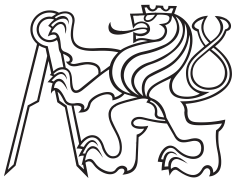
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Collision-free Navigation System for Robotic Helicopter

Mozgunova Anastasiia

Supervisor: Ing. Jan Chudoba
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
May 2021

Acknowledgements

I would like to thank my supervisor Ing. Jan Chudoba for the useful comments and remarks. I would like to thank my family for their incredible support during my entire studies.

Declaration

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 20. May 2021

.....

Signature

Abstract

This work is focused on the creation of a collision-free navigation system for a robotic helicopter. During this work the matematical model of the quadracopter is derived and linearized. The regulator for the UAV is designed based on this model. The solution for the localization problem is provided in the form of Kalman filter. Space-efficient octree structure is proposed to store robot configuration space and A* algorithm is used for navigation in this environment. The implementation of the proposed algorithms is done in programming language C++ and tested in simulation environment Webots.

Keywords: UAV, colision-free system, A*, octree, path planning

Supervisor: Ing. Jan Chudoba
CIIRC,
Jugoslávských partyzánů 3,
Praha 6

Abstrakt

Tato práce je zaměřena na vytvoření bezkolízního navigačního systému pro robotickou helikopteru. Během této práce je odvozen a linearizován matematický model kvadrakoptéry. Regulátor pro UAV je navržen na základě tohoto modelu. Řešení problému s lokalizací je poskytnuto ve formě Kalmanova filtru. Pro uložení konfiguračního prostoru robota bude navržena prostorově efektivní struktura octree a pro navigaci v tomto prostředí je použit algoritmus A*. Implementace navržených algoritmů je provedena v programovacím jazyce C ++ a testována v simulačním prostředí Webots.

Klíčová slova: UAV, bezkolizní system, A*, octree, plánování cesty

Překlad názvu: Bezkolizní navigační systém pro robotickou helikoptéru

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Problem overview

The field of autonomous navigation is developing rapidly. Small and maneuverable drones could be helpful in many fields, including rescue missions, medicine, communications, transport, agriculture and many others. In order to enable vehicle to fly autonomously stabilization and navigation algorithms need to be developed.

1.2 Main goals

The aim of the work is a research and design of a collision-free navigation system for an autonomous robotic helicopter in an unknown indoor environment. The robot should find the a path from the current position to the destination without colliding any obstacles. This path needs to be time and cost-efficient. In order to solve the autonomous navigation problem, the following subproblems need to be solved: localization, environment re-construction, and collision-free path planning.

1.3 Thesis outline

The testing of the designed methods was done in simulator Webots with quadracopter Dji Mavic Pro. First, the controller for the UAV needs to be designed because it is not provided by the simulator. Webots API allows to control quadracopter by setting the angular speed of propellers. The mathematical model of the system and design of the controller will be provided

in section 3. Next step is to build navigation system. Sections 3-5 will be dedicated to the description of that system. Finally, section 7 provides implementation details and presents some results.

Chapter 2

Sensors

Sensors help UAV to gain an information about its current state and the environment. Global Positioning System (GPS) and the Inertial Measurement Unit(IMU) sensors can be used to measure the current position and the rotation of the UAV. Whereas, sensor such as lidars and cameras help UAV gain information about its environment and make decisions based on the data. In the next section the description of the most popular sensors for UAV localization will be provided.

2.1 GPS

GPS, which stand for global positioning system, is a satellite navigation system that provides an information about distance, time and position in WGS 84 coordinate system that defines coordinates related to the center of Earth. The main purpose of this system is to determine the current location by measuring the time of flight of the synchronised signal from the satellites to the receiver. The accuracy can vary from several millimeters up to several meters.

2.2 IMU

IMU, which stand for inertial measurement unit, consists of one or more accelerometers and one or more gyroscopes. The magnetometer is often used as a part of the IMU. The accelerometer can measure acceleration vectors by measuring the amount of static acceleration due to gravity. Using the acceleration the direction of the gravity force vector is found. The magnetometer measures magnetic field. By combining vector of magnetic field and vector of gravity force, the rotation of the UAV can be obtained.

Gyroscope is a device that can react to the changes in rotation. Measurement of the angles from the gyroscope should be combined with the measurements from the accelerometer and magnetometers due to gyroscope drift caused by integration of inherent imperfections and noise within the device.

2.3 Lidar

Lidar, which stands for Light Detection and Ranging, is a device that measures information about remote objects using an active optical systems. Lidars use an infrared LED or a laser light and provides measurements based on the received reflected light. Light can be diffused in any environment, therefore lidar is a universal device for detecting objects. There are two types lidars based on principles of signal processing.

The first type of lidars measures the displacement in the phase between emitted and received signal. The advantages of this type of lidar includes high precision of distance measurement. One of the disadvantages is the low throughput.

The second type of lidars measures the Time-Of-Flight(TOF) of the light ray that was reflected from the object. This type of lidar allows to measure objects that are located far away from the device.

Key components of lidar are the light emitter and the light receiver. The emitter directs the light ray to the mirror that is rotating as a part of the housing of the lidar. After the ray is reflected, it returns to the mirror and then to the receiver, which is capable of calculating the signal distance.

2.4 Camera

The measurement of distance can be obtained from cameras by using the triangulation process. These cameras can take a picture of the same point in space from different angles. The location of this object can be obtained from the knowledge of the point projections and position of cameras.

Additionally, a RGB-D camera can be used for determining the position of an object by using structured light that is projected to the scene by a projector. The sensor(a camera) captures the light, and the position can be determined with the help of triangulation. The processing of data from the camera requires lots of computation power. In this work cameras will not be used. In future works the system can be extended with this option in order to gain more precise information about the environment.

Chapter 3

Mathematical model

In order to stabilize the UAV in the air, regulator needs to be designed. This regulator should be based on mathematical model, derivation of which will be provided in this section.

3.1 Coordinate systems

The coordinate systems is defined as following:

- Earth fixed frame or E-frame, where x and y axis forms a plane parallel to the ground and z axis is perpendicular to it.
- Body fixed frame or B-frame with the origin in helicopter's center of gravity.

To transform coordinates from E-frame to B-frame the following steps will be be applied:

1. Rotate system around z axis at an angle ψ. This rotation is called **yaw**.
2. Rotate the result of (1) around y axis at an angle θ. This rotation is called **pitch**.
3. Rotate the result of (2) around x axis at an angle ϕ. This rotation is called **roll**.

These transformation can be written in the form of matrices:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.1}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.2}$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$R(\phi, \theta, \psi) = R_z(\psi) R_y(\theta) R_x(\phi) \tag{3.4}$$

$$R(\phi, \theta, \psi) = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\cos\psi + \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{3.5}$$

3.2 Non-linear model

Let the drone velocity in the B-frame system be v_B and in the E-frame v. With this information the following equation can be used to translate vector of velocity from one coordinate system to an another:

$$v = R(\phi, \theta, \psi) v_B \tag{3.6}$$

Let drone angular velocity in the B-frame system be $w_B = [p, q, r]$ and in the E-frame $w = [\dot{\phi}, \dot{\theta}, \dot{\psi}]$. Angular velocity transformation is given by:

$$w = T(\phi, \theta) w_B \tag{3.7}$$

$$T(\phi, \theta) = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{3.8}$$

Newton–Euler equations:

$$F_E = m\dot{v} \tag{3.9}$$

$$\tau = I \cdot \dot{w_B} + w_B \times (I \cdot w_B) \tag{3.10}$$

where $\tau = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}$ is a torque and I is inertia matrix, which is diagonal because the rotation center coincides with the center of mass.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{3.11}$$

Forces in E-frame can be described by the following equation:

$$F = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ F_T \end{bmatrix} \tag{3.12}$$

where F_T is a total propellers thrust.

From 3.8 and 3.12 we obtain the following equations:

$$\dot{v} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} \cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi \\ \cos\phi \sin\theta \cos\psi - \sin\phi \cos\psi \\ \cos\phi \cos\theta \end{bmatrix} F_T \tag{3.13}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = T(\phi, \theta) w_B^T \tag{3.14}$$

$$\dot{w_B} = I^{-1}(\tau - w_B \times (I \cdot w_B)) \tag{3.15}$$

3.3 Model linearization

During this work, methods that require linear model will be used, therefore the model needs to be linearized. Linearization of the model will be done with an assumption that the system will be staying near the chosen equilibrium:

$$\mathtt{x} = \begin{bmatrix} x_e & y_e & z_e & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$u = \begin{bmatrix} mg & 0 & 0 & 0 \end{bmatrix}$$

For small angles an approximation can be used:

$$\sin x \approx x, \cos x \approx 1 \tag{3.16}$$

So that $\cos\phi$ and $\cos\theta$ can be approximated as 1, $\sin\phi$ and $\sin\theta$ can be approximated as ϕ and θ respectively. Considering $\phi \approx 0$ and $F_T \approx mg$ equation 3.13 can be linearized

$$\dot{v} = g \begin{bmatrix} \theta \\ -\phi \\ 0 \end{bmatrix} \tag{3.17}$$

Linearizing equation 3.10:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_x} & 0 & 0 \\ 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & \frac{1}{I_z} \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{3.18}$$

Linearization of equation 3.7 will look like as follows:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.19}$$

The resulting state space model can be represented as:

$$\dot{\mathtt{x}} = Ax + Bu \tag{3.20}$$

where

$$\mathtt{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]^T, u = [F_T, \tau_x, \tau_y, \tau_z]^T$$

$$A = \begin{bmatrix} O & I & O & O \\ O & O & A_1 & O \\ O & O & O & I \\ O & O & O & O \end{bmatrix}$$

where O is 3x3 zero matrix, I is 3x3 identity matrix,

$$A_1 = \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}$$

3.4 Controller

Controller for the UAV stabilization is going to be implemented as the following. The input controller will receive the desired position $x_{des}, y_{des}, z_{des}$ and yaw angle ψ_{des}.

3.4.1 Position controller

The position PID controller represents an outer loop of the regulator. The controller tries to exponentially minimize an error in the current UAV altitude values and the position in x-y plane.

$$F_T = m(K_{pz}e_z + K_{iz}\int e_z + K_{dz}\dot{e}_z + g) \tag{3.21}$$

where $e_z = z_{des} - z$.
For the x and y coordinates desired accelerations will be calculated:

$$a_x = K_{px}e_x + K_{ix}\int e_x + K_{dx}\dot{e}_x \tag{3.22}$$

where $e_x = x_{des} - x$.

$$a_y = K_{py}e_y + K_{iy}\int e_y + K_{dy}\dot{e}_y \tag{3.23}$$

where $e_y = y_{des} - y$.
a_x, a_y will be passed to the inner loop of the regulator, F_T will be passed directly to the motor controller.

3.4.2 Altitude controller

The relation between desired acceleration and angles is given as the following:

$$\begin{bmatrix}\phi_{des}\\ \theta_{des}\end{bmatrix} = \frac{1}{g}\begin{bmatrix}\sin\psi_{des} & -\cos\psi_{des}\\ \cos\psi_{des} & \sin psi_{des}\end{bmatrix}\begin{bmatrix}a_x\\ a_y\end{bmatrix} \tag{3.24}$$

a_x and a_y values will be obtained from the outer loop and ψ_{des} value will be given as the controller input. Then the PD controller for the moments calculation will be applied as:

$$\begin{bmatrix}m_x\\ m_y\end{bmatrix} = \begin{bmatrix}K_{p_\phi} & 0\\ 0 & K_{p_\theta}\end{bmatrix}\begin{bmatrix}e_\phi\\ e_\theta\end{bmatrix} + \begin{bmatrix}K_{d_\phi} & 0\\ 0 & K_{d_\theta}\end{bmatrix}\begin{bmatrix}\dot{e}_\phi\\ \dot{e}_\theta\end{bmatrix} \tag{3.25}$$

where $e_\phi = \phi_{des} - \phi, e_\theta = \theta_{des} - \theta$. Moment for ψ angle will be given as following:

$$m_z = K_{p\psi}e_\psi + K_{d\psi}e_\psi, \tag{3.26}$$

where $e_\psi = \psi_{des} - \psi$.
Resulting moments m_x, m_y, m_z will be passed to the motor controller.

3.4.3 Motor controller

The input for this controller is F_T from the outer loop and m_x, m_y, m_z from the inner loop. First, these values will be converted to the motor thrusts:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 0.25 & -c & -c & -c \\ 0.25 & c & c & -c \\ 0.25 & c & -c & c \\ 0.25 & -c & c & -c \end{bmatrix} \begin{bmatrix} F_T \\ m_x \\ m_y \\ m_z \end{bmatrix} \tag{3.27}$$

where c is an arm length, T_1, T_2, T_3, T_4 is the thrust values on the front right, rear left, front left and rear right motors accordingly.
Finally, the thrust values will be converted to angular speed of the motors, which can be passed to Webots API:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = K \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{3.28}$$

where w_{1-4} is an angular velocity values of the corresponding motors, K is the proportional coefficient between the thrust and the angular velocity. This coefficient was determined empirically: trust values of all motors was set to $\frac{mg}{4}$, so that the total thrust is equal to mg, and K was incremented until the quadracopter has risen up.

3.4.4 Implementation

The Matlab simulation results were obtained for the designed controller of the circular trajectory:

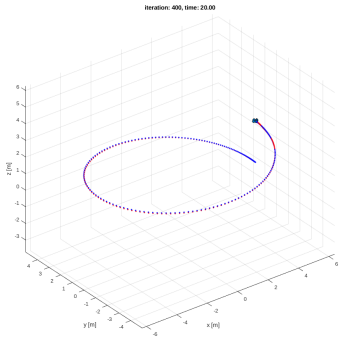


Figure 3.1: Trajectory tracking in Matlab simulation. The blue line represents a desired trajectory and the red line represents a measured trajectory

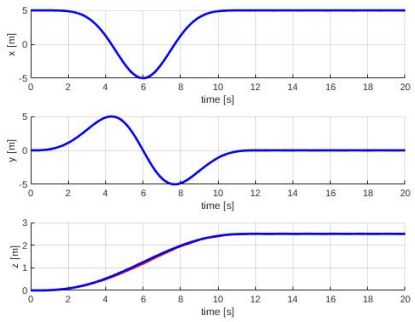
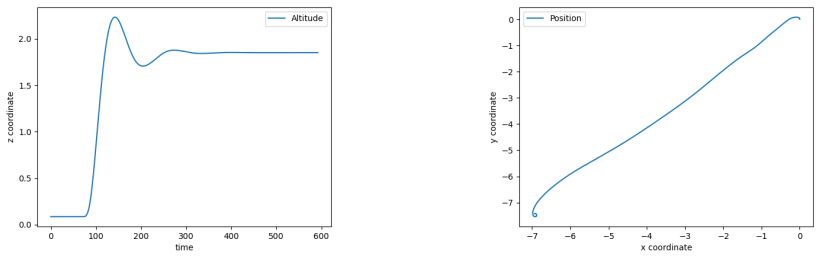


Figure 3.2: Referenced and measured values for the position of the UAV (blue line - desired, red line - measured)

The next step was to evaluate the designed controller in a physics simulator. The calibration of the coefficients were done empirically in a simulator.



(a) : Altitude of UAV **(b) :** Position of the UAV and start position $x = 0, y = 0, z = 0$

Figure 3.3: Results of the simulation of the flight with desired position $x_{des} = 2, y_{des} = -7, z_{des} = -7, \psi = 0$

The results are presented with the following coefficients:
$K_{p_z} = 1.8, K_{i_z} = 0.001, K_{d_z} = 0.9$
$K_{p_x} = 0.6, K_{i_x} = 0.001, K_{d_x} = 0.6$
$K_{p_y} = 0.6, K_{i_y} = 0.001, K_{d_y} = 0.6$
$K_{p_\phi} = 0.06, K_{d_\phi} = 0.012$
$K_{p_\theta} = 0.06, K_{d_\phi} = 0.012$
$K_{p_\psi} = -0.005, K_{d_\psi} = -0.0025$

Chapter 4

Localization

4.1 Problem description

A precise location estimation plays an important role in an autonomous navigation of Unmanned Areal Vehicles. There are many techniques, where some of them are suitable for an indoor navigation, some of them for outdoor navigation, some of them requires additional techniques such as filtering.

4.2 Localization techniques

There are many algorithms which are able to localize the UAV. **Simultaneous Localization and Mapping (SLAM)** algorithm is an algorithm where the position estimation is done by evaluation of the environment simultaneously with mapping process.

Particle filter algorithm is a SLAM method that relies on random sampling of the state. Each sample is assigned a weight value, which represents a probability of that sample to be taken from the probability density function. In the next iteration resampling is done based on these weights. This algorithm was incorporated by [YFZ+18], where the combination of lidar and RGB-D camera measurements were used in order to map the environment. In [Rig12] Particle filter is compared with other filter algorithms that use non-linear model.

Additionally, the Motion capture system, that follows the position of the robot, is used for an indoor navigation in laboratories. In an outdoor environment the information about the current location can be obtained from the **GNSS**. Measurements from the GNSS are not accurate, therefore a signal filtering technique should be introduced.

In simulation environment, the GPS signal error can be adjusted so that the signal will be accurate sufficient in order to localize the UAV accurately. In the laboratory environment, the measurements that are received from the Motion Capture system can also be used together with Kalman filter in order to obtain the current location. The Kalman filter would not be a sufficient solution to use in an outdoor environment due to an error in GNSS measurements that could become relatively large.

4.3 Kalman Filter

The Kalman filter is a recursive filter that estimates the state of the system using calculated predictions and measurements. It allows to decrease sensor measurement errors caused by noise. The Kalman filter algorithm consist of the following two phases:

1. Predict
   During this phase, the a priori state of the system is predicted based on a linear model of the system. Additionally, the a priori estimate covariance matrix is predicted.
   Prediction of the state using a state space model:

$$x_k = F\hat{x}_{k-1} + B_d u_k + w \tag{4.1}$$

   where x_k is predicted(a priori) state in the current iteration, $\hat{x}_{k-1}$ is a posteriori state in the previous iteration, w is a process noise that is caused by model approximation errors.

   Prediction of the estimate covariance matrix P_{k-1}:

$$P_k = AP_{k-1}A^T + Q \tag{4.2}$$

   where P_{k-1} is an estimate covariance matrix in the previous iteration and Q is the process noise.

2. Update
   During this phase new observations from the sensors will be obtained and processed. Optimal Kalman gain K_k calculation:

$$K_k = P_{k-1}H^T(HP_{k-1}H^T + R)^{-1} \tag{4.3}$$

   where R is a covariance matrix of noise in measurements, H is an observation model.
   Measurements processing:

$$y_k = Cy_{k_m} + Z_k \tag{4.4}$$

Update of the (a posteriori) state estimate:

$$\hat{x}_k = x_k + K(y_k - Hx_k) \tag{4.5}$$

Update estimate covariance matrix:

$$P_k = (I - K_k H)P_k \tag{4.6}$$

From 4.5 Kalman filter can be represented as an interpolation between predicted and measured state:

$$\hat{x}_k = (I - K_k H)x_k + K_k y_k \tag{4.7}$$

By changing the Kalman gain the contribution of the predicted and estimated states in resulting state estimate can be controlled.

Chapter 5

Space representation

Robot path planning determines a path which the robot should follow in order to reach its goal without colliding with any obstacles. The environment of the robot's configuration space will be build in order to determine collision-free path.

5.1 Related Work

There are numerous proposals on how to represent the configuration space for the robot. **Uniform grid map** is a representation of the space, where each cube, or a voxel, has an equal size. The advantages of using this approach is a high speed of search queries. However, this approach also has disadvantages. Firstly, the entire space must be represented with the same number of voxels. Large amount of redundant information has to be stored if the obstacles are infrequent. Secondly, searching algorithm will be slowed down by the large amount of traversed nodes. Space usage for 3D space of uniform grid maps can be decreased by 2.5D grid maps([NSC17]). In this maps, occupation information is represented as a starting point of unoccupied height of cells in the x-y plane. This approach causes problems in environment where some obstacles are located above the robot.

Non-uniform grid maps are more efficient way to store information about the environment. This methods are based on adaptive and exact cell decomposition. In case of exact cell decomposition the large cells is divided into cells with predefined size, whereas in adaptive one the size of the smaller cells can vary. Quadratic tree layout for 2d space and octree for 3d space is a frequently used example of exact cell decomposition. In this layout the density of cells is higher near the obstacles and sparse in free spaces. This approach also has some disadvantages. It is going to be harder to combine results from multiple sensors. Moreover, the time of the search for one node

will be increased.

In [CN06] distinct measurements from sensors were used to store information about the environment. The main drawback of this method is that the information about both free and unknown space could not be retrieved.

5.2 Octree

Octree representation belongs to a non-uniform grid map representation. The map consists of voxels of different sizes, where each voxel can be subdivided into eight smaller voxels of an the equal size. Each voxel stores information about its occupancy. Traditional space representation

[HWB+13] describes the probabilistic occupancy model based on octrees. Instead of storing boolean value indicating occupancy of the cell, the Voxel is considered as occupied when probability is greater than the predefined threshold. Due to the fact that data is stored as a tree, the time complexity of queries compared to uniform mapping will increase from $O(1)$ to $O(log(n))$, where n is a number of voxels. Each voxel is represented as the following:

```
struct Voxel{
    Voxel **children
    float occupancy_probability
}
```

where children is a pointer to an array of the sorted pointers to eight children voxels.

The depth of the current child, size and center can be retrieved from the parent values during the tree traversal.

When a new data(hit or miss) arrives from sensors, the tree expands all levels from the root to the lowest leaf and updates its log-odds value according to the following equation:

$$L(n) = L(n-1) + \frac{P(n)}{1 - P(n)} \tag{5.1}$$

where $L(n)$ is the current log-odds value, $L(n-1)$ is the previous log-oggs value and $P(n)$ is a prior probability of occupied/free node respectively. Finally, the occupancy probability can be calculated with the following equation:

$$P(occ) = 1 - \frac{1}{1 + \exp(L(n))} \tag{5.2}$$

Parent log-odds value is then set to the maximum value of children's log-odds values.

Storing the occupation probability reduces errors introduced by the noise in the lidar measurements and helps to deal with dynamic obstacles. In order to keep the size of the tree smaller, compression technique was introduced: for the log odds value two thresholds were introduced: l_min and l_max. If log odds value of the voxel is higher than l_max or lower than l_min it is considered to be stable. If all children voxels of the parent voxel are stable, then the children can be pruned.

5.3 Processing obstacle information

5.3.1 Limitations

With one layer lidar, only one layer of space can be scanned in one iteration. That means that the other layers of space remains unknown for the UAV. This results in the robot being unable to find the path even if a valid path exists. During flight UAV tilts at different pitch and roll angles, that allows to obtain information from other layers of the space. Because of that path finding algorithm will have enough information to construct the path with first several steps without collisions. Re-planning will be made during all flight and because of that UAV will be able to quickly react on the changes of the environment if some new obstacles will be found. Moreover, proposed path planning algorithm tends to find optimal path. It means that if the feasible path exists in the current layer of the space, planner will find it.

5.3.2 Data processing

Simulation environment Webots provides an access to lidar measurements as an array of distances where light hits an obstacle. If there was no obstacle within the scanning range, those distance would be the maximum lidar detection distance possible.

Assuming measured distance is returned as a distance from the UAV to the obstacle, this measurement will be first be rotated and translated according to the following equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R(\phi, \theta, \psi) \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \tag{5.3}$$

In order to incorporate new obstacle information into the environment map, Digital differential analyzer algorithm for 3D space([AW87]) is used. All voxels that the light intersects from the current position of the UAV, the

lidar measurement will be updated with a new information about free space. If the measurement is within the bounding box of the distance limit of the lidar, the last voxel that the ray intersects will be updated with a hit.

Chapter 6

Path planning

6.1 Problem description

For collision-free path planning, environment space should be represented as a discrete information about the occupancy of certain place. This space is then processed by the path finding algorithm, that will be able to generate a safe path.

6.2 Related works

Being a topic of great interest over the past years, there was an extensive research done in the field of path planning.

Probabilistic roadmaps(PRM)
The key idea of this algorithm is to connect randomly generated samples of states together to form a graph in such way so the edges of this graph does not intersect with obstacles. In order to fin a path from start vertex to goal vertex, Dijkstra algorithm can be applied. Particular representation of this approach is Voronoi graph([CB00], [SAC+08]).

Rapidly-exploring random tree(RRT)
The same as PRM, this approach works by generating random samples and connecting them together into a graph. The algorithm starts by placing first node in the start point, then every iteration it creates a new vertex in a random place. Next it checks if the new vertex lies outside any obstacles and connects it to the nearest existing vertex if no obstacles are lying inside the path between them. The algorithm finishes when it finds an obstacle close to the goal point. RRT is able to find some path, that may not be an optimal path. The optimized version of RRT is RRT* that will generate shortest path

to the goal within all generated vertexes. After finding a new vertex, the algorithm does not connect it to the nearest neighbour, instead it reconnects all nodes in the tree within the specified search radius in a way that maintains a tree structure and also minimizes the total path length. In [WZSM15] RRT algorithm was proposed as a solution of the problem for a feasible and safe path between UAVs and RRT* to optimize the path. Additionally, RRT was incorporated in [YJZ18] as a solution for the path planning problem.

Graph search methods
This group of methods is frequently used in various works. This group includes algorithms such as BFS, DFS, Dijkstra algorithm, A* and its modifications such as generalized adaptive A*, A*-lite, D*. These graph algorithms generate optimal path. For example, in [PGC18] author used bidirectional A* algorithm to find admissible path and convert this path into B-spline curve. In the work [YLX13] A* was used in combination with PRM.

Neural networks
Neural networks nowadays are used in many different fields. The concept of neural networks comes from the the principle by which the brain works. A signal(number) is transmitted from the input layer of neurons to the output layer of neurons based on connections that were build during the training process. According to the universal approximation theorem, neural network can approximate any continuous function with any precision. In path finding problem, they was incorporated by [XY18].

Genetic algorithms
This group of algorithms were inspired by natural processes. The algorithm first starts by generating random properties, followed by operations of crossing-over, mutations, and population selection each iteration. The population evolves by each iteration. This approach is presented in work [LPMM18], where properties were the trajectory of UAVs.

Markov decision process(MDP)
This method s used to solve stochastic, sequential problems. MDP is consist of state spaces, action spaces, transition probability and cost function. Every transition from one state to another have its cost or reward. The solution for this task an optimal policy, or the optimal action for the certain state, which is found by maximizing the total reward. This method is used in [YZZ17]. A generalization of MDP is POMDP(partially observable Markov decision process). In POMDP instead of storing state the probability distribution of different observations in state is stored. This method was used in [ECM17].

Model predictive control(MPC)
The goal of this method is to optimize function based on some chosen constraints. In the case of the UAV autonomous flight, this constrains can consist of collision constraints, as well as constrains on velocity and acceleration([MNG15]).

6.3 A* algorithm

At the first subsection of this section A* algorithm will be described, the second subsection will be dedicated to choice of heuristics functions and the following subsections will be focused on solving issues in using octree representation in A* algorithm.

6.3.1 Algorithm description

A* algorithm an algorithm for finding an optimal path. It aims to find the path, that minimizes the cost function

$$c(x) = g(x) + h(x) \tag{6.1}$$

where x is the current vertex, $g(x)$ is a distance to the current vertex from the start vertex and $h(x)$ is a heuristic function. Heuristic function is consistent when it satisfies the equation $h(x) \geq d(x, y) + h(y)$, where $d(x, y)$ is the distance between vertices x and y (or the length of the edge) for every edge of the graph. A priority queue is used for retrieving the vertex with the lowest cost with a time complexity $O(1)$ and sorting with time complexity of $O(log(n))$.

6.3.2 Heuristic function

It is important to choose an appropriate heuristic function in order to achieve high speed algorithm. For this work diagonal distance heuristic was used, where the distance to the destination is calculated assuming that object can perform diagonal translation. For the 3d space the equation is the following:

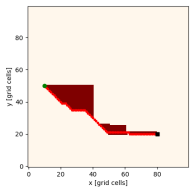
$$d = D(\Delta x + \Delta y) + (D_2 - 2D)min(\Delta x, \Delta y) \tag{6.2}$$

where Δx and Δy are the absolute distances between current cell and the goal. $D = 1$ and $D_2 = \sqrt{2}$ will be used. This distance function is called an octile distance. In situations where diagonal moves are possible Euclidean distance can also be used as a heuristic function. The equation for that function is the following:

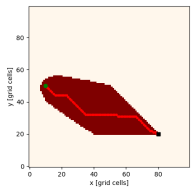
$$d = \sqrt{x^2 + y^2 + z^2} \tag{6.3}$$

In this case A* will expand more cells due to that fact that decrease in heuristic will not be proportional with the increase in the distance value of $g(x)$. This result is also shown in the 6.1, where the filled area represents

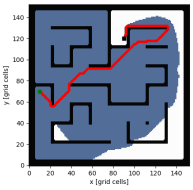
the expanded cells. Decrease in the number of expanded cells is especially noticeable in empty space, which is often the case for the natural environment. When there are many optimal paths in the map, A* will tend to explore all of them, because cost function will return the same results for all this paths. The solution to overcome this problem is to slightly increase the value of the heuristic function. Increase in 0.1% will be enough to solve this issue.



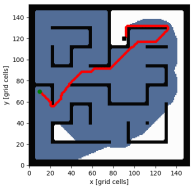
(a) : Euclidian distance in empty space. Number of expanded nodes: 2343.



(b) : Octile distance in empty space. Number of expanded nodes: 545.



(c) : Euclidian distance in space with obstacles. Number of expanded nodes: 7680.



(d) : Octile distance in space with obstacles. Number of expanded nodes: 7023.

Figure 6.1: Comparison of the expanded area of A* algorithm (a, c) with the octile distance heuristic function (b, d) with the Euclidean distance heuristic function

6.3.3 Neighbour finding strategy in the octree

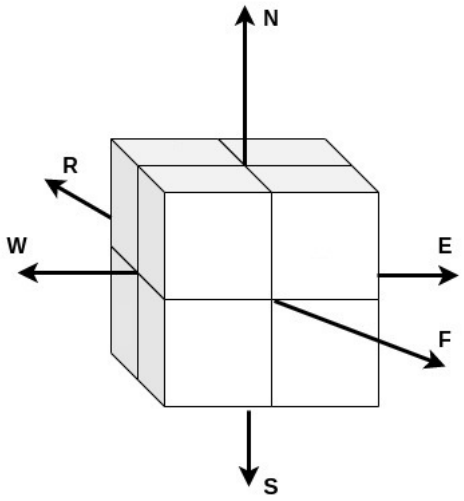
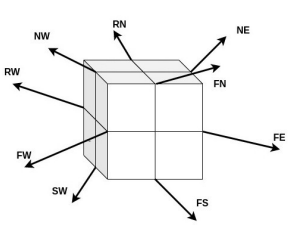
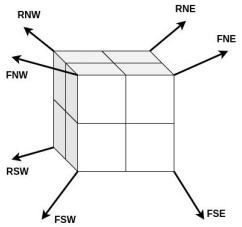


Figure 6.2: Face directions



(a) : Edge directions



(b) : Vertex directions

Some strategies were developed to find neighbours in the octree structure. One of the options is to traverse the entire tree and find cells that are sharing a common face, edge or vertex with the current cell. Also backtracking strategy can be used for neighbour searching that involves tree traversal through the octree from the current node.

Works [Vö00] and [KL09] present the relationship between neighbour voxels based on unique voxel number. This number is called a location code and it is assigned for each voxel based on the parent number and a position relative to the parent. The whole octree is then stored inside a hash map with a location code as a key and voxel information, such as the occupancy, as a value. The neighbours of the same size can be retrieved by searching its location codes inside lookup table. If the neighbour does not exist, the code of the neighbour is then adjusted towards the root of the tree until the code of an existing voxel will be reached. If the neighbour exists, the search for its children in appropriate direction will be executed. A voxel will be considered a neighbour if that voxel does not have children in the search direction. The

advantages of this method is a highly optimal search speed and reduced memory consumption for storing the octree structure.

In [RVZ16] authors present the possibility of finding neighbours during the octree construction and store them in a network graph. This method does not require an additional time for the neighbour finding during the search, this leads to time reduction during path finding process. Although, it is not suitable for hardware that has strict memory limitations.

If the size and the position of the voxel is known, then the query for the neighbour can be performed. For each of the faces, edges and vertexes of the voxel for the neighbour will be found by searching in the tree coordinates of the center of the current voxel with an offset equals to the size of the current voxel in appropriate coordinates. This search will be limited to the same size and larger voxels only. These neighbour voxels are checked for children in the direction opposite to the direction of the neighbour voxel.

For example, the voxel A for which the neighbours should be found, is located in depth 14, it has a size of 1 meter in all directions and a center at coordinates $(0, 0, 0)$. The neighbour B that lies in the northern direction (marked as N in 6.2) and shares a common face with A and is located at the same depth should have center at coordinate $(0, 0, 1)$. The voxel can be found by traversing the tree from the root node. If this voxel will not be presented in the tree, then its parent(voxel in lower depth) would be found by the search algorithm, and set as B. If B would be the same size as A(located at the same depth), the algorithm will check for children in the appropriate direction will be executed: in case of neighbour in the northern direction, there will be all children voxels that share the southern face(marked as S in 6.2) with the voxel B. That procedure will be done recursively for all found children. Neighbour of A would be the last voxels that do not have children in the southern direction.
The time complexity of search for one neighbour is then $O(d)$, where d is the depth of the tree.

6.3.4 Dealing with unexpanded voxels

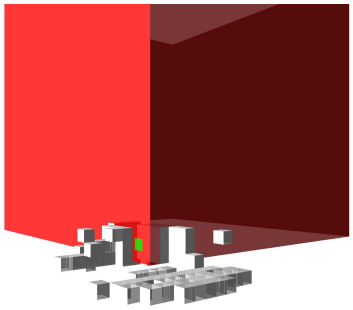


Figure 6.3: Destination is set too far away from the start. Goal voxel is colored red, start voxel is colored green

Lidar scans is limited to a relatively small distance. This leads to the low depth expansion of the octree in places where no lidar measurements are available. If the goal node will be chosen far away from the start node, the appropriate voxel where goal is located can be too large. In 6.3 the example of that behaviour is shown. Green voxel is a voxel where the UAV is located and the red voxel is the result of search 10 meters away in x and y coordinates from the starting point.

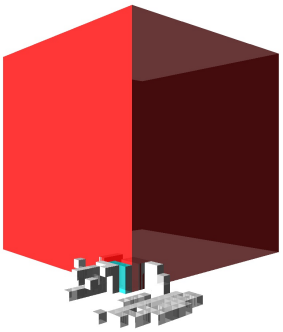


Figure 6.4: Common neighbours of the start and the goal voxel(filled with blue color)

6.5 shows the common neighbours for the start and goal nodes. In this case A* will find a path that consists of the one of the blue voxels and red voxel. In order to solve this issue depth limit will be introduced. If the tree is not expanded to the provided minimal depth limit in some point, new node will be created at this limit. This virtual node will not exist in the tree, however, its size and metrics can be determined, so that A* is able to process it.

6.3.5 Accessible neighbours

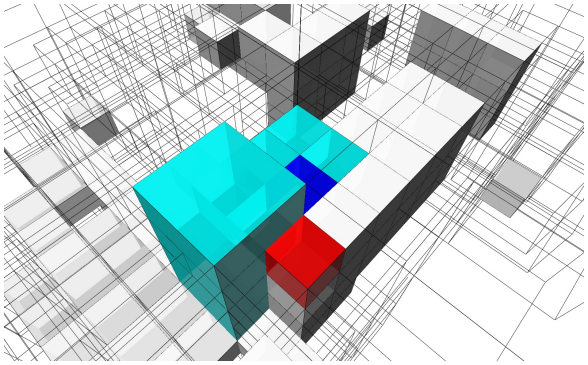


Figure 6.5: Neighbours of the blue voxel

For neighbours that share the same edge or a vertex require additional constraints to decide whether they are accessible or not. For example, 6.5 demonstrates neighbours of dark blue voxel as blue voxels. Red voxel should not be considered as a neighbour voxel because it can not be directly accessed from the initial dark blue voxel without causing collisions. That means that for every neighbour that shares the same edge with the current one, it is required to check two adjacent directions. For every neighbour that shares the same vertex accessibility can be checked by checking three adjacent directions. For example, for neighbour in the NW direction(6.3a) is accessible only when neighbours in the N direction and the W directions are not occupied. For neighbour in the FSW direction(6.3b) neighbours in the F, S and W directions should be checked.

Chapter 7

Implementation

7.1 Environment

The implementation of the proposed methods was made in the simulator Webots with the quadracopter Mavic Dji Pro, which has Quad X configuration. Lidar Robotis LDS-01 which is a one layer lidar with a 360 degree field of view and maximum distance up to 3.5 meters and a distance accuracy $\pm$ 15mm was used. Language that was chosen for implementation is C++. Open-source Octree solution, octomap, was incorporated and extended with neighbour search algorithm and coorditate searching with limited depth([HWB+13]).

7.2 The main loop

The main loop is implemented as following:

```
double current_state[3]
double desired_state[3]

// Hover to the appropriate position
while (current_state[2] != desired_state[2]){
    scanSpace()
    adjustPosition(0, 0, desired_state[2])
    scanState()
}

double next_goal
point3d path[]

// Generate initial path
path = AStar(desired_state)
double path_update_time = now()

while(current_state != desired_state){
```

```
    curr_time = now()
    // Path re-planning every 1 second
    if (curr_time - path_update_time >= 1){
        path_update_time = curr_time
        path = AStar(desired_state)
    }

    next_goal = calcNextState(curr_time - path_update_time, path
    )
    scanSpace()
    adjustPosition(next_goal)
    scanState()
}
```

Function *scanSpace* reads measurements from lidar and puts them into the map. *adjustPosition* function gives the desired position of the UAV to regulator and *scanState* function reads the measurements from gyroscope, IMU and GPS. Function *calcNextState* calculates next state based on linear interpolation of the points in the path:

```
function calcNextState(time, path){
    // Set constant velocity to 0.5m/s
    vel = 0.5
    traj_time = euclidean_dist(path[i], path[i-1])/vel for i =
    1..path.length

    // Calculate the cumulative sum
    traj_time_cum = cumsum(0, traj_time)
    idx = i if traj_time_cum[i-1] < time < traj_time_cum[i]
    k = time/traj_time[idx - 1]
    return (1 - k) * path[idx - 1] + k * path[idx]
}
```

7.3 Octree implementation details

Octomap framework does not allow to directly access the size, depth and coordinates properties of the voxel. Because of that new search function was implemented. In Octomap each voxel has a unique key. With the given coordinates the key of the leaf voxel can be retrieved. Then this key can be used to retrieve the index of the child in particular depth. Moreover, key of the node allows to get coordinates of the voxel center. So the search function is pseudocode will look like as following:

```
function search(x, y, z, minDepth){
   OcTreeKey leafKey = getKey(x, y, z)
   OcTreeNode currentNode = root
   for (d in 0...max_depth){
     int childIdx = getChildIndex(leafKey, d)
     if (currentNode.hasChild(childIdx)){
        currentNode = currentNode.getChild(childIdx)
     } else {
```

```
        if (depth < minDepth){
            depth = minDepth;
            currentNode = new OcTreeNode()
        }

        key = adjustKeyAtDepth(d)
        double size = getSize(d)
        point3d center = getCenter(key)
        return new NodeWithMetrics(currentNode, size, center)
     }
   }
}
```

Instead of hard-coding indexes of children in directions, byte operations will be used: each index of the child in binary form represents the shift in x, y and z coordinate from the center of the parent node. For example, index 4 can be written as 100. First digit, one, indicates the positive offset of the child center in x direction related to the parent's center and other digits indicates negative offset in other directions. That means that all children that are sharing the northern face with the parent voxel have 1 in the third place, all southern children have 0 in the third place and so on. The pseudocode for finding children in face directions will look like as following:

```
function faceChildren(){
    // Direction order: N, S, W, E, F, R
    int children[NUM_FACE_DIRS][4];
    for (i = 0...7){
        if (i & 1 == 0){
            children[4].push(i)
        } else {
            children[5].push(i)
        }
        if (i & 2 == 0){
            children[2].push(i)
        } else {
            children[3].push(i)
        }
        if (i & 4 == 0){
            children[0].push(i)
        } else {
            children[1].push(i)
        }
    }
}
```

For edge and vertex directions index of children can be found as an intersection of the children of adjacent directions. For example, RN direction the intersection of children in R direction and in N direction will give the appropriate set of children. For RNW direction the intersection of R, N and W children sets will be found.

7.3.1 Parameter adjustments

The maximum resolution of the map was taken as the bounding box of the UAV with additional 10 centimeter space, 0.7 meters, which allows to check occupancy of only one voxel to decide if it is free, whereas the maximum depth of the tree is 16. Experiments show that this size of bounding box is enough to ensure flight though the centers of the voxels without colliding with the neighbour voxel.

We want to set minimum depth of the node as high as possible and avoid paths that can be generated in situation 6.3. Because of that we will select a depth were voxel size is less that the maximum observed distance. In our case the size of the node in depth 14 is 2.8 meters in one dimension, whereas in depth 13 cells reaches 5.6 meters in one dimension. With maximum lidar resolution distance of 3.5 meters, the value of minimum query depth will be set to 14.

7.4 Results

The following experiment shows that the UAV is able to map an environment, generate collision-free path and follow it. The initial position of the UAV was (0.2, 0.3, 0). Random goal points where chosen. First the UAV hovered to the desired z point, which in our case is 1. Then it followed the trajectory, that was re-generated by A* every 1 second, at the speed 0.5 m/s. The following images demonstrate the resulting path of the quadracopter marked as blue. Mapped obstacles are represented as grey blocks.

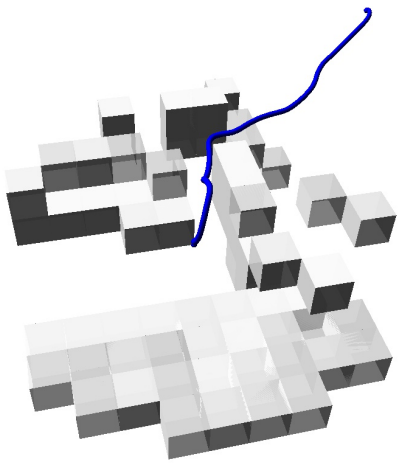


Figure 7.1: Path 1

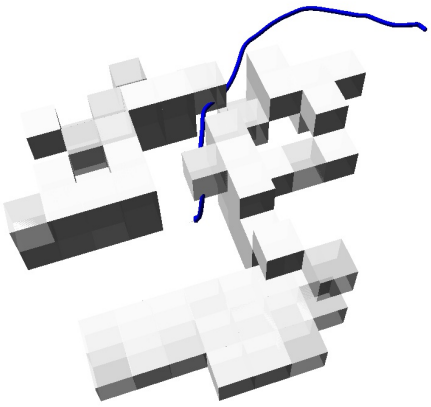


Figure 7.2: Path 2

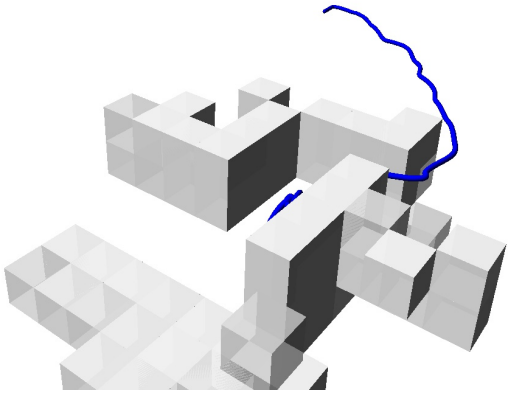


Figure 7.3: Path 3

Chapter 8

Future optimizations and improvements

1. Adding more sensors to the system will help to map the surrounding environment better. For example, it could be cameras or RGB-D camera.

2. More sophisticated physical model of the system can be proposed that better approximates the UAV physics. Wind effect, aerodynamics and body shape could be taken into account.

3. The proposed GNSS localization is not accurate enough for a real outdoor environment, therefore SLAM algorithm can be incorporated to deal with the localization problem.

4. The designed algorithm is able to generate only C_0-continuous trajectory. In order to achieve smooth flight, C_1(velocity) and C_2(acceleration) continuity should be ensured. Moreover, A* generates a path that lies near the obstacles. In order to solve this issue gradient descent optimization could be applied as described in [ZWY+20]. During this algorithm the B-spline curve is generated. Convex hull property of this curve allows to examine only those points, that are convex combinations of the control points.

5. Another optimization can be done in path planning algorithm. According to [KL02], D*-lite algorithm was primarily developed for dynamic environments. This algorithm make of use of previously constructed path and build new one on the top of previous calculations.

Chapter 9

Conclusion

During this work mathematical model of quadracopter was derived. Based on this model the regulator was developed and tested in both Matlab and simulation environment Webots.

Localization problem was solved by incorporating of the Kalman filter algorithm on raw sensor measurements.

The memory-efficient probabilistic space representation was proposed and solutions for the issues that are connected with that representation were provided.

Finally, time-efficient path planning algorithm was proposed.

All pieces were connected together and implemented in programming language C++. Program were tested in simulation environment Webots and results were provided. Provided results prove an ability of designed algorithm to generate collision-free path and quadracopter was able to follow this path.

Proposed autonomous navigation system is efficient in terms of memory consumption and speed of computations. Mapping the environment with one layer lidar brings some limitations to the whole algorithm. With some assumptions made, we were able to construct the path through the space, but in future system should be extended with sensor with larger field of view.

Appendix A

Abbreviations

Symbol	Meaning
BFS	Breadth-first search
DFS	Depth-first seach
IMU	Inertial Measurement Unit
GPS	Global Positioning System
GNSS	Global navigation satellite system
RRT	Rapidly exploring random tree
PRM	Probabilistic road map
TOF	Time-Of-Flight
UAV	Unmanned aerial vehicle
PID	Proportional - Integral - Derivative (controller)
SLAM	Simultaneous Localization and Mapping

Appendix B

List of Notation

Symbol	Meaning
X^T	Transpose matrix
x^T	Transpose vector
X^{-1}	Inverse matrix
$\dot{x}$	time derivation of x

Appendix C

Bibliography

[AW87] John Amanatides and Andrew Woo, *A fast voxel traversal algorithm for ray tracing*, Proceedings of EuroGraphics **87** (1987).

[CB00] Howie Choset and Joel Burdick, *Sensor-based exploration: The hierarchical generalized voronoi graph*, The International Journal of Robotics Research **19** (2000), no. 2, 96–125.

[CN06] D. M. Cole and P. M. Newman, *Using laser range data for 3d slam in outdoor environments*, Proceedings 2006 IEEE International Conference on Robotics and Automation (2006).

[ECM17] Christopher Eaton, Edwin Chong, and Anthony Maciejewski, *Robust uav path planning using pomdp with limited fov sensor*.

[HWB+13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard, *Octomap: an efficient probabilistic 3d mapping framework based on octrees*, Proc. IEEE Int. Conf. Virtual Environments, Human-Comput. Interfaces, and Meas. Sys (2013).

[KL02] Sven Koenig and Maxim Likhachev, *D* lite*, Eighteenth national conference on Artificial intelligence **1** (2002), 476–483.

[KL09] Jaewoong Kim and Sukhan Lee, *Fast neighbor cells finding method for multiple octree representation*, Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA (2009), 540–545.

[LPMM18] Bingxi Li, Sharvil Patankar, Barzin Moridian, and Nina Mahmoudian, *Planning large-scale search and rescue using team of uavs and charging stations**, 1–8.

[MNG15] Sina Sharif Mansouri, George Nikolakopoulos, and Thomas Gustafsson, *Distributed model predictive control for unmanned aerial vehicles*, 2015.

[NSC17] Tae Nam, Jae Shim, and Young Cho, *A 2.5d map-based mobile robot localization via cooperation of aerial and ground robots*, Sensors **17** (2017), 2730.

[PGC18] Bryan Penin, Paolo Giordano, and François Chaumette, *Minimum-time trajectory planning under intermittent measurements*, IEEE Robotics and Automation Letters **PP** (2018), 1–1.

[Rig12] Gerasimos Rigatos, *Nonlinear kalman filters and particle filters for integrated navigation of unmanned aerial vehicles*, Robotics and Autonomous Systems **60** (2012), 978–995.

[RVZ16] O Rodenberg, Edward Verbree, and Sisi Zlatanova, *Indoor a* pathfinding through an octree representation of a point cloud*, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences **IV-2/W1** (2016).

[SAC+08] Avneesh Sud, Erik Andersen, Sean Curtis, Ming C. Lin, and Dinesh Manocha, *Real-time path planning in dynamic virtual environments using multiagent navigation graphs*, IEEE Transactions on Visualization and Computer Graphics **14** (2008), no. 3, 526–538.

[Vö00] J. Vörös, *A strategy for repetitive neighbor finding in octree representations*, Image and Vision Computing **18** (2000), no. 14, 1085–1091.

[WZSM15] Naifeng Wen, Lingling Zhao, Xiaohong Su, and Peijun Ma, *Uav online path planning algorithm in a low altitude dangerous environment*, IEEE/CAA Journal of Automatica Sinica **2** (2015), no. 2, 173–185.

[XY18] Chen Xia and Ai Yudi, *Multi — uav path planning based on improved neural network*, 354–359.

[YFZ+18] L. Yao, Z. Fan, G. Zhu, Wenji Li, Chong Li, Yupeng Wang, and Honghui Xie, *A slam with simultaneous construction of 2d and 3d maps based on rao-blackwellized particle filters*, 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI) (2018), 374–378.

[YJZ18] Hongji Yang, Qingzhong Jia, and Weizhong Zhang, *An environmental potential field based rrt algorithm for uav path planning*, 9922–9927.

[YLX13] Fei Yan, Yi-Sha Liu, and Jizhong Xiao, *Path planning in complex 3d environments using a probabilistic roadmap method*, International Journal of Automation and Computing **10** (2013), 525–533.

[YZZ17] Xiang Yu, Xiaobin Zhou, and Youmin Zhang, *Collision-free trajectory generation for uavs using markov decision process*, 2017 International Conference on Unmanned Aircraft Systems (ICUAS), 2017.

[ZWY+20] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao, *Ego-planner: An esdf-free gradient-based local planner for quadrotors*, IEEE robotics and automation letters (2020).

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mozgunova Anastasiia** Personal ID number: **474312**

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

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Collision-free Navigation System for Robotic Helicopter

Bachelor's thesis title in Czech:

Bezkolizní navigační systém pro robotickou helikoptéru

Guidelines:

The aim of the work is the design and implementation of collision-free navigation system for autonomous robotic helicopter. The work include following tasks:
- do a research of available methods and sensors for obstacle detection suitable for small robotic helicopter,
- make yourself familiar with multi-rotor UAV control,
- choose a suitable base methods and design a systém allowing the helicopter to move to specified destination point without collision with obstacles,
- implement the designed navigation system and evaluate its performance on a helicopter model in laboratory or in the simulation.

It is expected, that certain restrictive conditions will be defined, allowing feasibility of the solution in the expected scope of the bachelor's thesis. These conditions may include assumption of helicopter position knowledge, which will be provided by the motion capture system in the laboratory. Other condition may restrict shapes or placement of the obstacles in the environment.

Bibliography / sources:

[1] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, Fei Gao - EGO-Planner: An ESDF-free Gradient-based Local Planner for Quadrotors - IEEE ROBOTICS AND AUTOMATION LETTERS, 2020
[2] Daniel Warren Mellinger - Trajectory Generation and Control for Quadrotors - University of Pennsylvania, 2012
[3] Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, Fei Gao - Survey of UAV motion planning - IET Cyber-systems and Robotics, 2020

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Chudoba, Intelligent and Mobile Robotics, CIIRC

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

Date of bachelor's thesis assignment: **06.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Jan Chudoba Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

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

. _____
Date of assignment receipt

_____
Student's signature