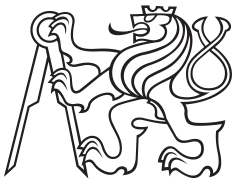
Master's Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Advanced payload control on DJI platforms using embedded on-board computer

Bc. Adam Svoboda

Supervisor: Ing. Tomáš Meiser
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
May 2018

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Svoboda Adam** Personal ID number: **420298**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Advanced payload control on DJI platforms using embedded on-board computer

Master's thesis title in Czech:

Pokročilé řízení užitečného zatížení palubním počítačem na platformách DJI

Guidelines:

1. Study manuals and reference documentation for target embedded computational unit and DJI autopilots communication API
2. Develop Open Embedded system configuration to provide OS for target embedded platform providing all necessary communication interfaces.
3. Develop communication layer based on DJI autopilots API to provide flight control and telemetry acquisition interfaces.
4. Run integrated system for visual data acquisition based on flight management using DJI UAV platform and digital camera sensor.
5. Evaluate and visualize results of realized experiments
6. Provide system documentation and users manuals

Bibliography / sources:

[1] Marwedel, Peter. Embedded system design. Vol. 1. New York: Springer, 2006.
[2] Lee, Edward Ashford, and Sanjit A. Seshia. Introduction to embedded systems: A cyber-physical systems approach. MIT Press, 2016.
[3] Marques, P., Da Ronch, A.: Advanced UAV Aerodynamics, Flight Stability and Control: Novel Concepts, Theory and Applications, John Wiley & Sons, 2017.

Name and workplace of master's thesis supervisor:

Ing. Tomáš Meiser, Department of Computer Science and Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **16.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Tomáš Meiser	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my very profound gratitude to my supervisor, Ing. Tomáš Meiser, for the help, support, and guidance he gave me through the time of the research. Without his useful comments and remarks, the thesis could not have been accomplished.

I would also like to thank my family, my girlfriend and my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of writing this thesis.

Thank you.

Declaration

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references.

In Prague on the 24. May 2018

.....

Abstract

Drones have potential to be used in almost every field of science and technology, for example, for the creation of 3D models using the photogrammetry method. For that purpose, it is important to capture a target area in detail and add the most accurate GPS coordinates to each captured photograph. While modern autopilot can handle a payload, such as a camera, the efficiency, and accuracy of this control is not enough.

This project aimed to develop a special control unit that would take control of the autopilot and the camera and ensure a more accurate and efficient collection of photographic data. The control unit, a Gumstix COM, was added on board the DJI Matrix 600 hexacopter. The project included configuration settings for building the Linux operating system and developing control applications using DJI development kit. These procedures are documented in the corresponding chapters of the diploma thesis.

The developed application was tested in both the DJI simulation environment and the field. The obtained data demonstrated the functionality of all the modules created, including the flight path planner and the data analyzer.

The application completely fulfilled the specifications and is ready for use. At the end of the diploma thesis, there are presented some possibilities of further development of the application.

Keywords: image data acquisition, DJI, autpilot, Gumstix, Yocto Project

Supervisor: Ing. Tomáš Meiser

Abstrakt

Drony mají potenciál na využití téměř v každém odvětví vědy a techniky, například při tvorbě 3D modelů metodou fotogrammetrie. Pro účely fotogrammetrie je důležité podrobně nafotit cílovou oblast a ke každé pořízené fotografii doplnit co nejpřesnější GPS souřadnice. Moderní autopiloty sice dokáží řídit užitečné zatížení, například fotoaparát, ale efektivita a přesnost tohoto řízení není dostatečná.

Tento projekt si vzal za cíl vyvinout speciální řídicí jednotku, která by převzala kontrolu nad autopilotem i fotoaparátem a dokázala zajistit přesnější a efektivnější sběr fotografických dat. Jako řídicí jednotka byl použit Gumstix COM, který byl přidán na palubu hexakoptéry DJI Matrice 600. Součástí projektu bylo nastavení konfigurace pro sestavení operačního systému Linux a vývoj řídicí aplikace s využitím vývojových nástrojů DJI. Tyto postupy jsou zdokumentovány v příslušných kapitolách diplomové práce.

Vyvinutá aplikace byla otestována jak v simulační prostředí DJI, tak i přímo v terénu. Na získaných datech byla předvedena funkčnost všech vytvořených modulů, včetně plánovače letové trasy a analyzátoru nasbíraných dat.

Aplikace kompletně splnila zadání a je připravena na použití. V závěru diplomové práce jsou navrženy možnosti dalšího rozvoje aplikace.

Klíčová slova: sběr obrazových dat, DJI, autopilot, Gumstix, Yocto Project

Překlad názvu: Pokročilé řízení užitečného zatížení palubním počítačem na platformách DJI

Contents

Figures

Tables

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), also called drones, are aircrafts, which are controlled by a computer instead of a human pilot aboard. This control is done either remotely by a human operator or autonomously by an onboard autopilot with a predefined trajectory.[6] There are several application fields, which imply the construction and the equipment of the aircrafts. The one, which was probably the main reason for the development of such aircrafts, is military. The others, which appeared heavily in recent years, are filmography, logistic, agriculture, forestry, civil engineering, scientific sensing, etc.

The variability of their movement results from the number of their rotors and propellers. There are categories named after that such as quadcopter, hexacopter, octocopter, etc. In our project, we are interested in multicopters in general. The components of these aircrafts are very similar to hobby aircraft models, which means that there is not any cockpit, environmental control system or life support system. These aircrafts are in general cheaper and easier to manufacture than usual manned aircrafts. There is an opportunity to use for example less robustly tested electronic control systems or lighter and less sturdy materials for the construction. Therefore, the research and usage of these technologies are open to smaller research organizations and universities.

The equipment of the aircrafts, also called payload, is mostly smaller and lighter than a human, so the aircrafts can be considerably smaller. To handle the stability of the aircraft in the air, it becomes very useful to use electric motors, because its speed can be controlled and changed much faster than if we used combustion engines. For the power supply, there are mostly used

some lithium-polymer batteries. The flight control units are, due to the recent development of computer technology, very powerful microcontrollers, which are often called autopilots. The position and attitude of the aircraft is obtained from a complex system of sensors. The most frequently used sensors are GPS and IMU (inertial measurement unit). These components are the core of the aircraft, which ensures the ability to navigate and stabilize it. Apart from that, the aircraft can carry any other useful device, which can weight several kilograms depending on the construction and power of the aircraft. All this results in relatively very light, fast and agile aircrafts, which offer many interesting and innovative usages in many various sectors.[13]

Most of the drones used for civil purposes are equipped with an aerial camera, which can provide wonderful landscape photographs. Compared to manned aircraft, it is definitely much cheaper to use a drone and take an elevated photo or video, than to rent a helicopter with a human pilot and a photographer. Some basic tasks, like taking a few photos or a video can be done directly by the autopilot. It has become very popular because drones can be used to take photos from hardly accessible places or even on the go. These properties can be very valuable in architecture, in sports or in professional photographing. Nowadays, the drones are capable of following a sportsman and capturing a high-resolution videotape of his performance. On the Winter Olympics 2018 in PyeongChang, we could see a very impressive light show, which was done by a synchronized flight of more than 1200 drones with LED lights. The development of the drones rushes forward and new applications rise every day.

However, there are still some challenging tasks, which can't be accomplished by the present autopilots. One of the recently most developed fields is so-called image data acquisition. It is a complex task, which provides a coherent set of photos of an area, which has to be taken in equidistant spots and saved along with its geodetic or telemetry data. Adding geodetic data to the images is called geotagging, and it forms the second important part of the dataset. The images with the added metadata could be used to produce any kind of a map or a landscape model. The process of reconstruction of the landscape is very sensitive to deviations in the metadata, which must be minimalized. The images have to be taken in several angles and directions, in order to simplify and support further processing. The control of the camera should be done at least with some basic feedback, to be sure, that every photo was captured in the determined moment. Standard present autopilots aren't capable of such a control and they are also missing some other useful features. When collecting such a dataset, it is crucial to capture all the images with no exceptions. When an image is not captured, the drone should return to that spot and try it again. This functionality demands much more sophisticated behavior than present autopilots can offer.

The last ideas, which may come to your mind, could be for example synchronization of more cameras or some basic communication with the operator, which may help to calibrate the cameras. Next level improvement could be a live stream preview of some taken photos directly to the operation computer. There will be always something to improve in this field. That is the main reason, why I started my development right here.

1.1 Goal description

The goal of this project was the development of a control unit, which would solve the problems briefly described in the introductory section. The purpose of this thesis was to describe properties of particular components, suggest the complex solution and document the progress and the final results.

To summarize the task, the control unit had to be capable of communicating with the camera and with the autopilot and this communication had to be bidirectional in both cases, because this unit was meant to be the main controller of the whole system. The input for the task was a file describing the target area and a file describing the requirements for the results. The output of the system had to be the set of image data along with the geodetic and telemetry metadata.

The resulting system is a composition of three devices: a control unit, a camera and an autopilot unit. There is obviously a fourth device, which is the drone carrying all the other devices, but the system controls the drone through the autopilot unit, so the system is not dependent on the drone itself. All these components will be described in detail in the next chapter. This section deals with the basics of connecting these components together, which represent the main tasks of this work.

The first great subtask was to establish the connection between the control unit and the camera. This connection has to be capable of transmitting two kinds of signals. At first, the control unit has to send the shutting signal. Next, there has to be secured fast feedback information saying, whether the photography has really been taken or not because the camera does not take the photos surely, whenever it gets an impulse. There may arise many accidental and undetermined problems, such as overflowed buffers, focus problems, memory problems, etc.

The second subtask was to establish the connection between the control

unit and the autopilot unit. The control unit has to be able to obtain the telemetry data provided by the autopilot. Apart from that, there should be an opportunity to affect the flight control. The system should be able to upload a flight plan and track its execution during the flight.

The third subtask was to implement an application, which would use the established connections to accomplish the image data acquisition within a given area. This program was divided into three parts. The first part is a flight planner, which is an algorithm for constructing a flight plan for a given area so that the camera could take photos of the area within the given demands for resolution and overlaps. The second part is the flight and shutter control application, which executes the flight plan and controls the shutter of the camera. The last part is the data analyzer, which produces the geotagging script, that contains the GPS coordinates of the gathered photos.

The fourth subtask was to implement a tool, which would visualize the outputs of the application. For this purpose, a few MATLAB scripts were created. Using these scripts, the user can visualize the target polygon, the planned trajectory of the flight, the actual trajectory of the flight and the image coverage of the target area.

The last subtask, which was partially implemented, was the communication between the control unit and the operator on the ground. The purpose of this communication is mainly to provide some preview data to allow the operator to check the functionality of the system before taking some more exhaustive measurements. Moreover, this communication could be used to control the system fully from the ground, which would allow the operator to modify the experiment without the need of landing the drone. However, this subtask was not a part of the thesis assignment so that it was partially left for further development.

1.2 Thesis overview

The following part of the thesis is divided into four chapters. The first chapter describes the hardware modules, which were used to design the solution. The next chapter is devoted to the core of the project, which is the software part of the solution. It contains a description of the build system used to provide the operating system, a description of the software development kit needed for communication with the autopilot and a description of the developed application.

The last but one chapter deals with the experimental evaluation. The simulation environment is presented, and the experiments are described and evaluated. The last chapter concludes the thesis. It summarizes the contribution of the project and suggests some possible improvements.

Chapter 2

HW solution design

This chapter describes the HW structure of the developed system, what components are used and how the components are physically connected. In the following sections, there are descriptions of all used components along with their attributes, characteristics and possibly used interfaces. The HW design of the whole solution along with the interfaces can be seen in Figure 2.1.

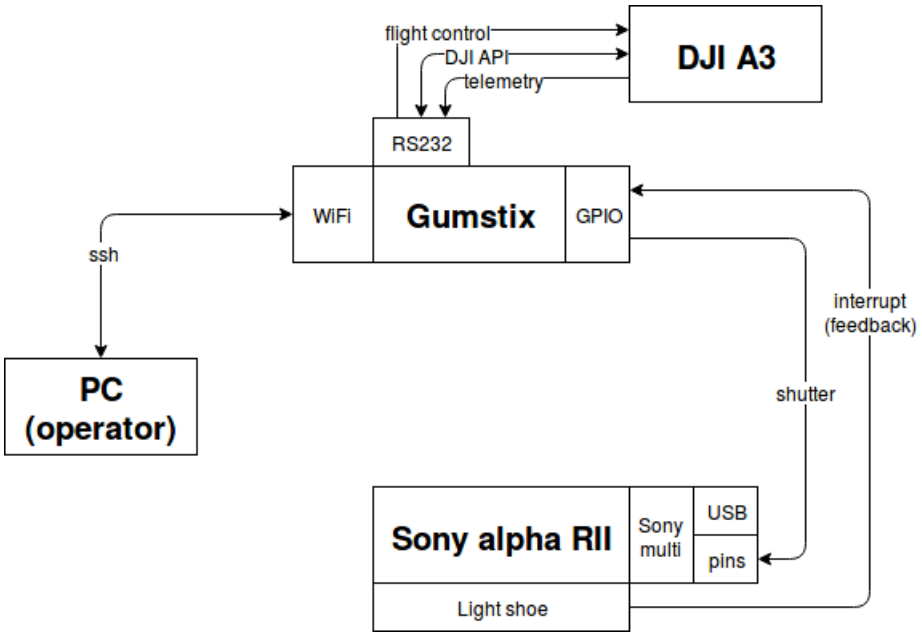


Figure 2.1: HW design

2.1 Control unit

As the main control unit, we chose Overo FireSTORM-P COM[7]. This computer-on-module has 1 GHz ARM Cortex-A8 processor, WiFi and Bluetooth module, microSD card slot and two 70-pin AVX connectors, which can be used to connect the computer to an expansion board. Such an expansion board can significantly extend connectivity of the computer. As the expansion board, we use Tobi[8], which provides various USB ports, ethernet port, DVI display, 40 GPIO pins and many other useful interfaces. This expansion board is one of the recommended for engineers to start their researches and when the solution is tested enough, a custom expansion board can be created to meet the specific requirements.

The main reason to use Gumstix was that it was already in our resources, but it was currently not in use, and this project seemed to be a good option to utilize this computer. The simple task containing only the triggering of the camera had been implemented before using an Arduino microcontroller, but when the task was extended by the mission planner and the analyzation subtasks, it was necessary to look for some more powerful platforms.

In contrary to the Raspberry microcontrollers, working with Gumstix is more complicated and exhaustive, because the documentation of the product is very poor and the community support is incomparable to the support of Raspberry developers. When searching the internet for some Raspberry sample codes, tips or troubleshooting hints, one can find dozens of articles, discussions and other helpful materials. When looking for the same in the Gumstix community, one has to be very lucky to find few pages dealing with the specific topic, while most of them contain only unanswered questions asked by some other struggling developers.

Development on the Gumstix platform is not easy, but when the developer is patient enough and doesn't lack time, the results always pay off. Development of a master thesis project on the Gumstix platform won't be really recommended by me, but as we can see, it is accomplishable.

2.2 Camera

To get high-quality photographs, we chose the SONY α7RII[15] camera with high resolution of 42.4 Mpx, high sensitivity and speedy response. A significant feature is 5-axis image stabilization, which successfully reduces blur. It is also capable of 4K movie recording. Connection with this camera is made through Sony-multi interface, which provides USB port and can be used to control the shutter.

The configuration connection can be made easily through the USB port. There can be used a specific library called Glib2, which implements all the basic methods needed for configuration and setup of the camera. On the other hand, the USB communication can also be used to provide the preview photos for the control unit, so that they can be sent to the operator. The other pins could be connected to the GPIO ports of the control unit. This connection could be used to control the focus and shutter of the camera. The last thing we mentioned is the feedback confirmation that the photo has been taken. This information could be obtained from the light shoe of the camera, which is directly connected to ground by the camera when the photo is being taken. Using a simple external pull-up, there is generated a falling edge, which can be delivered to the control unit as an interrupt, in order to be processed immediately.

The parameters of the camera and the lens determine the proportions of the flight trajectory. The parameters of the SONY α7RII have the following values (Table 2.1):

focal length	28mm
image height	5304px
image width	952px
sensor height	24.00mm
sensor width	35.90mm

Table 2.1: Camera parameters

For the trajectory planning, there are two important distances: *grid spacing* and *trigger distance*. The grid spacing is the distance between two successive line segments, which depends among others on the image width and sensor width. The trigger distance is the distance between two successive camera trigger spots and it depends on the image height and sensor height. There are some other determining inputs for the planning algorithm, which are

described in the section dealing with the SW components, specifically the section about the planner itself.

2.3 Autopilot

The autopilot unit, DJI A3[3], is a very powerful autopilot, which has many interesting features, such as intelligent flight modes, low voltage protection, motor overload detection, etc. Communication with this unit can be easily done through the standard TTL serial communication. There is prepared a very extensive API for this purpose, which can be used to manage the flight control and read the telemetry data provided by the autopilot unit.

2.4 Drone

The aircraft we use is called DJI Matrice 600, and it is a modern flying platform designed for professional aerial photography and industrial applications. It is equipped with powerful DJI technologies, including the A3 flight controller, Lightbridge 2 transmission system, Intelligent Batteries and Battery Management system, for maximum performance.[3]

The aircraft dimensions are 1668mm x 1518mm x 759mm when fully expanded and 640mm x 582mm x 623mm when frame arms and GPS mount folded. The weight of the aircraft along with batteries is about 9.5 kg, and maximal takeoff weight is 15.1 kg. Important performance parameters are maximal flight altitude above sea level, which is 2500m, and maximal speed which is 18 m/s when there it is windless. The hovering time is about 40 min while carrying no payload and about 18 min while carrying about 5.5 kg payload.

The Matrice 600 supports the DJI GO mobile application and the new brand DJI Assistant 2, which is a desktop application. The DJI GO application is designed to be used along with the remote controller, which allows the operator to communicate with the drone from a very long distance up to 5 km when used in an area, where no obstacles and no disturbances are present. The DJI Assistant 2 can be used to setup the configuration of the drone, and it allows the developer to use the built-in flight simulator.

Chapter 3

SW solution design

This chapter describes the SW design of the developed system, which is the operating system and the application running on the control unit. The first section describes the Yocto project, which is a tool for building custom Linux distributions. The second section describes the application, which runs on the control unit during the flight and controls the process of the image data acquisition. The last section describes the data analyzer, which synthesizes the data saved during the flight a produces the final output, which is the geo-tagging file for the captured images.

3.1 Yocto Project

When developing an application, which is supposed to be executed on an embedded computer, the developer has to give special focus to the operating system, which always has a crucial influence on the features and performance of the user program.

The Gumstix computer can run various operating systems. For our purpose, the best choice is the standard embedded Linux distribution with some minor modifications. Compilation of such a custom operating system can be done in the Yocto Project Build System[5].

This build system is an open-source collaboration project, which focuses on

the development of embedded Linux systems. It uses a build host based on OpenEmbedded project, which uses BitBake tool to construct complete Linux images. It provides a development environment for many different architectures like the ARM, MIPS, PowerPC, and x86 and it contains components and tools used to design, develop, build, debug and simulate any custom software application. While it has so many features, it is quite difficult to learn how to use it properly.

The first thing one has to deal with is setting up the host system. The Yocto Project is designed to be used on Linux distributions like Ubuntu, Fedora, Debian, etc. The easiest way of getting the Yocto Project working on a Windows machine is using some virtualization tool to run a Linux distribution virtually in the Windows environment. I can recommend Virtual Box for this purpose because this project was developed entirely in the Virtual Box.

The host system has to fulfill a few application version requirements which involve getting specific versions of Git, tar, and Python. Except for these, there are some system packages, which have to be downloaded and installed. The process of getting the host system into the competent state is quite a time demanding and it shouldn't be underestimated.

Learning the basics of how to use the Yocto Project can be comfortably done by following a Getting started manual, which presents how to build a custom Linux distribution in a few steps. It starts with downloading the Yocto Project from a Git repository, continues with the initialization of the build system and ends with building and simulating of a prepared image.

When it comes to the development of a custom operating system, one has to start using the Yocto Project Development Manual, which is comprehensive documentation containing almost everything one has to know to be able to achieve any goal. On the other hand, it is so extensive, that it is more time efficient to read only the segments, which are relevant for the particular project. The manual has quite a clear outline, so it is easy to find answers to specific questions.

The greatest advantage of the Yocto Project lies in the structuralism. All the properties of the operating systems are divided into logical groups called layers. The developer can include these layers to gain access to the features implemented in each of these layers. When a developer produces a new program or a function, he can create a new layer to keep his work consistent. Many layers containing many features are already prepared in the Yocto Project Git repository, and the development of a custom operating system

consists in the combination of these layers and features together.

The build of the custom image is coordinated by a few configuration files. There are two main configuration files, which contain the information about incorporated layers and modules. Then, there is a configuration file in every layer, which describes, how the specific layer should be compiled. Except that the developer produces some own application, his job also contains the maintenance of these configuration files. These files contain a huge amount of variables, which have to be set correctly to produce the desired result.

When all the modules are ready for compilation, and the configuration files are prepared, the developer can use the BitBake tool to construct the image of the operating system. Then, the image can be (using a Bmaptool, for example) copied to an SD card, which can serve as a bootable flash drive.

For this project, we chose a Gumstix console image, which can be obtained from the Gumstix Git repository dedicated to the Yocto Project Build system [2]. There is also a very detailed step-by-step manual, which helps to understand, how to setup and build the image.

3.1.1 Event driver

Even though the Git repository of the Yocto Project contains very many various modules and programs, which can be included in the custom operating system, the developer often comes across the situation, where he has to contribute with some own piece of code to achieve the desired behavior of the operating system. This includes especially the kernel modules, which can be introduced into the Linux kernel through the Yocto Project very comfortably.

The Gumtix console image contains almost everything we need to get our application working. The only thing, which was missing in the system, was GPIO interrupt handler. The GPIO pins can be accessed in user space quite comfortably. It is possible to export unexported pins, to set their direction to output or input, and to set and read the value of the pins. But if we wanted to read the feedback signal coming from the camera this way, we would have to check the value of the pin in a loop, and it would have a very negative influence on the performance of the program. This signal should be set up as an interrupt and should be handled by the operating system instead.

On the Wiki page of the Gumstix, there can be found a kernel module

called GPIO event driver[10], which was developed right for this purpose. The module can be used to set up an interrupt any desired GPIO pin along with the type of edge (rising or falling) and with debounce avoidance time. The information about the interrupt can be then obtained through a special system file in the /dev folder.

3.2 DJI software development kit

This section describes the procedure of setting up the connection between the Gumstix COM[7] and the DJI A3[3] flight controller and running the basic sample applications provided with the DJI OSDK[4].

3.2.1 DJI OSDK

The DJI SDK is a free software development kit provided for the DJI developers. There are three different kinds of SDKs, such as On-board SDK, Mobile SDK and Guidance SDK. All of them provide tools to control the aircraft along with its payload, which can be used to create custom applications that fulfil the potential of the aircraft. The On-board SDK and Mobile SDK are quite similar. They can be used to create a customized application, which can handle the mission setup, camera control and even the flight control itself. The difference between them is that the On-board SDK is designed for Linux computers and connects directly to the flight controller over a serial interface, while the Mobile SDK is designed for Android and iOS mobile devices that connect to the aircraft wirelessly through the remote controller. The Guidance SDK is a bit different. It comes along with a Guidance module, which is a revolutionary visual sensing system. It consists of a powerful processing core, visual cameras and ultrasonic sensors, which offers a new level of safety and confidence in flight. The Guidance SDK was developed to provide practical access to this module. For our purpose, we chose the On-board SDK (abbreviated as OSDK), because we develop an embedded application, which runs directly on the board when the vehicle is in the air.

The OSDK can be obtained in a few different versions to match requirements of the hosting operating system. Since we use a customized Linux distribution, we chose the open source C++ library version of the OSDK with support for Linux, ARM and STM32.

The features, which this SDK provides, are quite extensive. The SDK gives access to the aircraft telemetry, flight control, camera control, etc. This means that the developer can use the SDK to attach his own onboard computing device and use it to control the flight and the payload.

3.2.2 Connection

Contrary to the Mobile SDK, which is designed for mobile devices and supports wireless connection, the On-board SDK is designed for controllers, which have to be connected directly to the flight controller over a serial interface. The flight controller provides a UART serial interface with 3.3 volt TTL, but the Gumstix COM supports just 1.8 volt TTL. Therefore we had to introduce a level shifter into this connection. The power supply is secured by the Gumstix Tobi board since it provides both 1.8V and 3.3V pins. The level shifter may contribute to the communication by some extra delay, caused by opening time of the MOSFET transistor, which is about a few nanoseconds. However, the default baud rate of the communication is 230400, so the delay of a few nanoseconds is totally insignificant. When using the Tobi board, one of the UART interfaces can be found on pins 9 (RX) and 10 (TX). This interface is accessible from the operating system through /dev/ttyO0.

3.2.3 Cross-compilation

The OSDK for Linux operating system is distributed as an open source C++ library. Except for the core C libraries, it requires pthread library. Compilation of the library is not complicated when compiled on the device, where it is used lately. In our case, we don't want to compile any code directly on the Gumstix COM, so we need to set up a cross-compilation environment, where the library could be compiled for a specific processor. Such an environment for the Gumstix COM can be obtained using the Yocto Project utilities. The whole process is described in details in the "Cross-compilation" chapter.

3.2.4 Telemetry sample

The DJI OSDK comes with a few sample applications, which may be instrumental in the development of a custom application. There is a simple

application, which presents the accessibility of the telemetry data of the flight controller.

The first step of the procedure is a configuration of a so-called subscription. The application subscribes to some specific topics, which the flight controller then produces in specified frequencies. The most important topics are the GPS coordinates, the aircraft attitude, the flight status, etc. Most of the topics, including the GPS, have the highest possible frequency set to 50Hz.

3.2.5 Mission sample

In contrary to the previous example, which is only reading some information from the flight controller, the mission sample application presents, how a sequence of GPS locations, so-called waypoints, can be transformed to a mission plan, which can be then executed by the aircraft.

When a developer creates such an application, which deals with the control of the aircraft, he has to have a developer account on *developer.dji.com*, where he can obtain unique API key for his application, which he needs to get the application working. The key is required by the DJI SDK and it serves as a safety enhancement for the drones, which are used in autonomous mode. Whenever a drone is launched, it can be uniquely determined, whose application is responsible for its movement.

The mission plan has some extending settings. There can be set for example a bend length, which stands for a turn radius and offers an interesting option for the aircraft to fly the trajectory smoothly, without stopping at each waypoint. There can be also set a command to be executed in a waypoint, but this requires that the aircraft stops and hovers in that waypoint. This utility could be used to take a photo on specified GPS coordinates. If we wanted to use it to gather image data from a great amount of GPS coordinates, like the image data acquisition task, the aircraft would spend too much power to accelerating and decelerating over and over again, so the scannable area would be significantly smaller. This problem is the main handicap of the flight controller and the main reason to introduce a shutter unit.

3.3 Shutter application

In the following paragraphs, there is the description of the program, which controls the whole process of the image data acquisition. The input for the program is the target polygon, parameters of the camera and mapping options like overlaps and ground resolution. The output of the process is the set of images along with the GPS coordinates, where they were taken.

3.3.1 Mission planner

Introduction

The first part of the application is the mission planner, which computes the trajectory composed of so-called waypoints. The trajectory has to be computed in the way, that the whole input polygon is covered on the gathered images with specific overlaps.

The general task of planning paths is always a very complex and complicated challenge demanding a very sophisticated solution. Fortunately, we are not the first developers to deal with this topic. Our implementation of the planner builds on two open source planners, which are quite popular among drone users.

One of them is called MissionPlanner[16] and it is written in C#, which means we couldn't use any piece of code directly. The second one is called QGroundControl[1], and although it is written in C++, it is implemented using the QT library, which is again impracticable in our application. But since they are both open source, we could use the code as an inspiration and implement our own planner.

Inputs

This section explains the inputs for the planner, which are two files describing the polygon and the parameters of the mission.

The file describing the polygon has to be named *polygon.kml*, and it can be generated using Google Earth. Google Earth is quite a popular software, which is free to download, and it offers very many practical tools to work with maps offered by Google. Beside the other common utilities, it allows defining a polygon using a few map points. Such a polygon can be then exported and saved as *.kml file, which is directly supported by our application. There is one requirement, which the developer has to comply when defining the polygon, namely: the polygon has to be convex.

The second file, describing the parameters of the mission, has to be named *parameters.xml* and it is an XML file containing certain values defining the process of the planning. The file is divided into two parts. The first part contains parameters of the camera: name, focal length, image height, image width, sensor height and sensor width. The second part contains the parameters (the assignment) of the mission: ground resolution, side overlap, and frontal overlap.

The name of the camera is not required by the algorithm, but it may help to keep things organized when someone would use more cameras. The focal length along with the ground resolution determines the altitude, in which the photos have to be taken. The ground resolution is defined as the number of square centimeters of the ground projected to one pixel of the photography. It defines the preciseness of the result. The image height along with the sensor height and the frontal overlap determines the trigger distance, which is the distance between two successive photo captures. Vice versa, the image width along with the sensor width and the side overlap determines the grid spacing, which is the distance between two successive line segments of the trajectory.

Algorithm initialization

This subsection explains how the developed trajectory planner works. As we described in the previous subsection, the inputs are: set of GPS coordinates defining the polygon, parameters of the camera, and requirements for the resulting photos.

First of all, we need to put the parameters of the camera and the requirements for the result together and compute the three determining measures: altitude, grid spacing, and trigger distance.

The altitude depends on the focal length of the lens, on the ratio between the image size and sensor size, and on the required ground resolution (measured

in cm^2/px)[1]. The formula for the altitude is the following:

$$altitude = \frac{img\_w \times ground\_res \times foc\_l}{sen\_w \times 100},$$

where *img_w* is the image width (measured in px), *sen_w* is the sensor width (measured in mm), *ground_res* is the ground resolution (measured in cm^2/px), and *foc_l* is the focal length (measured in mm).
When the aircraft keeps the computed altitude, the grid spacing and trigger distance can be both computed directly from the image size and ground resolution:

$$grid_spacing = \frac{image\_width \times ground\_resolution}{100},$$

$$trigger\_distance = \frac{image\_height \times ground\_resolution}{100},$$

These values have to be then corrected using the overlaps defined by the user. The formula for this correction is straightforward:

$$corrected\_value = \frac{100 - overlap}{100} \times value,$$

where *overlap* is the percentage overlap value.

When the parameters are computed, the process can proceed to the next step, which is an examination of the polygon. The polygon consists of points, which are defined by their GPS coordinates. Planning the path using ordinary GPS coordinates would be quite a complicated task. It can be simplified by introducing NED coordinate system, which is planar, unlike the GPS coordinate system, which is spherical.

The NED coordinate system (north-east-down) is a coordinate system, which is also known as the local tangent plane, and its principle lies in mapping any points from the sphere to a tangent plane. It consists of three values: one for the position along the northern axis, one for the eastern axis and one for the vertical position. The vertical position is not used in our case, because we know the altitude from the previous computations and we are looking only for the two-dimensional path. Obviously, this coordinate system needs on origin, whose GPS coordinates are mapped to zero in the NED coordinate system.

Therefore, before the algorithm starts, all the points defining the polygon are converted to the NED coordinate system with the origin in the first point of the polygon.

Algorithm process

The resulting path is composed of parallel lines with the distance of the grid spacing. In the first step, we have to determine the direction of these lines. To introduce some optimality in the algorithm, the direction of these lines is taken from the direction of the longest border line of the polygon. This ensures that the trajectory will contain the smallest possible amount of turnings.

When this direction is computed, the algorithm can start. The algorithm takes the two points of the longest border line of the polygon as the first line segment of the path. It chooses the direction along this line randomly. Then, a new line is constructed. The new line is shifted by the distance of the grid spacing in the direction perpendicular to the direction of the lines. Then, the algorithm computes the intersection of this new line with the border lines of the polygon. The intersection computation produces two points, which are then added to the resulting path. The order of this points is determined according to their distance from the last point of the path. This procedure is repeated until the newly constructed line has no intersection with the polygon. That means, that the path is done. To be sure, that there is no place left, the last newly constructed line is added to the path too, while the position of its boundary points is assumed to be the same as the previous.

The boundary points of these lines then express the resulting path. They can be turned into so-called waypoints and uploaded to the drone, which can then follow these waypoints one by one.

Smoothing

We could take the path generated by the explained algorithm, upload it to the drone, and it would work. However, the flight would not be smooth. The drone would fly from waypoint to waypoint, and it would have to stop in each of them. This behavior is undesirable because it would not be very power efficient as the aircraft would have to accelerate and decelerate in every waypoint.

Solution to this problem can be found in the settings of the waypoint mission. The mission can be set in the way, that the aircraft does not have to stop in the waypoints, it even does not have to fly precisely through the waypoint, but it can fly only sufficiently near the waypoint. The sufficient

distance from the waypoint can be defined for each waypoint differently, and it is set through so-called bend length, which apparently corresponds to the turn radius.

With this feature, we can make the trajectory smooth. We set the bend length for all the waypoints of the elementary path to zero, and then we add two extra waypoints next to every two points, which form a turning. These extra points can have the bend length set to the half of the distance between the two parallel lines to make the aircraft fly along a half circle. The distance of the two extra points from the polygon was set to three-quarters of the distance between the parallel lines, in order to let the aircraft stabilize after the turning maneuver.

After this small modification, the path is completely done, ready to be uploaded to the drone and executed.

The functionality of the algorithm is shown on the Figure 3.1. There is a simple four-cornered polygon. The longest side is the upper one, so the direction of the lines is horizontal, and the path starts in the top right corner.

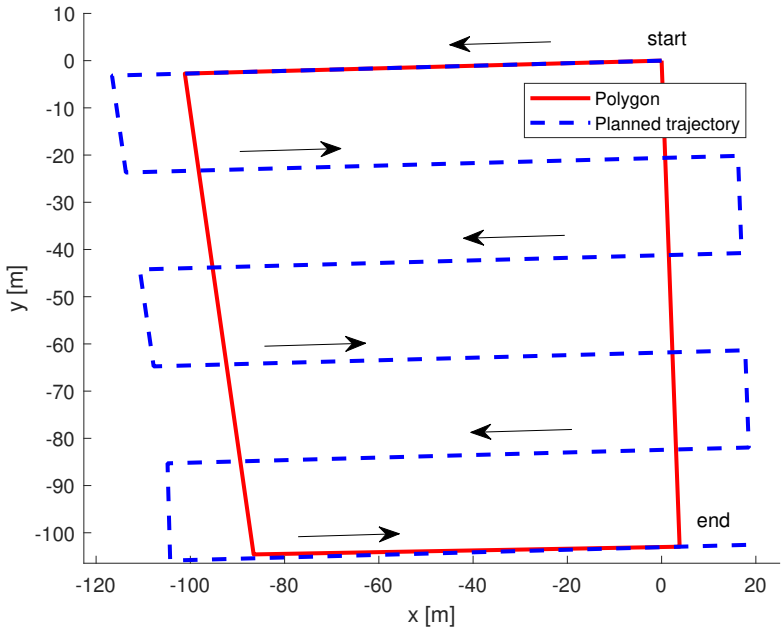


Figure 3.1: The output of the planning algorithm

3.3.2 Shutter handler

The main part of this project is the application, which controls the process of the image data acquisition. The task can be divided into three subtasks, which can run concurrently. To accomplish that, we used basic *fork()*[11] system call, which is the simplest way of creating multi-process programs in C.

Calling the fork function, the parent creates a copy of itself. The code of the program is the same for both processes, and the diversity of their behavior is done using if clause. The fork function returns the PID of the child process, so it can be used in the parent process to identify its child process and terminate it, for example. In the child process, the fork function returns zero. Using this returned value, the if clause can be constructed, so the programs can do different tasks.[12]

The central responsibility and the control of the whole operation are held by the parent process, which was named "Telemetry thread". It is quite an incorrect notation because the fork function creates a process instead of a thread, but nowadays multi-threading is used very much, and multi-processing might be a bit confusing for some developers, so we chose to use the term *thread*.

The parent thread gets the telemetry data, controls the execution of the flight path and instructs the other threads. The other threads have some minor tasks, which cannot be effectively done by the parent thread. They are mainly operating the camera. One of them triggers the shutter and the second one records the feedback signals.

Telemetry thread

The parent thread does the main part of the task. It communicates with the autopilot and decides when to take the photos. Apart from that, it creates and controls the other threads.

First of all, the communication between the threads has to be established. For this purpose, there is a tool called *pipeline*. A pipeline is created using the system call *pipe*. Then, each of the processes gets its end of the pipeline, which can be used to communicate with the other process.

At the beginning of the program, there is the initialization of one pipeline, which is then used for communication between the parent thread and the child thread which is controlling the trigger. When the pipe is successfully initialized, the first fork is called and the "Trigger thread" is created. Then, there is created and opened a log file for the telemetry data, which is called "log_telemetry.csv." This file is used to save the telemetry data for further usage.

After the preparation step is done, the DJI API is initialized. The communication with the aircraft is established, and the subscription for the telemetry data is registered. The program subscribes to the GPS data and the attitude data. The frequency of the data update is set to 50 Hz, which is the maximal possible for the GPS.

When the subscription is successfully set, the mission planner is started. As described in the previous section, the input files are loaded, computations are executed, and the path is defined. Before the path is uploaded to the autopilot, there are some necessary steps, which has to be done. The maximal allowed velocity of the flight is computed, to keep the camera trigger frequency lower than 0.75 Hz. The path is extended by adding the first and the last waypoint on the coordinates of the start-up of the application in the altitude of 50m, to avoid any collisions of the aircraft with the surrounding environment. After that, the waypoints list is uploaded to the autopilot unit and the mission is started.

During the mission, this thread obtains the data from the autopilot and saves the data to the log file. However, the main duty of this thread is tracking the aircraft's movement within the waypoint mission. The program knows the coordinates of all the waypoints and also the order in which they are flown through. Assuming, that after the start, the aircraft is heading to the first waypoint, the program repeatedly computes the distance to this waypoint, and when the distance starts to grow, the waypoint has been reached and the algorithm can proceed to the next waypoint. This way, the program knows in every moment, what part of the mission is being executed.

The last responsibility of this thread is deciding, when the photos have to be taken. The photo should be taken on the border of the polygon, which is right in the waypoint of the mission, and then every time, the aircraft moves the desired trigger distance. The maximal possible trigger distance is computed during the planning phase, but it is corrected (lowered) using the specific length of each line segment along which the photos are being gathered. The length of the line segment is divided by the maximal trigger distance, and the result is then rounded up, so we get the number of photos planned to take along this line. Then the optimal trigger distance is computed by

dividing the length of the line by the number of photos. The new trigger distance is never greater than the basic due to the rounding up. When the aircraft moves this trigger distance, this thread sends a message to the thread, which controls the trigger, and a photo is captured. This behavior is active only during the flights along the line segments, where the photos have to be taken. During the turnings and during the flights from the start-up point to the polygon and back, the photos are not taken to spare the memory and the shutter of the camera.

When the mission is over, this thread recognizes that by reaching the last waypoint. After that, the autopilot executes the command assigned to the end of the waypoint mission, which is in this case "landing." The aircraft automatically lands, and the rotors are turned off. The program then terminates the other threads and proceeds to the last phase, which is called "Data analysis."

Camera trigger thread

This thread is responsible for controlling the trigger of the camera and writing a log file containing the timestamps of the triggers.

On the beginning, the program sets the configuration of the GPIO port 144, which is connected to the trigger pin of the camera. The GPIO pin is exported at first, then the direction is set to output, and its value is initialized to zero. Then, the control loop starts. The program gets inputs from the "Telemetry thread" through the pipeline, and when the signal to capture a photo appears, the GPIO port is set to one for 500ms, and the timestamp is written to the log file. Then the GPIO port is set back to zero, and the thread is waiting for another signal.

Camera feedback thread

The last thread is called "Feedback thread" because it deals with the feedback signal coming from the camera. The signal is registered by the event driver kernel module[10], and the information about its occurrence along with the timestamp appears in the /dev/gpio-event file. Since this file is not a regular file, the information needs to be read, or it is lost. This thread ensures the blocking read on this file, which means, that the thread reads the file until all the data is read and then it waits for another data to appear.[14] Whenever

a new interrupt record appears, it writes its timestamp to a specific log file for the feedback information.

3.4 Data analyzer

The section describes the last part of the program, which is the analysis of the log files and producing the geo-tagging outputs.

The algorithm for the analysis is divided into three parts. In the first part, the log file containing the camera trigger timestamps is used. The timestamps written in this file represent the images, which were attempted to be taken. This information is used to initialize an array containing structures representing all these images and the time of the trigger is written to each one of them.

The second step is deciding, whether and eventually when was each of the photos taken. This decision is made using the camera feedback log file. However, not all of the timestamps present in the file are valid. Some cameras may have a special feature called pre-flash. They use a brief flash before the main flash. As a consequence of that, we may get each of the interrupts twice. It is necessary to exclude these duplicities, which is done according to the difference between the timestamp and the one before. If the difference is lower than one second, the timestamp is ignored. The camera is not capable of taking two photos in such a short time interval.
The algorithm of matching the photos with the feedback timestamps is simple. Every feedback timestamp is assigned to the one image, which has its trigger timestamp right before this feedback. Practically, the algorithm goes through the images and checks the timestamps. When the trigger timestamp is greater than the feedback, the previous picture is the right one.

The last step is dealing with the GPS coordinates of the images. Since the algorithm knows, when the images were taken, it is very easy to choose the right record from the telemetry log file and add the right coordinates to the image. Since the timestamps are never equal, the algorithm takes the two records, which surround the searched timestamps and linear interpolation of the coordinates is performed.

When the analysis is done, the geo-tagging script has to be generated. The process of geo-tagging requires a tool, which is capable of editing the

metadata of the photos. We chose to use the *exiftool*[9], which supports very many file formats and it allows to edit the GPS metadata comfortably.

Since the program does not know the names of the photos, which are obviously needed by the exiftool, it has to be done without the names, just following the order of the images. Assuming, that the names of the photos are generated as a combination of a repeated string and an increasing number of the photo, it is possible to use this pattern in the script. The script can be constructed, so it requires only the name of the first photo, and then the names of the other photos are predicted by increasing the number contained in the name.

Chapter 4

Experimental evaluation

This chapter contains the description of the tests, which were made during the development. Some of them were only simulated, some of them were done by in-field experiments.

4.1 Simulations

Simulation is a safe way to try out an application. Every developer makes mistakes and executing any applications on real hardware in a real environment is always risky. When developing on the DJI platform, developers can use a very practical built-in simulator.

4.1.1 DJI Assistant 2

When the developer wants to use the simulator, he needs to download a desktop application called DJI Assistant 2, which provides among others the simulation environment. The connection between the aircraft and the computer is made through the aircraft's micro USB port. The application offers much more than just the simulation environment. It is capable of configuring the aircraft, and it provides a summary of the aircraft's condition.

To start the simulation, we have to insert the initial coordinates, which are passed to the autopilot. The aircraft behaves the same way like it would behave if it were in the air on the given coordinates. Even various wind conditions can be simulated. The DJI GO application signals the GPS coordinates, which we have inserted, and the aircraft can be controlled using the remote control. On the desktop application, we can start an environment simulation, which shows the aircraft and its movement and the view is the same as in reality.

4.1.2 Simulation

The definition of the polygon can be seen in the Figure 4.1. For this simulation, the factual GPS coordinates are irrelevant. The definition of the polygon depends only on their relative position. However, the simulation environment has to be initialized with the start position near to the polygon due to the geo-fencing mechanism.
The parameters of the mission were set according to the expected application of the gathered images, which was producing a 3D model using photogrammetry: overlaps 70% and the ground resolution $1px/cm^2$.

Figure 4.1: Polygon: Charles Square

Flight planner and execution test

The first tested functionality is the planner, which should be able to create a flight plan according to the input parameters. Then, the plan should be uploaded to the autopilot and executed. In addition, the telemetry data should be gathered during the flight.

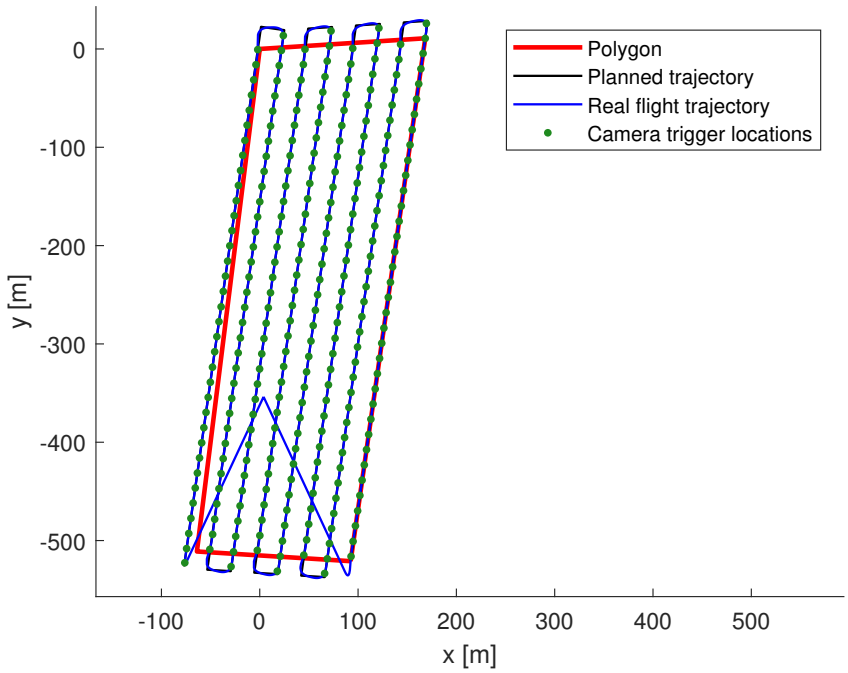


Figure 4.2: Trajectories

The process was simulated successfully. The trajectories are depicted on the Figure 4.2. The aircraft took off, flew through all the waypoints and landed as desired.

The functionality of the planner, along with the communication between the application and the autopilot unit, was successfully verified. The logging of the telemetry data was accomplished too.

Trajectory smoothness test

Another feature of the planner, which has to be verified, is the setting of the damping parameter, which should allow the autopilot to take smoother

trajectory and keep the velocity during the turning maneuver higher than half at least.

The detail of the trajectories can be seen in the Figure 4.3. The evolution of the velocity through the time can be seen in the Figure 4.4.

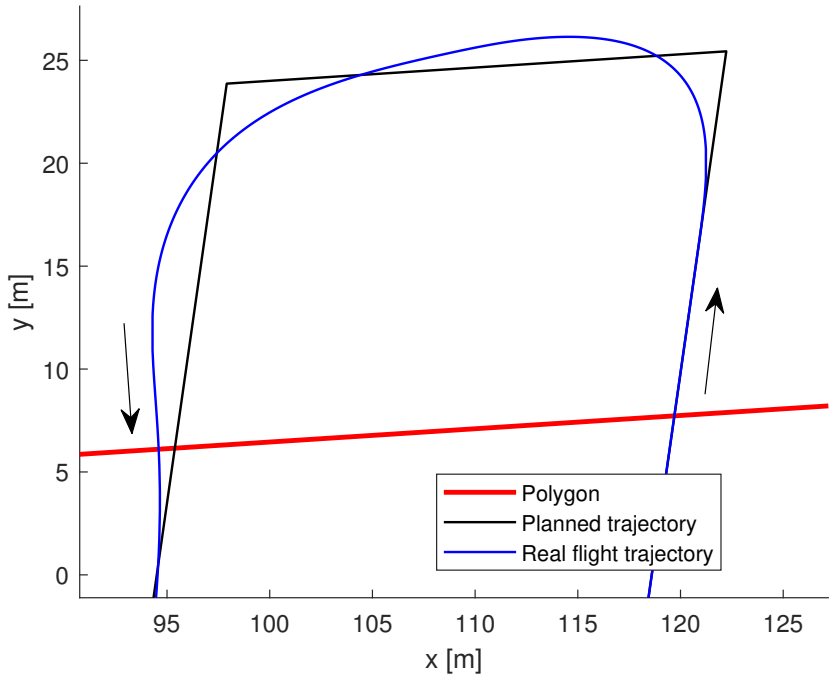


Figure 4.3: Detail of the trajectories

The Figure 4.3 shows, how the damping parameter of the boundary waypoints affects the resulting trajectory of the flight. The flight direction changes smoothly. As a consequence of that, the aircraft does not approach the polygon as planned. This is caused by the distance of the extra added waypoints from the polygon boundary. To decrease the deviation, the distance of the added points has to be increased.

The Figure 4.4 verifies, that the velocity is during the turning maneuver decreased from 7 m/s to 4.3 m/s, which fulfills our requirement.

Polygon coverage test

The main functionality, which has to be tested, is the resulting coverage of the target polygon, which should result from the demanded overlaps.

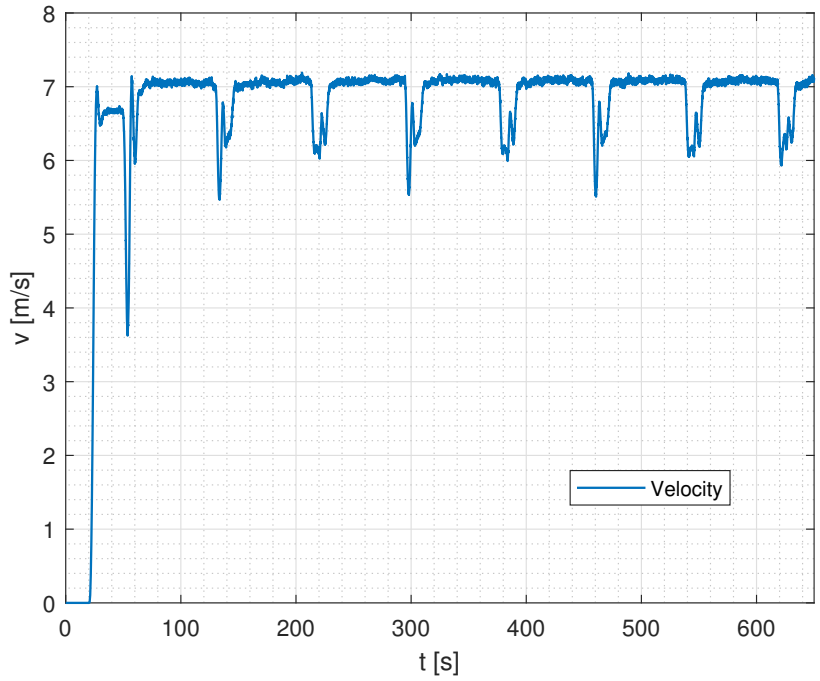


Figure 4.4: Development of the velocity

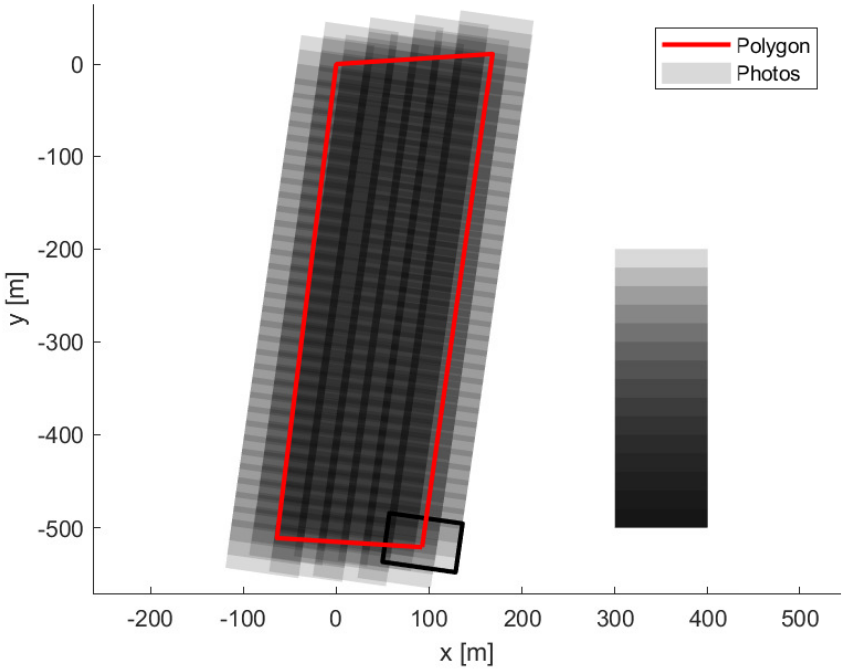


Figure 4.5: Polygon coverage

In the Figure 4.5, the coverage of the polygon is drawn. The shades of grey represent how many photos cover the specific area. The darker the area is, the more photos contain it, whereas each level of the shade adds one photo. The area covered by a single photo is shown in the bottom right corner.

It is evident, that the overlaps were accomplished. Moreover, due to the high overlaps of the photos, each point of the polygon is captured on at least six images. Most of the points are captured on even more images. This result is significant for the further processing, which is the 3D photogrammetry modeling, whereas the coverage is considered sufficient when each point is present at least on five images.

4.2 In-field experiment

When the simulations are successful, we can not proceed to the in-field experiments. For this purpose, we can use a private area near Bratronice, which is about 40 km from Prague. The area is a hexagon with the diameter about 450 meters.

The purpose of the experiment is to verify the functionality and practical usability of the application. In addition to the tests introduced in the simulation section, this section also contains some statistics concerning the camera functionality and response.

4.2.1 Safety

During the experiment, the control over the drone is handled by the control unit, but it can be anytime retrieved by the operator. Another fail-safe mechanism concerns the batteries. It is hazardous to fly when the charge level of the batteries gets under twenty percent. That is one of the most important limitations of the in-field experiments with drones. The DJI GO application has a geo-fencing mechanism, which contains, among others, settings for the highest allowed altitude and highest allowed distance from the operator. The program has to operate within these limits. Otherwise, its commands are not executed.

4.2.2 Experiment realization

The realization demanded some practical modifications. The power supply for the Gumstix COM has to be ensured. It was done using an extra battery added on board. The communication between the Gumstix and the operator was made through a portable WiFi access point using ssh. The application had to be executed in a detached screen, which allows the program to continue running even when the ssh communication is lost due to the great distance.

4.2.3 Execution

The input polygon can be seen in the Figure 4.6. The parameters of the mission were set the same as in the simulation: overlaps 70% and $1px/cm^2$ ground resolution.

Figure 4.6: Polygon: Bratronice

The experiment was executed almost successfully. However, the capacity of the batteries was insufficient, and the experiment had to be ended earlier. Fortunately, the omitted part contained only the last line segment, so the gathered dataset was sufficient.

4.3 Results and evaluation

The resulting trajectories are depicted in the Figure 4.7. The resulting coverage of the polygon is shown in the Figure 4.8.

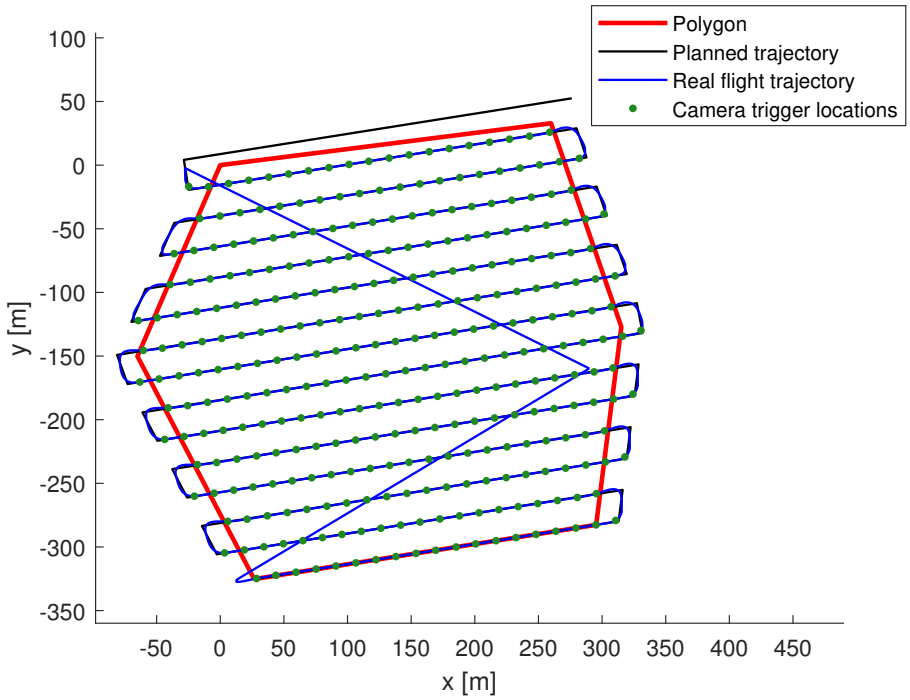


Figure 4.7: Trajectories

The following figures describe some obtained statistics of the camera. The Figure 4.9 contains a histogram of the time intervals between two consecutive triggers of the camera. A histogram of the response delay of the shutter can be seen in the Figure 4.10.

The flight trajectory was designed as expected and the resulting photo coverage of the polygon was more than satisfactory. The camera was triggered in a frequency lower than 0.75 Hz, which was demanded. The delay of the shutter was in average about 176 ms. Since the average velocity of the aircraft was 7 m/s, the shift of the photos was about 1 meter, which is acceptable.

The generated 3D model of the polygon can be seen in the Figure 4.11.

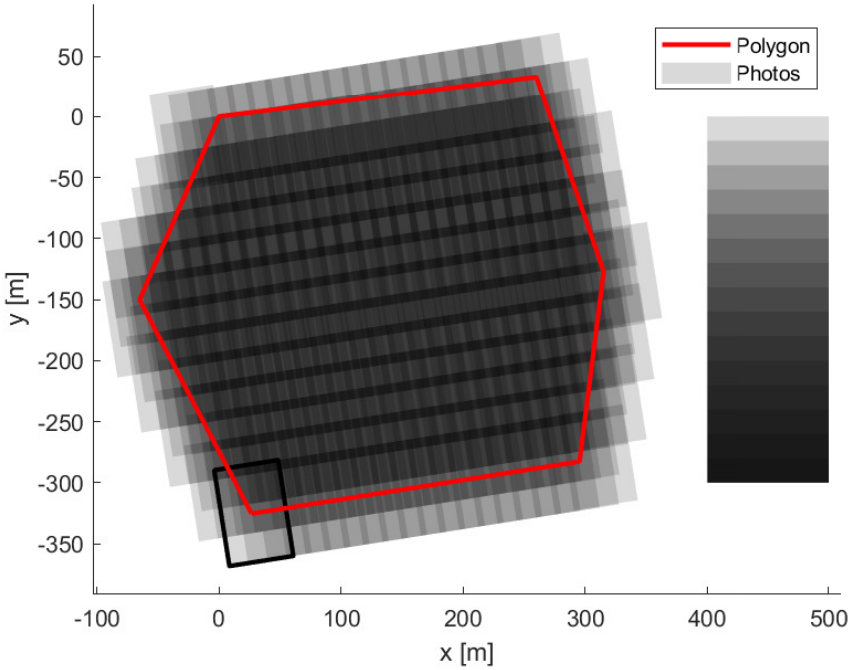


Figure 4.8: Polygon coverage

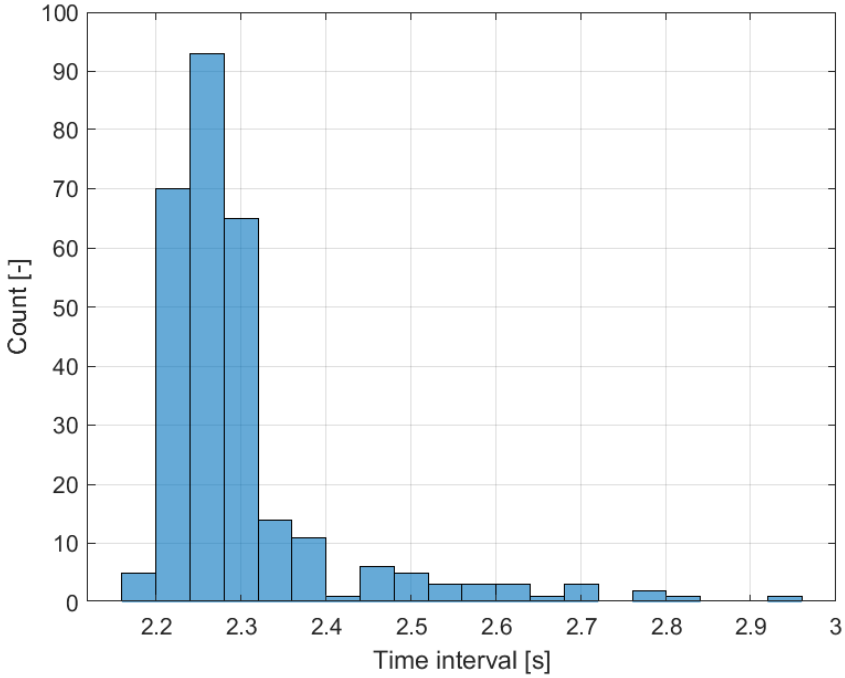


Figure 4.9: Trigger time interval

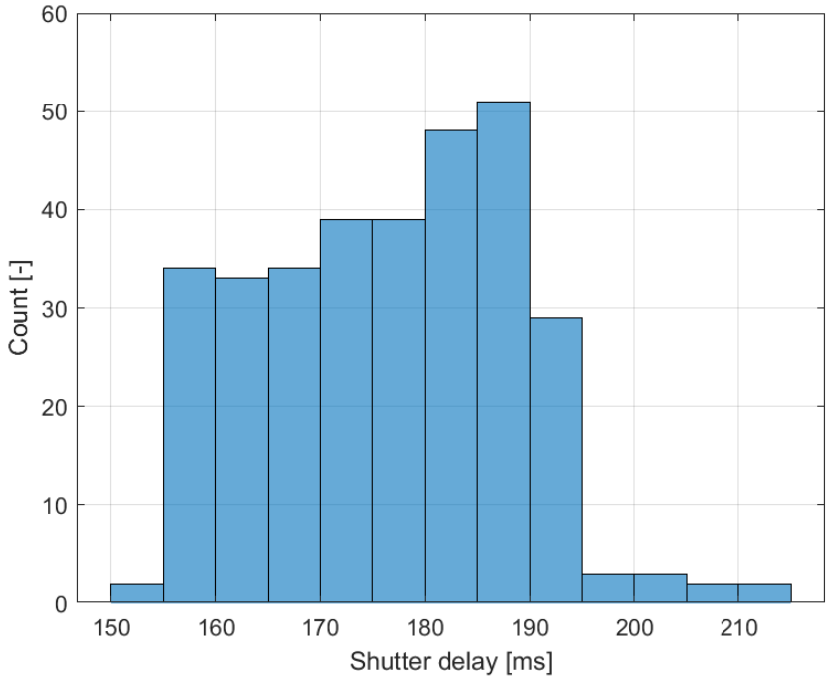


Figure 4.10: Shutter response delay

Figure 4.11: Resulting 3D model

To summarize the results, the application succeeded in the simulation and also in the in-field experiment so that it can be considered operational.

Chapter 5

Conclusion

This chapter contains the evaluation of the whole project, review of objectives with respect to the final solution and some possibilities for the future work.

5.1 Review of the project

As shown in the evaluation chapter, the results of this project are very good. The developed application corresponds to the assignment, and the performance is more than satisfactory. Moreover, the thesis contains detailed descriptions of all used technologies so that it can serve as a guide for other developers.

Even though some problems were encountered during the development, the solutions were always found. For example, when a software feature failed, then a hardware solution was introduced. Although a lot of work was done and a lot of documentation was written, there are still some possibilities for improvement, which are described in the last section of the thesis.

5.1.1 Review of the objectives

The supreme goal of this project was to develop a system for image data acquisition. Such a task could be divided into several subtasks, while all of the subtasks had to be accomplished to obtain a reliable result.

First of all, the documentation of the dedicated devices had to be studied. The first development phase concerned the build of the specific OS with the desired configuration of all necessary communication interfaces. The second development phase concerned the application itself. The application was composed of several modular parts, such as the mission planner, the flight and shutter control application, and the data analyzer. When the development was done, the project continued with the experiments. Most of them were simulated, but a few tests had to be done in the real environment. Using the data from the experiments the evaluation was done, and the project was finalized.

5.1.2 Review of the solution

The solution is a robust and user-friendly tool for image data acquisition, exactly how it was assigned. Although the process of image data acquisition was feasible without this tool, it was very complicated and unreliable. The operator had to use several different tools to prepare and upload the flight trajectory, the available shutter applications were inaccurate, and the further data analysis was inefficient.

Using the developed system is very simple. The user can generate the description of the polygon through the popular GoogleEarth software. Then he only creates a configuration file containing the parameters of the camera and parameters of the desired results, and the rest is done by the developed application. The control unit computes the flight trajectory and executes it, including the takeoff and the landing. During the flight, it controls the data gathering, and after the mission is over, it produces the geo-tagging script, which contains the GPS data to the photos. The user then downloads the images from the camera and executes the script. Processed images contain the information about GPS coordinates, where they were taken.

5.2 Future work

Even though the primary goal has been accomplished, there are always possibilities for improvement.

First of all, the presented tool is capable of using only one camera. There is a possibility to add two extra cameras to obtain side views, which would extend the coverage to the sides. This extension would lead to shortening of the flight distance needed, which would offer to cover a greater area.

Another improvement could be made in the handling of failed photos during the flight. The presented tool analyzes the photos in an offline mode, which means after the mission. It is possible to evaluate the information during the flight, to offer a possibility to deal with the photos, which were not taken as supposed. The control unit could, for example, change the flight plan and return to capture the missing photo.

Finally, the tool could also provide some preview photography during the flight so that the operator could modify the configuration of the camera or change some other parameters.

The design of the extended solution is presented in the Figure 5.1.

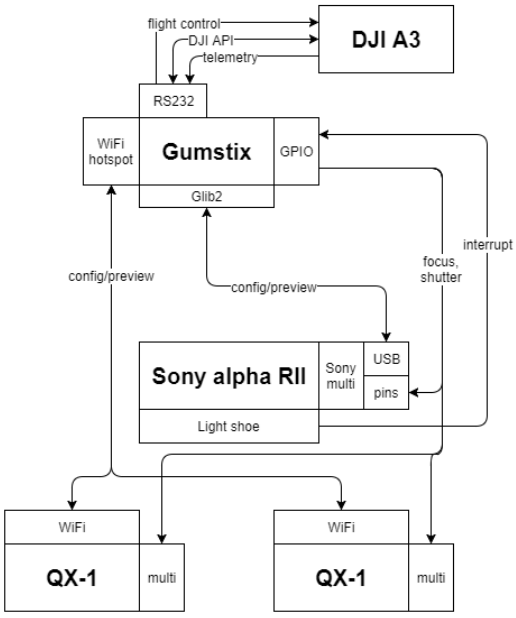


Figure 5.1: Design with the future extensions

However, these improvements demand extra potion of time and effort, which is not within the scope of this thesis.

Even so, the contribution if this thesis is not negligible and the presented tool is ready to be used practically.

Bibliography

[1] QGroundControl. http://qgroundcontrol.com/, 2016. Accessed on 2018-04-02.

[2] Ash Charles, Adam Lee, et al. Gumstix Repo Manifests for the Yocto Project Build System. https://github.com/gumstix/yocto-manifest, 2017. Accessed on 2018-01-31.

[3] DJI. DJI A3. https://www.dji.com/a3/info#specs. Accessed on 2018-02-11.

[4] DJI. DJI-OSDK. https://developer.dji.com/onboard-sdk/. Accessed on 2018-05-24.

[5] Linux Foundation. Yocto Project. https://www.yoctoproject.org/. Accessed on 2018-02-11.

[6] Jeremiah Gertler. U.S. Unmanned Aerial Systems. Congressional Research Service, 2012.

[7] Gumstix. Overo FireSTORM-P COM. https://store.gumstix.com/coms/overo-coms/overo-firestorm-p-com.html. Accessed on 2018-02-11.

[8] Gumstix. Tobi. https://store.gumstix.com/tobi.html. Accessed on 2018-02-11.

[9] Phil Harvey. ExifTool. https://www.sno.phy.queensu.ca/~phil/exiftool/, 2018. Accessed on 2018-05-02.

[10] Dave Hylands. GPIO Event Driver. https://wiki.gumstix.com/index.php/GPIO_Event_Driver, 2010. Accessed on 2018-03-31.

[11] Michael Kerrisk. Linux Programmer's Manual. `http://man7.org/linux/man-pages/man2/fork.2.html`, 2016. Accessed on 2018-04-15.

[12] Edward Ashford Lee and Sanjit A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.

[13] Xin Li and Lian Yang. Design and Implementation of UAV Intelligent Aerial Photography System. 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, 2012.

[14] Peter Marwedel. *Embedded system design. Vol. 1*. New York: Springer, 2006.

[15] SONY. Sony α7RII. `https://www.sony.co.uk/electronics/interchangeable-lens-cameras/ilce-7rm2`. Accessed on 2018-02-11.

[16] ArduPilot Dev Team. Mission Planner. `http://ardupilot.org/planner/`, 2016. Accessed on 2018-04-02.