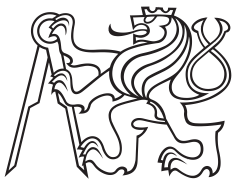
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

LiDAR based obstacle detection and collision avoidance in an outdoor environment

Jan Předota
Cybernetics and robotics

May 2016
Supervisor: Ing. Milan Rollo, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Jan Předota**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **LiDAR based obstacle detection and collision avoidance in outdoor environment**

Guidelines:

1. Study the problematics of navigation based on laser rangefinder in unknown outdoor environment
2. Integrate essential sensors onto an autonomous unmanned ground vehicle (UGV)
3. Implement methods for sensory data processing and representation and generate obstacles for autonomous mobile vehicles
4. Adjust the planning algorithms of existing system for command&control of autonomous vehicles so that the vehicles can react to new obstacles and change their trajectories in real time
5. Verify the functionality of the resulting system in real environment

Bibliography/Sources:

[1] Weitkamp, C. ed. Lidar: range-resolved optical remote sensing of the atmosphere (Vol. 102). Springer Science & Business, 2006, (online: http://home.ustc.edu.cn/~522hyl/)
[2] Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C. and Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots, 34(3), pp.189-206. 2013
[3] Vanneste, S., Bellekens, B. and Weyn, M., 3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap.

Bachelor Project Supervisor: Ing. Milan Rollo, Ph.D.

Valid until the summer semester 2016/17

L.S.

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

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 11, 2016

Acknowledgement / Declaration

I would like to thank to my supervisor, Ing. Milan Rollo, Ph.D., for his guidance, professional attitude and valuable consultations and to Ing. Martin Selecký for his constructive criticism and problem discussions. I would also like to thank to Tomáš Trafina, with whom I conducted hardware experiments, and to my family for their support during my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 26.5.2016

.....

Abstrakt / Abstract

Tato práce se zabývá problematikou scanování neznámého okolního prostředí pomocí LiDARu, vytvářením mračen bodů a OctoMap, detekcí nových statických překážek a úpravou plánovacích algoritmů pro vyhýbání se těmto překážkám. Při tom také popisuje konstrukci pozemního bezpilotního vozidla a integraci potřebných senzorů na toto vozidlo.

Klíčová slova: LiDAR; IMU; RTK GPS; UGV; mračna bodů; OctoMap; Point Cloud Library; RRT*; Tactical AgentFly; detekce překážek; plánování trajektorií.

Překlad titulu: Detekce překážek pomocí LiDARu a předcházení kolizním situacím

This thesis discusses the problematics of unknown environment scanning using LiDAR, point cloud and OctoMap construction, new static obstacles detection and modification of current planning algorithms for collision avoidance. It also describes construction of an autonomous ground vehicle and integration of necessary sensors.

Keywords: LiDAR; IMU; RTK GPS; UGV; pointcloud; OctoMap; Point Cloud Library; RRT*; Tactical AgentFly; collision detection; path planning.

Contents /

Tables / Figures

Chapter 1
Introduction

The use of unmanned aerial vehicles (UAV) and unmanned ground vehicle (UGV) has grown rapidly in recent years. The related technology is getting cheaper and for reasonable money it's possible to buy things we could hardly imagine few years ago. The time when drones were only an army matter is gone. Nowadays there are many emerging companies and research institutes trying to develop and use UAV and UGV technology for different purposes; e.g. filming in hardly accessible areas, security monitoring, package delivery, self driven cars etc.

This thesis originated from a project whose aim was to develop technology and methodology of modern remote sensing methods for forestry and game management based on UAV and LiDAR, which was being solved at AI Center FEE, Czech Technical University in Prague. We managed to develop SW framework in Java programming language and to integrate necessary sensors.

The goal of this thesis is to research the problematics of collision avoidance based on LiDAR range finder, construct an unmanned ground vehicle and integrate additional necessary sensors onto the vehicle. It also includes creating methods for sensory data processing, adding the functionality for real time collision avoidance, modifying current path planning algorithms and testing the results.

1.1 Thesis outline

In the beginning we shortly describe different techniques of navigation and explain why we use LiDAR and GPS. Then we present a detailed overview of all the sensors, devices and the computational unit that have to be integrated for successful data collection and robot movement. In chapter 4, the whole UGV, its construction, its characteristics and the components placement is explained. It also shows how all the devices are interconnected and how power is distributed.

With chapter 5 the SW part begins. We discuss the methods for collecting and processing data followed up by point cloud creation. Afterwards, we describe methods for point cloud processing which results into the introduction of OctoMap. Then we explain how collisions are found. Finally, we present available path planning algorithms, along with our modifications.

In the last chapter we verify the functionality of our solution by a series of experiments with real world data.

Chapter 2
Typical methods of navigation

In the last decades there has been remarkable development in localization and mapping algorithms of navigation systems for indoor/outdoor mobile robots. In indoor cases, personal service robots perform the missions of guiding tourists in museums, cleaning rooms or nursing the elderly [1]. In outdoor cases, mobile robots have been used for the purpose of patrol, reconnaissance, surveillance or exploring planets [2].

2.1 Indoor navigation

The indoor environment usually has variety of features such as walls, doors or furniture that can be used for mapping and navigation of mobile robots. Since microwaves are attenuated and scattered by roofs, walls and other objects, the Global Navigation Satellite System (GNSS) significantly loses precision. In addition, the reflections at surface and inside buildings cause multi-path signal propagation which results in another error. These very same effects are degrading all known solutions for indoor localization which uses electromagnetic waves from indoor transmitters to indoor receivers. To compensate for these problems, a bundle of mathematical and physical methods is applied. Typical example would be the use of alternative source of navigational information, such as inertial measurement unit (IMU), camera, or strength of signal from the nearest WiFi receivers [3].

There are many different systems that use some combination of the sensors described above. For example, there are WiFi-based positioning systems where the localization is based on measuring the intensity of the received signal, or grid concepts where a dense network of low-range receivers is arranged in a grid pattern throughout the space being observed [4]. There also exist methods based on time of arrival (ToA) where we measure the amount of time a signal takes to propagate from transmitter to receiver (but other calculations have to be used because of indoor signal reflection) [3]. Another largely used method is visual localization based on a system of external cameras recording small reflective balls attached to the mobile robot, such as Vicon system[1] [5]. Of course there exist also other methods, but the majority of them have one important thing in common—they are used for localization in a *known* environment.

In an *unknown* indoor environment the most frequent approach is implementing some kind of Simultaneous Localization and Mapping (SLAM) algorithm. SLAM is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time localizing the robot relative to the map [6]. SLAM consists of multiple parts: Landmark extraction, data association, state estimation and landmarks update. Each part can be solved by different way and there is a huge amount of different hardware that can be used. The main part of the SLAM process is extended Kalman filter (EKF), which always holds the current estimated position. The first step is to obtain data about the surroundings. For that a sonar, LiDAR or a vision system

[1] `http://www.vicon.com/`

can be used. Next step is to identify and extract landmarks. Landmarks should be easily re-observable from other positions, unique and stationary. Then the process of data association is performed and the same landmarks from different measurements are paired. Based on these land marks, the EKF can estimate robot position. Computational demands, together with achieved accuracy, depend on quality and quantity of used HW and desired output. It is possible to use SLAM solely with LiDAR or camera, but when additional HW is present (e.g. odometry from wheels rotation), the results should be generally more accurate[1].

2.2 Outdoor navigation

From the description of SLAM written in the last paragraph it may seem that SLAM is exactly what we are looking for. SLAM used to dominate the area of navigating in an indoor environment because generally there are more objects present in relatively small areas. Furthermore, the areas are usually surrounded by walls which make the SLAM process easier.Since the space in outdoor environment is more "empty", it is harder to extract land marks and to estimate the current position. Although there exist methods that slove this problem, they are computationally expensive. In this thesis we follow a different approach.

For the positioning and localization of the UGV we use Real Time Kinematic (RTK) GPS. It's a technique used to enhance the precision of position data from satellite-based positioning systems suchs as GPS, GLONASS or Galileo. It uses measurements of the phase of the singal's carrier wave, rather than the information content of the signal, and it provides centimetre-level accuracy [7].

We obtain data from the surroundings around the robot from a light detection and ranging device (LiDAR). It uses light in the form of a pulsed laser to measure range of objects around and produces thousands of points per seconds. These points usually also carry some additional information such as their intensity or a number of return. With these information we should be able to generate three-dimensional model of the environment and also its surface characteristics [8].

The third and also the last absolutely necessary sensor is an Inertial measurement unit (IMU). It is attached to LiDAR and tells us, how LiDAR is positioned at the moment. Points produced by LiDAR are relative to its centre at the time of the measurement, points produced by RTK GPS are relative to base (Chap. 3.3]), and IMU measurements are relative to IMU initialisation position. These measurement must be transformed into common coordinate system in order to perform calculations upon them.

By using these three sensors and combining them together we can explicitly create 3D maps (point clouds) in real time. We should also be able to navigate in them and also know our location at every moment. Another advantage of this solution is that we can later georeference models for another applications.

2.3 Pointcloud

A point cloud is a set of data points in some coordinate system. In a 3D coordinate system, these points are usually X, Y, Z coordinates and often represent the external surface of an object. They are usually created by 3D scanners—such as the one we

[1] A Tutorial Approach to Simultaneous Localization and Mapping, `http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf`

have built. They consists of all the points that were measured so they can be very detailed, as there is very little to no compression. However, this also means that the files are very large, so storing them on a hard drive or viewing them in specific SW can be computationally demanding. We also can't hold a point cloud in RAM and try to apply path planning or navigation algorithms upon it. Also sometimes recording all the points can be a disadvantage. For example, if there are moving objects (animals, people) in a scanned area, then the LiDAR measures them on different places in different time but puts them into one map, which means they look very fuzzy. Some of these problems are solved by using OctoMap 5.5. For illustration a picture of a pointcloud of our office is presented (figure 2.1).

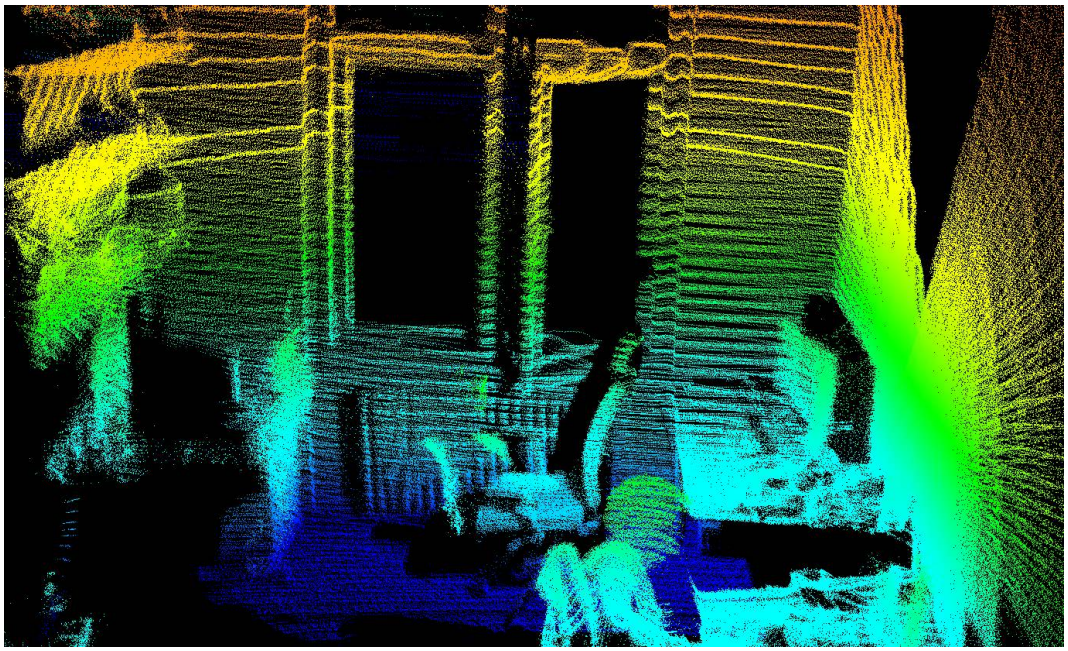


Figure 2.1. Height-colored pointcloud of our office. Notice the sitting person in the foreground, shelves on the left, computer screen on the right and window in the background.

Chapter 3
Sensors and other devices

In this chapter we describe all the sensors and devices that we use on the platform.

3.1 LiDAR

LiDAR Velodyne VLP-16 (figure 3.1) is about 7 cm height cylinder with a diameter of 10 cm. It's capable of measuring objects distant up to 130 m with precision of 3 cm. It weighs about 900 grams and it produces 300 000 points per second. It detects multiple laser beam returns which along with measured intensity can help to identify surface characteristics of a scanned object (in case of multiple returns typically windows or leaves). It has 360° horizontal field of view and 30° vertical, ±15° up and down, with 2° vertical and 0.1° - 0.4° horizontal angular resolution.

The LiDAR communicates via Ethernet. It uses its own protocol, which we had to implement, and sends packets over UDP as a broadcast or to a specified IP address.

Figure 3.1. Velodyne Puck VLP-16

3.2 IMU

Inertial Measurement Unit Microstrain 3DM-GX4-45 (figure 3.2) provides a wide range of triaxial inertial measurements. It includes direct measurement of acceleration, angular velocity or atmospheric pressure. Sensor measurements are processed through an extended Kalman filter algorithm to produce high accuracy computed outputs [9]. This device is highly customizable and for the best outputs it must be configured properly. Configuration includes setting initial heading, magnetometer calibration or output frequency. There are more output modes but we are using just the basic one which includes roll, pitch & heading. The computed output accuracy is 0.25° for roll & pitch and 0.8° for heading.

Although the IMU integrates also a GPS receiver, therefore it could give us the absolute localization, it is a standard GPS device whose accuracy is about 2 m. That

deviation is too much for our application. We need to know the location of the laser as precise as possible in every moment. Thus we only use IMU for measuring axis inclinations by which we rotate laser data.

The IMU communicates through a serial port using its own proprietary protocol which has also been implemented to the mutual framework.

a) b)

Figure 3.2. Microstrain 3DM-GX4-45

3.3 RTK GPS

For centimetre-level relative positioning accuracy we use Piksi RTK GPS from Swift Navigation (figure 3.3). It is a low-cost, high-performance GPS receiver with real time kinematics (RTK) functionality. From an architectural point of view, RTK consists of a base station, one or several rover users and a communication channel between them. The technique expects that in the neighbourhood of a clean-sky location, the main errors in the GNSS signal processing (satellite clock bias, satellite orbital error and the ionospheric & tropospheric delay) are constant, and hence they cancel out when differential processing is used. RTK devices measure carrier signal, instead of receiving the information carried (coarse-acquisition code)[1], because the phase of the carrier is changed approx. 1000 more often the C/A. However, the processing of carrier measurements is subject to so-called *carrier phase ambiguity*, an unknown integer number of times the carrier wave length, that needs to be fixed in order to rebuild full range measurements from carrier ones. That is done by re-broadcasting and exchanging the phase of the carrier between two reference stations. Then there are several different methods that can use these information and fix the ambiguity. Piksies are based on single frequency measurements with long convergence time about 10 minutes, because they only receive data at one frequency (most satellites transmit at two)[10].

It's worth mentioning once again that by using these devices we get centimetre-level accuracy but only relative to the base device. We could achieve absolute positioning by putting the base to a known location and then convert the points in X-Y-Z coordinate system to the geographic coordinate system, but that is not the goal of this thesis and we do not need it.

We use two of these devices, one as a rover and one as a base. They both have 2 UARTs providing high-speed 3.3 V LVTTL level asynchronous serial interfaces. One UART is used for their communicate over 3DR Radios (sec. 3.4) and from the second

[1] Coarse/acquisition code is a code with length of 1 millisecond, by which it modulates the carrier frequency of GPS signal. Each satellite has its own unique pseudo-random code.

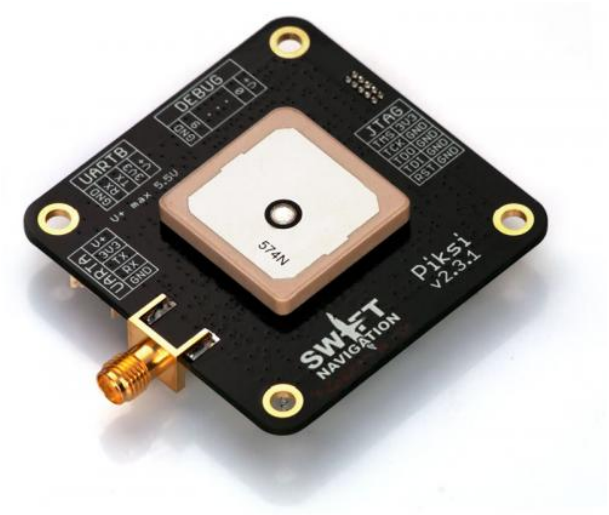


Figure 3.3. Swift Navigation Piksi RTK GPS

data is read by rover. Data sent correspond to The Swift Navigation Binary Protocol which we've implemented in the mutual framework.

3.4 3DR Radio

3DR Radio V2 (figure 3.4) is a small, light and inexpensive open source radio platform. Its working frequency is 433 MHz. The radio can be configured by AT & RT commands. In the documentation it was stated that the radios typically allow ranges over 300 m without any configuration, but with the "out of the box" settings we weren't able to get past 25 m. After some experimental testing we managed to reach slightly over 100 m distance by reducing the air speed[1] parameter from default 128 kbps to 64 kbps and increasing the transmit power from 11 dBm to 20 dBm. However, we still aren't able to get near 300 m or even more which should be possible with the customized settings we applied, at least according to the documentation. We blame these problems to the fact, that we were testing the radios and also the whole platform in Prague (at Karlovo náměstí, Letná or Strahov) and these locations are all very noisy, because this frequency is often used by a lot of short range devices such as remote car keylocks or smart home electronics. To further increase the range we could use similar radios working on a different frequency or mount antennas with higher gain.

3.5 Microhard nVIP2400

For our application, we need to be able to communicate with the onboard computer to execute control commands and to monitor the behaviour. Since the rover is moving, the only thing that comes into consideration is some kind of wireless module. At first, we were using two XBee[2] modules. However, they only have one UART port for communication (which was sufficient for the system console) but once we needed to

[1] Air speed controls the rate that the data flow through the air. The lower the air speed, the more robust the modulation can be which results to greater range. However it lowers the amount of data that can be transferred.

[2] XBee is a small chip (3 x 3 cm) that communicates on 2.4 GHz frequency using the IEEE 802.15.4 networking protocol. For more information look at [http://www.digi.com/lp/xbee]

Figure 3.4. 3DR Radio communication modules

connect to the sensors (mainly LiDAR) directly and not through the onboard CPU unit we needed an Ethernet port. So we started looking for more robust solution and ended up using Microhard nVIP2400. Moreover it has a better transmit power and range and it would also allow us to bridge internet connection from ground station (GS) to rover, if desired.

a) front side b) back side

Figure 3.5. Microhard nVIP2400 communication modules

The nVIP2400 is a high power broadband Ethernet & Serial Gateway. It is equipped with an Ethernet port, support for 802.11b/g devices and dual serial ports for integration with RS232, RS485 or RS422. It is designed to be shock-resistant. It carries data on 2.4 GHz frequency and uses Orthogonal frequency-division multiplexing (OFDM). The maximum data rate between two modules is up to 54 Mbps [11]. Theoretically that should make it possible to stream data from sensors directly to a GS in real time. However this data rate dynamically changes depending on SNR and would only keep this high only in a close distance or in an environment with low noise.

3.6 ArduPilotMega

ArduPilot Mega (APM) is an open source unmanned vehicle control board, able to control autonomous multicopters, fixed-wind aircraft, traditional helicopters and ground rovers. It is based on the Arduino open-source electronics platform [12]. It includes 3-axis gyroscopes, accelerometers and magnetometers and also requires a GPS to be

connected to it in order to work. APM board is used to control the vehicle during tests via the RC controller (we use Aurora 9) or during automated tests where the control commands are sent from the onboard CPU. We use the *APM:Rover* firmware on the board. It supports 8 RC channels with 4 serial ports, it can hold hundreds of 3D waypoints and it logs missions to the internal 6 MB memory. The logs can be later exported to KML files and opened in programs such as Google Earth.

Figure 3.6. ArduPilotMega 2.6

3.7 Toradex Iris and Colibri T30

Toradex Colibri T30 is a computational unit that runs the software for data acquisition an processing from all the sensors. It's a SoC based on NVIDIA Tegra 3 quad-core Cortex-A9 equipped with 1GB DDR3 RAM (32 bit) and a 4GB eMMC FLASH. This module is placed into the Iris carrier board that offers many connection interfaces [13]. We use 10/100 Mbit Ethernet for LiDAR connection, 3 UARTs RS-232 (for IMU, RTK GPS and system console) and USB OTG for GS connection during laboratory testing. Collected and processed data are logged to microSD card. Even though the sum of all data rates from sensors is not bigger than the maximum speed of a normal Class 10 card, during the experiments we found out that we need to use an UHS 3 card in order to be able to write all the data and prevent buffers overfilling. The CPU must switch processes quite frequently which results in peaks while writing the data. Also the settings of garbage collector and JVM had to be modified for the application to run more smoothly. To occupy the least RAM possible, the computational unit is running our own customized Linux Angstrom distribution that includes predominantly only the packages that are needed.

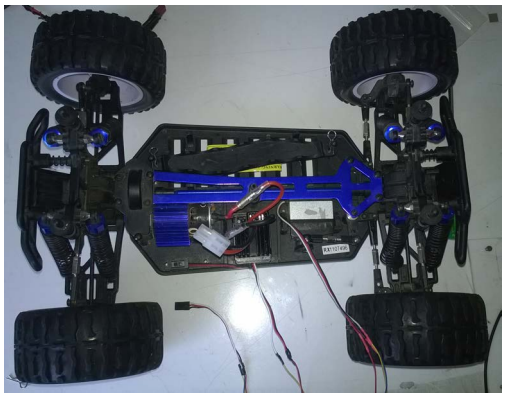
a) Iris carrier board b) Colibri T30

Figure 3.7. The computational unit—Toradex Colibri T30 and Iris carrier board

Chapter 4
Rover platform

During the research a need for an UGV platform arised and 1:8 model of RC car was acquired. It's equipped with brushed motor Himoto RC540 26T with electronic speed controller WP-1040. We detached its roof cover and we built our own platform on the chassis instead (figure 4.1).

a) an RC car



b) rover chassis

Figure 4.1. An illustrative figure of a similar RC car we used and its chassis

4.1 Communication layer

Figure 4.2 shows how to connect all the sensors and devices mentioned in the previous chapter. However, to do that physically, a mounting platform had to be constructed. While designing it we needed to consider sensors' coordinate systems and their inclination. We also had to think about possible magnetometer interference caused by rotating LiDAR mirrors and by the proximity of electronic or metal parts. On top of that, all GPS antennas (we have three[1]) should be placed as high as possible with the best possible unobstructed view around. Microhard modem is connected to Toradex by two means—RS-232 serial line and Ethernet cable.The former allows us to be connected directly to the system console, so it can be controlled without a delay and also to see Linux core debug messages. The latter is used for eventual transfer of bigger data or internet feeding. The designed platform is made from plastic and has 2 floors.

On the first floor (figure 4.3 and 4.4) we had to put the devices on both sides. On the bottom side, there is a 5-port 10/100 Mbps Ethernet switch. Because we would like to implement different GPS device (Novatel OEM628) in future, we've already reserved a place for it next to the switch. On the top of the first floor there is APM, Toradex, Piksi Rtk, the 9 channel radio receiver Optima 9 and Velodyne VLP-16 interface box. We put the CPU onto this floor partly also for the reason that it is covered from top and potential rain shouldn't get to it. In the back there can be seen a metal holder

[1] We have three GPS antennas, because IMU, RTK and APM each need one in order to work.

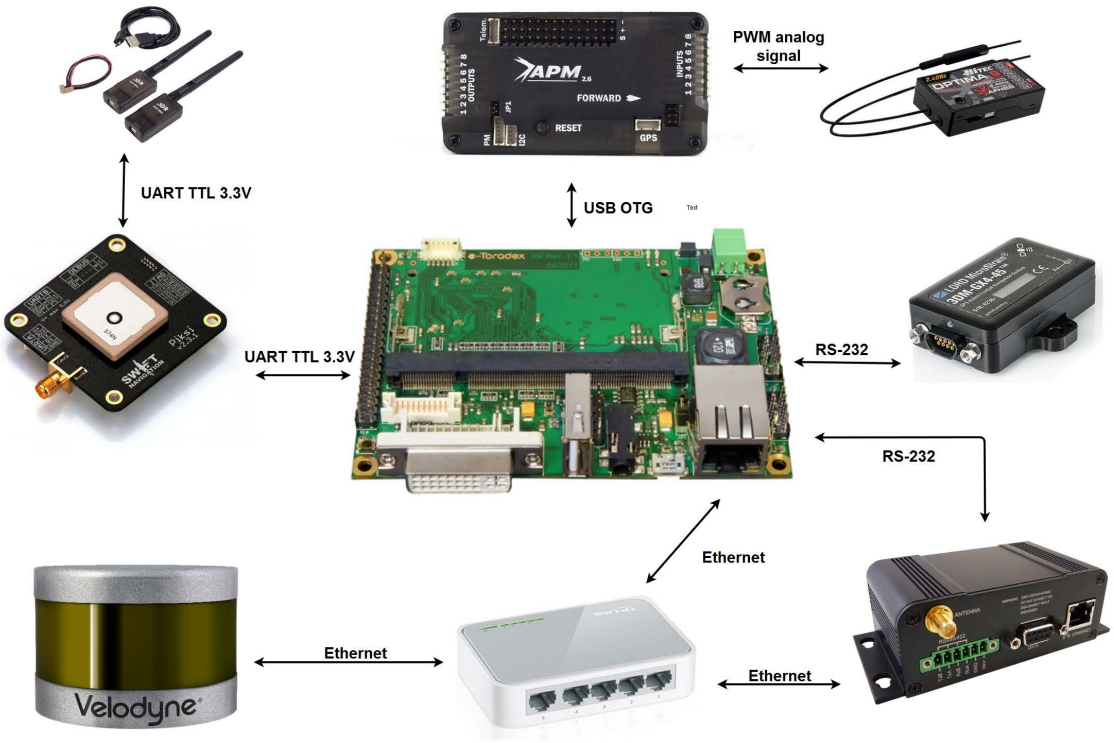


Figure 4.2. Interconnection scheme of used devices

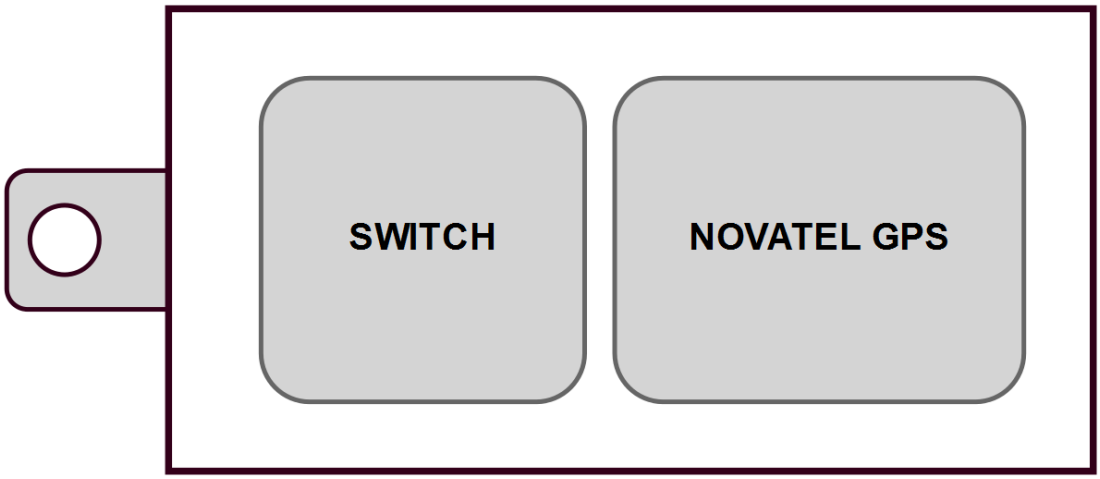


Figure 4.3. Design of bottom of the first floor of the mounting platform.

with a hole. It's intended for a stick which carries the RTK antenna which will then be approximately 30 cm above the rover. This antenna is placed so high because we need the RTK GPS to be as accurate as possible.

On the second floor (figure 4.5) there is a place for Microhard modem, 3DR Radio, IMU GPS antenna, 3DR GPS antenna (APM antenna) and Velodyne LiDAR. That must be positioned so that we get the maximum from its view. The antennas and other sensors aren't a problem for LiDAR, because we are not processing data closer than one meter. The metal plate in the back is the RTK antenna holder again. The similar plate

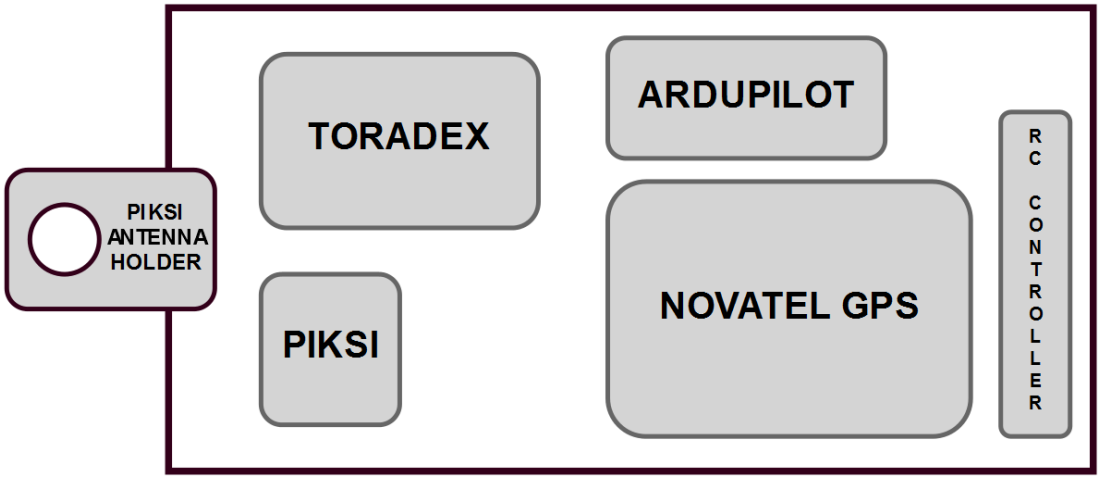


Figure 4.4. Design of top of the first floor of the mounting platform.

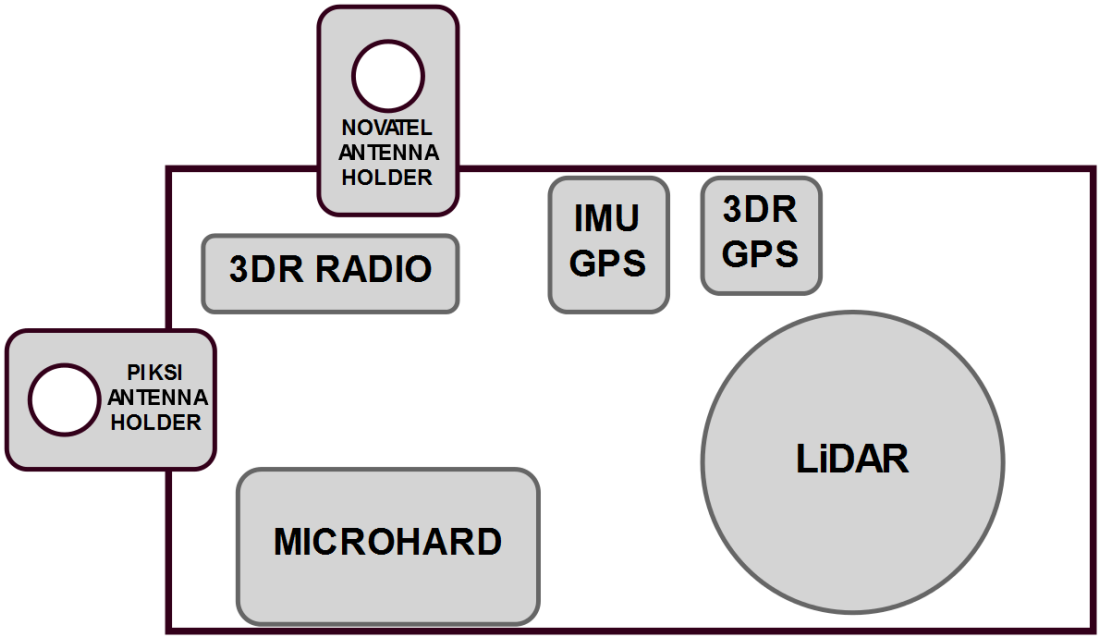


Figure 4.5. Design of second floor of the mounting platform.

on the left side of the second floor is a holder for an advanced[1] antenna for Novatel OEM628.

4.2 Power

The motor and the steering servos are powered from a NiMH 7.2 V / 3300 mAh battery. The ESC has 5 V / 2 A battery eliminator circuit (BEC) by which it supplies power to the APM and the RC receiver. However, we have a lot of devices onboard and this BEC wouldn't be sufficient (see table 4.1). Moreover, the devices are much heavier (approx. 5 kg) than what the RC model was designed for and sometimes there happen

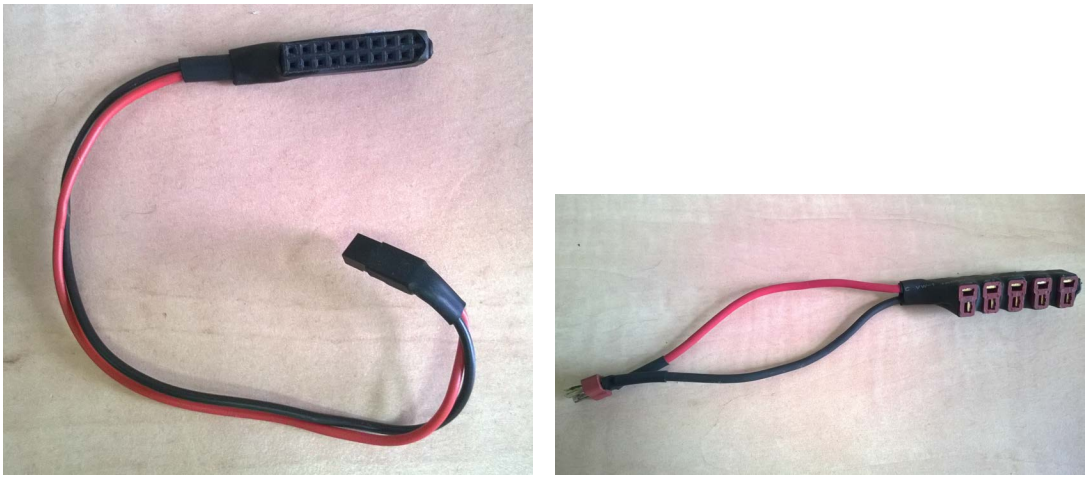
[1] With an ongoing GPS modernization, all the GPS satellites launched since 2005 don't only transmit coarse-acquisition code on the original L1(1575.42 MHz) frequency but they also transmit on L2 (1227.60 MHz) frequency which allows the receiver to calculate ionospheric corrections and get better accuracy [14] [15].

to be high energy peaks (caused by motor) during which the voltage drops resulting into APM restart. We can also divide the devices into to groups: those with an input voltage of 5 V and those with input voltage of 12 V.

Device	Max power consumption [W]
Microstrain IMU	0.9
Piksi GPS	0.6
Microhard modem	2
Velodyne LiDAR	8
switch	0.9
fan	4
Toradex Colibri T30	6

Table 4.1. Power consumption table. Values taken from devices product sheets.

Therefore, power for all the other devices is supplied from another battery—3-cell LiPo 11.1 V / 4000 mAh. We created our own power distribution splitters for both 5 V and 12 V (figure 4.6) to reduce the amount of power cables. The 12 V splitter is connected directly to the battery and powers LiDAR, switch, Microhard modem, cooling fan, a LED indicator and Toradex, The 5 V splitter is connected to Toradex 5 V output pin and delivers power to IMU and Piksi GPS.



a) 5 V b) 12 V

Figure 4.6. 5 V and 12 V power distribution splitters.

Because complete discharge can permanently damage both of the batteries, we had to make sure that would not happen. Concerning the NiMH battery, the ESC will take care of that. It reduces engine power when voltage drops below 4.5 V and completely shuts off the battery when it drops below 4 V. For the LiPo battery we've made a LED indicator (figure 4.7) from 4 Zener diodes that is able to indicate 5 voltage levels: under 9 V, 9-10 V, 10-11 V, 11-12 V and above 12 V.

Constructed rover is shown in figure 4.8.

Figure 4.7. Voltage LED indicator

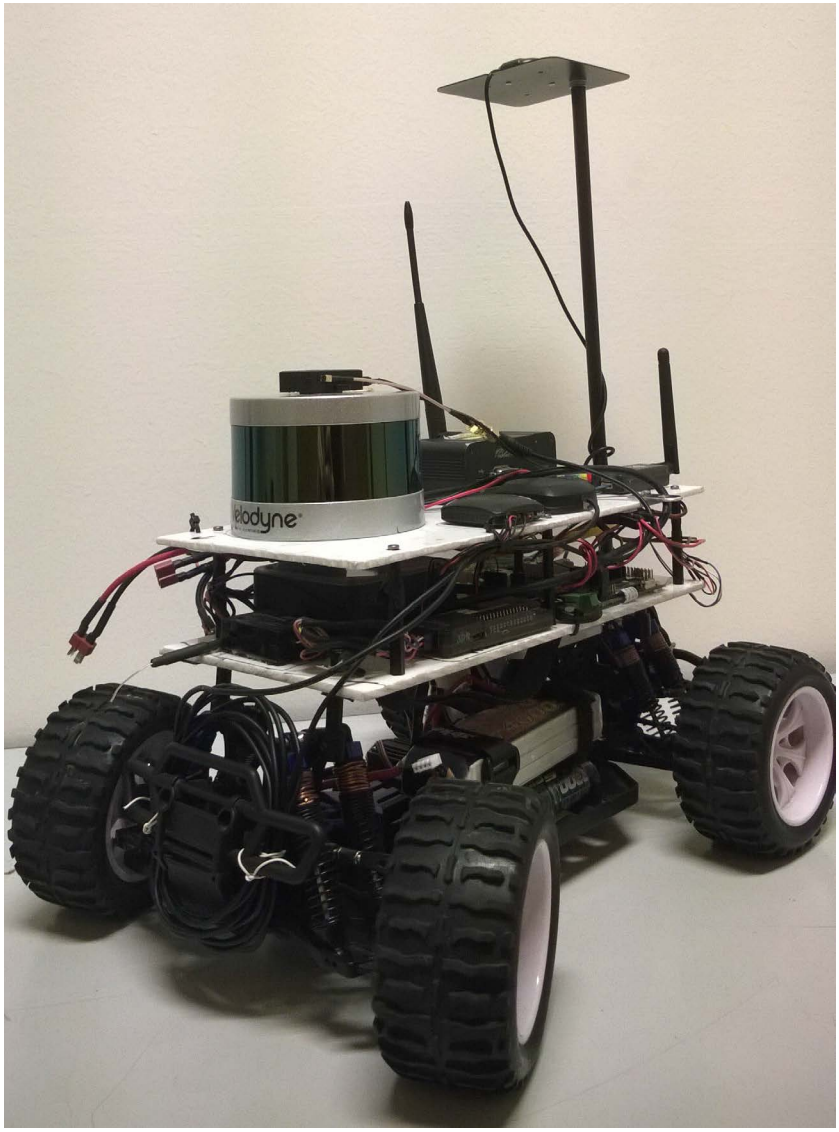


Figure 4.8. Complete rover. Model used for all tests.

Chapter 5
Data processing and representation

We've written the SW framework for processing data from sensors in Java. An emphasis was placed on modularity enabling us to easily parametrize or swap them. These modules are discussed in this chapter. They process incoming data, which ends in a creation of a point cloud (set of data points in 3D space representing external surface of an area or an object). An abstract scheme of the system architecture is presented in Fig. 5.1.

5.1 Initialization phase

When the application starts, the configuration file is loaded and the rest of the application behaves according to it. That relates to IP addresses and other communication ports of the sensors, mounting offsets and desired outputs. Then the sensors are initialized and data acquisition checker is started. That part assures that all necessary sensors are connected and sending valid data. If the initialization passes successfully, then the starting procedure is finished.

5.2 Time synchronization

The final processed point that is about to be put into a 3D map is a result of processing data from three sensors—LiDAR, IMU and RTK GPS. Every sensor sends data with a different frequency, has a different (or its own) time system a is connected to the CPU in a different way. Therefore the time it takes data to reach Toradex differs. Also it is impossible to determine, when will data be processed. Because of these reasons we've implemented two-step time synchronization.

5.2.1 Time systems shift correction

All calculations are done in system time of Toradex. LiDAR VLP-16 has its own time system, that starts with time 0, when LiDAR is started. Microstrain IMU uses GPS time that it obtains from satellites. To convert these time systems to Toradex system time, the time synchronization module is used. It measures the difference between Toradex system time when data were received at the Toradex side and data creation time in sensor time system. By these measurements we should obtain the time shift between individual time systems. However, these time shifts are very inaccurate, because the transmission time is protracted by transmission lines data flow control and Toradex and Java buffers. Therefore we are constantly repeating this measurement for 10 seconds and the lowest values measured are declared as the time shift. These values are then added to data creation time in sensor time system which gives us data with Toradex system time. These calculations are performed for LiDAR and IMU. Piksi GPS uses the same GPS time as IMU so its data are shifted in the same way. Possible difference of different GPS time on different satellites is in nanoseconds and that can be omitted [16].

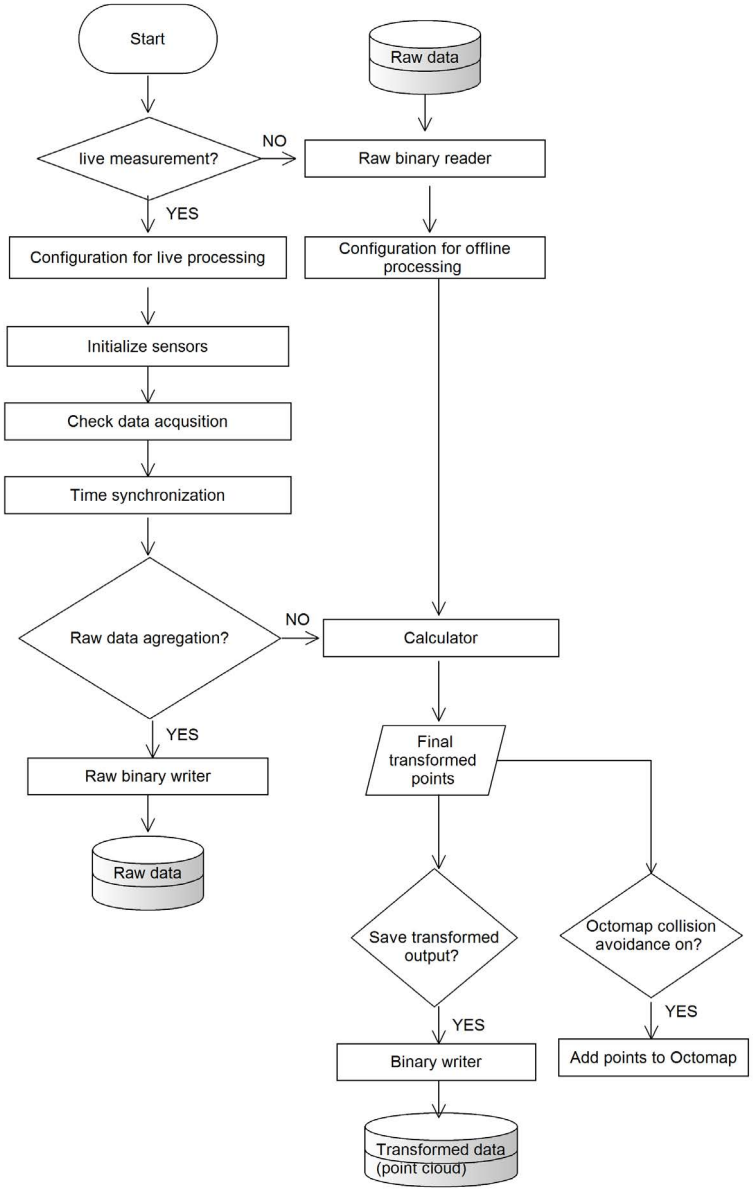


Figure 5.1. An abstract scheme of data processing necessary for point cloud creation.

5.2.2 Static time delay at transmission lines

The IMU is connected to Iris by a serial line (RS-232). The transmission at this line is protracted by a constant delay. We measured this delay using round trip mechanism to be 1409 μs. Similarly we measured time delay for LiDAR, which is connected by Ethernet and uses UDP protocol, to be 3195 μs. Once these constant delays are measured, they are deducted from the corresponding time frame shift (described in Subsec. 5.2.1) yielding us the offset value used for the correction of all the measurements from corresponding sensor.

5.3 Data aggregation

After time synchronization is completed, data gather in a data aggregation module from where they are either directly passed to an output writer, which saves them to the SD card or they are passed to a calculator, which further processes them in real time.

5.4 Calculation

When sensor data have correct timestamp, they are ready to be transformed (to recalculate the position of all the points measured by LiDAR from LiDAR local coordinate system to a common coordinate system). This can be done either in real time while sensing—and we have to do that while we want to use LiDAR data for navigation—or offline in the lab from saved data as mentioned in previous section. The latter has an advantage that a human eye may supervise the process and specifically tune it for better outcome. Also GPS positioning data can be processed with available offline correction data, which would result into more precise point cloud.

5.4.1 Time interpolation

Data from sensors are sent with a different frequency. In one second up to 300 000 points are received from LiDAR, while it's only 100 records from the inertial measurement unit and only 10 positioning records from RTK GPS. Therefore, we can not match the measurements one-to-one. The process is a as follows. For every LiDAR point two bounding IMU records are found and linearly interpolated based on time resulting in one IMU record that is finally used for the original LiDAR point. The position record from Piksi GPS is obtained using the very same principal.

5.4.2 Transformation

After the time interpolation, points can finally be localized in the common coordinate system. The localization consists of two parts. First part is compensation of LiDAR inclination and rotation which is taken care of by the IMU. The second part is LiDAR (rover) translation. We get that by subtracting the base position from the actual position.

When the transformation is done, the point is passed along. Now it is either saved to a binary file and then it can be converted to one of the two maps we use (pointcloud or OctoMap) or it's passed to the OctoMap directly, which is the case of this thesis.

5.5 OctoMap

OctoMap is an efficient probabilistic 3D mapping framework based on OcTrees. Since OcTrees are the key component of OctoMap, we describe them first.

5.5.1 OcTree

An OcTree is a tree data structure where each node corresponds to a single cube and has exactly eight sub-nodes. They are the 3D analogy of quadtrees. OcTrees are most often used to partition a three dimensional space by recursively subdividing it into eight octants (figure 5.2). The subdivision point is implicitly the center of the space it represents (i.e., the point defines one of the corners for each of the eight children). These cubic volumes are usually called voxels. The root node represents a finite bounded space

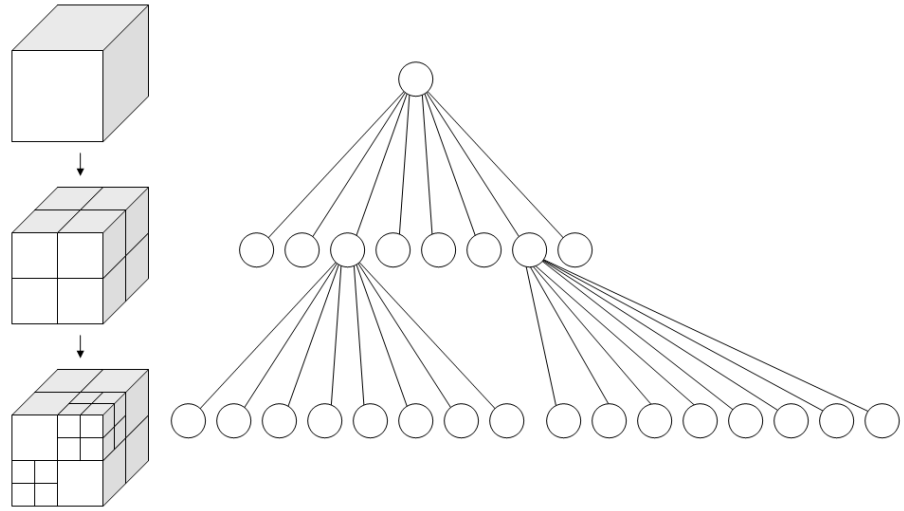


Figure 5.2. Recursive subdivision of a cube into octants and corresponding OcTree [17].

so that the implicit centers are well defined. The maximum space the map (root node) can represent depends on the resolution (on the size of the smallest cube) by formula:

$$r \cdot 2^n$$

where r stands for resolution and n for depth. For example, OctoMap implementation uses OcTrees with a depth of 16, which gives us a cubic area with a volume of 655.36 m^3 for $r = 1$ cm. The tree can be cut at any level to obtain a coarser subdivision. An example of an OcTomap for occupied voxels at several resolutions is shown in figure 5.3.

Figure 5.3. Multiple resolutions of the same object. Occupied voxels are displayed in resolution 8 cm, 64 cm and 128 cm. [18]

The OctoMap library[18] implements a 3D occupancy grid mapping approach, it is written in C++ and it is designed for:

- **Full 3D model**. The map is able to model any environment without prior assumptions about it. It models occupied areas as well as free space but unlike pointcloud it also implicitly encodes unknown areas into the map. This would be very important, if we wanted to implement autonomous exploration of an environment.
- **Updatable.** It is possible to add new information from sensors at any time. Modeling and updating is done in a *probabilistic* fashion. This is useful for getting rid of sensor noise or dynamic, moving objects.
- **Flexible.** The extent of the map does not have to be known in advance. The map is dynamically expanded as needed instead. Because it is multi-resolution a high level

planner is able to use a coarse map while a local planner may operate using a fine resolution.

- **Compact.** Thanks to the probabilistic approach, instead of storing all the points in the memory, only their probabilistic representation in voxels is stored. This allows for efficient path planning or other mathematically complex algorithms [18].

Chapter 6
Obstacle detection

For the UGV to safely move around it's necessary to avoid collisions with obstacles. In our approach, we are not trying to locate and to segment individual obstacles. The environment is represented by an OcTree which is created by OctoMap and we take all the occupied voxels as one obstacle instead. Therefore, the UGV has to move in a way it doesn't intersect with the OctoMap. The OcTree is used because of low memory requirements and probability approach in the environment modelling. And in this OcTree the collisions are found using Flexible Collision Library (FCL) (Sec. 6.3).

6.1 Data preparation

We want the OcTree to represent the obstacles as accurately as possible. However, the OcTree must not contain ground, because that is not an obstacle, so we have to separate it out. We also need to filter out noise, otherwise we would detect false collisions with obstacles that don't exist.

During the creation of an OcTree, with addition of every point a ray casting operation (Sec. 6.2) is performed. This operation is slow, but necessary because of the probabilistic nature of OctoMap. It wouldn't be a problem in a simulation but in real time collision avoidance application, we need to reduce the time it takes as much as possible. To reduce the number of voxels every ray casting operation updates, we just decrease the OcTree resolution. However, the tests showed that it is not enough with the amount of data Velodyne VLP-16 produces. Therefore, downsampling of the initial point cloud, which would reduced the number of these operations, had to be implemented.

So before we actually create the OcTree, it's necessary to prepare and modify the point cloud that we obtain from LiDAR as described in Chap. 5. The majority of the described problems are solved by using Point Cloud Library (PCL)[1]. It is an open-source C++ library containing many modules for point cloud processing. It can be used, for example, for filtering outliers from noisy data, stitching 3D point clouds together or object recognition based on their geometric appearance. It is split into more smaller modules, which can be compiled separately, therefore saving memory space [19].

6.1.1 Noise filtering

Typically, during laser scanning and point cloud creation there are measurement errors. These errors can be caused, for example, by wrong time frame adjustment of sensors or by a simplification of the calculations. So the first thing that is applied to incoming 3D points is noise filtering. Although OctoMap probabilistic nature also allows "noise filtering", it filters rather dynamic objects. Therefore the PCL is used for filtering the actual noise.

Different parts of a point cloud have usually varying densities so the point cloud has to be filtered in a statistical manner. The Statistical Outlier Removal filter module[2] of

[1] `http://ns50.pointclouds.org/about/`
[2] `http://pointclouds.org/documentation/tutorials/statistical_outlier.php`

the PCL is capable of that. It's based on the computation of the distribution of point-to-neighbours distances in the input dataset. For each point the mean distance from k neighbours are computed. By assuming that the resulted distribution is Gaussian with a mean μ and a standard deviation σ, all points whose mean distances are outside the interval

$$\langle \mu - \sigma \cdot std\_mul, \mu + \sigma \cdot std\_mul \rangle \tag{1}$$

are considered as outliers and filtered. The *std_mul* is a standard deviation multiplier and is set by user, as well as k. We set the values to $k = 50$ and $std\_mul = 0.55$. It basically means that all points with a distance larger than 0.55 of the standard deviation of the mean distance to the query point will be removed. An example of a point cloud filtration is shown in Fig. 6.1[1].

Figure 6.1. An example of noise filtration. Original point cloud in the first picture, processed final point cloud in the second picture and removed points (noise) in the third picture.

6.1.2 Downsampling

Since OctTree is in essence a grid (we use OctoMap with resolution of 20 cm), we do not need huge amount of points (a precise point cloud) to be inserted into it. The insertion would slow down data processing and bring no benefit. Instead, a grid can be inserted into an OctoMap directly. If it is dense enough, there will be no difference (grid containing 1/4 points of the initial point cloud is enough). All we need to do is to down sample the original points in a way it would still accurately represent measured point cloud in terms of its size, shape and proportional density of points.

We accomplished this by using The PCL Voxel Grid filter module [2]. First step is to divide the whole space into 3D boxes of specified size. Then there are two ways how downsampling can be done. The points can either be approximated with one point in the center of every occupied voxel or the exact location of their centroid can be computed. In the Voxel Grid filter the latter approach is used. It's a bit slower but more accurate. See Fig. 6.2 and 6.3 for an example.

6.1.3 Ground segmentation

The PCL Sample Consensus module[3] is used for segmenting ground from incoming point clouds. It's based on Random sample consensus (RANSAC) iterative model parameters fitting algorithm [20], which randomly selects a set of data points from a model and creates a sub-model (a plane in our case). Then it counts how many points from the original model would fit this sub-model according to specified parameters. This process

[1] Data source: `https://raw.github.com/PointCloudLibrary/data/master/tutorials/table_scene_lms400.pcd`

[2] `http://pointclouds.org/documentation/tutorials/voxel_grid.php`

[3] `http://docs.pointclouds.org/trunk/group__sample__consensus.html`

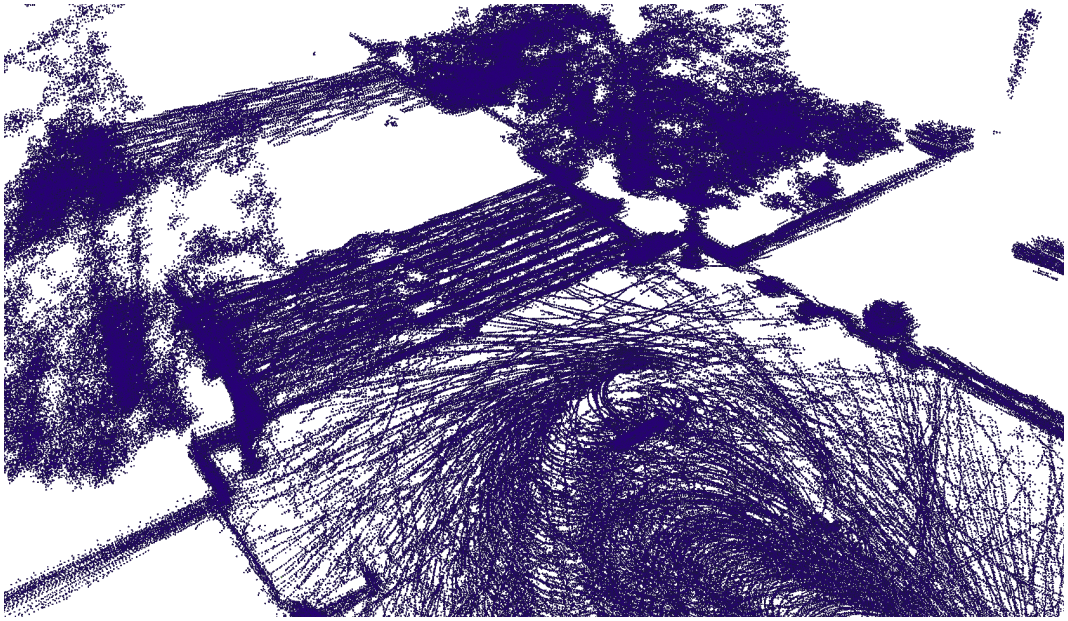


Figure 6.2. Voxel grid downsampling example of point cloud taken at Letná. The point cloud is downsampled using voxels with 5 cm edge. It contains 832 082 points.

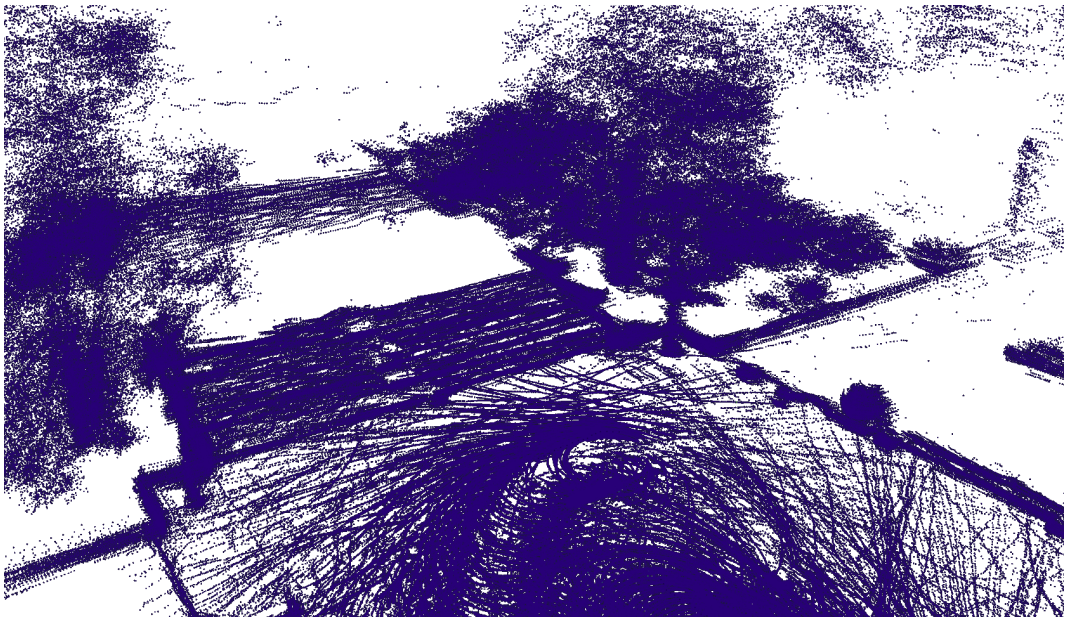


Figure 6.3. The same image as 6.2 but without the grid downsampling. It contains 2 098 699 points.

is repeated for a fixed time and then the best output is picked. The estimated plane is defined with perpendicular normal vector, thus by formula:

$$ax + by + cz + d = 0. \tag{2}$$

Parameters of this filter, that have to be set in advance in order to work properly are: maximal angle of plane inclination from horizontal plane α, maximal vertical distance from the plane v and maximal distance of the points inside the plane from each other s. An example of segmented point cloud can be seen in Fig. 6.4.

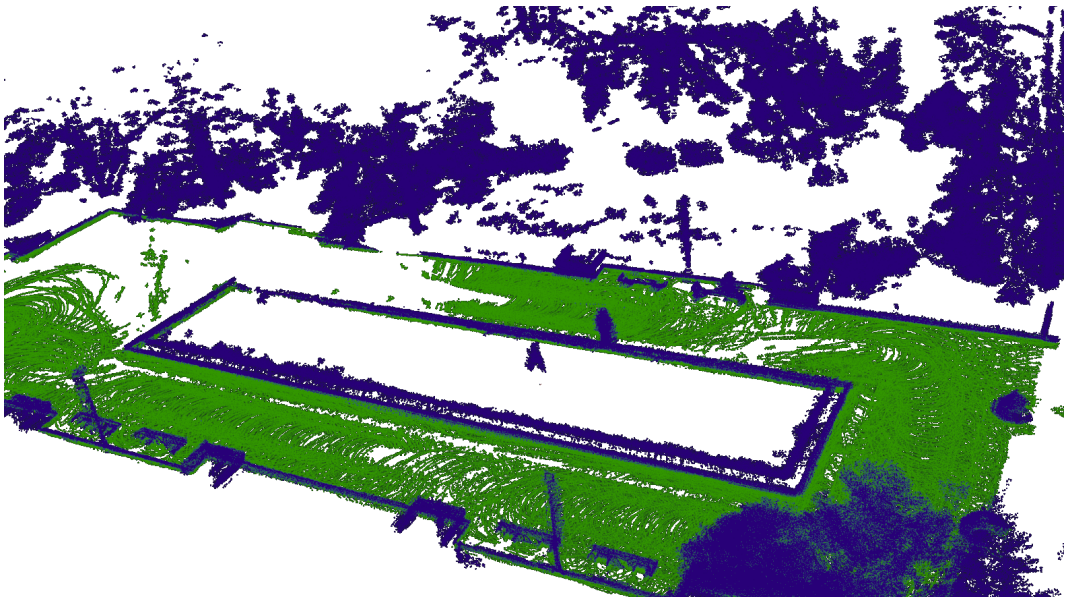


Figure 6.4. Segmentation of the ground plane from point cloud captured at Letná. Segmented points are coloured green.

6.2 OctoMap updates and ray tracing

OctoMap assumes that endpoints of a measurement correspond to obstacle surfaces and that the line of sight between sensor origin and endpoint doesn't contain any obstacles. To efficiently determine the map cells which need to be updated, a ray-casting operation is performed that determines voxels along a beam from the sensor origin to the measured endpoint. An example can be seen in figure 6.5. This operation is executed on every point that is inserted into OctoMap, so it needs to be as efficient as possible.

The probability $P(n|z_{1:t})$ of a leaf node n to be occupied after the given sensor measurements $z_{1:t}$ is estimated as

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{3}$$

where z_t is the current (latest) measurement, $P(n)$ is a prior probability, and $P(n|z_{1:t-1})$ is the previous estimation. The term $P(n|z_t)$ denotes the probability of voxel n to be occupied given the measurement z_t. The common assumption of a uniform distribution between occupied and free voxel leads to a prior probability $P(n) = 0.5$.

By using log-odds notation, Eq. (3)can be rewritten as

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{4}$$

where

$$L(n) = \log\left[\frac{P(n)}{1 - P(n)}\right]. \tag{5}$$

This formulation speeds up the map update process because multiplications are replaced by addition. Note that log-odds values can be converted into probabilities and vice versa and OctoMap therefore stores these values for each voxel instead of the probability so logarithms do not have to be computed during each update.

From Eq. (4) it's evident that if a voxel was observed as *free* for m times, then it has to be observed *occupied* at least m times again before it is considered occupied

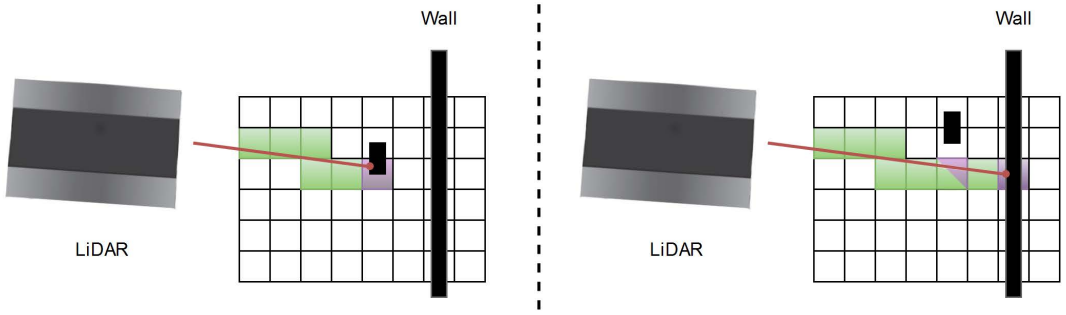


Figure 6.5. Updating OctoMap with two consecutive measurements. The black small rectangle can represent a moving object or noise in the environment. Green cells represent free space, purple occupied space, and unknown voxels are left white. The half purple half green square is either occupied or free depending on OctoMap the currently set parameters.

(assuming that *free* and *occupied* measurements have equal weight). While this property is desirable in static environments, in mobile robotics we want the map to be able to adapt to dynamic changes quickly. To ensure this, OctoMap uses an upper and lower bound on the occupancy estimation and instead of using Eq. ((4)) directly, occupancy estimations are updated according to the formula:

$$L(n|z_{1:t}) = max(min(L(n|z_{1:t-1}) + L(n|z_t)), l_{max}), l_{min}) \tag{6}$$

where l_{min} and l_{max} denote the lower and upper log-odds clamping bounds. This modified update formula limits the number of updates that are needed to change the state of a voxel [18].

In the last paragraph we considered the nodes being updated as *misses* and *hits* had the same weight. However, that was done just for the simplicity of the explanation. These values, along with the clamping thresholds, are not intended to be the same, they are strongly sensor-dependent and we had to experimentally adjust them to fit Velodyne VLP-16 and our application needs.

6.3 Collision detection

When a point cloud is processed we can finally create an OcTree. An example of one is shown in Fig 6.6.

Collision detection is done with the help of Flexible Collision Library (FCL) [1]. FCL is a collision and proximity library that integrates several techniques for fast and accurate collision detection and proximity calculations. It is designed to perform multiple queries on different model representation - including OcTree. Its four main capabilities are discrete collision detection, continuous collision detection, separation distance computation and penetration depth estimation [21]. We use the first two.

The usage is very simple. First we set the size of a block representing an UGV ($40 \times 30 \times 60$) cm. Then, in the case of discrete collision detection, we set a position of an UGV and ask the FCL if it collides with the OcTree. In the case of continuous collision detection, we set two positions and the FCL computes if there is an intersection with the environment between those two.

[1]]http://www.willowgarage.com/blog/2012/02/29/flexible-collision-library]

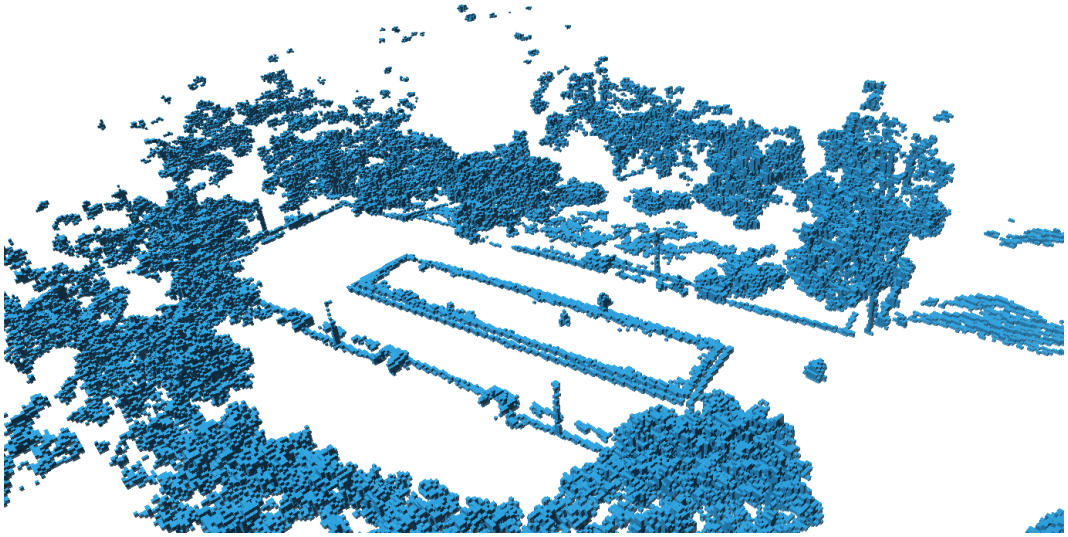


Figure 6.6. Final OcTree map of Letná. Filtered ground and noise. It is the same environment as shown in Fig. 6.4. Voxel resolution is 20 cm.

Chapter 7
Planning algorithms adjustments

To use the collision detection ability we've discussed in Chap. 6, we must integrate it into a planning framework. At AI Center, a system for simulation and command & control of unmanned aerial vehicles is being developed.

7.1 Tactical AgentFly

TAF is a system written in Java which utilizes Groovy files for its configuration. It focuses on a research and development of planning and coordination algorithms for automated data collection by teams of cooperating UAVs. It supports both persistent surveillance of a selected area and tracking of mobile ground targets. TAF is an agent-based system. An agent in computer science refers to a software entity which has its own intelligence, is situated in some environment, can sense it, and is capable of autonomous actions in this environment in order to meet its design objectives [22]. There are usually more agents and they can interact with each other. The graphical user interface of TAF can be seen in Fig. 7.1.

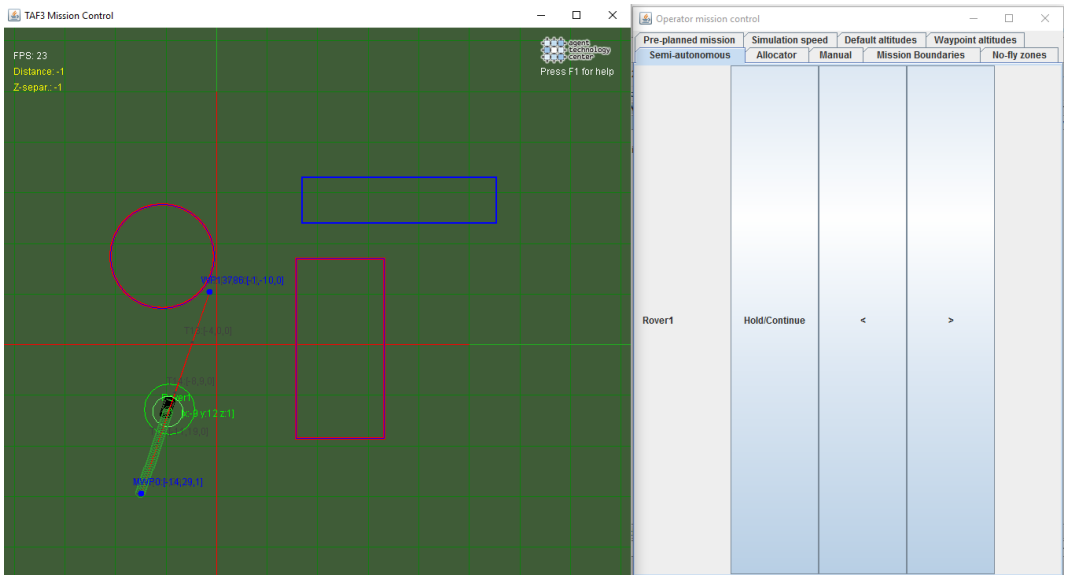


Figure 7.1. TAF GUI. Left part of the picture shows the environment. Blue dots are the waypoints, small circles along the line in front of rover represents the remaining plan. The purple circle and rectangle are obstacles which a vehicle should avoid. The blue rectangle shows a surveillance area, but it has no vehicle assigned. In the right part of the picture mission control window is shown.

7.2 TAF modification

Tactical AgentFly was developed for work with unmanned *aerial* vehicles. Since the constructed rover we used during the experiments is a *terrestrial* vehicle, not only

the existing planning algorithms but also the parts responsible for correct behaviour according to physical parameters and movement model constraints had to be modified.

An unmanned system in TAF consists of a Ground Control Station (GCS) and unmanned vehicles (agents). A GCS is a control part of the system that runs remotely on a different (often portable) computer, where a human operator can give orders (missions). It communicates with UAVs and UGVs via radio.The agents are represented by two parts:

- Aviator. An aviator contains a set of planning algorithms that together ensure a complex behaivour of an agent. It is the part of the agent that is responsible for higher level control. For example, it can add vehicle a capability of surveillance, tracking, cooperative collision avoidance etc. Since the majority of these functions aren't needed, we used settings from a hexa copter and only replaced the trajectory planner to our RRT based one; Sec. 7.2.2.
- Embodiment. An embodiment represents physical parameters, sensors and actuators of a vehicle in the system. By sensors it is meant, for example, a position (GPS) or a battery. Actuator is a device that can perform an action or operation, in our case the APM autopilot. We had to change physical parameters like minimum, optimal and maximum speed, autopilot serial port or the distance to waypoint to be marked as "passed". Embodiment also ensures mutual data exchange. It sends data from sensors to aviator, which returns back computed plans. OctoMap is provided to the system through embodiment.

During the development of the planning algorithm, we had to modify two parts of TAF system:

- Rover model for simulation. When an algorithm or a concept is being developer, it is first tested in a simulation and after proven working, it can deployed on a real platform. Since TAF offers models of planes and copters, but no ground vehicle, it had to be implemented. However, this was not the main part of our research, we basically took the copter one and changed its parameters to match our ground vehicle and it turned out to be sufficient.
- A planner. To study and implement (or modify) planning algorithms was also a part of this thesis. First, we started with modification of a simple planner included in TAF based detecting the coordinates of a collision point and trying to somehow bypass this point. That turned out to be inefficient and improper, predominantly in an environment with many obstacles. For that reason, an RRT algorithm[23] was implemented and used for trajectory planning.

7.2.1 Simple trajectory planner

Firstly, we tried to modify a simple trajectory planner already implemented in TAF system, that supported cylindrical and spherical obstacles and we added support for a block obstacle. The functionality is following: Every time the (re)planning algorithm is started, it iterates over all known obstacles, finds intersections points and tries to push them away, out of the borders of the obstacle. Regarding obstacles with circular shape, there are always two intersection points. A point lying in the middle of them is calculated and shifted perpendicular to their connection line, towards the obstacle boundary and outside. Example can be seen in Fig. 7.2.

Avoiding a block obstacle is different. When the intersection points are found, again the middle point is calculated. Now however, it is moved to the closest corner (or one of the two closest, depends on the intersection line). From this corner it connects the

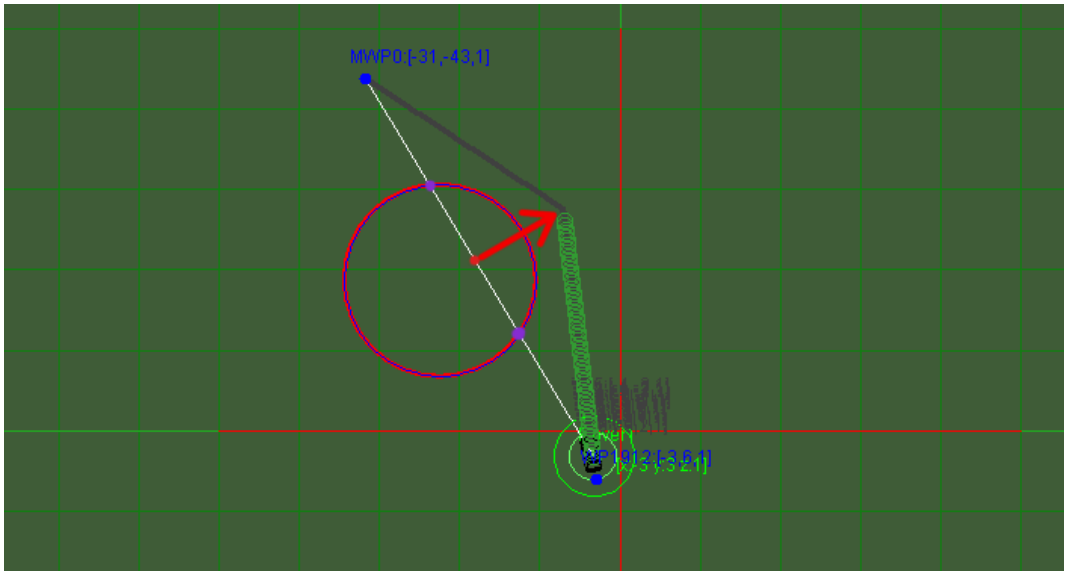


Figure 7.2. Simple collision avoidance algorithm example on a round shaped obstacle.

starting point and the end point. If a middle point is found again, previous procedure is repeated and the new middle point is moved to another corner. Example shown in Fig. 7.3.

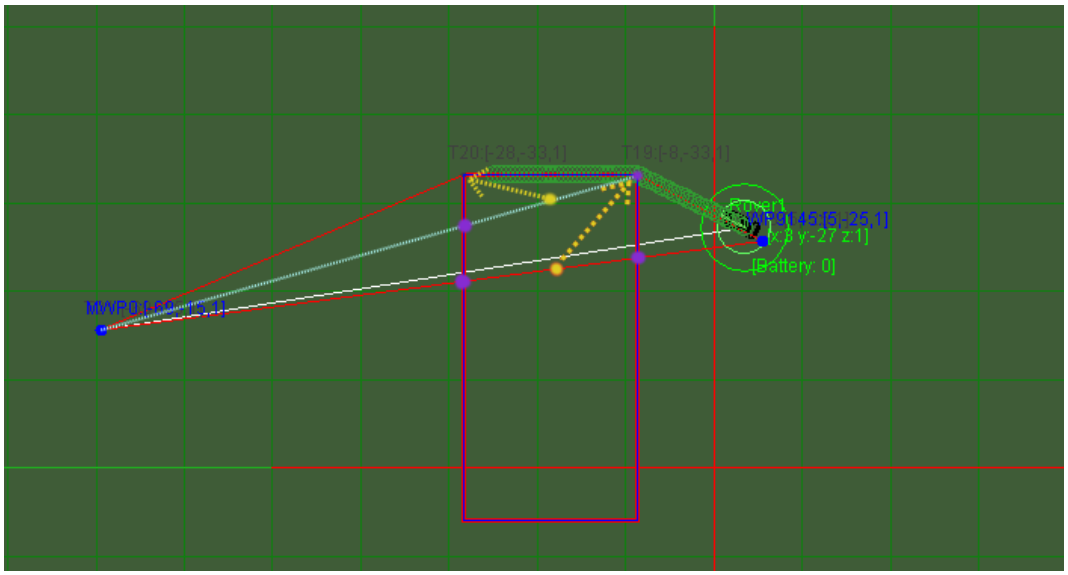


Figure 7.3. Simple collision avoidance algorithm example on a block obstacle.

This planner works when the blocks are far from each other. But if we wanted to use this planner with an OcTree, that would definitely not be that case. Let's just say an object in real life has dimensions $1 \times 1 \times 1$ m. And we use an OcTree with voxel edge length of 20 cm. That would result into 125 block cubes. Let's also say that the trajectory intersects with 100 of these cubes. The planner solves the intersection of the first cube. However, it moves the middle point into another cube, that previously was not intersected. Now by applying the same algorithm again, the new middle point could be moved back to the first cube or to any other. This would result in cycles in the algorithm, producing no or incorrect results. Of course, in the previous scenario, we could merge all the cubes into one big obstacle, but generally, that is not possible.

7.2.2 RRT* trajectory planner

The problems described in the previous section led us to an algorithm based on RRT (Rapidly-exploring Random Trees). A rapidly exploring random tree is an efficient algorithm for path planning in continuous environment with non-convex obstacles. It does so by "randomly" filling the space with points connecting them to the closest, already connected, points. The word *randomly* is put into quotation marks, because it is not complete random generation of a point within a specified area. RRT is focused on exploring places not yet visited. With this approach it should search all the imaginary parts (locations) of the environment with similar time and not prioritize any.[24]

One advantage of RRT path planners is, they don't need to know the coordinates of a collision. If there is a collision between a point and a closest point from a tree, then the point is not added. So the tree is growing randomly in the space with no collision. The algorithms need to know where the beginning and the goal is. Around the goal coordinate there is a target region, which needs to be reached because hitting the goal point by another randomly generated point has zero probability. The algorithm can be stopped when the first path from the beginning to the end is found or it can be limited by time or number of iterations and the search process may continue to possibly obtain better result.

This planner was merged with Flexible Collision Library. Since the RRT algorithm is simpler than RRT* but works on the similar principal, our modifications including FCL incorporation is described on it. The explanation of RRT with the image 7.4 is taken from [23].

- 1. The algorithm is initialized with graph that includes single vertex (beginning) and no edge.
- 2. At each iteration a point is sampled. In this step we had to make sure, that the sampled point doesn't lie in a occupied position by OctoMap. If it does, another point is sampled.
- 3. The nodes of the tree are traversed and the nearest node is found.
- 4. An attempt is made to connect the nearest node. In this step, the method for checking if two points could be connected, was implemented. A size of a vehicle has to be set. Therefore, between two points there can be a path available for a small vehicle while being no for the other.
- 5. If such a connection was successful, the sample point is added to the vertex set and the edge between it and the nearest node is added to the edge set.
- 6. As soon as the tree contains a node in the goal region, the algorithm is stopped.

Algorithm 3: RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{rand}})$ **then**
    $V \leftarrow V \cup \{x_{\text{rand}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{rand}})\}$;
return $G = (V, E)$;

Figure 7.4. A scheme of RRT path planning algorithm.

Opposite of RRT, RRT* includes a cost function—in our case distance. Also instead of further tree expansion in case of RRT, RRT* algorithm tries to improve already

found solution by generating the points closely to the found trajectory. An example of paths generated by RRT* algorithms is shown in Fig. 7.5 and 7.6. The image is taken with the help of iRRT simulator[1] created by researches from Correll Lab at the University of Colorado.

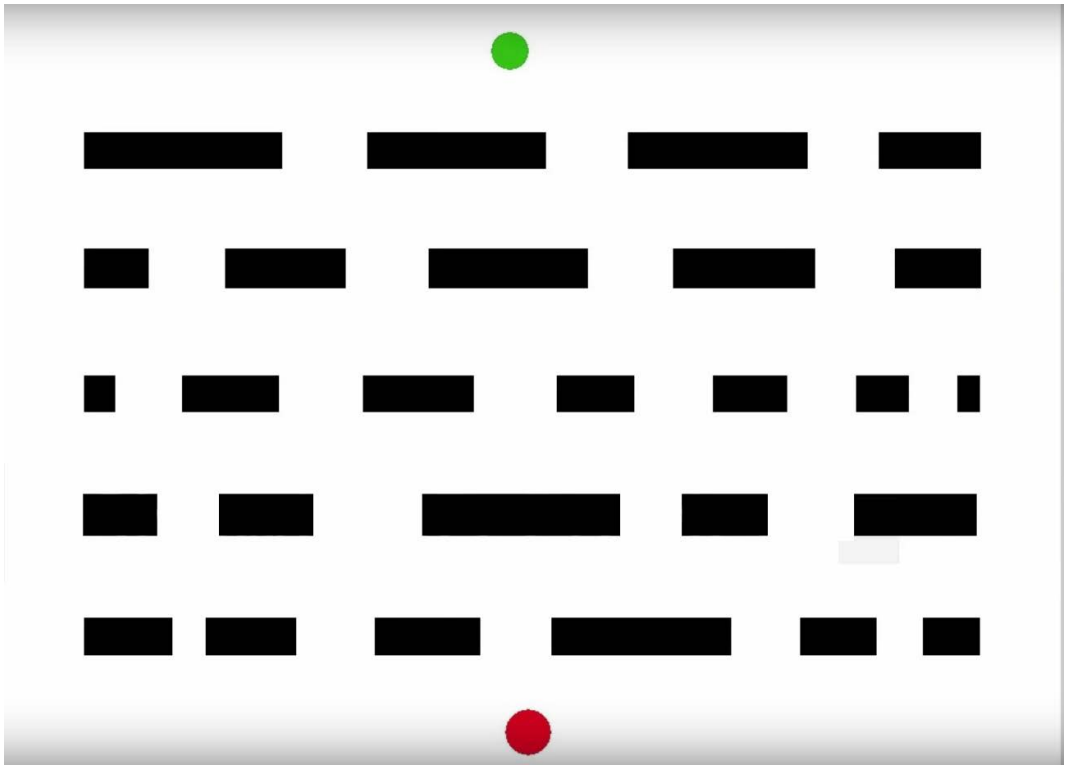


Figure 7.5. The environment in which the RRT* planning algorithm should find a path from the green circle to the red one.

[1] http://correll.cs.colorado.edu/?p=1623

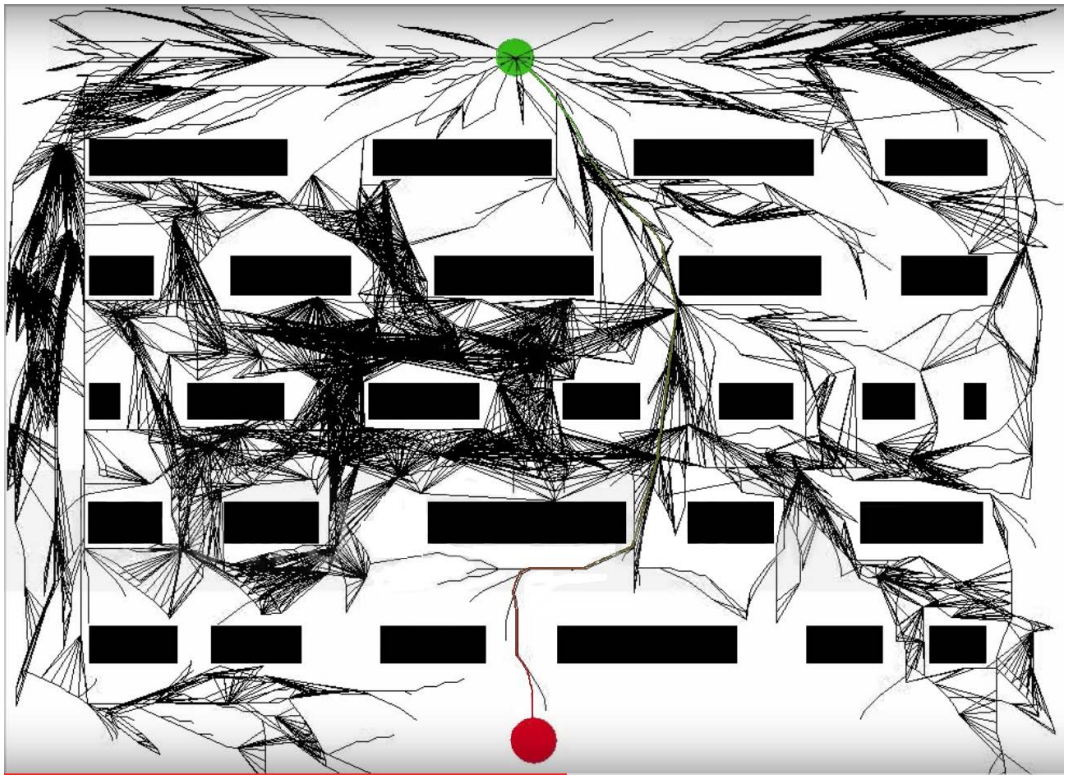


Figure 7.6. Found path between green and red circle along with the whole generated tree. This is the first path found, but it can be improved by additional iterations.

Chapter 8
Experiments

Two types of experiments were conducted. In the first one we tested point cloud creation and in the second one we tested collision avoidance with path planning. Despite the fact that the invented algorithms proved to be working, there are still several problems that prevent them to be put onboard instantly, and have to be solved. These problems are also discussed here.

8.1 Point cloud creation tests

The first point clouds of an outside environment were taken at Charles square and Strahov. The measurement were performed 10 times in each location. The best results are shown in Fig. 8.1 and 8.2.

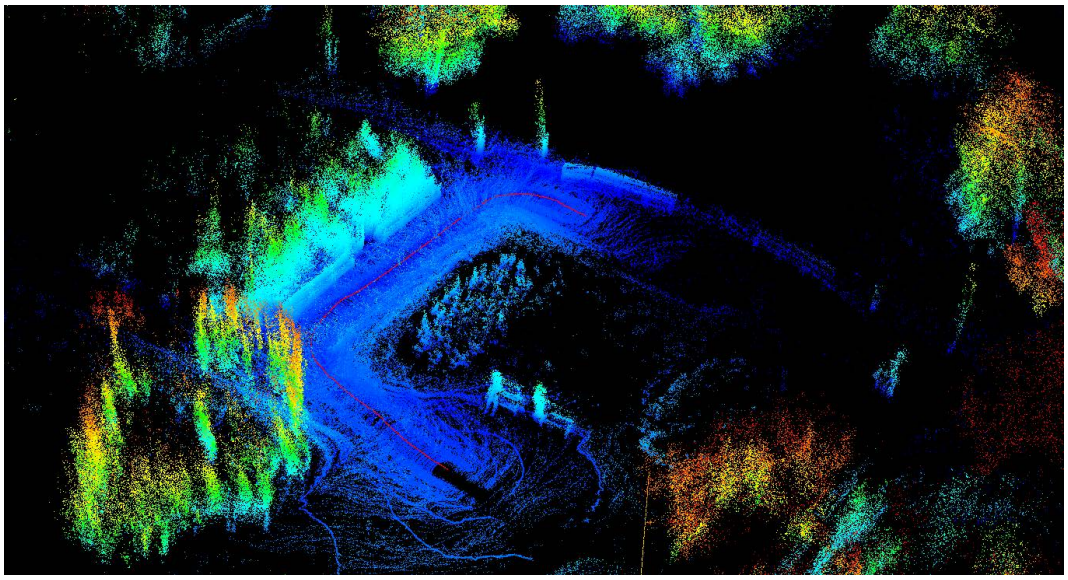


Figure 8.1. The best scan of Charles square.

After we analysed those point clouds, we discovered the first problem—using RTK float solutions of Piksi 8.1.1.

8.1.1 RTK float solution imprecision

Piksi RTK have three output types. One of them is classical single point GPS and the other two are RTK solutions—float and fixed. Which one is provided strongly depends on the number of visible satellites and weather conditions. Fixed one has centimeter-level localization accuracy, while the accuracy of the float one is between 5 to 22 cm. Two tests regarding this problem were conducted. First one shows the behaviour during statical scanning. The point cloud is shifted only in z axis while the horizontal position is held. It is shown in Fig. 8.3. In the second test the point cloud creation with moving

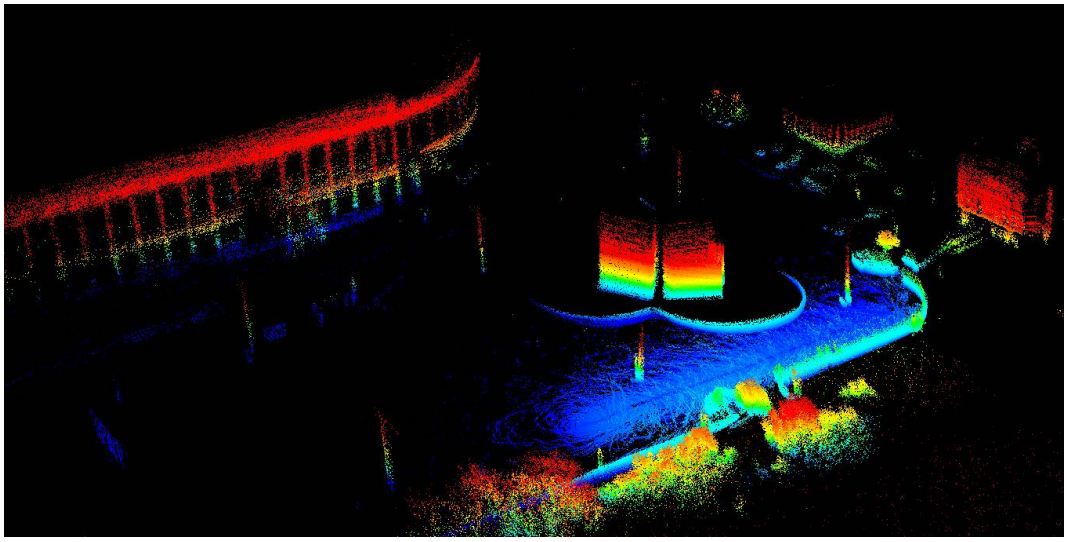


Figure 8.2. The best scan of Strahov.

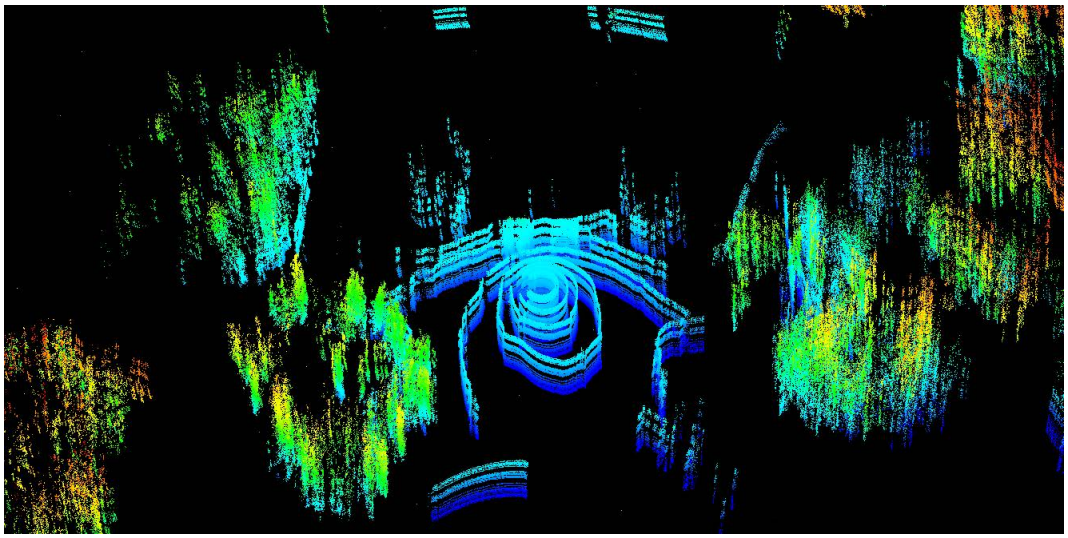


Figure 8.3. Static scan with float RTK solution. Even though the position is held in the horizontal plane and doesn't move, notice how shifted in the z direction the point cloud is.

rover and float RTK solution was examined. The result can be seen in Fig. 8.4. The point cloud is blurred in every direction.

The solution is to pair LiDAR data with fixed RTK GPS positioning data only.

8.1.2 RTK antenna offset

If a UGV moves along a straight line, the translation measured by RTK GPS is the same for both the rover and the GPS antenna. But if the rover turns, the LiDAR starts going "around" the GPS antenna. Theoretically, if the rover went in a circle with radius r around the GPS antenna, the GPS would show the same coordinates the whole time, but the centre of LiDAR could be shifted by $2r$. This is illustrated in Fig. 8.5. If we say that on the rover $r = 0.5$ m, then the fixed RTK precision, that was gained by using Piksi, would degrade back to the level of normal GPS.

The consequences of this problem are shown in Figures 8.6 and 8.7. The objects are shifted in a pattern corresponding to rover trajectory. This was solved by adding

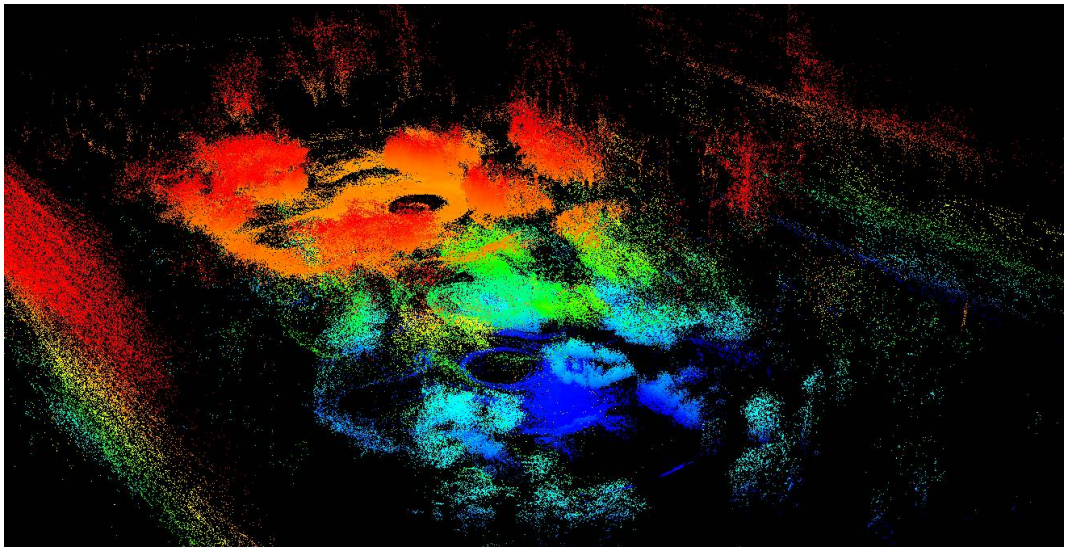


Figure 8.4. Scan with float RTK solution. In this case the rover was moving so the translation is applied. The point cloud is fuzzy in all directions and absolutely not suitable for any application.

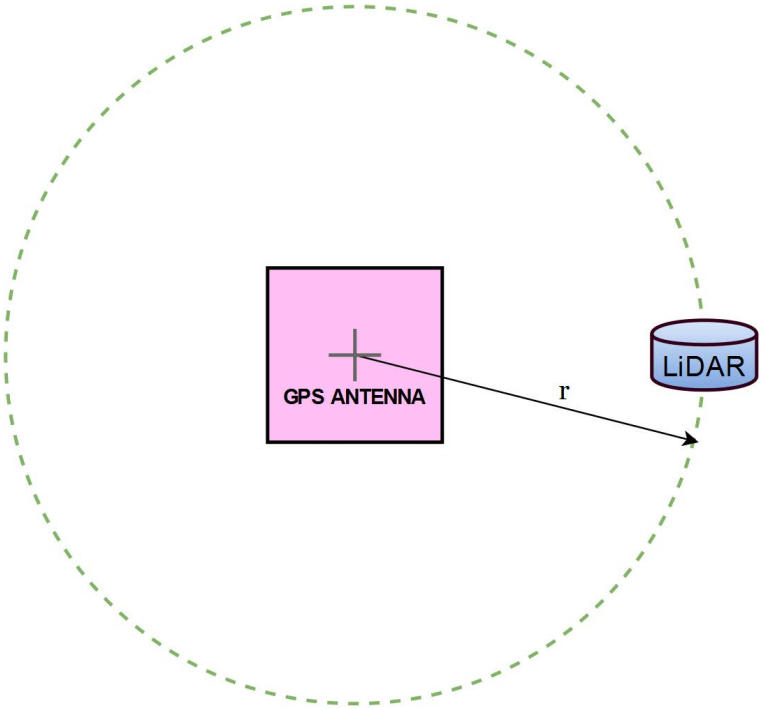


Figure 8.5. Antenna offset from LiDAR.

an antenna offset into the transformation process of point cloud creation described in 5.4.2. Refer to [25] in case of deeper interest.

It's worth to mention that at this point of the development, all the outputs come from real time processing. All the models above were computed onboard. While this saved some disk space and largely reduced the amount of data being written to memory card, it also made the debugging and post processing much harder. In the end we decided to simplify output operations (reduce redundant data and change from double types to float) and added the possibility to write directly all data from sensors to the card,

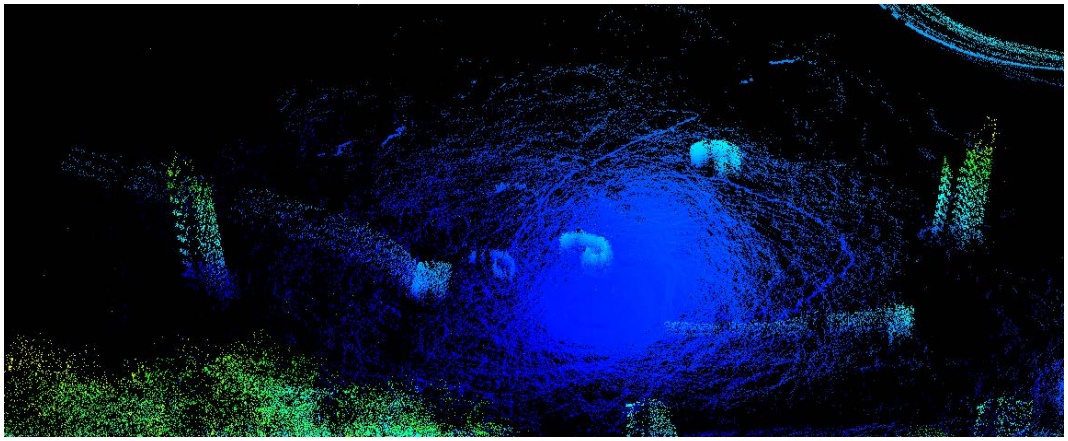


Figure 8.6. Shifted objects as a result of antenna offset.

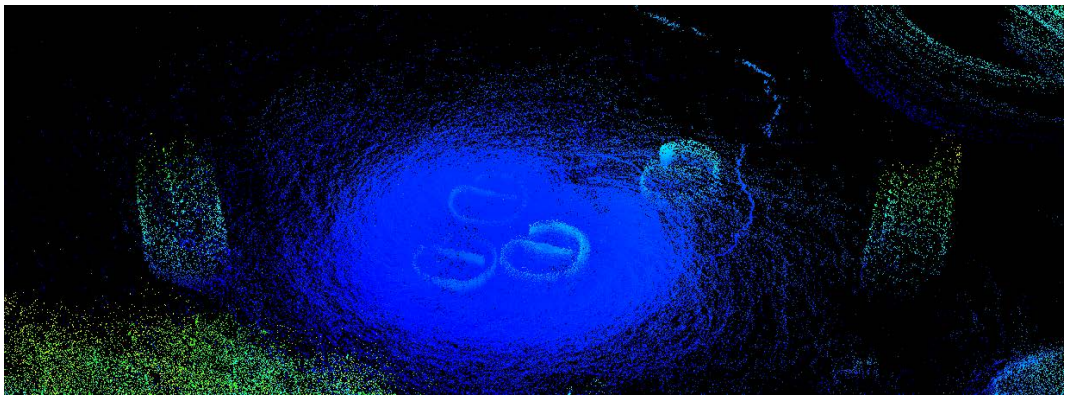


Figure 8.7. Shifted objects as a result of antenna offset. The rover took a "reverted G" trajectory as can be seen in the picture.

leaving the post processing for later. Of course this should be enabled only when the goal is accurate point cloud creation, not a real time collision avoidance.

8.2 Improved point clouds

When the problems mentioned in previous section were solved, we identified another issue—difference between IMU and RTK coordinate system. The following experiments were performed in two locations. Photographs 8.8 and 8.9 show how these locations look.

8.2.1 Difference between IMU and RTK coordinate system

X and Y axis of IMU coordinate system don't match the same axis of RTK Piksi coordinate system. Piksi x axis is aligned to south-north direction. According to the documentation IMU yaw is 0° when pointing to north. But tests showed this angle varied ±30°. Magnetometer measurements and calibration tests were performed, but the problem was not fixed. Resultant point clouds are shown in Fig. 8.10 and Fig. 8.11.

Although the problem was not fixed in terms of finding an automated real time solution, we managed to fix the pointcloud in offline processing. Corrected point cloud are presented in Fig. 8.12 and 8.13. These two point clouds also served for following experiments regarding collision avoidance (Sec. 8.3).

Figure 8.8. Photo of one of the areas where experiments described in Sec. 8.2 and Sec. 8.3 were conducted.

Figure 8.9. Photo of one of the areas where experiments described in Sec. 8.2 and Sec. 8.3 were conducted.

8.3 Collision avoidance

As an input point cloud data sets we took the corrected point clouds discussed in Subsec. 8.2.1. Upon them, 2 different tests were conducted. In the first test, the processing (noise reduction, downsampling, ground segmentation; see Chap. 6), as well as the OctoMap creation, was prepared in advance. Then the OctoMap was loaded and path planning with collision avoidance was tested. The area is a flat, square-shaped, with few placed obstacles marked in red ellipses in Fig. 8.8.

In the second test, point cloud processing, along with the OctoMap creation, was run in parallel with path planning and collision avoidance simulating behaviour in real

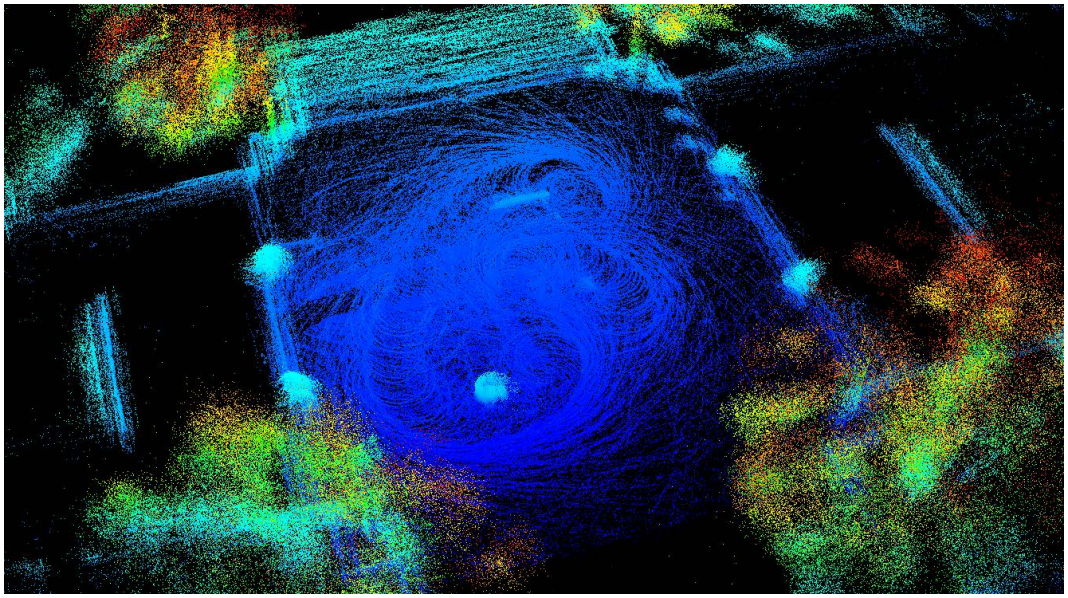


Figure 8.10. The result of point cloud creation assuming that IMU and RTK GPS coordinate systems are identical. In this picture they differ by 5°.

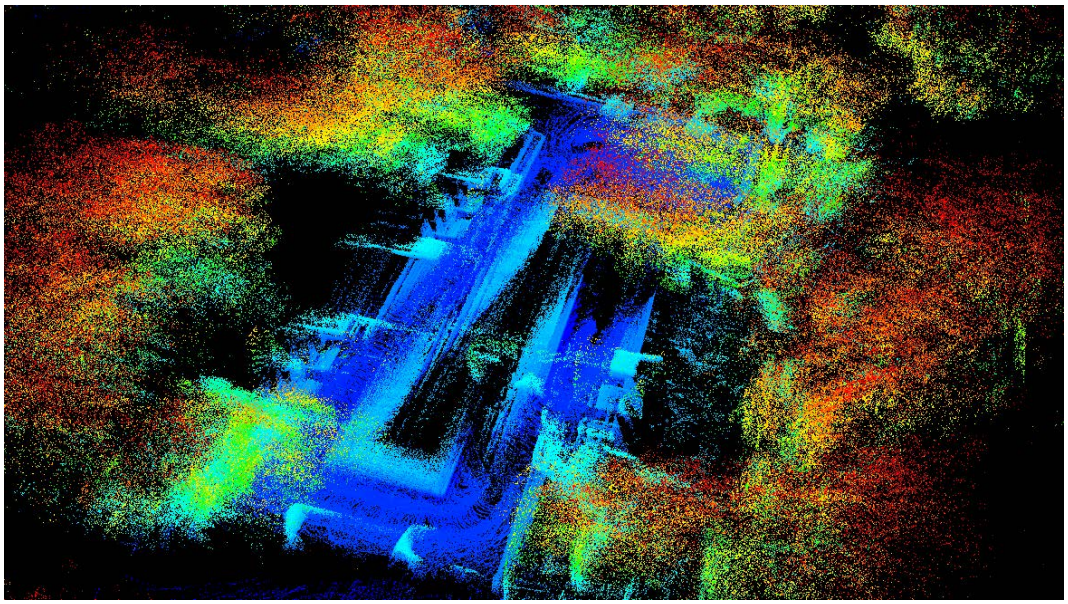


Figure 8.11. The result of point cloud creation assuming that IMU and RTK GPS coordinate systems are identical. In this picture they differ by 11°.

environment.It is a rectangle-shaped area with granite benches around and a 60 cm high flowerbed inside. Fig. 8.9 shows how the place really look.

8.3.1 OctoMap prepared in advance

This test consisted of two parts. The first verified the functionality of point cloud processing and the second one the functionality of collision avoidance. Modules parameters were:

- OctoMap: resolution 20 cm; hit probability 0.58, miss probability 0.4; max. clamping threshold 0.97; min. clamping threshold 0.12; OctoMap raycasting range clamping 60 m.

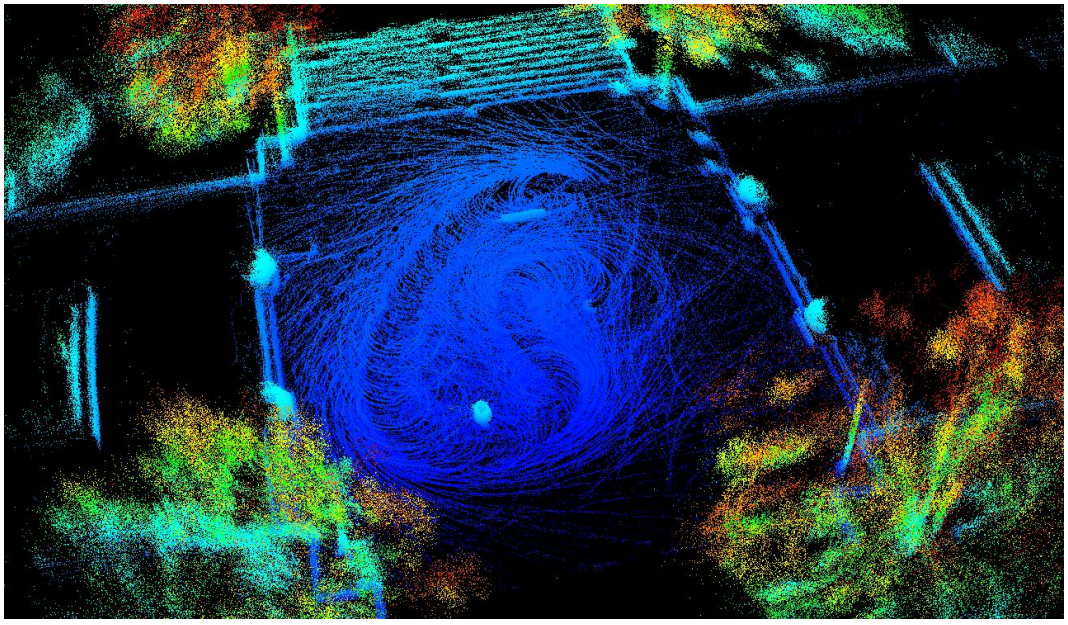


Figure 8.12. Manually corrected point cloud. This point cloud also serves as an input data set for the first experiment of path planning and collision avoidance.

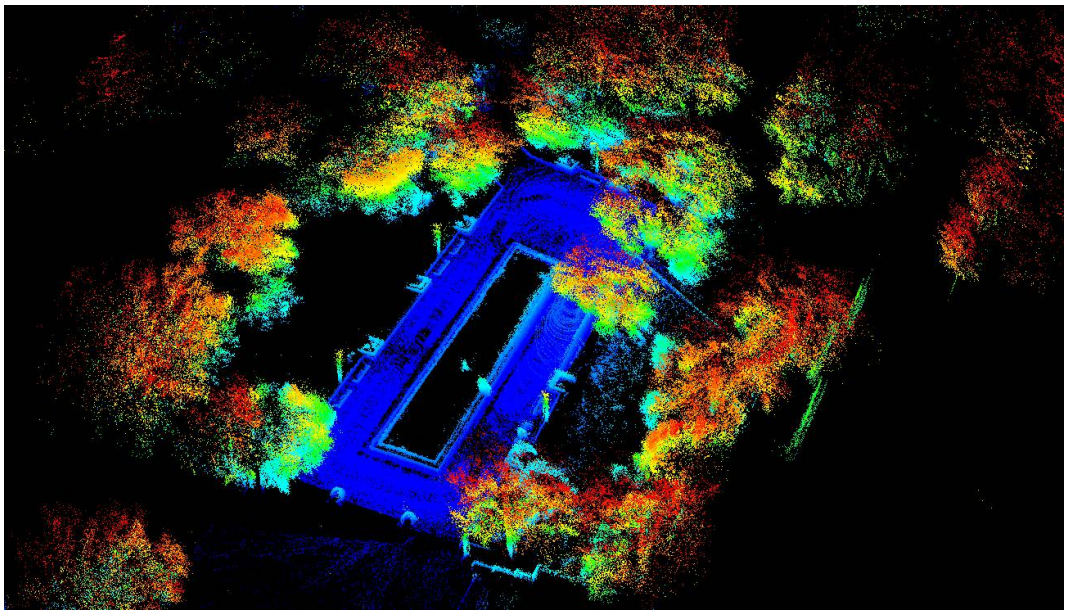


Figure 8.13. Manually corrected point cloud. This point cloud also serves as an input data set for the second experiment of path planning and collision avoidance.

- Downsampling: voxel size 5 cm.
- Ground filter: max. points horizontal distance 0.1975; ground filter max. d 0.3 (Sec. 6.1.3); max. ground plane angle inclination 0.3 rad.

Firstly the OctoMap representation of the environment prepared. The point cloud 8.12 was divided into 5 parts according to the time when they were taken, then these parts were sequentially processed and inserted into the OctoMap. The processing part is presented in Fig. 8.14, the final OctoMap is in Fig. 8.15.

Collision avoidance test is performed in TAF (Chap. 7.1). Robot size is set to $1 \times 1 \times 2$ m to be sure that the collision is always detected and to add some safe

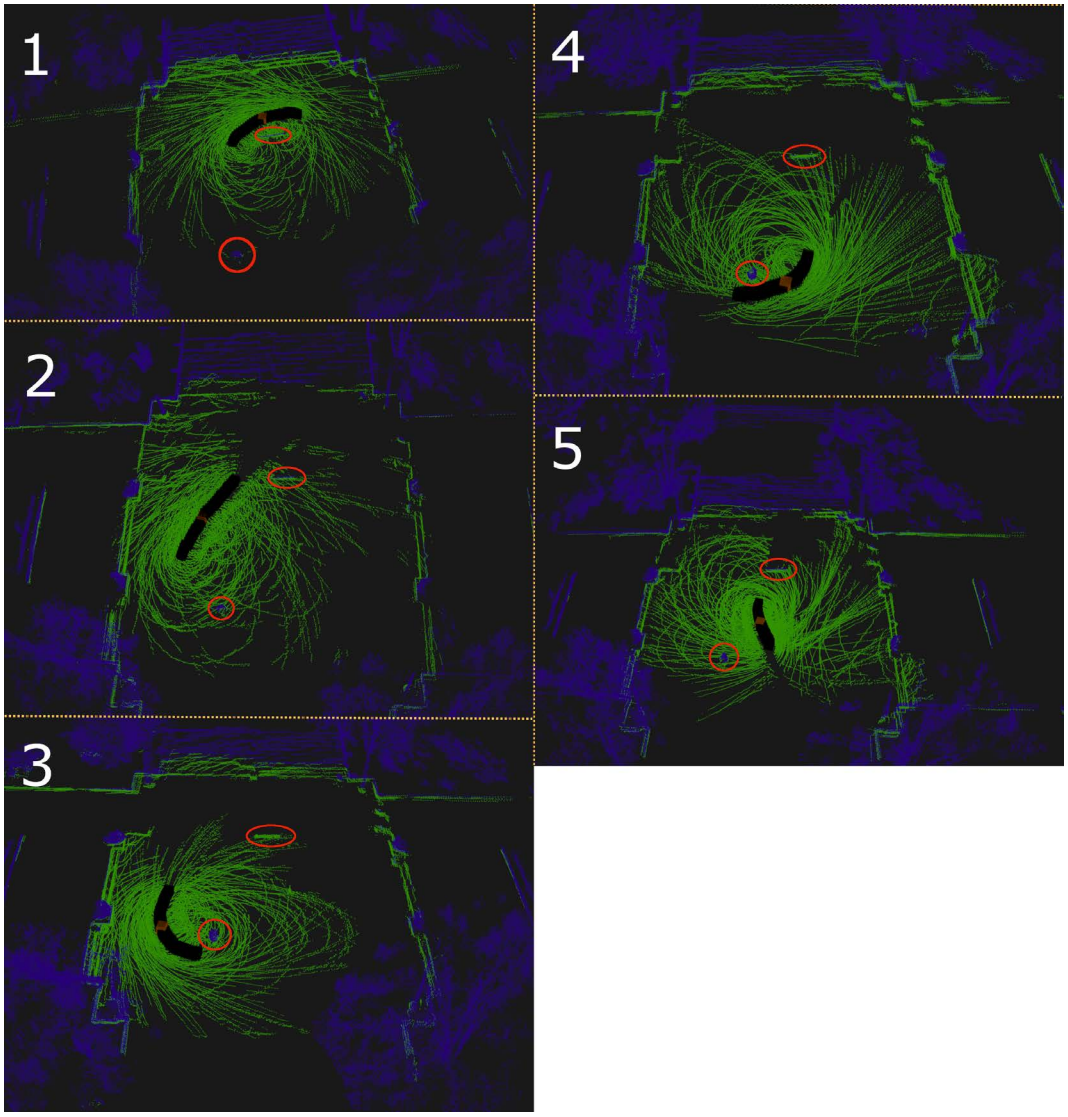


Figure 8.14. Point cloud preparation. Each fifth is processed independently and then inserted into OctoMap. Black line represents rover trajectory. Blue points represent points that are passed along, green points are filtered. Notice the obstacles marked in red. Each part is processed independently (e.g. the filtered plane has slightly different coefficients) that is why the long brick is filtered in cases 3 and 4 but is kept in steps 1, 2 and 5. The trash bin is always kept. The small brick (can be seen in Fig. 8.8) is too low and therefore filtered as ground. Original size of the PC is 2 624 000 points, 1 015 142 points after downsampling and 715 503 after ground segmentation.

margins around the obstacles. The path is set manualy through waypoints. How the planner solved the situation is captured in Fig. 8.16.

During this test we focused on three obstacles, they are marked in red ellipses in Fig. 8.8. The trash bin and the long brick on the left were successfully recognized, while the small brick "behind" the trash bin was filtered as a ground. We did not manage to find the parameters of ground filtration that would keep the brick as an obstacle. If the parameter of point vertical distance was lowered, it would not filter the whole ground, because the points in the point cloud belonging to the ground are vertically spread in about 20 cm.

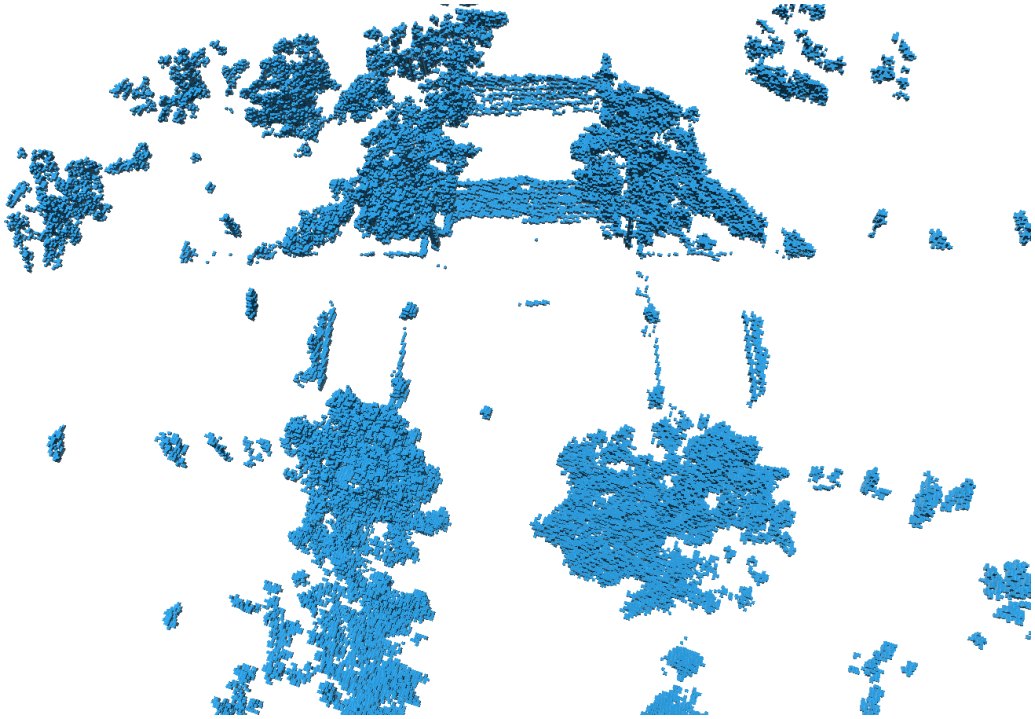


Figure 8.15. OctoMap for path planning and collision avoidance test #1. OcTree resolution is 20 cm.

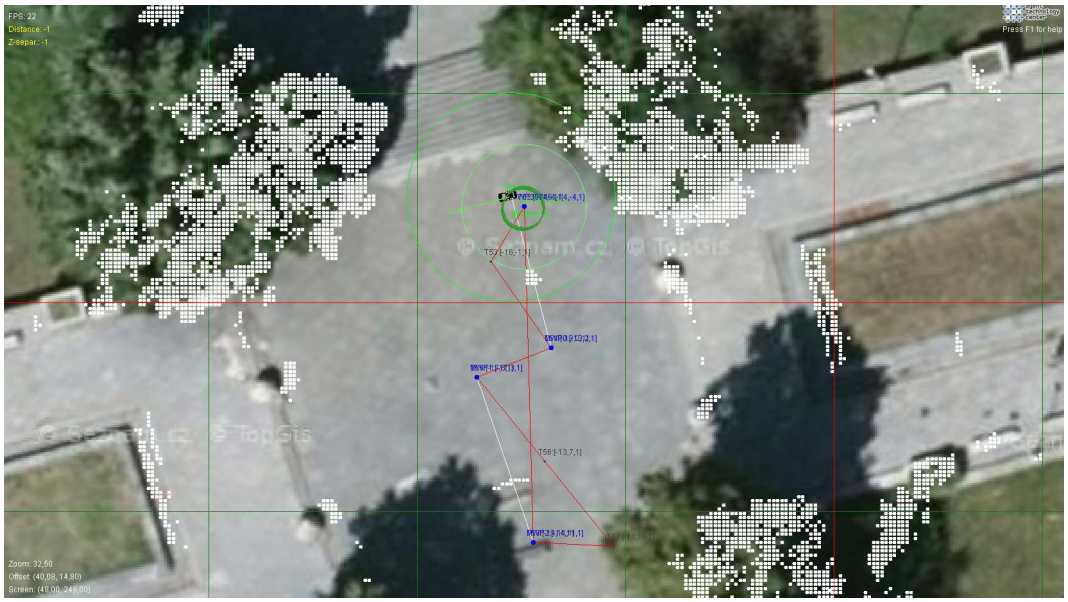


Figure 8.16. Collision avoidance test #1. Blue dots represent OctoMap obstacles (Fig. 8.15). The path was set by the blue waypoints (connected by white lines). How the collision was solved and obstacles bypassed show the grey waypoints connedcted with red lines. This solution was found in 0.5 seconds which is the RRT* planner time limit.

8.3.2 On the fly processing with collision avoidance

In this test the point cloud processing was run in parallel with path planning and collision avoidance simulating the map exploration. Point cloud points were mapped on the trajectory where the UGV sensed them in a real environment. When the UGV

in the simulation got close to the trajectory, corresponding parts of the point cloud were processed and sent into OctoMap. Right after the OctoMap received them, the planner could avoid obstacles in these parts. The OctoMap model after discovering all the points was saved and is shown in Fig. 8.17. The path planning with collision avoidance part of the test is captured in Fig. 8.18 and Fig. 8.19. Modules parameters were:

- OctoMap: resolution 20 cm; hit probability 0.63, miss probability 0.4; max. clamping threshold 0.97; min. clamping threshold 0.12; OctoMap raycasting range clamping 60 m.
- Downsampling: voxel size 5 cm.
- Ground filter: max. points horizontal distance 0.3; ground filter max. d 0.3 (Sec. 6.1.3); max. ground plane angle inclination 0.1 rad.

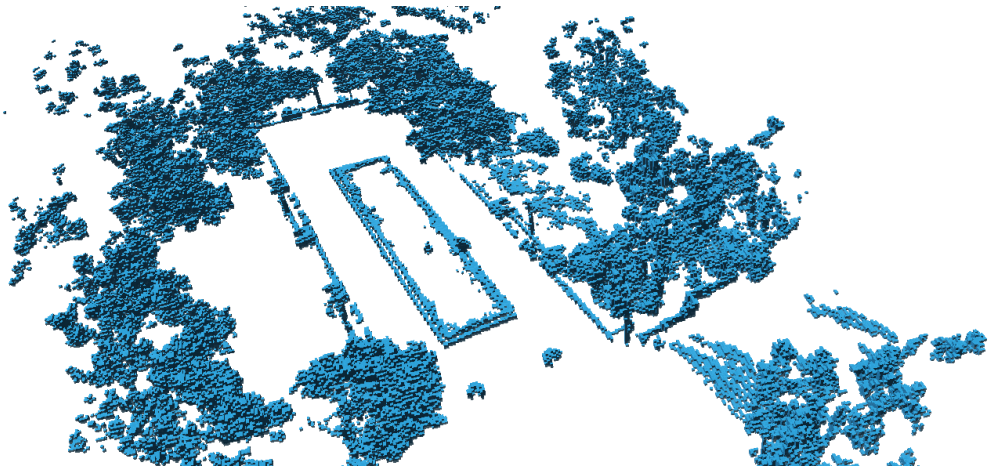


Figure 8.17. OctoMap from the second test. OcTree resolution 20 cm. For illustrative purposes.

The rover was given two waypoints and managed to get on them successfully while avoiding progressively added obstacles. The points corresponding to the rover driven trajectory were processed and added to OctoMap every 0.5 second (the same period as RRT* search max. time). The CPU of the machine performing the simulation (Intel i5 2430m 2.4 GHz) was on 75% of maximum load.

A video of this experiment is on the attached CD. Because the point insertion into OctoMap and collision detection has higher priority, there is about 0.5 seconds delay in displaying the obstacles in TAF.

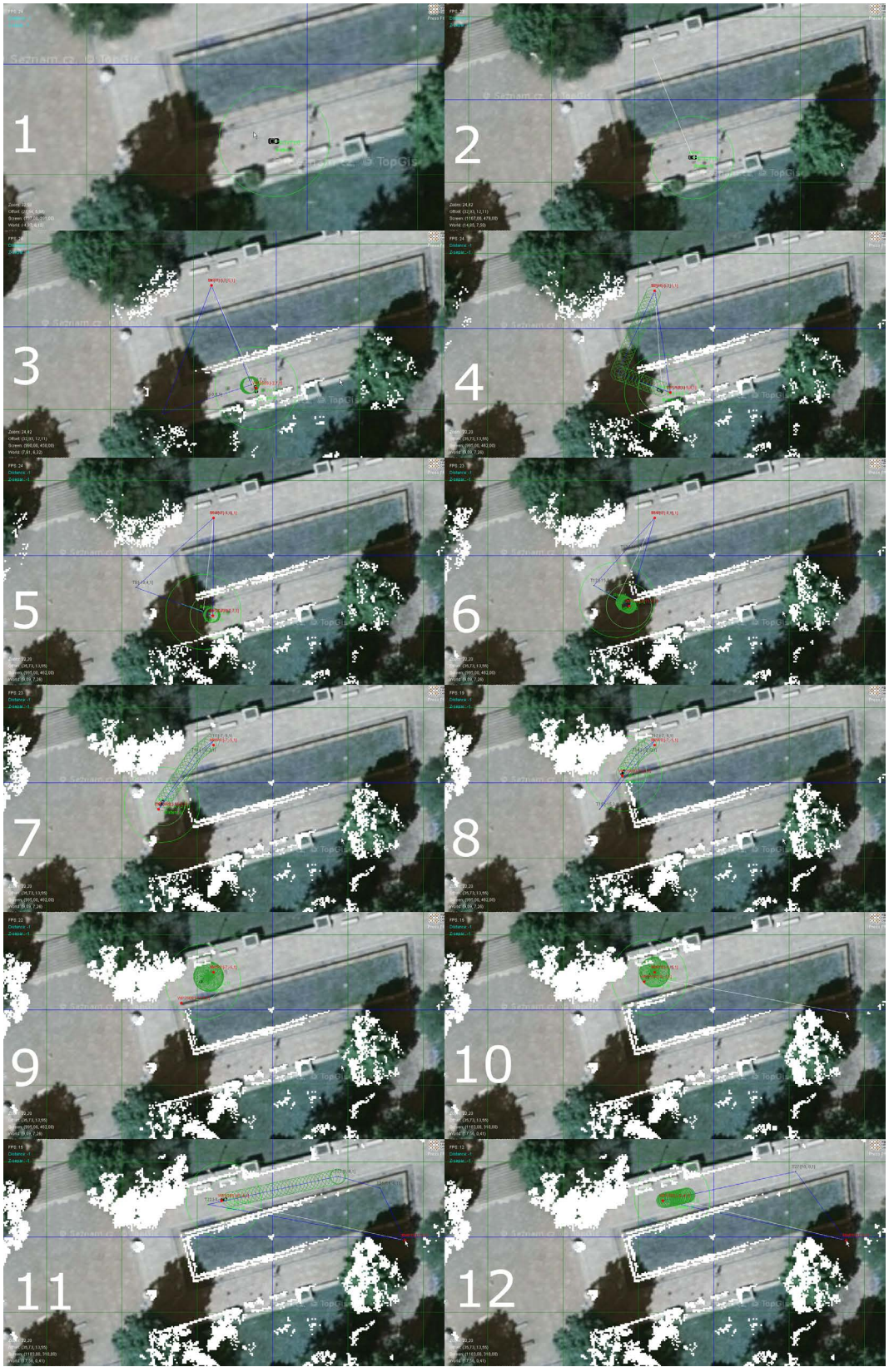


Figure 8.18. Collision avoidance test #2 part 1. Picture number 2 shows the starting and goal position of first target. Pictures 2-9 show how the UGV was moving and the map was incrementally built. In picture 10 another target waypoint was selected and the pictures 11-24 show the progress again. Continuation in Fig. 8.19.

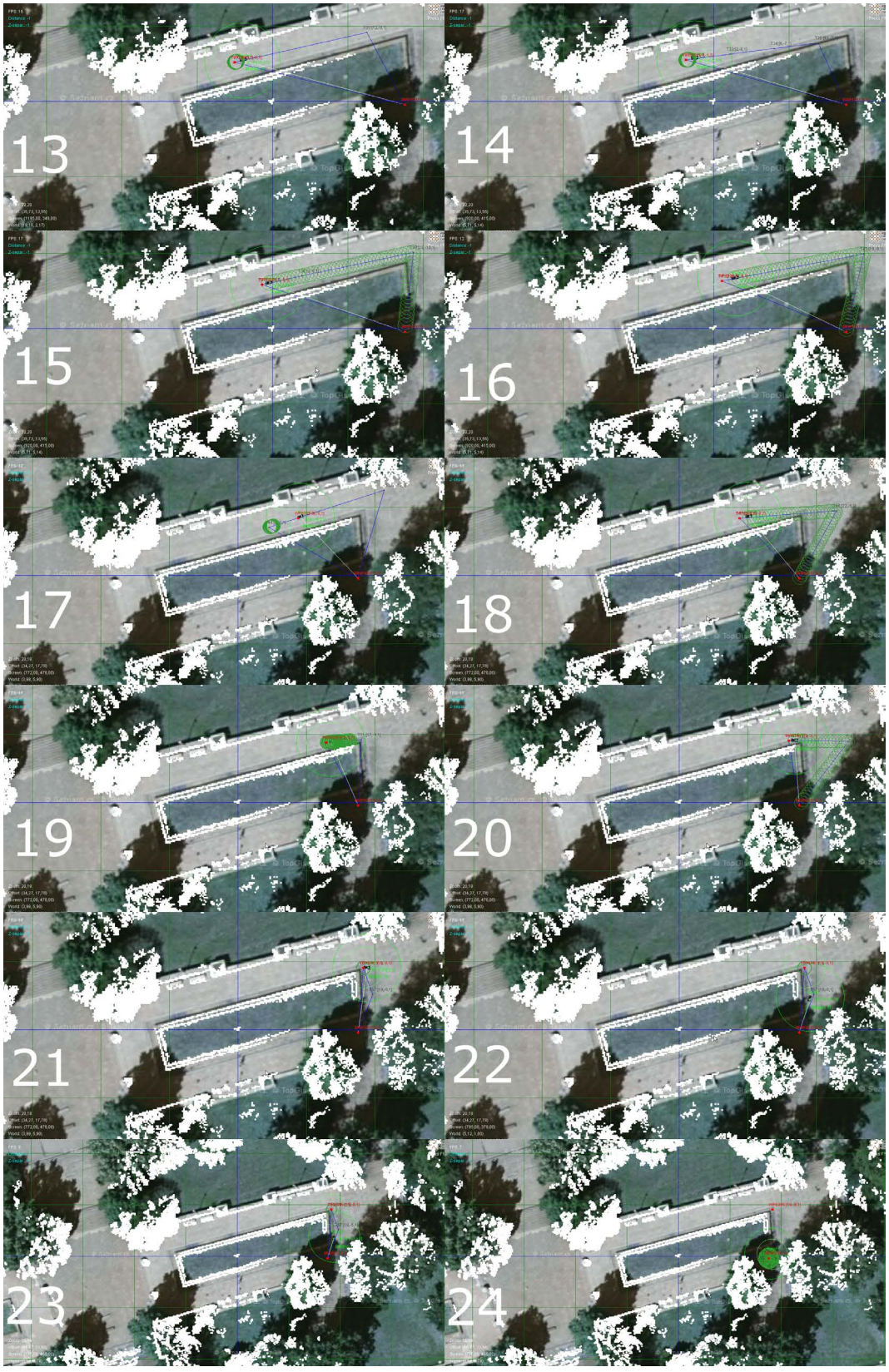


Figure 8.19. Collision avoidance test #2 part 2.

Chapter 9
Conclusion

In this work we dealt with problem of navigation of unmanned vehicles. Firstly we reviewed existing approaches to point cloud creation, path planning and collision avoidance. Then we started working with sensors. We implemented their communication protocols and integrated them under both Windows and Linux operating systems. We designed Java framework for point cloud creation and implemented it. To carry out field experiments a rover platform was built. We acquired an RC model, used its chassis, servos and controller and built extra two-floor platform onto which we placed the sensors.

When we started sensing the environment, the framework could only create the point clouds in real time. That turned out to be very inefficient because point cloud was very difficult to analyse. To overcome this issue the possibility for sensors raw data acquisition was implemented. After that, during offline point cloud creation, we identified and fixed several both software and hardware errors. The software ones were usually related to issues that were omitted in invited solutions, e.g. alignment between coordinate systems of different sensors or wrong time frames adjustments. The hardware errors included wrong sensors placement or use of inadequate devices (radios).

Once we were able to produce correct point clouds, we started integrating the software required for collision avoidance and path planning. We modified specific parts of TAF system, whose main focus were aerial vehicles, to match our requirements for rover. We implemented OctoMap. We tested and incorporated Flexible Collision Library into collision detection module of TAF. We found out that a point cloud has to be filtered, downsampled and the ground has to be segmented before inserting into OctoMap. Otherwise the collision detection would not work in the desired manner.

Last thing was the modification of planning algorithms in TAF for rover usage. Firstly we implemented a simple planner, but since it turned out not to be working properly with the higher amount of present obstacles, with few modifications we integrated already available RRT* planner.

The result is a robust software framework that could be used in applications of unmanned ground vehicles equipped with LiDAR, IMU and RTK GPS, for path planning and collision avoidance.

9.1 Future work

To be able to put the presented framework onboard the usage or the environment of operation should be discussed. The algorithms and presented ways of collision avoidance will work but proper parameters of models processing must be set accordingly. Because they depend on the environment, one possibility could be measuring some sample areas, surfaces and objects and create a table of values and measured objects.

Also hardware computational requirements should be measured. The tests showed high CPU load. That would even increase while running in real time, because of

point cloud creation, which itself usually consumes about 160% out of 400% available performance on Toradex (quad-core). But while that could be solved by a proper parallelization, providing there is still enough computational power available, problems with RAM are expected. While PC creation is running, there is about 0% - 20% out of 1 GB RAM available. That is not enough and RAM upgrade should be considered.

However, the most important task is to find an automated fix for elimination of the misalignment between IMU and RTK GPS coordinate systems. A solution to fix this issue offline was found and used during the experiments, but without proper point cloud available in real time, we can not process it, create OctoMap, therefore perform collision checks and navigate in the environment.

References

[1] Sebastian Thrun, Michael Beetz, Marren Bennewitz, Wolfram Burgard, Armin B. Creemers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Chuck Rosenberg, Nicholas Roy, Jamieson Schulte, and Ruth Schulz. Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *International Journal of Robotics Research*. 2000, 19 (11), 972–999.

[2] Richard Thrapp, Christian Westbrook, and Devika Subramanian. Robust localization algorithms for an autonomous campus tour guide. *Proc. of IEEE Int'l Conf. on Robotics and Automation*. 2065–2071.

[3] Vladimir Maximov, and Oleg Tabarovsky. Survey of Accuracy Improvement Approaches for Tightly Coupled ToA/IMU Personal Indoor Navigation System. *Proceedings of International Conference on Indoor Positioning and Indoor Navigation*. 2013,

[4] Y. Chen, and H. Kobayashi. Signal strength based indoor geolocation. *Proceedings of the IEEE International Conference on Communications*. 2002, 1 436–439.

[5] Alberto Fernández, Omar Fres, Ignacio Alonso, and Huosheng Hu. Visual localisation of mobile devices in an indoor environment under network delay conditions. *International Journal of Distributed and Parallel Systems*. 2011, 2 (2),

[6] Hugh Durrant-Whyte, and Tim Bailey. Simultaneous localization and mapping: part I.. *IEEE Robotics & Automation Magazine*. 2006, 13 (2), 99-110.

[7] *Real Time Kinematic*. 2016.
`https://en.wikipedia.org/wiki/Real_Time_Kinematic`. Accessed: 2016-03-07.

[8] *What is LiDAR*. National Oceanic and Atmospheric Administration.
`http://oceanservice.noaa.gov/facts/lidar.html`. Accessed: 2016-03-29.

[9] *3DM-GX4-45 Inertial Navigation System User Manual*. 2014.

[10] *RTK Fundamentals*. GMV. 2011.
`http://www.navipedia.net/index.php/RTK_Fundamentals`. Accessed: 2016-03-18.

[11] *Miniature 2.4 Ghz OFDM Broandband Ethernet Bridge / Serial Gateway*. 2016.
`http://www.microhardcorp.com/nVIP2400.php`.

[12] *Ardupilot*. 2015.
`https://en.wikipedia.org/wiki/Ardupilot`. Accessed: 2015-12-28.

[13] *Iris V1.1 Datasheet*.
`www.toradex.com`.

[14] Yuan Gao. *Difference between military and GPS data*.
`https://www.quora.com/Is-there-any-difference-between-military-GPS-data-and-civilians-in-terms-of-accuracy`. Accessed: 2015-12-28.

[15] *GPS Accuracy*. Official U.S. Government information about GPS.
`http://www.gps.gov/systems/gps/performance/accuracy/`. Accessed: 2016-04-05.

[16] Edwin Olson. A passive solution to the sensor synchronization problem.. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems*. 2010, 1059-1064. DOI 10.1109/IROS.2010.5650579.

[17] WhiteTimberwolf. 2010.
`https://en.wikipedia.org/wiki/File:Octree2.svg`.

[18] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*. 2013, DOI 10.1007/s10514-012-9321-0. Software available at
`http://octomap.github.com`.

[19] Radu Bogdan Rusu, and Steve Cousins. 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation (ICRA)*. 2011,

[20] Martin Fischler, and Robert Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*. 1981, 24 (6), 381-395.

[21] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries . *2012 IEEE International Conference on Robotics and Automation (ICRA)*. 2012, 3859 - 3866. DOI 10.1109/ICRA.2012.6225337.

[22] Michael Wooldridge, and Nicholas Jennings. Intelligent agents: theory and practice. *The Knowledge Engineer Review*. 1995, 10 (2), 115-152.

[23] Sertac Karaman, and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*. 2011, 30 (7), 846-894. DOI 10.1177/0278364911406761.

[24] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Technical Report(Computer science department)*. 1998, 98 (11),

[25] Tomáš Trafina. *Bachelor thesis: Construction of 3D Point Clouds Using LiDAR Technology*. 2016.

Appendix A
The attached CD-ROM

The picture below shows the tree structure with included files :

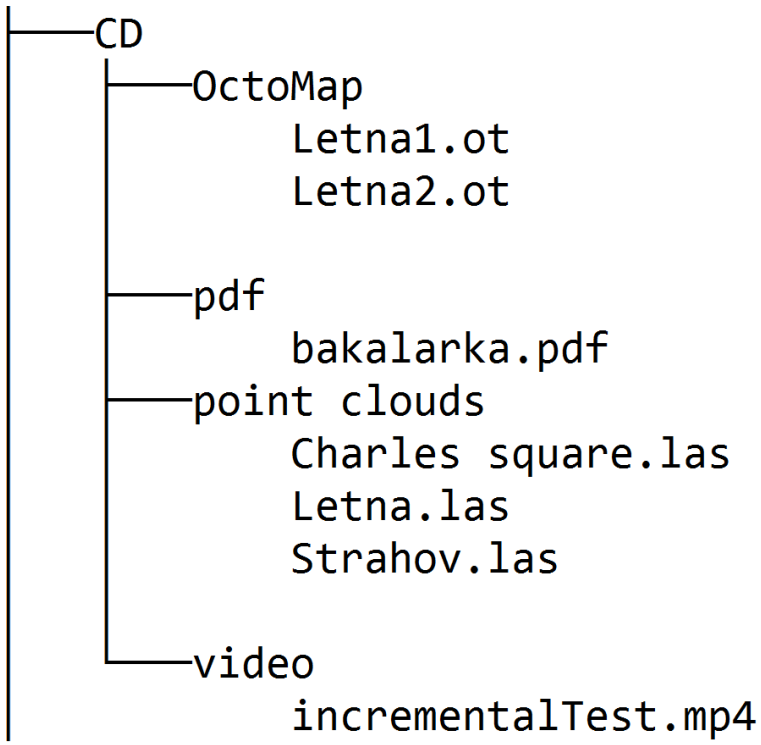


Figure A.1. CD content

- The pdf version of this thesis can be found in the *pdf* directory.
- The OctoMap files regarding test in Sec. 8.3 are located in *OctoMap* directeroy. They can be opened in Octovis[1].
- Point cloud las files in *point clouds* directory were used for tests described in 8.1 and 8.3.2. They can be open with one of the many available las viewers such as Quick Terrain Reader[2], SAGA Gis[3] or even online variants.
- The video in the *video* folder is captured the experiment described in 8.3.2.

[1] http://wiki.ros.org/octovis
[2] http://appliedimagery.com/download/
[3] http://www.saga-gis.org/en/index.html

Appendix B
Abbreviations

UAV	**Unmanned aerial vehicle**, commonly known as a drone, as an unmanned aircraft systems, is an aircraft without a human pilot aboard.
UGV	**Unmanned ground vehicle**. The same as UAV except it's terrestrial and operates on the ground.
LiDAR	LiDAR stands for an acronym of **Light Detection and Ranging**. It's a surveying technology that measures distance by illuminating a target with a laser beam.
FEE	**Faculty of electrical engineering.**
GPS	**Global position system** is a space-based navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth, where these is an unobstructed line of sight to four or more GPS satellites.
GNSS	**Global Navigation Satellite System** is a satellite system that is used to determine the geographic location of a user's receiver anywhere in the world.
IMU	An **inertial measurement unit** is an electronic device that measures and reports a body's specific force, angular rate and the magnetic field surrounding the body.
ToA	**Time of Arrival** is the travel time of a radio signal from a single transmitter to a remote single receiver.
SLAM	**Simultaneous localization and mapping** is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.
GLONASS	**GLObal NAvigation Satellite System** is a space-based satellite navigation system operating in the radionavigation-satellite service and used by the Russian Aerospace Defence Forces providing an alternative to GPS.
UDP	**User Datagram Protocol** is one of the core members of the Internet protocol suite.
RTK	**Real Time Kinematic** satellite navigation is a technique used to enhance the precision of position data derived from satellite-based position systems.
UART	**Universal Asynchronous Receiver/Transmitter** is usually an individual integrated circuit used for serial communications over a computer or peripheral device serial port.
LVTTL	**Low-Voltage Transistor-Transistor Logic** is a class of digital circuits built from bipolar junction transistors and resistors.
GCS	**Ground Control Station** is a control center that provides the facilities for human control of unmanned vehicles on the ground or in the air.
APM	**ArduPilot Mega** is an open source UAV platform, able to control autonomous multicopters, fixed-wing aircraft, traditional helicopters and ground rovers.

RC **Radio control** is used for control of model vehicles from a hand-held radio transmitter.

LiPO **Lithium-ion polymer battery** is a rechargeable battery of lithium-ion technology in a pouch format.

NiMH **Nickel-Metal Hydride** battery is a type of rechargeable battery.

ESC **Electronic speed controller** is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake.

SNR **Signal-to-noise ratio** is a measure used in science and engineering that compares the level of a desired signal to the level of background noise.

RANSAC **RANdom SAmple Consensus** is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers.

PCL **Point Cloud Library** is a standalone, large scale, open project for 2D/3D image and point cloud processing.

FCL **Flexible Collision Library** that integrates several techniques for fast and accurate collision detection and proximity calculations.

TAF **Tactical AgentFly** is a agent-based software system for command & control and simulation of autonomous vehicles with main focus on an area surveillance or mobile ground target tracking.

RRT **Rapidly-exploring Random Tree** is an efficient algorithm for searching and path planning in nonconvex environments.

CA **Collision Avoidance.**