



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 9, 2019

Contents

1	Introduction	2
2	Literature review	4
2.1	Basic aviation terminologies	4
2.1.1	UAV or Drone	4
2.1.2	Body axes used in aviation	4
2.1.3	Ground Control Station	5
2.1.4	Multicopters	5
2.2	Simultaneous Localization And Mapping - SLAM	5
2.2.1	Visual SLAM	7
2.2.1.1	Monocular camera	8
2.2.1.2	Stereo camera	8
2.2.1.3	LSD-SLAM	9
2.2.1.4	ORB-SLAM	10
2.2.2	Ranging sensor based SLAM	11
2.2.2.1	3D LIDAR SLAM to improve localization accuracy	11
2.2.2.2	3D LIDAR SLAM for mapping	12
2.2.3	Cartographer SLAM	13
2.3	Similar products available on the market	15
2.3.1	Terabee TeraRanger Tower	15
2.3.2	Crazyflie Multi-ranger deck	16
3	Simulation	18
3.1	Robot Operating System	18
3.2	Gazebo Simulator	19
3.3	PX4 Autopilot Software	20
3.3.1	PX4 simulation using MAVROS	20
3.3.2	PX4 simulation using Gazebo with ROS wrapper	20
3.4	QGroundControl	21

4 System design	23
4.1 Pixhawk 4	23
4.2 VL53L1X LIDAR sensor	24
4.3 LIDAR layout design	24
4.3.1 Layout design for 2D SLAM	25
4.3.2 Layout design for 3D SLAM	25
4.4 LIDAR data collection	26
4.4.1 Standalone data collector design	26
Bibliography	28

Chapter 1

Introduction

A prerequisite for future expansion of mobile robots is the ability to navigate autonomously and safely in their ever-changing environment without collision. The positioning of drones in ad hoc indoor environment where no infrastructure is available is needed to fulfill this prerequisite. To know the position of a robot in an unknown environment, the robot needs to build a map of its surroundings so its position can be referenced. Mapping and positioning were considered as two separate problems at first and research focused on either of them, but the solution to these problems separately proved to be divergent. Only by considering these two problems together and finding solutions simultaneously, it became convergent [8]. This process is referred to as Simultaneous Localization And Mapping (SLAM).

The proposed issue by SLAM has many solutions depending on the use-case. SLAM technique can be used to with a focus on improving localization accuracy in GPS degraded environments, by fusing the outputs of a SLAM algorithm with GPS and IMU data [11]. Or as presented in [6], the solution focuses on building highly accurate maps of building interiors or even outdoors online, but as a byproduct position becomes more accurate as well. The above mentioned papers present a mapping approach that uses industrial-grade hexacopter and octocopter and the same precise LIDAR scanner that can measure up to 100m.

Localization and mapping of the environment for a small quadcopters can be challenging, because adding any extra equipment means extra weight that shortens flight time and changes flight behavior. Adding the above mentioned 3D LIDAR scanner that weighs 830g to a quadcopter that is under 1kg means significant extra weight and the drone might turn out to be incapable of flying. These scanners use LIDAR sensors spinning around the horizontal axis to provide ranging measurements on 360°. The stepper motor used for spinning the sensors that could also affect the flight behavior, because a spinning weight acts like a gyroscope. Monocular, stereo or RGB-D cameras can be placed even on small aerial vehicles and therefore offer a great solution. These visual sensors however produce higher amount of data by magnitudes, but still needs to be considered for such purposes, because highly detailed maps can be created by using them.

By using stationary 1D LIDAR sensors placed around the quadcopter, these unwanted effects can be eliminated and the weight of extra equipment can be kept significantly lower, than using a 3D LIDAR scanner discussed before. The sensors need to be placed in a layout that offers an optimal scan of the environment, therefore the movements of quadcopter during flight needs to be taken into account to be able to scan the environment around the vehicle.

In this thesis the design of a 3D positioning and mapping system is being discussed, that uses stationary 1D LIDAR sensors to acquire distance measurements from an Unmanned Aerial Vehicle's (UAV) surrounding and uses SLAM technique to map its surrounding, estimate its position and velocity. The goal is to provide a lightweight hardware and software alternative to accurate planar LIDAR scanners, that are too heavy both computationally and physically for a UAV under 1kg. The focus during the design is on the quality of the map and accuracy of the position estimates.

In chapter 2 basic aviation terminologies are introduced that will be used in the thesis, followed by introduction of SLAM techniques and similar products on the market. Visual SLAM techniques describe the use of monocular cameras, stereo cameras or RGB-D cameras to be used for data acquisition, while ORB-SLAM and LSD-SLAM are two popular SLAM techniques for processing data coming from these sensors. Ranging sensors are non-visual sensors that can use sound or light to measure distance, like ultrasonic sensors or LIDAR sensors. Two interesting researches are discussed, that use LIDAR scanners to improve position or map quality online. Cartographer SLAM is a software package that will be used for the first prototypes and to evaluate layout design of the stationary sensors, because it has compatibility with ROS system.

In chapter 3 software components and their interaction is discussed, that will be used for future simulations. Simulations help to evaluate sensors, layout designs and SLAM algorithms without the need to build the system physically.

The last chapter 4 gives an overview of the chosen LIDAR sensor for the first prototype, first designs of the layout of stationary sensors and a unit for collection of measurements from the sensors.

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 called 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.

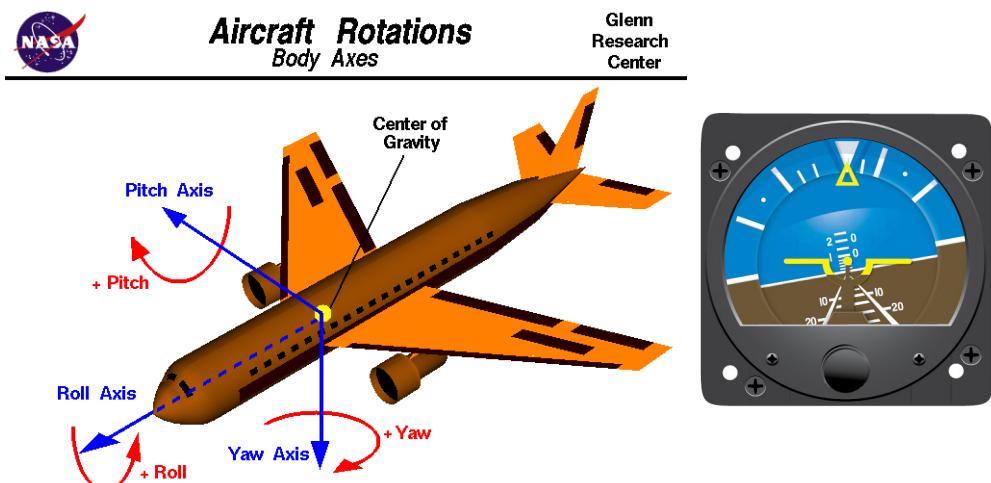


Figure 2.1: Aircraft body axes[14] and attitude instrument

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, hexacopters etc.

Quadcopters are quickly gaining popularity thanks to their easy usability, and because of commercially available drones that can be used for filming. Octocopters have eight rotors in total and mostly used by professionals. 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 is 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 [8] and [4] focusing on finding a solution to the above proposed problem in general, with no dependency on the type of sensor to be used.

During SLAM 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

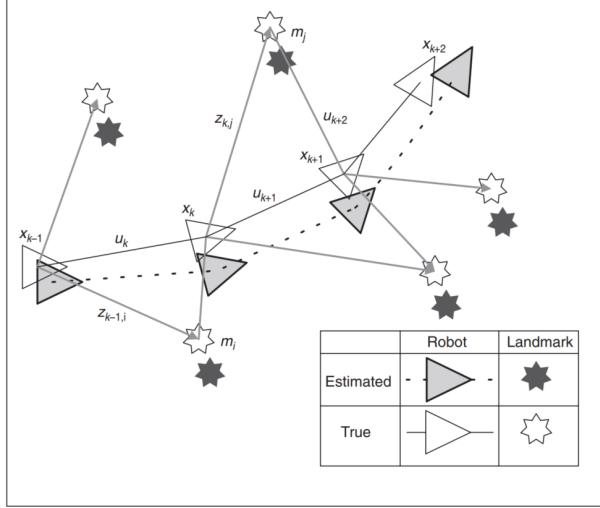


Figure 2.2: The essential SLAM problem presented in [8]

- \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 i th landmark at time k
- $\mathbf{X}_{0:k} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$ A set of all robot location vectors until time k
- $\mathbf{U}_{0:k} = \{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k\}$ A set of history of all control vectors until time k
- $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_n\}$ A set of all landmarks
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$: 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.

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \quad (2.1)$$

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.

$$P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) \quad (2.2)$$

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

$$P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m}) \quad (2.3)$$

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 .

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

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.

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

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

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

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

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

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

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

These problems introduced in 2.7 and 2.8 are dependent on each other and need to be solved 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 camera based and ranging sensor based. Visual observers mostly use simple or special camera configurations for depth sensing, while ranging sensor based observers use a physical phenomena to measure distances like sonars or LIDARS.

2.2.1 Visual SLAM

Visual SLAM is a subset of SLAM algorithms that processes camera images to determine the camera's position and orientation relative to its surroundings, while building a map of its environment. In the simplest case a monocular camera is used to collect images,

but more advanced active or passive stereo cameras and RGB-D cameras are also used. A pro of using images for SLAM is that images contain high amounts of data, therefore very detailed maps can be made. On the other hand, processing big amounts of data means higher complexity and demands higher processing capabilities.

Two main categories of visual 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 use these features to extract depth information. Direct SLAM algorithms however don't search for features, but use the regions of high intensities on the image to estimate the location and surroundings. Two popular visual SLAM methods are feature-based ORB-SLAM and direct LSD-SLAM.

2.2.1.1 Monocular camera

The pro of using monocular camera for localization and mapping is that each image contains high density of information and cameras will always be cheaper easier to use than other depth sensing camera rigs. A big pro, but in the same time a big con of using a monocular camera for SLAM is that a single camera cannot detect the scale. It is an advantage, because the same camera can be used indoor on a small scale even down to centimeters, and used outdoors on a scale of kilometers. Nonetheless during transmission between these scales, the scale can drift and therefore size of landmarks will not be presented correctly. This is called scale-drift, that algorithms try to compensate.

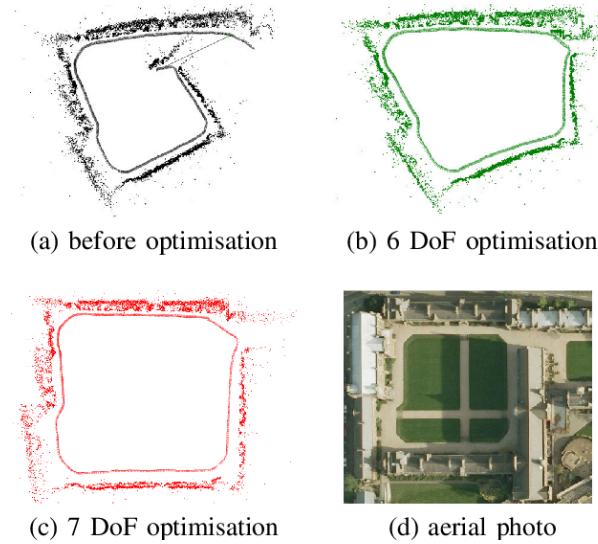


Figure 2.3: Scale drift and compensation illustration from [18]

2.2.1.2 Stereo camera

A stereo camera is a type of camera with typically two image sensors. Two image sensors mimic the human binocular vision and makes depth perception possible. The two cameras are placed relatively close together so there is a high correlation between the images taken by these cameras. To find correlations between the two images sufficient details and structure needs to be present on the images, therefore light conditions affect the performance of depth-sensing. This technique is also called passive stereo vision.

Active stereo cameras solve this problem employing a projector onboard, that projects a structured light to simplify the stereo matching problem. Using this technique the depth sensing performance is less affected by lighting conditions and reoccurring patterns will not confuse the matching of the images.



Figure 2.4: Passive stereo camera on the left, concept of active stereo camera on the right

2.2.1.3 LSD-SLAM

LSD-SLAM for monocular cameras were proposed in [9] and for stereo or RGB-D cameras in [10]. LSD stands for Large-Scale Direct SLAM and it can be seen by the name that it is a direct SLAM algorithm. The map of the surroundings is created based on keyframes that consists of three fields, a camera image, an inverse depth map and the variance of the inverse depth map. The solution of the problem proposed by SLAM is divided into three parts: tracking, depth map estimation and map optimization as seen on figure 2.5.

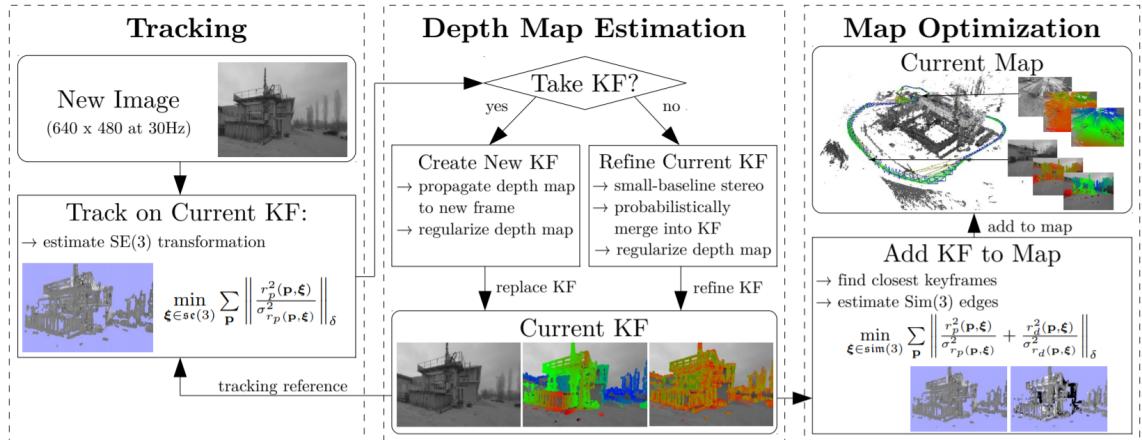


Figure 2.5: LSD-SLAM overview of the complete algorithm [9]

New images are input during the tracking phase and the position is estimated with respect to the current keyframe from previous iteration. The depth map estimation component uses tracked frames to either refine or replace the current keyframe. If the camera has moved more than a set threshold, a new keyframe is initialized with the combination of current and new keyframe. If the tracked keyframe is not taken, then it is used to refine the

current keyframe by performing small-baseline stereo comparisons and regularization of the depth map. Once the current keyframe is updated, it is built into the global map, by the map optimization. Keyframes that are close to each other can be detected and closed, this process is called loop-closure. For loop closures and scale-drift compensation, similarity transform is used. The map is further optimized using pose graph optimization from g2o package.

2.2.1.4 ORB-SLAM

ORB-SLAM is a feature-based SLAM algorithm introduced in [13]. The features extracted are represented by ORBs, therefore the name of the algorithm. ORB-SLAM has three threads that run in parallel: tracking, local mapping and loop closing.

The tracking phase is responsible for localizing the camera and to decide when to insert a new keyframe. The features extracted from the new image are FAST corners, where FAST is a computationally efficient process to find corners on images. These corners are described using ORB, that is a method to describe FAST corners by a center of mass and an orientation parameter. This way the place and orientation is known of each feature. The initial pose is estimated using a constant velocity model and if the tracking is lost, relocation is done instead of pose estimation. Track local map is a map of keyframes that share a common map point. Finally it is decided if new keyframe is need to be created.

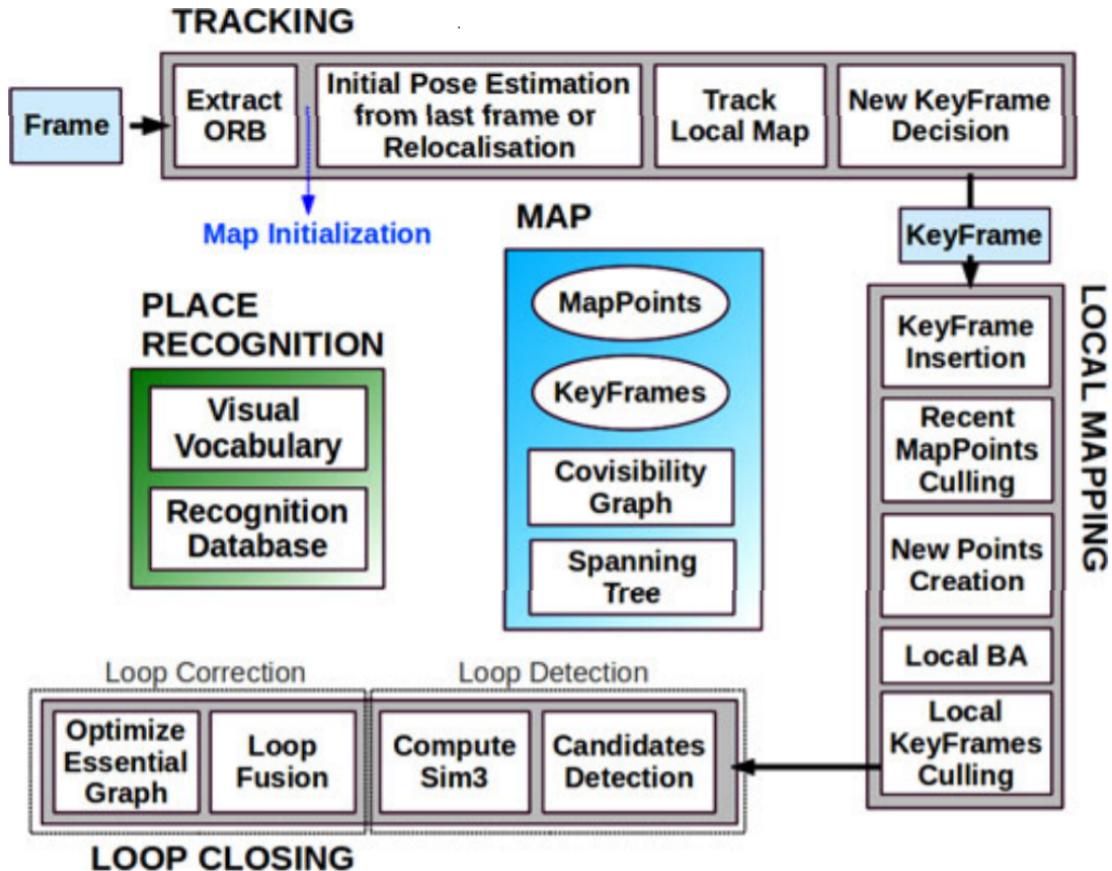


Figure 2.6: ORB-SLAM overview [13]

In the first step of local mapping the new keyframe is inserted into the co-visibility graph and it is linked keyframe to keyframe with the most points in common. ORB features from

the connected keyframes are used to calculate new map points using triangulation. Unmatched ORBs are compared with other unmatched ORBs of other keyframes. Projection error and consistency is checked, then the map is projected to the matched keyframes. The new map points need to be validated, by going through a test to increase the likelihood that these map points are valid part of the map. In local bundle adjustment step, the current keyframe and connected keyframes through the co-visibility graph are optimized. Finally abundant keyframes get discarded to keep simplicity.

Loop closing is quite effective in ORB-SLAM, because matching ORBs can be done with relatively high confidence. Similarity transformation is calculated.

2.2.2 Ranging sensor based SLAM

Cameras are cheap, popular and can be used to create highly detailed maps, but they produce high amounts of data that demands high processing capabilities. Ranging sensor based approaches produce lower amount of data that is dependent on the scale, these sensors have a minimum and maximum range they can measure in between.

Ultrasonic sensors have been used for distance measurements for many years, but they suffer from significant measurement noise, that comes from background noise and the variability of speed of sound. Almost no development has been seen on ultrasonic sensors in the previous years LIDAR sensors on the other hand are actively developed, because this technique uses light to measure distance, offer more accurate measurements, typically 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.

A planar laser scanner is a LIDAR sensor that is extended to make distance measurements in 2D. Typically it is used for scanning the horizontal plane. Usually the scanning is solved by rotating a LIDAR sensor in 360° while it is making measurements evenly distributed on the plane. Planar scanners are good for building floor maps and navigating in indoor environments. Some scanners are further extended for 3D scanning.

2.2.2.1 3D LIDAR SLAM to improve localization accuracy

The study of University of California[11] presents an algorithm that fuses LIDAR, IMU and GPS signals to have a more accurate estimation of the absolute position and velocity of a UAV. The LIDAR scanner provides local position updates using SLAM technique,

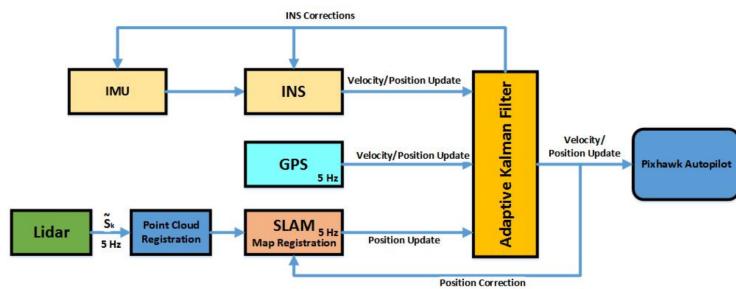


Figure 2.7: 3D LiDAR SLAM Integration with GPS/INS for UAVs in Urban GPS-Degraded Environments [11] - algorithm overview

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 2.7.

The 3D LIDAR scanner used is Velodyne VLP-16, that has a vertical field of view of 30° with a vertical resolution of 2° . On the horizontal axis the resolution is $0.1\text{-}0.4^\circ$ and rotation rate can be adjusted between 5-20Hz. The sensor weighs 830g according to the company website.

During the experiment an octocopter is used to carry the sensor. Octocopters are stable and can carry high amounts of weight. The proposed filter is 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 result in a more accurate position estimate in GPS-degraded environments. The capabilities of the LIDAR was also tested on an even moon-like surface, with very few landmarks that can be used by the SLAM algorithm. In this case GPS data is much more accurate than LIDAR SLAM, therefore LIDAR data is not used for calculation of position updates in these cases.

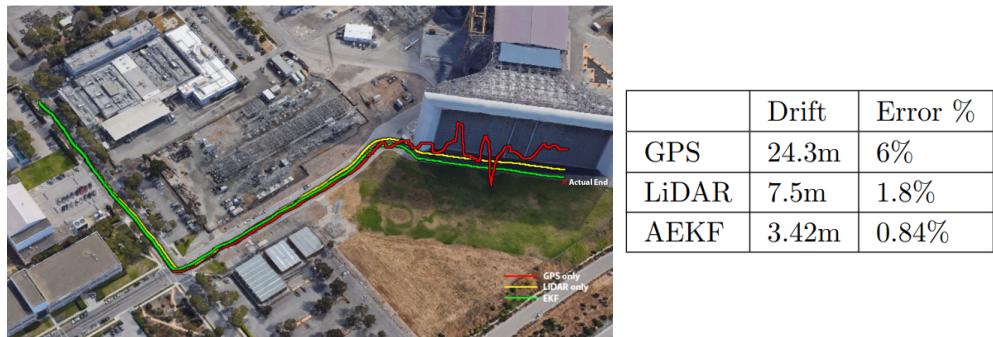


Figure 2.8: GPS, LIDAR and EKF position estimates and position drift after 405 meters[11]

2.2.2.2 3D LIDAR SLAM for mapping

The solution seen in 2.2.2.1 uses a LIDAR scanner to improve position and velocity data, by fusing the output of SLAM algorithm with sensors. The map created simultaneously is not used in that solution. 3D LIDAR data and SLAM can be also used to create highly detailed maps and this paper presented by David Droeßel and Sven Behnke [6], discusses an efficient implementation of SLAM with focus on improving online mapping quality.

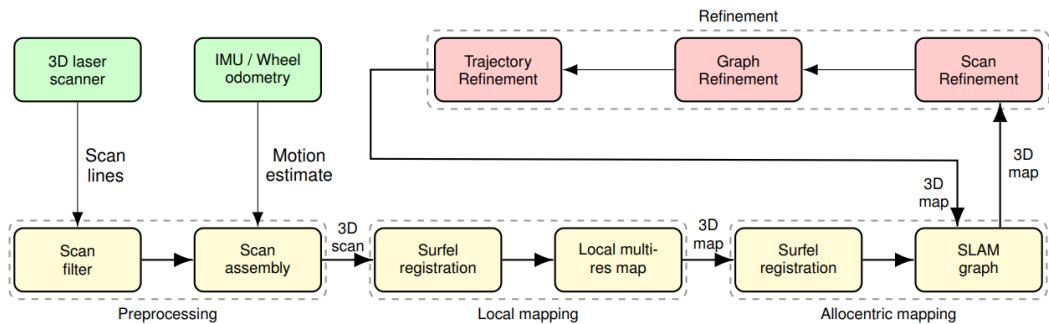


Figure 2.9: Efficient Continuous-time SLAM for 3D LIDAR-based online mapping[6] - algorithm overview

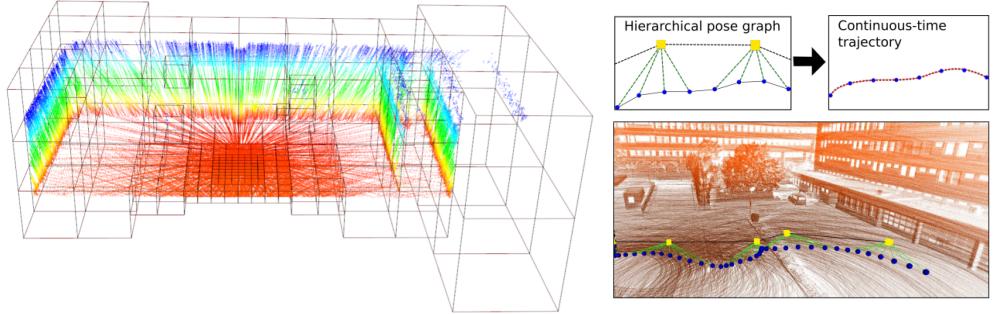


Figure 2.10: Grid-map presented in [7] on the left, estimated trajectory and built map on the right

The system first aggregates range measurements into a robot-centric grid-map [7], that has high resolution in the close proximity and lower resolution with increasing distance. A grid cell stores the cell's occupancy probability and a surface element(surfel), that describes the mean of samples and covariance. IMU or odometry measurements are used to account for motion of the sensor during acquisition, to compensate rolling shutter artifacts. After the preprocessing step, surfels become registered and the local map is updated. The older local map measurements are overwritten with the current 3D scan. During allocentric mapping phase, the 3D local multiresolution maps from different view poses are aligned with each other using surfel based registration. Local maps that are registered to the SLAM graph are subject to refinement. During these steps, the scans are refined based on available local maps, then graph and sensor trajectories are also refined. Local mapping, allocentric mapping and refinement phases are independent from each other, so these can run parallel.

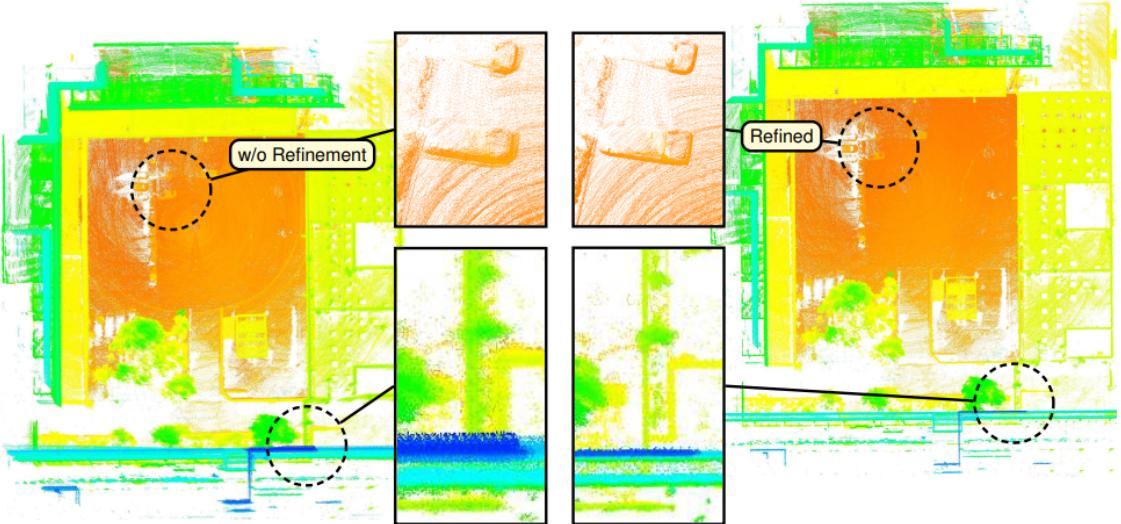


Figure 2.11: Comparison of maps with and without refinement[6]

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 mapping of building interiors, using a backpack mounted LIDAR scanner.

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

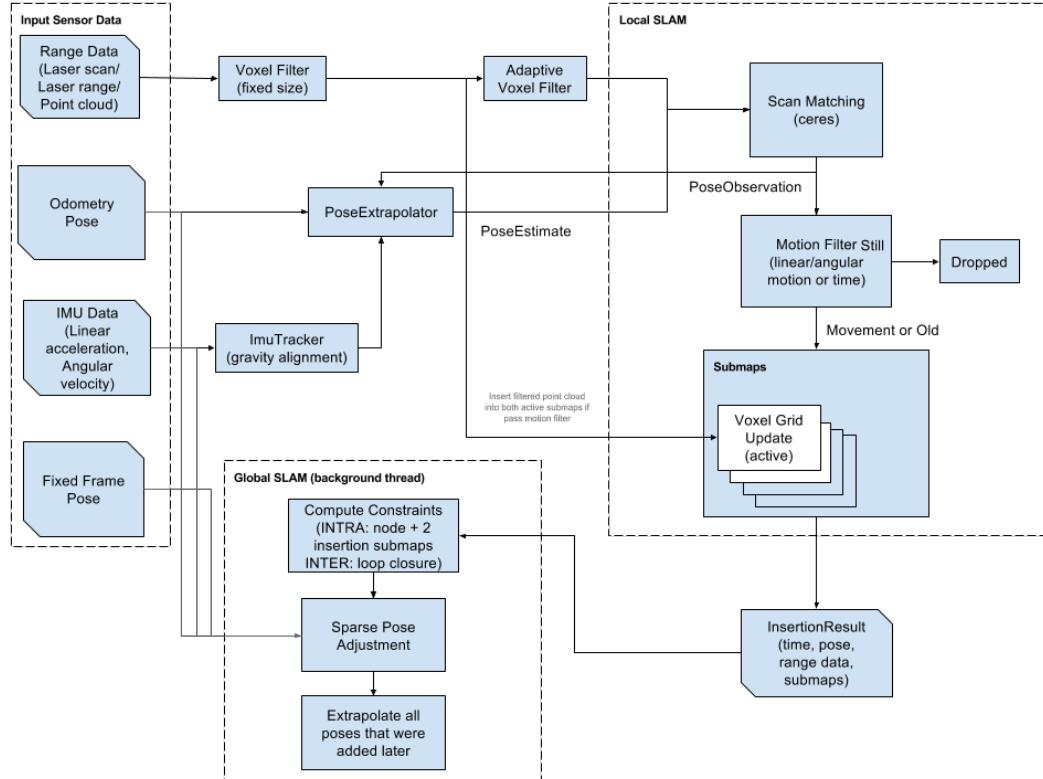


Figure 2.12: Cartographer SLAM overview[17]

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 2.12. 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 by updating the whole map increases complexity quadratically, but by using local maps or submaps the complexity can be limited to the contents of the submap that is preferable during online processing. In this case alignment of submaps needs to be solved, but it requires less processing and it can be run in parallel.

The most important input of Cartographer is the Range Data, range measurements 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, and some sensors set unsuccessful range measurements to a value that is significantly higher than the maximum range of the sensor. These outliers need to be filtered and Cartographer uses a bandpass filter 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 down-samples raw points where density is higher, but leaves scarce measurements to reduce computation needed, while keeping data integrity.

Inertial Measurement Units (IMU) provide a direction of gravity vector and a noisy estimation of the robot's movement. 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.

In Local SLAM phase, first scans are inserted into a submap at the best estimated position and scan matching is done only on submap and recent scan. During this process the error of pose estimates accumulate in the world frame. To reduce the accumulated error, pose optimization is run regularly. All finished submaps are considered for loop closure, that happens on a background thread Global SLAM.

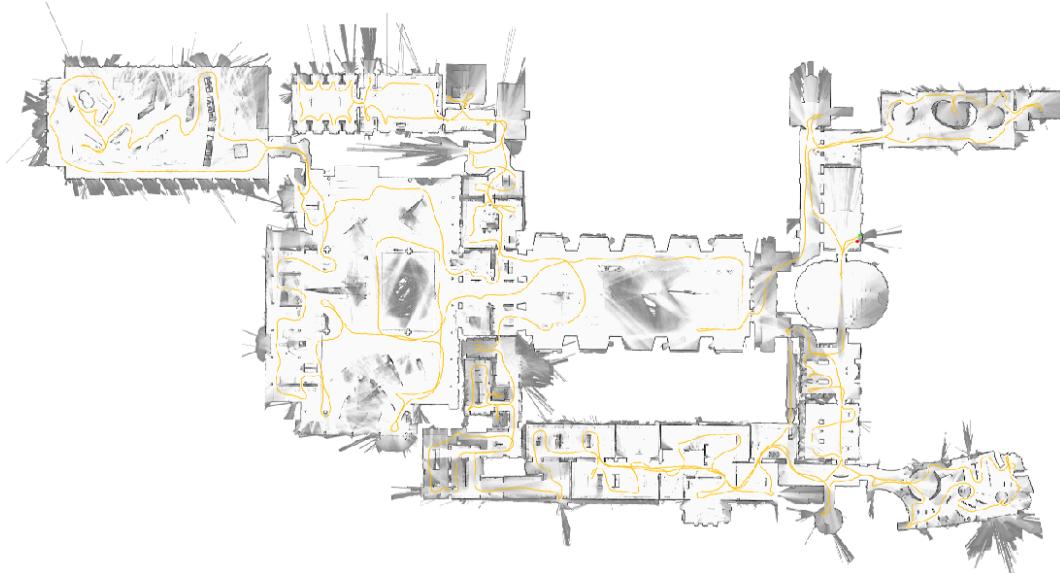


Figure 2.13: Cartographer SLAM example[17]

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 components.

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 and 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[20].

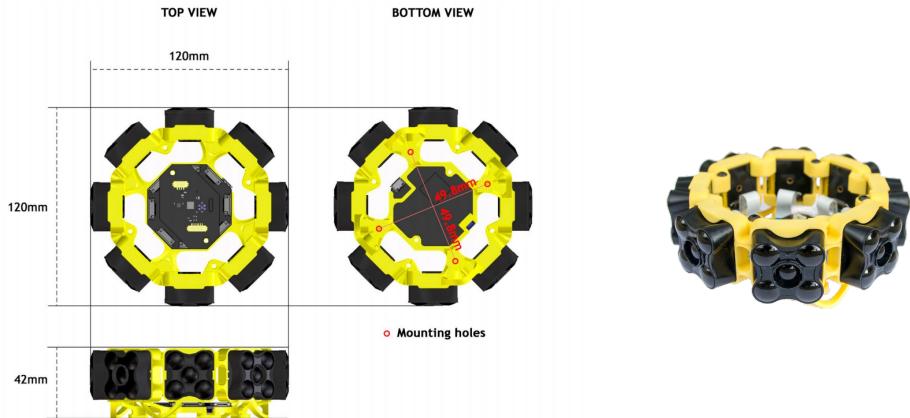


Figure 2.14: Terabee TeraRanger Tower Evo dimensions

	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	±4cm in the first 14m, 1.5% above	±12cm

Table 2.1: TeraRanger Tower Evo specifications

Terabee provides a tutorial video on their website[20] 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 [12], where a SLAM algorithm is used to create a map and localize the drone. This serves as a proof of concept, that stationary 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 2.15: 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[19]. 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.

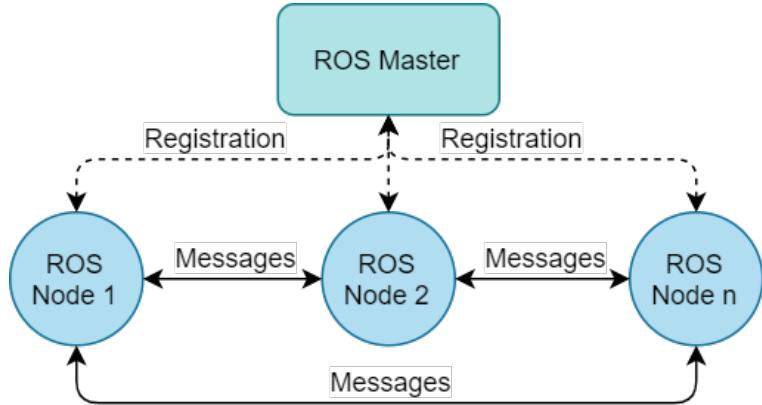


Figure 3.1: ROS publish subscribe communication

3.2 Gazebo Simulator

Gazebo[16] 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.

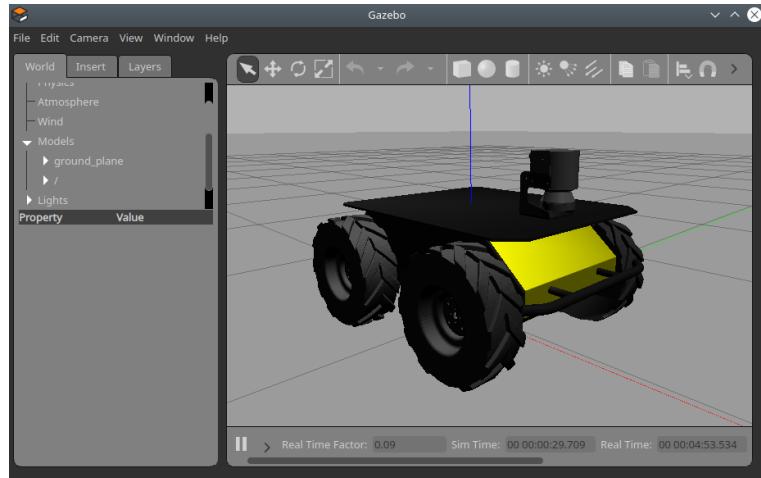


Figure 3.2: Gazebo ROS Husky robot demo

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 all components of the 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.

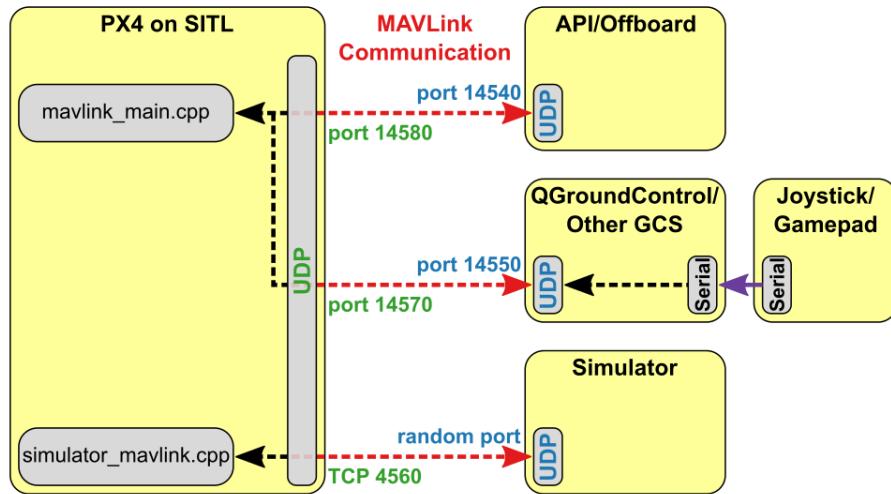


Figure 3.3: PX4 SITL Simulation Outline [5]

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 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 and 3.5. MAVROS is still needed to be run, because there are some messages that would not be published to ROS otherwise.

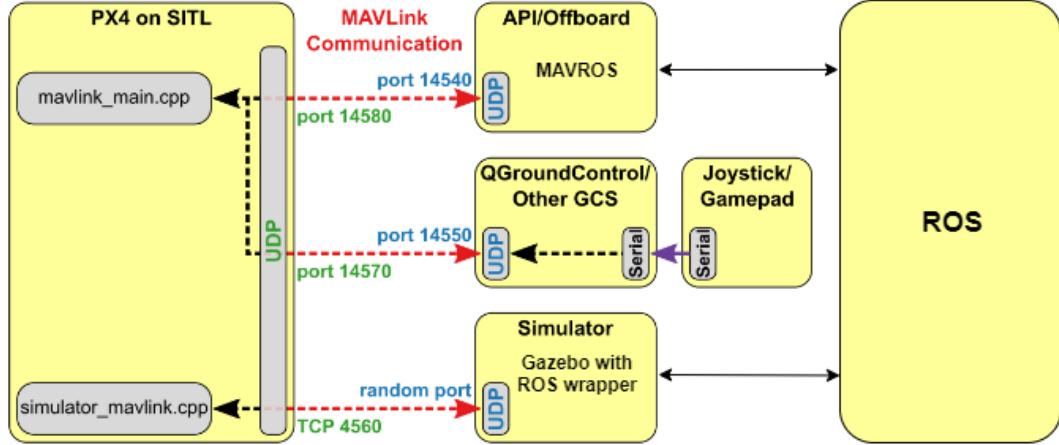


Figure 3.4: PX4 Simulation logical unit overview using Gazebo with ROS wrapper

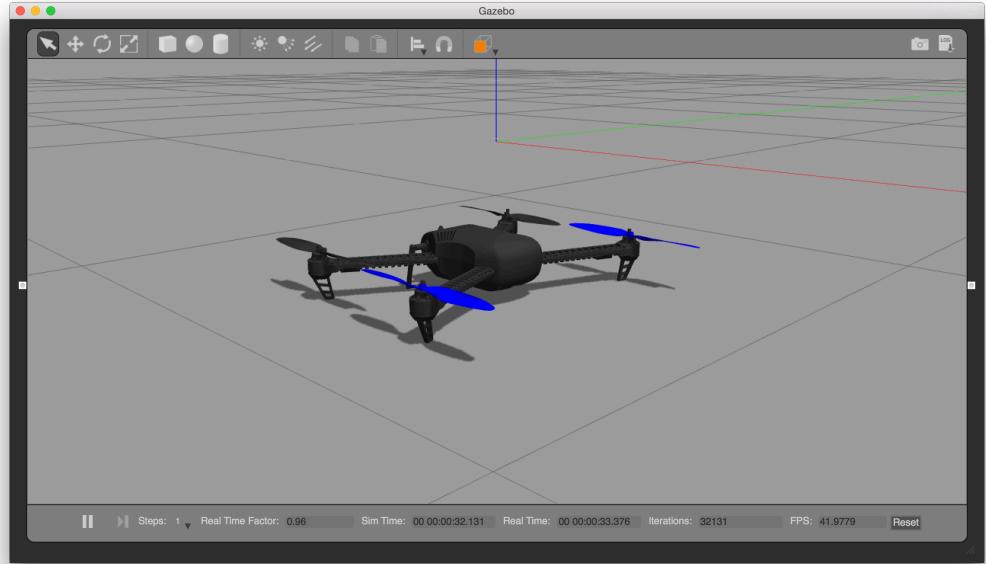


Figure 3.5: Iris drone in gazebo

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 3.6: QGroundControl main screen[15]

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 the simulator.

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 [22]. 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.

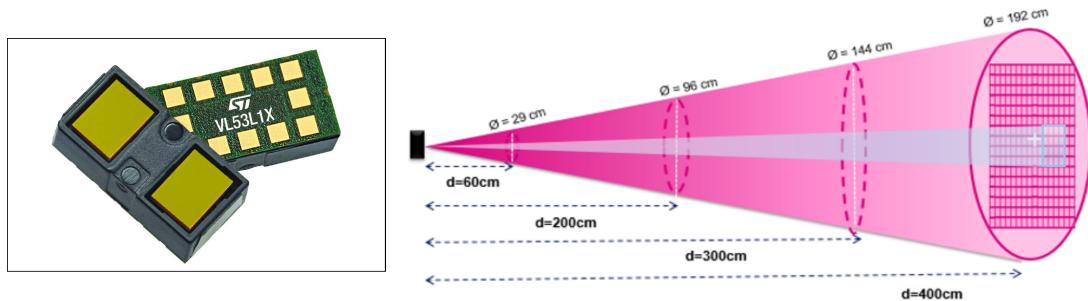


Figure 4.2: VL53L1X sensor [22] and region-of-interest [21]

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 uncovered zone.

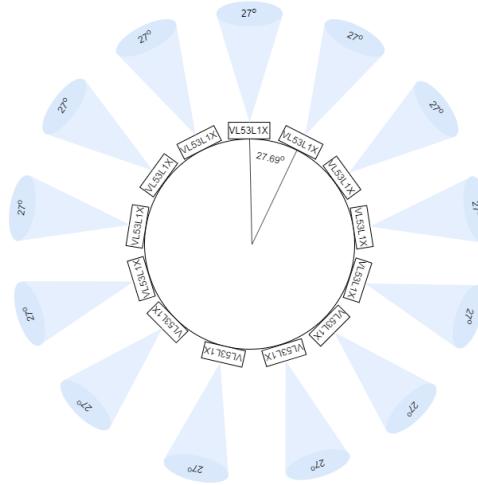


Figure 4.3: Layout of 13 LIDAR sensors

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 yaw 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[12] 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° .

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, using 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.

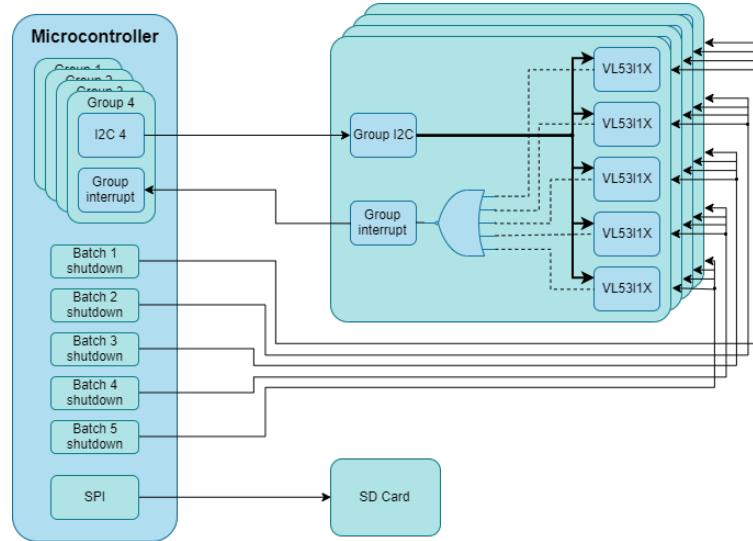


Figure 4.4: Design of data collector

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 connected to the microcontroller, while no functionality is lost. This solution also makes software development easier and the built system is clearer.

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] David Droseschel and Sven Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.
- [7] David Droseschel, Jörg Stücker, and Sven Behnke. Local multi-resolution representation for 6d motion estimation and mapping with a continuously rotating 3d laser scanner. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5221–5226. IEEE, 2014.
- [8] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.
- [9] 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.
- [10] Jakob Engel, Jörg Stücker, 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.
- [11] 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.
- [12] Bitcraze Crazyflie multiranger deck. Website. www.bitcraze.io/multi-ranger-deck. Accessed: 2019-11-30.
- [13] 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.

- [14] NASA Official. Aircraft rotations - body axes. www.grc.nasa.gov/www/k-12/airplane/rotations.html. Accessed: 2019-12-02.
- [15] QGroundcontrol. User manual. <https://docs.qgroundcontrol.com/en/>. Accessed: 2019-12-04.
- [16] Gazebo Simulator. Website. www.gazebosim.org. Accessed: 2019-11-28.
- [17] Cartographer SLAM. Documentation. https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html. Accessed: 2019-12-05.
- [18] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Scale drift-aware large scale monocular slam. In *Robotics: Science and Systems*, 2010.
- [19] Robot Operating System. Website. www.ros.org. Accessed: 2019-11-27.
- [20] Terabee TeraRanger. Terabee teraranger tower evo. www.terabee.com/shop/lidar-tof-multi-directional-arrays/teraranger-tower-evo. Accessed: 2019-11-18.
- [21] 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.
- [22] VL53L1X. Datasheet. www.st.com/resource/en/datasheet/vl53l1x.pdf, . Accessed: 2019-11-30.