Several datasets are pictured in chapter 3.2.3, including explanations for the signals' characteristics and proposed measures to achieve the desired behavior.

The final version of the system is captured in the video attached to this thesis and corresponds with the very last figure of this thesis (figure 3.14). As stated in chapter 3.2.3, the reason behind the oscillating tendencies is the low frequency of the control loop combined with the delays introduced by the median filters. Future improvements could include predicting the position based on data from the FC's Inertial Measurements Unit and fusing them with the high-delay image data via, for example, an Extended Kalman Filter.

Bibliography

[1] OptiTrack for Robotics. *Optitrack* [online]. USA: Inc. DBA OptiTrack, 2018 [cit. 2018-05-04]. Available from: `https://optitrack.com/motion-capture-robotics/`

[2] KENDALL, Alex, Nishaad N. SALVAPANTULA a Karl STOL. On-board object tracking control of a quadcopter with monocular vision. In: *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings* [online]. Orlando, FL, USA: IEEE, 2014, May 2014, p. 404-411 [cit. 2018-05-06]. DOI: 10.1109/icuas.2014.6842280. ISBN 978-1-4799-2376-2. Available from: `https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6835384`

[3] WANG, Chaolei, Tianmiao WANG, Jianhong LIANG, Yang CHEN, Yicheng ZHANG a Cong WANG. Monocular visual SLAM for small UAVs in GPS-denied environments. In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)* [online]. Guangzhou, China: IEEE, 2012, p. 896-901 [cit. 2018-05-06]. DOI: 10.1109/ROBIO.2012.6491082. ISBN 978-1-4673-2125-9. Available from: `https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6482476`

[4] SCHAUWECKER, Konstantin a Andreas ZELL. On-Board Dual-Stereo-Vision for the Navigation of an Autonomous MAV. In: *Journal of Intelligent & Robotic Systems* [online]. Dordrecht: Springer Netherlands, 2014, **74**(1), 1-16 [cit. 2018-05-07]. DOI: 10.1007/s10846-013-9907-6. ISSN 0921-0296. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.400.6493&rep=rep1&type=pdf`

[5] BOŠNAK, Matevž, Drago MATKO a Sašo BLAŽIČ. Quadrocopter Hovering Using Position-estimation Information from Inertial Sensors and a High-delay Video System. *Journal of Intelligent & Robotic Systems* [online]. Dordrecht: Springer Netherlands, 2012, **67**(1), 43-60 [cit. 2018-05-07]. DOI: 10.1007/s10846-011-9646-5. ISSN 0921-0296. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.7278&rep=rep1&type=pdf`

[6] UPTON, Eben. Raspberry Pi 3 Model B+ on sale now at $35. In: *Raspberry Pi Blog* [online]. UK: Raspberry Pi Foundation, 2018 [cit. 2018-05-12]. Available from: `https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/`

[7] Raspberry Pi 3 Model B. In: *Raspberry Pi Products* [online]. UK: Raspberry Pi Foundation, 2016 [cit. 2018-05-12]. Available from: `https://www.raspberrypi.org/app/uploads/2017/05/Raspberry-Pi-3-hero-1-1571x1080.jpg`

[8] COTTAFAVI, Stefano. MSP – The MultiWii Serial Protocol. In: *A.Quarter.To.Seven* [online]. [cit. 2018-05-10]. Available from: `http://www.stefanocottafavi.com/wp-content/uploads/2015/05/MultiWii_MSP_graphics.png`

[9] GARRIDO-JURADO, S., R. MUÑOZ-SALINAS, F.J. MADRID-CUEVAS a M.J. MARÍN-JIMÉNEZ. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* [online]. Elsevier, 2014, **47**(6), 2280-2292 [cit. 2018-05-21]. DOI: 10.1016/j.patcog.2014.01.005. ISSN 0031-3203. Available from: `https://www-sciencedirect-com.ezproxy.lib.vutbr.cz/science/article/pii/S0031320314000235`

[10] GARRIDO-JURADO, S., R. MUÑOZ-SALINAS, F.J. MADRID-CUEVAS a R. MEDINA-CARNICER. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition* [online]. Elsevier, 2016, **51**, 481-491 [cit. 2018-05-21]. DOI: 10.1016/j.patcog.2015.09.023. ISSN 0031-3203. Available from: `https://www.sciencedirect.com/science/article/pii/S0031320315003544`

[11] REIZENSTEIN, Axel. *Position and Trajectory Control of a Quadcopter Using PID and LQ Controllers*. Sweden, 2017, 69 p. Master of Science Thesis. Linköping University.

5. List of abbreviations used

UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System
SLAM	Simultaneous Localization and Mapping
EKF	Extended Kalman Filter
IMU	Inertial Measurements Unit
FC	Flight Controller
ESC	Electronic Speed Controller
PC	Personal Computer
PCB	Power Distribution Board
I^2C	Inter-Integrated Circuit
MSP	MultiWii Serial Protocol
ROS	Robotic Operating System
PPM	Pulse Position Modulation
PWM	Pulse Width Modulation
USB	Universal Serial Bus
CSI	Camera Serial Interface
FPS	Frames Per Second
RC	Remote Control
MCU	MicroController Unit

6. List of attachments

Attachment A: Video CD