TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

IAY70LT

Lauri Kurs 122209IASMM

TRAJECTORY PLANNING FOR EXPERIMENTAL UNMANNED AERIAL VEHICLE

Master thesis

Thomas Hollstein, Ph.D., Professor
Uljana Reinsalu, Ph.D., Research Scientist
Karl Janson, M.Sc., Early Stage Researcher

Tallinn 2016

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Lauri Kurs

03.06.2016

Abstract

This thesis presents the design of an experimental autonomous unmanned aerial platform for implementing autonomous navigational capabilities. Most suitable environmental perception system for such platform is introduced, designed and built. A LIDAR-based environmental perception system uses one laser range-finder sensor and a motorized gimbal stabilization system to sense the environment in 3D, being undependable from the aerial vehicle's movements. Such system is capable of perceiving the environment in a range up to 40 meters, however longer-range (15 – 40 m) measurements may lead to inaccurate representation of an environment. With the point-cloud-like result from a LIDAR-based system, the X, Y and Z coordinates of the target objects in the environment are calculated – based on the object's distance and the sensors location in the environment. With such dataset, it is possible to create a representation, using OctoMap model, based on octrees. Main issue of the LIDAR-based environmental perception system can be proper configuration: scanning granularity and scanning angle must be carefully considered, as the actual scanning-time is limited. It has been shown, that approach for such system is successful - results of the environmental scans are promising and further improvements may greatly add more flexibility to this system.

Annotatsioon

LIDARi-põhine ümbruskonna tajumise süsteem eksperimentaalsele mehitamata õhusõidukile

Käesoleva projekti eesmärgiks on ehitada autonoomne mehitama õhusõiduk, mis suudab ohutult navigeerida algpunktist lõpp-punkti. Autonoomse funktsionaalsuse eesmärgi saavutamiseks on kõigepealt uuritud erinevaid mehitamata õhusõidukite platvorme ja tööpõhimõtteid. Tulemusena ehitatakse eksperimentaalse õhusõiduki platvorm.

Käesoleva lõputöö eesmärgiks on uurida ümbruskonna tajumise süsteemide võimalusi eksperimentaalsele mehitamata õhusõidukile, mis aitab saavutada autonoomse funktsionaalsuse ja ohutu navigeerimise. Ümbruskonna tajumiseks on valitud avatud lähtekoodiga tõenäosuslik 3D-kaardistamise mudel OctoMap, mis põhineb Octree puustruktuuril.

Lõputöö eesmärgina on vaadeldud erinevaid lähenemisi, kuidas ümbruskonna tajumise süsteeme luuakse, milliseid mudeleid on realiseeritud ning millise põhimõttega süsteem sobiks käesolevasse töösse kõige enam. Antud süsteemiks on valitud LIDARi-põhine ümbruskonna tajumise süsteem, mis koosneb laser-kaugusmõõtja sensorist, motoriseeritud stabiliseerimise süsteemist ja juhtkontrollerist, mille abil on seda süsteemi võimalik juhtida. Selline süsteem on võimeline teostama mõõtmisi 3D-keskkonnas, võimaldades väljastada punktipilve tulemusi – sihtmärgi-objektide XY Z koordinaate antud keskkonnas. Sellist punktipilve võib kasutada ümbruskonna mudeli loomiseks ja lennu ajal takistuste avastamiseks. Ümbruskonna mudelit on võimalik kasutada õhusõiduki ohutu teekonna planeerimiseks õhusõiduki autonoomsuse saavutamiseks.

On oluline õigesti valida ümbruskonna kaardistamise karakteristikud – kaardistamise detailsuse samm, mõõtmise kaugus ja mõõtmise ulatus, et süsteemi väljund oleks võimalikult täpne. Tulemused näitavad, et antud süsteem suudab edukalt ja täpselt

ümbruskonda tajuda ning selle süsteemi väljundit võib kasutada edukalt ruumi kaardistamiseks. Süsteemi täiendamiseks on võimalik kasutada mitmeid lähenemisi, parendades süsteemi ümbruskonna kaardistamist kiiremaks ning süsteemi efektiivsemaks. Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 106 leheküljel, 6 peatükki, 58 joonist, 15 tabelit.

Table of abbreviations and terms

2D	Two-Dimensional
3D	Three-Dimensional
ADC	Analog-To-Digital Converter
API	Application Programming Interface
ARM	Advanced RISC Machine Architecture
BLDC	Brushless DC Electric Motor
CAN	Controller Area Network
CCW	Counter Clockwise
CW	Clockwise
ESC	Electronic speed controller
ESC	Electronic Speed Controller
GPLV3	GNU General Public License Version 3
GPS	Global Positioning System
I/O	Input/Output
I2C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
IR	Infrered
LIDAR	Light Detection and Ranging
LiPo	Lithium polymer (Type of battery)
LIPO	Lithium Polymer
MAV	Miniature Unmanned Aerial Vehicle
MAVLink	Micro Air Vehicle Link
MCU	Microcontroller Unit
MPP	Mission Path Planning
PCL	Point Cloud
PID	Proportional-Integral-Derivative
PPM	Pulse Position Modulation
PPM-SUM	Sum Signal for Pulse Position Modulation
PWM	Pulse-Width Modulation
RC	Radio Control
ROS	Real-Time Operating System
RPM	Revolutions Per Minute
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System

SF	Slow-Fly
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
UBEC	Universal Battery Elimination Circuit
US	Ultrasonic

Contents

List of Figures

List of Tables

1 Introduction

Statement

The analysis and construction of an experimental unmanned aerial vehicle is collaboration work and Sections 1, 2, 3 with their subsections are shared in current thesis and [1].

Unmanned aerial vehicles offer wide application usage – these systems are usually autonomous and operate without human-intervention. Such small scale vehicles can perform tasks as search, rescue and disaster response, inspection of dangerous places, GIS, mapping and observation of indoors or outdoors areas, environment monitoring, security surveillance and inspection. Fixed-wing unmanned vehicles, suitable searching large area, are successfully used for agricultural inspection. Various payload-based applications are also suitable – for delivery of low-mass supplies.

Today, different approaches are used for designing such applications from standalone vehicles to aerial vehicle swarms. Systems can be designed on GPS control, which simplifies the localization problems, however, accuracy of the sensor must be considered. Such vehicles cannot fly in cities near large buildings as GPS reception is reduced and as those mission are safety critical. For achieve better accuracy with GPS, another approach of redundant design with sensor fusion to achieve better accuracy can be used -- from 2.5 m to centimetre-level accuracy [2]. Systems can be also designed for indoor use, using other sensors and approaches to achieve localization within the environment.

To add: Drone's goal, (miks puude kohal ei lenda) ----- Main purpose is to achieve autonomous flight mission capabilities for small-scale aircraft, which is able to avoid obstacles based on light detection and ranging-based (LIDAR) perception system. Vehicle is able to carry out missions in GPS available areas of old-grown forest, with low density of forest-floor, below treetops to add: Eva’s part intro

To add: Eva's part intro [1]

To add: Lauri's part intro

To add: Process flow, how we thought it would work, very briefly

Construction of such system is a challenging task – selection of overall design such as frame construction, weight balance, type of aircraft and selection of flight system must be carefully considered. Flight dynamics of such vehicle are hard to model. Main criteria for the project is to build compact aircraft to fly in narrow areas with large enough payload area and lift power to carry a LIDAR-based perception system.

For such design, four-rotor configuration allows maneuverability with capability of generating enough lift power for desired payload over fixed-wing aircrafts. Multi-rotors are highly maneuverable and thus especially suited for challenging indoor [3], [4], [5], [6] and outdoor [7], [8], [9], [10] tasks. Maneuverability allows such systems to traverse in narrow and complex areas. Such vehicles are able to carry wide range of scientific payloads for deploying UAV (unmanned aerial vehicle) for the task of an autonomous exploration.

Our system consists of four main architecture blocks, see Figure 1.

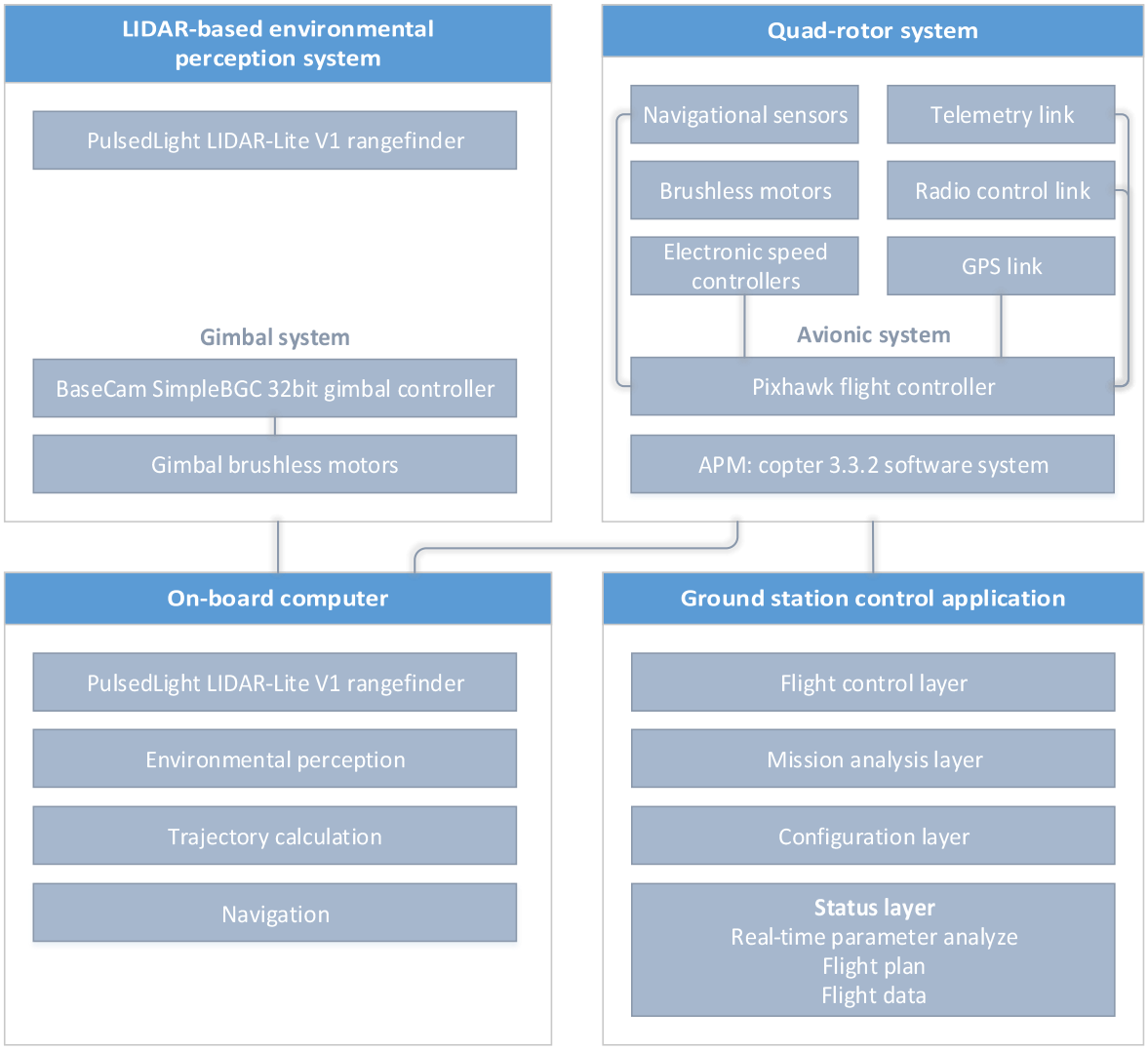


Figure 1. System Architecture Block Diagram.

Quad-rotor System. Quad-rotor system consists of physical components for aerial vehicle, navigational sensors including (i) an inertial measurement unit (IMU), (ii) global positioning system (GPS), (iii) magnetometer and avionic system. Main functionality of an avionic software system is: [11] (i) analyzing and collection in-flight information, (ii) executing automatic control laws: flight control algorithm execution, actuator control driving, (iii) communicating with ground control station, and (iv) logging necessary in-flight data.

LIDAR-based environmental perception system. For environmental sensing LIDAR-based system, described in chapter 6 in [1] will be used, which consists of gimbal system and

laser range-finder, device that measures distance from the observer to a target. Gimbal is used for stabilizing and directing sensor beam to desired direction. Laser range-finder module works in various environmental conditions such as sunlight, rain or fog which has advantage over ultrasonic or infrared sensors as range-finder provides good measurement accuracy and longer distance sensing.

On-board computer system. On-board computer is used to handle control routines to implement autonomy for the vehicle.

Main routines hosted on on-board computer are (i) environmental perception (scanning task to create map of environment; real-time safety scanning task while navigating), (ii) trajectory calculation, and (iii) navigation for multi-rotor.

To model environment, OctoMap mapping model based on *octree* structures will be used, described in chapter 3. To create a world-map of possible obstacles and free space, the OctoMap tools will be used, input for the map is used from LIDAR-based system. Based on the map is it possible to understand the environment and plan for best suitable trajectories for UAV to follow.

Trajectory calculation is executed based on environmental perceptional model. The A* search algorithm is implemented on OctoMap tree-structure to find most suitable trajectories for the multi-rotor (reference to Lauri's work). For finding best possible trajectory from source to destination cost model for the vehicle is carried out. Various parameters are counted such as (i) security measures (width of corridor of possible path), (ii) speed for the vehicle, (iii) path length, (iv) time limitation for our model, and (v) energy model. According to cost model most effective trajectory is selected to implement the GPS-based way-points as a possible route for navigation task.

Ground station control application. Ground station control application realizes communication between avionic system and ground station control system via wireless link. This system is preferable but not compulsory, main purpose is to see in-flight data in real-time via user interface. Application can be categorized as follows: (i) flight control layer, (ii) mission analysis layer, (iii) configuration layer and (iv) status layer. Flight control

layer is responsible for sending desired executable flight control messages from ground station application to avionic system. Mission analysis layer handles generation and maintenance of desired flight trajectories and logging mission data. Configuration layer allows to read and edit UAV's flight and configuration parameters. Status layer allow user to monitor in-flight data in real-time.

2 Configuration for Experimental Unmanned Aerial Vehicle

For the model to be effective - frame construction, weight balance, type of aircraft and selection of flight system must be carefully considered. Section 2.1 introduces quad-rotor's dynamics for selection components like motors, propellers and frame as these configurations are most important as physical components for air-vehicle.

Another, yet very important configuration for quad-rotor, see section 2.4, is selection of actual avionic system, which includes flight controller with avionic software system, which is responsible for executing automatic control laws: flight control algorithm execution, actuator control driving, communicating with ground control station and logging necessary in-flight data.

Estimated models with possible suitable components are designed, see section 2.2. To make rough evaluation on the models, viability calculation is made with *ECalc* tool. It is essential to evaluate that all components, specially rotors and propellers would suit for specific frame, for quad-rotor it is to lift the frame with estimated payload, which would result in working vehicle.

Most suitable configuration is selected and final quad-rotor model is built, see section 2.3.

Flight results on a real model is introduced in section 2.5.

Error! Reference source not found. shows overall model of the system. Figure 34 [12] shows in detail components for the quad-rotor model.

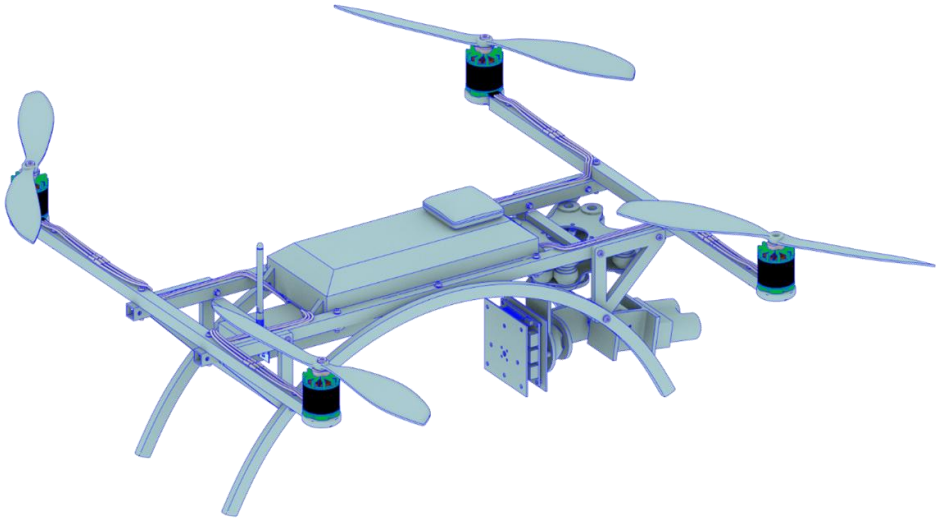


Figure 2. Model of Multirotor and Perception System.

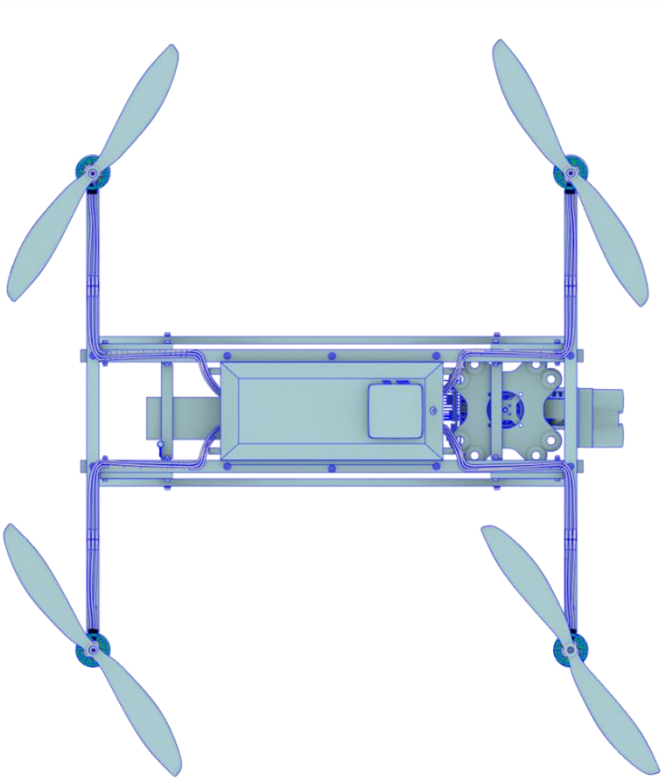


Figure 3. Model of Quad-rotor and Perception System.

2.1 Quad-rotor's Model

2.1.1 Quad-rotor's Dynamics

For air vehicle model quad-rotor-typed air vehicle is constructed. Four rotors generate upwards lift, independent control of relative thrust to each rotor results desired movement of the model. With a change of a speed of each rotor, possible desired turning force is achieved.

Rotors are aligned in shape of rectangle, two rotors turn in clockwise (CW) direction and the other two rotate in the opposite direction (CCW), as shown on **Error! Reference ource not found.**. The aerodynamic torque of the first rotors pair cancel out the torque created by the second pair which rotates in the opposite direction. This rotation configuration neutralizes rotors' tendency to make multi-rotor rotate so if all four rotors apply equal thrust, the multi-rotor will maintain its direction.

Multi-rotor has four controllable degrees of freedom: altitude, yaw, pitch and roll. Each degree of freedom is controlled by changing speed of rotors. To maintain the overall balance and desired position, sophisticated control system must be used. Pixhawk avionic flight control system is used as a multi-rotor's control system.

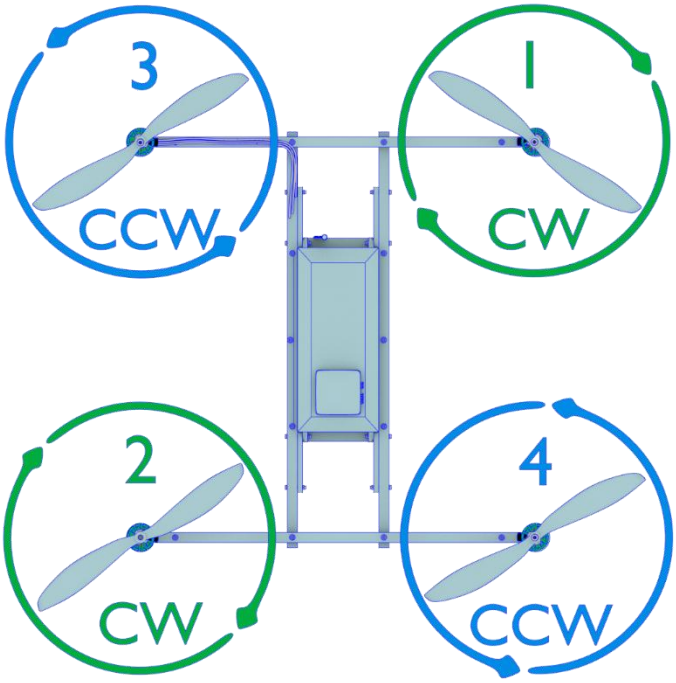


Figure 4. H-type frame configuration.

To control position in any degrees of freedom, speed of rotors are changed. Different axis are shown on **Error! Reference source not found.**.

Figure 5. Quad-rotor's axis configuration.

2.1.2 Steering

For controlling direction in yaw axis, the control system is to slow down opposite pairs of rotors relative to other pair. Rotation is to take place as angular momentum of the two pairs of propellers will not be in balance. Multi-rotor is to rotate in either direction by changing rotor speed in different pair of rotors.

For controlling direction to roll and pitch axis, control system is to change speed of two rotors on the side. One side of the model is to have more thrust than other side, causing multi-rotor to tilt.

Axis-tilt position changes in causes multi-rotor to move. With the tilt, multi-rotor can move to different directions as lift force produced by the rotor is not directed downward resulting in pushing the multi-rotor. For this movement it is important for control system to be able to maintain the altitude since less rotor thrust power is directed downward while tilting.

2.1.3 Frame

Frame is the structure that holds the multi-rotor and its components together. For the model it is to be as light as possible. Most available materials are carbon fiber, aluminum or polyurethane foam. For the final model, hollow aluminum square rods are used for their relatively light weight, rigidness and affordability. However as damping effect for aluminum is not so good, to reduce some vibrations to the frame, plastic motor attachments are used. H-typed frame (with configuration)

Frame consists of 3D printed parts: bottom, top, motor attachments and aluminum rods. Frame is light enough for our settings. Design is expandable if needed, easy to repair and as light as possible.

Frame design testing showed that most suitable frame for project's configuration was with few 3D custom parts which is light enough for vehicle to be able to motion outdoors.

First version of the frame was handmade polystyrene foam square size of 500x500x50mm with 4 holes inside with diameter about 180mm. Two aluminum profile rods were attached as motor attachments. This frame was too soft and small for this vehicle.

To reduce problems, another version of the frame was constructed with 3D printed parts. Frame did not suit because of the weight of the frame and that it was not elastic nor strong enough. Severe blow would harm the frame.

Next version was designed with polystyrene body with more elaborated design but it also was too soft to the impacts with the objects as well in strong wind the frame did not have good effect on the vehicle. Problem with this frame was weight and strong wind drag.

Final design of the frame was without external bumpers and for outside flight only.

2.1.4 Motors and Propellers

For heavy lift as well as slow and stable ride brushless DC motors are used along with electronic speed controllers for each motor. Each motor is controlled separately by a speed controller via avionic controlling system. The final selection of motor and propeller is to match the overall model. Main characteristics considered for motor selection is KV-rating, which indicates the revolutions per minute for number of volts. For final motor selection it is to estimate the final weight of the model for create required thrust for lifting the multi-rotor.

General rule is to provide twice as much thrust than weight of the multi-rotor. For larger multi-rotors that carry payloads, low KV motors work better as they have more rotational momentum and maintain multi-rotor's stability.

Propellers are selected based on motor characteristics. For selection propellers length and pitch are important. Pitch is a parameter for travel distance of single propeller rotation. Higher pitch means slower rotation, but increase of the multi-rotor's speed with usage of more power. Lightness of overall model is very important as excess weight reduces battery life and maneuverability. To have an optimal flight time it is important to find balance between final selection of rotors and propellers.

Increase of propeller's size and pitch characteristics, more thrust can be generated therefore more lift force for multi-rotor is achieved. The bigger propeller's diameter, the more efficiently model hovers, but control of the model is decreased. Although carbon fiber typed propellers are expensive in respect to plastic ones, carbon fiber material is selected as it is significantly stronger than plastic propellers. (high performance, durable, expensive, stiff, rigid, strong - propellers flexes and decreases efficiency of the blade)

With intention of having stable ride with heavy multi-rotor, low-RPM rotors are to be selected. Based on rotor selection, suitable carbon fiber slow-fly typed propellers are used. Slow-fly propellers [13], that usually generate higher amount of thrust in respect to speed propellers, have wide taper and broad flat blades, usually lower pitch. With this configuration, the multirotor has enough power to lift the estimated payload.

For controlling rotors, suitable electronic speed controller (ESC) is used. Selection of the component depends on selection of rotor and rotor's current consumption. Li-Po as power source is to be used. Type of a battery depends on motors; bigger capacity of battery estimates longer flight time.

2.2 Estimate on Components. Viability Calculation

To find suitable electric components for the model, Ecalc [14] tool is used to make a rough estimate for correct setup. Ecalc is to simulate and evaluate electric motor driven systems for remote controlled models. This tool is to show problematic areas when selecting components for the model. It is essential to evaluate that all components, specially rotors and propellers would suit for specific frame for multi-rotor is to lift the frame with estimated payload, which would result in working vehicle.

Largest propeller for the frame, the bigger the total disc area the more efficient multi-rotor hover is achieved, but the slower gets the control response. Final propeller selection is based on final rotor, for air vehicle dynamics please refer to Section 2.1.4.

Table 1. Estimate on Model's Dimensions.

Parameter	Value	Unit

General		
Number of motors	4	-
Type of rotor configuration	Flat	-
Frame size	388×388×100	mm
Flight controller tilt limitation		-
Estimate		
Multirotor system	1200	g
Battery	350	g
Payload (LIDAR system)	300	g
Total estimate weight	1800	g

Figure 6. Estimate of Suitable Rotor/Propeller Configuration.

Estimate on rotors. Weight of the model and payload is essential, refer to Table 1. For most optimal flight based on our configuration, slow rotors are proposed (estimated KV rate: 770 - 1120), see Figure 6. Two-bladed 11" propellers are proposed, final selection results on a specific motor. Approximate motor power is estimated between 275 - 480 watts, which would result in using electronic speed regulators between 25 - 45 amperes.

Many different combinations were considered for the same frame configuration. Best possible results based on estimated weight and frame type are shown in Table 2.

Possibilities of low-RPM rotors were considered. Turnigy Multistar motor along with 11"×4,7" and 12"×4,5" propellers gave most promising estimation. Hover flight time with usage of 5000mAh 3-cell battery in all cases is around 11 minutes (estimated time in min: 10"×4,5" - 11,1; 11"×4,7" - 11,4; 12"×4,5" - 12,1). Enough lift power is generated in all cases, but with combination of SunnySky 980KV motors and 10"×4,5" propellers the estimated load to rotor is near to maximum (estimated: 154W, max: 160W), which is to lead to possible motor failure.

Estimate on Propellers. Two different slow-fly propellers can be used with Turnigy Multistar 800KV motor for our model. With 11"×4,7" propellers better estimated motor efficiency is achieved, but efficiency of both configurations are very similar. At total drive, estimated efficiency at hover is 11"×4,7" - 75%, 12"×4,5" - 73%. At total drive, estimated efficiency at maximum throttle is 11"×4,7" - 72%, 12"×4,5" - 68%. In both cases 11"×4,7" and 12"×4,5" propellers result in similar efficiency.

Throttle Input. With longer propeller blades, better estimated throttle input at hover is achieved (est, %, 12"×4,5" - 50%, 11"×4,7" - 57%). With our specific configuration best throttle aim at hover is to be less than 60%, the lower, the better. Below 50% used for racer quad-rotors.

Specific Thrust. In both cases (est, g/W, 12"×4,5" – 7,9; 11"×4,7" – 7,5), specific thrust of propeller is efficient, results above 6 g/W considered as good efficiency. Specific thrust indicates grams of produced thrust with one watt of electric input on motor. Thrust to weight ratio is better with configuration of 12"×4,5" (est, 12"×4,5" – 1,9; 11"×4,7" – 1,6) although this estimate is rough as for final model carbon fiber propellers are to be used. Calculator's estimation is based on APC SF propellers, which have different *PConst* constant value from final selected-to-be propeller. But as usually this information is not published by manufacturer, rough estimation is used in this case.

Current Consumption. Estimated current per motor in both cases stays under 20A, at hover estimate around 5A (est, A, 12"×4,5" - 5,6; 11"×4,7" - 5,3). At full throttle current estimate per one motor is around 11A (est, A, 12"×4,5" - 10.2; 11"×4,7" - 10,2). Based on estimation 20A ESC's can be used.

Temperature. With both setups, load on a motor is not resulting motor overheat. With maximum estimated load on a motor and good cooling presumed, temperature change is minimal, around 11 degrees (est, degrees, 12"×4,5" - 11; 11"×4,7" - 14). Temperatures of rotor case over 80 degrees and higher might result in motor failure.

RPM limits for Propellers. Both propellers fit within permissive RPM range. At hover, 11"×4,7" estimated RPM represent 58% of max RPM, 12"×4,5" estimated RPM represent 61% of max RPM on specific propeller. At full throttle RPM of propellers are estimated 83% versus 94%. 11"×4,7" propellers show better result with Turnigy Multistar 800KV motor as lower load on a propeller is preferred.

Table 2. Different Multi-Rotor Possible Setups

Parameter	Unit	SunnySky @ 10×45	Turnigy @ 11×47	Turnigy @ 12×45
Battery LiPo 25/35C				
Configuration		3S1P	3S1P	3S1P
Load	C	12,11	9,81	11,64
Total capacity	mAh	5000	5000	5000
Minimum flight time	min	4,2	5,2	4,4
Mixed flight time	min	8,8	9,3	9,4
Hover flight time	min	11,1	11,4	12,1
ESC				
Current	A const.	20	20	20
	A max	25	25	25
Motor at maximum				
Current	A	15,14	12,27	14,55
Voltage	V	10,19	10,36	10,22
Estimated RPM	rpm	8239	6665	6253
RPM at full battery	rpm	12348	10080	10080
Maximum RPM for propellers	rpm	10500	8000	6667
Maximum motor power	W	160	220	220
Electric power	W	154,2	127,1	148,7
Mechanical power	W	123,4	98,4	110,2
Efficiency	%	80	77,4	74,1
Estimated temperature	°C	13	11	14
Motor at hover				
Propeller	Speed	10"×4,7"	SF 11"×4,7"	SF 12"×4,5"

Current	A	5,72	5,58	5,27
Voltage	V	10,76	10,76	10,78
Revolutions	rpm	5409	4615	4050
Throttle	%	51	57	50
Efficiency	%	80,1	76,7	74,3
Specific Thrust	g/W	7,31	7,5	7,92
Estimated Temperature	°C	5	5	5
Total drive				
Model Estimate	g	1800	1800	1800
Estimated Maximum Thrust	g	3240	2880	3420
Estimated Thrust per Rotor	g	810	720	855
Thrust-Weight		1,8	1,6	1,9
Current @ Hover	A	22,88	22,3	21,08
P(in) @ Hover	W	254	247,5	234
P(out) @ Hover	W	197,1	184,1	168,8
Thrust @ Hover	g	1857	1857	1853
Efficiency @ Hover	%	77,6	74,4	72,1
Current @ max	A	60,57	49,06	58,18
P(in) @ max	W	672,3	544,6	645,8
P(out) @ max	W	493,8	393,5	440,8
Efficiency @ max	%	73,5	72,3	68,3
Quadcopter				
Additional Possible Payload	g	1008	710	1095
Maximum Tilt	degrees	50	44	52
Maximum Speed	km/h	39	28	29
Rate of Climb	m/s	5,4	3,6	4,1

Legend for Table 2

- Battery
  - Configuration: Setup for configuration, number of cells.
  - Load: Actual discharge rate in relation to the capacity. Total Capacity: Used setup of capacity of the battery.
  - Minimum Flight Time: Expected minimum flight time, based on maximum throttle of maximum discharge % of battery and is independent of the weight.
  - Mixed Flight Time: Based on all-up weight when moving, result on max. discharge % of battery, base is geometric mean value of current difference from

hover to maximum throttle. Hover Flight Time: Expected flight time based on all-up weight when hovering only on max. discharge % of battery.

- Motor at Maximum
  - Current: Maximum estimated current draw per rotor.
  - Voltage: Maximum estimated voltage per rotor at maximum current.
  - Estimated RPM: Maximum revolutions for rotor at full throttle.
  - RPM at Full Battery: Revolutions for rotor at 100% full battery, at 3S1P, 12.6V.
  - Maximum RPM for Propellers: Maximum estimated revolutions for specific propeller, common RPM limits from manufacturers. Estimate taken: Graupner SF CF propellers 88000 RPM/diameter [15], APC Multirotor Speed propellers 105000 RPM/diameter [16].
  - Maximum Motor Power: Maximum load on a specific rotor.
  - Electric Power: Maximum electric input power.
  - Mechanical Power: Maximum mechanical output power or shaft power.
  - Efficiency: Efficiency at maximum ampere draw.
  - Estimated Temperature: Temperature of the rotor case. Temperatures over 80C result in rotor failure.
- Motor at Hover (for each rotor)
  - Propeller: Propeller type for setup.
  - Current: Estimated current for hovering. The hover current should be close to the optimal current.
  - Voltage: Rotor voltage for hovering.
  - Revolutions: Rotor revolutions at hover.
  - Throttle: Stick position to hover in manual mode as input signal. Indication of power signal to rotor at hover, to aim for 50-60%. Under 50% used for racer vehicles.
  - Efficiency: Rotor efficiency at hovering.
  - Specific Thrust: How many gram of thrust is produced with one watt of electric input power at the rotor.

- Estimated Temperature: Predicted rotor temperature - subject to the motor cooling.

- Total Drive
  - Model Estimate: Total estimate on the weight of the model.
  - Estimated Maximum Thrust: Total estimate for full thrust on a model.
  - Estimated Thrust per Rotor: Total estimate for full thrust per rotor.
  - Thrust-Weight: Dimensionless ratio of thrust to weight of a rocket, jet engine, propeller engine, or a vehicle propelled by such an engine that indicates the performance of the engine or vehicle. [17]. Flying below 1.2 is almost impossible.
  - Current @ Hover: Sum of all rotors when hovering.
  - P(in) @ Hover: Electric input power at battery when hovering.
  - P(out) @ Hover: Mechanical output power or shaft power when hovering.
  - Thrust @ Hover: Calculated thrust for hovering, based on specific thrust.

  - Efficiency @ Hover: Total efficiency when hovering.
  - Current @ max: Sum of all motors at full thrust.
  - P(in) @ max: Electric input power at battery at full thrust.
  - P(out) @ max: Mechanical output power or shaft power at full thrust.
  - Efficiency @ max: Total efficiency at full thrust.
- Quadcopter
  - Additional Possible Payload: Maximum additional payload possible to hover with 80% throttle to guarantee maneuverability.
  - Maximum Tilt: Theoretically maximum possible tilt of the copter to maintain level flight (neglecting down force due tilt).
  - Maximum Speed: Theoretically maximum attainable forward speed in flight at max. tilt and throttle (neglecting copter aerodynamic drag and down force due tilt)
  - Rate of Climb: Estimated maximum achievable rate of climb (neglecting copter aerodynamic drag).

Final Results: Both propellers generate enough thrust for the model with 800KV motor (est, g/W, 12"×4,5" - 7,9; 11"×4,7" - 7,5), which would result in working vehicle. Estimate on setup with propellers of 11"×4,7", thrust-weight is estimated at 1,6. With usage of 12"×4,5" propellers, estimate is 1,9. As propeller's *PConst* value remains undocumented by manufacturer of possible propellers that are to used, reliability of an estimation remains unknown.

Throttle input at hover is efficient in both cases, lower throttle is preferred (est, %, 12"×4,5" - 50; 11"×4,7" - 57). Current consumption per motor in both cases is similar, resulting in usage of same ESC of 20A for motors. Temperature changes are similar if model is to have good cooling.

Based on estimation, combination of motors of 800KV with 12"×4,7" propellers is to be used, see Figure 7. Although both setups are very similar, combination of selected motor and propellers result in most efficient setup as thrust-weight ratio is very important metric for selection of final components:

1. Total drive thrust has better estimate: Maximum thrust of 3420 grams by 12"×4,5" propellers to be generated while 11"×4,7" would result in 2880 grams of thrust on model all-up-to 1800 grams (est, 12"×4,5": 1,9; 11"×4,7": 1,6).
2. Better throttle at hover is achieved: With longer propeller blades, better estimated throttle input at hover is achieved (est, %, 12"×4,5": 50; 11"×4,7": 57).
3. Longer hover time estimate: With longer propeller blades, longer estimated hover flight time is to be achieved (est, min, 12"×4,5": 12,1; 11"×4,7": 11,4).

Figure 7. Estimated results for final model with Turnigy 800 KV 12"×4,5"

2.3 Specifications of Components used for a Model

Real model is built on estimated result of possible usable components. Estimated weight of 1800 grams resulted in reduced model weight of 1732 grams, for specifications refer to Table 3. Real dimensions of the model are 390 x 365 mm. Model, refer to Figure 8 is set up using H-typed Frame as deck space is needed for mounting the payload. This design provides more available area for components than usual X-typed Frame.

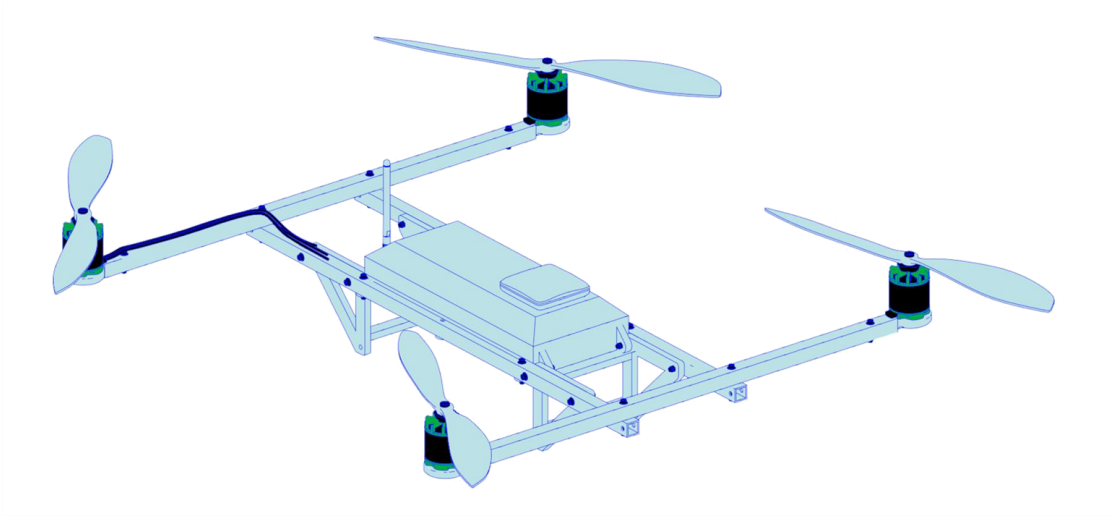


Figure 8. Multi-rotor's model

Motors and Propellers. Four Turnigy Multistar 2216 Outrunner motors (800 KV), refer to Table 4 for specifications, are used with 12"×4,5" Carbon Fiber Slow-fly propellers, see Table 6. For testing purposes also propellers 11"×4,5" are used.

Motors are controlled with Hobbywing 20A Skywalker Quattro uBEC 4-in-1 Brushless ESC, for specifications refer to Table 5. This controller is capable handling continuous current consumption of 20A for each motor, burst of 25A, enough of source current the selected motors require. Model is compact and comfortable as wiring complexity is simplified.

Power Supply. ZIPPY Compact 5000mAh 3S 25C LiPo Pack is used, see Table 7. Weight of 342 grams, this compact battery helps to reduce overall weight on a model. Battery's capacity is 5000 mAh resulting in maximum 12-minute hover flight time with payload on board.

Autopilot System. As an avionic system - 3DR Pixhawk flight controller is used with *APM: Copter 3.3.2* source configuration, refer to Table 10 and section 2.4 for details. 3DR uBlox GPS with Compass Kit is used for positioning.

Radio System. Three different radio modules are used for the model: telemetry link, manual radio control and for debug and navigational link for user interface operations.

For telemetry link, SiK Telemetry Radio module v1 433 MHz are used, refer to Table 8. Telemetry application is used for primarily sending data from aircraft to ground station application [18], it gives possibility to monitor vehicle's status while in operation. Basic and quick configurations of avionic software's parameters can be done via telemetry link. Updating, creating and loading autonomous missions to aircraft with simple point-and-click way-point entry on Google or other maps is possible via telemetry link and Mission Planner application software [19].

Manual control of the quad-rotor is achieved by using Taranis X9D Plus remote control unit [20] with X8R 16ch Receiver, connected directly to flight controller's SBUS port, see Table 9 for specifications. As safety precaution, this radio system gives user full control over quad-rotor if it is necessary to take over autonomous control operations. Remote control also offers testing possibilities of flight dynamics while developing autonomous flight implementations.

It would be possible to implement debug and navigation link over 2,4 GHz Xbee radio devices for user interface operations. Devices would be connected to an on-board computer and ground station computer. Possible implementation of software will be needed to see possible debug data as well as real-time view of the flight mission and map of the space.

Adjustments to radio configurations are made, to ensure correct usage of radio devices according to general requirements for the use of radio transmission equipment in Estonia [21].

Table 3. Configuration of Real Model. Weight vs Size

Parameter	Value	Unit
General		
Number of Rotors	4	-
Type of Rotor Configuration	flat	-
Estimated Frame Size	400 × 400	mm
Real Frame Size	390 × 365	mm

Real Frame Size with Propellers	696 × 672	mm
Estimate		
Multi-rotor System	1200	g
Battery	350	g
Payload (LIDAR system)	300	g
Total Estimate Weight	1800	g
Real Model		
Multi-rotor System	1081	g
Battery	342,4	g
Payload (LIDAR system)	308	g
Total Real Weight	1732	g

Table 4. Model Components: Motor Specifications

Parameter	Value	Unit
Model	Turnigy Multistar 2216 Outrunner	-
KV	800	rpm/V
Poles	14	-
Weight	83	g
Maximum Current	20	A
Idle Current	0.5	A
Maximum Voltage	12	V
Maximum Power	222	W
Connector	3.5mm bullet-typed	-
Dimensions		
Length	34	mm
Diameter	28	mm

Table 5. Model Components: ESC Specifications

Parameter	Value	Unit
Model	Hobbywing 20A Skywalker Quattro uBEC 4-in-1 Brushless ESC	-
Continous Current (per 1)	20	A
Burst Current (per 1)	25	A
BEC Output	5; 3	V; A
Battery Cell:	2S-4S, 7,4V-14,8V	-
Size:	70, 62, 11	mm, mm, mm
Weight:	112	g

Table 6. Model Components: Propeller Specifications

Parameter	Value	Unit
Model	Carbon Fiber Propeller 12"x4,5" Black	-
Material	Carbon Fiber	-
Pitch	04.jaan	inch
Diameter	12	inch
Weight:	15	g
Shaft Diameter:	6	mm
Hub Thickness:	8	mm
Number of Blades:	2	inch
PConst:	Unknown	-
TConst:	Unknown	-

Table 7. Model Components: Battery Specifications

Parameter	Value	Unit
Model	ZIPPY Compact 5000mAh 3S 25C Lipo Pack	-
Capacity	5000	mAh
Voltage	11.jaan	V
Cells	3	-
Continuous Discharge Rate	25C	-
Burst Discharge Rate	35C	-
Weight:	354	g
Balance Plug:	JST-XH	-
Discharge Plug:	XT60	-
Dimensions		
Length	162	mm
Height	21	mm
Width	46	mm

Table 8. Model Components: Telemetry Specifications

Parameter	Value	Unit
Model	RCTimer Radio Telemetry Kit 433Mhz	-
Supply Power	5	V
Operating Frequency	433	MHz
Receiver Sensitivity	-121	dBm
Maximum Transmit Power	100	mW
Used Transmit Power	10	mW
Continuous Discharge Rate	25C	-

Air Data Rate	250	kbps
Standard Interface:	UART	-
UART Baud Rate:	57600	bps
Used Firmware:	SIK Telemetry Radio 1,9	-
Data Protocol:	MAVLink	-
Weight:	15	g

Table 9. Model Components: RC Control Specifications

Parameter	**Value**	**Unit**
Transmitter		
Model	Taranis X9D Plus	-
Operating Frequency	02.jaan	GHz
Number of Channels	16	-
Operating Voltage Range	juuni.15	V
Maximum Operating Current	260	mA
Maximum Transmitting Power	100	mW
Configured Transmitting Power	10	mW
Receiver		
Model	X8R 16ch Receiver	-
Operating Frequency	02.jaan	GHz
Number of Channels	16	-
Operating Voltage Range	04.okt	V
Operating Current	100 (@5V)	mA
Operating Range	1,5	km

Table 10. Model Components: 3DR Pixhawk Flight Control System

Parameter	**Description**
3DR Pixhawk	
Processor	
Main Processor	32bit STM32F427 Cortex M4 core with FPU
Failsafe Processor	32 bit STM32F103 failsafe co-processor
Operating Frequency MHz	168
RAMKB	256
Flash Memory MB	2
Sensors	
	ST Micro L3GD20H 16-bit gyroscope
	ST Micro LSM303D 14-bit accelerometer/magnetometer
	Invensense MPU 6000 3-axis accelerometer/gyroscope

	MEAS MS5611 barometer
Interfaces	
	5× UART, 1× high-power capable, 2× with HW flow control
	1× CAN with internal transceiver
	1× CAN on expansion connector
	Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
	Futaba SBUS® compatible input and output
	PPM sum signal input
	RSSI (PWM or voltage) input
	I2C, SPI, 2× ADC inputs
	Internal microUSB port and external microUSB port extension
External	
	3DR uBlox GPS with Compass Kit 5Hz

2.4 Avionic System: Autopilot and Software

Our Avionic system consists of Pixhawk flight controller and APM software which coordinates all the hardware components on board in an appropriate sequence. Figure 9 explains the framework of an avionic software system, where each task is shown as a block.

Navigational data control task manages collection of sensor data. Motor control task generates appropriate motor control signal to drive the motors. Communication control ensures communication between avionic system and ground control system application. Data logging provides log of in-flight data. Flight control implements the automatic flight control laws. Main control block manages all tasks.

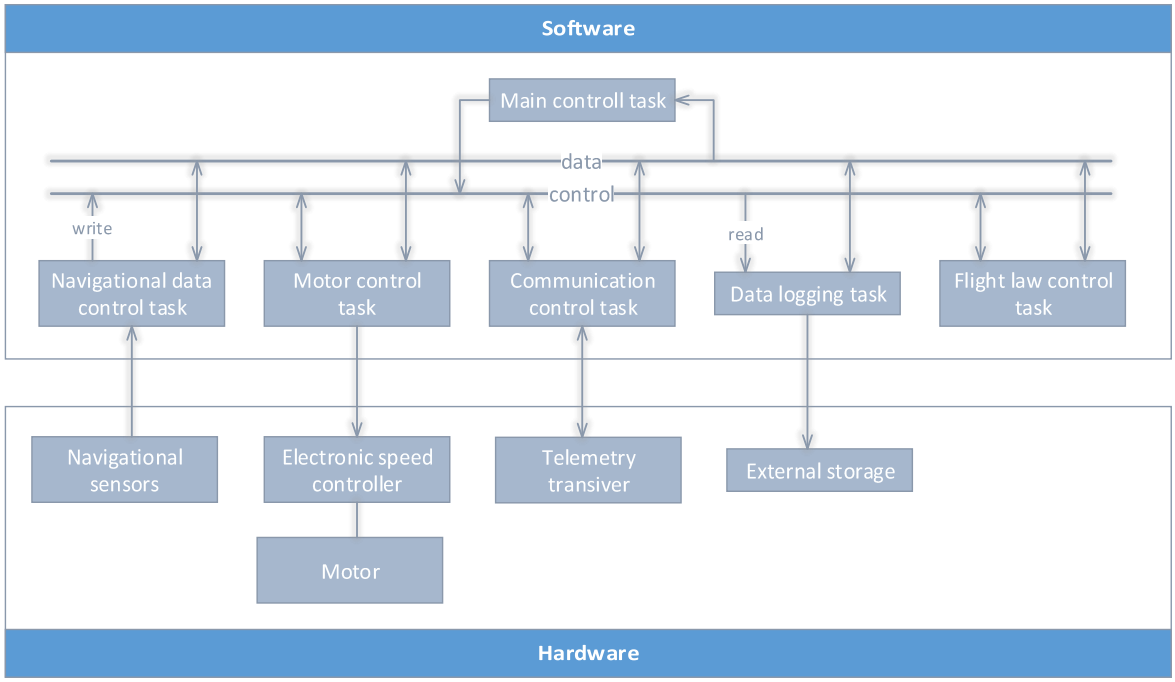


Figure 9. Avionic Software System Framework Block Diagram

Important requirement for avionic system selection for research and development is an open software and hardware platform, widely active community and up-to-date software development in progress. Pixhawk flight controller is mainly used for high-end research and amateur usage, can be considered as new but mature platform.

Pixhawk project is a further evolution of the PX4 flight controller system [22]: a single board controller having powerful 32-bit processor with an additional failsafe backup controller and extensive memory. Board is equipped with IO interfaces, refer to Table 10, and advanced sensor profile: (i) 3-axis 16-bit ST Micro L3GD20H gyroscope for determining orientation, (ii) 3-axis 14-bit accelerometer and compass for determining outside influences and compass heading, (iii) external HMC5883L kit: magnetometer and GPS unit, (iv) MEAS MS5611 barometric pressure sensor for determining altitude, (v) voltage and current sensing for battery condition determination.

System is optimized to provide control APM flight navigation software with high performance and capacity. For vehicle control APM open source flight stack is used, licensed under GPLv3 [23]. It is actively developed and has large community. Flight

software runs on NuttX real time operating system, which features high performance, flexibility, and reliability for controlling any autonomous vehicle [24].

Basic structure [25] of the flight software stack and the configuration is shown on Figure 11. *APM Flight Stack* is responsible for state estimation and flight control. *PX4Firmware* is base middle-ware and driver layer for Pixhawk board, licensed under BSD [26]. APM flight stack interfaces through *Hardware Abstraction Layer*, which makes APM software portable for Pixhawk board. Libraries block represents structure of essential libraries such as core, sensor and other libraries [27].

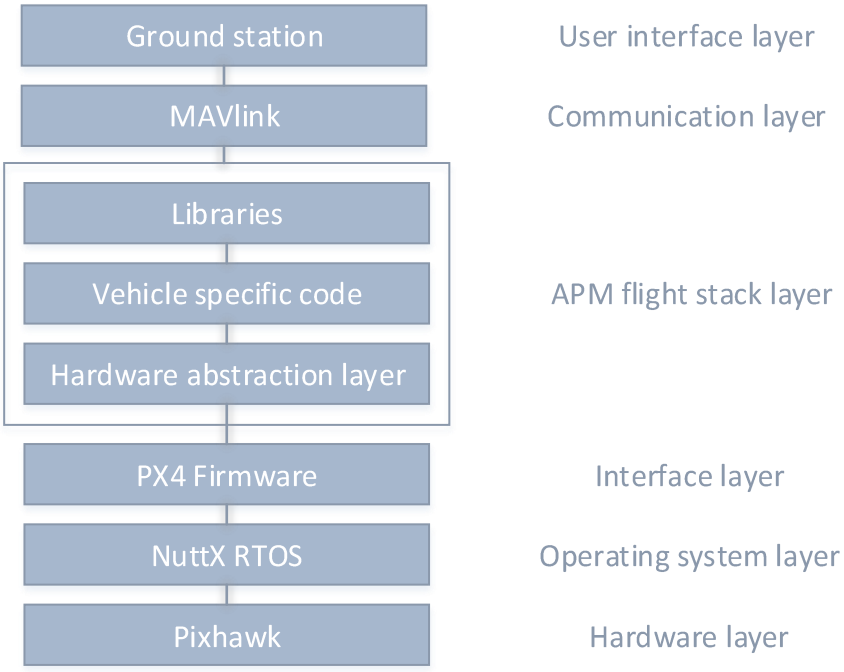
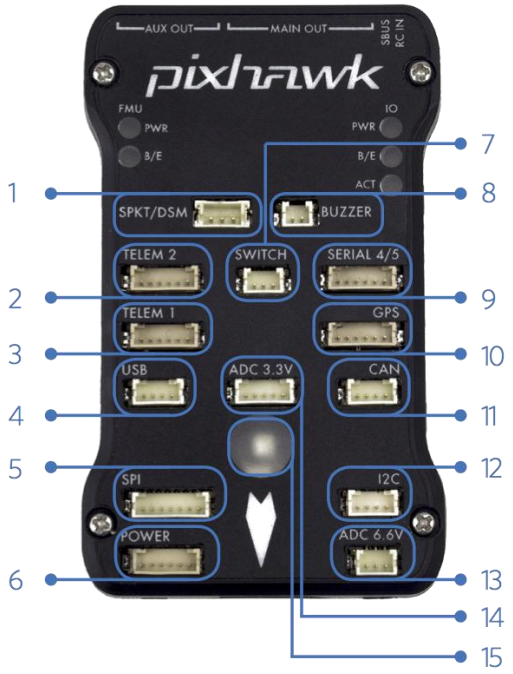
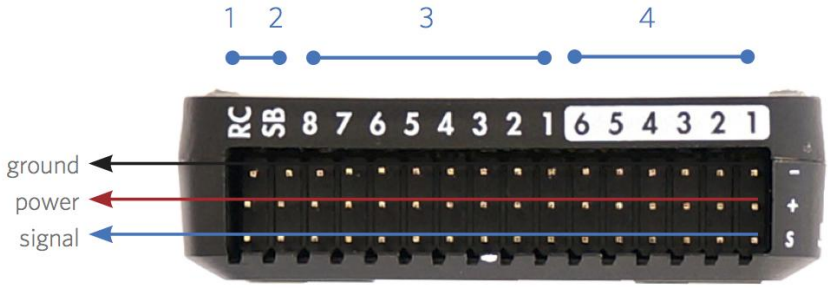


Figure 10. APM configuration.



1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I²C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator

Figure 11. Pixhawk flight controller



1 Radio control receiver input
2 S.Bus output
3 Main outputs
4 Auxiliary outputs

Figure 12. Pixhawk flight controller's output

2.5 Quad-rotor's Setup Routines and Flight Results

Setup Routines

Setup for quad-rotor includes proper assembly of the system as well as installation of ground control application. Software is used as a configuration utility and dynamic control application for autonomous vehicle, proper setup and configuration of quad-rotor is done after full assembly (I will include here the link of the setup file for our drone).

Ground control application [28] allows to (i) load the firmware into flight controller; (ii) setup and configure the vehicle for an optimum performance; (iii) create, save and load autonomous mission into autopilot; (iv) download and analyze mission logs created by flight controller software; (v) use telemetry link to monitor vehicle's status while in operation, record, view and analyze telemetry logs.

After assembling the vehicle, firmware must be loaded to the board and mandatory initial setup must be completed for achieving best performance for vehicle's operation. Initial setup requires setting up the frame configuration for mapping the motors for software. Proper flight modes are configured supporting different types of flight stabilization, autopilot, follow-me system etc. Most used modes are *Stabilize*, *Altitude Hold*, *Loiter*, *Return-to-Launch*, *Auto tune*, *Follow-Me*, *Guided Mode* etc.

For advanced configuration, *auto tune* functionality helps to automatically configure control loop feedback mechanism (PID gains) parameters of stabilizing algorithm for flight controller for providing vehicle's highest control response without significant overshoot. Auto tune functionality can be triggered manually while vehicle is operating in the air.

To ensure safety, failsafe mechanisms [29] are set up for geo-fence, battery, radio and ground control application to be triggered when vehicle is in operation, if needed. Flight software supports return-to-launch or landing functionality in cases where contact between RC transmitter and flight controller's receiver is lost. Failsafe for battery can be set up to trigger return-to-launch or landing functionality when battery voltage has crossed below configurable threshold. Geo-fence failsafe ensures that vehicle will remain in desired area, if flying too far away from allowed area, failsafe will be triggered switching vehicle to return-to-launch or land.

Pre-arm safety check is enabled for safety routines to be analyzed before airborne. These checks will prevent vehicle from arming if any problems are discovered including missed calibration, configuration or a bad sensor. Failure messages may include device failures or mis-calibrations with RC, magnetometer, GPS or problems with accelerometer or gyroscope [30].

Calibration of sensors must be done before first flight, which include calibrations of: (i) magnetometer; (ii) RC device; (iii) accelerometer; (iv) ESCs. Electronic speed controllers are responsible for spinning the motors at the speed requested by the autopilot. Most ESCs need to be calibrated so that they know the minimum and maximum control values that the flight controller will send [31]. Before each flight it is recommended to perform calibration of magnetometer, other devices should be calibrated optionally if needed.

Flight Results

Smooth flight. With our configuration of weight of 1732 grams, quad-rotor is capable of smooth flight with payload for around 11,5 minutes, as seen from Figure 13. Flight was conducted in strong winds using altitude hold mode, holding desired altitude of 1,2 meters. Altitude hold mode uses barometer sensor readings, as readings may vary in different weather conditions, it can be seen that altitude change was between 1,0 – 1,5 meters, with peaks of 2 and 0,8 meters.

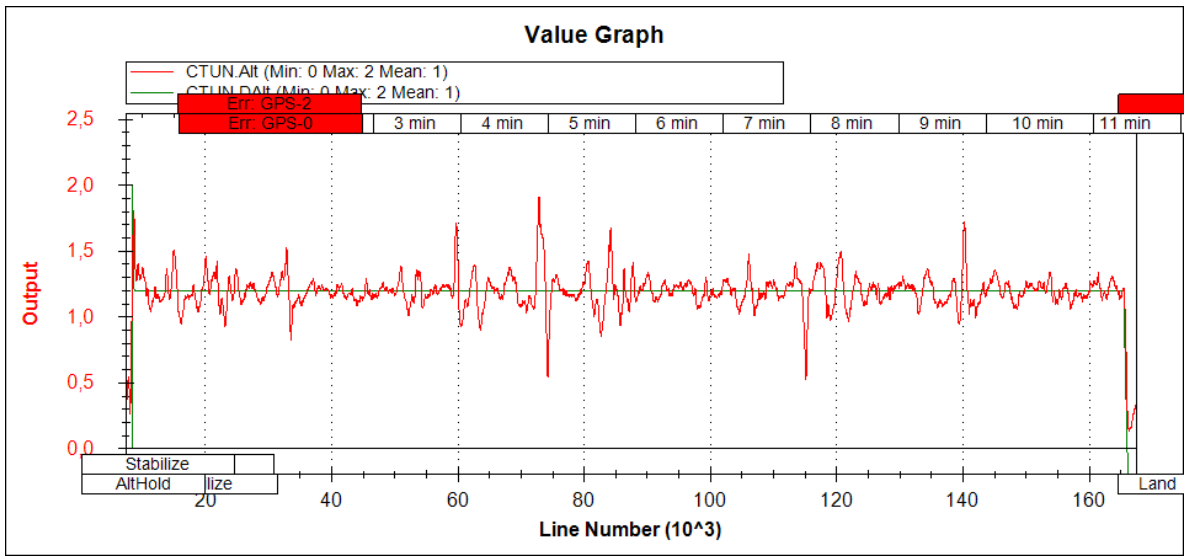


Figure 13. Flight results. Smooth flight. Altitude Changes

Rapid flight. Test results in flight time of 9,5 minutes, see Figure 14 and Figure 15. Flight was engaged in moderate weather conditions in auto-tune mode, which is used for vehicle's advanced configuration. Figure 14 shows altitude of the vehicle. Although barometer sensor readings, see red line, are sensitive due to moderate weather conditions, quad-rotor holds its desired altitude. During the flight, desired altitude is set to 3.5 meters, see green line, resulting real altitude to change minimally, see blue line.

Figure 15 shows the functional operation of auto-tune mode. For first eight minutes, vehicle was tilting the roll axis, see red rapid lines. Last two minutes were used to tilt pitch axis, see blue rapid lines, until the control response became sufficient. In the beginning, between second and third minute, see less rapid red lines, the tuning operation was manually paused because of the strong winds.

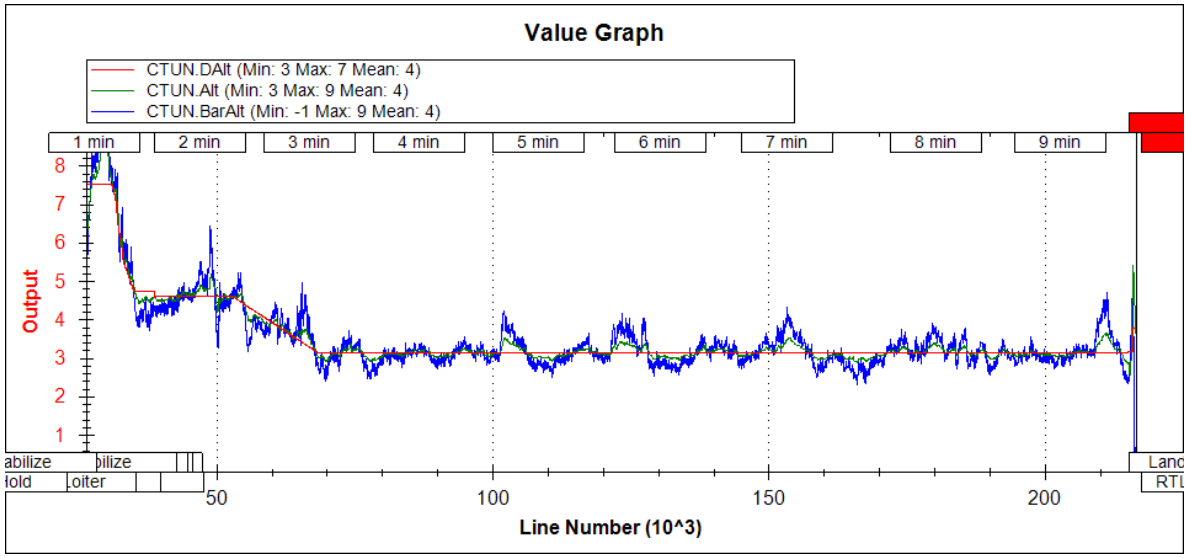


Figure 14. Flight results. Rapid flight. Altitude Changes

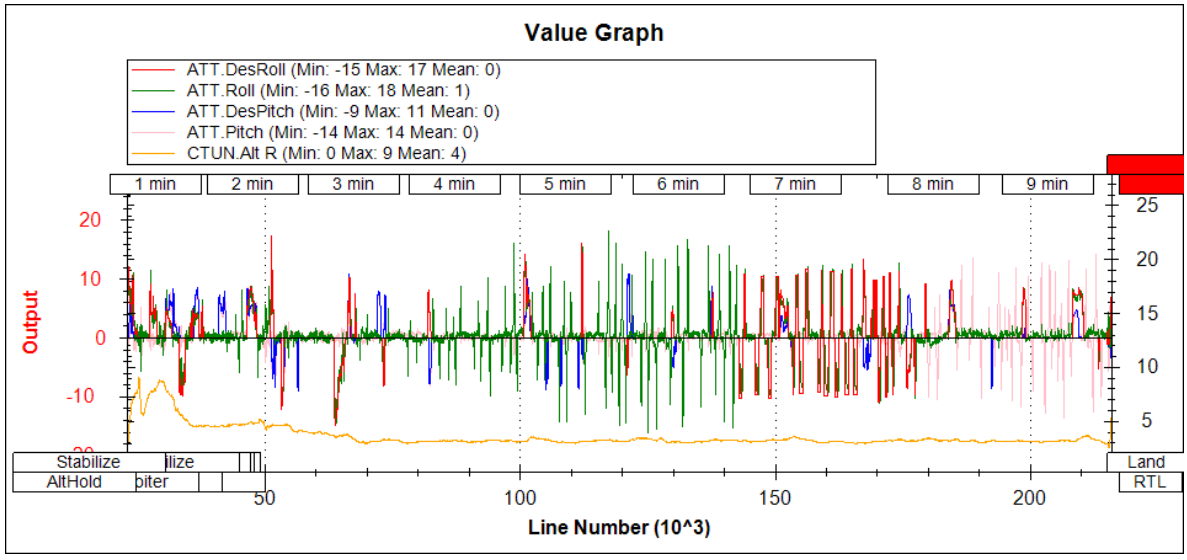


Figure 15. Flight results. Rapid flight. Roll and Pitch Axis Changes in Autotune Mode

Guided mode. Flight test was conducted in guided mode to invoke automatic mission execution. Mission consists of ten independent way-points, planned holding altitude of 7 meters with 5-second delays between every way-points. Execution time of the flight mission was 3 minutes.

Figure 16 represents the way-point data and real passed trajectory by quad-rotor. Point precision was set to 1 meter in the mission configuration, so all way-points were taken accurately.

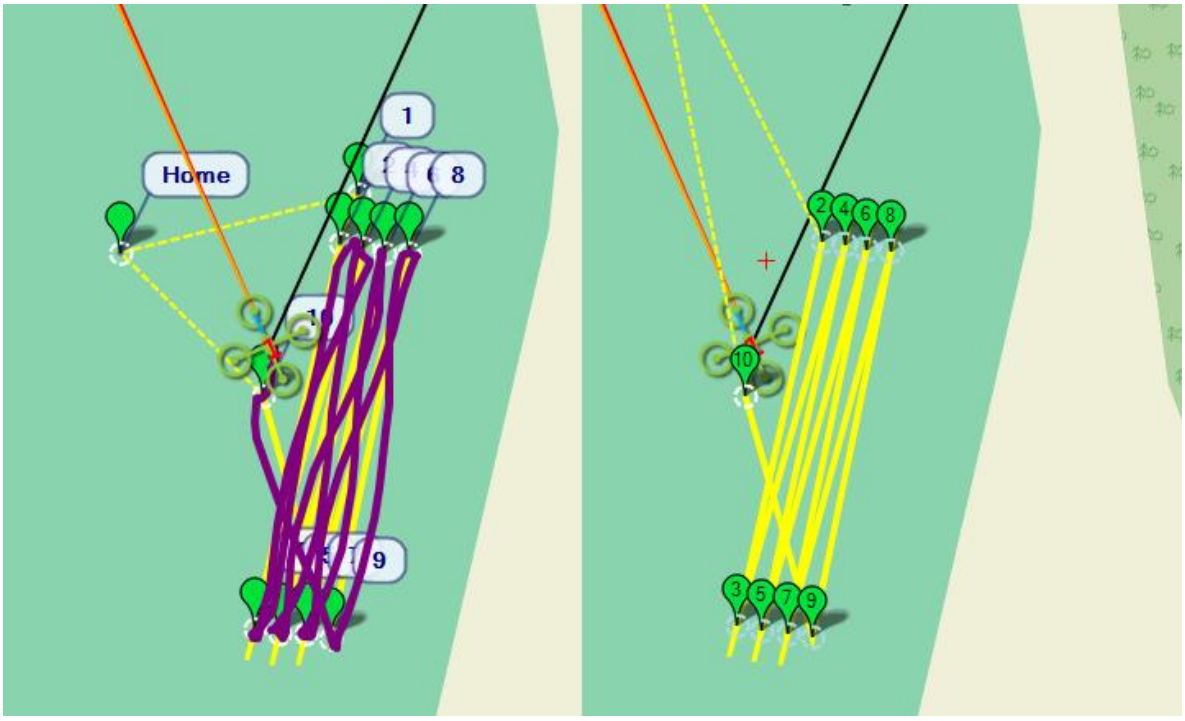


Figure 16. Flight Results. Guided Mode. Way-points and Real Trajectory

Maximum flight target speed between way-points is set to 500cm/s (18km/h) as an internal parameter. Such flight speed is successfully carried out by quad-rotor, see Figure 17, Figure 19 and Figure 18.

Figure 17 shows constant battery voltage drop. During three minutes of mission execution, battery voltage dropped from 12.4V to 10.9V. Red line represents battery voltage, green line shows horizontal speed of quad-rotor.

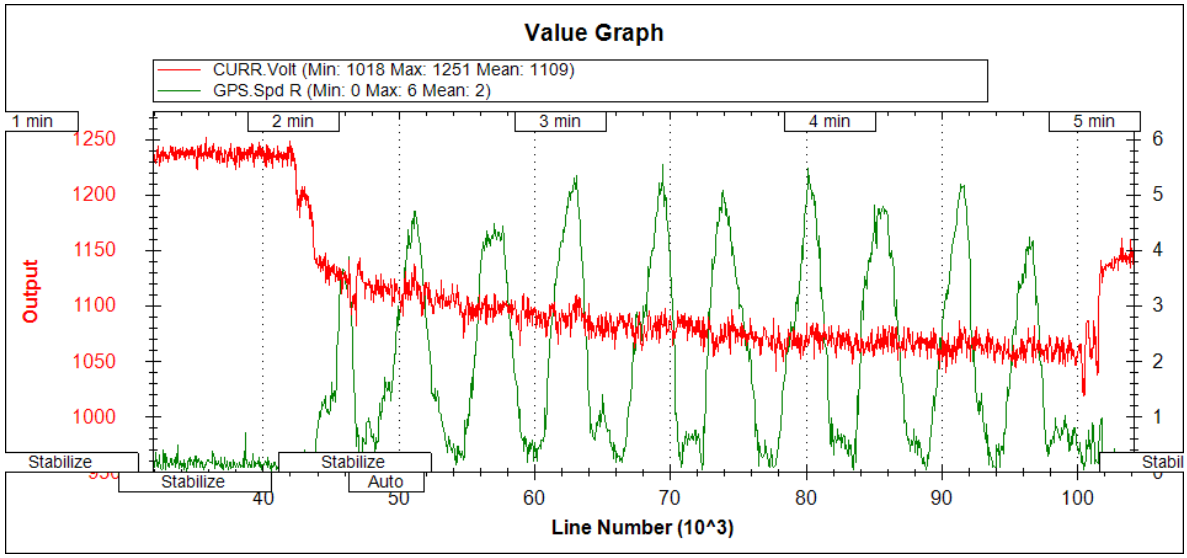


Figure 17. Flight Results. Guided Mode. Battery Voltage Change

During the flight, current consumption was between 22A - 25A. After reaching certain way-point, 5-second delay is invoked. Figure 18 contains small segment of **Error! Reference source not found.** - here it can be seen that speed remains around 0 m/s during 5 seconds, see orange line.

After the delay, quad-copter reaches target speed of 500cm/s (5m/s) with constant acceleration, see purple line. During acceleration, slight overall current consumption raise can be noticed, see Figure 18 and *Appendix* **Error! Reference source not found.**. Yellow bar represents an area, where constant acceleration takes place, current consumption rises around 2,5A, such trend takes place upon every acceleration as can be seen from *Appendix* **Error! Reference source not found.**.

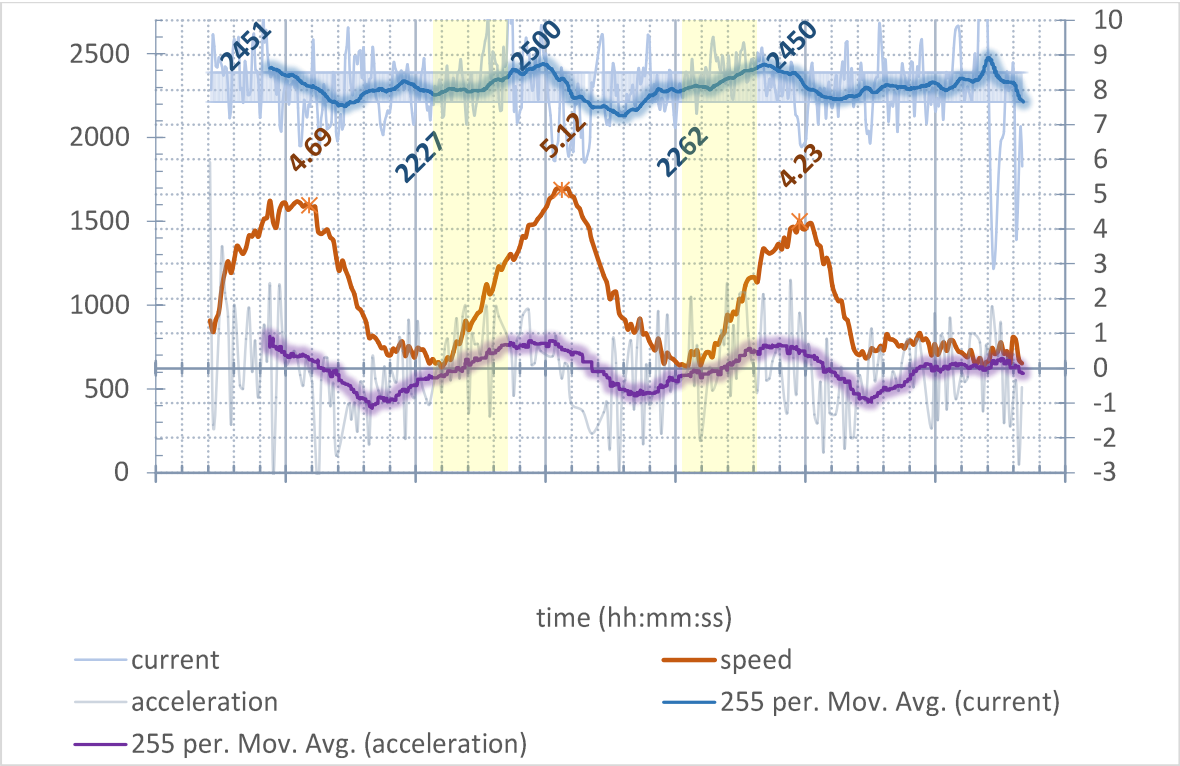


Figure 18. Flight Results. Guided Mode. Current Consumption vs Flight Dynamics

Figure 19 shows mission's altitude. Green line represents desired altitude set beforehand. Red line shows barometer's sensor-readings, which may vary in weather conditions, blue line represents real altitude of the vehicle. Real altitude is calculated fusing barometric sensor data and GPS readings. Real altitude differs slightly from desired altitude. Changes in altitude can occur due to constant speed changes, see white line.

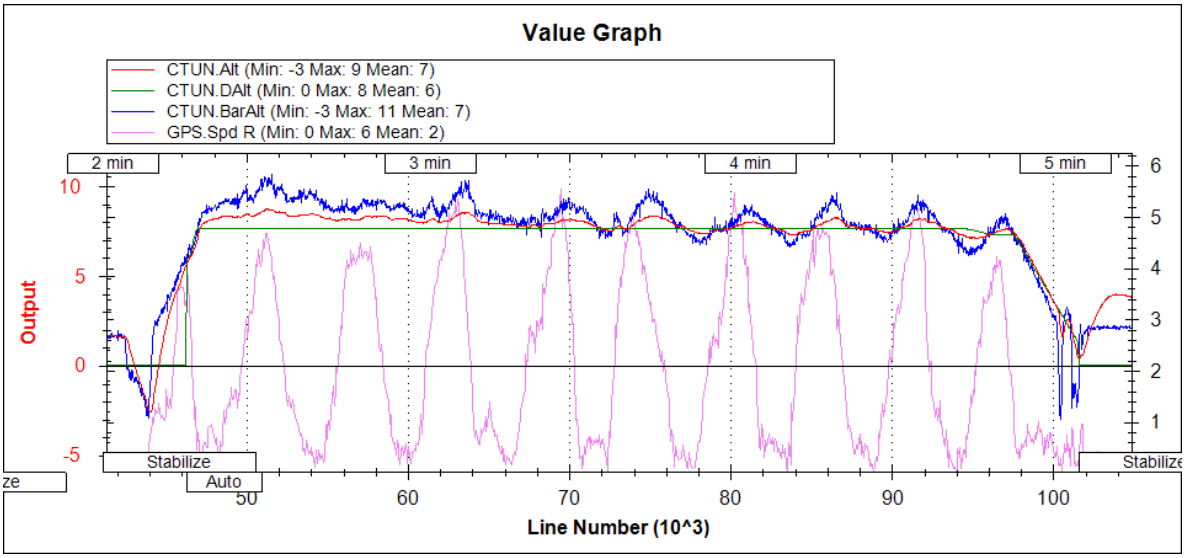


Figure 19. Flight Results. Guided Mode. Altitude Changes

3 Environmental Mapping

For an autonomous behaviour, the mapping and localization of the UAV in the 3D space is an important component. With a LIDAR-based approach, the UAV senses the environment by taking 3D-range measurements (see Figure 20). Storing the raw measurements is not reasonable, therefore suitable modelling framework to represent the environment needs to be considered. Localization in the environment is an important issue, which should not be underestimated. In this project, GPS-signal is expected for localization implementation.

Based on a specific modelling framework, the map of the space is created. Such map is considered the central component for autonomous operations as it will be used for path-planning and navigational operations. Such map has to be effective and efficient in respect to access and size, so the large outdoor environments could be mapped.

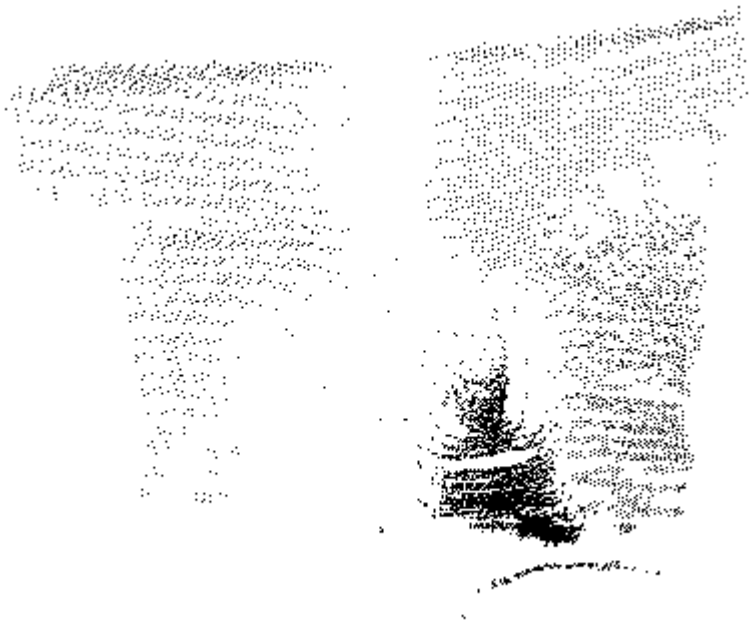


Figure 20. Example of generated point cloud by a LIDAR System.

For our configuration, following requirements are set: we would use a probabilistic representation for modelling the occupied, free and unknown space in addition with optimum runtime and memory usage. The modelling framework has to be capable of transforming environmental readings to an environmental map. We are looking for an implementation from open-source project that could be refined to specific needs for this project.

3.1 Environmental Mapping Models

Although several environmental modelling concepts are available, lack of finalized, working and successful implementations leads to barrier of using such concepts.

POINT CLOUD MODELS. Point clouds (PCL) are very precise and are proved to be used in static environments, however these models are not memory efficient. Large raw point cloud data, that is stored without organizing nor segmenting readings into structures, introduce modelling and computational problems. Several measurements for the same space-segment can exist with such approach. With the growing amount of raw readings, the representation of the model increases with no upper-bound. Without structuring and processing the data, it is impossible to differentiate between obstacle-free space. Such models have no information about unknown nor free areas.

ELEVATION MODELS. These structures store the height information in each cell of a discrete grid of the surface. These models provide maps of discretization the space in vertical dimension, not the actual volumetric representation. Whereas the elevation maps provide a compact representation, they lack the ability to represent vertical structures on multiple levels [32]. Upper surface of the environmental space for a specific height is stored on such maps and useful for the navigational tasks for ground [33] mobile vehicles. These models do not have full distinction between free and unknown space, they may have large memory consumption, particularly outdoors. With 3D precise dataset, precision will be lost as the map of the surface does not represent an actual space.

OCTREE MODELS. Probabilistic octree-based models avoid one of the main shortcomings of the fixed grid-map structures. Octree structure can be used as a multi-resolution

representation -- structure contains multi-node elements for obtaining coarser subdivision. OctoMap [34] is an octree-based framework, that is able to address large point clouds and integrate measurements into memory efficient volumetric occupancy map. Octree structure represents spatial subdivision in 3D, represented as a voxel (cubic volume). Structure is divided into eight substructures until minimum cube size is reached, which defines the resolution of a structure. This structure can be decreased at any level, allowing to have a coarser subdivision for obtaining another resolution. Different resolutions, as can be introduced on Figure 21 and Figure 22**Error! Reference source not found.**, rendered with OctoVis tool from publicly available model [34], [35].

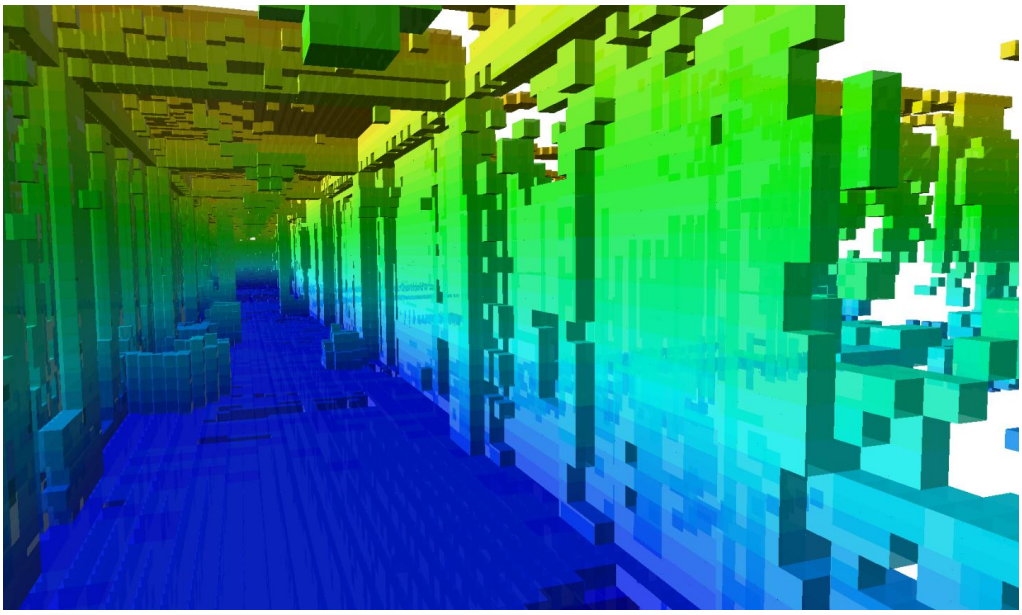


Figure 21. OctoMap corridor vizualization at resolution 10cm [35].

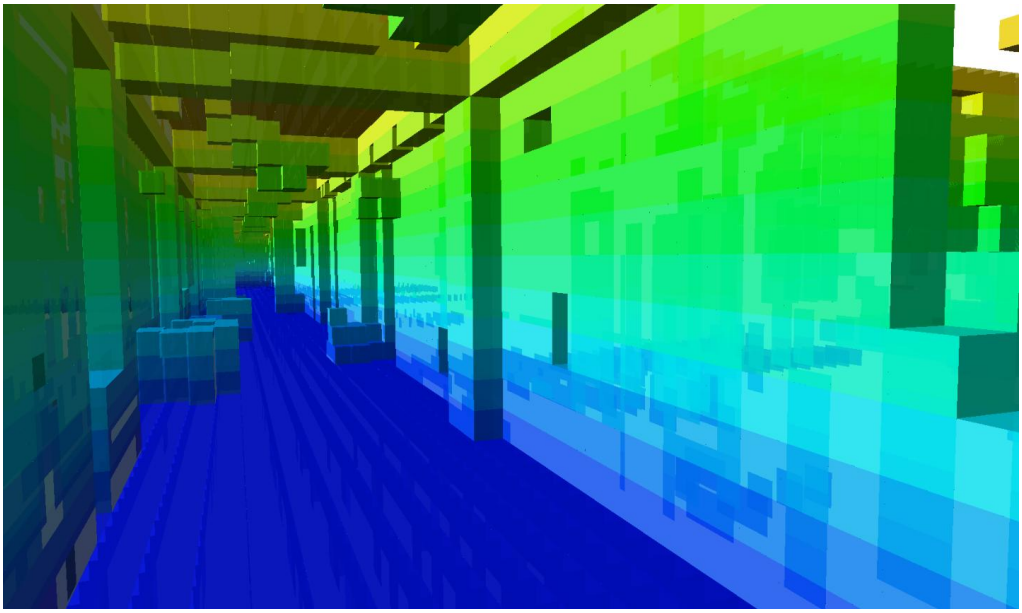


Figure 22. OctoMap corridor vizualization at resolution 20cm [35].

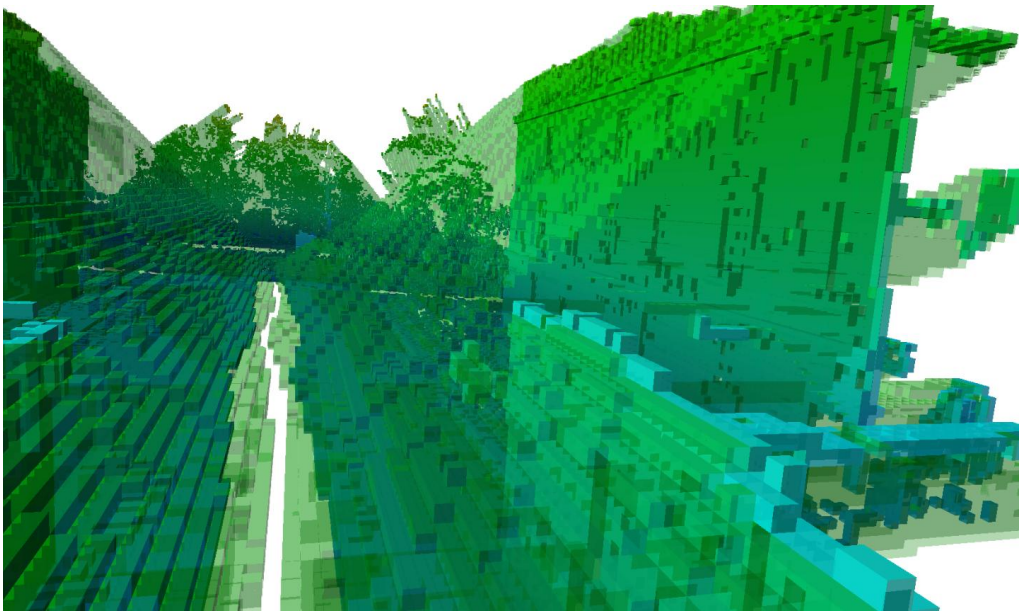


Figure 23. Corridor Visualization, Cube 20 cm. Occupied; Free; Unknown Space [35].

OctoMap modelling framework is an implementation for the octree model, which uses tree-based representation for modelling the environment. This approach uses probabilistic occupancy estimation to ensure updatability and cope with sensor noise. Method provides compactness on the resulting models as Boolean occupancy states are used. If certain space

is considered occupied, the node in a structure is initialized. Unoccupied volumes in the space are also represented. Figure 36 included to Appendix, represents OctoMap visualization for an occupied space and Figure 23 shows occupied, free and unknown areas.

For the compact implementation structures are maintained as follows: if all substructures of a node have the same state of probability parameter, e.g. occupied or free, the structure is pruned. OctoMap ensures that confidence of the map remains bounded and the model itself can adapt changes in the environment quickly. Compression is not completely lossless in terms of full probabilities, as structures with close thresholds ([0; 1]) are considered as a stable in high confidence and pruned. In between the thresholds, full probabilities are preserved. In static environments all voxels will converge to a stable state after a sufficient number of applied measurements. In such cases, as all substructures get the same occupancy state, they are pruned from the main node. When new measurements are applied that would conflict the corresponding inner-node, the sub-structures are accordingly regenerated and updated.

OctoMap framework is available as an open-source BSD-licensed C++ library and has been successfully applied to several robotics projects. There are publicly available real-world datasets that can be used for simulation and testing while implementation process takes place. OctoMap approach is able to update the environmental representation efficiently keeping the memory requirements at a minimum.

4 Strategic Outline of Path Planning

In this part of the thesis a strategy for an appropriate path planning method for the developed and tested UAV platform will be investigated. First, based on a final decision for an efficient data presentation using OctoMap [1], algorithms for path planning will be compared. Here the following major constraints and goals for Mission Path Planning (MPP) have to be considered:

- Optimum path result based on a defined cost function.
- Initial cost function: flight distance only (later: speed, energy, …).
- Path planning method must be capable of dynamic replanning in case of occurrence of OctoMap changes (or even cost function changes (e.g. dynamically modified distance/power/speed trade-off!).
- Short path re-calculation times/re-planning times, since energy plays a role and copter cannot "stand" in the air too long, until computation is finished (→ computational efficiency).
- Memory efficiency of the MPP algorithm, since limited memory resources are available in the computational unit of the UAV.
- Optimum re-planning efficiency in combination with OctoMap models.
- Currently no detailed mathematical model on flight dynamics is available: path planning method has to be model-independent from flight dynamics.

Furthermore, the envisaged Path Planning Algorithm should be ready for the following intended novel contributions to path planning methods:

- Minimized service completion times for UAV missions: requires consideration of flight speed (which is depending on corridor width and straightness of the flight corridors, e.g. minimization of directional changes).
- Power optimization (minimization of acceleration, deceleration, height changes; minimization of overall flight time).

4.1 Introduction

Careful and safe navigation is most important requirement for unmanned aerial vehicle. This is to be achieved by careful path planning. For path planning it is necessary to understand the environment from real world model.

107: Exploration module uses computed 3D map for autonomous functionality: trajectory planning, later navigation and exploration.

For this project one should carry out careful navigation route minimizing constraints that are important to this projects. Paths are to be optimized for speed and distance due to limited battery life and time vehicle can fly - so routes must be energy efficient. How to achieve one needs to understand from vehicles energy model. Speed efficiency can be achieved from map - to pass only big cubes which are empty.

Speed efficient: trajectory in in very detail: big cubes which are empty. Distance: Optimal. Cost function: speed vs distance. and then think which model is more optimal. Rough step model: discrete flying model to sum up energy consumption - slow down/accelerate costs more energy. Cost function for operational tasks we have to do. And then find optimal trajectory minimizing the cost in some constraints (for ex time). Parameters for cost function and to define cost function.

Optimization for path:

1. Speed (trajectory in in very detail: big cubes which are empty)
2. Time (constraint)
3. Distance (optimal)
4. Energy point (limited battery life, more time to be on air). Energy consumption: slow down/accelerate costs more energy vs flying most time at just hovering?
5. Precision (precise vs not preside) + time
6. Acceleration model?

To think on this topic, you should start for thinking on distance and speed and then refine this to energy model (some energy cost function). Cost function: speed vs distance. and

then think which model is more optimal. Cost function for operational tasks we have to do. And then find optimal trajectory minimizing the cost in some constraints (for ex time). As long as scanning takes = route can be calculated (test).

Algorithm selection: Theoretically calculate how much time we have for path planning (theoretical numbers how long could use that) and then find algorithm, which fit into that and which does not fit. And also based on processing power of the Raspberry CPU consumption. Constraints have to be good enough!

Localization: is the process of determining where the robot is located, relative to objects in its environment.

Mapping: is the process of building maps based on data acquired from one or more sensors. Once the map is built, the robot can begin performing its navigation and obstacle avoidance functions. Navigation is comprised of path planning to reach a goal within a map, along with detection of new obstacles that are not part of the map. This is where the preceding elements of LiDAR, scanning, and mapping are all brought together into a functioning system. During navigation, the software provides real-time updates to the proposed path to keep the robot within the mapped area, whilst avoiding any new obstacles. In the picture below, we see a navigation 'goal' being set. The user clicks a location on the map where they would like to robot to go, and the navigation software displays a large green arrow to show this goal. In deriving a path to the goal, the robot must allow a safe distance around all obstacles. To this end, the purpose of obstacle inflation is to make obstacles appear larger than they are, and with a safety margin, to help ensure the robot does not collide with them. So the turquoise areas are areas the robot should avoid: they are the danger spots where collision could occur. During path planning, the robot footprint (in red) should not be made to overlap significantly with those inflated areas.

Path planning: is the process of determining a path for the robot to follow, in order to reach a 'goal', whilst avoiding obstacles (a goal is just where you want the robot to go).

Obstacle detection: is the process of detecting objects in the environment that were not present during the mapping process, but are now nonetheless present.

Avoidance: is the process of path planning around dynamically occurring objects.

- Purpose of this work is to investigates path-planning algorithms that can be realized for navigational operation for this project and represent ideas behind path planning strategies. The main idea of our project is to have experimental fling platform that can autonomously navigate in an environment where might have multiple, mainly not dynamic, obstacles.

Before executing the mission, the system has to perform the initial environment scanning even if there is some map available so that it can generate new map or update old one. After initial scanning, depending of current location, there might still remain some unknown areas what perception system isn't able to scan. After the initial scanning the representative result can be seen in Figure 24 (a and b) where light green represents the fully scanned and free area and white represents unknown or not scanned area. On that figure blue drone-model represents our experimental fling platform with a perception system on-board. Dark grey squares represent obstacles in environment and bright green dot represents the goal location. On a Figure 24(b) light transparent grey squares represent occupied voxels on a map and those are out of limit for UAV to fly. Variety of blue tones squares on a map represent partially scanned voxels what can hold weight values, for example free voxel has a value 1 and occupied has a value 0, then squares represented as blue have values in between. For simplicity the idea in this work is that if voxel is scanned more than 90% and obstacle is not detected there, it is considered as free voxel. Initial location where UAV is located at start, is considered as free. Must be mentioned for clarity that all figurers describing environment has illustrated in 2D instead of 3D.

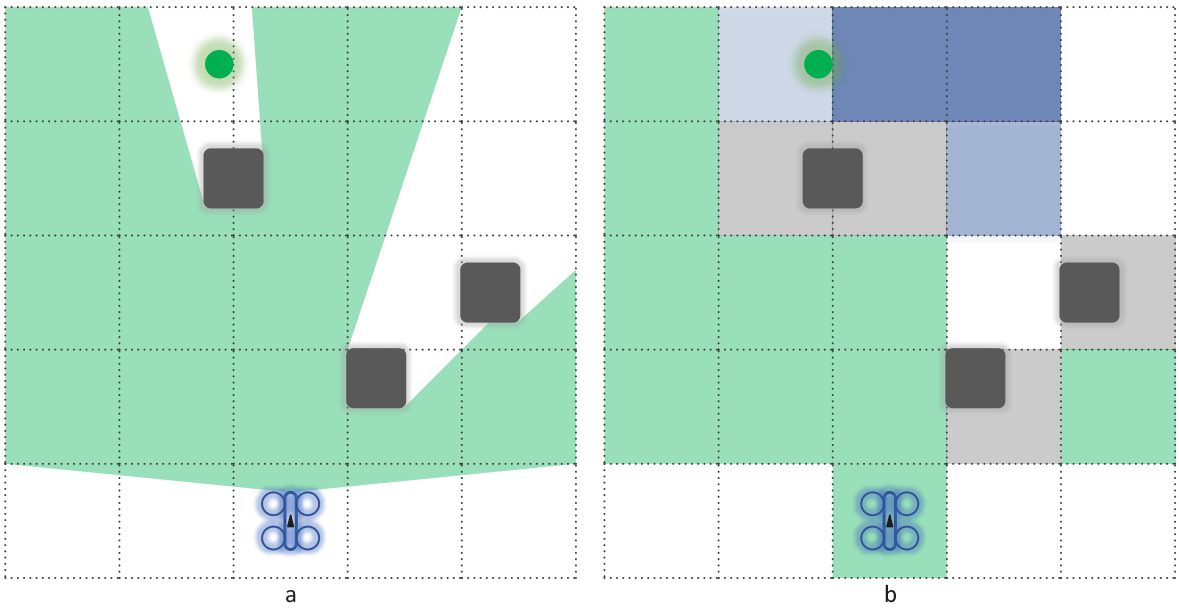


Figure 24. Environment scanning and map generation.

Path will be given to autopilot step by step in form of waypoints. Autopilot what is used in this UAV supports communication protocol called MAVLink [36]. This protocol supports sending waypoint coordinates with its radius from on-board computer to autopilot. Coordinates is calculated from the map and represents the centre point of the voxel. Radius of the waypoint means that UAV does not reach exact point of that coordinate before targeting next one but starts fling towards next waypoint before reaching to the current one. This kind of possibility gives an easy way to construct a smooth flight. In Figure 25 is illustrated how different radius size affects flight smoothness.

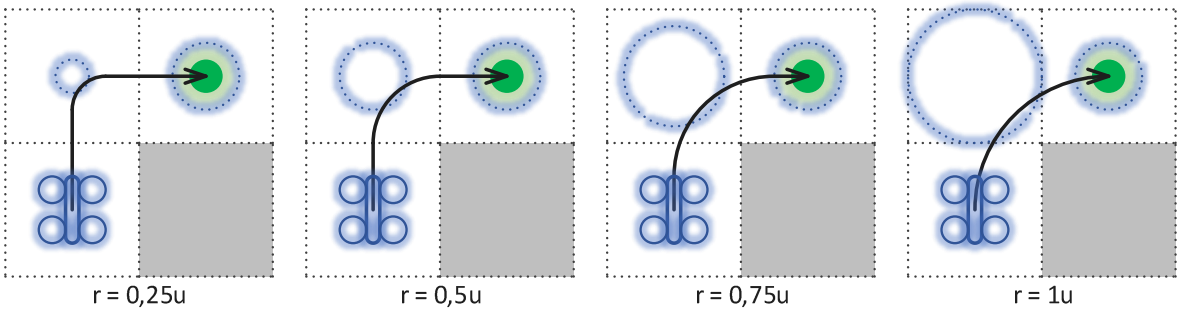


Figure 25. Meaning of the waypoint radius.

Test flights have shown that the result can be seen in Appendix at Figure 35, that rapid and sharp turns increase current consumption noticeably therefore, smooth flightpath is preferred.

Because of the sensor-system what is used for this platform [1] is relatively slow, scanning for fast moving objects and overall scanning takes time, there are some restrictions.

Firstly, while taking a full-scan UAV does not move [1] thus it takes more time to accomplish the mission what is not preferred of the limited amount of battery life.

Secondly, the system does not able to sense whole environment, therefore the system does not have complete map and navigation must be conducted in partially-known environment.

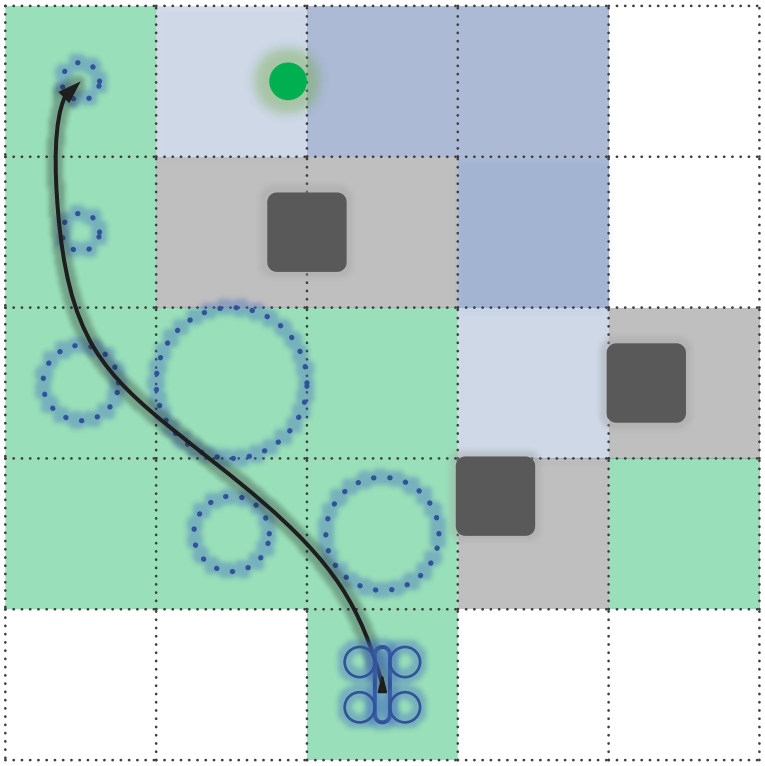


Figure 26. Representation of one possible calculated route with waypoints

Waypoint radius defines the radius (in meters) of all waypoints in the mission. The radius of the waypoint determines the size of the waypoint in three dimensional space and controls how close to the specified location the vehicle needs to be before continuing to the next waypoint.

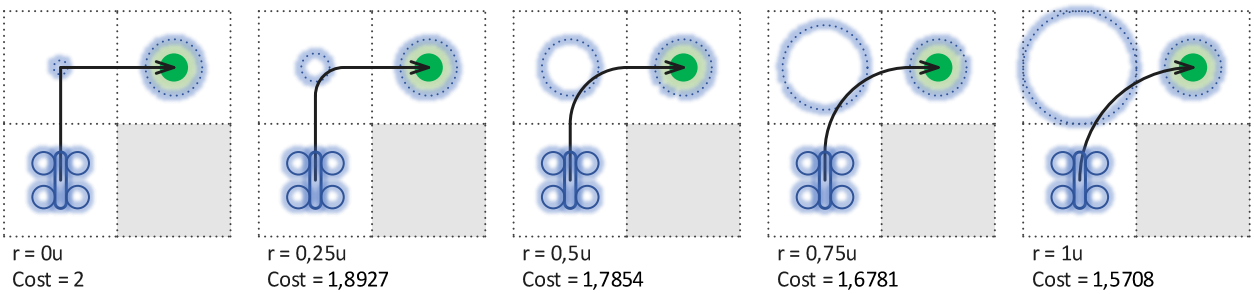


Figure 27. Cost of the different paths

4.2 Path planning algorithms

According to [37] several approaches exist for computing paths given some representation of the environment. In general, the two most popular techniques are deterministic, heuristic-based algorithms (Hart, Nilsson, & Rafael 1968; Nilsson 1980) and randomized algorithms (Kavraki et al. 1996; LaValle 1998; LaValle & Kuffner 1999; 2001). According to our platform restriction, capabilities and the fact that randomized algorithms, like Probabilistic Roadmaps (PRM) and Rapidly Randomly Exploring Tree (RRT), need complete map of the environment [38] we cannot use those and need to find required algorithm from many variety of heuristic-based algorithms.

4.2.1 Dijkstra

The Dijkstra algorithm is the most famous algorithm. The algorithm evaluates the moving cost from one node to any other node and sets the shortest moving cost as the connecting cost of two nodes. Initial node is marked as current state and all others is marked as unvisited.

Dijkstra algorithm woks so that it gives every node (state or vertex) infinite value except start node, it gets cost equal to zero. Then for a current node it calculates initial cost/distance for all of its unvisited neighbors, compares new cost to the current one and give it smaller value if exist. If all the neighbors of the current node have been evaluated, then current node is marked as visited and never checked again. Algorithm is finished when goal node is marked as visited or when there is no connection between start node and remaining unvisited nodes [39].

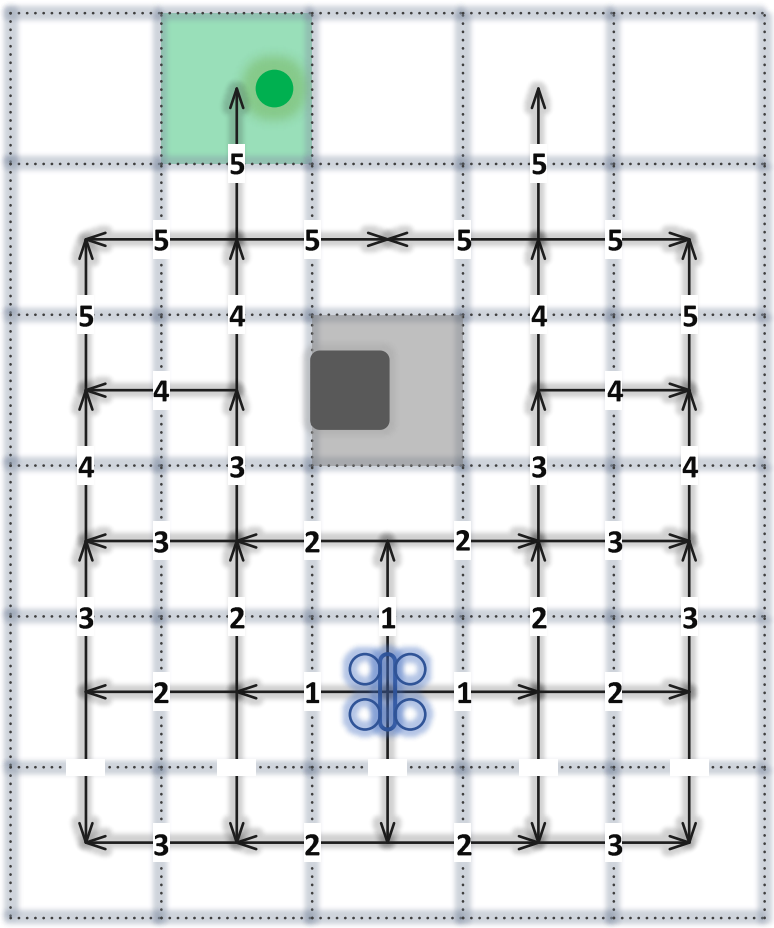


Figure 28. Concept of Dijkstra's algorithm.

Dijkstra algorithm is one of the first algorithm designed for finding shortest path [40]. It is simple by just oriented to find the shortest path between start and goal point. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark nodes as visited

when done with neighbors. But the simplicity is what makes it also slower than many others algorithms, like one of the most used algorithms A* and D*. Although it was superior to its predecessors [41]. Dijkstra is a special case of the A* when the heuristics parameter is set to zero.

4.2.2 A*

A* is one of the most used pathfinding algorithm today [37] [42]. It was developed by Peter E. Hart at 1968 to solve minimum cost path. The A* approach differs from other methods as it incorporates an estimate of the cost of path-completion. For certain classes of estimating functions, A* will find the optimal path [41].

A* plans a path from an initial point to a goal point just like Dijkstra's algorithm. But instead of starting to visit all neighbors around start position (in circular wave front) it have directed search to a goal point. That what makes A* algorithm potentially faster than Dijkstra's algorithm. At the beginning the A* sets Start state cost to zero and then places this state into a priority queue witch is called *open list*. Elements in this list is ordered by the sum of their path cost from start position and heuristic estimation of their path to goal position. The heuristic function estimates the minimum cost from any vertex to the goal and it is used to focus the search. [37]

Like seen in Dijkstra's algorithm there is lot of wasted moves while searching an optimal path to the goal position. Therefore, in A* algorithm there is introduced evaluation function which decide which way the expansion should move. Let say that the evaluation function is f(n) in that case f(n) = g(n) + h(n) [43] where g(n) is cost of an optimal path from start point to a current point and h(n) is an optimal path cost from current point to goal point. In this work the value on g(n) is equal to a traversed distance, like seen in Figure 30, where arrows show the path and the number on the arrow means the distance what is traversed. The second member of this function h(n) is presented in Figure 30 as a red arrow, in our case it represents the Euclidean distance between current point and goal point [43]. It can be calculated easily from coordinates of the goal point n_g and current point n as seen in (1). In our case the pathfinding takes place in 3D environment therefore we use three coordinates x, y and z. Given our environment restrictions, this formula gives good value for evaluating

node suitability as next waypoint and decreases number of nodes what must check for finding the path.

$$(\quad) \sqrt{(\quad) \quad (\quad) \quad (\quad)} \qquad (1)$$

$$- \qquad - \qquad -$$

This cost value of the path from current point to goal point is used in A* to calculate the optimal path from s to n_g and it is used to focus the search.

```
1    initialize the open list
2    initialize the closed list
3    g(s)=0, for all other nodes g(n)=∞
4    initialize goal node //this is the target node
5    initialize start node //add the node to the open
6    while open list is not empty
7    {
8         get node n from the open list with the lowest f(n)
9         add n to the closed list
10        if n is equals the goal node then stop;
11        return solution;
12        generate each successor node n' of n;
13        for each successor node n' of n
14        {
15             set the parent of n' to n;
16             //heuristic estimate distance to goal node
17             set h(n')
18             set g(n') = g(n) + cost from n to get to n'
19             set f(n') = g(n') + h(n')
20             if n' contained in open and the existing node is as good
               or better then discard n' and continue;
21             if n' is contained in closed and the existing node is as
               good or better then discard n' and continue;
22             remove all occurrences of n' from open and closed list
               and add n' to the open list;
23        }
24   }
25   //if we searched all reachable nodes
26   //and still have not found a solution then return
27   return failure;
```

Figure 29. A* algorithm pseudo code. [44] [45] [37]

On Figure 29 is shown pseudocode of the A* algorithm. In line 1 and 2 there is created two lists, open and closed list. In an open list will be stored all vertexes what can be visited. It is sorted based on f(n) value. All the others nodes initial value g(n)=∞, line 3. In line 8, algorithm will take the node *n* with smallest value of the f(n) from the open list and puts it in a closed list, line 9. If node taken from open list equals the goal node, then algorithm has found the path and is finished and returns solution, line 11. Otherwise algorithm continues

and tires to finds successors for this element *n*. Successors of *n* are the direct neighbors of that element which means that from *n* to neighbor *n'* it is possible to move with one step. Steps are illustrated in Figure 30 as black arrows. From line 17 to 19, f(n) value is calculated for each successor and added to an open list, line 22, in case existing node has better f(n) than *n'* then this particular neighbor will be elected as a part of the path.

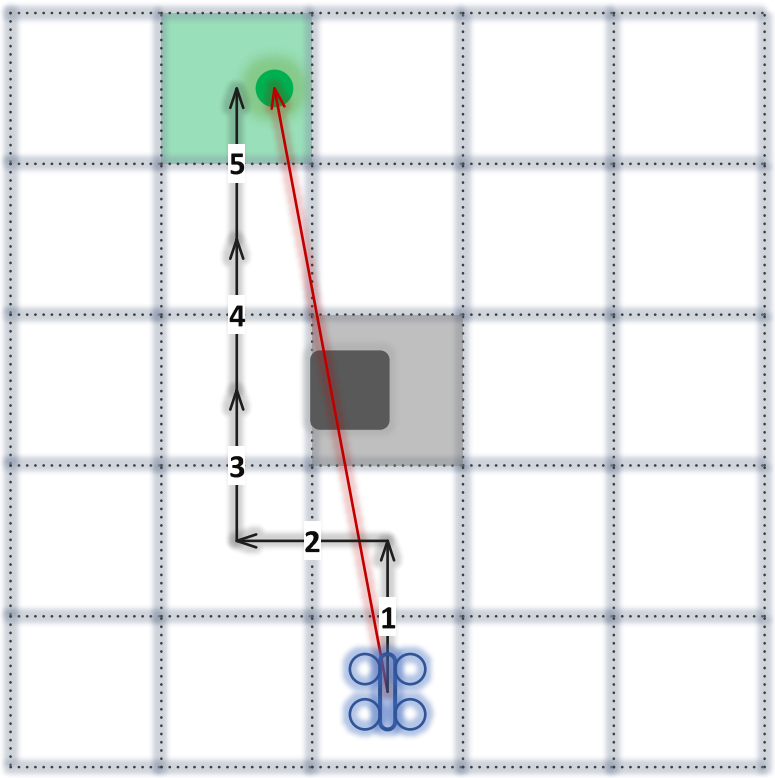


Figure 30. Concept of A* algorithm.

Advantage: delivers optimum path.

Disadvantage: static, if anything in the OctoMap changes, a full re-computation is needed.

4.2.3 D* Lite

D* algorithm is also called Dynamic A* algorithm, which is proposed by Stentz in his two papers in 1994 [46] and 1995 [47]. Instead of recalculating the full path from zero if environment changes like A* algorithm does, D* calculates only remaining part again. Compared to A*, D* and D* Lite is more efficient and therefore it is also more wildly used for a path planning tasks [48]. D* Lite algorithm mimics the D* algorithm but uses some properties of LPA* (Lifelong Planning A*) algorithm to find new route as UAV moves on

[49]. Because of these two algorithms (D* and D* Lite) are fundamentally quite similar, hereinafter the attention is turned to D* Lite, which has been found to be slightly more efficient [48].

D* Lite uses a heuristic to limit the states processed and process only those states that has influence on the path cost of the initial state. As a result, it can make D* Lite up to two orders of magnitude more efficient then A*. Also D* Lite stores a least-cost path from a start state to a goal state, for that it stores an estimate cost g(s) from the state s to the goal, look at a Figure 31 line 1 and 6. It also stores a one-step lookahead cost rhs(s) which is sum of the cost from state s to its successor state and cost of the path from successor to goal.

```
key(s)
1 return [min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s))];

UpdateState(s)
2 if s was not visited before
3    g(s) = ∞;
4 if (s ≠ s_goal) rhs(s) = min_{s' ∈ Succ(s)}(c(s, s') + g(s'));
5 if (s ∈ OPEN) remove s from OPEN;
6 if (g(s) ≠ rhs(s)) insert s into OPEN with key(s);

ComputeShortestPath()
7 while (min_{s ∈ OPEN}(key(s)) < key(s_start) OR rhs(s_start) ≠ g(s_start))
8    remove state s with the minimum key from OPEN;
9    if (g(s) > rhs(s))
10        g(s) = rhs(s);
11        for all s' ∈ Pred(s) UpdateState(s');
12   else
13        g(s) = ∞;
14        for all s' ∈ Pred(s) ∪ {s} UpdateState(s');

Main()
15 g(s_start) = rhs(s_start) = ∞; g(s_goal) = 1;
16 rhs(s_goal) = 0; OPEN = ∅;
17 insert s_goal into OPEN with key(s_goal);
18 forever
19   ComputeShortestPath();
20   Wait for changes in edge costs;
21   for all directed edges (u, v) with changed edge costs
22        Update the edge cost c(u, v);
23        UpdateState(u);
```

Figure 31. D* Lite algorithm basic version [48].

On Figure 31 is brought out pseudo code of the basic version of the D* Lite algorithm, it consists of four parts: key(s), UpdateState(s), ComputeShortestPath() and Main(). Thus D* Lite starts searching path backwards in contrast to A* from goal to start, then firstly $g(s_{start})$, $rhs(s_{start})$ and $rhs(s_{goal})$ is set to zero and $g(s_{goal})$ getting value 1, in Main() line 15 and 16. Now when the state s_{goal} is over consistent it inserted to *open* list. *Open* list holds the inconsistent states, these are the states that need to be updated and made consistent. After that algorithm starting to find path and does it until complete path is found or *open* list is empty, line 7 in ComputeShortestPath(). Complete path is found if $rhs(s_{start}) = g(s_{start})$. *Open* list cleared in order of lowest priority first. When ComputeShortestPath() expands a locally over consistent state (line 9), then it makes the vertex locally consistent by setting g-value of the state to its rhs-value. Otherwise if ComputeShortestPath() expands a locally under consistent state (line 12) then it simply sets the g-value of the state to infinity (line 13). If the expanded state is over consistent, then the change of its g-value can affect the local consistency of its successors (line 11) or if the expanded state is locally under consistent, then state and its successors can be affected (line 14). In UpdateState(s) all unvisited states g-value valued as infinity. In lines 4 – 6 the rhs-values will be updated so that algorithm the condition to be met. Requirements that must be satisfied are [49]:

( ) { ( ) ( )

( )

-

- *Open* list always contains exactly the locally inconsistent states.
- The priority of a state in the *open* list is always the same as its key.

Advantage: delivers optimum path and allows partial re-calculations after local OctoMap changes.

Disadvantage: doesn't allow quick and rough computations, computation may take a while until completed.

4.2.4 AD*

Planning for systems, operating in the real world, involves of challenges. Firstly, the real world is an inherently relatively uncertain and dynamic place. Accurate models for planning are difficult to compose and may quickly become out of date. Although in this project there are restriction to the environment where the UAV operates it is still reasonable this platform to have property to manage dynamic objects. For that problem there are replanning algorithms, like D*, what can solve this problem. Secondly, when operating in the real world, the time is constraint. This is especially important with this platform where the power supply determines the time limit, with smaller fling robots waiting is not on option because the even maintaining current position drains power supply quite rapidly. For this problem there are developed anytime algorithms which have shown promising results solving these kind problems. The main idea with anytime algorithms is that firstly they give suboptimal path and while the time pass by they improve it.

AD* (Anytime Dynamic A*) have properties to deal with both problems [48], it can handle dynamic environment as well as complex planning problems. At the beginning, AD* quickly computes a suboptimal path to the goal, and continually improves the bound on the initial solution based on available planning time. The heuristic inflation factor is the key by decreasing the inflation factor, the solution improves in terms of optimality but more states are explored.

```
key(s)
01. if (g(s) > rhs(s))
02.  return [rhs(s) + ε · h(s_start, s); rhs(s)];
03. else
04.  return [g(s) + h(s_start, s); g(s)];

UpdateState(s)
05. if s was not visited before
06.  g(s) ∈ ∞;                ∈
07. if (s ≠ s_goal) rhs(s) = min_{s' Succ}(s)(c(s, s') + g(s'));
08. if (s ∉ OPEN) remove s from OPEN;
09. if (g(s) ≠ rhs(s))
10.  if s   CLOSED
11.       insert s into OPEN with key(s);
12.  else
13.       insert s into INCONS;

ComputeorImprovePath()
14. while (min_{s OPEN}(key(s)) ˙< key(s_start) OR rhs(s_start) ≠ g(s_start))
15.  remove state s with the minimum key from OPEN;
16.  if (g(s) > rhs(s))    ∪
17.       g(s) = rhs(s);
18.       CLOSED = CLOSED   {s};
19.       for all s'   Pred(s) UpdateState(s');
20.  else            ∈          ∪
21.       g(s) = ∞;
22.       for all s'   Pred(s)   {s} UpdateState(s');
```

Figure 32. Anytime dynamic A* ComputeorImprovePath function [48].

AD* performs a series of searches using decreasing inflation factors ε. Inflation factor ε is a variable what is used to change heuristic value for improving path search.

When changes occurred in the environment and the cost of the edges change, locally affected states are placed into an OPEN list with priorities equal to the minimum of their previous key value and their new key value.

```
Main()
01. g(s_start) = rhs(s_start) = ∞; g(s_goal) = ∞;
02. rhs(s_goal) = 0; ε = ε_0;
03. OPEN = CLOSED = INCONS = ∅;
04. insert s_goal into OPEN with key(s_goal);
05. ComputeorImprovePath();
06. publish current ε-suboptimal solution;
07. forever
08.  if changes in edge costs are detected
09.       for all directed edges (u, v) with changed edge costs
10.       Update the edge cost c(u, v);
11.       UpdateState(u);
12.  if significant edge cost changes were observed
13.       increase ε or replan from scratch;
14.  else if ε > 1
15.       decrease ε;
16.  Move states from INCONS into OPEN;
17.  Update the priorities for all s ∈ OPEN according to key(s);
18.  CLOSED = ∅;
19.  ComputeorImprovePath();
20.  publish current ε-suboptimal solution;
21.  if ε = 1
22.       wait for changes in edge costs;
```

Figure 33. Anytume dynamic A* Main function [48].

On the Figure 32 and Figure 33 is presented AD* algorithms pseudocode. At the beginning Main function sets some initial values and gives inflation factor ε a sufficiently high value ε_0 so that algorithm generates first suboptimal path quickly (lines 1 – 6). After that if changes in edge costs are not detected, the Main function start to decrease ε step by step and improves quality of its solution until it is optimal (lines 14 – 20). Final solution is optimal when ε = 1. Each time when ε is decreased, all inconsistent states are moved from INCONS list to OPEN list and CLOSED list is made empty. In the OPEN list the priority of each state s is stored. CLOSED list contains all states already expanded once in the current search. INCONS contains all states that have already been expanded and are inconsistent. In case the environment changes then there is a change that already path is not ε-suboptimal. Now algorithm must calculate new path although if changes are substantial then repairing current path might be computationally expensive to recover ε-suboptimality. For making calculations faster, algorithm increases ε what gives less optimal solution but does it quickly (line 12 – 13).

The increase of the edge costs may cause some states to become under consistent, therefore states need to be moved into OPEN list with a key-values. In Figure 32 the new key-values

are calculated just like in D* Lite algorithm. In order to guarantee that under consistent states propagate their new costs to their affected neighbors, their key values must use uninflated heuristic values. This means that for under consistent (line 1 – 2) states and over consistent (line 3 – 4) states different key values must be computed.

Advantage: dynamic replanning, allows trade-off between computation time and accuracy with respect to path minimality.

4.2.5 VFH

DVFH (Double Vector Field Histogram) and VFH (Vector Field Histogram)

the planner constructs a polar histogram centered at the MAV's current position

Describe the concept: polar coordinate histogram technique, starting from a local view into the environment.

Advantage: memory efficient for small environments.

Disadvantage: complex cost function calculations, reducing the efficiency and the scalability of the approach for application in large sceneries.

4.3 Comparison and algorithm selection

A* provides a static optimum path calculation for a given scenery, but the required computation maybe intensive and the mission cannot be started before completion of the mission. In case of obstacle changes (relevant (= critical) cases: relocation, new obstacles), the whole computation has to be repeated. This is infeasible, once new obstacles occur in a path which has been assumed to as free and open. Therefore, the additional functionality of partial re-calculations in case of OctoMap changes provided by D* (best implementation: D*-lite) is definitively needed. With D* still the disadvantage remains, that the computed optimum solution is visible not before the completion of the algorithmic computations, which may be time-consuming in extensive scenarios. This disadvantage is addressed by A*-Anytime, which provides the capability of calculating rough solutions quickly and

improving it over time. Therefore, in case of scenario changes, missions can be smoothly continued with improved path availability during the mission.

For that reason, A*-Anytime has been selected as backbone algorithm for the mission path planning in this project. Because of its capability of dynamic precision improvement over time it also allows a trade-off between the usage of computation resources (processing time, memory usage) and therefor the method is always scalable to operate in combination with large OctoMap models.

DVFH works well with smaller scenarios, but the overall scaling is expected to be handled more easily with A*-Anytime.

Dijkstra slow < A* faster < D* dynamic

5 Future work

Since the time for the thesis has been limited, no time for the implementation has been left. The plan which should be applied for implementation is as follows:

1 Implementation of a simple Dijkstra Algorithm in order to set up and test the algorithm-OctoMap cooperation in a lower complexity environment; Approval in real UAV flight scenarios and test.
2 Implementation of the A*-Anytime Algorithm; Approval in real-world.

Once these basic path planning software modules are in place the following optimisations can be targeted:

- Optimisation of the mission service time by raising up the flight speed.
- Minimisation of energy consumption.
- Inclusion of flight dynamics models for improved trajectory determination.

6 References

[1] E. Udod, "LIDAR-based Environmental Perception System for Experimental Unmanned Aerial Vehicle," 2016.

[2] "DJI Matrice 600," [Online]. Available: http://www.dji.com/product/matrice600. [Accessed 18 05 2016].

[3] "Design of a Quadrotor System for an Autonomous Indoor Exploration," in *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014*, 2014.

[4] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer and M. Pollefeys, "PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight using Onboard Computer Vision," *Autonomous Robots,* vol. 33, no. 1, pp. 20-39, 2012.

[5] L. M. P. T. F. F. M. P. Lionel Heng, "Autonomous Obstacle Avoidance and Maneuvering on a Vision-guided MAV using On-board Processing," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, Shanghai, 2011.

[6] L. Meier, P. Tanskanen, F. Fraundorfer and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *Robotics and Automation (ICRA), 2011 IEEE International Conference*, Shanghai, 2011.

[7] M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, S. Weiss and L. Meier, "Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments," *IEEE Robotics Automation Magazine,* vol. 21, no. 3, pp. 26-40, 09 2011.

[8] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3D environment," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*, Alcala de

Henares, 2012.

[9] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske and S. Singh, "River mapping from a flying robot: state estimation, river detection, and obstacle mapping," *Autonomous Robots,* vol. 33, no. 1-2, pp. 189-214, 08 2012.

[10] A. J. Barry , "High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo," 2016.

[11] G. Cai, B. M. Chen and T. H. Lee, Unmanned Rotorcraft Systems, Springer, 2011.

[12] "Pixhawk Infographic, Jethro Hazelhurst," [Online]. Available: http://diydrones.com/profiles/blogs/pixhawk-infographic. [Accessed 05 01 2014].

[13] "Rc hobby, faq propellers," [Online]. Available: http://www.headsuprc-info.com/propeller-faq.html. [Accessed 18 05 2016].

[14] "Ecalc, the RC Calculator," [Online]. Available: http://www.ecalc.ch. [Accessed 02 01 2016].

[15] "Rc model aviation knowledge base," [Online]. Available: http://2bfly.com/knowledgebase/powerplants/propellers/types. [Accessed 18 05 2016].

[16] "Suggested rpm limits for apc propellers," [Online]. Available: https://www.apcprop.com/Articles.asp?ID=255. [Accessed 17 05 2016].

[17] "Thrust-to-weight ratio," [Online]. Available: https://en.wikipedia.org/wiki/Thrust-to-weight_ratio. [Accessed 17 05 2016].

[18] "SiK Radio - Advanced Configuration," [Online]. Available: http://ardupilot.org/copter/docs/common-3dr-radio-advanced-configuration-and-technical-information.html. [Accessed 17 05 2016].

[19] "Mission Planner," [Online]. Available: http://ardupilot.org/planner/index.html#home. [Accessed 22 11 2015].

[20] "Taranis X9D Plus," [Online]. Available: http://www.frsky-rc.com/product/pro.php?pro_id=137. [Accessed 17 05 2016].

[21] "Riigi teataja," [Online]. Available: https://www.riigiteataja.ee/akt/23174. [Accessed 17 05 2016].

[22] "Autopilot Hardware Options: Pixhawk Overview," [Online]. Available:

http://ardupilot.org/copter/docs/common-pixhawk-overview.html. [Accessed 06 04 2016].

[23] "APM Copter Source Code Licence (GPLv3)," [Online]. Available: http://ardupilot.org/dev/docs/license-gplv3.html. [Accessed 01 05 2015].

[24] "A Gas Powered Quadcopter," [Online]. Available: https://hackaday.io/project/1230-goliath-a-gas-powered-quadcopter/log/12723-firmware-and-flight-stacks. [Accessed 05 04 2016].

[25] "ArduPilot: Basic Structure," [Online]. Available: http://ardupilot.org/dev/docs/learning-ardupilot-introduction.html. [Accessed 01 01 2016].

[26] "PX4 Source Code," [Online]. Available: https://pixhawk.org/firmware/source_code. [Accessed 01 05 2015].

[27] "ArduPilot Programming (Core) Libraries," [Online]. Available: http://ardupilot.org/dev/docs/apmcopter-programming-libraries.html. [Accessed 01 01 2016].

[28] "Mission Planner Overview," [Online]. Available: http://ardupilot.org/planner/docs/mission-planner-overview.html. [Accessed 06 04 2016].

[29] "Failsafe Topics and Setup for APM Flight Software," [Online]. Available: http://ardupilot.org/copter/docs/failsafe-landing-page.html. [Accessed 06 04 2016].

[30] "Pre-arm Safety Checks," [Online]. Available: http://ardupilot.org/copter/docs/prearm_safety_check.html. [Accessed 06 03 2016].

[31] "ESC calibration," [Online]. Available: http://ardupilot.org/copter/docs/esc-calibration.html. [Accessed 06 04 2016].

[32] P. Pfaff and W. B. R. Triebel, "Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing," 2006.

[33] P. Pfaff and W. Burgard, "An Efficient Extension of Elevation Maps for Outdoor Terrain Mapping," in *Field and Service Robotics: Results of the 5th International Conference*, P. Corke and S. Sukkariah, Eds., Berlin, Heidelberg Germany,, Springer

Berlin Heidelberg, 2006, pp. 195-206.

[34] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots,* vol. 34, no. 3, pp. 189-206, 02 2013.

[35] "Albert-Ludwigs-Universität Freiburg. Autonome Intelligente Systeme. OctoMap 3D scan dataset.," [Online]. Available: http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/. [Accessed 31 05 2016].

[36] "Qgroundcontrol," [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed 25 05 2016].

[37] D. Ferguson, M. Likhachev and A. Stentz, "A Guide to Heuristic-based Path Planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, Pittsburgh, 2005.

[38] W.-M. Chan, "Comparison of Path Planning Algorithm," 1999. [Online]. Available: http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/wmchan/. [Accessed 18a' 05 2016].

[39] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269-271, 12 1959.

[40] P. L. Frana and T. J. Misa, "An interview with Edsger W. Dijkstra," *Communications of the ACM,* vol. 53, no. 8, pp. 41-47, 08 2010.

[41] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for A*," *International Journal of Geographical Information Science,* vol. 23, no. 4, pp. 531-543, 28 01 2009.

[42] D. Imms, "Growing with the Web, A* pathfinding algorithm," [Online]. Available: http://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html. [Accessed 28 05 2016].

[43] P. E. Hart, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, pp. 100-107, 07 1968.

[44] R. Eranki, "Pathfinding using A* (A-Star)," 2002. [Online]. Available: http://web.mit.edu/eranki/www/tutorials/search/. [Accessed 30 05 2016].

[45] P. Muntean, "Mobile Robot Navigation on Partially Known Maps using a Fast A Star Algorithm Version," *ArXiv e-prints,* 10 05 2016.

[46] A. Stentz, "Optimal and efficient path planning for partially-known environments," *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on,* vol. 4, pp. 3310-3317, 05 1994.

[47] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on Pittsburgh, PA,* vol. 1.1, pp. 425-432, 1995.

[48] M. Likhachev, D. Ferguson, G. Gordon, A. (. Stentz and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS),* 2005.

[49] S. Koenig and M. Likhachev, "D* Lite," *American Association for Artificial Intelligence,* pp. 476-483, 28 07 2002.

[50] "FRSKY Radio System. Taranis X9D Plus," [Online]. Available: http://www.frsky-rc.com/product/pro.php?pro_id=137. [Accessed 01 01 2015].

[51] "XBee® ZigBee Devices," [Online]. Available: http://www.digi.com/products/xbee-rf-solutions/rf-modules/xbee-zigbee#specifications. [Accessed 01 03 2015].

Appendix 1 - UAV's General Components

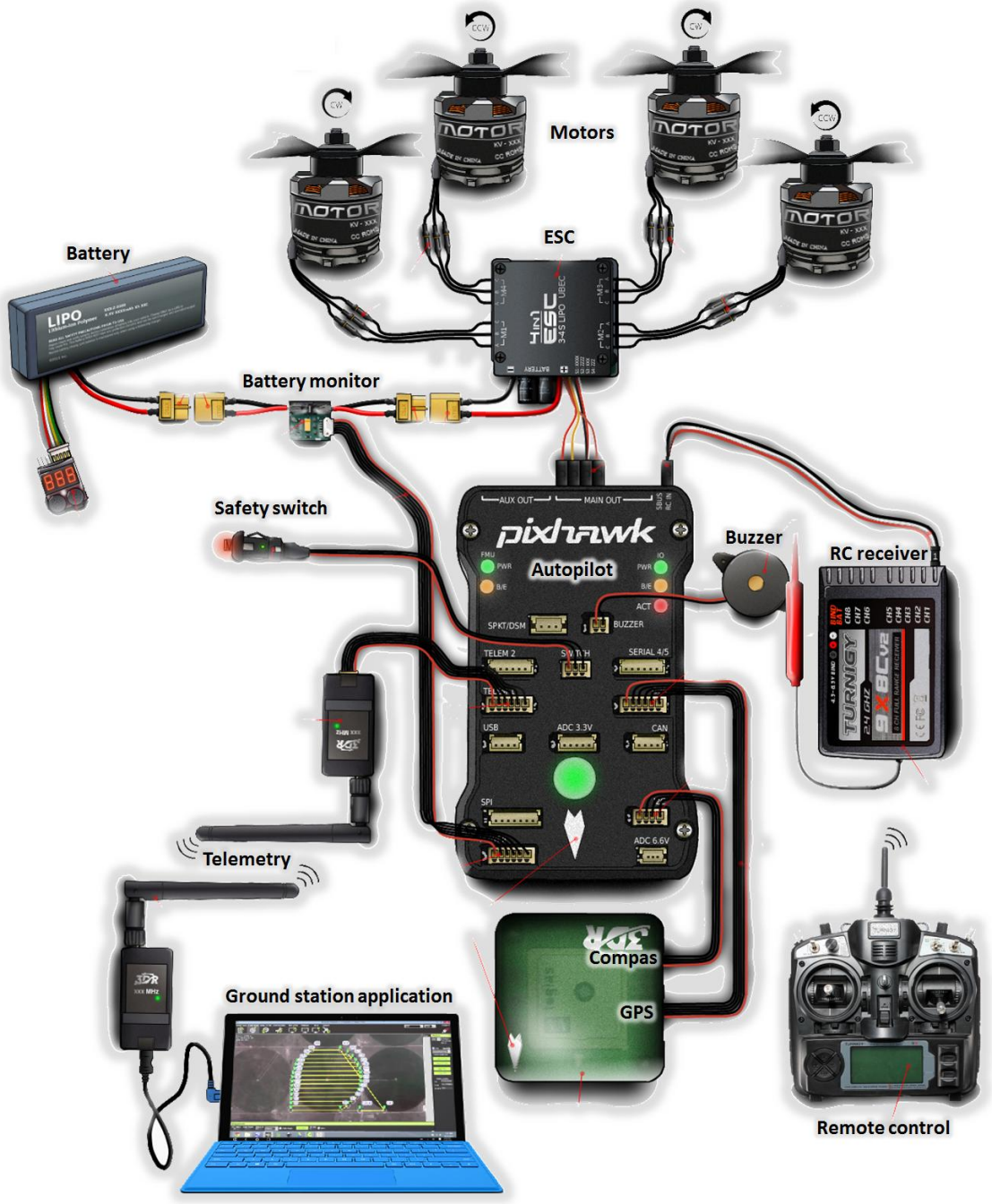


Figure 34. Components for the Quad-rotor Model.

Appendix 2 - UAV's Flight Dynamics

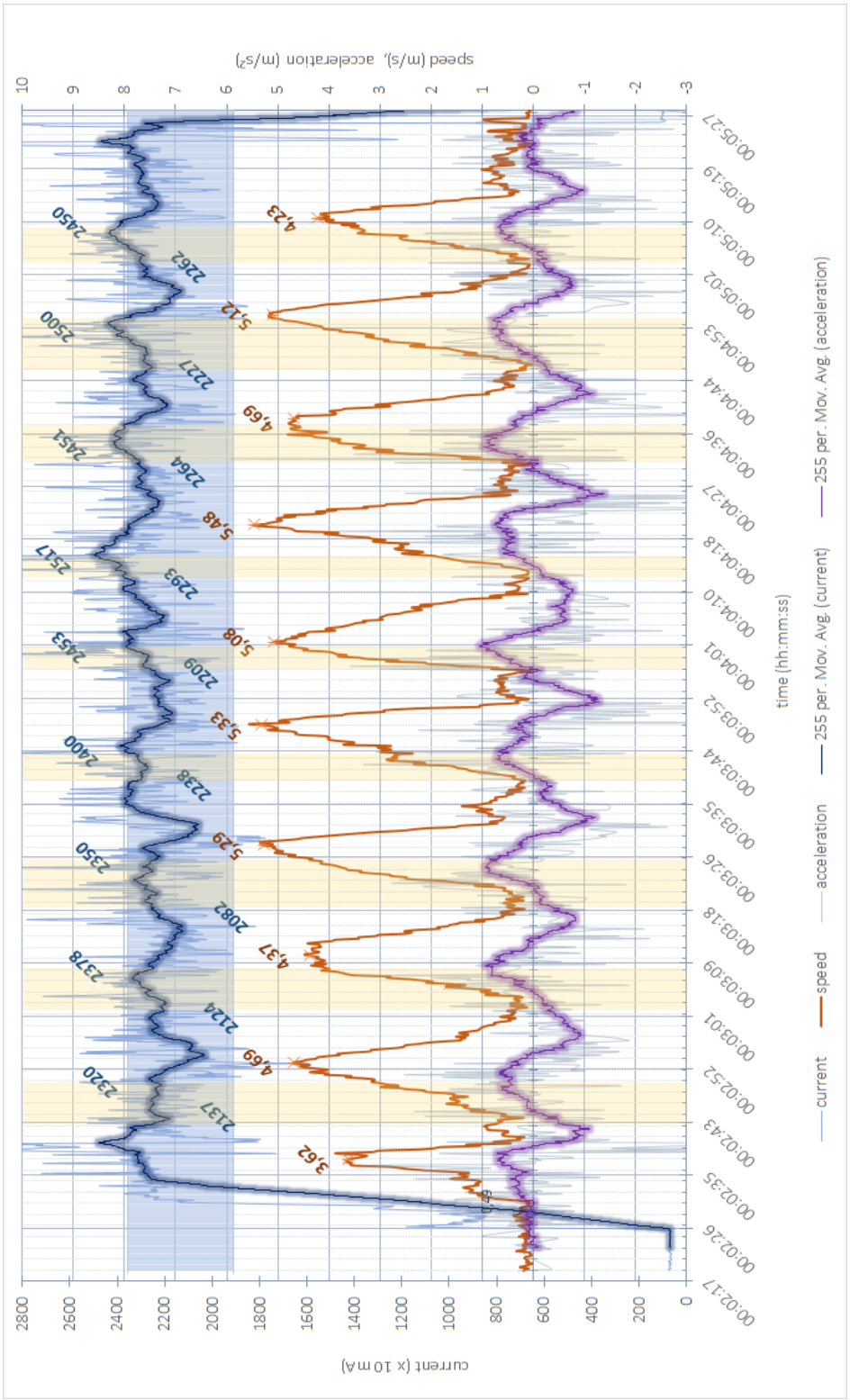


Figure 35. Full Flight Results. Guided Mode. Current Consumption vs Flight Dynamics.

Appendix 3 - Environmental Mapping Model

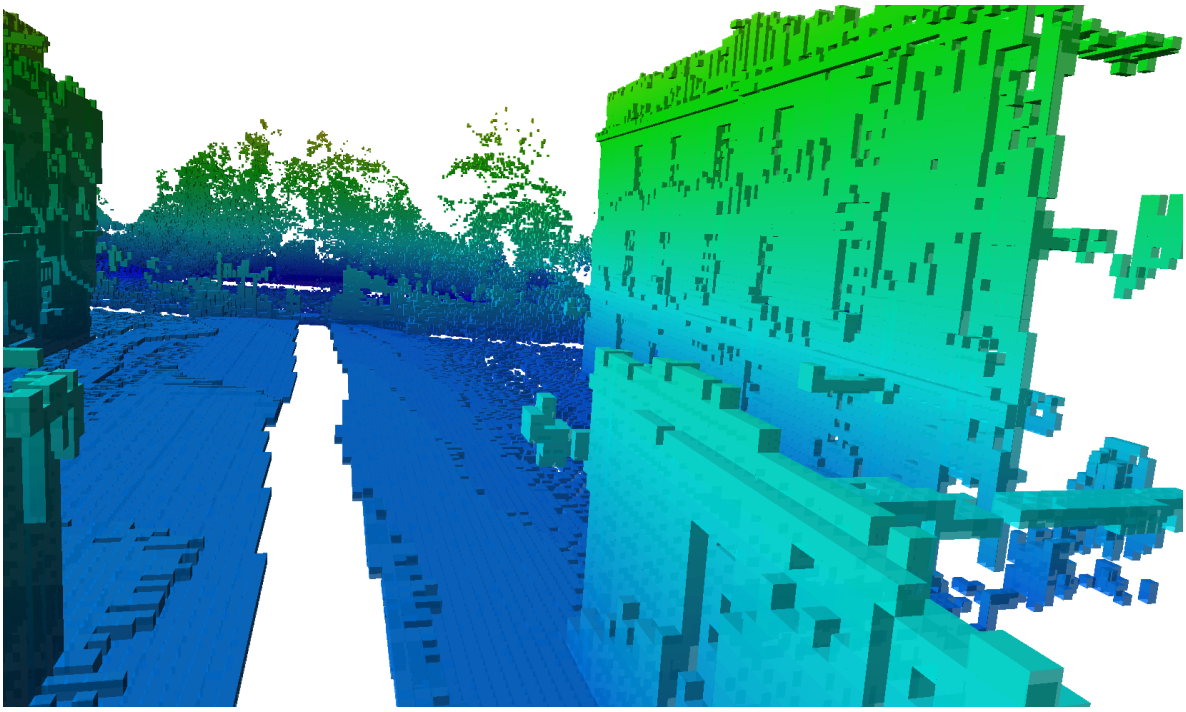


Figure 36. OctoMap Corridor Visualization, Cube 20 cm. Occupied Space [35].