Autonomous Indoor Drone

Rafael Mäuer et al.

February 2020

Special Aspects of Autonomous Mobile Systems

Supervisor: Prof. Dr.-Ing. Chunrong Yuan

Fakultät für
Informations-, Medien-
und Elektrotechnik

Technology
Arts Sciences
TH Köln

Abstract

In this project an autonomous indoor flying drone was built, that is capable of creating a map of its surrounding room and navigating through it on its own. The drone in this project was custom built and equipped with a variety of sensors to be able to create an environmental understanding. Additionally, a part of the project was the creation of a simulated version of the drone to test the mapping and navigation algorithms. Most of the development for the drone software was done in Python using ROS and the package ROSflight. Some sensor components are based on C/C++ and several service scripts were created with bash.
In the end the drone proved itself to be slightly too unstable to maneuver on its own, while the software achieved a stable mapping process and accurate navigation results.

Contents

1 Introduction

Today, autonomous drones are gaining more and more popularity. Usually, they are used for outdoor use cases. This includes especially fields of application like photography, video recording, parcel delivery and search and rescue missions[1]. However, there are also several indoor use cases where an autonomous flying drone can be helpful. Examples are e.g. carrying light objects from one room to another, showing the path to a particular room or house cleaning.
The goal of this project is to construct a drone that is capable of autonomously detecting its surroundings and flying to defined goals in an indoor environment. While flying, the drone should be able to avoid obstacles. To realize this kind of drone, a map of the room needs to be created at fly time. Additionally, it has to be displayed in some kind of application for the user to be able to retrace the drone's understanding of the room and to define points it should fly towards. In the beginning of the project it has been defined to create a 3D map. Because of the high complexity of this task, the team decided later to only focus on 2D localization.
It was decided to split the project in two major parts: hardware and software. The hardware parts should deal with gathering the necessary equipment parts, assembling them and making sure the sensor data could be read from a central system. The software part should concern itself with constructing the logic necessary to be able to create the map from the provided sensor data and using this map to realize the autonomous flight. To simplify development it was decided to test the software in a simulation until the real drone was finished.

Figure 1: Flying drone [23]

[1] https://www.digi.com/blog/post/iot-drones-how-use-cases-for-drones-are-changing

2 Related Work

There are a lot of projects and research focusing on autonomous moving vehicles. Some should be mentioned in the following.
One topic that gets increasing attention are autonomous driving cars. There are companies like Tesla [33] which are currently developing and testing cars that can drive from one place to another without requiring human control. This includes a lot of complexities like outdoor mapping, navigation, recognizing traffic signs or people and other traffic participants. For that purpose, cars require various sensors, like cameras, lidar, ultrasonic, GPS and IMU sensors [27].
Other fields of application are robot vacuum cleaners and lawn mowers where it is required to know the environment to execute the desired task reliably. Compared to autonomous cars, however, they have less complexities to face since they do not endanger people. Moreover, their intended movement area is relatively small. There exist different approaches using ultrasonic sensors [9], bumper and cliff sensors [22] or even with camera, sonar and GPS [35]. These are only examples showing how many different possibilities there are.
When it comes to flying robots, there are also other factors that need to be considered. Controlling a flying vehicle is far more complex, because it additionally depends on further conditions like wind, gravity etc. Autonomous flying drones need to be able to navigate to particular positions. For outdoor use cases like product home delivery, GPS and IMU can be used for sensing the position of the drone and navigating towards the target [21]. There is also a particular project called *Roscopter* [19] that is based on ROSflight and uses GPS and IMU data for waypoint navigation. It includes a simulation and is meant to be used on real drones.
For indoor purposes, GPS is not suitable, because it is not precise enough for estimating the position within a room. *De Croon and de Wagter* [10] point out problems associated with indoor navigation and sensor technology, in particular the confined space and the difficulty of estimating the position. Furthermore, they provide a list of sensors that are suitable for realizing indoor navigation, e.g. different camera types, radar, sonar, laser scanner or infrared sensors.
This knowledge serves as a good basis for the conceptual design of an indoor flying drone, which is the purpose of this project.

3 Equipment

To realize such a project, a certain basic amount of special equipment is necessary. This includes not only the actual components from which the drone is assembled, but also a remote control for tests and safety as well as a battery charger. To use the simulation software, a computer with Linux as operating system is required. For wireless connection to the drone's onboard computer, a WiFi router is necessary too.

3.1 Remote Control

In order to test the correct functionality of a drone and to find the correct settings before starting with autonomous flight, a remote control is needed to command the drone. During autonomous flight it is an additional safety feature, as you can get manual control of the drone in critical situations or errors to prevent damage to equipment or the environment.

Figure 2: Flysky TM10 Remote Control [28]

In this project a Flysky TM10 remote control is used, which is optimally suited to control a drone (Figure 2). It offers 10 channels on 2.4 GHz digital radio to control the drone. At least 4 channels are necessary, one channel each for the transmission of throttle, pitch, roll and yaw. Additionally it is recommended to use one more channel for arming, therefore the use of 6 channels is recommended. In addition, there is one more channel available for transmitting telemetry data, for example.

3.2 Battery Charger

During the flight of drones, a large amount of energy is required to enable them to float in the air. To cover this demand, light and very powerful batteries are needed. Therefore it is very common to use Lithium-Polymer batteries (LiPo) in model making, because they can cover a very high energy demand at very low weight.

Figure 3: IMax B6 LiPo Charger [29]

However, LiPo batteries require particularly careful handling, as otherwise serious accidents can occur if they are not handled properly. A LiPo battery usually consists of several individual cells which must always have a similar state of charge. If the cells are unevenly charged or discharged, or if they are discharged below a certain threshold, the battery may become unstable and, in the worst case, catch fire or explode.
For this reason it is necessary to use a special charger, which prevents this danger. To do this, it must be able to charge each of the cells independently when the battery is being charged, and to monitor the condition of each cell closely. This process is called balancing, because the state of the individual cells is adjusted to each other and set to the same level.
Therefore, an IMax B6 charger is used, which supports the automatic balancing function for LiPo batteries with 1 to 6 cells (Figure 3).

3.3 Linux Computer

In order to use the simulation software, a Linux computer was necessary, because Linux was too imperformant in a VM or in the Windows subsystem. Therefore an Ubuntu 18 Linux was installed on a Lenovo Flex 2 laptop (Figure 4).

Figure 4: Lenovo Flex 2 [2]

3.4 WiFi Router

A WiFi router was used to enable development on the drone's onboard computer without a cable connection (Figure 5). In addition, this allows commands to be sent to the drone during autonomous flight, as all devices in the private network can communicate with each other.

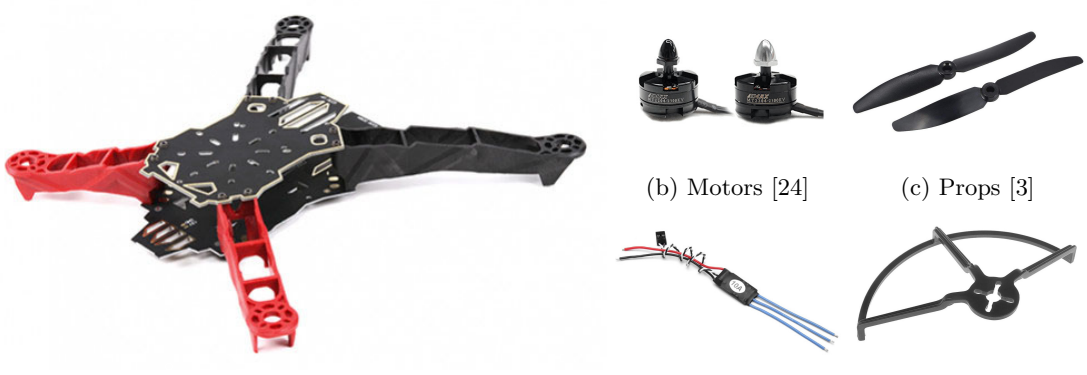
Figure 5: FritzBox Fon WLAN 7112 [5]

4 Hardware

First the underlying hardware will be introduced and explained before proceeding with the software part. This includes explaining which parts are all necessary to build a drone, how the parts are connected together and which sensors are used to detect the environment.

4.1 Drone Parts

Let's start with the components needed to build a drone: first of all, a frame is needed to which the engines and all other components can be attached. A Storm Q330 frame is used here, as it still has plenty of space to accommodate other components (Figure 6a). In order for the drone to fly, two opposite motors each must rotate clockwise (CW) and two counterclockwise (CCW). The four motors of the type EMAX MT2204 KV2300 Racing (Figure 6b) are bolted to the ends of the four arms of the frame. For safety purposes it is advised to mount suitable Prop-Guards (Figure 6e) underneath, to avoid damage to the drone and the environment, should it bump against an obstacle. Then the appropriate propellers of type 5030 5X3 (Figure 6c) must be mounted on the motors and fastened well. Here too, the correct direction of rotation of the propellers must be ensured. Propellers for CCW rotation are marked with an "R". Finally the three cables of the motors must be connected to the correct 3 cables of an Electronic Speed Controller (ESC) each of type BLHeli 10A (Figure 6d). As these are brushless motors, the combination decides in which direction the motor rotates when connected.



(a) Frame [25] (b) Motors [24] (c) Props [3] (d) ESC [18] (e) Prop-Guard [4]

Figure 6: Drone Parts

4.2 Power Supply

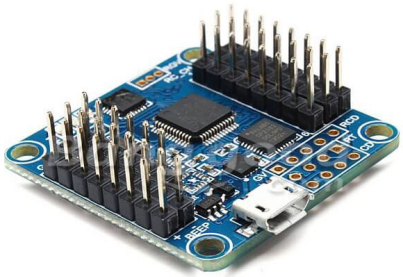
In order to supply all the necessary parts with sufficient current, a LiPo battery with 3 cells (3S), 11.1 Volt and 2200 mAh power of type HRB is used (Figure 7a). This battery, with dimensions of 105 x 34 x 24 mm and a weight of 168 grams, is an optimal compromise and allows about 15-20 minutes of flight. As already mentioned in section 3.2, the use of LiPo batteries is not entirely risk-free. Therefore, a LiPo alarm (Figure 7b) is also used, which emits an extremely loud warning signal if the cells discharge unevenly or the voltage drops below an adjustable threshold. However, all other components require 5 volts, so a DC/DC converter is used (Figure 7c), which reduces the 11.1 volts to 5 volts while delivering 3 amps.

(a) HRB LiPo 3S [6] (b) LiPo Alarm [8] (c) DC Converter [7]

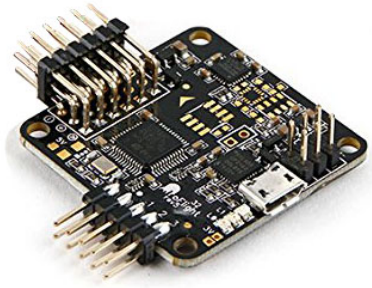
Figure 7: Power Supply

4.3 Flight Controller

The flight controller is the heart of a drone, as it is responsible with the automatic position correction for the drone to reach a hovering state. On the flight controller there is an IMU with different degrees of freedom (DOF), usually 6 DOF: three axes each of the gyroscope and accelerometer. Some IMUs also have a magnetometer and a barometer and can reach 9 and 10 DOF respectively.
As a flight controller for the drone, there is only a limited selection of compatible models if ROSflight should run as firmware on it. F1 (STM32F103 processor) and F4 (STM32F405 processor) flight controllers are supported, although support for the less powerful F1 controllers has already been discontinued[2]. The designation F stands for the performance of the processor, the higher the following number the better the performance of the processor. Common F1 flight controllers are the Flip32 and Naze32 flight controller, as F4 flight controller the CC3D Revo is recommended, but it is hard to get outside the USA. For this reason the Flip32 and Naze32 were used in this project. First the Flip32 flight controller was used, but since it has straight connectors, it is not very compact when connectors are plugged in (Figure 8a). Considering all the other components that have to be placed in the frame, it was not really suitable. As an alternative, the Naze32 flight controller came into question, because it has angled connectors, which allows a much more space-saving arrangement of the components (Figure 8b).



(a) Flip32 Flight Controller [17]



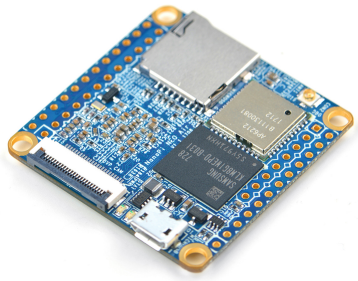
(b) Naze32 Flight Controller [1]

Figure 8: Flight Controller

4.4 Companion Computer

In order for the drone to perform calculations based on the measured sensor data, create a map from it and determine its own position in it, a companion computer must be positioned on the drone itself. For this purpose a Raspberry Pi 3+ was used initially for testing purposes to control and read out the sensors and to establish the communication with the flight controller (Figure 9a). Since there is only very little space available on the drone itself, it was required to switch to a NanoPi Neo Air (NanoPi) as the computing unit on the drone itself later (Figure 9b).

(a) Raspberry Pi 3+ [15]



(b) Nano Pi Neo Air [36]

Figure 9: Companion Computer

[2] `https://docs.rosflight.org/user-guide/flight-controller-setup`

4.5 Receiver

Since the drone must be able to receive control signals from the remote control, a receiver must be mounted on the drone and connected to the flight controller. For this purpose a Flysky FS-A8S Mini Receiver is used, which transmits on 2.4GHz and can handle up to 8 channels (Figure 10a). By supporting the PPM, i-BUS and SBUS protocols, an extremely small design is made possible, as all 8 channels can be transmitted to the flight controller via a single cable.
As the NanoPi has no onboard antenna for WiFi, it was necessary to connect a WiFi antenna to the designated IPX port to ensure sufficient signal reception. Therefore a 2.4GHz band antenna from Particle was attached (Figure 10b).

(a) Flysky FS-A8S 2.4G 8CH Mini Receiver [34]

(b) Particle 2.4GHz band antenna [31]

Figure 10: Signal Receiver

4.6 Status LED

To be able to read the status of all components easily and quickly, an LED status bar with 8 individually controllable RGB LEDs was used (figure 11). The individual LEDs indicate the system status of the individual components. Red indicates an error, the meaning of yellow and green can be obtained from the following status table:

LED	red	yellow	green
1	-	-	OS is active (blinks)
2	Error	ROS is running	ROSflight is running
3	Error	IMU is calibrated	Drone is armed
4	Error	RC is connected	Failsafe is active
5	Error	Ultrasonic measures	Ultrasonic publishes
6	Error	LIDAR measures	LIDAR publishes
7	Error	Webserver is running	Map is generated
8	Error	Autopilot is running	Autopilot has control

Table 1: LED Status Table

The first LED has a special function, it shows by continuous green flashing if the system is still active. When the LED stops blinking, it is a sign that the system has hung up. When all LEDs turn green, this indicates that all systems are ready for autonomous flight.

Figure 11: WS2812 LED Light Board for CC3D and Naze32 [26]

4.7 Sensors

To scan the environment around the drone for generating a map and to recognize obstacles, distance scans in two separate axes must be provided:

- The horizontal plane around the drone, oriented on the drones horizon
- The vertical axis over and under the drone, providing distances to floor and roof of the room

For scanning the horizontal plane the 360° laser scanner RPLIDAR A2 by SLAMTEC (LIDAR) is used (Figure 12b). This scanner uses reflecting laser rays to calculate distance values while its scan head spins ten times per second. The range of the scanner reaches from 20 centimeters to 12 meters with a distance resolution of $< 0.5mm$ for ranges under 1.5 meters and $< 1\%$ of the distance otherwise, while its angular resolution per data point has a value of 0.9° at the default 10Hz scan rate. To measure the distances, the LIDAR uses a 3 mW-peak infrared laser pulse, that lasts for 87 μs. Thus, the laser corresponds to the FDA laser class 1 and is therefore non-hazardous for humans.
The LIDAR is fixed on the top of the base plate of the drone with four screws and spacers so that no components of the drone are interfering when measuring the environment. If the sensor would be mounted below the drone, the landing gear would cause blind spots in the environment measurement.

(a) HC-SR04 ultrasonic sensor [30] (b) RPLIDAR A2 360° laser scanner [16]

Figure 12: Sensors

For the vertical axis two HC-SR04 ultrasonic sensors (US) were used, one for the upwards direction and another for the downwards direction (Figure 12a). Theses sensors have a measuring range from 2 centimeters to 4 meters over a measuring angle of 15°and a resolution up to $3mm$. After triggering, the sensor will send out eight 40 kHz sonic bursts and get an echo signal back over the GPIO pins.
Since the GPIO pins on the companion computer can only handle 3.3V and the US works with 5V, a resistor (330Ω) needs to be added between the ECHO output of the sensor and the corresponding GPIO pin. Additionally the pin is connected over a second resistor (470Ω) to GND to set the signal on the pin to 0, when there's no output from the sensor.

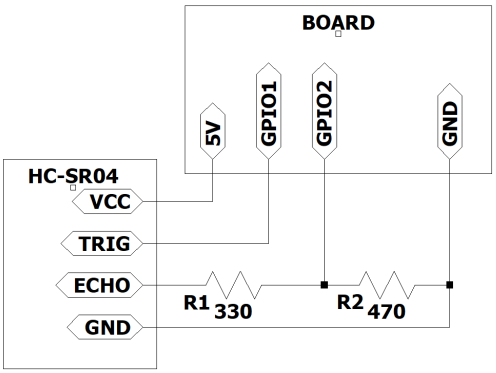


Figure 13: Circuit for the sensor connection to the companion computer pins

Two US are attached to the rear part of the drone to measure the distance to the floor and ceiling. They are connected to each other at the back and attached to the drone so that they are aligned upwards and downwards.

The process of how to get measured data from the sensors during flight is described in the software part in section 5.12 (Sensor Reading).

4.8 Schematics

To connect all hardware parts correctly, it was necessary to use the pinout schemes of both the Companion computer and the flight controller to find the correct connections (Figure 14 and 18). The motors are supplied with power directly from the LiPo battery via the ESC's. Only the minus and signal pins are connected to the control terminals of the ESCs. The plus pins on the motor rail of the flight controller are therefore free and can be used to supply power to other components. The NanoPi is thus supplied with power via the 5V-In pin.
Since the NanoPi can only provide a limited amount of power for connected components, the power supply of the US and LED status bar has been outsourced to the power connectors on the motor rail of the flight controller, which are directly supplied with power by the power converter. Only the LIDAR needs to be powered directly from the NanoPi, as it is convenient to connect it via the USB port of the included adapter. All components share a common minus so that the signal flow of data over the different data buses can work. This way a stable power supply for the whole system was achieved even during heavy power consumption during autonomous flight in the air.

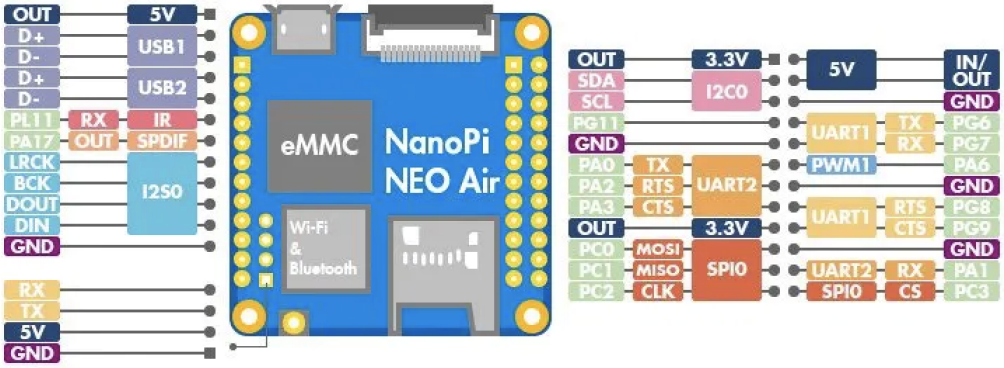


Figure 14: NanoPi Neo Air Pinout [20]

For the connection of US and LED status bar to the GPIO-Pins of the NanoPi the library WiringNP[3] was used. It is important to note that each pin can have 4 different definitions. Depending on the implementation the correct numbering has to be used:

- Name: Pin description in pinout diagram
- Physical: Number of physical connector position
- BCM: Pin number in BCM mode
- wPi: GPIO number in BOARD mode

While US 1 is connected to GPIOs 16 (Trigger) and 18 (Echo), US 2 is connected to GPIOs 22 (Trigger) and 24 (Echo) in BOARD mode. This corresponds to pins PG8, PG9, as well as PA1 and PC3 in the NanoPi's pinout diagram (figure 14).
To be able to use the LED status bar with the NanoPi, it was necessary to find a working software library. Finally, it was necessary to modify the rpi_ws281x library[4] slightly in order to get it work (more details in section 6.1). The rpi_ws281x library can be used with different methods (PWM/PCM/SPI), but since SPI has the lowest CPU usage, it was chosen. Therefore the data pin of the LED bar is connected to GPIO 10 (BOARD mode). This corresponds to pin PC0 in the NanoPi's pinout diagram (Figure 14), which is the MOSI pin of SPI0.
The connection between the Companion Computer and the flight controller is established via a serial interface. For Flip32 via UART and for Naze32 via USB (see section 6.1 for more details). Since USB on the NanoPi is not available via female connectors, but only via GPIO pins, cables

[3] https://github.com/friendlyarm/WiringNP
[4] https://github.com/jgarff/rpi_ws281x

had to be assembled with a micro-USB connector at one end and a GPIO connector at the other end. This way, the Naze32 is connected to the GPIOs D+ and D- of USB2 and the nearby pins 5V OUT and GND to the NanoPi. For the LIDAR the same USB cable was necessary again to be connected to the GPIOs D+ and D- of USB1. However, 5V and GND had to be taken from the nearby debug port (UART0).

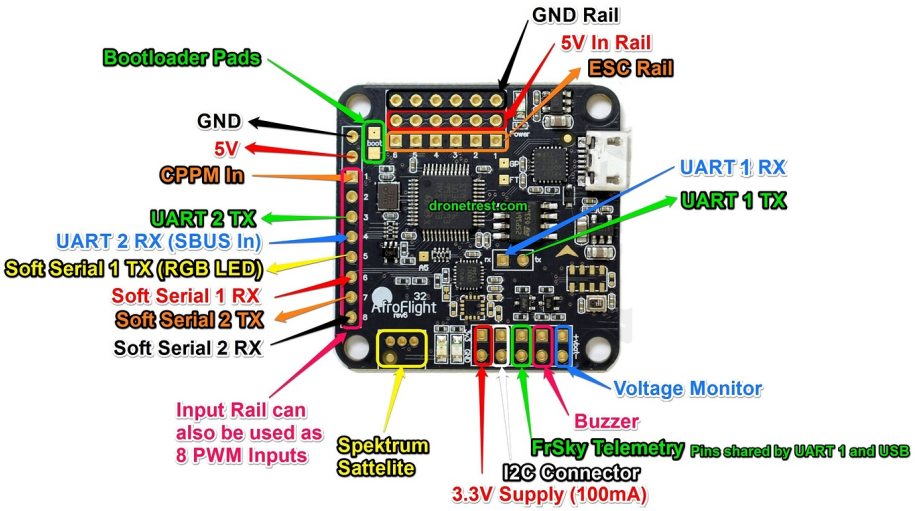


Figure 15: Naze32 Pinout [14]

The RC receiver is connected to the Naze32 via pin CPPM IN and supplied with power via the nearby pins 5V and GND (Figure 18). To visualize the complete overview of all interconnected components, Figure 16 gives an overview in form of a hardware connection diagram.

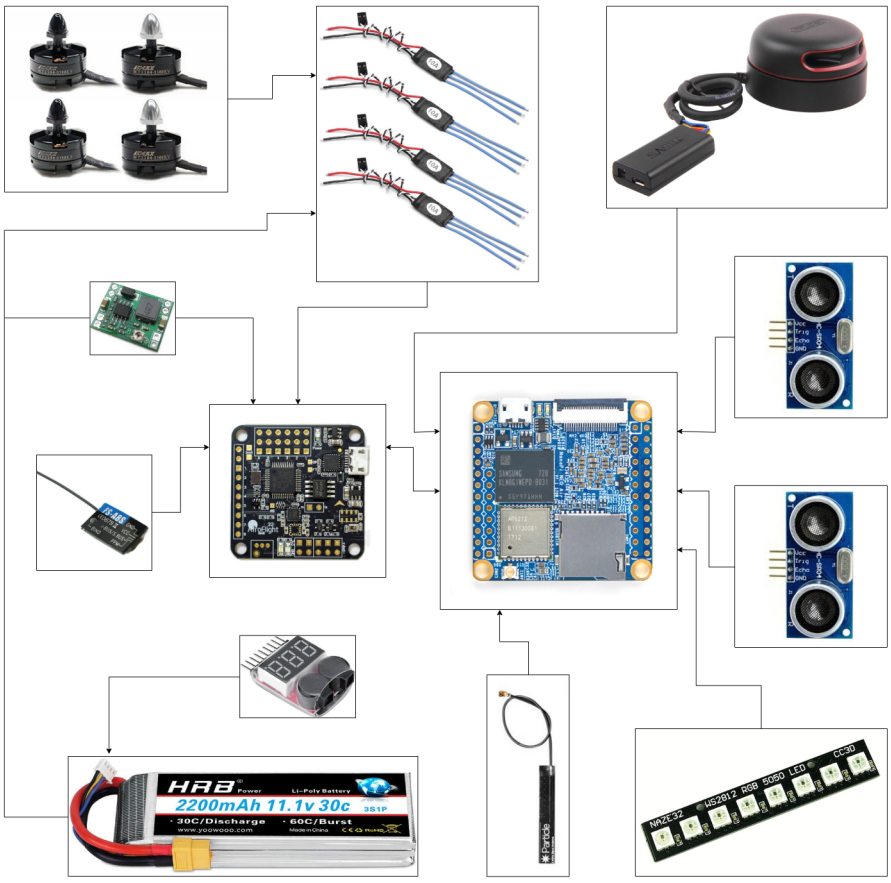
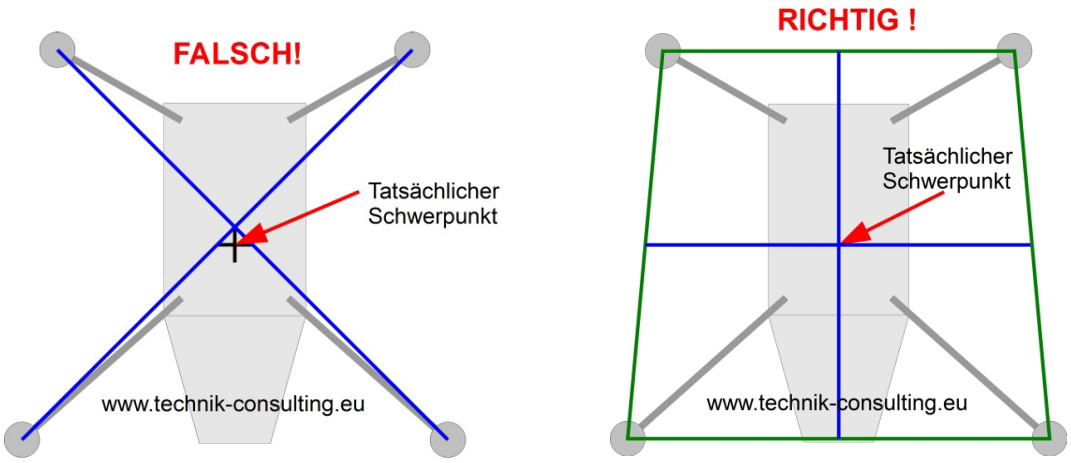


Figure 16: Hardware Connection Diagram

4.9 Build Setup

During the process of placing all the components on the drone, several things have to be taken into account to ensure that the drone can fly correctly.
First, the correct centre of gravity (C.G.) of the drone must be calculated. With a symmetrical drone this is quite simple: just take the center of the drone. With an asymmetric drone this is a bit more difficult. To determine the C.G. correctly, you must not simply take the intersection of the diagonals of the opposite motors, as might be assumed at first (Figure 17a). Instead, you have to build a square around all the motors, whose center corresponds to the real C.G. (Figure 17b).



(a) Determine Center of Gravity wrong [32]

(b) Determine Center of Gravity right [32]

Figure 17: Center of Gravity

Once the correct C.G. has been determined, it is advisable to mount the flight controller exactly at this point, as this requires the least amount of adjustment to stabilize the drone and the calculations of the flight controller are most effective.
The first attempt to accommodate all components in the drone's housing was based on ROSflight's recommendation[5] to mount the flight controller with a vibration protection system so that the vibration of the motors does not have too strong an effect on the IMU sensors (Figure 18).

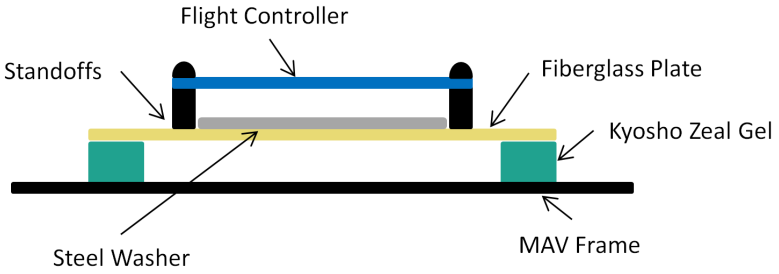


Figure 18: Vibration Isolation [12]

Unfortunately, this arrangement could not be realized, because the pins of the companion computer did not have enough space and were disturbed by the mountings on the drones' frame arms. Therefore revision 1 of the setup had to be discarded and a revision 2 needed to be created. Unfortunately, the vibration isolation had no more space left, but apart from that all important components could be accommodated.
The battery was mounted underneath the drone and some additional spring-loaded landing gear was added, so that the drone and the battery would not be damaged during a slightly harder landing. After attaching a few small weights to the rear two legs to achieve a perfect balance, the perfect hover state had to be achieved so that the software only had to take care of navigation and positioning. Unfortunately there appeared a difficulty here, which will be discussed in section 6.1.

[5] https://docs.rosflight.org/user-guide/hardware-setup/#vibration-isolation

5 Software

After the hardware has been presented and explained, the software used for the simulation and autonomous flight will be explained in the following.

5.1 OS and Setup

The requirements for the companion computer are, besides sufficient performance to calculate all necessary tasks, only that a certain LTS Ubuntu version runs on it with the appropriate version of ROS[6]. The supported versions of Ubuntu LTS and the corresponding ROS are:

- Ubuntu Xenial (16.04) and ROS Kinetic
- Ubuntu Bionic (18.04) and ROS Melodic

Since the highest supported Ubuntu version on the NanoPi is Ubuntu Xenial[7], only the ROS version Kinetic can be used. For the installation of Ubuntu on Raspberry Pi, preinstalled server images (PSI) are required. Since PSI for Ubuntu Xenial is only available for the Raspberry Pi 2[8], you have to use Ubuntu Bionic on the Raspberry Pi 3+[9], which means that only the ROS version Melodic can be used.
Since the installation of ROS on Ubuntu is not easy, ROS setup scripts[10] were used and modified. This enables to install not only ROS Melodic on Ubuntu Bionic but also ROS Kinetic on Ubuntu Xenial very easy. But the differences between the two different platforms also led to problems, which will be discussed in section 6.1.

5.2 Service Structure

To maintain a certain modularity, all software components are implemented as Linux services. This enables individual components to be restarted independently of the rest of the system, both during development and during preparation for autonomous flight. In addition, the status of each component can easily be queried and displayed on the status LED. Since most software components require some environment variables to function properly, Linux services basically run a shell script. Environment variables can be set there and a log file can be defined in which the output logs of the program are recorded. This makes it possible to read the log files of all services at once to get a complete overview of what is happening in the system and to debug errors.
To run all services automatically at system startup, an autorun script is executed which starts all necessary services in the correct order:

1. led: display system status
2. core: start ROScore node
3. flight: start ROSflight node
4. lidar: start LIDAR measurement
5. sonic: start US measurment
6. main: start autopilot and generate map
7. web: start server for Web-UI

5.3 ROS

The basis for the implemented system is the *Robot Operating System* (ROS)[11]. It is a framework that supports especially the development of autonomous robots which use different sensor data that need to be combined for creating appropriate actions. One of its main features is the infrastructure of different nodes that can be used modularly. These nodes can exchange data over so called topics which are published or subscribed accordingly. Furthermore, this architecture allows to run nodes on different machines that can communicate over a master node which is connected with all other nodes over a TCP network.
ROS software is structured in packages. This makes it possible to include publicly available software

[6] `https://docs.rosflight.org/user-guide/ros-setup`
[7] `http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Air#UbuntuCore_16.04`
[8] `http://cdimage.ubuntu.com/releases/16.04.6/release`
[9] `http://cdimage.ubuntu.com/releases/18.04.3/release`
[10] `https://github.com/ryuichiueda/ros_setup_scripts_Ubuntu18.04_server`
[11] `https://www.ros.org`

easily into your own project or to share your own projects with other developers.
Since normally several nodes have to be started simultaneously, the nodes to be executed can be defined in a start file, which can then be executed with the *roslaunch* command.
Moreover, there are suitable tools available with ROS that support the development process. *Gazebo*[12] is an application for 3D simulations. By this, software for robots and sensors can be developed and tested even without having the required hardware. The *ROS Visualization* tool (RVIZ)[13] visualizes data like positions, laser scans and paths. *RQT Graph*[14] helps to get an overview of the running nodes and how they are connected to each other. When it comes to positioning, the hierarchy of transformations plays an important role. The *RQT TF Tree*[15] provides a visualized tree showing the hierarchy of transformation frames.

5.4 ROSflight

For this project, it is required to be able to control the drone using software. For that purpose, the ROS package *ROSflight*[16] is very suitable, since it offers among other useful functions especially the possibility to control a drone via software commands.
A firmware[17] is provided along with the ROS software components, which can be installed on the flight controller. Using a serial connection (MAVLink[18]), a companion computer running the ROSflight IO node can communicate with the flight controller (Figure 19).

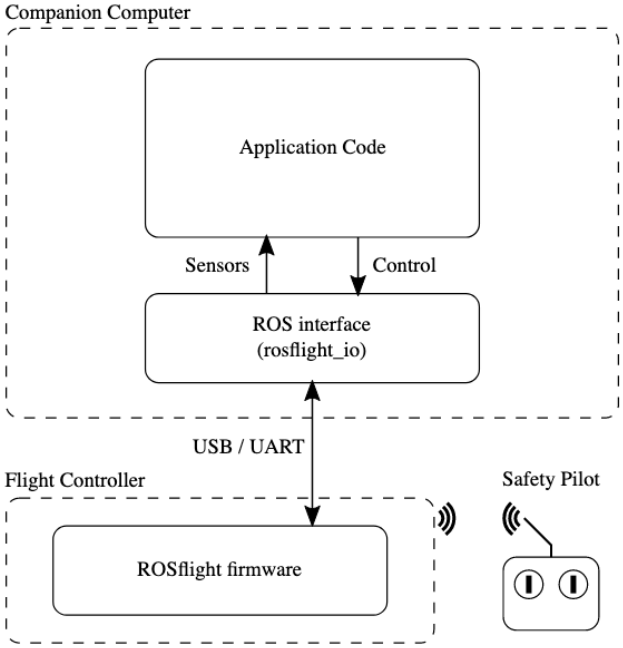


Figure 19: Basic logic of ROSflight[13]

It is possible to download the compiled ROSflight firmware from the github site, but the last release is from May 2018, so it is advisable to compile the latest version of the firmware yourself and flash it to the flight controller. This requires setting up the ARM embedded toolchain and building the firmware with the "gcc-arm-none-eabi" compiler[19].

[12] http://gazebosim.org
[13] https://wiki.ros.org/rviz
[14] https://wiki.ros.org/rqt_graph
[15] https://wiki.ros.org/rqt_tf_tree
[16] https://docs.rosflight.org
[17] https://github.com/rosflight/firmware
[18] https://mavlink.io/en
[19] https://docs.rosflight.org/developer-guide/building-flashing

To flash the firmware to the flight controller the latest version of cleanflight[20] can be used. In case of problems, there is still the possibility to put the flight controller into bootloader mode by bridging the boot pins. Then the STM32 Flash loader demonstrator tool[21] can be used to directly write the firmware to the memory chip[22].

For the firmware on the flight controller to work as desired, some parameters must be set correctly. The ROSflight node offers the possibility to get, set and save the value of parameters:

```
//get Parameter
rosservice call /param_get PARAMETER_NAME

//set Parameter
rosservice call /param_set PARAMETER_NAME

//save Parameters
rosservice call /param_write
```

To configure the flight controller correctly the values of the following parameters must be set:

- FC_ROLL: 180
- FC_YAW: 270
- MIXER: 2
- MOTOR_PWM_UPDATE: 490

FC_ROLL and FC_YAW are offsets for the respective axis and depend on the positioning of the flight controller on the drone. MIXER describes the motor layout of the drone, where 2 stands for the type Quad-X (Figure 20). MOTOR_PWM_UPDATE determines the update rate at which the ESCs are controlled.

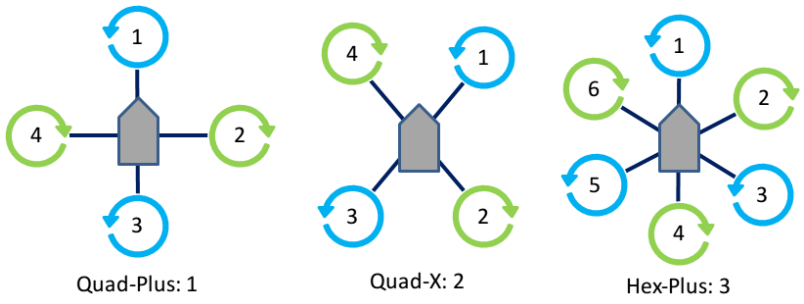


Figure 20: Motor Layouts[11]

5.5 Simulation

The algorithms for autonomous flying should be tested in a simulated environment. This is essential for reducing the security risks that come up when using a real drone. Also, failures can lead to damage of the drone and thus increase the costs. The required components are described in the following as well as how these components are used.
Firstly, a virtual environment is needed including a room and a drone model that can be controlled. ROSflight contains a subpackage called *ROSflight Sim*[23] with a full set up drone simulation for the Gazebo simulation tool. The drone model is defined in a xacro[24] file, where the visualzed objects, nodes and sensor behavior are configured. It also provides IMU data and a Sonar sensor pointed to the bottom.
Arming and controlling of the drone should work independently from any hardware. Therefore,

[20] http://cleanflight.com
[21] https://www.st.com/en/development-tools/flasher-stm32.html
[22] https://www.dronetrest.com/t/how-to-update-firmware-on-the-flip32-naze32-flight-controller/646
[23] https://wiki.ros.org/rosflight_sim
[24] https://wiki.ros.org/xacro

the package *ROSflight Joy*[25] is used. It allows to simulate a remote control either with keyboard or with a joystick. With the following commands, the simulation can be started together with the keyboard control:

```
//start ROSflight main node:
rosrun rosflight_io rosflight_io _udp:=True

//start simulation node
roslaunch rosflight_sim multirotor.launch

//start keyboard RC
rosrun rosflight_joy rc_keyboard RC:=/multirotor/RC _auto_arm:=True

//set flight controller channels accordingly and trigger IMU calibration
rosservice call /param_set ARM_CHANNEL 4
rosservice call /param_set MIXER 2
rosservice call /calibrate_imu
```

Next, for creating a map, the simulated drone requires a LIDAR sensor. A model for the RPLidar sensor (cf. section 4) could not be found. A similar sensor called Velodyne Laser provides a ROS package[26] with a Gazebo model and a node for publishing scan data. As the LIDAR sensor has to be attached to the drone, the xacro file of ROSflight sim need to be customized by referencing the description file of the Velodyne laser (velodyne_description/urdf/HDL-32E.urdf.xacro) and adding a HDL-32E object as follows:

```
<xacro:include filename="$(find velodyne_description)/urdf/HDL-32E.urdf.xacro"/>
<HDL-32E parent="${namespace}/base_link" name="velodyne"
  topic="/velodyne_points" hz="5" samples="100">
    <origin xyz="0 0 -0.01" rpy="0 0 0" />
</HDL-32E>
```

When launching the simulation, a Velodyne Laser is now attached to the drone and the ROS node for publishing laser data is running.
While RPLidar is publishing sensor_msgs/LaserScan data, the Velodyne node publishes sensor_msgs/PointCloud2 data per default. By running a *velodyne_laserscan*[27] node, the data of the Velodyne laser can be translated into the sensor_msgs/LaserScan format.
Finally, the drone is meant to fly inside a room. Otherwise, the mapping would not work. Gazebo provides a Building Editor that allows to easily create a room with custom shapes. In Figure 21 the rooms for testing the algorithms are shown.

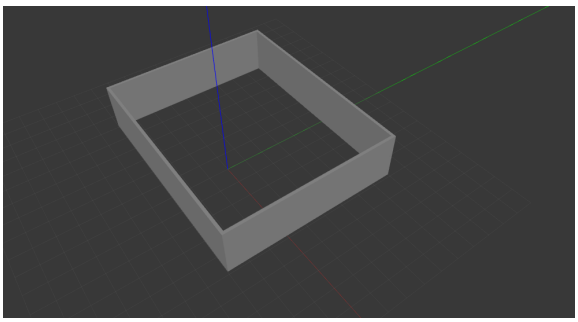
5.6 Take-off and Landing

To realize taking off and landing, the data provided by the sonar sensors are used to estimate the height of the drone. To take off, the throttle of the drone is adjusted to reach a certain predefined height (in our case 1m). Once it reaches its target height, the drone tries to hold it. To land the drone the throttle is set to zero.

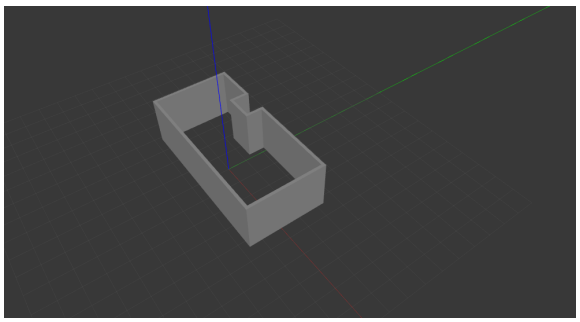
5.7 Map and Localization

A commonly used ROS package to create maps is called *gmapping*[28]. It uses LIDAR data, to calculate a map of the room using SLAM. However, as it requires additional odometry data which is usually provided by ground vehicles, gmapping was not suitable for our case. Instead, another package called *hector_mapping* was found which uses LIDAR data for map creation and IMU data for stabilization. Both are provided by the drone's hardware.
In addition to map creation, hector_mapping also provides localization of the drone within that map. The 2D position of the drone is relative to the map. The mapping and localization has to consider that the robot may be rotated. This is crucial for the stability. In ROS you can define so called transformation frames (TFs) to define the hierarchy. In this project the tranformation hierarchy

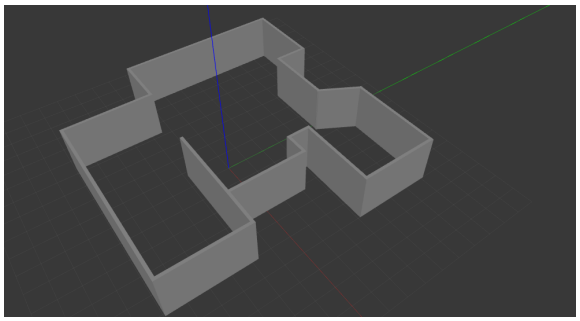
[25] https://github.com/rosflight/rosflight_joy
[26] https://wiki.ros.org/velodyne
[27] https://wiki.ros.org/velodyne_laserscan
[28] https://wiki.ros.org/gmapping



(a) Room 1



(b) Room 2



(c) Room 3

Figure 21: Rooms

is setup as shown in Figure 22. The map-frame represents the world coordinate system. The scanmatcher_frame is the x-y-position of the robot, measured by hector_mapping. Using a Kalman filter, the measured position is smoothed for improving the stability. The robot rotation is defined by the transformation from base_stabilized to base_frame. For that purpose, hector_mapping provides a subpackage hector_imu_attitude_to_tf[29] which uses IMU data to create the according TFs. The multirotor_velodyne represents the position of the laser sensor relative to the robot ground position. In order to create a full map, the drone needs to fly around the room. The process of map creation can be observed using RViz. Figure 23 shows the created map at the start of the scan (a) and the final map (b). As the localization is quite sensitive, the robot must not move too fast.

5.8 Navigation

After the map is calculated by hector_mapping, it can be used for navigation. In this context, navigation means flying towards a point on the map whilst avoiding collision with obstacles. The *ROS Navigation Stack*[30] provides the calculation of a cost map (see Figure 24) which is used to determine a path towards any point on the map while keeping a safe distance around obstacles. The save area can be determined via a collision polygon around the drone.
In order for the navigation stack to work, the *move_base*[31] package has to be installed. Additionally, four different configuration files need to be created for setting different parameters like the update rate of the cost map, the 2D shape of the robot and observation sources including the laser scan topic. The navigation stack does also provide velocity commands, that need to be executed to reach the destination following the path. However, those are more applicable for on-the-ground robots. A flying robot is far more difficult to control. Thus, the navigation was realized by iteratively flying towards points on the path (cf. 5.9). As the resolution of the provided path is quite high, the drone would take a long time to reach its destination when approaching every single point. For that reason, the performance is improved by taking only every fifth point of the path into account.

[29] `https://wiki.ros.org/hector_imu_attitude_to_tf`
[30] `https://wiki.ros.org/navigation`
[31] `https://wiki.ros.org/move_base`

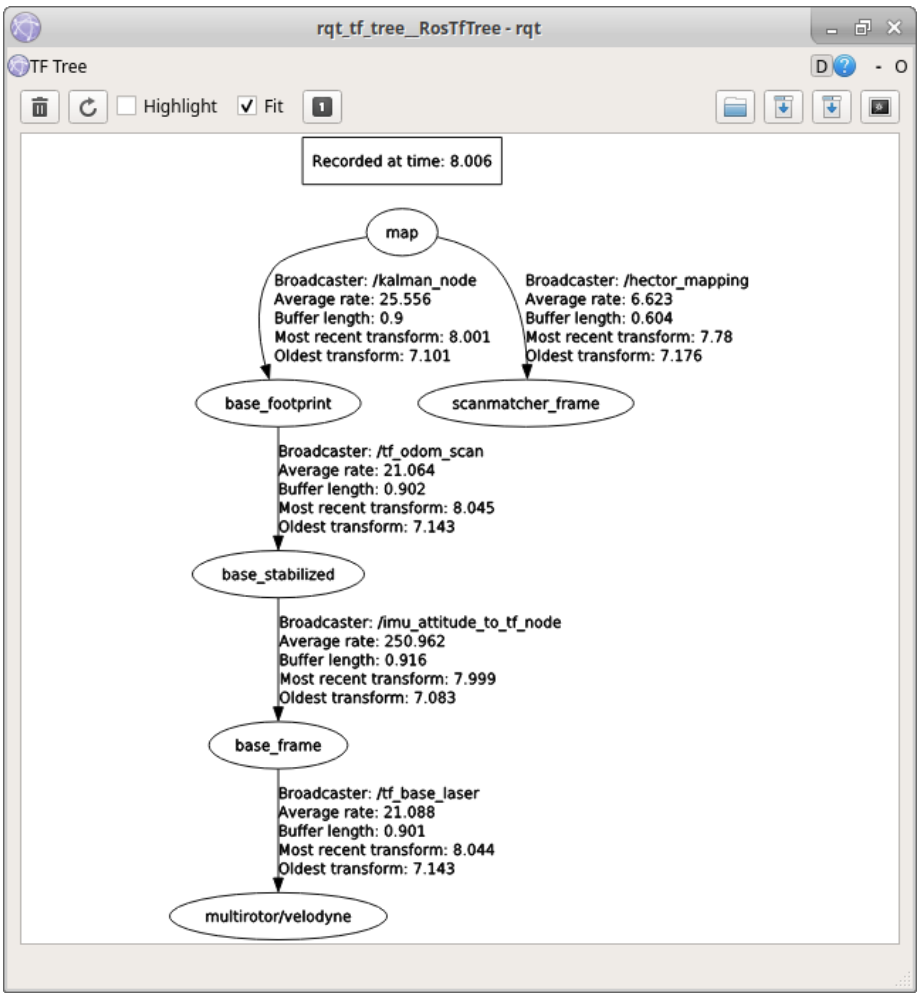
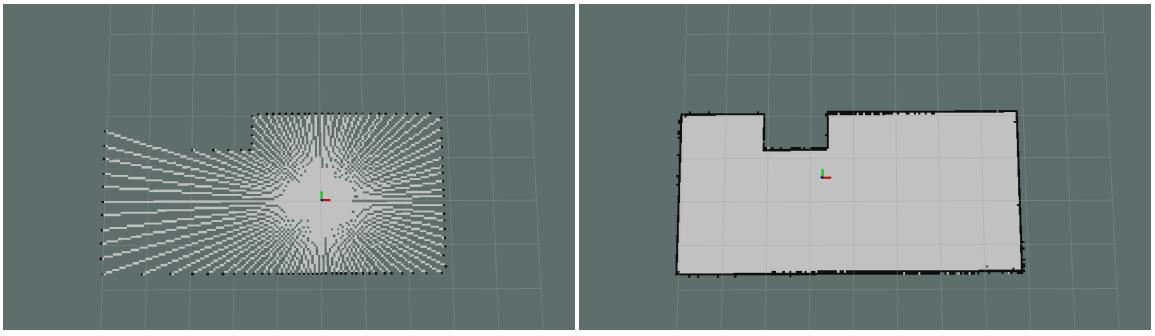


Figure 22: TF Tree



(a) Initial Map

(b) Final Map

Figure 23: Mapping

5.9 Point following

To follow the path provided by the navigation stack, an algorithm that iteratively approaches points along the path was implemented. Depending on where the drone is compared to the target point, the yaw and pitch of the drone are adjusted. Once a point has been reached within a certain tolerance, the drone will start to approach the next available point. If no further point is available, the drone has reached its destination.

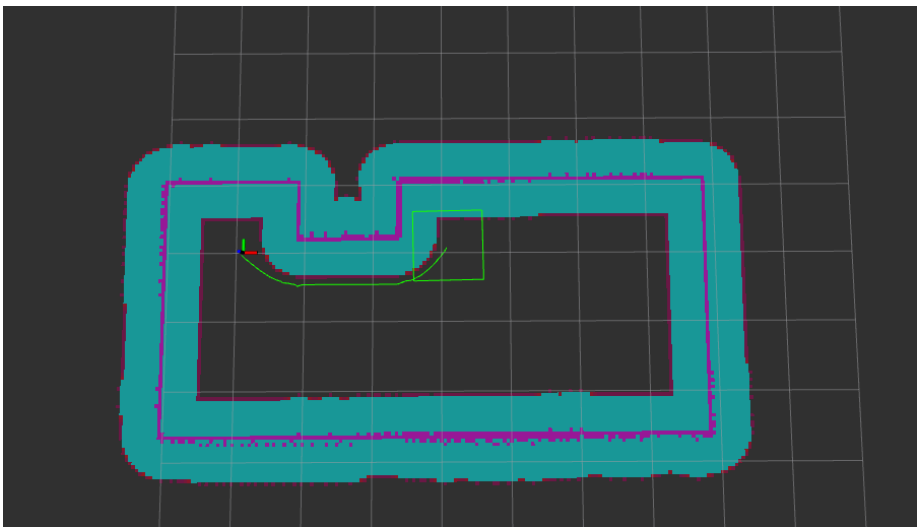


Figure 24: Costmap and Path

5.10 Web UI

To have control over the drone, a web UI was implemented (see Figure 25). Using a websocket the service is capable of both receiving and publishing commands. While simulating, the drone can be reset using the *Reset Drone* button. When used with the real drone, the button's functionality changes to resetting only the map. Furthermore, commands for take-off and landing can be published. Also, it receives the map data from hector_mapping and draws it in a frame. Clicking on a point within that map, publishes the command to calculate a path and finally approach that point.

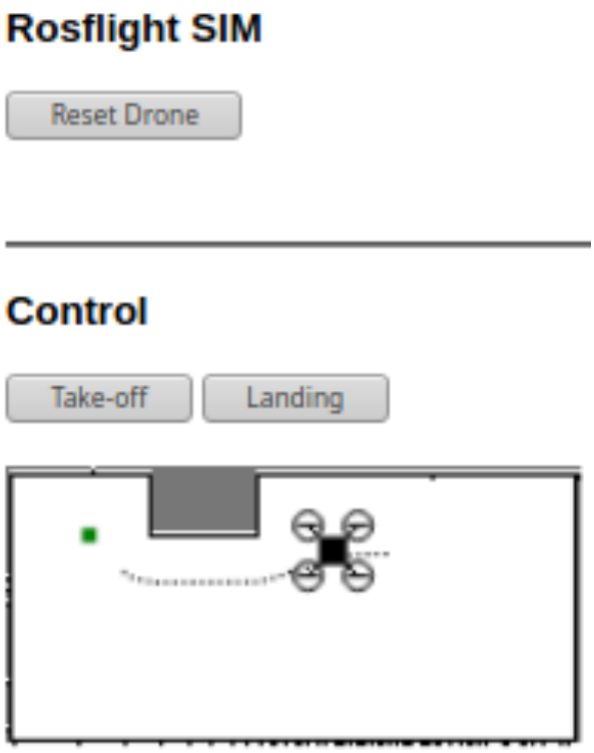


Figure 25: Web UI

5.11 Data Flow

Figure 26 gives a full overview over our software architecture. The central node in this graph is the *Autonomous Flight* one. It gets the IMU and sonar data from ROSFlight as well as the filtered

position from the Kalman filter and the path from the navigation node. The Kalman filter does get its position from hector_mapping which in turn gets the scan data from the Lidar sensor. The filtered position and the map generated by hector_mapping serve as input for the navigation node. Once calculated, the autonomous flight node publishes the necessary fly commands to ROSflight which passes it as raw data to the drone.

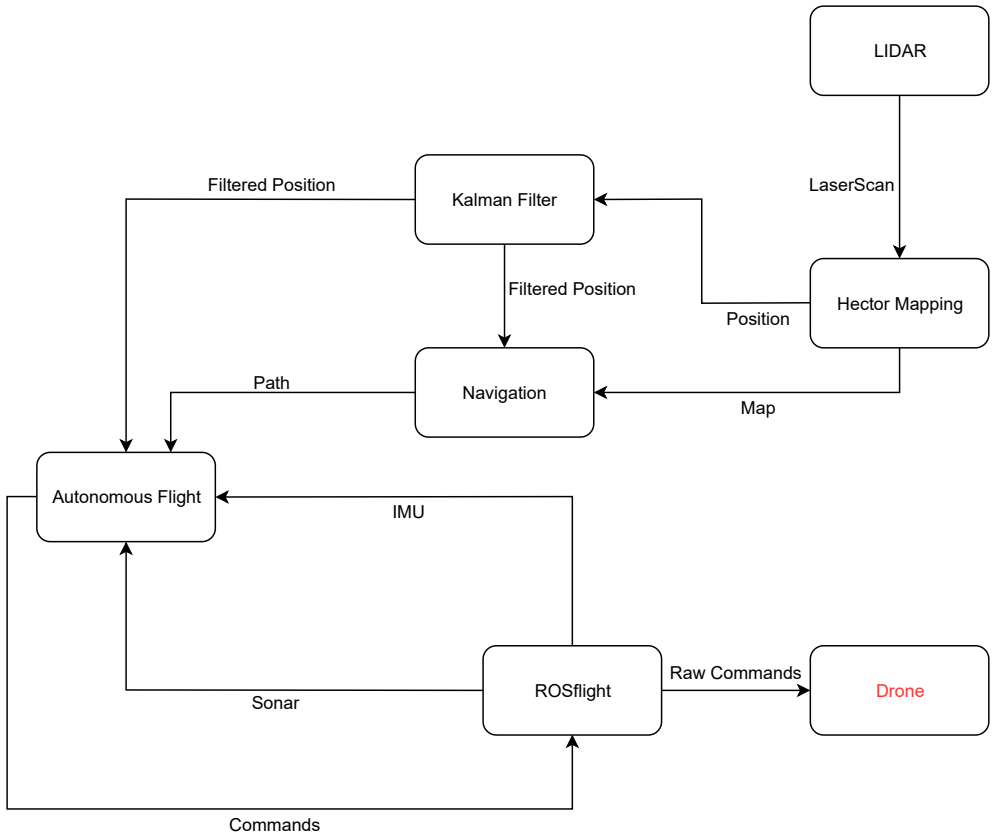


Figure 26: Software Overview

5.12 Sensor Reading

To get the sensor data into the system, special ROS nodes were needed to calculate this data from the sensors and publish it as ROS topics to the system.
Initially, a custom ROS-node based on python was created, inspired by an existing package[32] to calculate the distances above and below the ultrasonic sensors. As the solution turned out to be unstable and the sensors stopped publishing data after a certain time, the python node was replaced by a solution based on C++[33]. The measured distance is published as a ROS topic, using the Range data type from the sensor_msgs package in ROS. This data type contains a specific timestamp for the measurement, the type of radiation used for the measurement (ultrasonic or infrared) the range at which distances were measured and the specific distance value measured by the sensor.
To get data from the 360° laser scanner, the ROS package "rplidar_ros" by SLAMTEC[34] was used and implemented in the sensor environment. The node calculates the time interval between the individual laser pulses. It publishes a ROS topic using the LaserScan data type, also from the sensor_msgs package. The data type contains several information about the angle and distance range, time information, as well as the range and intensity values for each direction.

[32] https://github.com/engcang/HC-SR04-UltraSonicSensor-ROS-RaspberryPi
[33] https://github.com/matpalm/ros-hc-sr04-node
[34] https://github.com/slamtec/rplidar_ros

6 Problems and Limitations

In a project of this complexity, setbacks and difficulties are unavoidable, so here are some problems and limitations that have occurred in the development of the project.

6.1 Hardware

The very limited space on the drone made it quite difficult to fit all components. Therefore 2 revisions of the setup were necessary to find a working solution.
The serial communication between the flight controller and the companion computer was originally intended to be implemented via UART, as it is known from cases where USB connections have become loose due to strong vibrations[35]. This worked well with the Flip32 as flight controller. After switching to the Naze32 as flight controller, this did not work properly anymore and a change to USB as connection was necessary.
As mentioned at the end of section 4.9 there were problems in setting the perfect hover state of the drone. For unknown reasons, the setting of possible offsets due to constructional inaccuracies using RC-Trim calibration[36] could not be applied correctly. The drone adapted the corrections itself after a certain time, which indicates a sensor drift of the gyroscope. Unfortunately, the problem could not be solved by tuning the K_p and K_i gains of the estimator algorithm[37].
Another problem that occurred after mounting the LIDAR sensor was that the IMU of the flight controller could not be calibrated while the measurement was running because the motor of the LIDAR caused too much vibration. The solution to this problem was to stop the LIDAR motor for the calibration of the IMU and then to reactivate it. During the initial autostart, the sequence was adjusted so that the IMU was calibrated once before activating the lidar. For development purposes, a script was written that could stop the LIDAR before IMU calibration and activate it again afterwards.
The differences between the test platform (Raspberry Pi 3+ with Ubuntu Bionic) and the live platform (NanoPi with Ubuntu Xenial) also led to different problems. At various points differences appeared which complicated the porting from one platform to the other. This concerns e.g. the use of the log function of Linux services, which does not yet work in Xenial as it does under Bionic.
Another problem was the control of the LED bar. Most of the software libraries for this special kind of LEDs (WS2812) work with the Raspberry Pi without problems (e.g. the very common neopixel library from Adafruit). On the NanoPi, however, these libraries usually do not work because of a different hardware layout, which is not supported in most cases. After trying 3 different libraries without success, one library was finally able to work by applying a few hacks from a github issue[38].
Furthermore, there were problems with the Python library for reading the ultrasonic sensors. On the one hand the node did not deliver sensor data after an unpredictable time, on the other hand the processes reading the sensors were not stopped when the node was closed. This caused processes to remain active in the background, consuming much of the CPU power. This problem could be solved by switching to Linux services, because all processes are killed hard when the service is terminated. The problem of unstable measurement could only be solved by using another C++ based library. However, this library was also difficult to get up and running, because different GPIO layouts needed to be used in 2 places, which was not documented at all.

6.2 Software

There are some limitations and problems left regarding software. Firstly, the map created is only two-dimensional in contrast to the project goals which contain a 3D map. In addition the simulation of the drone is not perfectly matching the real drone. The LIDAR and sonar sensors are not exactly the same models as on the real drone. Apart from that, the real drone has two sonar sensors pointing to different directions, while the simulation and the software only consider the sonar sensor pointed to the bottom. In the simulation, the tested rooms are clean and do not contain any obstacles. The map looks equal independently from the vertical position of the drone. In a real scenario, this would be different. For that reason, the stability of the implemented algorithms cannot be guaranteed. Another limitation is that the simulated drone currently flies only at a height of 1

[35] https://docs.rosflight.org/user-guide/hardware-setup/#connecting-to-the-flight-controller
[36] https://docs.rosflight.org/user-guide/performance/#rc-trim
[37] https://docs.rosflight.org/algorithms/estimator/#tuning
[38] https://github.com/jgarff/rpi_ws281x/issues/293

meter. It was quite difficult to set appropriate throttle values for keeping this particular height. For other heights, the values need to be changed and tested manually.
In the process of testing the algorithms on the real drone, it was found that the point following does not work on the real drone. The drone could be controlled, but the sensor data let to confusion. A reason for that is probably that the IMU is attached differently than in the simulation, giving false orientation values. The map creation worked quite reliable, even better than in the simulation, and is thus not expected to be the reason of failure. This problem can probably be fixed, but needs a lot more investigation which was not possible within the duration of this project.
Two other problems concern CPU usage. One problem was a service (brcm_patchram_plus), which seems to have something to do with the wireless chip, which used 100% of a CPU core almost every time it booted. First, a script was run to look for this process, and as soon as it was started, it terminated. When it was discovered later in the development process that this process is registered as a Linux service, it could be easily terminated in the autostart-script. The second problem concerns the total CPU usage when all processes are running for autonomous flight. Here the CPU seems to come to its limits, which shows the load of all 4 cores at nearly 100% (Figure 27). Especially the calculation of the map and cost-map consumes a lot of resources. Here it has to be considered whether in the future a companion computer with more power has to be used or if some calculation has to be outsourced to a connected groundstation control computer.

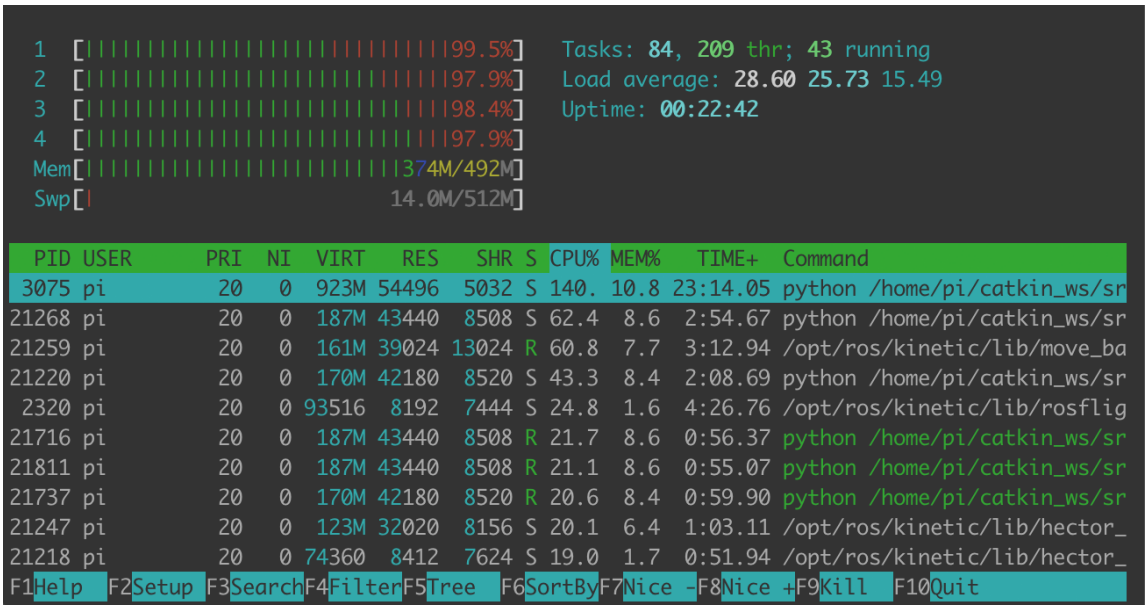


Figure 27: Total CPU load with all processes running

7 Conclusion and Outlook

In this project, the team has learned a lot about drones and the Robot Operating System. A concept for the development of an indoor flying drone could be created and realized in many parts. Although the real drone did not fly autonomously in the end, the drone was equipped with all required sensors. A simulation serves as a proof of concept, showing that the setup is suitable for the implementation of a real autonomously flying drone.
The problem of trim calibration must be solved so that the autonomous flight can be transferred completely from the simulation to the real drone. For this purpose, further research was done and a potentially compatible F4 flight controller[39] was found, which will be supported in the future and can be ordered within Europe. It has to be tested if the ROSflight firmware on it is fully functional and the drone setup has to be modified again.
To increase the performance limit of the Companion computer there is the possibility to switch to a Raspberry Pi 3 Model A+[40], even if it is still much larger than the NanoPi. Unfortunately, there is no more powerful NanoPi in the same size.
In order to make the project even more cost-effective, it might make sense to switch to an RPLIDAR A1 sensor[41], as this is significantly cheaper but should still provide sufficient scanning performance.
In the future, the simulation may be adapted to fit better to the real drone. This means that an additional sonar sensor for the ceiling should be attached to the simulated drone. Also the size and the weight of the drone should be changed according to the real drone.
To make the algorithms work with the real drone, the IMU values need to be observed and the control commands in the software need to be changed. This will need a lot of time, since it is far more complex to test with the real drone, due to the limited battery capacity and the risk of damage.
The team decided to publish the repositories which contain all the implemented software, in order to help other developers that work on similar projects and to get support for possibly realizing all goals of this project in the future.

Figure 28: Drone fully assembled

[39] http://copterjungle.de/f4-revo-stm32f405-flight-controller
[40] https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus
[41] https://www.slamtec.com/en/lidar/a1

List of Figures

List of Tables

References

[1] *amazon.com - HobbyKing AfroFlight Naze32 Rev5 Acro FunFly Controller*. URL: https://amazon.com/HobbyKing-AfroFlight-Naze32-FunFly-Controller/dp/B01MF5QLLA (visited on Jan. 28, 2020).

[2] *amazon.com - Lenovo Flex 2 15.6-Inch Touchscreen Laptop*. URL: https://amazon.com/Lenovo-15-6-Inch-Touchscreen-Laptop-59418271/dp/B00K6ZIL0K (visited on Jan. 28, 2020).

[3] *amazon.de - 4 Paar CW / CCW 5030 Propeller Props*. URL: https://amazon.de/Gugutogo-Propeller-Quadcopter-Ersatzteile-Component/dp/B083NQ7XYL (visited on Jan. 28, 2020).

[4] *amazon.de - 4pcs/Set Propeller Protect 250 RC Racing FPV*. URL: https://amazon.de/gp/product/B07M78WYZN (visited on Jan. 28, 2020).

[5] *amazon.de - Fritzbox Fon WLAN 7112*. URL: https://amazon.de/AVM-Fritz-Fritzbox-WLAN-7112/dp/B0031H653K (visited on Jan. 28, 2020).

[6] *amazon.de - HRB 11,1V 2200mAh 3S 30C Lipo Batterie*. URL: https://amazon.de/HRB-Batterie-Quadcopter-Hubschrauber-Multi-Motor/dp/B07X288HLF (visited on Jan. 28, 2020).

[7] *amazon.de - MP1584EN ultra Small DC-DC 3A power Step-Down*. URL: https://www.amazon.de/gp/product/B01DBW6O2G (visited on Jan. 28, 2020).

[8] *amazon.de - Virhuck 1S-8S Lipo Warner*. URL: https://smile.amazon.de/gp/product/B071ZYC327/ (visited on Jan. 28, 2020).

[9] T.B. Asafa, T.M. Afonja, E.A. Olaniyan, and H.O. Alade. "Development of a vacuum cleaner robot". In: *Alexandria Engineering Journal* 57.4 (2018), pp. 2911–2920. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2018.07.005. URL: http://www.sciencedirect.com/science/article/pii/S1110016818300899.

[10] Guido Croon and Christophe De Wagter. "Autonomous flight of small drones in indoor environments". In: Oct. 2018.

[11] *docs.rosflight.org - Hardware Setup - Motor Layouts Isolation*. URL: https://docs.rosflight.org/user-guide/hardware-setup/#motor-layouts (visited on Jan. 31, 2020).

[12] *docs.rosflight.org - Hardware Setup - Vibration Isolation*. URL: https://docs.rosflight.org/user-guide/hardware-setup/#vibration-isolation (visited on Jan. 31, 2020).

[13] *docs.rosflight.org - Overview*. URL: https://docs.rosflight.org/user-guide/overview/ (visited on Jan. 28, 2020).

[14] *dronetrest.com - Naze 32 Revision 6 Flight Controller Guide*. URL: https://dronetrest.com/t/naze-32-revision-6-flight-controller-guide/1605 (visited on Jan. 30, 2020).

[15] *elektor.de - Raspberry Pi 3 B+*. URL: https://elektor.de/raspberry-pi-3-model-b-plus (visited on Jan. 29, 2020).

[16] *exp-tech.de - RPLiDAR A2M8 360 Grad Laser-Scanner-Kit*. URL: https://exp-tech.de/sensoren/entfernungnaeherung/8948/rplidar-a2m8-360-degree-laser-scanner-kit-12m (visited on Jan. 29, 2020).

[17] *fpv.tv - Flip32 Flight Controller With 32-bit STM32*. URL: https://fpv.tv/flip32-flight-controller-32-bit-stm32-naze32-multirotors (visited on Jan. 28, 2020).

[18] *fpvracer.lt - DYS 10A 2-4S BLHeli ESC*. URL: https://fpvracer.lt/model/dys-10a-2-4s-with-blheli-program-brushless-esc-for-multicopter-748.html (visited on Jan. 28, 2020).

[19] *github.com - ROScopter*. URL: https://github.com/byu-magicc/roscopter (visited on Jan. 31, 2020).

[20] *hackster.io - Introduction to NanoPi NEO Air*. URL: https://hackster.io/idreams/introduction-to-nanopi-neo-air-8b5451 (visited on Jan. 30, 2020).

[21] Md Haque, Muztahid Muhammad, Dipayan Swarnaker, and M Arifuzzaman. "Autonomous Quadcopter for product home delivery". In: Apr. 2014, pp. 1–5. DOI: 10.1109/ICEEICT.2014.6919154.

[22] Kazi Hasan, Abdullah Nahid, and Khondker Jahid Reza. "Path planning algorithm development for autonomous vacuum cleaner robots". In: May 2014, pp. 1–6. ISBN: 978-1-4799-5180-2. DOI: 10.1109/ICIEV.2014.6850799.

[23] *helipal.com - Storm Q330 Classic Quad.* URL: http://helipal.com/storm-q330-classic-quad-with-rc-tx-fpv-system.html (visited on Jan. 29, 2020).

[24] *hobby-wing.com - EMAX MT2204 KV2300 Brushless Motor for RC 250.* URL: https://hobby-wing.com/emax-mt2204-multicopter-motor.html (visited on Jan. 28, 2020).

[25] *hobbyking.com - HobbyKing Totem Q330 Quadcopter Kit.* URL: https://hobbyking.com/en_us/hobbykingtm-totem-q330-quadcopter-kit.html (visited on Jan. 28, 2020).

[26] *hobbyking.com - WS2812 LED Light Board für CC3D und Naze32.* URL: https://hobbyking.com/de_de/ws2812-led-light-board-for-cc3d-and-naze32.html (visited on Jan. 29, 2020).

[27] Jelena Kocic, Nenad Jovičić, and Vujo Drndarevic. "Sensors and Sensor Fusion in Autonomous Vehicles". In: Nov. 2018. DOI: 10.1109/TELFOR.2018.8612054.

[28] *rcgroups.com - Flysky FS-TM10 2.4G 10CH AFHDS RC Transmitter.* URL: https://amazon.de/gp/product/B07FY6C17Z/ (visited on Jan. 28, 2020).

[29] *rcmoment.com - IMAX B6 1-6 Cells XT60 LiPo Battery Charger.* URL: https://amazon.de/gp/product/B01AXUZSO4 (visited on Jan. 28, 2020).

[30] *reichelt.de - Ultraschall Abstandssensor, HC-SR04.* URL: https://reichelt.de/entwicklerboards-ultraschall-abstandssensor-hc-sr04-debo-sen-ultra-p161487.html (visited on Jan. 29, 2020).

[31] *store.particle.io - ARGON KIT.* URL: https://store.particle.io/collections/wifi/products/argon-kit (visited on Jan. 31, 2020).

[32] *technik-consulting.eu - PID-Regler einer Drohne richtig einstellen.* URL: https://technik-consulting.eu/Optimierung/Drohne_PID-Regelung.html (visited on Jan. 31, 2020).

[33] *tesla.com - Autopilot.* URL: https://www.tesla.com/autopilot (visited on Jan. 31, 2020).

[34] *unmannedtechshop.co.uk - Flysky FS-A8S 2.4G 8CH Mini Receiver.* URL: https://unmannedtechshop.co.uk/product/flysky-fs-a8s-2-4g-8ch-mini-receiver-v2-with-ppm-i-bus-sbus-output/ (visited on Jan. 30, 2020).

[35] M. Wasif. "Design and implementation of autonomous Lawn-Mower Robot controller". In: *2011 7th International Conference on Emerging Technologies*. Sept. 2011, pp. 1–5. DOI: 10.1109/ICET.2011.6048466.

[36] *wiki.friendlyarm.com - NanoPi NEO Air.* URL: http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Air (visited on Jan. 29, 2020).