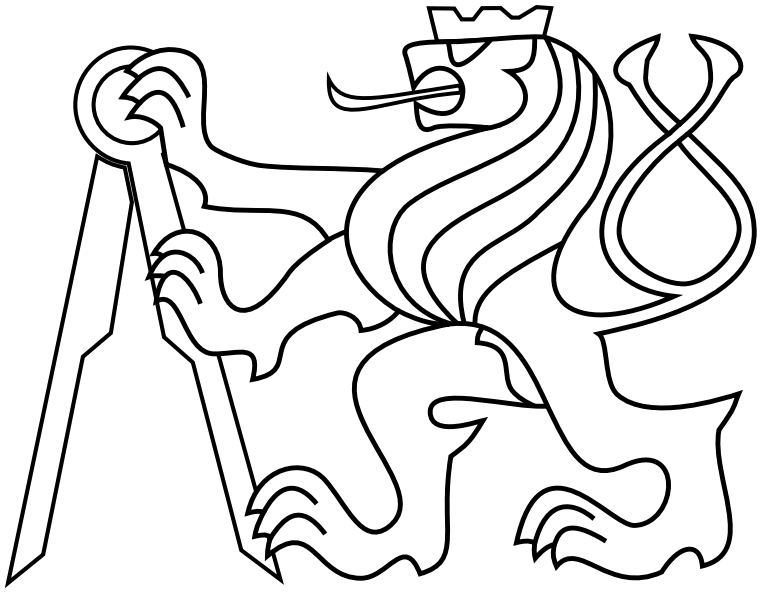
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Aleš Novotný

Reactive collision-free motion planning of a helicopter using data from onboard stereo camera

Department of Cybernetics

Thesis supervisor: **Ing. Vojtěch Spurný**

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne. . . . . Podpis . . . . . .

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Novotný Aleš** Personal ID number: **457209**

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

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

Study program: **Cybernetics and Robotics**

Branch of study: **Systems and Control**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Reactive collision-free motion planning of a helicopter using data from onboard stereo camera

Bachelor's thesis title in Czech:

Reaktivní plánování bezkolizního letu helikoptéry s využitím dat z onboard stereo kamery

Guidelines:

A motion planning algorithm will be designed that enables to avoid currently detected obstacles (trees) in the task of autonomous flying in a forest-like environment. The following points will be solved:
- To design and implement methods for reconstruction of objects (trees) in 3D Point Cloud obtained from the stereo camera during flight.
- To create a local map of detected objects.
- To design and implement a reactive collision-free technique for control of a Micro Aerial Vehicle (MAV) (e.g., VFH [1]).
- To integrate the methods into the ROS system being designed at MRS group, CTU in Prague [2,3].
- To implement a model of stereo camera sensor into the Gazebo robotic simulator.
- To verify system functionalities in the simulator.
- To conduct a real-world experiment with MAV (MAV localization in GPS-denied environment is not a part of this thesis; a combination of GPS data and the designed obstacle detection technique will be used in the experiment).

Bibliography / sources:

[1] J. Borenstein and Y. Koren, The vector field histogram-fast obstacle avoidance for mobile robots, in IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 278-288, 1991.
[2] T. Baca, P. Stepan and M. Saska. Autonomous Landing On A Moving Car With Unmanned Aerial Vehicle. In The European Conference on Mobile Robotics (ECMR), 2017.
[3] G. Loianno, V. Spurny, J. Thomas, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar. Localization, Grasping, and Transportation of Magnetic Objects by a team of MAVs in Challenging Desert like Environments. In IEEE ICRA and RAL, 2018.

Name and workplace of bachelor's thesis supervisor:

Ing. Vojtěch Spurný, Multi-robot Systems FEL

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

Date of bachelor's thesis assignment: **30.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

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

Acknowledgements

First of all, I would like to thank Ing. Vojtěch Spurný for his supervision in this project. I would also like to thank other people from Multi-robot Systems group for their advice and assistance with technical problems and eventually my family, relatives, and friends for support during studying.

Abstract

The aim of this work is to design obstacle detection system for Unmanned Aerial Vehicle (UAV) equipped with a depth camera. Especially, we propose methods for image filtering, real-time automatic detection of obstacles in a forest-like environment in which the UAV fly, and obstacle avoidance algorithm based on Vector Field Histogram (VFH) which is used for reactive collision-free motion planning. The proposed system is modular and it can be used for processing of data from different types of cameras with the same function principle. The functionality of the system has been tested in several simulations and real-world experiments.

keywords:
[unmanned aerial vehicle, depth camera, vector field histogram, reactive motion planning, obstacle avoidance]

Abstrakt

Cílem této práce je navrhnout systém, který detekuje překážky pro bezpilotní letadla vybavené hloubkovou kamerou. Především navrhujeme metody pro filtrování vstupního obrazu, automatická detekce překážek v reálném čase a nezávisle na prostředí, ve kterém se bezpilotní letadlo pohybuje a algoritmus pro vyhýbání se překážkám založený na metodě vector field histogram, která se používá pro reaktivní bezkolizní plánování letu. Navržený systém je modulární a může být použit pro zpracování dat z různých typů kamer založených na stejném principu funkce. Funkčnost systému byla testována v několika simulacích a experimentech v reálném světě.

klíčová slova:
[bezpilotní letadlo, hloubková kamera, vector field histogram, reaktivní plánování letu, vyhýbání se překážkám]

Contents

List of Figures

1 Introduction

In the recent years, there has been a significant progress in the aerial vehicles technologies. This led to the creation of aerial vehicles without a pilot aboard called Unmanned Aerial Vehicles (UAVs) or Micro Aerial Vehicles (MAVs). UAVs are currently very popular due to their relatively low cost and very diverse range of activities that suit them. They can be used for example for shooting videos from high heights, to operate in environments that are dangerous for humans [1], transport medication [2], [3], [4], area mapping [5], [6], [7] etc.

The UAV in order to achieve its goals can be controlled either by a person using some remote control or fly autonomously if it is capable. One of the main advantages of a flight which is controlled by a human is that the pilot who controls the UAV can easily react to unpredictable situations. On the other hand, the piloted flight is expensive, the information comes delayed, and in some situations, the human factor can cause a collision. These UAVs often carries a large number of sensors which can quickly and accurately detect an obstacle at the direction of the flight at any moment. Thanks to this information, it is possible to react quickly and replan the trajectory before colliding.

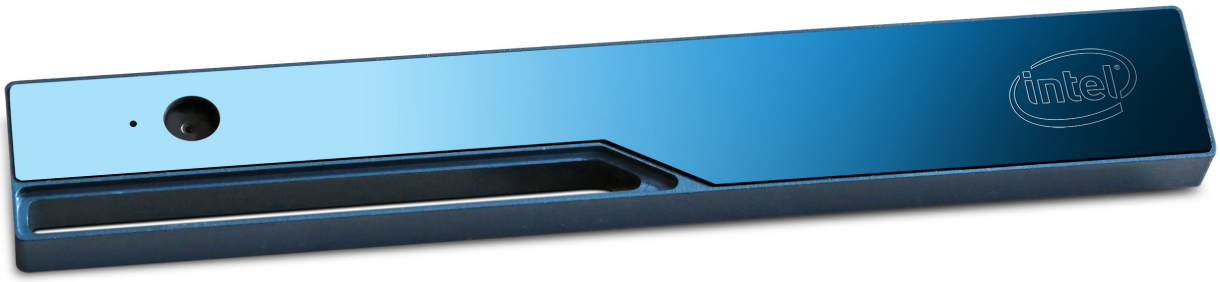


Figure 1: Intel RealSense R200 [2].

Majority of the sensors have a strictly limited area of usage to specific environments and applications. Most of the sensors for detecting obstacles are based on technology using the emission of electromagnetic radiation at a certain frequency and evaluating detected obstacles from radiation reception.

One of the most commonly used obstacle detection technology is Radio detection and ranging (RADAR), which uses electromagnetic radiation at frequencies ranging from MHz to tens of GHz. It is mainly used to identify objects over long distances. RADAR is used in larger UAVs [8] that weigh hundreds of kilograms, but there are also lower power versions suitable for smaller UAVs weighing kilograms. However, this technology, even in smaller versions puts a lot of emphasis on dimensions (see Figure 2a).

[2]Source: http://reconstructme.net/qa_faqs/intel-realsense-r200-review/

Another technology is Light Detection and Ranging (LIDAR), the method of remote measurement of distances using a wavelength of roughly 1000 nm [9]. There are many types of these sensors. Some of them are directional and designed to measure depth in a predefined direction, others are rotating and scan depth in all directions (see Figure 2b). The dimension of these sensors is appropriate for using on small autonomous vehicles for reactive collision-free technique [10]. However, when these sensors are in small dimensions, the measurement is usually done only in 2D that is not providing enough information for recognizing shapes of obstacles and their reconstructing.

In this thesis, we deal with a depth camera which is small, can be mounted on UAV easily (see Figure 2c) and provide a large amount of information about shapes of detected obstacles. In our case, we work with Intel RealSense R200 (see Figure 1) that is based on stereo vision technology assisted by the infrared laser projector and two infrared imagining sensors. This camera outputs depth video stream, which is similar to color video stream but every pixel has the value representing the distance from the camera instead of color information. The distance is calculated using shift between images from both imagining sensors [11].

(a) UAV with RADAR [3]. (b) UAV with LIDAR [4]. (c) UAV with depth camera[5].

Figure 2: Examples of using sensors for detection of obstacles mounted on UAVs.

This thesis further deals with the processing of data from this camera which is mounted on the UAV, and with the use of this data for reactive collision-free technique, which allows the UAV to reach the target position without collision. Collision-free flight planning is done by motion planning algorithm called Vector Field Histogram (VFH) [12]. The proposed collision avoidance methods do not use any information about the location of obstacles in the space and respond to obstacle detection by rescheduling the direction of flight. The reconstruction of obstacles is done by approximation by cylinders and polygons. All of the presented methods are embedded into Robotic Operating System (ROS) [13].

The thesis is structured as follows. Firstly in chapter 2 is described processing input point cloud, starting with transformations into a convenient frame, simplifying point cloud and continuing with removing outliers. Furthermore, in this section, a processed point

[3]Source: https://aerotenna.com/aerotenna-releases-360-sense-avoid-radar-advances-drones-closer-autonomous-flight/

[4]Source: https://www.spar3d.com/news/lidar/vol14no5-discovering-the-potential-of-affordable-lidar/

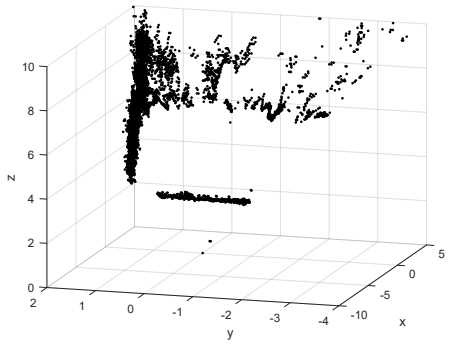
[5]Source: https://software.intel.com/en-us/aero

cloud is used for object reconstruction. In following chapter 3 is described the creation of a local map and in chapter 4 are data from this map used for an implementation of the collision-free technique. In chapter 5 is described the implementation of a simulated model of the camera and in chapter 6 is verified its functionality. Finally in the last chapter 7 are presented results from real-world experiments.

2 Processing of point cloud

Figure 3: Post-processing pipeline.

The R200 depth camera has resolution 480x360 and can be running at frame rate 60 Frames Per Second (FPS). The reader can come to a conclusion that every sample has information about 172,800 points and every second come information about 10 million points which need to be processed. The input point cloud also contains a large amount of noise, unnecessary information such as the points representing the ground or excessive density of points representing the object (see Figure 4a). For the fast running of the program and reducing a probability of obstacle false detection, the incoming data need to be processed before we start reconstructing and use data for reactive collision-free technique. The sequence of procedures which we apply to the point cloud is shown in figure 3. In this section, we will present functionality of the proposed procedures on data from the scene shown in figure 4a.



(a) Example point cloud detected by depth camera.

(b) Example view detected by normal camera.

Figure 4: Comparison of view detected by different cameras.

2.1 Transformation to UAV frame

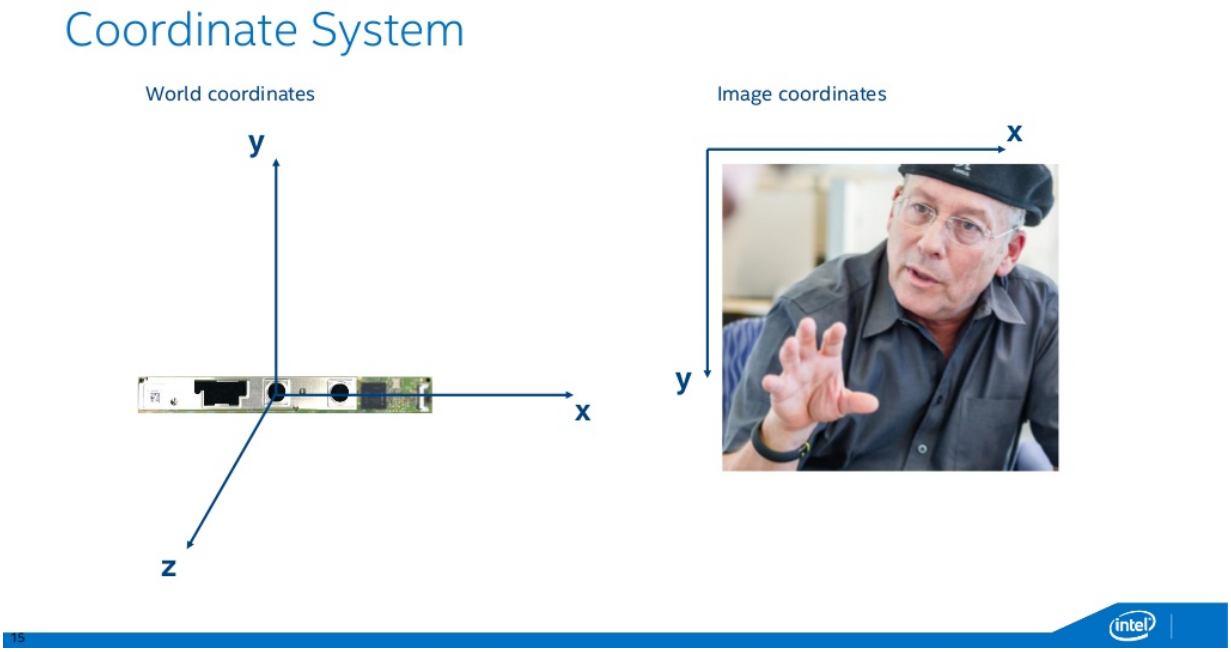


Figure 5: Rotation of the input data [6].

The input data that comes from the camera are oriented as it is shown in figure 5. It is convenient to transform point cloud into UAV frame first due to this step the obtained data are more understandable and other parts of processing are easier. For this transform (see equation 1), affine transformation is used.

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ d \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} \tag{1}$$

where R is rotation matrix, T is translation matrix. To determine the direction in which the camera is focused on and where the obstacles are located, it is necessary to know the UAV rotation, this information is obtained by Inertial Measurement Unit (IMU). It is important to synchronize the data rotation with camera data because processing of data from the camera takes some time. Otherwise, the UAV could transform point cloud using the current orientation instead of the orientation when the data were taken, this will lead to inappropriate placement of the objects respectively to the UAV. By applying maneuvers to avoid these objects, the UAV can in the worst case collide with a real obstacle. For

[6]Source: https://www.slideshare.net/bemyapp/first-steps-with-intel-realsense-sdk-by-xavier-hallade

synchronization of IMU and data from the camera we use a buffer, that store information about orientation and rotation, therefore we are able to look back and assign the right orientation to data from the camera (see results of transformation in Figure 6).

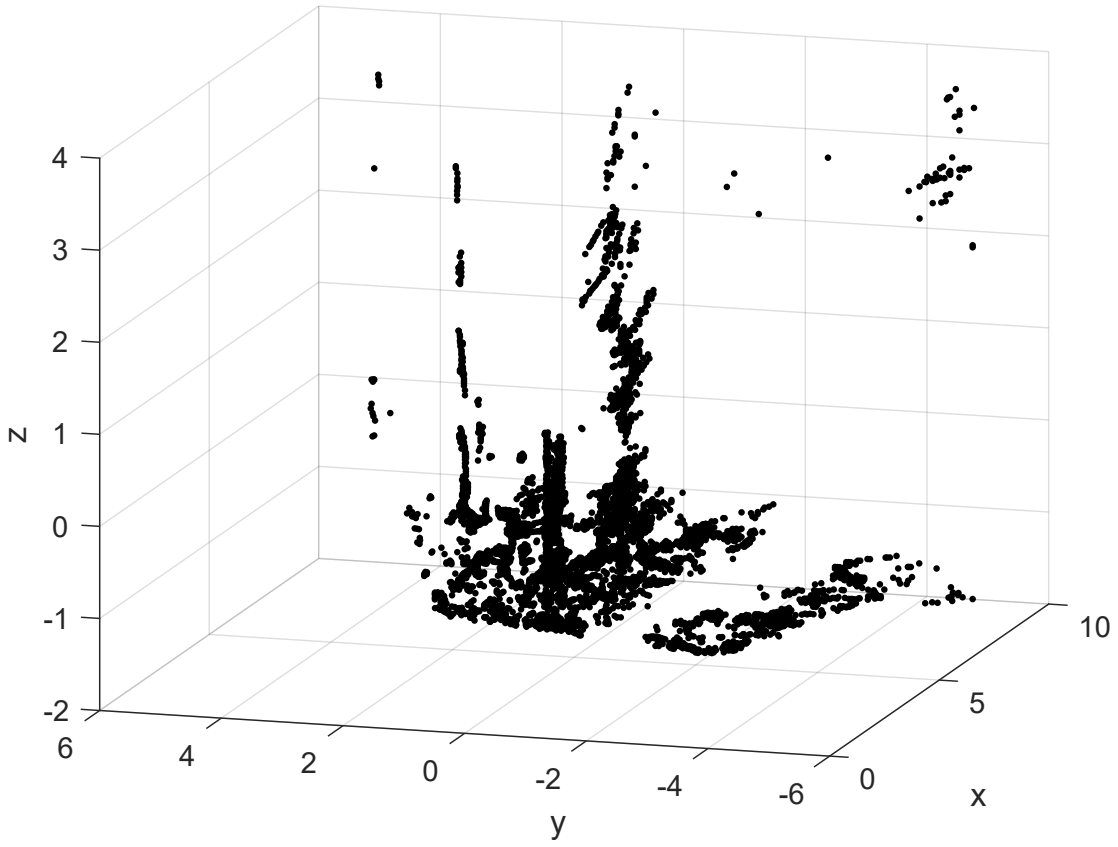


Figure 6: Input point cloud from depth camera transformed into UAV frame.

2.2 Introduction into filtering

For processing the input cloud, the Point Cloud Library (PCL) is used. This is an open source library that is focused on dealing with point clouds. Due to the size of the input data, it is important to make modifications that are not computationally demanding and removes a lot of information that is not useful for detection of obstacles. One of these modifications is for removing points representing the ground. For the UAV flying in low altitudes. The ground is represented by a lot of points which in our case do not give any useful information because this information may lead to confusing the algorithm which detects objects. Furthermore, the objects detected by the camera are presented by very dense point cloud. For increasing speed of the program is convenient to find an algorithm

that reduces the density of the point cloud with the cost of minimal loss of information. Finally, the input data usually contains a lot of outliers, which may influence the object detection, thus it is important to find the way how to remove outliers or eliminate the impact of outliers on the obstacle detection.

2.3 Removing ground

For this task is used a filter called PassThrough filter, which removes from the input data points which do not lie in the area defined by criterion. We have experimentally found out that when the UAV flies with low velocities (up to 1 ms^{-1}) the tilt of the UAV is small. Therefore we can make an assumption that points that are below 10 cm on the z-axis represent ground and can be removed (see Figure 7). This range needs to be adjusted due to concrete application and placement of the camera on the UAV.

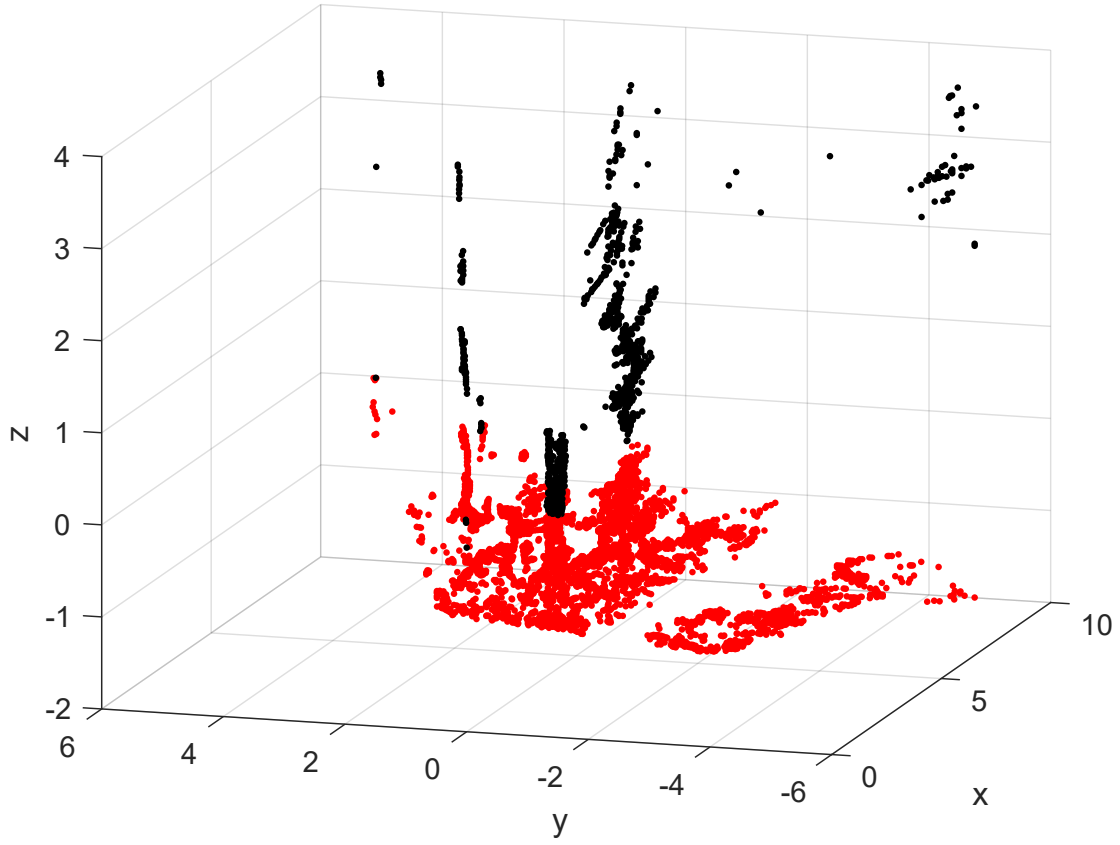


Figure 7: Point cloud after application of PassThorugh filter. (Red points which are under 0 on z-axis are removed.)

2.4 Downsampling point cloud

After removing the ground, the point cloud still contains a large number of points. To speed up processing it is convenient to use a method that would reduce its number without losing information or only with a minimal loss of information. For our purposes, we use the VoxelGrid filter (described in [14]). Using this method, the three-dimensional space is first divided into voxels (3D box in a space). In the boxes that contain a one and more points from the point cloud, the geometric center (centroid) of these boxes is calculated to represent all points in a cube and the remaining points are removed (see Figure 8). Using this method, it is very important to choose the correct size of the box. If the dimension of the box is too big and a sensor does not give us enough points, it will lead to loss of information. On the other hand, if box size is too small, the downsampling will not have the effect.

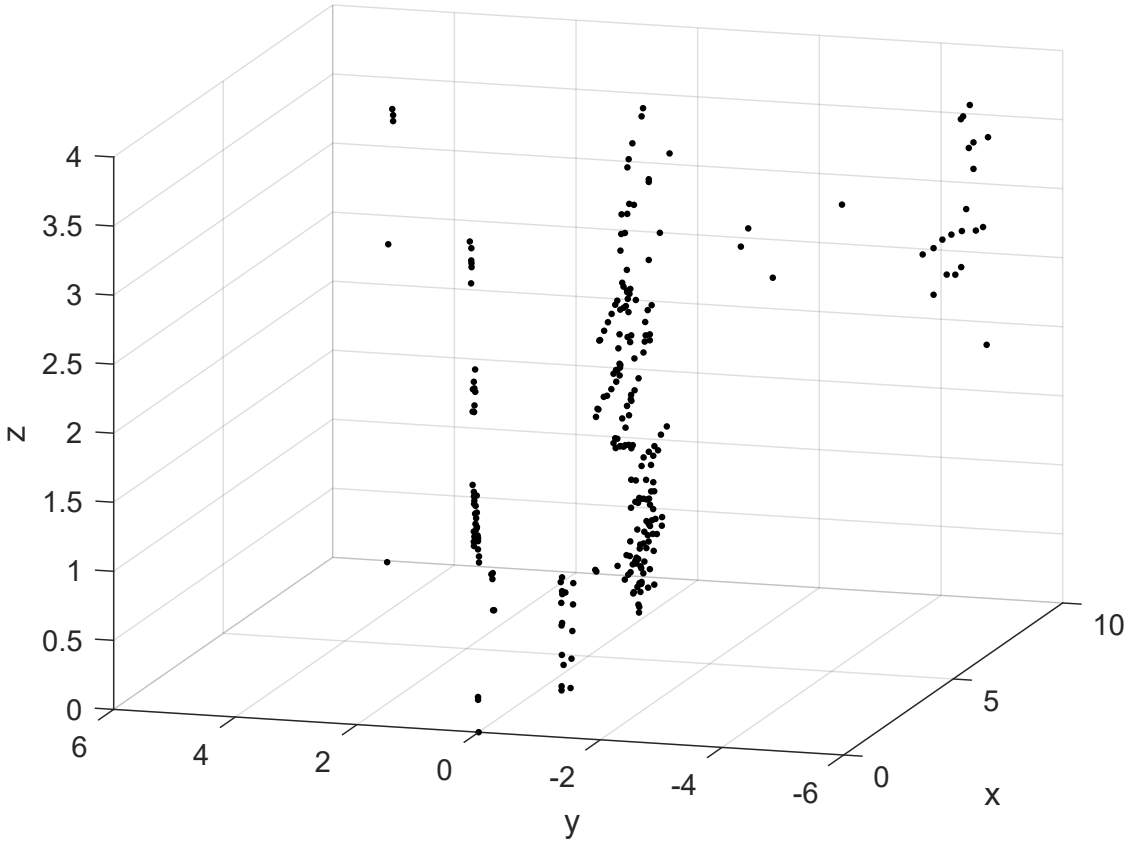


Figure 8: Point cloud after downsampling.

2.5 Removing outliers

After applying methods to reduce point cloud size, it is important to identify and remove individual points distant from others (outliers) because these outliers could confuse reconstructing algorithm. For this purpose, we compare two methods for removing outliers based on different principles.

2.5.1 Radius outlier removal filter

For using this filter we define a minimum number of neighbors which must each point have and the radius in which the neighbors are counted. The filter then counts the total number or neighbors for every point in defined radius. If the number of neighbors is bigger or equal to defined one, this point is not considered as an outlier, otherwise, the point is removed from point cloud (see Figure 9). For the proper use of this filter, it is necessary to adjust parameters, the minimum number of neighbors and radius, to the type of depth camera and environment where the UAV is flying. The number of minimum neighbors must correspond to the density of point cloud and to the defined radius. The radius depends on a minimum distance between obstacles and the density of the point cloud. If the radius is bigger than the minimum distance between obstacles, all outliers do not have to be recognized. On the other hand, if the radius is too small, points in the parts of the obstacles where the density is lower are removed. The example of removing the outliers from a point cloud is shown in figure 10.

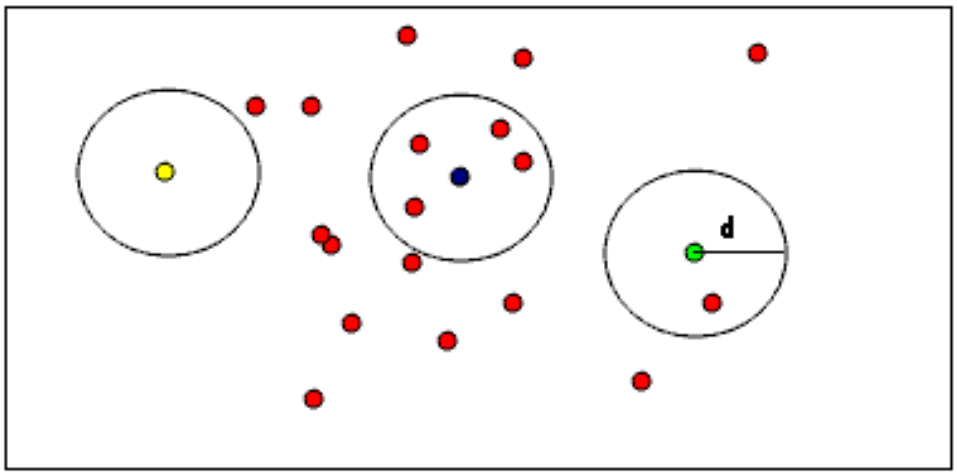


Figure 9: Principle of using radius outlier removal filter. Yellow point will be removed.

Source: http://pointclouds.org/documentation/tutorials/remove_outliers.php

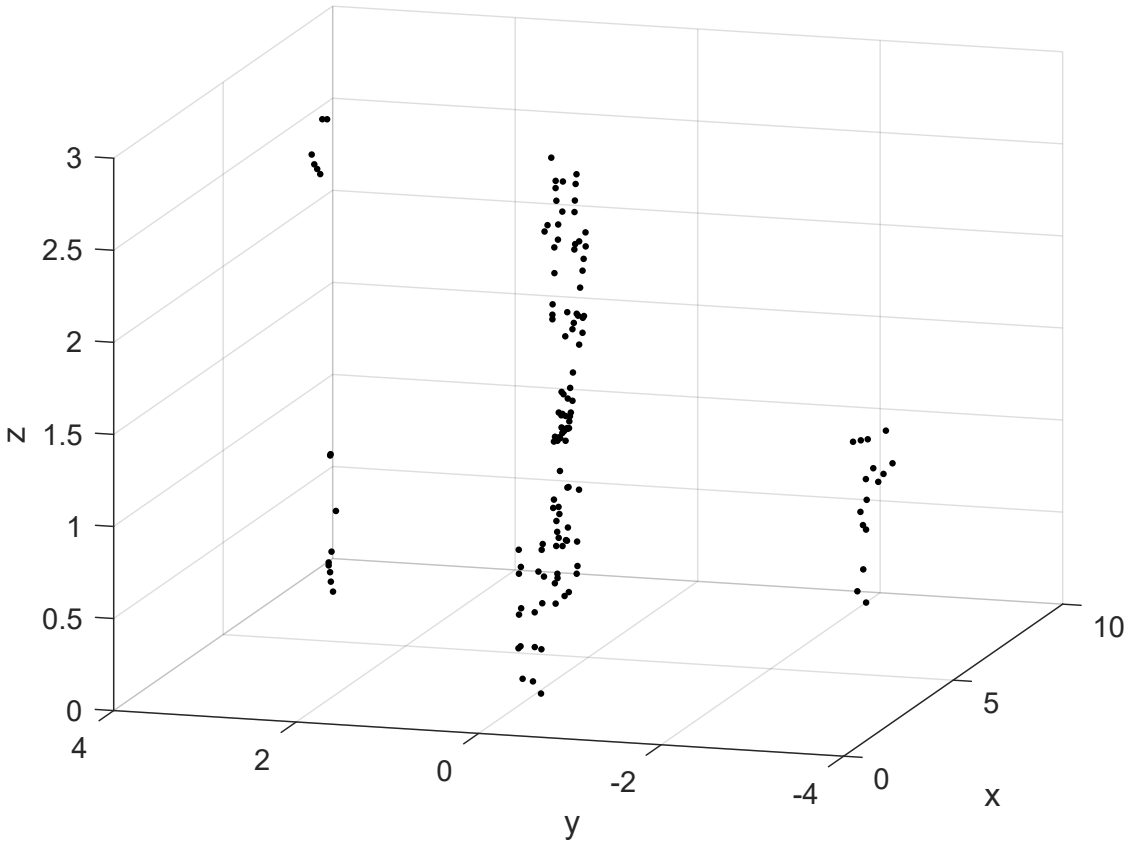


Figure 10: Point cloud after removing outliers with radius outlier removal filter.

2.5.2 Statistical outlier removal filter

The second filter that we tested is a statistical filter which uses mathematical analysis for its function. For each point, the mean distance from all neighbors is calculated. Assuming Gaussian distribution, points which have mean distances outside of an interval defined by global distances mean and standard deviation are considered to be outliers and removed (see Figure 11).

For this filter, two parameters are needed to be set. The first one is a number of neighbors used for the analysis and the second one is a standard deviation multiplier, which means how many times the average distance of point can differ from the global average distance of two points without the point being considered as an outlier.

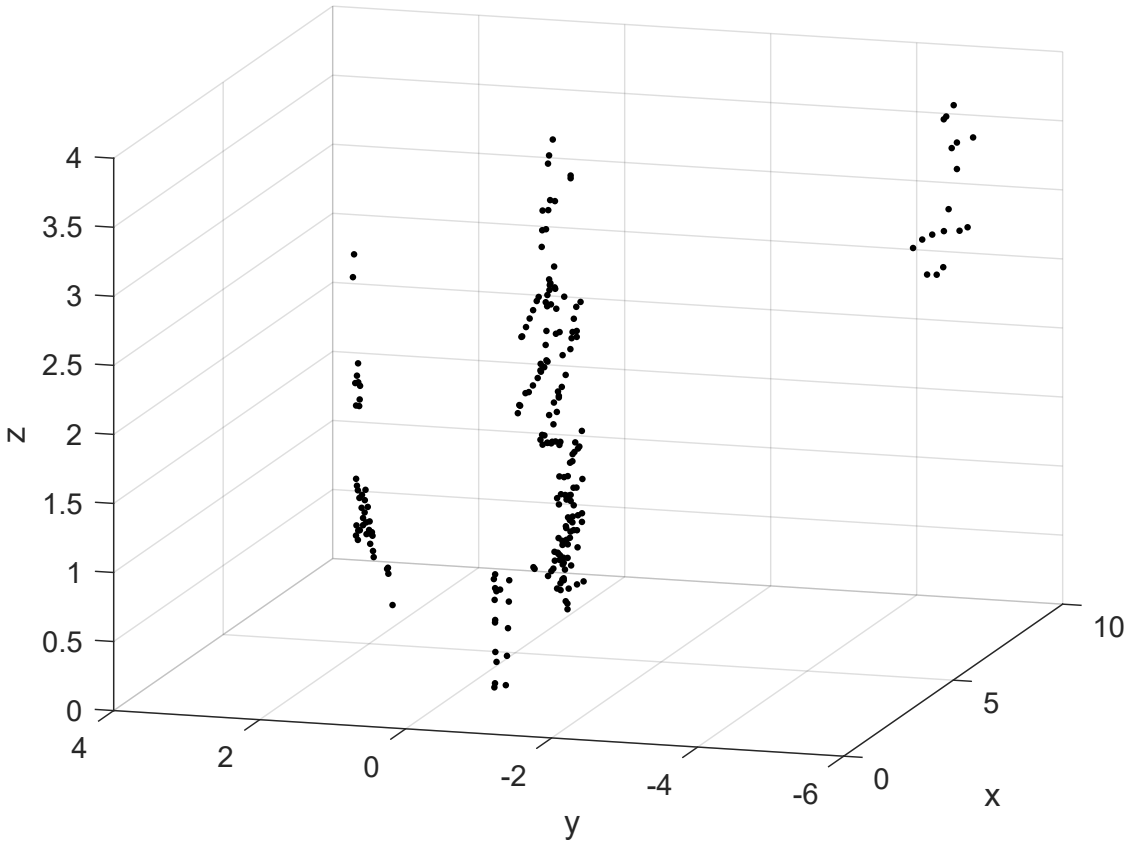


Figure 11: Point cloud after removing outliers with statistical outlier removal filter.

2.6 Reconstructing objects

After deploying of the filters and transformations we have obtained a point cloud prepared for application of reconstructing algorithm. This point cloud contains clusters of points when each of these clusters represents one detected obstacle. It is convenient to replace these clusters with objects that would clearly determine boundaries of detected obstacles, allow the fast and accurate determination of obstacle position and size.

There are many ways to reconstruct obstacles from the depth camera image. In this thesis, we present two reconstructing methods. First is so-called Delaunay triangulation. This method uses triangles for reconstructing shapes of the object. The second method is based on the assumption that we have some knowledge about terrain and we know that the detected obstacles are trees. Thus we can use this knowledge and approximate the cluster by a simple geometric object that represents tree trunks, a cylinder.

2.6.1 Delaunay triangulation

This method is based on the principle that a triangle can be constructed between every three points in the point cloud, if there is not another point inside the circumcircle in this triangle [15]. Firstly, it is necessary to find the center of circumcircle for every three points in the point cloud (see Figure 12). By definition, the center of the circumcircle is defined as an intersection of the axis of the triangle sides. To find the center of the circumcircle from three points in 3D, it is necessary to determine the intersection of at least two lines located on the plane defined by the three points. For that, we need to know the normal vector of the plane on which the axes must lie. Then it is possible to determine the parametric equations of the axes and in their intersection is the searched center of the circumcircle. The Delaunay triangulation is explained by pseudocode in algorithm 1.

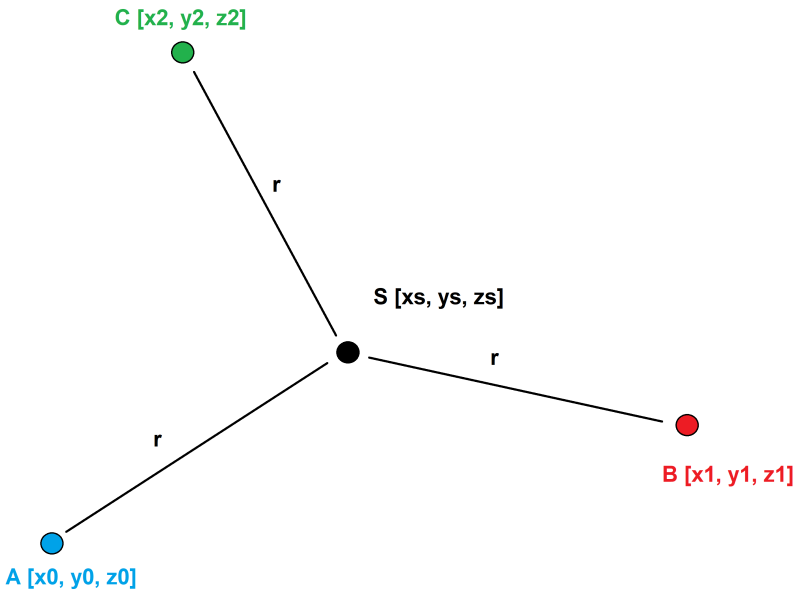


Figure 12: Center of circumcircle.

The result from this method assures that the surface will be smoothly connected to neighbors (see Figure 13). The surface is formed largely only by triangles represent shapes. It reduces the number of triangles, improves results of triangulation and saves computation power.

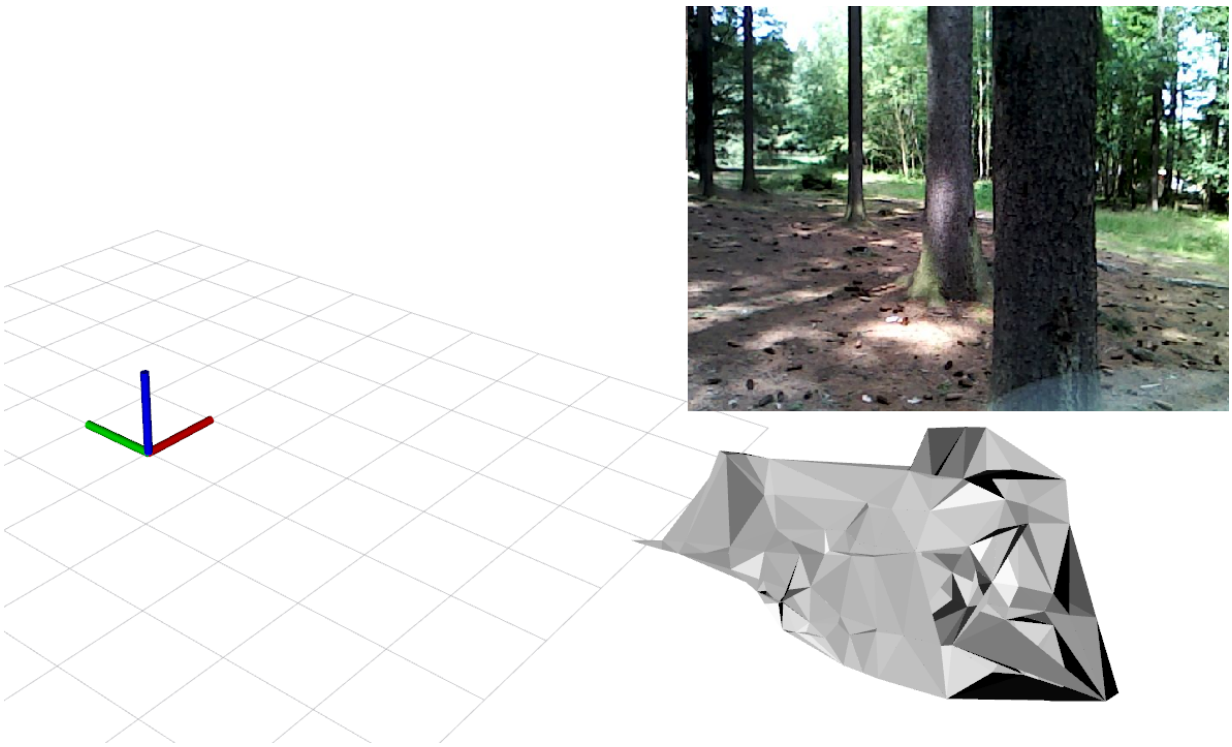


Figure 13: Result of triangulation. (In the top-right is for comparison, image from the color camera.)

2.6.2 Reconstructing into cylinders

This algorithm is based on the knowledge of the obstacle shapes in the terrain where UAV is flying. We assume a specific case, that the obstacles are trees with the shape of a stem which is similar to a cylinder. With this assumption, we can describe reconstructed objects more simply than by the triangulation. Each detected object is described by a point in space representing the center of the cylinder, a quaternion for the exact definition of cylinder rotation, and by two variables defining radius and height.

For proper identification, it is important to separate the points representing individual trees into individual clusters using clustering algorithm. This algorithm use the maximum distance from a point to the neighbor point to consider if the point is a part of the cluster or not. Furthermore, the minimum and maximum cluster sizes are set. The algorithm is explained by pseudo code in the algorithm 2.

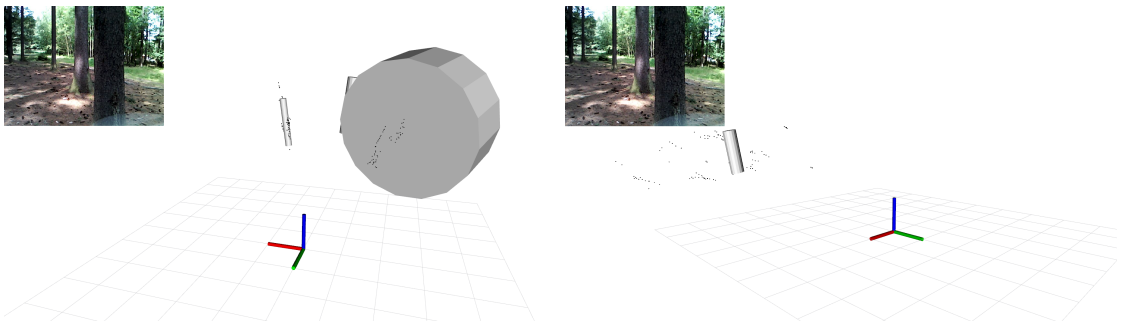
It should be noted that this clustering algorithm performs the same work as the previously mentioned outlier removal filter does. It is therefore possible to turn off the filter for this type of reconstruction and thus speed up the overall identification process. After dividing the point cloud into individual clusters that represent detected objects, it is necessary to apply procedures that accurately determine the centers of the cylinders, their

sizes, and orientations. As the first, we calculate the centroid that represents the center of the cylinder that tree. This is the geometric center of all points in the cluster and has the average value of the coordinates of all points in the cluster. It is also necessary to determine the sizes of the cylinder, its radius, and height. Ideally, this task could be solved by detecting a limit value of each coordinate in the cluster. The height would be determined as the difference between the maximum and minimum values on the z-axis. The diameter could be determined as the average of the maximum and minimum difference values on the x and y-axes. In our case, this cannot be applied, it is clear from experimental data that there is a large error in the size of the longitudinal sides of the obstacles (see Figure 14). Obstacles seem to be more elongated than they really are. If we decide to ignore this fact, this error would cause the diameter of the cylinders to be much larger than it is in reality, and the rotation of the trees would not match too.

Figure 14: Longitudinal elongation of detected trees received from the camera (Marked by red ellipse).(In the top-right is for comparison, image from the color camera.)

The reader may notice that the width corresponds to reality and can be therefore used to correct this error. Due to this fact, we apply rotation on the points in the cluster that are rotated so that the tail is centered in the y-axis direction. Now we use x-axis deviation which must correspond to y-axis deviation and remove error points. In the same time, we determine the diameter of the cylinder which equals to maximum x-axis deviation, height corresponds z-axis deviation and center is evaluated as a centroid of this cluster. Next, we use a so-called Principal Component Analysis (PCA) algorithm to determine the vector in which the largest number of points is located [16]. This information allows us to determine the most likely direction of cylinder orientation. The rotation in 3D space is represented in this thesis by a quaternion. In the case when the UAV accidentally leans and detects the ground or detect an inappropriate obstacle that is not similar to the expected shape, it is necessary to insert a safety check that defines the maximum and minimum limits for

the size of the tree and its rotation (see Figure 15b). If such a check is not set, incorrect fitting of the cylinder could occur in exceptional cases (see Figure 15a).



(a) Accidental ground detection. (b) Accidental ground detection with fuse.

Figure 15: Comparison between two modification of reconstructing by cylinders. (In the top-left is for comparison, image from the color camera.)

Reconstructing of a point cloud from the forest-like environment using approximation by cylinders is faster, more resistant against the influence of noise, and the detected objects are described more clearly than by using reconstructing method based on triangulation (see Table 1). Due to our specific application, the advantage of the method based on triangulation will not appear because our UAV will fly only in a forest-like environment. For this reason, in the rest of this thesis, we will consider only method which reconstructs objects into cylinders.

Method	**Average publishing rate [Hz]**
Delaunay triangulation	6.066
Reconstruction by cylinders	12.647

Table 1: Comparision of publishing rate between two reconstructing methods.

Algorithm 1: Delaunay triangulation algorithm

```
input  : pointCloud input - filtered input point cloud
         int k - k nearest neighbors
output: vectorOfTriangles output - triangles representing shapes of obstacle
begin
    int iterationNumber = 0;
    // While there is any possible combination of triangle, iterate
    while isUniqueTriangle(input, iterationNumber) == true do
        // Get unique combination of new triangle for testing
        triangle testedTriangle = getTriangle(input, iterationNumber);
        iterationNumber++;
        // Center of the circle and radius
        point S = cicumcircleSearchAlgorithm(testedTriangle);
        float r = getDistance(S, testedTriangle);
        // Get subcloud including only closest neighbors of center
        pointCloud neighbors = getKnearestNeighbors(input, k, S);
        // Count forbidden area of circle where no points can be
           situated except points of tested Triangle
        bool isFineTriangle = pointsInsideOfCircle(neighbors, testedTriangle, S,
         r);
        // If fine triangle add it to output vector
        if isFineTriangle then
            output.add = testedTriangle;
        end
    end
    return output;
end
```

Algorithm 2: Clustering algorithm

```
input : pointCloud input - input cloud
        float radius - maximum radius of search
        int minSize - minimum size of cluster
        int maxSize - maximum size of cluster

output: vectorOfCLouds clusters - center of circumcircle
begin
    queue q;
    // Go through all points in cloud
    int i = 0;
    for i <input.points.size do
        // Is point actually member of any cluster?
        if input.points.at(i).processed == false then
            // If no, add point to a queue and search for all his
               neighbors representing cluster
            q.pushBack(input.points.at(i));
            input.points.at(i).processed = true;
            int j = 0;
            for j <q.points.size() do
                pointCloud neighbors = getNeighbors(q.points.at(j), radius);
                int k = 0;
                // Test if all neighbors are participants of any cluster,
                   if no, add them to new cluster
                for k <neighbors.points.size() do
                    if neighbors.points.at(k).processed == false then
                        q.pushBack(neighbors.points.at(k));
                        neighbors.points.at(k).processed = true;
                    end
                end
            end
            // When cluster is complete, test if size is appropriate and
               then add it to output
            if q.size() <maxSize AND q.size() >minSize then
                clusters.pushBack(q);
                q.makeEmpty();
            end
        end
    end
end
return(clusters);
```

3 Local map

Until now, we can process incoming point cloud from the camera into a set of detected obstacles. However, to meet the goal to be able to implement an avoidance algorithm, this is still not enough. At this level, the obstacles are too dynamic, frequently changing their position and shape, depending on each camera shot. Obstacle information is at the moment series of flashbacks with actual information about a position which does not have a long duration and thus cannot be used as a relevant information for planning.

In order to plan a collision-free flight, we need to know the approximate position of the obstacle which would be ideally placed statically in one place, or changing position only little. Therefore, it is necessary to design an algorithm that will suitably handle data about the detected obstacles, save them regardless of whether new data has come from the camera or not. It must be recognized if the new shoot from the camera describes new obstacles or if they have been already detected. In addition, errors due to an inappropriate UAV flight need to be filtered out. For example, the UAV accelerates too quickly and the depth camera leans and detects the ground despite of application PassThrough filter. It is unacceptable for the UAV to consider the ground as an obstacle and to start an evasive maneuver. Such behavior leads to the wasting of battery capacity, it could cause oscillation and in the worst case, collide with an object located a few meters behind the UAV. All this has to be incorporated into an algorithm that defines a local map. This map must contain short-term information about obstacles in the environment and be updated by each shot.

3.1 Creating of the local map

The simplest local map, that could be implemented, is just method that stores the received data in memory and erased them after a certain time. However, such a simple solution has a number of disadvantages. In particular, it is computationally and memory demanding for trajectory planning in a more complex terrain. Thus it is necessary to make a local map which takes into account how old detected obstacles are. By using ROS we can advantage of time information included in each ROS message. Due to this information, it is possible to compare the time at which the obstacle was created with the current time, and set a frequency of deleting old obstacles in the map memory according to the planned maximum UAV speed. All detected obstacles have information about the number of occurrences and only when a certain limit is exceeded, they are marked as detected objects. This is for an elimination of an error when some object is detected once and considered as a potential obstacle (For example accidentally detected ground). Additionally, the number of detected objects is reduced by comparing obstacles that are currently detected with the obstacles that have occurred in new shot. Positions of the new detected obstacles are tested whether they match to some already detected obstacle. In this case the position of the already detected obstacle is updated, otherwise the new obstacle is

embedded. Implemented local map has information about how far the obstacle is located and in what direction from the UAV is. After applying all aforementioned rules, we have a local map that reliably stores information about detected obstacles in the surroundings (see Figure 16).

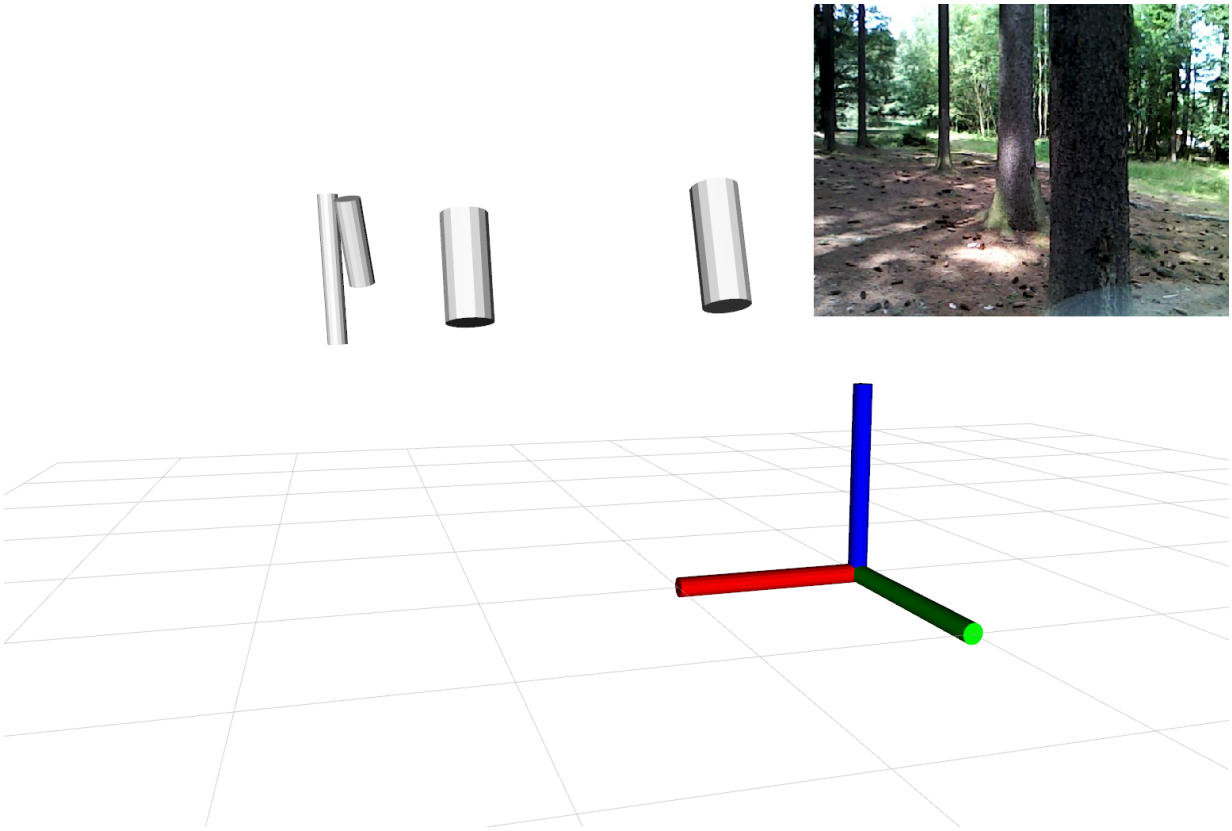


Figure 16: Local map with detected cylinders. (In the top-right is for comparison, image from the color camera.)

3.2 Preparing data for reactive collision-free technique

In this thesis, we present reactive collision-free technique, which does not keep information about all detected objects within the map. It only keeps information about the detected obstacles in the closest surrounding, this allows us to work only with the necessary information. It is therefore important to divide space and define where obstacles are located and where the UAV can fly. If we detect trees in the UAV frame, where the UAV is the center of the coordinate system, the individual obstacles are represented as a position and a certain direction towards the UAV. Thus it is possible to divide the space into directions in which the UAV can flight. Each of these directions will have information about a number of detected obstacles and it will provide us information how safe is to fly this way (see Figure 17). This is called polar histogram [17].

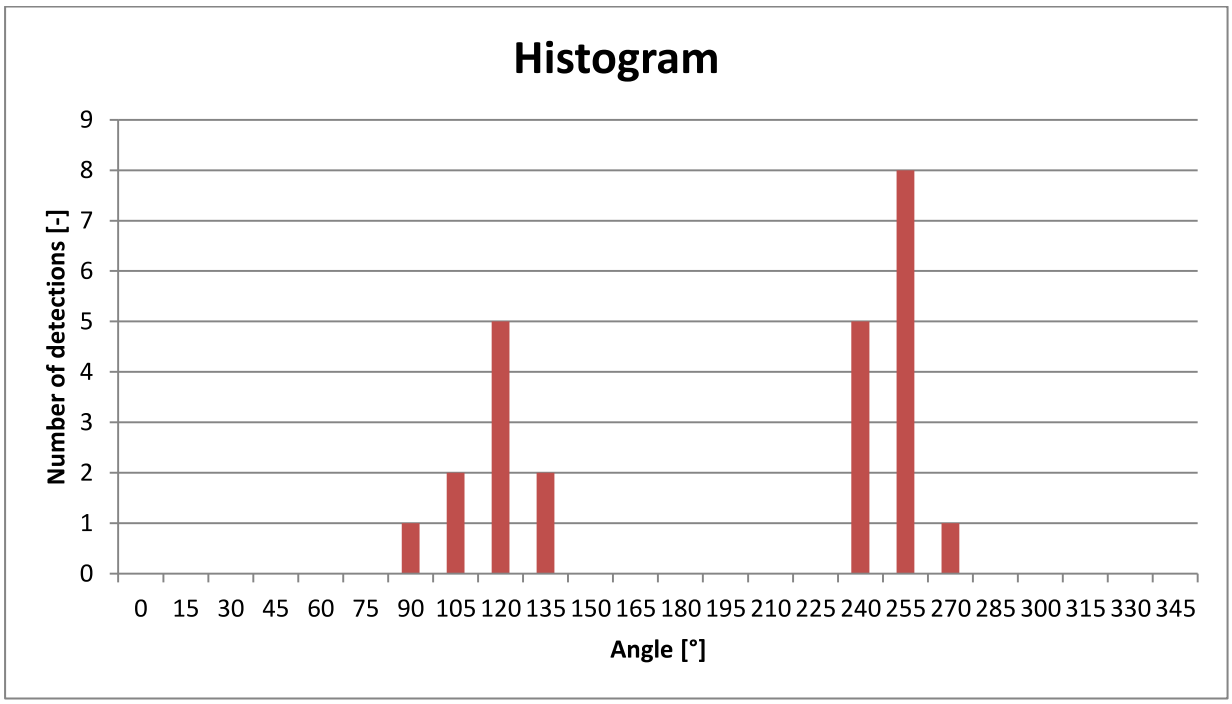


Figure 17: Example of histogram with obstacle detection distribution.

For safety purposes, it is necessary to work with the thickness of the obstacles. If we only consider the central point for simplicity, we would not be able to identify obstacles going beyond other corridors, which by following the collision-free path in the worst case would lead to a collision. It is important to incorporate the diameter and count angle which is occupied by the obstacle using formula

$$\Phi = \Psi \pm \arctan(r/l), \tag{2}$$

where Φ is maximum and minimum angle occupied by an obstacle, Ψ is a direction of the obstacle center, r is a radius of a tree and l is a distance of tree from UAV. If we use the angle of the center of the obstacle and occupied angle we can mark all corridors that are occupied by the obstacle. Now we have information about free and occupied directions and we are able to implement the reactive collision-free technique.

4 Implementation of reactive collision-free technique

The algorithm for planning collision-free trajectories for UAVs is the most important part of this thesis. It covers the entire processing of input data and it decides about the success of the mission or its failure. Initially, a suitable method for reactive collision-free planning has to be selected (described in [18]). Such a method is a VFH (Vector Field Histogram) based on polar histograms. This is the field of n-sectors where each sector is m degrees wide. Each sector holds the number of occurrences of obstacles that have occurred in that direction. If this number exceeds the defined value, the sector is considered as ineligible. After time t the histogram is reset (see section 3.2). More complexly described in [12], [18] and [17].

4.1 Introduction into VFH algorithm

The VFH algorithm is based on the polar histogram which holds the information of all the obstacles and on the knowledge of the direction towards the target. If the number of detections in this direction exceeds the threshold, there is a slight deflection of flight direction into the nearest sector, where the number of detection is still below the limit and therefore it is possible to fly this way. For the application and proper function of the VFH algorithm for collision-free flight planning, several assumptions need to be met.

We need to know the coordinates of the UAV and the target to which the UAV is heading. This is essential for planning the flight path and for planning evasive maneuvers. It is not important how precisely the location is determined, since inaccurate localization of the UAV the result in the same displacement of localized obstacles and the error is eliminated. The depth camera must be mounted at the front of the UAV, or the UAV must fly, so the depth camera is focused in the direction of the flight. This is necessary in order to be able to detect the obstacles approaching from the front because these obstacles need to be detected. A setting of the polar histogram corridor width must match with the environment in which the UAV moves. If the corridor width is too large, it may happen that the UAV is not able to reach the target in the finite time, because no collision-free path toward target position is found. Objects that are detected must be embedded in a polar histogram with a certain margin reserve to avoid tight passes or collisions. In practice, this means that the diameter of trees was inflated by the size of the UAV or the requirements for the safe flight.

4.2 Implementation of VFH algorithm

Implementation of the algorithm itself is divided into three parts. The first is the part that takes care of the evaluation of the data stored in the polar histogram and the

determination of the flight strategy. There are also two parts that are in direct contact with the MPC (Model Predictive Control) tracker, implemented in Multi-robot Systems Group in Department of Cybernetics, CTU in Prague, and take care for the UAV translation and rotation planning [19].

4.2.1 Control

To design simple and effective UAV controller and trajectory planner, it is important to determine the number of events that may occur. If the number of possible events is not big and it is easily identifiable, it is possible to split the UAV behavior in several cases and use a finite-state machine. In our case, the UAV can record a total of four events. The first is the "Do nothing" instruction. This instruction is recorded if the UAV has already reached its destination and have the proper rotation, thus nothing needs to be done. In this case, the UAV use only small interventions to keep in the target position and wait for further instructions. The second instruction is "Rotate about the angle". This instruction must be called before any change, of course, to make the depth stereo camera still focused on the objects before the UAV. Similarly, this instruction may be recorded in the target position (including specific rotation) and the UAV must then rotate in this case too. The third instruction is "Move from position A to position B". At this point, it is assumed that the UAV is rotated correctly and a trajectory from A to B is planned. The last is the information "You are in a collision course". In the case of receiving this information, if the UAV is not in the target position, an evasive maneuver must be performed and the trajectory must be rescheduled over the nearest free angle. At this point, we use a transition point, which is located at a certain distance in the collision-free direction and where the UAV flies until the desired angle is marked as the collision. Every combination of these events is a big number (256 combinations if we assume using only unique combinations) and it is not fine for using the finite-state machine. But assuming only the real situations, in which the UAV can really occur, simplify solutions and we are able to make flight manager composed from a finite-state machine with only 4 states (see Figure 18).

4.2.2 Translation

In order to create a path from point A to B, to be executed by the UAV, it is necessary to know the step to follow, which depends on the MPC tracker settings, in our case, it requires information about the position of next point per 200 ms. Thus when determining the UAV velocity, for example at 1 ms^{-1}, we can easily determine how far the points need to be for keeping set speed (see Formula 3).

$$step = v \cdot d = 0.2\ m \tag{3}$$

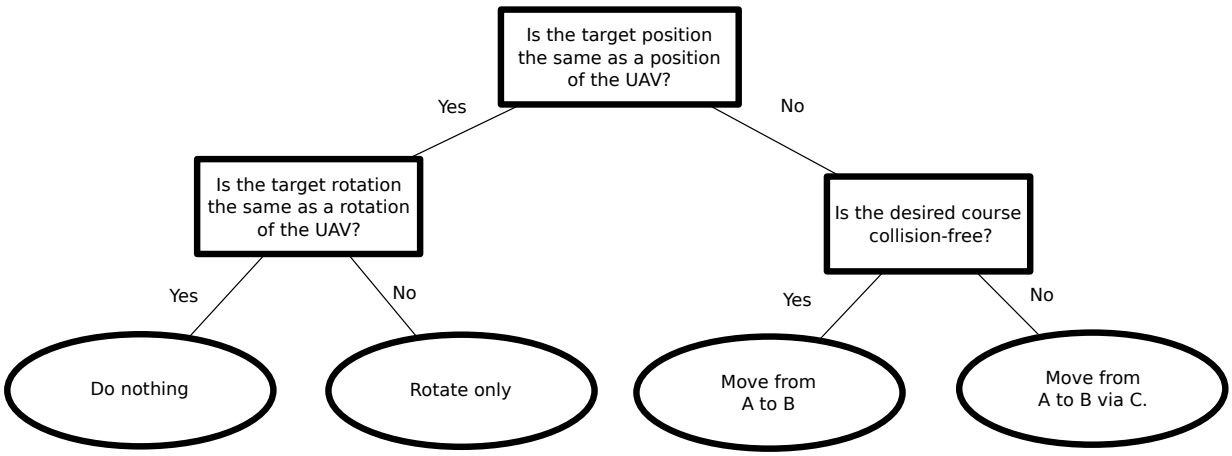


Figure 18: Decisions tree for the collision-free control of the UAV.

When establishing a trajectory approximation into a straight line is sufficient, there is no reason for joining two or three points with a difficult curve. The vector which defines the direction of the flight is known as well as the starting point. Thus we can shift starting point with step in direction of the vector and get all points from point A to point B. The sum of these points is searched trajectory (see Algorithm 3).

4.2.3 Rotations

When planning a rotation, as in scheduling translation, splitting the rotation into a few steps is necessary. In the case of a shift, we determined the UAV translation velocity. In this case, we have to determine the angular velocity and the phase shift as a step. First, it is necessary to determine whether it is faster to rotate in the clockwise direction or in the counterclockwise. For that two angles need to be found; first angle, if the UAV rotates clockwise and second if it rotates counterclockwise. Direction with smaller angle is then selected. After defining the rotation direction, it is necessary to define the desired angular velocity and the step that is added or subtracted from the current angle depending on the direction of rotation. This defines all the points on the trajectory (see Algorithm 4).

Everything necessary for implementing the VFH algorithm is now available. There is a logic that evaluates the current situation in which the UAV is and also that plans adequate responses.

Algorithm 3: Get shift trajectory

```
input  : float v - velocity of UAV
         point A - point A in local origin frame
         point B - point B in local origin frame
output: trajectory output - trajectory from start to end
begin
    int counter = 1;
    float step = 0.2v;
    float t = step/getDistance(A, B);
    point tmp = A;
    vector u = getVector(B, A);
    while getDistance(B, tmp) >0 do
        // set point shifted about u · t · counter
        tmp = setPoint(tmp, u, t, counter);
        trajectory.pushBack(tmp);
        counter = counter + 1;
    end
    return(trajectory);
end
```

Algorithm 4: Get rotation trajectory

```
input  : float ω - angular velocity of UAV
         point A - point A in local origin frame
         point B - point B in local origin frame
output: trajectory output - trajectory from start to end
begin
    int counter = 1;
    int dir = getShortestDirection(A, B);
    float step = 0.2ω;
    point tmp = A;
    while getAngularDiference(B, tmp) >0 do
        // set point shifted about step · counter
        tmp = setPoint(tmp, dir, step, counter);
        trajectory.pushBack(tmp);
        counter = counter + 1;
    end
    return(trajectory);
end
```

5 Implementation of a depth camera model in Gazebo robotic simulator

The basis of a successful project in the robotics sector is the possibility of simulating the functionality of the program. For nearly every program of a control system composed of mechanical components, it is necessary to set parameters that define certain quantities, which then influence the behavior of the whole system. This is especially true in the aviation industry, where each failure can be fatal.

For our purposes, we use the Gazebo simulator to realistically simulate outdoor conditions for a wide range of robotic applications ranging from robotic vehicles, robot populations to flying objects. Into the world of Gazebo simulator, it is possible to interfere with the ROS interface and use service and messages to control models and gain information from the world of Gazebo simulator (see Figure 19).

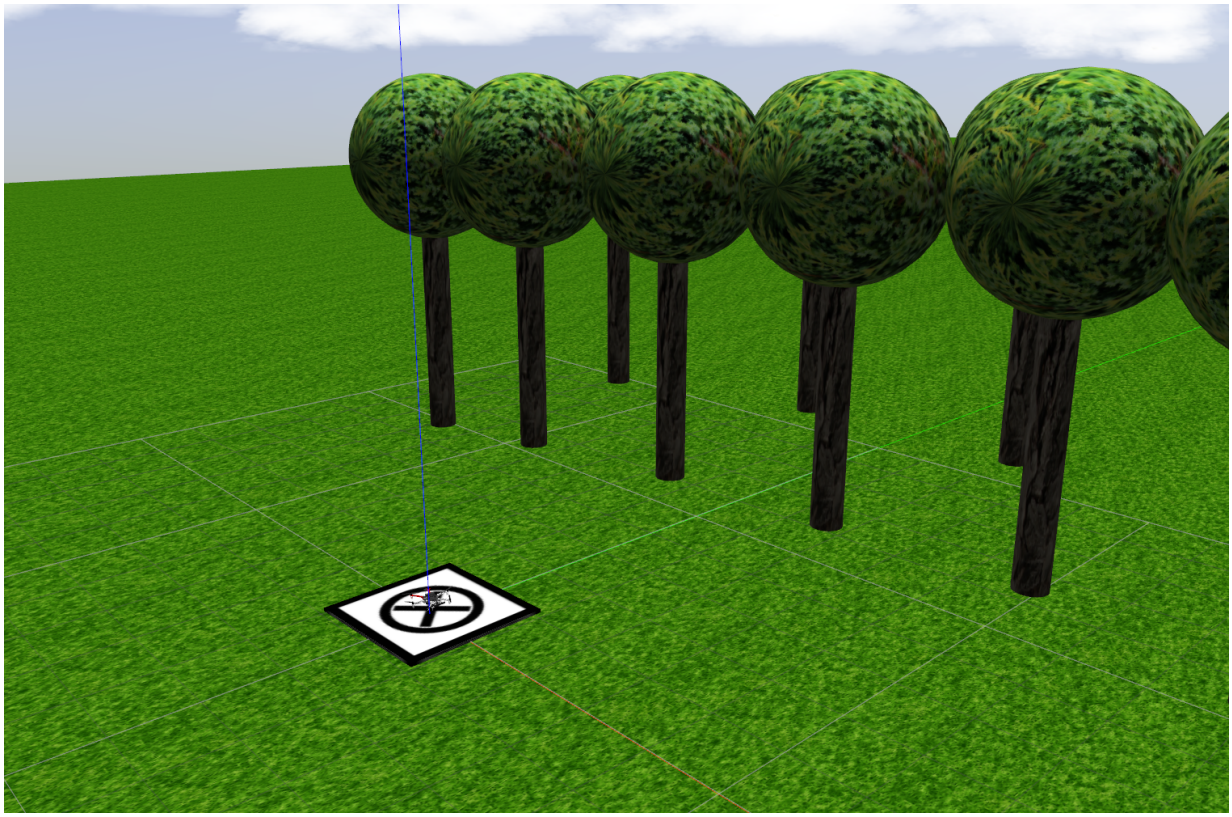


Figure 19: Example of Gazebo simulation world with terrain and model.

5.1 Introduction

Models in Gazebo simulator are described using SDF (Simulation Description Format). SDF is an XML (eXtensible Markup Language) format that describes objects and environments in robotic simulators and visualizations. This is a years-proven format capable of describing all aspects of robots, environments as well as physics.

From the stereo camera model, features such as resolution, FOV (Field Of View), minimum detection distance, maximum range, and UAV positioning are required to match the real sensor. On the other hand, the characteristics of camera dimensions can be defined approximately because they are not so important for our application.

5.2 Depth camera model implementation

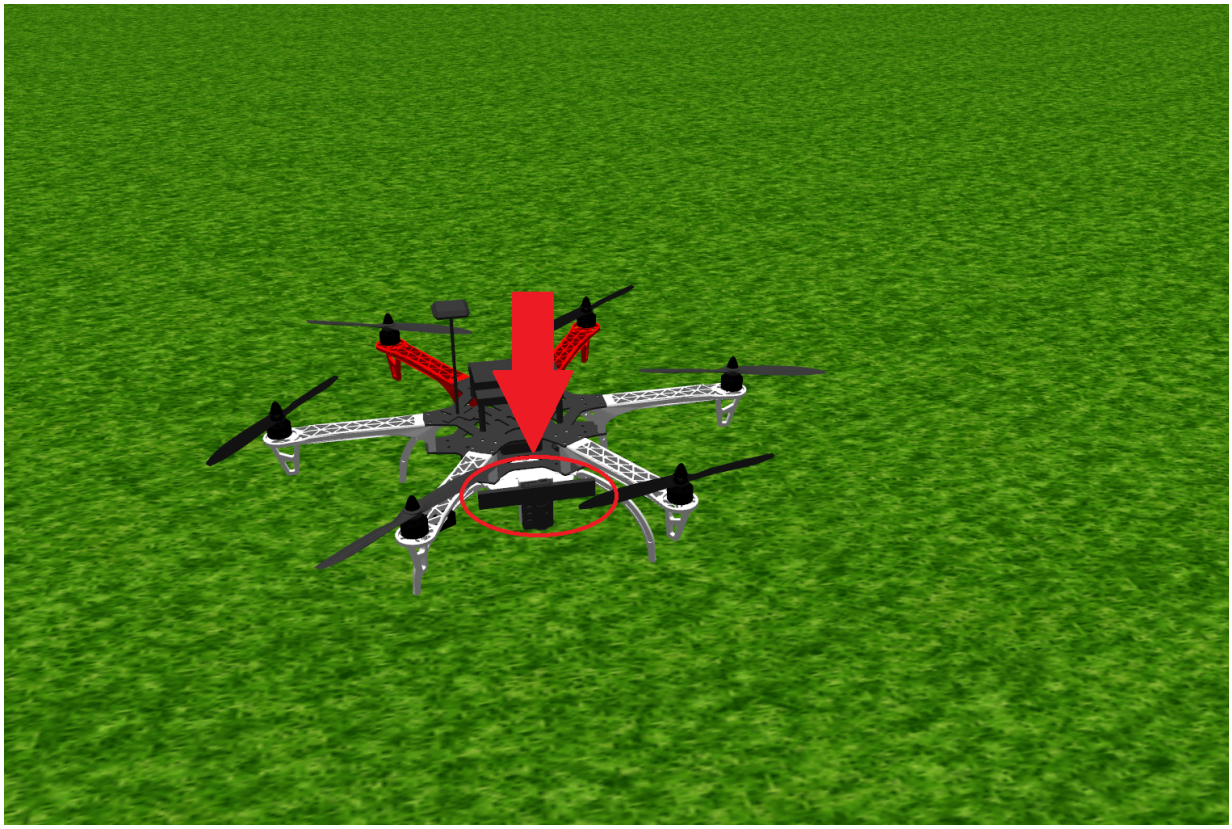


Figure 20: Visualization of implemented Gazebo model representing depth camera R200.

It would be possible to implement the entire environment, the robot properties and obstacles exclusively using XML files, but the file containing this information would be very long and unclear. This is one of the reasons why the so-called Xacro (XML macro) files

are used. Using these macros, which represent larger XML expressions, keeps information describing the world in more readable format.

These macros have one more useful feature. For example, if we have a macro that defines a depth camera, we can use this macro to define the next level of macros that more specifically defines the properties of a specific camera, such as resolution, the minimum distance from the obstacle etc.

For our purposes, a general depth camera Xacro is implemented and then explicit R200 camera Xacro is defined. Parameters are set as the manufacturer defines, for example, resolution, the FOV, etc. For visualization is used a block with approximate dimensions of the R200 camera (see Figure 20).

6 Simulation of flight

Simulations are extremely important in the aerospace industry. In our case, we try to prepare a simulation for a situation where the UAV is in a plane with trees where it detects trees and its goal is to fly from UAV position to target position. The environment contains a finite number of obstacles with more unspecified sizes. All we know is that the obstacles in the environment are trees. It is therefore very important to set simulations and check the functionality of the proposed technique to correct setting of all parameters. The result of the simulation is ideally a tuned program which is able to deal with every real situation for which is prepared for. For this thesis, we have set four simulations in which the program's functionality is tested in particular. It is necessary to test the functionality of all the parts we have implemented so far. From filtering and processing of a point cloud to object reconstruction and trajectory planning.

6.1 Simulation of trajectory planner

First of all, it is important to test the trajectory planner and logic that control the UAV. In this simulation, we test the functionality of trajectory planning for co-ordinates entered during the flight and track the UAV behavior during achieving goals. There is no obstacle present on the map in this simulation.

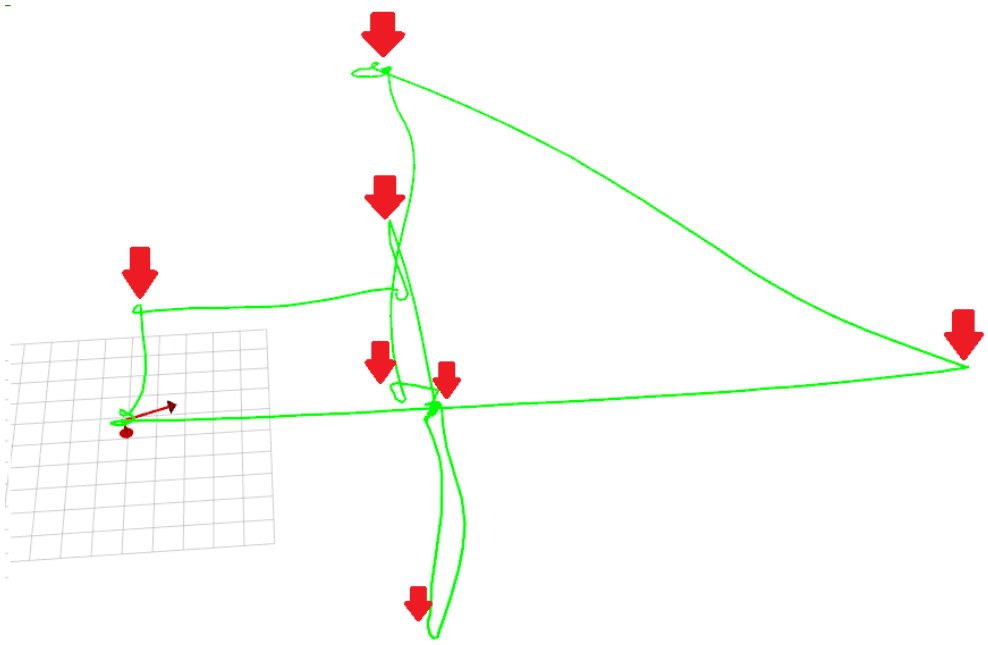


Figure 21: The trajectory of UAV during simulation of trajectory planner (Target positions are marked by red arrows.)

First, a simple, rotate-free trajectory was set up to determine if the UAV is capable of planning. Additionally, the target for which the UAV has to be rotated in the direction of the target has been added, and the sequence of instructions and UAV rotation speeds have been observed. In addition, it was tested how the UAV shift while rotating in place. Finally, the functionality of in-flight re-planning was tested. When the UAV was assigned a target, the target coordinates were repeatedly changed during the flight and the response and magnitude of the fluctuations were observed. The whole trajectory is shown (green line) on the Figure 21.

6.2 Simulation of avoiding a single obstacle

In the second simulation, we test the decision making of the trajectory scheduler's control unit to detect obstacles on the collision course. This is the easiest possible situation to test the evasive maneuvers, but it is essential for testing the functionality of the program. A single tree model is built on the map directly in front of the camera so that the UAV does not have to rotate and there is no deflection that could cause the UAV not to be in a collision course when passing a tree pattern. On a line that passes through the geometric center of the UAV and the tree, a target is defined at a certain distance from the tree so that the UAV has to fly around the tree and apply the evasive maneuver when flying the shortest possible trajectory.

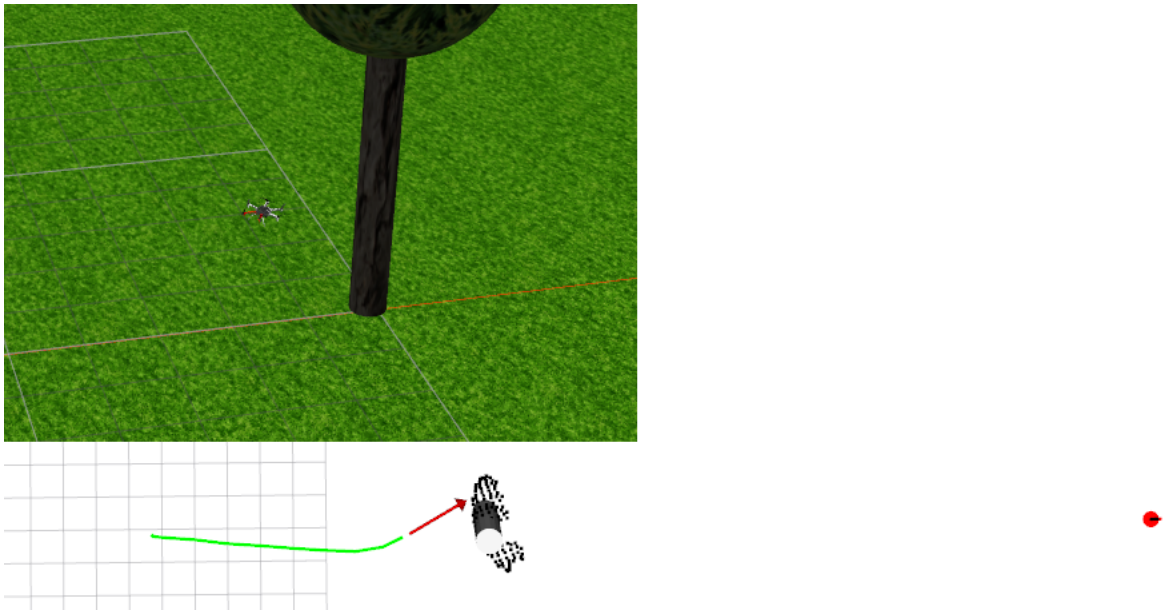


Figure 22: Avoiding maneuver of UAV. On the top-left is shown the actual situation in simulation world, below this, is the situation when the UAV detects tree (cylinder) and change direction (red arrow) to avoid the obstacle. The target position is presented by the red arrow pointing down into the map.

In the Figure 22 , the target is shown with a red arrow pointing down into the map. Furthermore, the direction in which the UAV wants to fly is shown with a red arrow followed by a green line showing the UAVs trajectory. After detecting the obstacle by the UAV, the flight direction is diverted to the nearest collision-free course leading to the target. The collision-free trajectory of the UAV after reaching the target is shown in Figure 23.

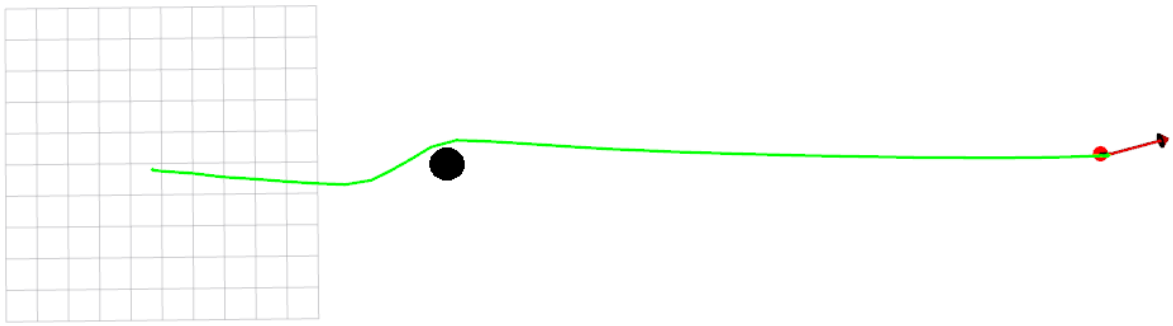


Figure 23: Trajectory (green line) after achievement target position (red arrow pointing into the map) of UAV during simulation of avoiding the single obstacle. (Position of the obstacle is highlighted by the black circle)

6.3 Simulation of avoiding tree wall

For the third simulation, we designed a specific tree formation to test how the control unit handles a situation where there are more obstacles placed close to each other. The goal was to find out if proposed method evaluates these obstacles as one big obstacle, or whether the algorithm is able to recognize them correctly.

It can be seen in Figure 24 that the algorithm correctly detects the obstacles. During the flight, the algorithm chose the most advantageous direction, which is in this case larger than in simulation with one obstacle, shown in Figure 22.

The situation which appeared after a certain time when the UAV has forgotten that there are the obstacles between the UAV and target position is described in Figure 25. Due to that, the UAV tried to fly directly to the target position. However, it began again to detect the obstacles and thus to fly the collision-free course. It is clear that this will be repeated once again because in the current situation the UAV is again considering the direct direction as free, which is shown the red arrow pointing directly to the target. Now, there will be only slight deflection because there are no more trees on the sides that would force the UAV to use such a disadvantageous angle as previous.

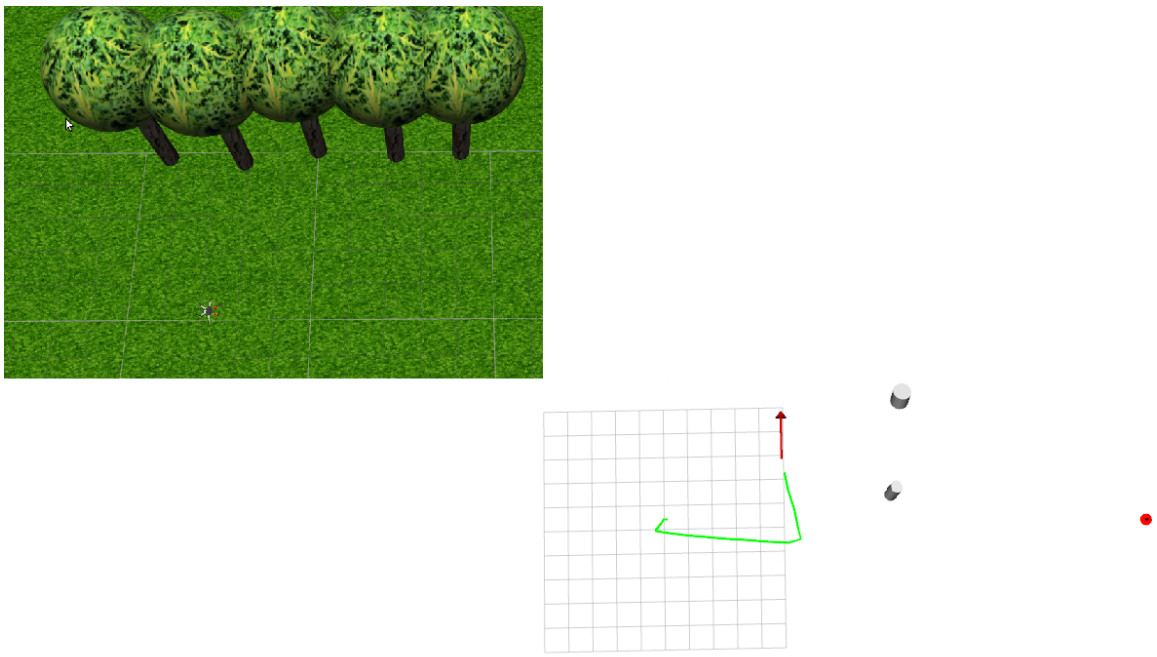


Figure 24: The initial response of the UAV due to tree wall. The UAV detects obstacles (marked as cylinders) which do not allow using a straight line to the target position (red arrow pointing down). The UAV changed direction (red arrow) to closest collision-free direction.

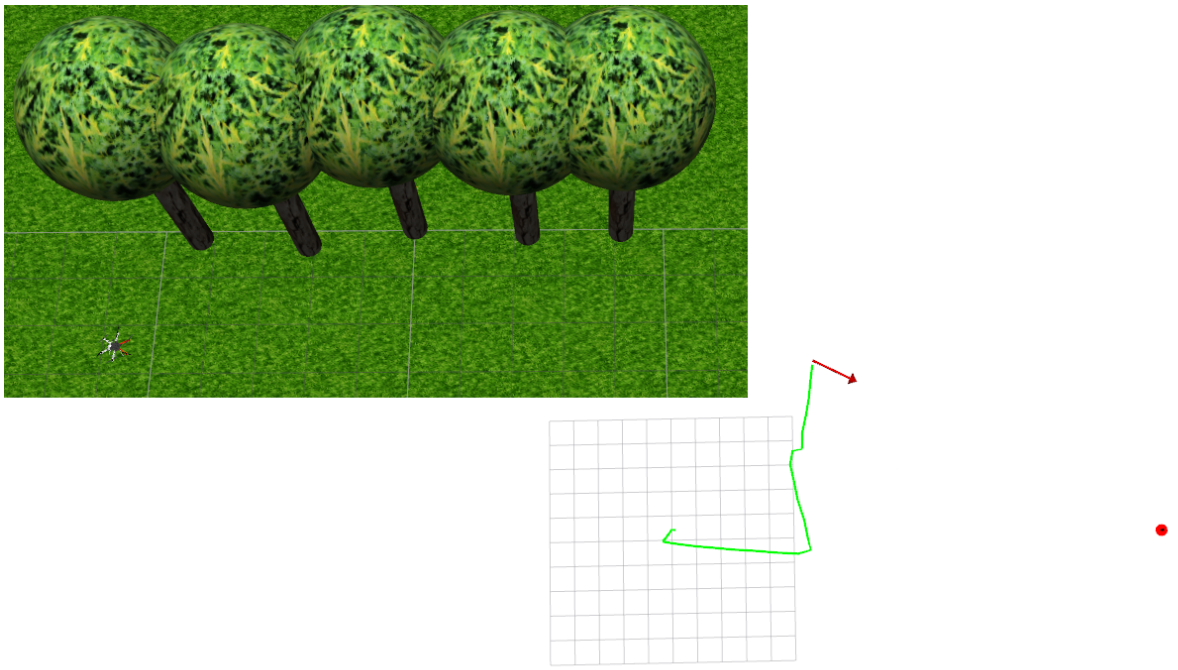


Figure 25: The UAV after forgetting information about obstacles. The UAV is going to rotate and try to fly in this direction if it is free.

6.4 Simulation of complicated environment

In the last simulation, we test a complicated situation when a group of trees is approximately as wide as the long (see Figure 26). The aim of this simulation is to test the ability of the implemented logic to deal with situations where the obstacles form a dense tree arrangement.

Figure 26: Simulation with complicated dense tree arrangement.

The result from this simulation is shown in Figure 27 where the UAV detects the trees and selects the most advantageous collision-free angle that leads to the target. This direction it flew as long as the angle is still considered to be the most advantageous, or until the direction is still marked as a collision.

The final trajectory is displayed in Figure 28. There can be seen that the original direction was not, in the end, sufficient and collision-free. The UAV had to change direction once again to select another collision-free direction. Subsequently, no obstacle was detected in this direction and previously detected obstacles were already forgotten. Therefore, the most advantageous direction that would lead to the goal was reevaluated. Then it was possible to choose direct direction because there were no obstacles in the way. Therefore we verified the proper function of the proposed method in Gazebo simulator and thus it is prepared for testing in by real-world deployment.

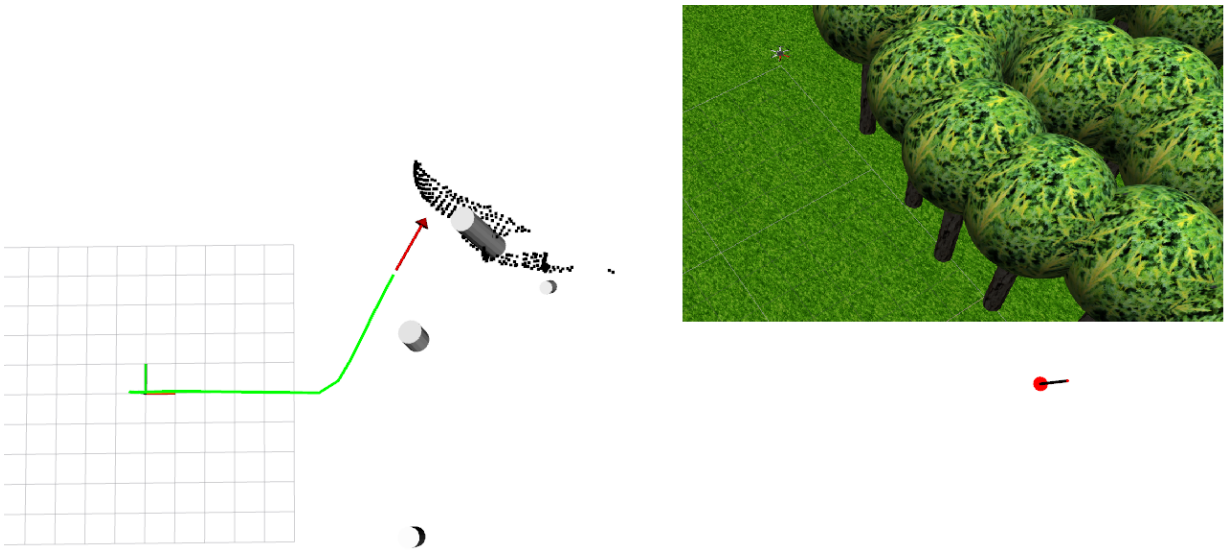


Figure 27: Initial tree detection and choosing flying the most advantageous direction.

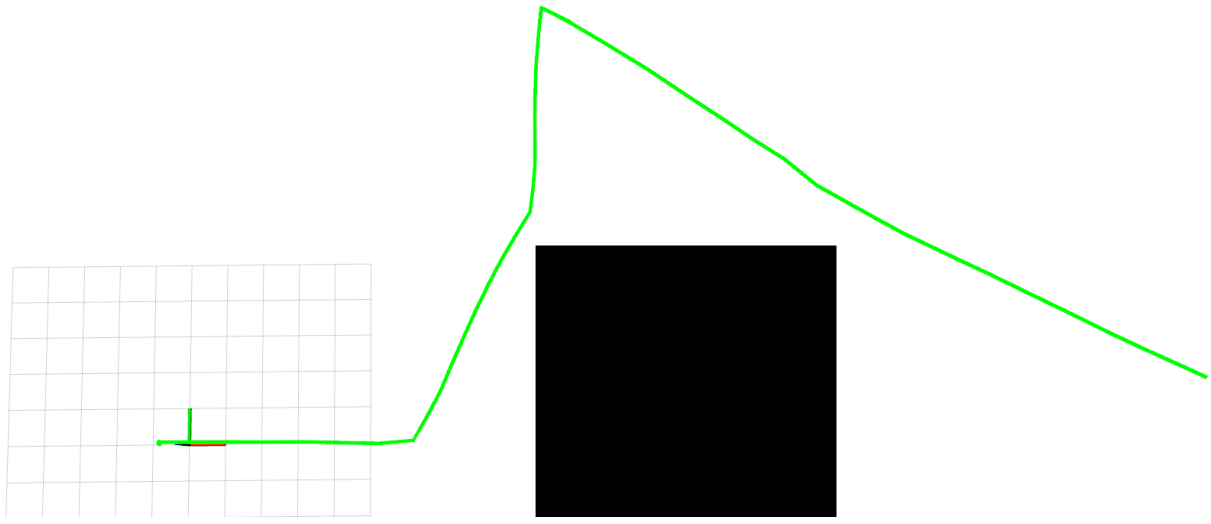


Figure 28: The final trajectory of UAV during simulation in the complicated environment (Position of obstacles is highlighted by the black rectangle.).

7 Real-world experiments

Every program designed for the autonomous control of the system should be tested in practice before it is normally used. Even though the program is tested in the simulator and has fail-safe checks, during real-world deployment new errors which were not exposed in the simulator can appear. A set of three experiments was selected to ensure that the proposed method works properly. For this experiments, was chosen a remote place, where a probability of contact with the rest of the world is minimized (see Figure 29). For safety reasons, the UAV was controlled by the safety pilot, who was responsible for interfering with unpredictable UAV behavior to prevent the collision.

Figure 29: Terrain for the real-world experiment.

UAV localization is not a part of this thesis and the UAV platform used for evaluation of the proposed method is not ready for flying without the Global Navigation Satellite System (GNSS), the system was tested by deployment in the simulated forest-like environment under GNSS. For tree simulations, cylindrical obstructions have been used. A video attachment to this thesis is available at website `http://mrs.felk.cvut.cz/novotny2018thesis`.

7.1 Experimental verification of trajectory planner

The first experiment aims to test trajectory planner functionality. The map is for this experiment without any obstacles in surroundings. The UAV is set to fly into the target position (see Figure 30).

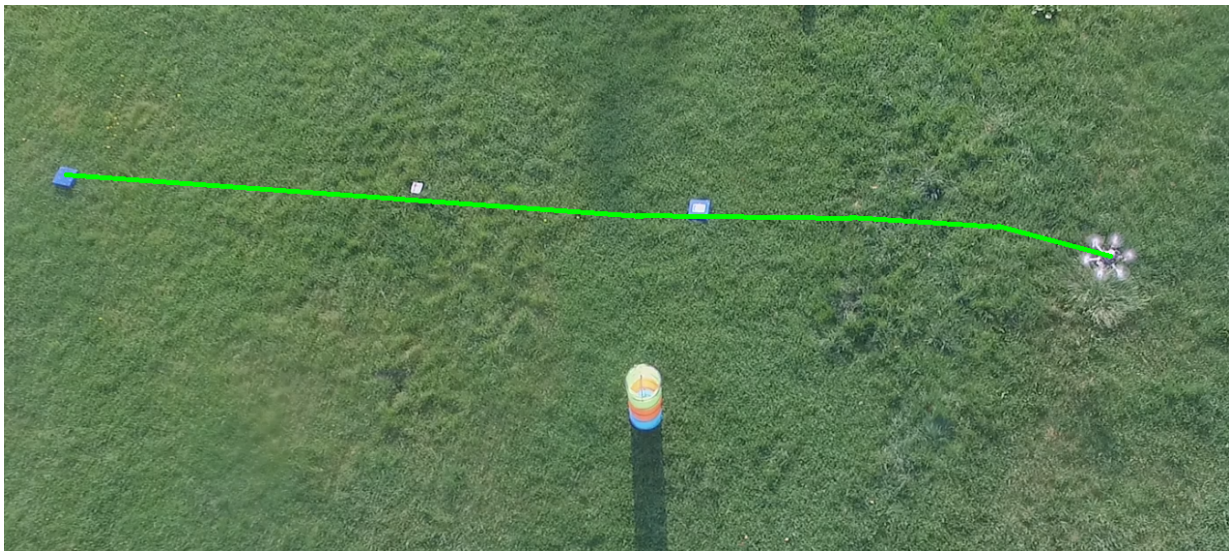


Figure 30: The trajectory of the UAV during experimental verification of trajectory planner.

7.2 Experimental verification of the proposed reactive collision-free technique

The second experiment aims to test evasive maneuvers in the simplest case where there is only one tree in the neighborhood. The target position is in this case on a collision course with the obstacle and the UAV must apply an evasive maneuver to avoid the obstacle and reach the target position (see Figure 31a). In the second part of the experiment, the UAV should return to the initial position. However, the UAV is in start position inappropriately rotated and the obstacle is in immediate proximity. Therefore the UAV has less time for application evasive maneuver but still, there wes no collision with the obstacle (see Figure 31b).

(a) Testing of the collision-free technique for the longer distance.

(b) Testing of the collision-free technique from immediate proximity.

Figure 31: The trajectory of UAV during experimental verification of reactive collision-free technique (Obstacles are approximately marked by black circles.).

7.3 Experimental verification in a more complicated surroundings

In the last experiment, the UAV aims to test evasive maneuvers in a more complicated environment where it has to deal with several obstacles. In this experiment, UAV has to fly through the formation of obstacles shown in Figure 29. Initially, the UAV has to handle with two obstacles placed closed to each other which has to test recognition ability and finally there is an obstacle in the straight line distance to the goal which has to test the ability of quick adapting to new environment. The result of this experiment is shown in Figure 32. All of three experiment was successful and thus we have verified the proper function of the proposed method in real-world experiments.

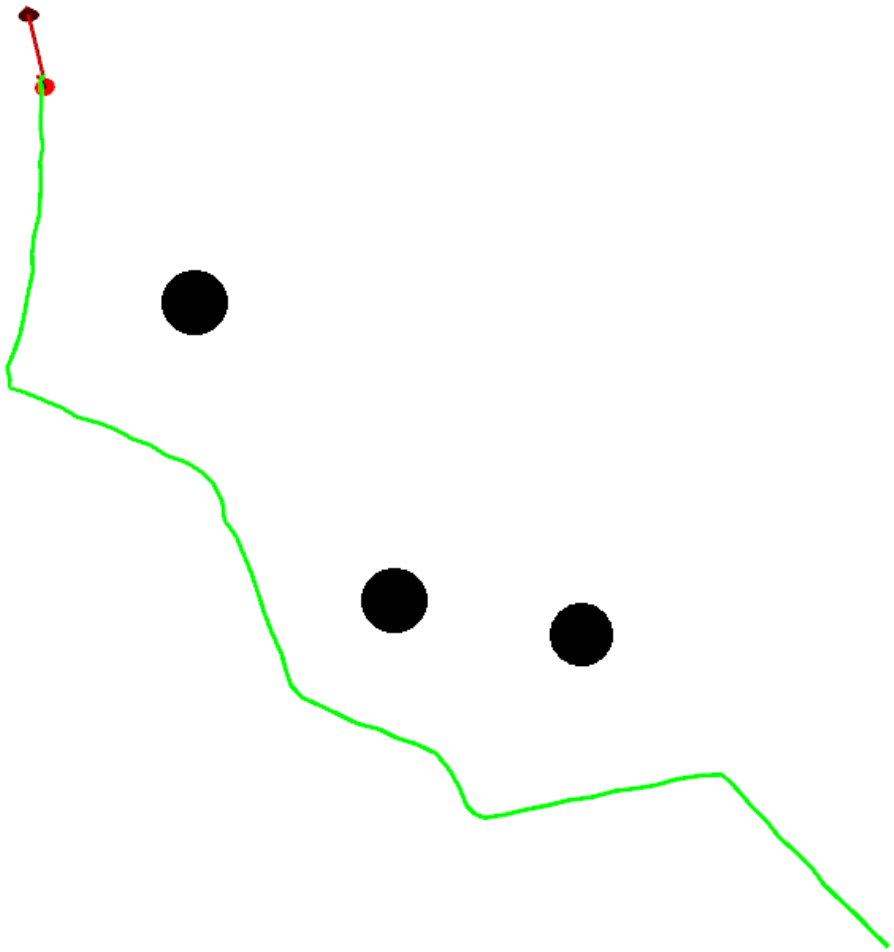


Figure 32: The final trajectory of UAV in the experiment with many obstacles (Obstacles are approximately marked by black circles.).

8 Conclusion

This thesis deals with reactive collision-free planning for the UAV equipped with the depth camera in the forest-like environment.

The first subtask of this thesis was to implement methods allowing the reconstruction of objects from the input cloud of points. This is solved in chapter 2. For that, at first, it is necessary to transform a point cloud into the correct frame, that brings information about the points relative to the UAV. Furthermore, it is also convenient to simplify the point cloud and apply the filters so that the reconstruction is not affected by the external influences and does not evaluate false conclusions. Two methods were proposed for reconstructing of obstacles one dealing with the reconstruction of the point cloud by triangulation, and one that uses the assumption of knowledge about the shape of the obstacles and reconstructs them as cylinders. In practice, the approximation by cylinders is faster, more accurate and less sensitive to noise than triangulation in the forest-like environment (see results in Table 1). Therefore this method was selected for reconstructing obstacles.

The next subtask was to create a local map of detected objects. This map is implemented in the form of a polar histogram that is convenient for the following subtask, that is the reactive collision-free technique for flight trajectory planning based on the VFH algorithm.

The next point was to integrate the proposed technique into the ROS system designed by the MRS group at CTU in Prague [19] [20] [21]. This point enabled real communication between the UAV and the proposed technique.

For safe testing, it was necessary to create a stereo camera model in a Gazebo robotic simulator (explained in chapter 5) and to do the set of simulations that proof the functionality of the complete system extended by proposed collision-free planning method (explained in chapter 6).

The last subtask of this thesis was to experimentally verify the functionality of the complete system by real-world deployment. The results from the set of three experiments have been presented in chapter 7, all three have been successful and did not lead to a collision with the obstacles.

References

[1] N. Wen, X. S. L. Zhao, , and P. Ma, "Uav online path planning algorithm in a low altitude dangerous environment," *IEEE/CAA JOURNAL OF AUTOMATICA SINICA*, vol. 2, no. 2, pp. 173–185, 2015.

[2] C. A. Thiels, J. M. Aho, S. P. Zietlow, and M. D. H. Jenkins, "Use of unmanned aerial vehicles for medical product transport," *Air Medical Journal*, vol. 34, no. 2, pp. 104–108, 2015.

[3] L. A. Haidari, S. T. Brown, M. Ferguson, E. Bancroft, M. Spiker, A. Wilcox, R. Ambikapathi, V. Sampath, D. L. Connor, and B. Y. Lee, "The economic and operational value of using drones to transport vaccines," *Vaccine*, vol. 34, pp. 4032–4067, 2016.

[4] Otiede, Odeyemi, Nwaguru, Onuoha, Awal, Ajibola, Adebayo, Itoro, Ejejigbe, Ejofodomi, and Ofualagba, "Delivering medical products to quarantined regions using unmanned aerial vehicles," *Journal of Applied Mechanical Engineering*, vol. 6, no. 1, pp. 1–4, 2017.

[5] S. Siebert and J. Teizer, "Mobile 3d mapping for surveying earthwork projects using an unmanned aerial vehicle (uav) system," vol. 41, pp. 1–14, 2014.

[6] Y. Lyu, Q. Pan, Y. Zhang, C. Zhao, H. Zhu, T. Tang, and L. Liu, "Simultaneously multi-uav mapping and control with visual servoing," *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pp. 125–131, 2015.

[7] A. M. Samad, N. Kamarulzaman, M. A. Hamdani, T. A. Mastor, and K. A. Hashim, "The potential of unmanned aerial vehicle (uav) for civilian and mapping application," *System Engineering and Technology (ICSET), 2013 IEEE 3rd International Conference on*, pp. 313–318, 2013.

[8] C. E. Schwartz, T. G. Bryant, J. H. Cosgrove, G. B. Morse, and J. K. Noonan, "A radar for unmanned air vehicles," *The Lincoln Laboratory Journal*, vol. 3, no. 1, pp. 119–143, 1990.

[9] T. Agawal, "All you know about lidar systems and applications," *IEEE Transactions on*, vol. 3, no. 1, pp. 119–143, 2015.

[10] C. Luo, G. E. Jan, J. Zhang, and F. Shen, "Boundary aware navigation and mapping for a mobile automaton," *IEEE*, 2017.

[11] "Intel®realsense™ camera r200," 2016, available at https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/realsense-camera-r200-datasheet.pdf.

[12] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[13] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," *ICRA workshop on open source software.*, vol. 3, no. 3, pp. 1–5, 2009.

[14] S. Orts-Escolano, V. Morell, J. García-Rodríguez, and M. Cazorla, "Point cloud data filtering and downsampling using growing neural gas," *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2013.

[15] P. Maur, "Delaunay triangulation in 3d," *Doctoral Thesis on University of West Bohemia in Pilsen*, 2002.

[16] Lindsay and Smith, "A tutorial on principal components analysis," 2002, available at http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf.

[17] H. Wang, L. Wang, J. Li, and L. Pan, "A vector polar histogram method based obstacle avoidance planning for auv," *IEEE Transactions on*, pp. 1–5, 2013.

[18] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Cambridge University Press*, vol. 33, pp. 463–497, 2014.

[19] G. Loianno, V. Spurny, T. J, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Thou, A. Cho, M. Saska, and V. Kumar, "Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert-like environments," *IEEE Transactions on*, vol. 3, no. 3, pp. 1576–1583, 2018.

[20] T. Baca, P. Stepan, and M. Saska, "Autonomous landing on a moving car with unmanned aerial vehicle avoidance for mobile robots," *IEEE Transactions on*, pp. 1–6, 2017.

[21] V. Spurný, T. Báča, M. Saska, R. Pěnička, T. Krajník, G. Loianno, J. Thomas, D. Thakur, and V. Kumar, "Cooperative autonomous search, grasping and delivering in a treasure hunt scenario by a team of uavs," 2017, (submitted to the special issue on "MBZIRC 2017 - Challenges in Autonomous Field Robotics").

Appendix A DVD Content

In Table 2 are listed names of all root directories on CD.

Directory name	**Description**
code	implemented program for UAV
doc	this thesis in LaTeX and pdf
cam	videos from experiment and simulation

Table 2: DVD Content

Appendix B List of abbreviations

In Table 3 are listed abbreviations used in this thesis.

Abbreviation	Meaning
FOV	field of view
FPS	frames per second
GNSS	global navigation satellite system
IMU	inertial measurement unit
LIDAR	light detection and ranging
MAV	micro aerial vehicle
MPC	model predictive control
PCA	principal component analysis
PCL	point cloud library
RADAR	radio detection and ranging
ROS	robot operating system
SDF	simulation description format
UAV	unmanned aerial vehicle
VFH	vector field histogram
Xacro	XML macro
XML	extensible markup language

Table 3: Lists of abbreviations