Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Drone Localization in Ad-hoc Indoor Environment

Master's Thesis

Author
Marcell Rausch

Advisor
dr.Gábor Fehér

December 5, 2019

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Rausch Marcell*, szigorló hallgató kijelentem, hogy ezt a diplomamunkát meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2019. december 5.

Rausch Marcell
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Chapter 1

Introduction

A bevezető tartalmazza a diplomaterv-kiírás elemzését, történelmi előzményeit, a feladat indokoltságát (a motiváció leírását), az eddigi megoldásokat, és ennek tükrében a hallgató megoldásának összefoglalását.

A bevezető szokás szerint a diplomaterv felépítésével záródik, azaz annak rövid leírásával, hogy melyik fejezet mivel foglalkozik.

Chapter 2

Literature review

2.1 Basic aviation terminologies

2.1.1 UAV or Drone

UAV is short for Unmanned Aerial Vehicle, in other words an airborne vehicle that is capable of movement without having a pilot on board. Drone is a subset of UAV, a vehicle that is capable of autonomous flight. Usually UAV and the word Drone are used interchangeably in the literature, and it is used so in this thesis.

2.1.2 Body axes used in aviation

In aviation Euler angles are used to describe the orientation of an aircraft. Euler angles are three angles that describe the orientation of a rigid body with respect to a fixed coordinate system. These axes are referred to as Yaw, Pitch and Roll. Rotation around these axes are Yaw, Pitch and Roll angles respectively. All three angles follow the right-hand rule.

Magnetic north is used as reference for Yaw, so an angle of 0° or 360° means that the vehicle is heading North, 90° means it's heading East. Yaw is sometimes called Heading.

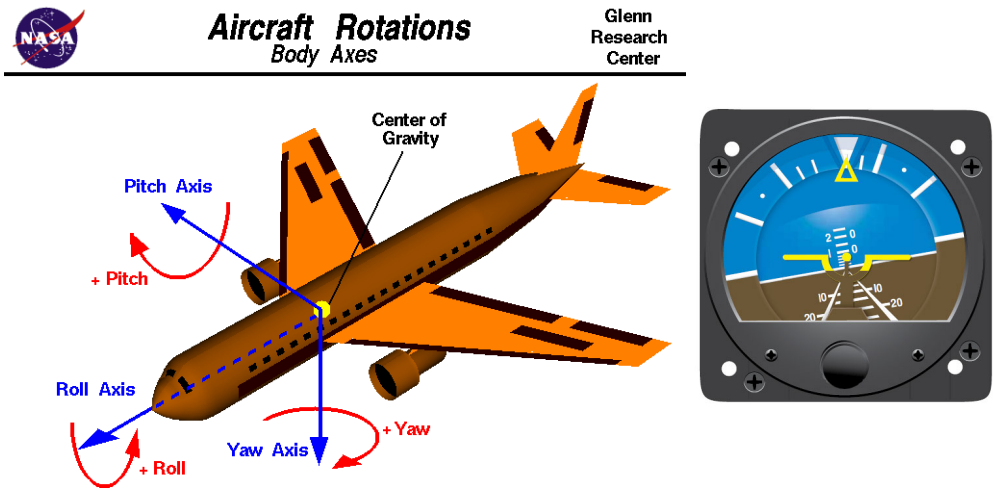

Figure 2.1: Aircraft body axes[12] and attitude instrument

Pitch and Roll angles are calculated in reference to the horizontal plane. The normal vector of the horizontal plane is used for the calculations, and it is the norm of the gravity vector. It is favorable to use the gravity vector, because its direction can be measured using an accelerometer. Both Roll and Pitch angles are measured from -90° to 90°. A Pitch of 90° is straight up and 0° is Horizon. If an aircraft flies with 0° Roll angle, the vehicle is horizontal. On the other hand a 90° Roll angle means the vehicle is turning right and is perpendicular to the horizon. Pitch is sometimes referred to as Tilt.

The limitation of using Euler angles is reached when Pitch or Roll angles approach 90°, because in this case one of these angles become parallel with the gravity vector and the other angle cannot be determined, due to lack of reference. Euler angles and singularities are well described in [4]. To avoid singularities of Euler angles, Unit quaternions can be used. Quaternions are mainly used for calculations, while Euler angles are used to provide humanly readable values.

2.1.3 Ground Control Station

A software running on a ground computer, used for receiving in-flight information via telemetry from a UAV. It displays status and progress of mission, that often includes sensor or video data. It can also be used for sending commands up to the UAV during flight.

Ground Control Station is often referred to as GCS.

2.1.4 Multicopters

Multicopter or Multirotor is a generic term to describe a UAV with more than two rotors. The term covers quadcopters, octocopters, hexicopters etc.

Quadcopters are quickly gaining popularity thanks to the easy to use, commercially available drones that can be used for filming. Octocopters have eight rotors in total and mostly used by professional users. It provides a more stable flight, can lift more weight and it is safer, because it can fly with any of the rotors damaged.

2.2 Simultaneous Localization And Mapping - SLAM

Simultaneous Localization And Mapping the computational problem that asks if it is possible to place a robot in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its position. A good overview of SLAM is presented in[6] and [4] focusing a solution to the above proposed problem in general, with no dependency of sensors to be used.

During SLAM process a robot is placed in an unknown location and it is capable of estimating its trajectory and location of all landmarks online, without the need of priori knowledge of the location. The robot makes relative observations of its surroundings, a number of unknown landmarks using sensors as seen on figure 2.2. The following quantities are defined at a time instant k:

- $\mathbf{x}_k$: State vector describing the location and orientation of the robot
- $\mathbf{u}_k$: Control vector, applied at *k-1* and drove the robot to state $\mathbf{x}_k$ at time k

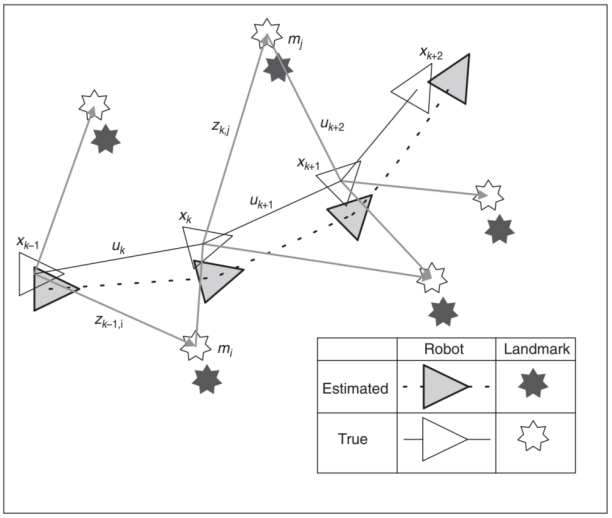

Figure 2.2: The essential SLAM problem presented in [6]

- $\mathbf{m}_i$: Landmark location vector. A time independent vector that describes the location of a static landmark
- $\mathbf{z}_{ik}$: Observation taken from the robot to the ith landmark at time k
- $\mathbf{X}_{0:k} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$ A set of all robot location vectors until time k
- $\mathbf{U}_{0:k} = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\}$ A set of history of all control vectors until time k
- $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_n\}$ A set of all landmarks
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_k\}$: A set of all landmark observations until time k

In probabilistic form of SLAM, the probability distribution function 2.1 needs to be computed for all times k. In other words the current state vector and the landmark location probabilities need to be calculated based on all k observations, control vectors and the initial state of the robot.

$$P\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) \tag{2.1}$$

A recursive solution is desirable for the SLAM problem. Starting with an estimate for the posteriori distribution (equation 2.2) at time k-1, then a control vector $\mathbf{u}_k$ is used to estimate the next state and observations using Bayes theorem. This calculation requires an observation model and a state transition model.

$$P\left(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0\right) \tag{2.2}$$

The observation model 2.3 describes the probability of making an observation $\mathbf{z}_k$ given the current robot and landmark locations.

$$P\left(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}\right) \tag{2.3}$$

The state transition model 2.4 describes the probability of the next position of the robot based on the previous state and the control vector at time k. The state transition is assumed to be a Markov process, so the next state $\mathbf{x}_k$ depends only on the current state

$\mathbf{x}_{k-1}$ applied control $\mathbf{u}_k$. $\mathbf{x}_k$ is also independent from both landmark locations $\mathbf{m}$ and observations $\mathbf{z}_k$.

$$P\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k\right) \tag{2.4}$$

Using equations 2.2, 2.3 and 2.4, the SLAM algorithm can now be described using a two-step recursive form: prediction and correction. The prediction step or time-update is used for estimating the next state and map, based on all posteriori observations, control vectors and initial state.

$$P\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) = \int P\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k\right) \times P\left(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0\right) d\mathbf{x}_{k-1} \tag{2.5}$$

The measurement update provides a correction to the prediction state, using observations made at time k.

$$P\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) = \frac{P\left(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}\right) P\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0\right)}{P\left(\mathbf{z}_k \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}\right)} \tag{2.6}$$

The localization problem can be solved with the assumption the map is known:

$$P\left(\mathbf{x}_k \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{m}\right) \tag{2.7}$$

And the mapping problem can be solved with the assumption the location is known:

$$P\left(\mathbf{m} \mid \mathbf{X}_{0:k}, \mathbf{Z}_{0:k}, \mathbf{U}_{0:k},\right) \tag{2.8}$$

These problems introduced in 2.7 and 2.8 are dependent on each other and need to be solves simultaneously. In the early stages of SLAM mapping and localization were tried to be solved independently and work often focused on either of them. Once the realization came that combined mapping and localization can be formulated as a single estimation problem, the problem became convergent.

Many solutions can be found for probabilistic SLAM problem, the most common is the use of the extended Kalman filter (EKF) that provides an optimal estimation to non-linear models with additive Gaussian noise. The motion model can also be represented with a non-Gaussian probability distribution, that leads to the use of the Rao-Blackwellized particle filter, or FastSLAM algorithms.

The above introduced SLAM problem was a general introduction, it is not dependent on the sensor used for observations. Many kind of observers can be used for this purpose, but the two main categories are visual or camera based and sensor based. Visual observers mostly use simple or special camera configurations for depth sensing, while sensor based observers use some other technique or physical phenomena to measure distances like sonars or LIDARS.

2.2.1 Visual SLAM

Visual SLAM uses a simple camera or a modified camera that is capable of depth sensing. This technique is similar to how humans and animals perceive and their surroundings.

In solutions where a single camera is used, called monocular camera, depth sensing is done by using the differences between consecutive images. Two main categories of monocular techniques are feature-based and direct SLAM algorithms. Feature-based techniques detect certain key-points called features in images, like corners and edges, and only uses these feature to extract depth information. Direct SLAM algorithms use the image intensities to estimate the location and surroundings. LSD-SLAM proposed in [7] is a good example for feature-based, while ORB-SLAM[11] uses direct SLAM approach.

The pro of using monocular camera for localization and mapping is that each image contains high density of information and cameras are cheap and popular. The drawback of using monocular camera based SLAM is that a single camera cannot detect the scale. This is referred to as scale-drift and often attempted to be fixed by trying to detect a scenery that has been previously scanned and therefore drift can be corrected.

Stereo camera or RGB-D cameras are also popular for Visual SLAM, these avoid the scale-drift that monocular cameras suffer from [8].

2.2.2 LIDAR SLAM

Cameras are cheap and easily available, but they produce high amounts of data that demands high processing capabilities. Sensor based approaches produce lower amount of data that already contains depth information without the need for further processing.

Ultrasonic sensors are easy to use for distance measuring, they are cheap and easy to use, but they suffer from significant measurement noise, because of background noise and the change of speed of sound. LIDAR sensors on the other hand use light to measure distance, offer less noisy measurements, lower form factor, higher resolution and update rates. LIDAR sensors are also significantly more expensive than most of ultrasonic sensors and camera based SLAM systems.

The study of University of California[9] presents a data fusion algorithm for estimation of velocity and position of a UAV in urban environments where GPS data is partly or completely blocked and is unreliable. A LIDAR sensor provides local position updates using SLAM technique, GPS provides corrections when available and an Internal Navigation System is used as an additional input to the Adaptive Extended Kalman filter. The outline of the filter can be seen on figure 2.3

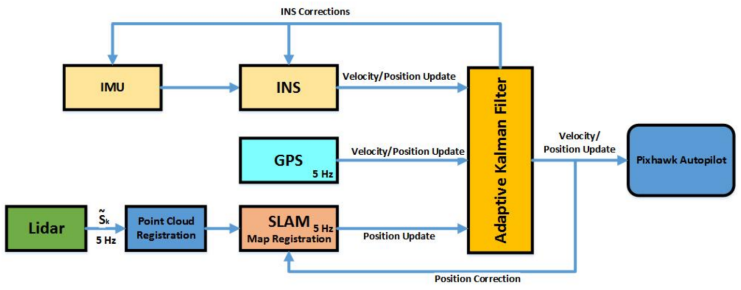

Figure 2.3: Schematic of LIDAR SLAM, GPS and IMU integration [9]

The LIDAR used for measurements is a Velodyne VLP-16, that has a vertical field of view of 30° with a vertical resolution of 2°. On the horizontal axis it's resolution is 0.1-0.4° and rotation rate can be adjusted between 5-20Hz.

The proposed filter is was capable of reducing the GPS drift of 24.3m and LIDAR drift of 7.5m to 3.42m. This shows that fusing GPS, LIDAR and IMU measurements results in a more accurate position estimate in GPS-degraded environments. The capabilities of the LIDAR was tested on an even moon like surface, with very few objects that can be used by the SLAM algorithm. In this case GPS data is much more accurate, than LIDAR SLAM results.

	Drift	Error %
GPS	24.3m	6%
LiDAR	7.5m	1.8%
AEKF	3.42m	0.84%

Figure 2.4: GPS, LIDAR and EKF position estimates and position drift after 405 meters[9]

2.2.3 Cartographer SLAM

Cartographer is a system developed by Google that provides real-time simultaneous localization and mapping in 2D and 3D across multiple platforms and sensor configurations. Cartographer is used by Google street view for internal mapping of building interiors, using a backpack mounted LIDAR scanner.

There officially available SLAM packages in ROS, the two most popular packages are Gmapping and Cartographer. Gmapping package contains a ROS wrapper for OpenSlam's Gmapping that provides laser based SLAM as a ROS node. Gmapping is limited to 2D map building, therefore it is not considered for this project.

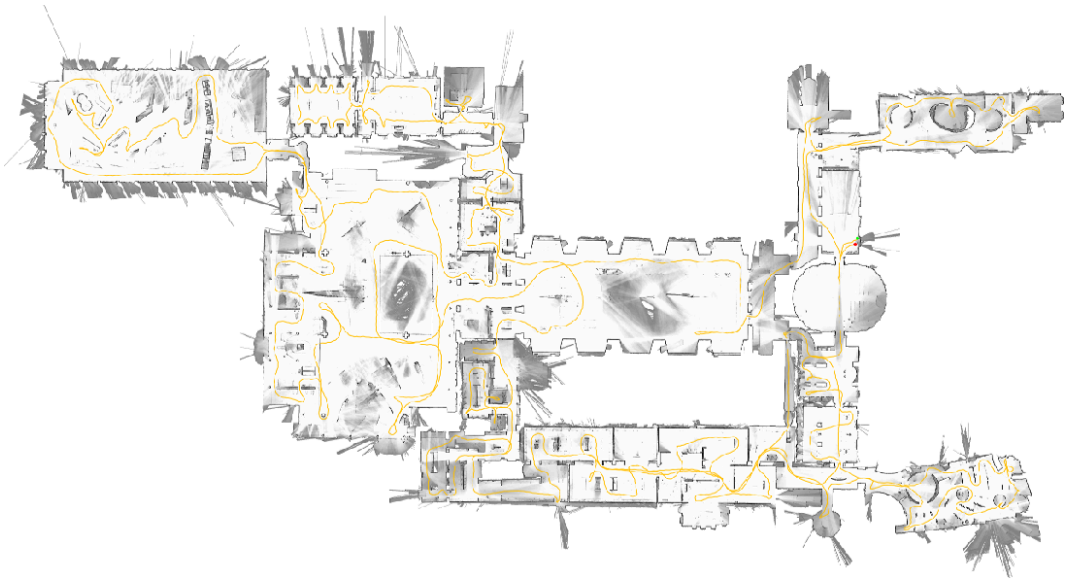

Figure 2.5: Cartographer SLAM example[15]

On the other hand Cartographer ROS package supports both 2D and 3D SLAM and has a great and active community. The overview of Cartographer is seen on figure 2.6. The core components are sensor inputs, Local SLAM and Global SLAM. On a higher abstraction the job of local SLAM is to build submaps and the job of global SLAM is to tie submaps the most consistently together. It is common to build submaps or local maps in SLAM implementations, because building a single map increases complexity quadratically, but by using submaps the complexity can be limited to the contents of the submap.

The most important input of Cartographer is the Range Data, depth information coming from LIDAR sensors. The data is first pre-filtered, because some measurements are irrelevant the sensor might be directed to a part of the robot or it can be covered by dust. Some sensors set unsuccessful range measurements to a value that is significantly higher than the maximum range of the sensor. A bandpass filter is used for pre-filtering, that keeps the values in a predefined range. The minimum and maximum values need to be set according to the sensor specifications. Close objects are very often hit by the LIDAR measurements and offer more points, however distant objects offer much less points. The Voxel filter downsamples raw points where density is higher to reduce computation needed.

Inertial Measurement Units (IMU) provide a direction of gravity vector and a noisy estimation of the robot's rotation. IMU data has to be provided for 3D SLAM because it greatly reduces complexity of scan matching, while IMU data is optional for 2D SLAM. Odometry Pose and Fixed Frame Pose are optional inputs of Cartographer, that can further reduce complexity.

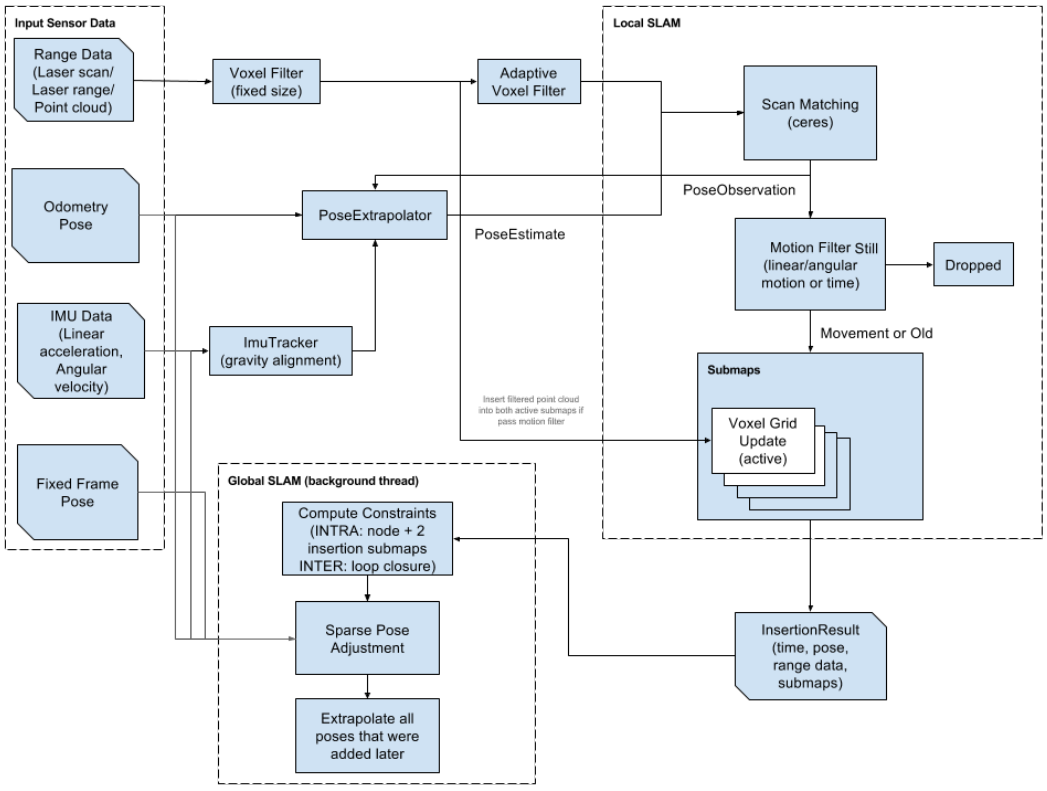

Figure 2.6: Cartographer SLAM overview[15]

2.3 Similar products available on the market

2.3.1 Terabee TeraRanger Tower

TeraBee offers an off-the-shelf solid-state LIDAR system with the purpose of collision avoidance for drones. In their solution 8 sensors are evenly distributed around the vertical axis with a controller board in the middle. TeraRanger's interface is compatible with Pixhawk 4 flight controller board and PX4 flight controller software, that makes integration easy into systems based on these.

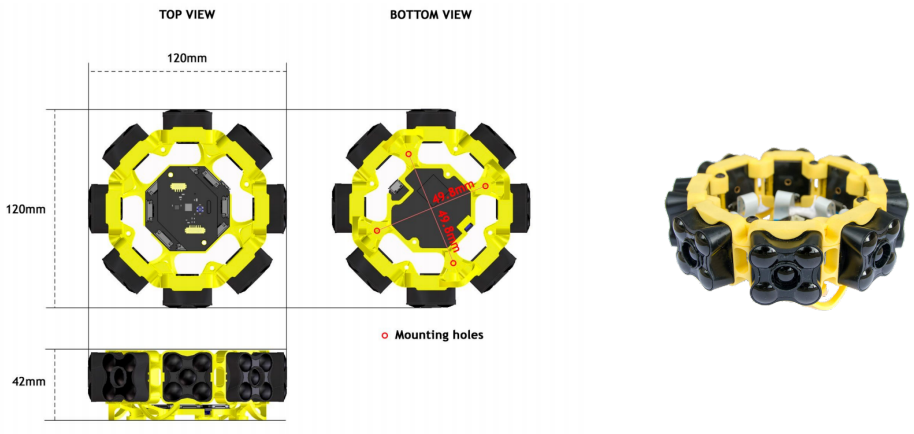

Figure 2.7: Terabee TeraRanger Tower Evo dimensions

Each block of the array is a standalone LIDAR sensor, that can be used separately and supports different mount configurations. The company offers a long-range and fast-ranging version of these sensors, depending on the type an update rate of 320 Hz can be achieved. The fact that system comes with a ROS package and a 2° field of view makes it a potential candidate for indoor mapping and positioning in two dimensions. The price of this setup starts from 599 euro[17].

	Long-range	Fast-ranging
Range	0.5m up to 60m	0.75m up to 8m
Update rate	120Hz/sensor	320Hz/sensor
Field of View	2°	2°
Accuracy	$\pm 4cm$ in the first 14m, 1.5% above	$\pm 12cm$

Table 2.1: TeraRanger Tower Evo specifications

Terabee provides a tutorial video on their website[17] how to connect the sensor array to a UAV that uses Pixhawk 4 flight controller and the process of configuration in two different GCS programs. I have learned that PX4 flight stack supports lidar measurements for obstacle avoidance and it can be configured from a GCS program. It seems a reasonable choice to integrate this feature into such product.

2.3.2 Crazyflie Multi-ranger deck

The company Bitcraze has developed a mini quadcopter mainly for educational purposes. The current version is called Crazyflie 2.1 and measures only 92x92mm with a height of

29mm and a weighs 27g. Extra sensors and peripherals can be attached to the top of the quadcopter using extension boards.

The extension board called Multi-ranger deck has 5 VL53L1X sensors by STMicroelectronics facing forwards, backwards, left, right and up. This project is similar to the product of Terabee described in 2.3.1, but in a smaller size factor and with significantly lower weight.

An introduction video can be found on the product website [10], where a SLAM algorithm is used to create a map and localize the drone. This serves as a proof of concept, that static VL53L1X sensors can be used for mapping and positioning in two dimensions. The company provided no information of the SLAM algorithm used or from the quality of the map.

Figure 2.8: Crazyflie Multi-ranger deck, SLAM example project

Chapter 3

Simulation

Before actually building the system and experimenting on an expensive quadcopter in real world environment, it is cheaper, safer and easier to do tests in a simulated environment.

The PX4 Firmware repository[2] contains Gazebo simulation files to give developers a head start in simulation of their product. Upon these simulation quadcopter models LIDAR sensors can be placed and moved around with ease, without the need of wiring or external infrastructure. This simulation is suitable for Software in the loop (SITL) testing, meaning that the same software developed for the simulation can be used on a real quadcopter.

3.1 Robot Operating System

Robot Operating system (ROS) is a set of software libraries and tools that help developers to build robust general-purpose robot applications[16]. ROS provides hardware abstraction of the underlying robot, generalizes the interfacing of these robots and allows simple high-level usage.

The core component of ROS is a public-subscribe communication protocol similar to MQTT protocol used on the field of IoT. The ROS system is comprised of a number of independent communicating parties called nodes. Each node can subscribe to multiple topics and receive a stream of messages published to these topics by other nodes. For example a temperature filter node might subscribe to a topic of "/temperature_sensor" and publish the filtered temperature values to a topic called "/filtered_temperature_sensor". The benefit using this solution is that the nodes achieve communication exclusively via this protocol so each node lives independently from another. Nodes in ROS do not need to be on the same system or even on the same architecture. Nodes can be run on different computers or even on microcontrollers or smartphones making the system flexible.

To start a ROS session, a ROS Master needs to be started first. After starting the master, nodes can be started and each one registers the topics it wants to subscribe/publish to. The master handles these requests and helps the nodes to establish connection between each other so communication can be started.

ROS is language independent so nodes can be implemented in different programming languages. While C++ is the most common language for product development, official Python wrappers can be used for fast prototyping.

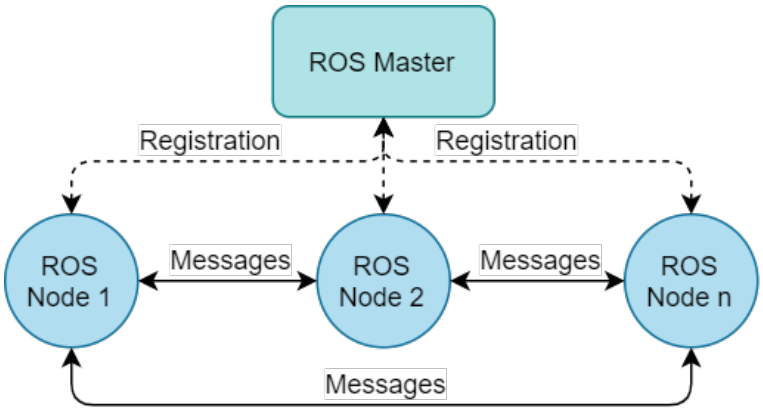

Figure 3.1: ROS publish subscribe communication

3.2 Gazebo Simulator

Gazebo[14] is an open-source 3D simulator for robotics, with the ability to accurately simulate robots in complex indoor or outdoor environments. It is similar to game engines, but produces more realistic, physically correct behavior. Gazebo is free, open-source, actively developed and has gained high popularity during the last years.

Gazebo comes with a growing number of robots and sensors. For simulations one can choose from the officially available robots or create a custom robot using SDF files. Any of these robots can be customized and any number of sensors can be added. Using simulations sensor data can be generated and development can be started without having the actual hardware.

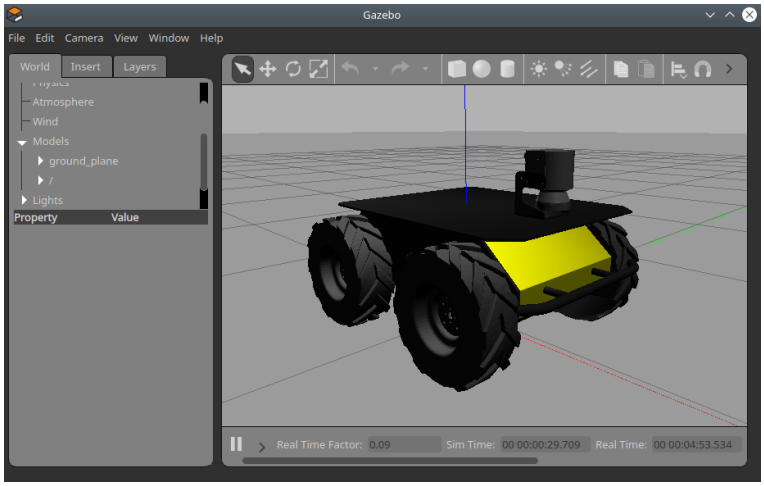

Figure 3.2: Gazebo ROS Husky robot demo

Why Gazebo is particularly interesting, is because it can be started in a ROS compatible mode. Gazebo started with ROS wrapper will actively channel all sensor readings, states and many more to ROS topics, allowing nodes outside of the simulation to subscribe and publish to these topics. Using this technique nodes can be developed in a truly hardware independent manner. The simulation can be replaced by a real robot and the nodes developed during this phase can be used without changes on the real hardware. For example a Husky robot can be simulated in Gazebo and controlled through ROS. The Gazebo simulation of a Husky can be seen on figure 3.2.

3.3 PX4 Autopilot Software

PX4 is a powerful open-source autopilot flight stack. PX4 can control many different vehicle frames and vehicle types, supports wide variety of sensors and peripherals. The project is a part of Dronecode, a nonprofit foundation working on open-source projects to achieve standardization of drones.[3]

The core component of PX4 is the flight stack or flight controller that controls the motors based on sensor measurements and executes commands received from ground control. Besides the flight stack the repository also contains Gazebo simulation files of multiple vehicles and an extensive `Makefile` that makes starting simulations relatively easy.

3.3.1 PX4 simulation using MAVROS

The logical units of the simulation can be seen on figure 3.3. By executing a `make` command the Gazebo Simulator, the PX4 SITL flight stack and the API/Offboard units are started at once. The PX4 SITL flight stack connects on a TCP port to the simulator, receives sensor data, calculates and sends motor and actuator values to the simulated vehicle. There are two options for the navigation of the simulated drone: using an Offboard API or a Ground Control Station. For autonomous flight control an Offboard API needs to be used and MAVROS is a suitable choice, it basically offers a ROS interface for the drone. It forwards all messages from the flight stack to ROS topics and vice versa.

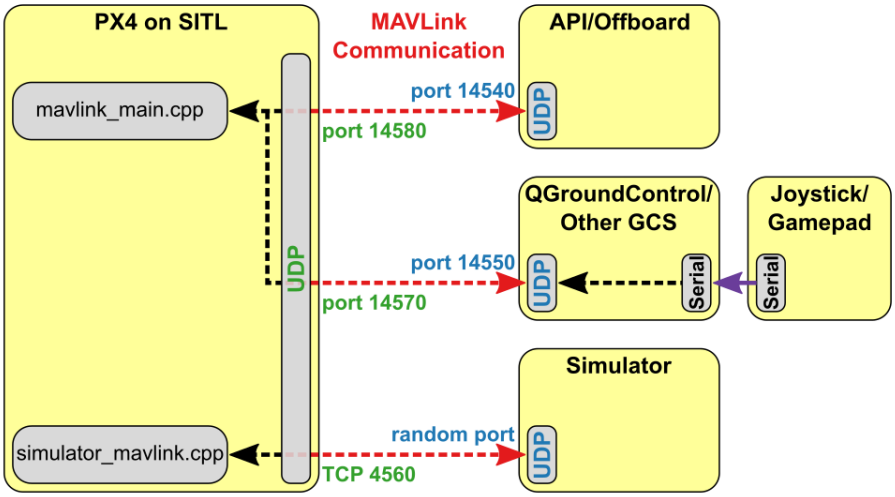

Figure 3.3: PX4 SITL Simulation Outline [5]

3.3.2 PX4 simulation using Gazebo with ROS wrapper

Drone models are described using SDF files that are XML based and by modifying these files new sensors can be added. Gazebo loads the selected vehicle and the sensors, but if a selected sensor is not supported by PX4, its measurements will not be forwarded to MAVROS and therefore to ROS topics. It can be troublesome to always use sensors that are supported by the flight controller, not to mention that the MAVLink protocol is designed for small messages so the bandwidth can be too low in some cases. Luckily Gazebo can be started with a ROS wrapper and therefore all sensors can be accessed on ROS topics. However this technique cannot be used on a real drone with the original

PX4 firmware, because it purely relies on MAVLink messages, but for simulation purposes it's an elegant solution. This modified overview of the units used for simulation and a screenshot of an Iris drone simulated in Gazebo can be seen on figure 4.1. MAVROS is still needed to be run, because there are some messages that would not be published to ROS otherwise.

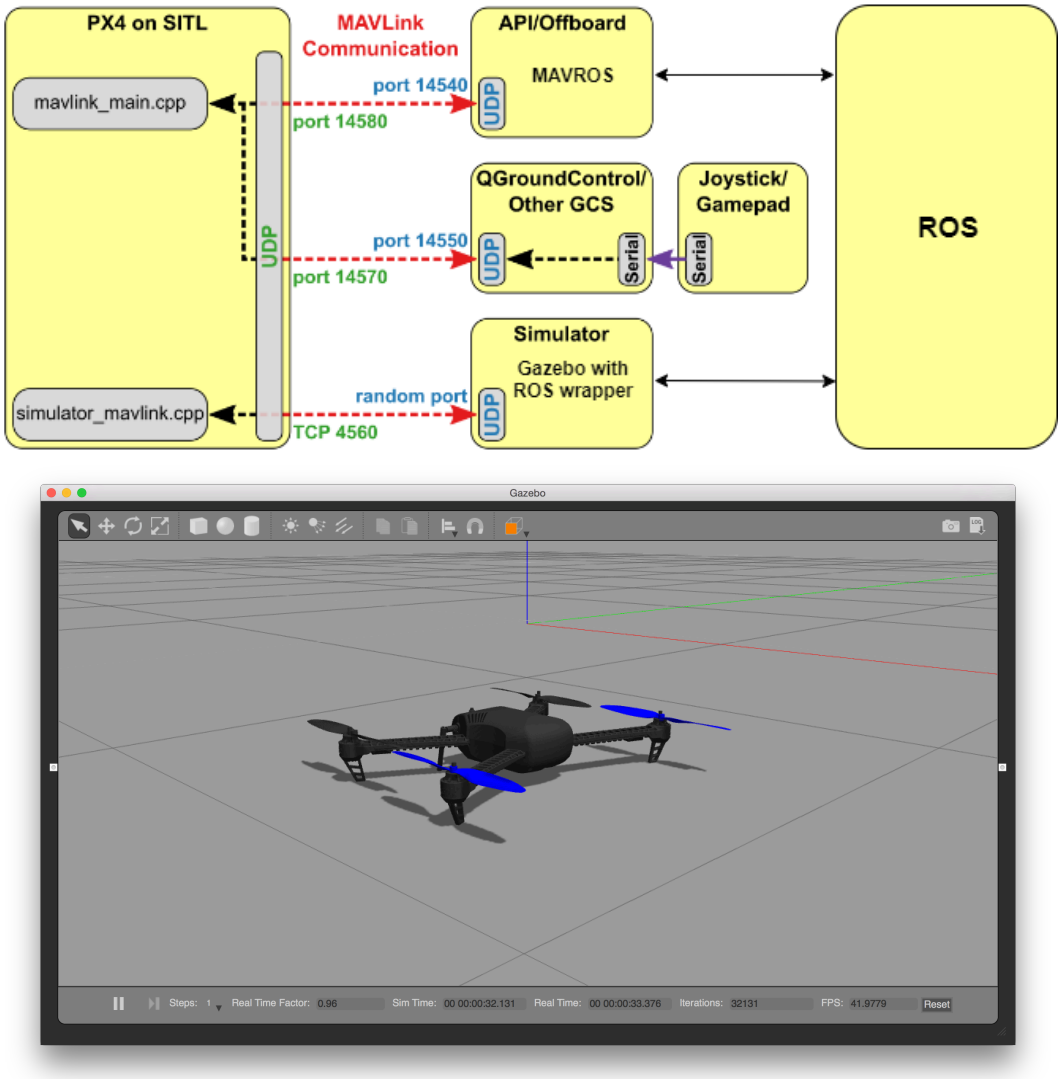

Figure 3.4: PX4 Simulation using Gazebo with ROS wrapper

3.4 QGroundControl

QGroundControl(QGC) is an open-source ground control station software, developed as a part of Dronecode. It provides full flight control and vehicle setup for PX4 and ArduPilot powered vehicles and for any flight controller that uses MAVLink communication protocol. It provides easy and straightforward usage for beginners, while delivering high end features for experienced users too.

QGC is mainly used for flight controller setup and configuration. When setting up a PX4 autopilot based flight controller, the controller needs to know the type of the vehicle and the frame that is used. Besides selecting vehicle type, internal sensors need to be calibrated

and telemetry messages need to be configured. Low level parameters can also be adjusted and obstacle avoidance can also be setup here, as seen in the product of Terabee in 2.3.1, but even low level flight controller parameters can be changed.

QGC can also be used for mission planning for autonomous flight and gives instant feedback of the progress of a mission by visualizing the drone on a map and the most important sensor data.

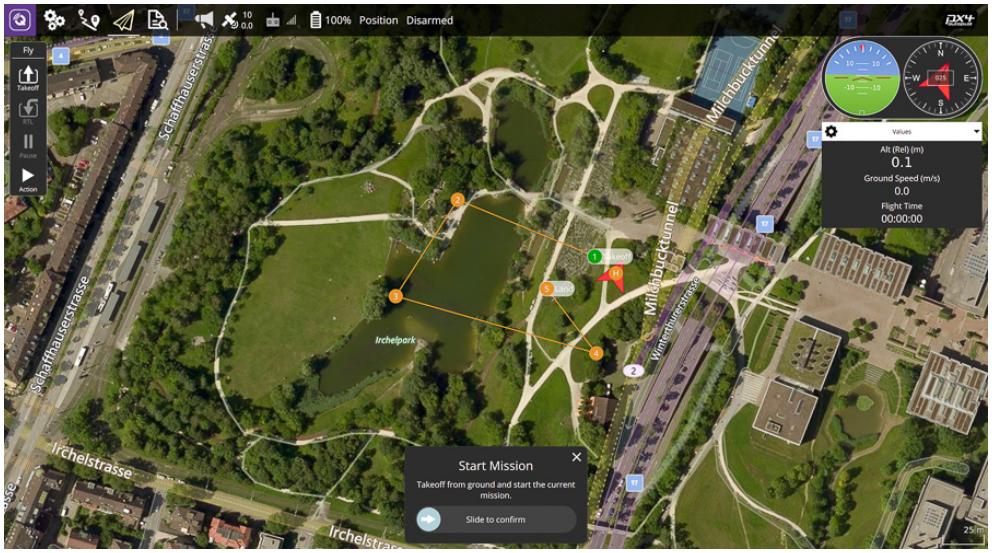

Figure 3.5: QGroundControl main screen[13]

Chapter 4

System design

In this chapter I'm introducing the initial design of the above mentioned system from both hardware and software aspects. Since there are several unknown factors during this phase, the designs described in this chapter are presented as initial designs. The actual system design will be based on the experience gained during several trial-and-error based iterations in simulations.

Multiple aspects need to be taken into account to determine the number and the placement of the LIDAR sensors. The specifications of the LIDAR and the characteristics of SLAM algorithms also drive these aspects. Higher sampling rate can be achieved by placing multiple sensors close to each other in a way that their scanned trajectories overlap. The resolution needed by the chosen SLAM algorithm also affects the number and placement of the sensors, the resolution can be increased by programmatically narrowing down the field of view of the LIDAR.

Quadcopters have a natural tendency to increase their pitch and roll angles when moving around on the horizontal plane. By determining the characteristic pitch and roll angles during an indoor flight, the placement of the sensors can be refined. This way fewer LIDARs are needed for scanning the whole range. Using fewer sensors is beneficial, it reduces complexity in hardware and software too. While during simulations we don't need to worry about physically attaching the sensors on the frame or even supplying power to them, in a real implementation these might be limiting factors.

On a real drone LIDAR measurements cannot be sent straight to ROS topics, because a low level communication needs to be established to configure and read out measurements from the sensors. There needs to be a device that handles communication with the sensor and forwards messages to ROS topics. This device can be the Pixhawk 4 autopilot board using a custom firmware or a custom built hardware independent from the autopilot board or a combination of these two.

4.1 Pixhawk 4

Pixhawk 4 is an advanced autopilot designed by the collaboration of Holybro and PX4 team [1]. It is optimized to run the official PX4 flight stack. Because of its modular-, open-design and open-source firmware, it is a good choice for rapid development and academic purposes. Its modularity comes from the connectors on the top of the device that are used to supply power for the board, output PWM signal for the motors and to

add peripherals like GPS antenna, telemetry or integrate custom devices using available bus connectors.

When moving to real world applications, the PX4 SITL flight stack seen on the most left block on figures 3.3 and 4.1, is to be replaced by the Pixhawk 4 board.

Figure 4.1: Pixhawk 4 advanced autopilot

4.2 VL53L1X LIDAR sensor

VL53L1X is a long distance ranging Time-of-Flight sensor produced by STMicroelectronics [19]. The size of the module is 4.9x2.5x1.56mm, it emits 940nm invisible laser and uses a SPAD(single photon avalanche diode) receiving array with a field of view of 27°. The module runs all digital signal processing on a built in low power microcontroller that frees up the host processor from these tasks. The sensor according to the datasheet is capable of 50Hz ranging frequency and up to 400cm distance measurements. The laser receiving array's size can be programmatically changed by setting region-of-interest(ROI) size. This way the sensor provides multizone operation and a higher resolution than by using the whole SPAD array.

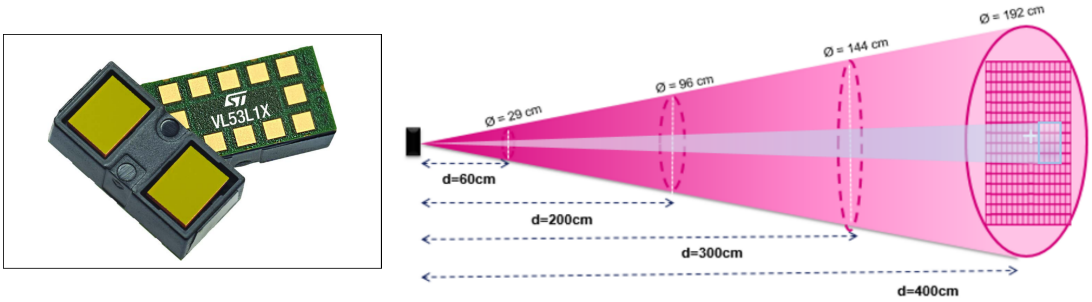

Figure 4.2: VL53L1X sensor [19] and region-of-interest [18]

I chose to use this sensor for prototyping, because of its small size, availability, low hardware infrastructure needs and previous experience.

4.3 LIDAR layout design

As mentioned before, multiple aspects need to be taken into account when placing LIDAR sensors onto a quadcopter and use the measurements for 3D SLAM. There are several parameters of the sensor that are highly affecting the quality of the map built by SLAM

algorithms. These are the field of view, the sampling frequency and the maximum distance that it can measure. To better understand the effects of these parameters it is reasonable to first reduce complexity and build a system for 2D mapping. Based on the experience gained during this experiment it can be better estimated how many sensors are needed for 3D SLAM. Also a possible outcome is that VL53L1X is not suitable for this purpose and a different sensor needs to be used, with more suitable parameters.

4.3.1 Layout design for 2D SLAM

To have the best possible layout without overlapping scans, 360° of the horizontal plane needs to be covered by the field of view of the LIDAR sensors. The field of view of VL53L1X is 27°, so altogether 13 sensors are needed to cover 97.5% of the plane, with 27.69° in between, leaving only 9° of dead zone.

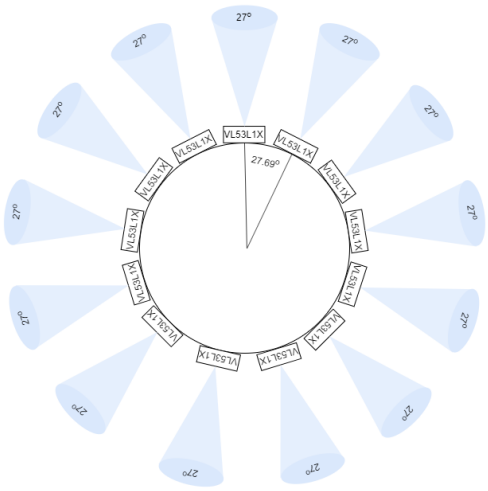

Figure 4.3: Layout of 13 LIDAR sensors

In this layout the scanning on the horizontal plane is done purely by evenly placing the sensors around the vertical axis and doesn't take advantage of the movement of the drone during flights. Especially turns around the vertical axis. The usage of 13 LIDAR sensors just for 2D scanning would mean a lot of overhead when stepping up to 3D, so a more optimal solution needs to be found, that takes advantage of the movement of the drone during turns and therefore uses fewer sensors.

The lower limit for the number of sensors is what can be seen on the solution of Bitcraze, the Multi-ranger deck[10] uses 4 sensors on the horizontal plane facing forward, backwards, left and right. In their demonstration video, the map of a simple room is built in about 20 seconds with update rate of about 10Hz per sensor.

4.3.2 Layout design for 3D SLAM

The layout for 3D SLAM depends on the experience gained during simulations of 2D SLAM. In 3D layout, the pitch and roll angle changes need to be taken into account too. These differ from vehicle to vehicle, faster drones can have a pitch of 30°, but in indoor environment nor pitch nor roll will go above 15°.

TODO Basic layout to start simulation with

4.4 LIDAR data collection

On a real drone configuration of LIDAR sensors and measurement forwarding to ROS need to be solved. There needs to be a device that communicates with the VL53L1X LIDAR sensors using I2C communication protocol and forwards messages to a ground station that handles data collection and processing.

The data collector device needs to be fast enough to keep up with the pace of 10-20 sensors, each with a maximum of 50Hz update rate. The distance data from VL53L1X sensor consists of 12bytes, so 20 sensors on 50Hz generate about 12kb of data per second. This load is calculated using the effective data size, but on the I2C bus other messages need to be sent to trigger data read, which means even higher load.

To evaluate the SLAM algorithms on real-world measurements, data collection needs to be done. For evaluation it is not necessary to send data in real-time on wireless network, but saving data on an SD card serves this purpose. Logging on SD card has low complexity and high speed.

PX4 firmware is an open-source firmware, it seems an obvious choice to use the Pixhawk 4 autopilot board as a data collector device. It has an SD card slot for logging and I2C connectors available to communicate with the LIDAR sensors and even comes with an I2C splitter board to connect more sensors. The microcontroller inside Pixhawk 4 is powerful, it has an ARM Cortex-M7 core running on 216MHz. It is used to run highly timing sensitive control loops to produce actuator values for stable flights, besides many other tasks. If these control loops are delayed by other processes, that can cause instable flight or even crash.

Because of lack of deep understanding of PX4 firmware, to make sure that timing of control loops are untouched, I decided to build a standalone data collector device. A dedicated microcontroller that handles I2C communication with the LIDAR sensors and writes data to an SD card using SPI protocol.

4.4.1 Standalone data collector design

In I2C protocol every device needs to have a 7 bit address, that needs to be unique on that specific I2C bus. VL53L1X sensor has a default address of 0x29 when the sensor is booted, but it can be changed by using I2C commands. Address conflicts need to be avoided by having exactly 1 sensor active per channel. It is hard to find a microcontroller that has 20 I2C buses, one for each sensor, so multiple sensors need to be placed on the same bus. By having multiple sensors on the same channel, the sensors need to be released from reset one by one to change their addresses, this way address conflicts can be avoided. VL53L1X can be kept in reset by pulling its shutdown pin to ground and activated by bringing it to supply voltage.

The sensors have altogether 4 lines that need to be driven: 2 I2C, 1 shutdown and 1 interrupt pin. In the simplest case, by connecting each wire to the microcontroller, 80 wires would be needed for 20 devices and would take 42 pins of the microcontroller. Although it is possible to find a microcontroller that has enough pins and connect all sensors one by one to the same I2C bus, but by grouping them into groups of 5 and connecting each

group to different I2C buses, a significant improvement can be achieved on the number of connections.

Sensors on different I2C buses can have the same address, therefore the same 5 addresses can be used in each group. Shutdown pin of sensors sharing the same addresses can be common, this way number of wires needed for shutdown is reduced from 20 to 5.

Handling 20 interrupt lines can be overwhelming for any CPU, while bringing little or no extra benefit. Using only one interrupt line per group is enough to signal the microcontroller that measurements are ready to be read out in all sensors in that group. Interrupt pins are active-low, so to produce the group interrupt signal a NOR gate needs to be used. When the state of all interrupt pins are low, group interrupt signal will become logical 1 and 0 otherwise.

In the simplified design seen on figure 4.4 only 17 wires are leaving the microcontroller, while no functionality is lost. This solution also makes software development easier and the built system is clearer.

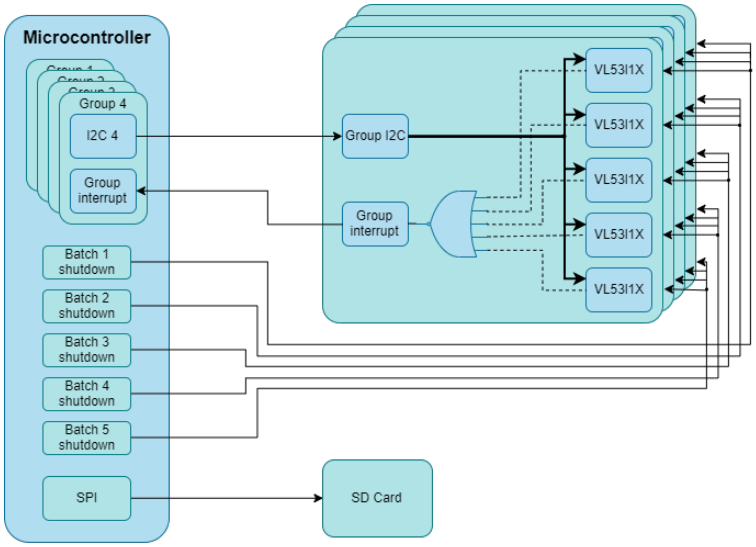

Figure 4.4: Design of data collector

Bibliography

[1] Pixhawk 4. Website. `docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html`. Accessed: 2019-11-30.

[2] PX4 Drone Autopilot. Firmware repository. `www.github.com/PX4/Firmware`, . Accessed: 2019-11-27.

[3] PX4 Drone Autopilot. Website. `www.px4.io`, . Accessed: 2019-11-28.

[4] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[5] PX4 Simulation Documentation. Website. `dev.px4.io/v1.9.0/en/simulation`. Accessed: 2019-11-28.

[6] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[7] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[8] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942. IEEE, 2015.

[9] Sebastian Hening, Corey A Ippolito, Kalmanje S Krishnakumar, Vahram Stepanyan, and Mircea Teodorescu. 3d lidar slam integration with gps/ins for uavs in urban gps-degraded environments. In *AIAA Information Systems-AIAA Infotech@ Aerospace*, page 0448. 2017.

[10] Bitcraze Crazyflie multiranger deck. Website. `www.bitcraze.io/multi-ranger-deck`. Accessed: 2019-11-30.

[11] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5): 1147–1163, 2015.

[12] NASA Official. Aircraft rotations - body axes. `www.grc.nasa.gov/www/k-12/airplane/rotations.html`. Accessed: 2019-12-02.

[13] QGroundcontrol. User manual. `https://docs.qgroundcontrol.com/en/`. Accessed: 2019-12-04.

[14] Gazebo Simulator. Website. `www.gazebosim.org`. Accessed: 2019-11-28.

[15] Cartographer SLAM. Documentation. https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html. Accessed: 2019-12-05.

[16] Robot Operating System. Website. www.ros.org. Accessed: 2019-11-27.

[17] Terabee TeraRanger. Terabee teraranger tower evo. www.terabee.com/shop/lidar-tof-multi-directional-arrays/teraranger-tower-evo. Accessed: 2019-11-18.

[18] VL53L1X. Application note. www.st.com/content/ccc/resource/technical/document/application_note/group0/53/ea/8b/72/2d/07/4f/21/DM00516219/files/DM00516219.pdf/jcr:content/translations/en.DM00516219.pdf, . Accessed: 2019-11-30.

[19] VL53L1X. Datasheet. www.st.com/resource/en/datasheet/vl53l1x.pdf, . Accessed: 2019-11-30.