Royal Military Academy
165th Promotion Polytechnics
Lt-Col avi Vaerten

Academic year 2014-2015
2nd Master

Autonomous guidance for a UAS along a staircase using visual servoing

Second lieutenant Commissioned Officer Candidate De Meyst
Second lieutenant Commissioned Officer Candidate Goethals

Master Thesis Department Mathematics – Chair of theoretical mathematics
presented to obtain the title
of civil engineer in polytechnics
under supervision of Major Dr. Ir Robby HAELTERMAN
Dr. Ir Geert DE CUBBER
Brussels, 2015

Abstract

In the last years, unmanned aerial systems (UAS) have become more popular due to the vast amount of manufacturers, offering different systems at sometimes very low prices. Since these systems are becoming more accessible and a lot of guides are available to build do-it-yourself (DIY) drones, the difficulty to make or use such a drone has decreased significantly. At first, UAS were difficult to control. Especially quadcopters were not very stable. Nowadays, open source controllers are available to get your personal quadcopter up and flying in no time. Therefore, researchers started facing the next big challenge in the world of unmanned aerial systems: autonomous control. The scope of this work is in line with this research trend.

The objective of this thesis is to implement a navigation system for a quadcopter which has two goals. The first goal is the detection of a staircase using only the images provided by a monocular camera on-board of a quadcopter. The second goal consists of guiding the drone to the top of the detected staircase.

As for the first goal, multiple algorithms are considered to increase the probability of detection and decrease the amount of false positives. Detection is based on two different sources of information. The first source is the 2D image that is taken at a certain time. Line Segment Detection is used for the detection of quasi-horizontal lines with respect to the world. A second algorithm that is used on the 2D images of the staircase is a cascade classifier. This type of classifier finds its origin in face detection and has already been used in many real-time detection set-ups. This classifier was trained on a large data-set of images and has proven to be useful in previous staircase detection publications. The second source of information is a depth image that is generated from a point-cloud. The point-cloud of the surroundings of the quadcopter is generated using a monocular simultanious localisation and mapping (SLAM) algorithm called Large-Scale Direct Monocular SLAM (LSD-SLAM). An existing algorithm for curvature detection is used to determine if curvature is present at the location of the detected lines in the 2D image. If a staircase is present, curvature will be present at the location of the detected lines. Guided sampling is used to project the lines to a 3D space. The curvature of these projected lines is then analysed. Additionally an angle detection algorithm processes the depth images and selects regions that have an angle similar to that of a staircase.

All the detection algorithms were carefully evaluated and fine-tuned on a single dataset and the combination of the methods was then tested on different staircases.

Besides detection, a proper control system is necessary for autonomous control of the quadcopter. This is the second goal of this thesis. An Extended Kalman Filter (EKF) is used to fuse all available data and make an estimation of the current and future position of the quadcopter. This data consists of the internal navigation data of the drone, the data from the Inertial Measurement Unit (IMU), echo sounder, commands that are send to the drone and a visual pose estimation of Parallel Tracking and Mapping (PTAM) which is another SLAM algorithm.

The controller makes use of four proportional-integral-derivative (PID) controllers to control the four controllable degrees of freedom of the quadcopter: pitch, roll, yaw and vertical velocity. The other two degrees of freedom cannot be controlled because a quadcopter is underactuated.

Foreword

A new trend in the armed forces around the world is the huge amount of interest and investment in unmanned systems. It is without doubt the future for modern warfare. The land component, the air component as well as the naval component can explore more in less time with no risk for their crews. Although we know already a lot about these unmanned systems, lots of research still has to be done especially in the autonomous functioning of these systems. By developing software to make a Unmanned aerial systems (UAS)/Remotely piloted aircraft system (RPAS) autonomous in a GPS denied environment, we are helping the development of future systems that can be used by our ground forces to scout inside buildings. The research that will be done on image recognition and visual navigation, could however be generalised and used for other unmanned systems.

The research question is quite straight forward: develop software that handles autonomous detection of a staircase and enables the drone to guide itself along the stairs. The UAS that has to be used is the Parrot AR.Drone 2.0 and the framework that handles the steering and navigation will be based on visual servoing and visual odometry. The navigational data from the on board sensors of the Parrot AR.Drone 2.0 will be used as input for the algorithms. Furthermore the developed algorithms should be as efficient as possible. The purpose of this thesis thus contains two goals. First of all there is the problem of recognizing the staircase and secondly a framework has to be made to guide the drone to the detected point.

Writing a thesis as a polytechnic student is not something you can do without help from others. We, the authors of this thesis, would like to take this opportunity to thank some special people for their good help and support.

First of all, a special thanks to the promoter Major Dr. Ir Robby Haelterman and second reader Dr. Ir Geert De Cubber for their guidance and counselling. They learned us to think in a structured way and to be critical of our own work. Thank you for the patience that you exercised and the collaboration opportunities that you presented us with. Especially the invitations for the Icarus project lectures were very inspirational.

Furthermore, we would like to express our thanks to the numerous amount of researchers that made their work available for the community. In particularly we would like to thank Engel, Schöps, and Cremers for releasing their work in the public domain. We also would like to thank Lee, Leung, and Medioni for making their results publicly available upon our request.

Then, we also wish to thank all the people who read this thesis very carefully. Their critical evaluation and comments have certainly given this thesis an additional value.

We can not forget to thank all our friends and colleagues who gave us a wonderful time as a student during the last five years in the Royal Military Academy.

Finally, we would like to say thank you to our family. They gave us the opportunity to study and complete our master. We would have never come this far without their continuous support and trust.

Contents

List of Figures

Glossary

AI artificial intelligence. 63–65

AP access point. 6, 27, 28

API application program interface. 77

ARM® Acorn RISC Machine. 75

AWGN additive white Gaussian noise. 21

BA bundle adjustment. 31, 37, 38

CNN convolutional neural network. 76

CPU central processing unit. 75

DIY do-it-yourself. i

DSSM dynamic state space model. 17

EKF Extended Kalman Filter. i, 14, 15, 17, 22, 23, 26, 36, 37, 39, 48, 67, 71, 76, 77, 79

FAST Features from accelerated segment test. 46

FPGA field programmable gate array. 74, 75

GPGPU General-Purpose computing on Graphics Processing Units. 74

GPU graphics processing unit. 32, 74, 75, 79

GUI graphical user interface. 34, 62, 63, 65, 67

HMM hidden markov model. 18

HSA Heterogeneous System Architecture. 74, 75

IBVS image-based visual servo. 22

ICMP Internet Control Message Protocol. 26

IMU Inertial Measurement Unit. i, 22, 60

KF keyframe. 32, 33, 53, 70

LF liveframe. 32, 53, 54, 70

LLA level-line angle. 47, 48

LSD Line Segment Detector. 49, 51

LSD-SLAM Large-Scale Direct Monocular SLAM. i, 2, 31–36, 39, 44, 53, 64, 68, 76, 77

MLP multilayer perception. 76

NFA Number of False Alarms. 47

ORB Oriented FAST and Rotated BRIEF. 46

PBVS position-based visual servo. 22

PCA principal component analyses. 51

PCL Point Cloud Library. 8, 57, 70, 75

PDF probability density function. 17, 19, 20

PID proportional-integral-derivative. i, 1, 27, 29

POV point of view. 11, 25

PTAM Parallel Tracking and Mapping. i, viii, 1, 9, 13, 15, 18, 23, 25, 31, 33, 36–39, 61, 66, 67, 76

RANSAC Random sample consensus. 48, 49, 57, 77

ROS Robot Operating System. 4, 8–10, 12, 14, 15, 22, 31, 33, 34, 44, 62–64, 69, 70, 78

SDK software development kit. 9, 75

SE special euclidean group. 25, 53

SfM structure from motion. 22, 31

SIFT scale-invariant feature transform. 46

SIM similarity transformation. 31

SLAM simultanious localisation and mapping. i, 1, 30–33, 36, 37, 39, 43, 46, 76, 79

SMC sequential Monte Carlo. 19

SO special orthogonal group. 30

SSD sum of squared differences. 40

SSE Streaming SIMD Extensions. 75

SVM support vector machine. 51

TBB thread building blocks. 8

ToF time-of-flight. 75

TUM Technische Universität München. 32

UAS unmanned aerial systems. i, ii, 17, 77, 78

VO visual odometry. 22

VRAM Video random-access memory. 74

Introduction

The objective of this thesis is to implement a navigation system for a quadcopter which has two goals. The first goal is the detection of a staircase using only the images provided by a monocular camera on-board of a quadcopter. The second goal consists of guiding the drone to the top of the detected staircase.

An additional condition is the fact that the Parrot AR.Drone 2.0 has to be used for the development. The mandatory usage of such a small consumer drone results on the one hand in a very low cost but on the other hand, this type of quadcopter has low quality sensors such as the camera which affect the performance of the algorithms and, consequently, the outcome of the staircase guidance system that is developed in this thesis.

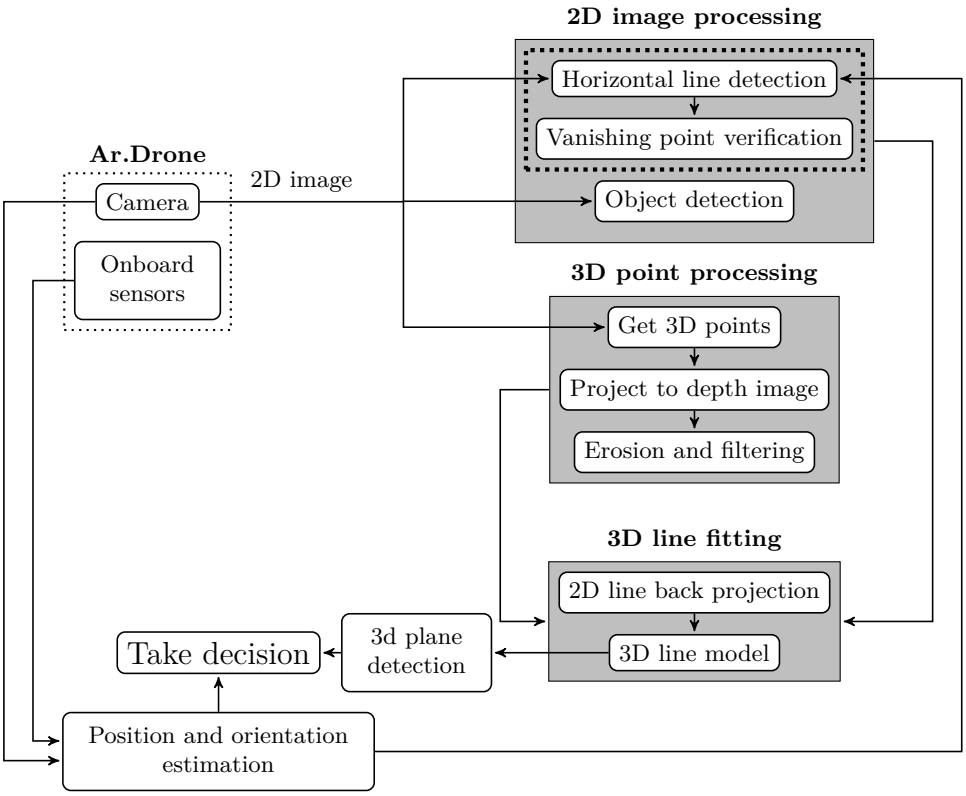
This thesis proposes a unique combination of detection algorithms in order to detect successfully a staircase. Throughout this thesis, all methods will be explained in a logical order. The aim of each chapter is to provide an overview of state of the art techniques, explain the most relevant methods and to point out the strengths and weaknesses of the algorithms. The specific modifications and implementations are also explained throughout the thesis.

In chapter 1 and chapter 2, the used hardware and software respectively will be explained. The hardware describes the characteristics of the quadcopter but also the optimal setup that is used. Various software packets are used to process, fuse, visualize and interpret the data of the sensors. The goal of chapter 2, is to justify the choices of the platforms and to describe their functionality and their role in the software that is developed in this thesis. Vision processing, position estimating and controller implementation are the three principal categories that are mentioned. Special attention is given to the correct calibration process of the camera. Two different procedures were used to calibrate the camera in this work.

The first part of chapter 3 provides the reader a more mathematical foundation of the possible state estimation systems for a quadcopter. The second part is in turn, more focussed on the actual implementation that is used to estimate the quadcopter's position and pose in the context of this thesis. In the third and last part of this chapter, the Proportional-integral-derivative (PID) controller that is specifically designed for the AR.Drone is elaborated. The authors used a tweaked version of the `tum_ardrone` package.

Because Simultanious localisation and mapping (SLAM) plays a very important roll in this thesis, a whole chapter (chapter 4) has been allocated to explain the principles behind it. The representation of the pose is explained in the beginning of this chapter. Two different approaches to SLAM are then described together with well-known algorithms that implement these approaches. The authors chose to use two different SLAM algorithms in this thesis. One algorithm uses visual features to help the state estimation with the localization of the quadcopter and the other algorithm generates a 3D pointcloud of the environment in near real-time. The former is based on Parallel Tracking and Mapping (PTAM) while the latter is

based on Large-Scale Direct Monocular SLAM (LSD-SLAM) and is extensively modified to be compatible with the setup of this thesis.



Chapter 5 is the core chapter of this work and is summarised in the diagram that is included above. The detection is build from the ground up and can be subdivided in three parts: 2D detection, 3D detection and the combining strategy. To detect the staircase, the authors chose to incorporate different approaches to obtain a maximal rejection of false positives. This, in the hypothesis that the different detection algorithms have very few or almost no false negatives. For the 2D detection, a Line Segment Detector (LSD) is implemented together with a self-implemented rejection criterion that is based on the vanishing points of the detected lines and the outcome of an object detector that is based on a cascade classifier. In 3D, the generated pointcloud is projected to the drone's point of view into a depth image and subsequently a 3D line fitting is performed to search the 2D lines that were previously detected. The detected 3D lines, are then fitted to a 3D plane model. The angle of the plane with regard to the horizontal plane is calculated and should be typical for a staircase. Another section of this chapter creates a theoretical model for the inter distances of the projected lines of a staircase in a 2D image. However, this model is not implemented in this thesis.

All the different detection steps were carefully evaluated and fine-tuned on a single dataset and the combination of the methods was then tested on different staircases.

A graphical user interface is created specifically for this thesis to control the quadcopter and visualize what is going on. The software allows the user to manually control the drone but also to initialize the autonomous mode which is supervised. Meaning that the drone indicates what it wants to do and asks the user if it can continue. The guidance software listens to the output of the detection node and makes use of an agglomerative clustering technique to implement

temporal consistency and filter out different detections. The implementation of this software is described in chapter 6.

In chapter 7 and chapter 8 respectively the methodology of this thesis is described and the possible improvements for future works are denoted. This thesis ends with a set of conclusions and recommendations in Conclusion.

1. Hardware platform

A lot of possibilities exist nowadays regarding quadcopters. Some examples are the Arducopter[1], Spiri[2], DJI Phantom[3], Parrot AR.Drone 2.0[4] ...

From all these possibilities, the Parrot AR.Drone 2.0 was used. This because the drone was already in use in the department of mechanics in our university. They have chosen this drone because of the low cost, relatively good image quality, robustness, off-the-shelf availability and the existing software platforms that are built for this particular drone.

In the following sections, the hardware specifications will be presented. The software platforms that are used are described in chapter 2.

1.1. Parrot AR.Drone 2.0

The most relevant specifications of the Ar.Drone 2.0 from Parrot are given in Table 1.1

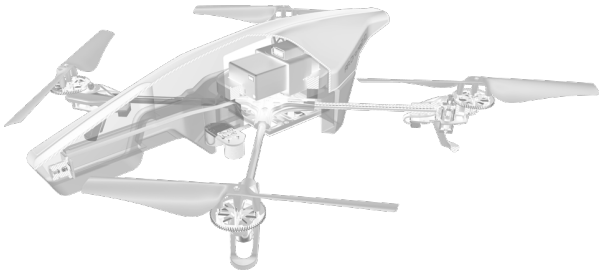


Figure 1.1.: Parrot AR.Drone

1.2. Camera calibration

Both cameras of the Parrot AR.Drone can be calibrated by the camera calibration tool that is integrated in Robot Operating System (ROS) (section 2.2).

[1] http://copter.ardupilot.com/
[2] http://pleiades.ca/
[3] http://www.dji.com/product/phantom
[4] http://ardrone2.parrot.com/

Property	Value
Front camera	
Resolution	720p 30 FPS
Lens	92° diagonal
Codec	H264
Other	
Bottom camera	QVGA 60 FPS
3 axis gyroscope	2000 °/s precision
3 axis accelerometer	±50 mg precision
3 axis magnetometer	6° precision
Pressure sensor	±10 Pa precision
Ultrasound sensors	
Battery	1000/1500 mA h
Motors	Brushless 14.5 W 28.500 RPM
Reduction gears	1/8.75 Nylatron gears propeller reductor
Structure	
Weight	380/420 g (outdoor hull/indoor hull)
Dimenstions	52.5 cm × 51.5 cm (without hull)
Electronics	
Processor	1 GHz 32 bit ARM Cortex A8
Video	800 MHz DSP TMS320DMC64x
RAM	1 GB DDR2 200 MHz
Software	
Linux kernel	2.6.32

Table 1.1.: Parrot AR.Drone 2.0 specifications

1.3. Other hardware

1.3.1. Off-board processing

While the Parrot AR.Drone 2.0 uses an on-board video system, most of the processing is done off-board. Normal laptops (Core i3/i5/i7 2 GHz, 8 GB DDR3 1600 MHz) are used to do the necessary processing. However it is advised to use a powerful system because the load on the system may be quite high especially when the detection software is running (Figure 1.2).

1.3.2. Connectivity with the Parrot AR.Drone 2.0

In order to statically test the drone, an external pc power supply is used to remove the need for replacing the batteries. Another possibility is to use a wired setup[5]. This requires a high current DC (≥ 20 V) power supply on the ground and a voltage stabilizer on-board of the drone. The weight of the cable also influences the dynamics and is therefore not an ideal solution.

[5] `https://www.youtube.com/watch?v=4dVAnzUzWso`

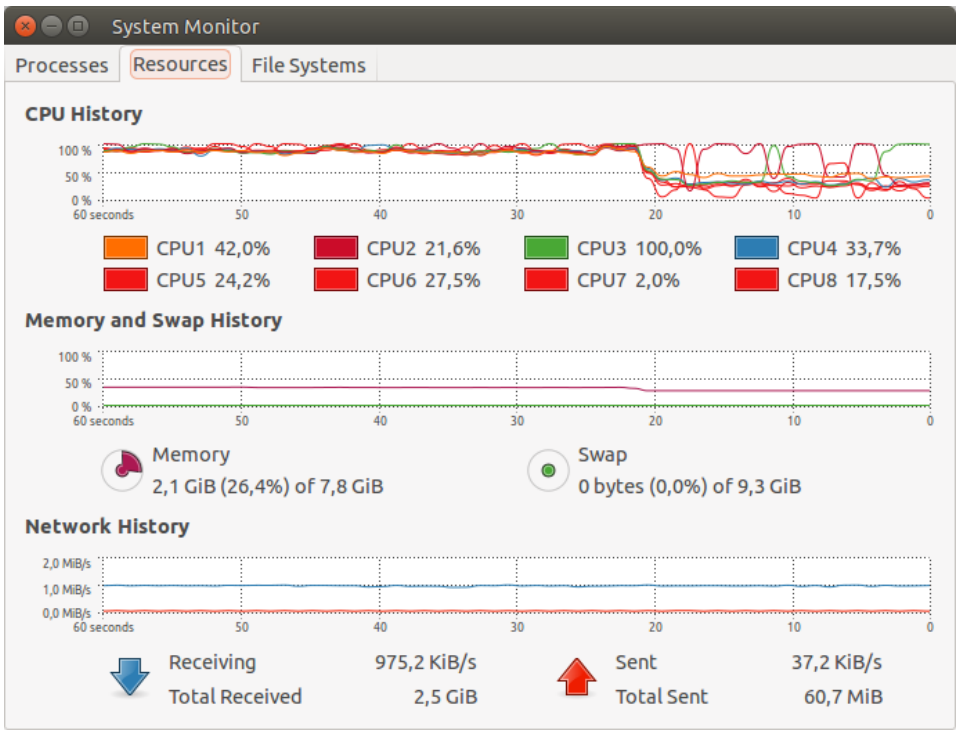


Figure 1.2.: Load of the detection software on a laptop that has a first generation i7 1.6GHz with 8GB of ram. One can clearly see where the detection software is stopped.

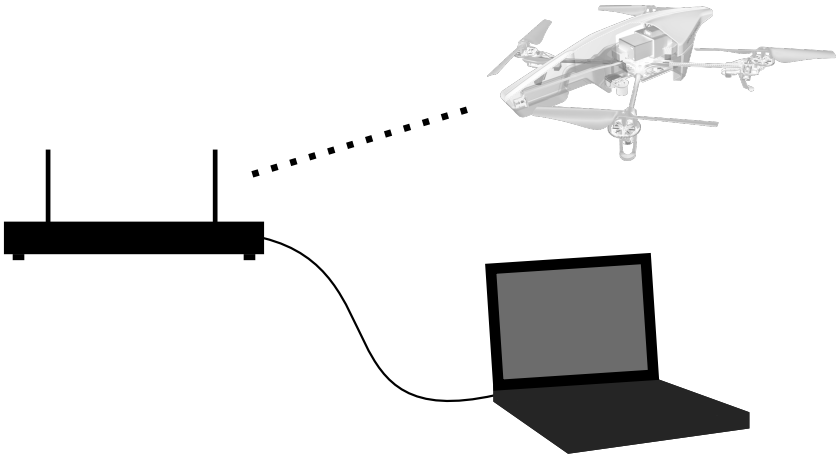


Figure 1.3.: Connectivity setup: The Parrot AR.Drone is connected to a wireless Access point (AP) and the computer that is used, is connected to the AP with an ethernet cable.

Regarding the connectivity of the drone, a wireless Access point is used to enhance the quality of the connection as discussed in subsection 3.3.2 and to extend the range by a couple of meters. The AP's IP address is fixed (eg 192.168.2.1), it's ESSID is set (eg DRONE-WIFI) and the DHCP server disabled. The computer is connected with an ethernet cable and should have an IP address within the chosen subnet (eg 192.168.2.2). Now the Parrot AR.Drone can be connected to the AP by configuring the network interface of the drone via telnet[6]: telnet

[6]The computer that is used to configure the quadcopter over telnet should be connected to the wifi of the

DRONE-IP (eg telnet 192.168.1.1). Once access is acquired to the drone, the following bash commands should be executed:

wifi-configuration.sh

```
killall udhcpd;
iwconfig ath0 mode managed essid DRONE-WIFI;
ifconfig ath0 192.168.2.165 netmask 255.255.255.0 up;
```

AR.Drone itself (eg ardrone2_306905).

2. Software Platform

In this thesis, a lot of existing software packages were used. First of all there is the core of the system: ROS which is the interface between the Parrot AR.Drone and the software that is off-board running. Secondly OpenCv is used for the 2D vision processing and PCL for processing in 3D on the point cloud. Thirdly different SLAM software packages were experimented with: LSD SLAM[1], PTAM[3] and ORB SLAM[4]. Additionally, different packages were used for the state estimation.

2.1. Libraries

2.1.1. OpenCV

OpenCV[5] is a computer vision toolkit that was originally developed by Intel but is now a standalone project released under a BSD license and hence free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing by using different multi-threading libraries like OpenMP or Intel thread building blocks (TBB). Enabled with OpenCL or CUDA, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. OpenCV has a user community of more than 47 thousand people and more than 9 million downloads.

2.1.2. PCL

Point-cloud processing (section 5.3) is done using Point Cloud Library (PCL). This library was originally part of ROS but became an independent platform that is now widely supported by the industry and by a big community. The library provides a vast amount of interfaces to widely used functions in point-cloud processing with different optimisations over former naive implementations.

2.2. ROS

Given that a lot of existing vision and robotics software is written for ROS and that a ROS driver exists for the Parrot AR.Drone[6], it was a logical choice to use the Robot Operating System (ROS) as the platform to work on. Good literature to start using ROS is the wiki of course[1] but also the free book: *A Gentle Introduction to ROS*[7].

[1] `http://www.wiki.ros.org`

The ROS Package `ardrone_autonomy`[6] provides the interface to the Parrot AR.Drone. It makes use of the software development kit (SDK) (version 2.0) that is provided by Parrot and offers basic interface functionalities such as video streaming of both cameras (but not simultaneously), updates about the navdata, magnetometer, pressure sensor, ...

2.3. Camera calibration

Before processing the images, camera distortions need to be eliminated. In this work, two different programs are used to calibrate the camera of the Parrot AR.Drone. The first algorithm is based on the model that is used in OpenCV [8]. The second algorithm is a built-in function of PTAM [9].

2.3.1. Default ROS camera calibration

As already mentioned in the introduction of this section, the default ROS camera calibration tool makes use of a well-known OpenCV algorithm [8] to compute the camera matrix, radial distortion coefficients and the tangential distortion coefficients.

The pinhole camera model [10] is used to characterize the camera. This model is characterised by the camera parameters and is shown in Figure 2.1. The used parameters are

- f the focal distance
- u_0 the first ordinate of the principle point (image center)
- v_0 the second ordinate of the principle point

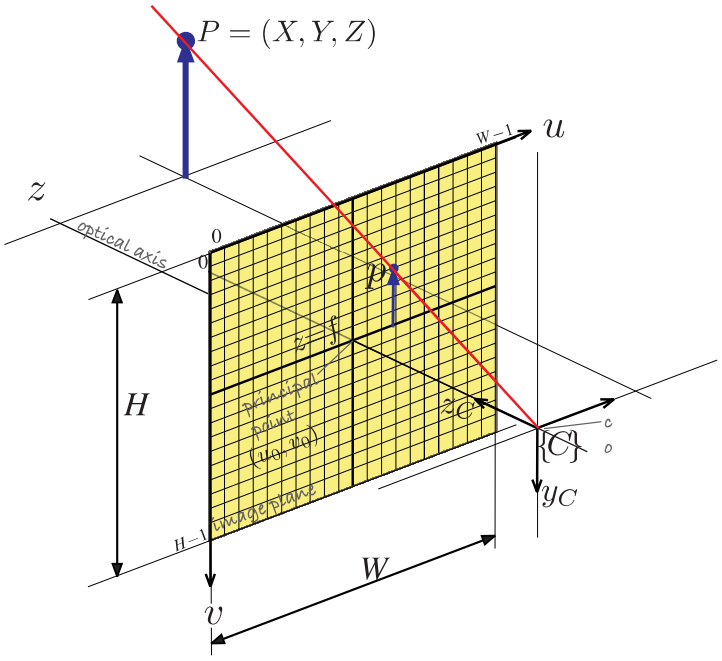


Figure 2.1.: Central projection model showing image plane and discrete pixels,[10]

The following projection equation is obtained [10]:

$$\tilde{p} = \begin{bmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \end{bmatrix} \tilde{P}_C = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{P}_C \tag{2.1}$$

where $\tilde{P}_C$ is the homogeneous world coordinate of the point P in the camera frame and $\tilde{p}$ is the homogeneous point in pixel coordinates. The pixel sizes are given by ρ_w and ρ_h and the focal lengths that take those pixel sizes into account are respectively f_x and f_y. Doing some further expansion of this equation the following equation is obtained:

$$\tilde{p} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} ({}^0T_C)^{-1} \tilde{P} \tag{2.2}$$

The matrix ${}^0T_c$ is the matrix notation of the transform from the world to the frame. The OpenCV library [8] uses a slightly different notation:

$$sm' = A \left[R|t\right] M' \Leftrightarrow s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[R \mid t\right] \tilde{P} \tag{2.3}$$

with c_x and c_y another notation for u_0 and v_0, R is the rotation matrix and t the translation matrix, s is the scaling factor, and u and v the ordinates of the pixel on the image. The joint rotation-translation matrix, $[R \mid t]$, is also called a matrix of extrinsic parameters. The matrix A is called the camera matrix and contains all the intrinsic parameters.

Formula (2.3) is equivalent to:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \Rightarrow \begin{cases} x' = \frac{x}{z} \\ y' = \frac{y}{z} \end{cases} \Rightarrow \begin{cases} u = f_x \cdot x' + c_x \\ v = f_y \cdot y' + c_y \end{cases} \tag{2.4}$$

However, if one wants to take radial k_i and tangential p_i distortions into account, the model should be extended:

$$\begin{cases} x'' = x' \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2p_1x'y' + p_2(r^2+2x'^2) \\ y'' = y' \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2p_2x'y' + p_1(r^2+2y'^2) \end{cases} \Rightarrow \begin{cases} u = f_x \cdot x'' + c_x \\ v = f_y \cdot y'' + c_y \end{cases} \tag{2.5}$$

2.3.2. ROS image topics

In ROS, an image that is streamed from a camera usually has its own namespace. For the AR.Drone there is for example a namespace `/ardrone/front` and `/ardrone/bottom` for respectively the front and bottom camera. Within this namespace, two topics are declared. Live images of the quadcopter are streamed to the `image_raw` topic and the camera parameters are sent to the `camera_info` topic.

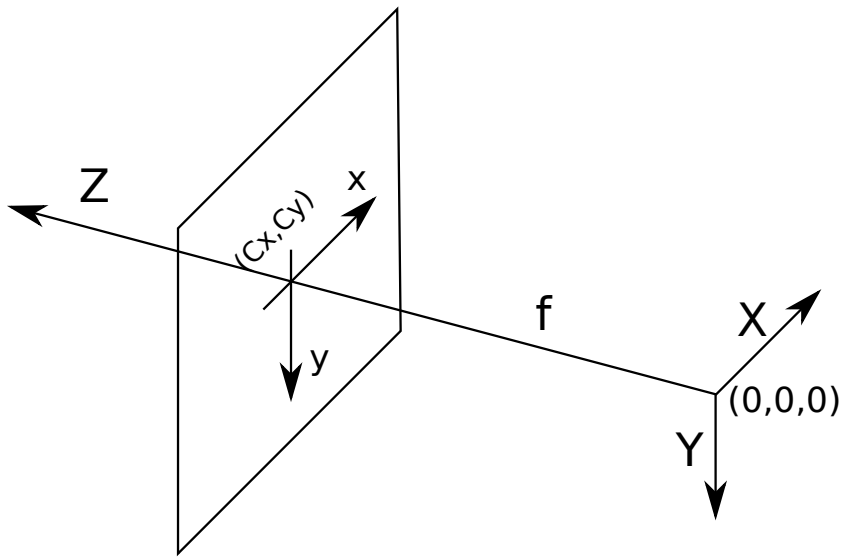


Figure 2.2.: Camera coordinate system

Topic information

The `camera_info` topic contains a header with a timestamp and the frame ID. Furthermore, the image resolution, distortion model, distortion vector D, camera matrix K, rectification matrix R (stereo cameras only), projection matrix P and some additional operational parameters regarding the region of interest are included [11].

For a monocular camera, only the distortion vector $D = [k_1, k_2, p_1, p_2, k_3]$ that specifies the parameters for Equation (2.5) and the camera matrix,

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

are of interest.

The higher order distortion coefficients from (2.5) are not included [12].

Monocular rectification

The goal of rectification is to remove the distortions from the original image. Additionally, rotation, in-plane translation, and scale changes can be added. However in the case of a simple monocular rectification, only the distortions are removed (Figure 2.3).

To create a rectified image from the camera image, a transformation $K - D - K'$ is required. $K - D$ normalizes and undistorts the image. $K' = K^{-1}$ converts the image back to pixel coordinates. X' are the coordinates as seen from the camera's Point of view (POV) and X are the homogeneous world coordinates.

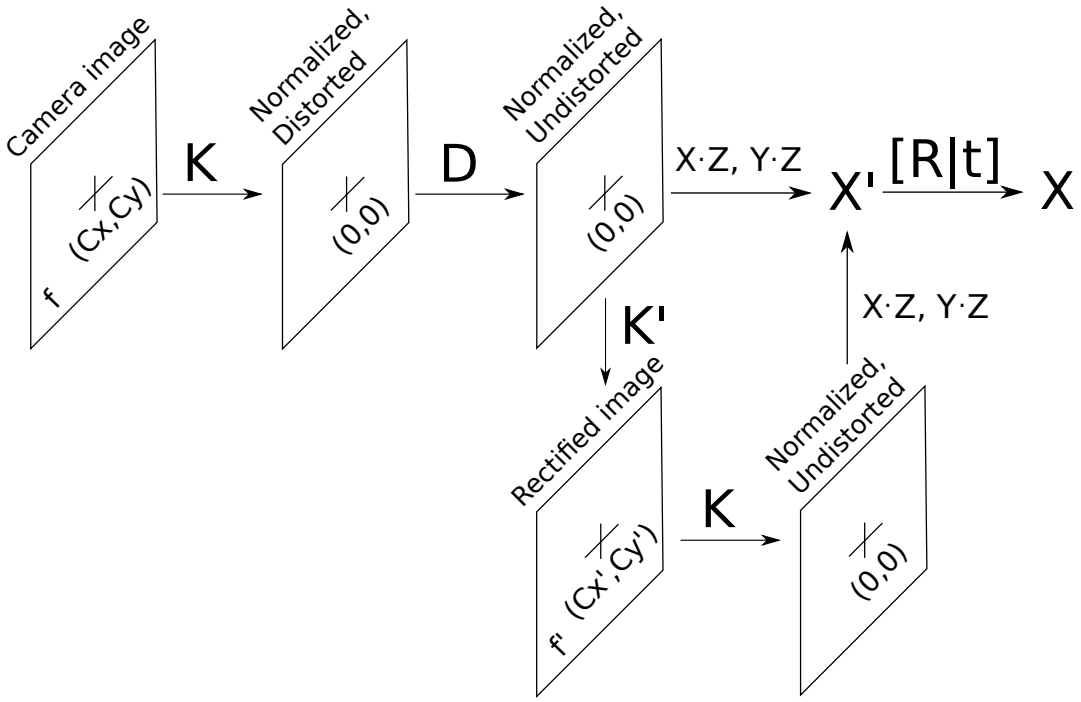


Figure 2.3.: Simple monocular rectification scheme

2.3.3. ROS camera calibration tool

The package that is provided in ROS to handle the camera calibration is called `camera_calibration` and is part of the more general `image_pipeline` package.

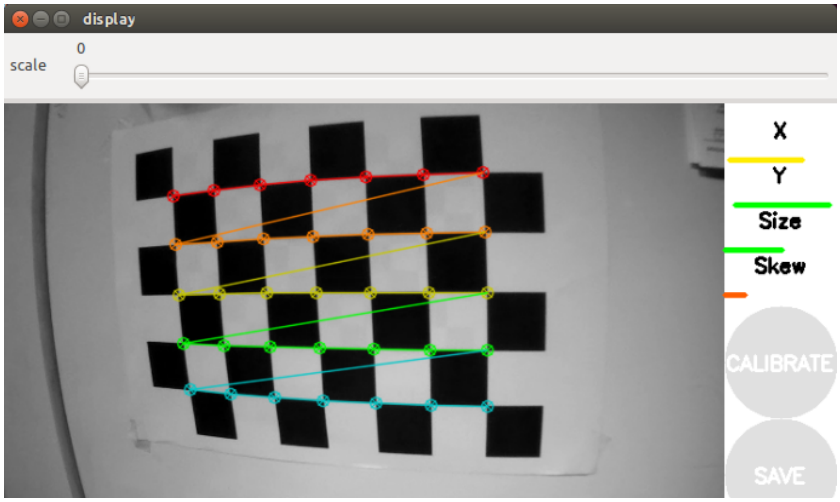


Figure 2.4.: OpenCV camera calibration

The calibration requires a preferably large chequerboard image such as the one that is illustrated in Figure 2.5. For this thesis a 7×5 (inner corners) chequerboard was used with $107.5\,\text{mm} \times 107.5\,\text{mm}$ squares. The result of the calibration can be found in Appendix E.1. The command to start the calibration is the following:

```
rosrun camera_calibration cameracalibrator.py --size 7x5 --square
    0.1075 image:=/ardrone/front/image_raw camera:=/ardrone/front/
    camera_info
```

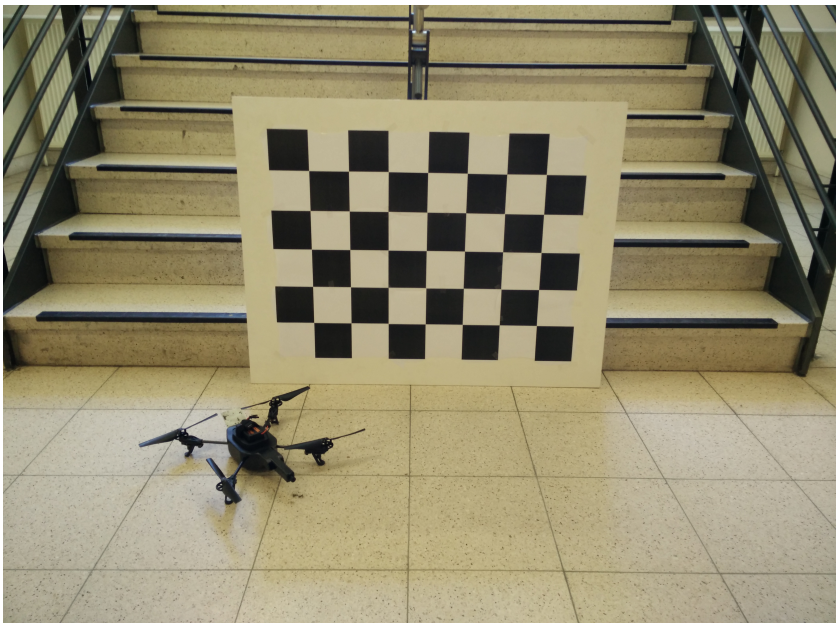


Figure 2.5.: Camera calibration chequerboard

2.3.4. PTAM camera calibration

The PTAM calibration tool that is provided in the `ethzasl_ptam` package calculates the intrinsic camera parameters of a camera with a wide angle lens. The five fixed PTAM parameters (Appendix E.2) are $\{f_x, f_y, c_x, c_y, s\}$. This calibration tool works only on grayscale images so an intermediary node is needed to convert the RGB images from the Parrot AR.Drone to grayscale images. This can be accomplished by the `image_proc` package [13]. Once the grayscale image topic is created, the PTAM calibrator can be launched. The launch file can be found in Appendix E.3. A good calibration has a re-projection error (RMS pixel error) below 0.3 for wide angle lenses.

Listing 2.1: PTAM callibration procedure

```
# Create a /ardrone/front/image_mono topic
ROS_NAMESPACE=/ardrone/front/ rosrun image_proc image_proc
# Start the PTAM Cameracalibrator
roslaunch ptam cameracalibrator.launch
```

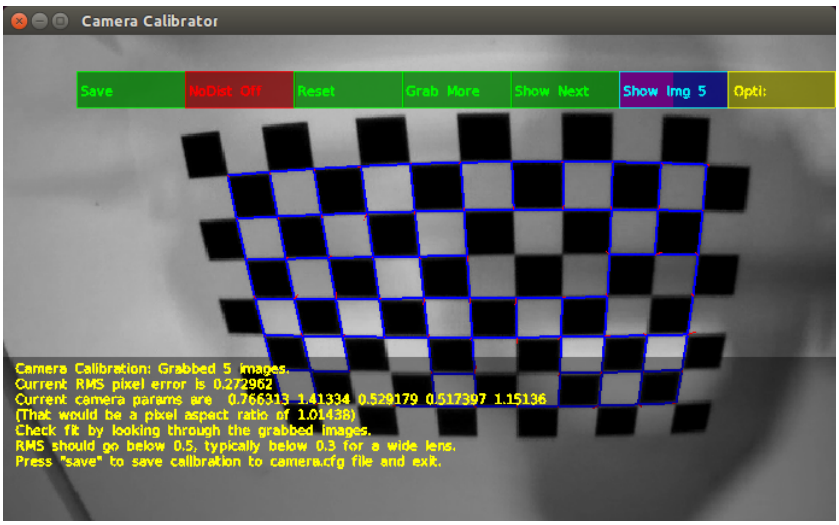


Figure 2.6.: PTAM camera calibration

2.4. SLAM

2.5. Simulation and visualization

Some good software tools exist for visualising what is going on with the quadcopter. A possible way to evaluate certain software packages, is the usage of Gazebo where a simulated drone can be used. Gazebo is the most popular software packet for this job. Using the `hector_quadrotor` package[14], a quadcopter can be simulated. A more specific implementation of the AR.Drone for Gazebo is also available under the package `tum_simulator` [15].

A powerful tool to visualize what is happening in real-time with the drone, is rviz[16]. It is a 3D visualization tool for ROS that allows you to visualize the pose of a robot, such as a quadcopter, and to plot markers and all kind of shapes. In other words, it can visualize the position of the drone, or at least where it thinks it is, and project possible path planning markers. All this can be very useful for debugging new software.

2.6. State estimation

There exist multiple integrated state estimation packages for ROS. Some examples are `robot_pose_ekf` [17], `robot_localization` [18] and `tum_ardrone` [19]. The latter is used in this thesis because it implements an Extended Kalman Filter (EKF) that is specifically designed for the Parrot AR.Drone. More theoretical background can be found in chapter 3.

2.7. Control software

The Parrot AR.Drone 2.0 can be controlled by a lot of different controllers. There exist applications for Android, Apple iOS, and Windows Phone[2]. Not only can the AR.Drone be controlled

[2] `http://ardrone2.parrot.com/usa/apps/`

from mobile devices but also from a computer. Applications such as `node-ar-drone`[3] (implementation for node.js), `tum_ardrone`[4] (manual and autonomous control in ROS), `ardrone_tutorials`[5] (manual control in ROS), `wii-drone`[6] and many more exist [20].

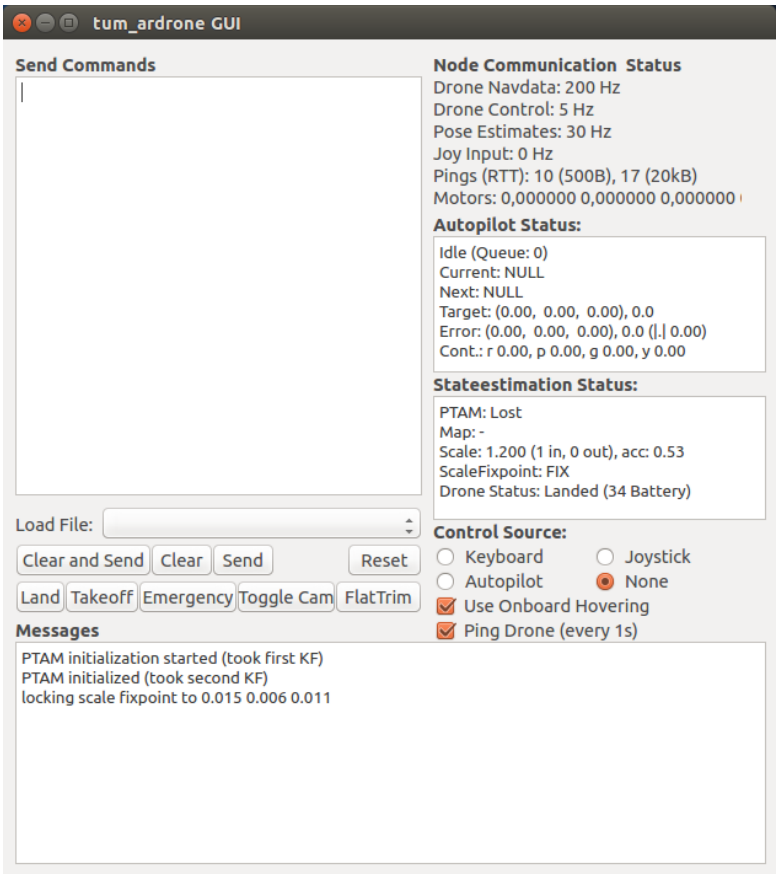


Figure 2.7.: Graphical user interface of tum_ardrone

The `tum_ardrone` package [19] is used in this thesis to control the drone because it is integrated in ROS and has an integrated EKF to track the movements of the quadcopter. More information about this package is given in section 3.2. In Figure 2.7, the GUI is displayed and contains all the essential information regarding the drone's status. The node communication status box provides real-time update speeds, ping delays, and the status of the motors. The autopilot status box gives information about the commands that are queued, the current target position, and the error between the estimated position and the target position. The state estimation status gives information about PTAM, the EKF, and the drone status. Manual control is possible with a keyboard or a joystick. If the autopilot function is used, a quick switch to keyboard control can be made by pressing the ESC button. The input window on the left is used to send commands to the autopilot. A list of commands can be read out from a text file. The visual output of PTAM and the EKF are given in separate windows (Figure 4.6).

[3] `https://github.com/felixge/node-ar-drone`
[4] `https://github.com/tum-vision/tum_ardrone`
[5] `http://robohub.org/up-and-flying-with-the-ar-drone-and-ros-getting-started/`
[6] `https://github.com/voodootikigod/wii-drone`

2.8. Cooperation and methodology

The authors of this thesis use a private server to share code, thoughts, and tutorials (wiki). Gitlab[7] is used to accomplish all this. Gitlab was installed on a server running Debian with Nginx as webserver. The connection to the server is established by an OpenVPN tunnel.

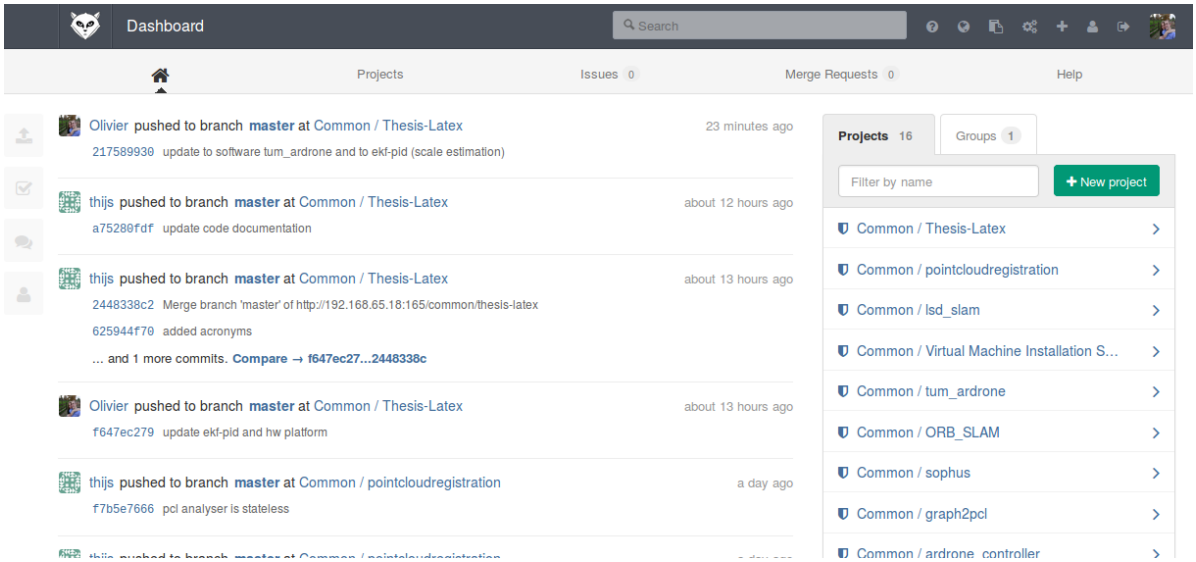


Figure 2.8.: Web interface of Gitlab

Gitlab offers multiple options for a fruitful cooperation. The most important functionalities are code version management, wiki pages and issue management. All three functionalities are used in the development of this thesis. More info is given in chapter 7.

Google keep is used for the exchange of quick notes and to-do-lists.

[7] `https://about.gitlab.com/`

3. Position estimation and control system

In the literature, a lot can be found about control systems for unmanned aerial systemss (UASs). The most popular are based on the EKF. This is mainly because most robotics models are non-linear and the Extended Kalman Filter happens to be the de facto standard in the theory of non-linear state estimation. This chapter will first describe some general concepts concerning state estimation in order to justify the chosen framework. In the second half of this chapter, the actual implementation for this thesis will be elaborated.

3.1. State estimation

In robotics and control applications, one needs to know the state of the system. The state is every property of the system. The scope of the state in robotics is regularly limited to the position and orientation.

A dynamic state space model (DSSM) involves time dependant states $\mathbf{x}_t$ that are dynamically linked. Mostly these time steps are discrete. More generally the dynamic link may be stochastic and the link is described by a probability density function (PDF) as shown by

$$f(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots).$$

A dynamic model tries to describe the PDF with an analytical expression.

3.1.1. Markov model

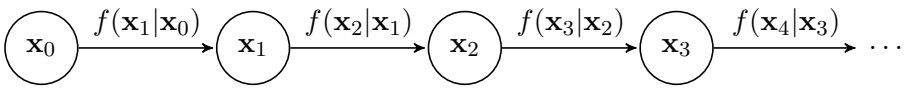
A dynamical model is Markov of order p if the time dependency is limited to p steps or.

$$f(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots) = f(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_{t-p}).$$

A first order Markov system only depends on the previous state. Consider a $\mathcal{X}$-valued discrete-time Markov process of order one, $\{X_n\}_{n\geq 1}$, described by

$$X_1 \sim \mu(\mathbf{x}_1) \text{ and } X_n|(X_{n-1} = \mathbf{x}_{n-1}) \sim f(\mathbf{x}_n|\mathbf{x}_{n-1})$$

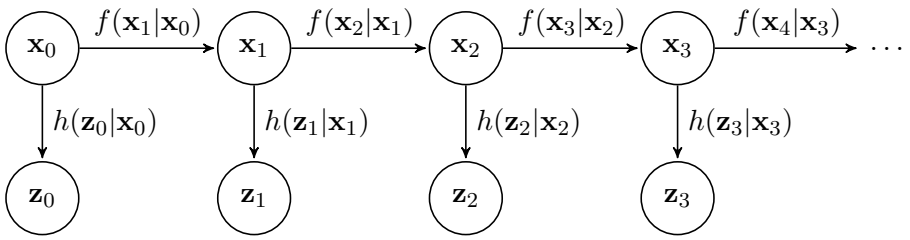
where $\sim$ means distributed according to and $\mu(x)$ is a PDF. This kind of Markov model is symbolised below.



In a hidden markov model (HMM) the state is not observable or not completely observable. However, the $\mathcal{Z}$-valued process $\{Z_n\}_{n\leq 1}$ is observable. It is assumed that, given $\{X_n\}_{n\geq 1}$, the observations $\{Z_n\}_{n\geq 1}$ are statistically independent and their marginal densities given by

$$Z_n|(X_n = \mathbf{x}_n) \sim h(\mathbf{z}_n|\mathbf{x}_n). \tag{3.1}$$

The states in a HMM are by convention discrete and the observables continuous. A graphical representation is given below.



A Kalman state space model describes more or less the same dynamics, but the hidden states and observations are Gaussian and all relations in the model are linear.

3.1.2. Filtering or observing

Filtering is characterising the distribution of the state of the HMM at the present time given all previous received observations. This practice is also referred to as using an observer. The filter must be matched to the system to obtain good results. Within the scope of this thesis the state is the pose and the outputs of the system are the readings from the on-board sensors and the input from PTAM (section 3.2).

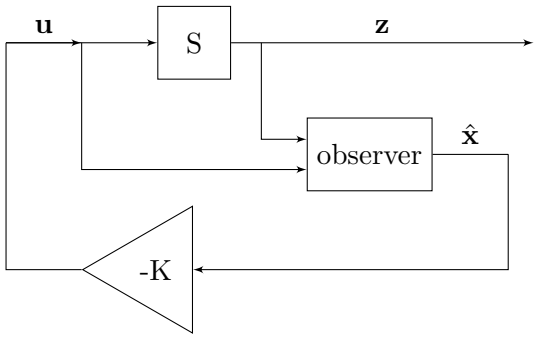


Figure 3.1.: Control system with feedback by observer [21]

3.1.3. Bayes filter

A Bayes filter, or recursive Bayesian estimation, uses the basic principle of estimating a HMM. Almost all other techniques are practical implementations or specific cases of the Bayes filter. The bayes filter assumes a Markov process of order one and, thus, Equation (3.1) is satisfied. Bayes theorem is used to calculate the a posteriori distribution of the states $\{X_n\}_{n\geq 1}$ given the measurements $\{Z_n\}_{n\geq 1}$ as shown by

$$p(\mathbf{x}_{1:k}|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_{1:k}|\mathbf{x}_{1:k})p(\mathbf{x}_{1:k})}{p(\mathbf{z}_{1:k})} \tag{3.2}$$

with

- $p(\mathbf{x}_{0:k})$ is the prior distribution defined by the dynamic model given by

$$p(\mathbf{x}_{0:k}) = \mu(\mathbf{x}_1) \prod_{m=2}^{k} f(\mathbf{x}_m|\mathbf{x}_{m-1}),$$

- $p(\mathbf{z}_{1:k}|\mathbf{x}_{0:k})$ is the likelihood model for the measurements given by

$$p(\mathbf{z}_{1:k}|\mathbf{x}_{0:k}) = \prod_{m=1}^{k} g(\mathbf{z}_m|\mathbf{x}_m),$$

- $p(\mathbf{z}_{1:k})$ the normalisation constant defined as

$$p(\mathbf{z}_{1:k}) = \int p(\mathbf{z}_{1:k}|\mathbf{x}_{1:k}p(\mathbf{x}_{1:k}))d_{\mathbf{x}_{1:k}}.$$

3.1.4. Particle filter

A recursive Bayesian estimation technique that fairly recently gained interest is the so called particle filter. This technique omits the need of having and exploitable analytical distribution by only using samples but it is computational expensive [22]. Therefore, current real time implementations are GPU based. Next to the introduction in this thesis based on [22] a good graphical introduction to the subject can be found in [23].

Particle filters and sequential Monte Carlo (SMC) are in literature sometimes represented as the same. Although it is true that particle filters are almost solely based upon SMC methods, SMC extends to much more applications.

Basics of Monte Carlo

Consider a generic PDF $\pi_k(\mathbf{x}_{1:k})$ for a fixed number of k variables. Sequential Monte Carlo methods are a general class within the Monte Carlo methods that sample sequentially from PDFs $\{\pi_k(\mathbf{x}_{1:k})\}$. Each PDF is defined on the product space $\mathcal{X}^k$. If

$$\pi_k(\mathbf{x}_{1:k}) = \frac{\gamma_k(\mathbf{x}_{1:k})}{Z_k} \tag{3.3}$$

then $\gamma_k : \mathcal{X}^k \to \mathbb{R}^+$ only needs to be known pointwise. The normalising constant

$$Z_k = \int \gamma_k(\mathbf{x}_{1:k}) d\mathbf{x}_{1:k}$$

is not needed to be known.

The aforementioned PDF will sampled by N independent variables, $X^i_{1:k} \sim \pi_n(\mathbf{x}_{1:k})$ for $i = 1, \ldots, N$. The PDF $\pi_k(\mathbf{x}_{1:k})$ is thus approximated by

$$\hat{\pi}_k(\mathbf{x}_{1:k}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{X^i_{1:k}}(\mathbf{x}_{1:k}) \tag{3.4}$$

where $\delta_{\mathbf{x}_0}(x)$ denotes the Dirac delta mass function at $\mathbf{x}_0$.

Based on Equation (3.4), any marginal $\mathbf{x}_m$ is given by

$$\hat{\pi}_k(\mathbf{x}_m) = \frac{1}{N}\sum_{i=1}^{N}\delta_{X_m^i}(\mathbf{x}_m).$$

The expectation of a test function $\varphi_k : \mathcal{X}^k \to \mathbb{R}$ is given by

$$I_k(\varphi_k) := \int \varphi_k(\mathbf{x}_{1:k})\pi_k(\mathbf{x}_{1:k})d\mathbf{x}_{1:k}$$

is estimated by Equation (3.5).

$$I_k^{MC}(\varphi_k) := \int \varphi_k(\mathbf{x}_{1:k})\hat{\pi}_k(\mathbf{x}_{1:k})d\mathbf{x}_{1:k} = \frac{1}{N}\sum_{i=1}^{N}\varphi_k(X_{1:k}^i) \tag{3.5}$$

The main advantage of this approach is that arbitrary PDFs are possible and that the approximation error decreases at a rate of $\mathcal{O}(1/N)$.

Sampling of complex high-dimensional PDF is not practically possible as most systems only can sample a uniform distribution or a Gaussian distribution. Therefore, importance sampling is used. Here we sample with a distribution that is easy to sample like a multivariate Gaussian distribution and take into account the difference with the real PDF by using weights. This type of sampling is the basis of all the algorithms. Importance sampling relies on the introduction of an importance density $q_k(\mathbf{x}_{1:k})$ such that

$$\pi_k(\mathbf{x}_{1:k}) > 0 \Rightarrow q_k(\mathbf{x}_{1:k}) > 0.$$

Applying this to Equation (3.3) results in

$$\pi_k(\mathbf{x}_{1:k}) = \frac{w_k(\mathbf{x}_{1:k})q_k(\mathbf{x}_{1:k})}{Z_k},$$

where $w_k(\mathbf{x}_{1:k})$ is the unnormalised weight function that corrects for the difference between the real PDF and the sampling distribution. The normalising constant Z_k is adapted likewise.

Monte Carlo applied to filter

Specifically in the filter context a numerically approximation of $\{p(\mathbf{x}_{1:k}|\mathbf{z}_{1:k})\}_{k\geq 1}$ needs to be calculated. More variations on mapping target functions are possible. To present the algorithm re-sampling is done at every time stamp.

Particles are initialised based on the dynamic model and are samples of the importance density function at time $n = 1$. The weight of every particle is calculated and a set of weighted particles $\{W_1^i, X_1^i\}$ is obtained. The weighting is based on the observations and can be seen as an update or correction step conform the kalman filter (see further).

To eliminate the samples with a lower weight and to keep more or less the same number of particles N resampling is done and results in the set $\left\{\frac{1}{N}, \bar{X}_1^i\right\}$. This is sequentially repeated for further time steps. The complete algorithm is shown in Listing 3.1.

While not discussed explicitly in this part control commands can be taken into account by plugging them in the motion model $f(\mathbf{x}_k|\mathbf{x}_{k-1})$;

Listing 3.1: Sequential importance resampling on sequential Monte Carlo algorithm for Filtering [22]

1 At time $n = 1$
2 Sample $X_1^i \sim q(\mathbf{x}_1|\mathbf{z}_1)$
3 Compute the weights $w_1 = \frac{\mu(X_1^i)g(\mathbf{x}_1|X_1^i)}{q(X_1^i|\mathbf{z}_1)}$ and $W_i \propto w_1(X_1^i)$ with $\mu(X_1^i)$ the initial PDF of X_1^i
4 Resample $\{W_1^i, X_1^i\}$ to obtain N equally−weighted particles $\left\{\frac{1}{N}, \bar{X}_1^i\right\}$
5
6 At time $n \geq 2$
7 Sample $X_n^i \sim q(\mathbf{x}_n|\mathbf{z}_n, \bar{X}_{n-1}^i)$ and set $X_{1:n}^i \leftarrow \left(\bar{X}_{1:n-1}^i, X_n^i\right)$
8 Compute the weights $\alpha_n = \frac{g(\mathbf{x}_n|X_n^i)f\left(X_n^i, X_{n-1}^i\right)}{q(\mathbf{x}_n|\mathbf{z}_n, X_{n-1}^i)}$ and $W_i \propto \alpha_n(X_{n-1:n}^i)$
9 Resample $\{W_n^i, X_{1:n}^i\}$ to obtain N equally−weighted particles $\left\{\frac{1}{N}, \bar{X}_{1:n}^i\right\}$

3.1.5. Extended Kalman Filter

The kalman filter is a Bayes filter that is based on the kalman model and, thus, all density functions are Gaussian. As discussed before, the hidden state space model depends linearly on the previous states with zero mean additive white Gaussian noise (AWGN) noise in a Kalman model. The general model also takes into account the control input $\mathbf{u}$. The general discrete form is given by

$$\mathbf{x}_n = \mathbf{F}_n\mathbf{x}_{n-1} + \mathbf{B}_n\mathbf{u}_n + \mathbf{W}_n \tag{3.6}$$

with

- $\mathbf{F}_n$ the state transition model;
- $\mathbf{B}_n$ the control input model applied to the control vector u_n;
- $\mathbf{W}_n \sim N(0, Q_n)$ the process noise with covariance Q_n.

The discrete observation is given by

$$\mathbf{z}_n = \mathbf{H}_n * \mathbf{x}_n + \mathbf{V}_n \tag{3.7}$$

with

- $\mathbf{H}_n$ the observation model;
- $\mathbf{V}_n \sim N(0, R_n)$ the observation noise with covariance R_n.

In practice $\mathbf{F}_n$, $\mathbf{B}_n$, $\mathbf{W}_n$, $\mathbf{H}_n$ and $\mathbf{V}_n$ are not time dependent and the index n is dropped. By consequence the covariances $\mathbf{Q}_n$ and $\mathbf{R}_n$ are also assumed constant. The Kalman filter will recursively estimate the state. This estimation is subdivided in two phases: predict and update.

Predict

Compute a priori state estimate	$\hat{\mathbf{x}}_{k\|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1\|k-1} + \mathbf{B}\mathbf{u}_k$
Compute a priori estimate error covariance	$\mathbf{P}_{k\|k-1} = \mathbf{F}\mathbf{P}_{k-1\|k-1}\mathbf{F}^{\mathrm{T}} + \mathbf{Q}$

Update

Innovation or measurement residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k\|k-1}$
Innovation (or residual) covariance	$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k\|k-1}\mathbf{H}_k^T + \mathbf{R}$
Optimal Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\|k-1}\mathbf{H}^{\mathrm{T}}\mathbf{S}_k^{-1}$
Updated (a posteriori) state estimate	$\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (a posteriori) estimate covariance	$\mathbf{P}_{k\|k} = (I - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k\|k-1}$

Typically a lot of phenomena can not be characterised by normal laws and the standard Kalman filter can not be applied. A strategy to overcome this is to linearise the input. This is called the EKF.

Some of the most popular ROS packages that already exist are `robot_pose_ekf` [17] and `robot_localization` [18]. The former is specifically made for ground robots and takes the Inertial Measurement Unit (IMU), visual odometry and wheel odometry into account while the latter is a more general package that can be used for different kind of robotic systems.

In stead of implementing an EKF, this work made use of an already implemented version that exists specifically for the AR.Drone: `tum_ardrone` [19, 24–26]. This implementation will be elaborated in section 3.3.

3.2. Control system: `tum_ardrone`

Before describing the control system that is used in this thesis, an important note has to be made about the definitions about position-based visual servo (PBVS), image-based visual servo (IBVS) and visual odometry (VO). Different algorithms exist in the literature and the exact definition of these two concepts is not clearly described. To clear the ambiguity around these definitions, their meaning as it is interpreted from [10] by the authors of this thesis will be retaken in this work.

There are two fundamentally different approaches to visual servo control: position-based visual servo and image-based visual servo. PBVS uses observed visual features, a camera that is calibrated and a known geometric model of the target to determine the pose of the model with respect to the camera. In contrast, IBVS omits the pose estimation process and uses the image features directly to position itself. The desired camera pose with respect to the target is defined implicitly by the image feature values at the goal pose.

Visual odometry is a structure from motion (SfM) technique that records the camera motion from a set of images.

One can say that this thesis makes use of both VO and IBVS. These concepts might become clearer after reading the remainder of this chapter.

3.2.1. Overview of the `tum_ardrone` package

As described in the papers related to `tum_ardrone` [24–26], three different parts can be distinguished. First of all there is the monocular SLAM that is based on PTAM (more information in section 4.4) [3] discussed in section 4.4, secondly there is the implementation of the EKF discussed in section 3.3 that fuses all available data, and thirdly there is the PID controller discussed in section 3.4 to steer the AR.Drone based on the estimates of the EKF.

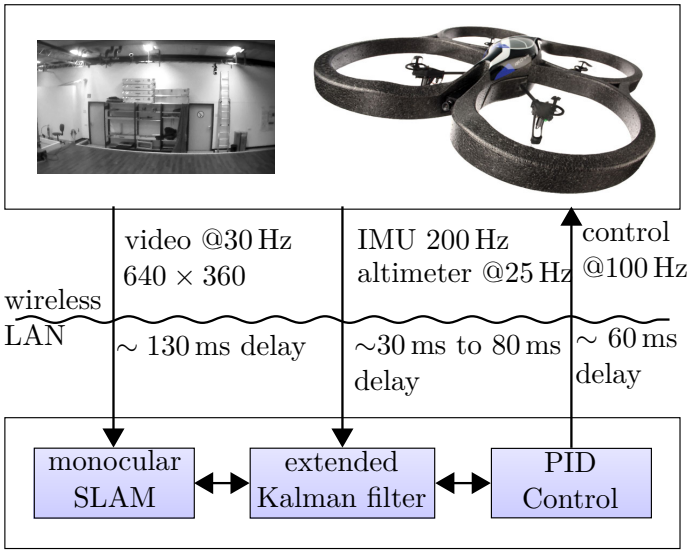


Figure 3.2.: Structure of the `tum_ardrone` package[24]

3.2.2. Modifications

Some minor modifications were made to integrate `tum_ardrone` in this thesis. All changes are listed in section B.1. Here is a summary of the changes:

- A custom camera calibration file is added to the package (section E.2).
- The drone's custom IP address is changed in the launch file and in the *PingThread* class.
- The keyboard control keys were adapted to an azerty keyboard: (e,d) for pitch, (s,f) for roll, (j,l) for yaw and (i,k) for height. Furthermore: t is take off, y is land and u is emergency.
- A tf position broadcaster is added to the package to publish the current estimated position of the quadcopter.

3.3. Extended Kalman Filter in `tum_ardrone`

Data fusion between the onboard sensor readings is done by an EKF. These sensors are already described in chapter 1 but not all of them are used in `tum_ardrone`.

To be able to use an EKF, a full motion model should be provided for the AR.Drone's flight dynamics and for its reaction to control commands. Different publications [24, 27, 28] already

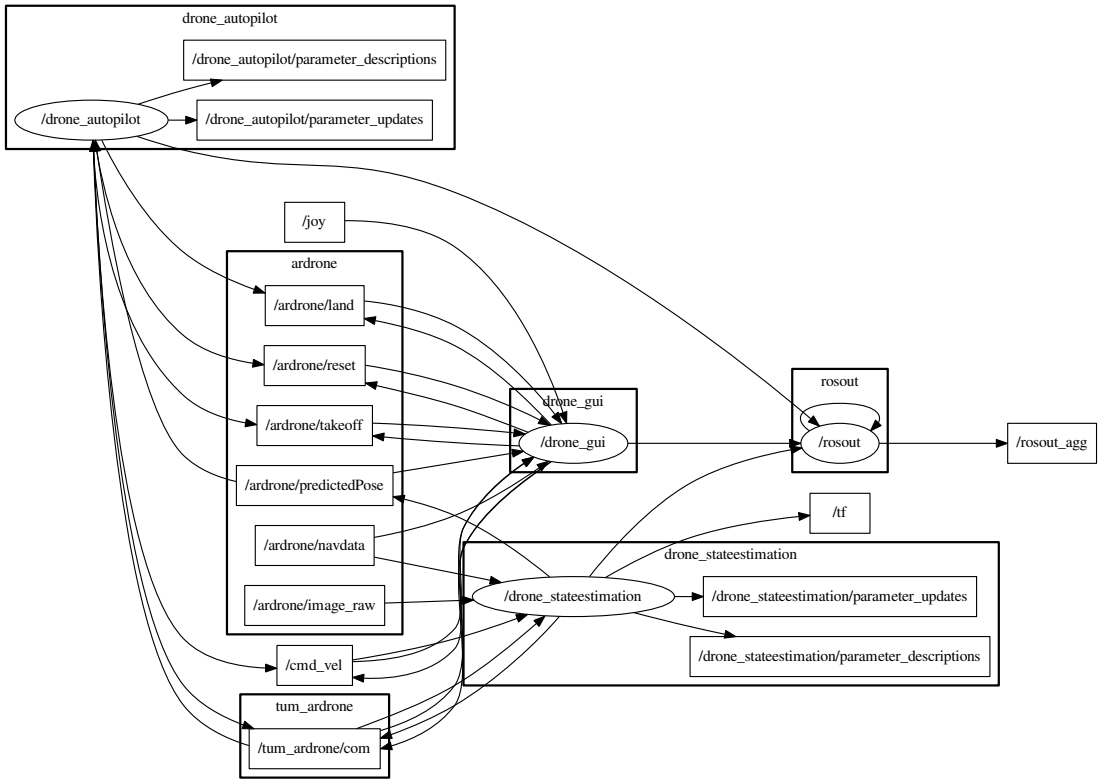


Figure 3.3.: Graph of the `tum_ardrone` package

made such a model for the Parrot AR.Drone 2.0 . Therefore, this thesis will use one of these models instead of creating a new one.

The model that is used for this thesis is the one that is already implemented in `tum_ardrone`. The derivation of the model is more elaborated in the thesis of Tang: [27]. To grasp the modelling concept, some elaboration is given below. First, the state space is described, followed by the observation functions $h_i(x_t)$ of each sensor.

3.3.1. Full motion model

State space

`tum_ardrone` makes use of a state space that consists of a total of ten state variables

$$\mathbf{x}_t := (x_t, y_t, z_t, \dot{x}_t, \dot{y}_t, \dot{z}_t, \Phi_t, \Theta_t, \Psi_t, \dot{\Psi}_t)^T \in \mathbb{R}^{10} \tag{3.8}$$

$$\text{where} \begin{cases} (x_t, y_t, z_t) & \text{Position[m]} \\ (\dot{x}_t, \dot{y}_t, \dot{z}_t) & \text{Velocity[m s}^{-1}\text{]} \\ (\Phi_t, \Theta_t, \Psi_t) & \text{roll, pitch, yaw[°]} \\ \dot{\Psi}_t & \text{yaw-rotational speed[° s}^{-1}\text{]} \end{cases}$$

Odometry Observation Model

The observation function $h_I(\mathbf{x}_t)$ of the quadcopter is defined as follows:

$$h_I(\mathbf{x}_t) := \begin{pmatrix} \dot{x}_t \cos\Psi_t - \dot{y}_t \sin\Psi_t \\ \dot{x}_t \sin\Psi_t + \dot{y}_t \cos\Psi_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \dot{\Psi}_t \end{pmatrix} \tag{3.9}$$

The speed is measured in the horizontal plane in the local coordinate frame of the quadcopter. In the first two elements of Equation (3.9) the projection to the global frame is shown $(\hat{v}_{x,t}, \hat{v}_{y,t}) \rightarrow (\dot{x}_t, \dot{y}_t)$. Roll and pitch are directly measured by the accelerometer. The height and yaw-rotational speed are measured by differentiating successive measurements. The measurement vector is given by:

$$\mathbf{z}_{I,t} := (\hat{v}_{x,t}, \hat{v}_{y,t}, \frac{\hat{h}_t - \hat{h}_{t-1}}{\delta_{t-1}}, \hat{\Phi}_t, \hat{\Theta}_t, \frac{\hat{\Psi}_t - \hat{\Psi}_{t-1}}{\delta_{t-1}})^T \tag{3.10}$$

Visual Observation Model

The moment that PTAM tracks the position of the front camera, it outputs a pose estimate (section 4.1) from the camera's POV and is, after projection to the coordinate system of the quadcopter, a direct estimate.

$$\begin{aligned} &\text{Observation function} && h_P(\mathbf{x}_t) := (x_t, y_t, z_t, \Phi_t, \Theta_t, \Psi_t)^T \\ &\text{Observation vector} && \mathbf{z}_{P,t} := f(E_{DC} E_{C,t}) \end{aligned} \tag{3.11}$$

In Equation (3.11), one can see that the camera pose $E_{C,t} \in SE(3)$ is projected to the AR.Drone's body coordinate system with the transformation matrix $E_{DC} \in SE(3)$. The function f simply converts the $SE(3)$ elements to the roll-pitch-yaw representation.

Prediction Model

The model that is used for the prediction of the AR.Drone movements is based an a couple of assumptions.

First of all, the accelerations $\ddot{x}$ and $\ddot{y}$ are estimated from the current state of the AR.Drone i.e. $\ddot{x}$ and $\ddot{y}$ are proportional to the horizontal force and the drag force acting on the quadcopter. These are respectively considered proportional to the tilt angle and the horizontal velocity at the low speeds on which the drone operates. This is formalised by:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} \propto f_{acc} - f_{drag} \Rightarrow \begin{cases} \ddot{x}(\mathbf{x}_t) = c_1(\cos\Psi_t \sin\Phi_t \cos\Theta_t - \sin\Psi_t \sin\Theta_t) - c_2 \dot{x}_t \\ \ddot{y}(\mathbf{x}_t) = c_1(-\sin\Psi_t \sin\Phi_t \cos\Theta_t - \cos\Psi_t \sin\Theta_t) - c_2 \dot{x}_t \end{cases} .$$

The assumption that was made to calculate c_1 and c_2 is the following: overall thrust generated by the four motors is constant.

Secondly there are the vertical acceleration $\ddot{z}$, yaw-rotational acceleration $\ddot{\Psi}$, roll rotational speed $\dot{\Phi}$ and pitch rotational speed $\dot{\Theta}$. These parameters are estimated using the current state $\mathbf{x}_t$ as well as the control commands $\mathbf{u}_t = (\bar{\Phi}_t, \bar{\Theta}_t, \bar{\dot{z}}_t, \bar{\dot{\Psi}}_t)$ that are send to the quadcopter.

$$\begin{aligned} \dot{\Phi}(\mathbf{x}_t, \mathbf{u}_t) &= c_3\bar{\Phi}_t - c_4\Phi_t \\ \dot{\Theta}(\mathbf{x}_t, \mathbf{u}_t) &= c_3\bar{\Theta}_t - c_4\Theta_t \\ \ddot{\Psi}(\mathbf{x}_t, \mathbf{u}_t) &= c_5\bar{\dot{\Psi}}_t - c_6\dot{\Psi}_t \\ \ddot{z}(\mathbf{x}_t, \mathbf{u}_t) &= c_7\bar{\dot{z}}_t - c_8\dot{z}_t \end{aligned} \tag{3.12}$$

Thus, the total state estimation can be written as follows:

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \Psi_t \\ \dot{\Psi}_t \end{pmatrix} + \delta_t \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \ddot{x}(\mathbf{x}_t) \\ \ddot{y}(\mathbf{x}_t) \\ \ddot{z}(\mathbf{x}_t, \mathbf{u}_t) \\ \dot{\Phi}(\mathbf{x}_t, \mathbf{u}_t) \\ \dot{\Theta}(\mathbf{x}_t, \mathbf{u}_t) \\ \dot{\Psi}_t \\ \ddot{\Psi}(\mathbf{x}_t, \mathbf{u}_t) \end{pmatrix} \rightarrow \begin{pmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \\ \dot{z}_{t+1} \\ \Phi_{t+1} \\ \Theta_{t+1} \\ \Psi_{t+1} \\ \dot{\Psi}_{t+1} \end{pmatrix} \tag{3.13}$$

This model that is presented by `tum_ardrone` comes with no guarantees neither do they claim physical correctness. However, it seems to perform rather well in practice and provides good estimates for a short period of time ($\sim 125\,\text{ms}$).

3.3.2. Delay handling

One of the most important factors to take into account in the EKF is the delay between al the observed data and the commands that are sent. The `tum_ardrone` team designed the EKF so that it stores all observational data and command messages in a temporal observation buffer. At $t - \triangle t_{vis}$ the last visual observation was done. From that moment on, the EKF uses all the available sensor data and control commands to calculates the AR.Drone's position ahead up to $t + \triangle t_{control}$. This is visualized in figure 3.4. Because the time delay is not always constant and depends on the quality of the wireless connection, Internet Control Message Protocol (ICMP) echo requests are send back and forth in order to check this quality and better determine the delay.

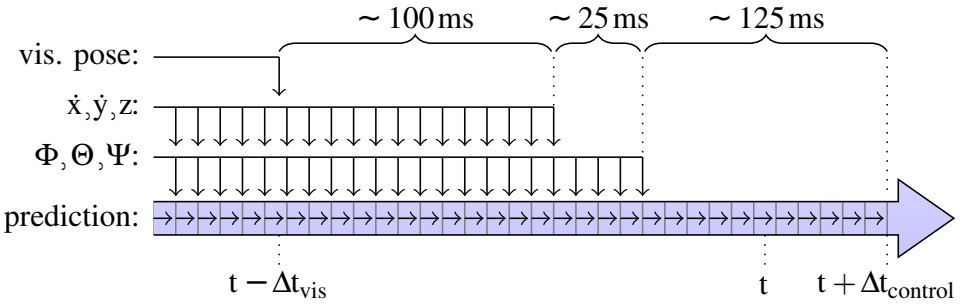


Figure 3.4.: History of observations and commands between $t - \triangle t_{vis}$ and $t + \triangle t_{control}$

To determine the influence of an access point (AP), some ping data was captured and analysed. The results are shown in Figure 3.5, 3.6 and 3.7. The histograms clearly show the instability of the wireless connection between the internal network card of a laptop and the drone. The ping results between the laptop that is connected to an AP via a wire, which has a wireless connection to the drone, clearly shows the superiority of the connection. The difference between the static measurements and in-flight measurements is negligible. The mean ping value for the 500 B package is 3.55 ms and 19.07 ms for the 20 kB package.

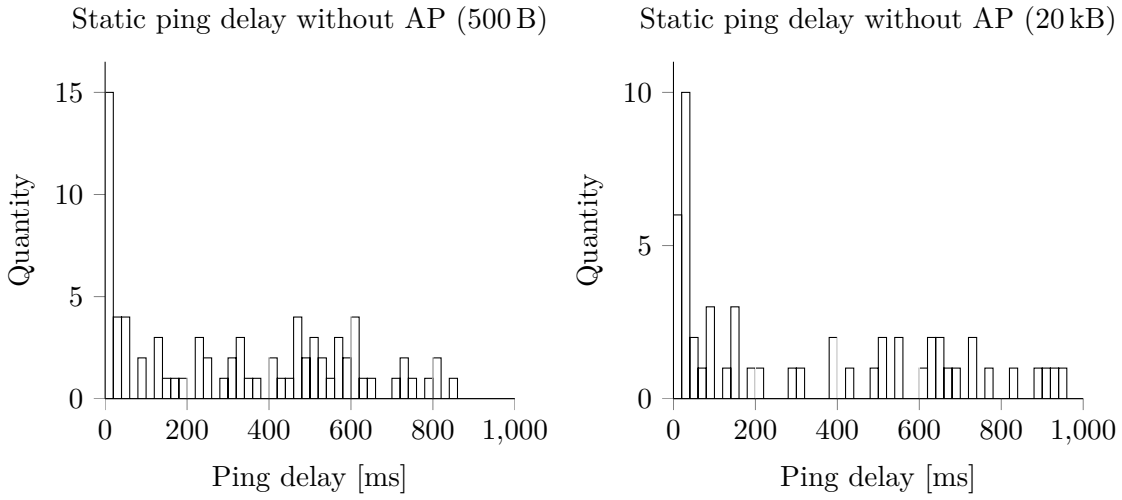


Figure 3.5.: Static ping delay without an access point. The large spread and high mean value of the ping delays indicate the low quality of the wireless link.

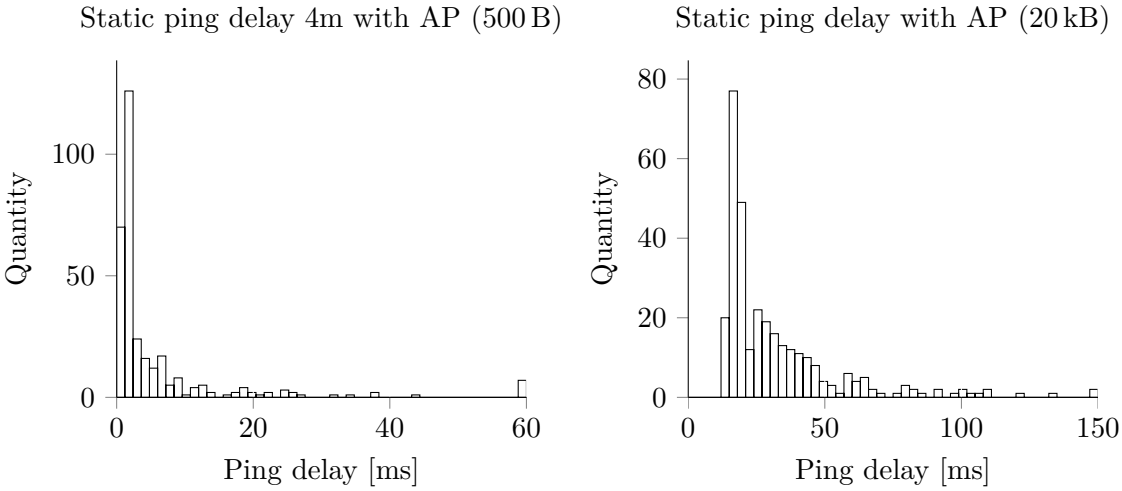


Figure 3.6.: Static ping delay with an access point. The use of a wireless AP significantly lowers the delay and stability of the wireless connection.

3.4. PID control

A PID controller is implemented in `tum_ardrone` for each of the four degrees of freedom. The generated commands make use of the position and velocity estimates at $t + \triangle t_{control}$ and steer

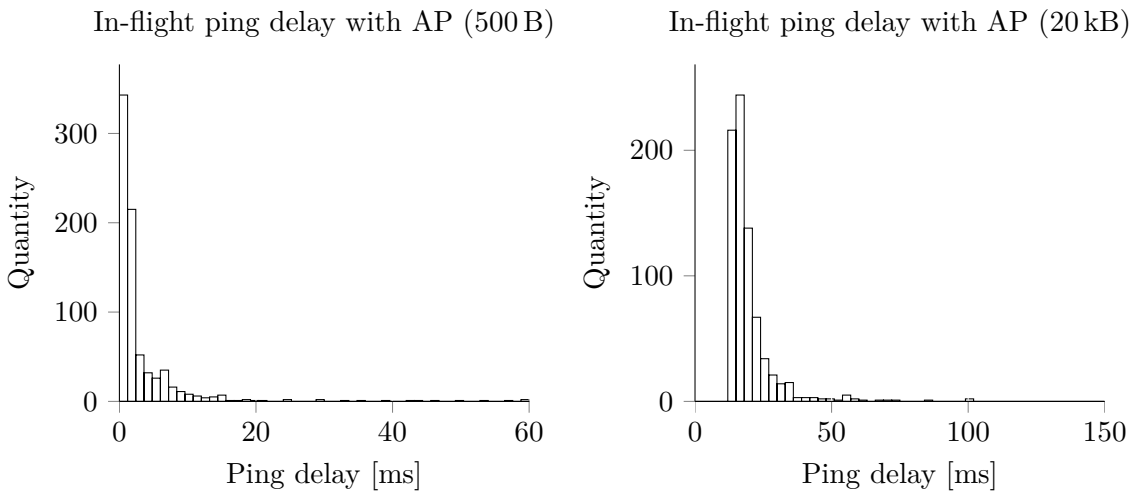


Figure 3.7.: In-flight ping delay with an AP. One can observe that there is no notable difference between the static and in-flight ping delay

the AR.Drone towards its desired location $p = (\hat{x}, \hat{y}, \hat{z}, \hat{\psi})^T \in \mathbb{R}^4$.

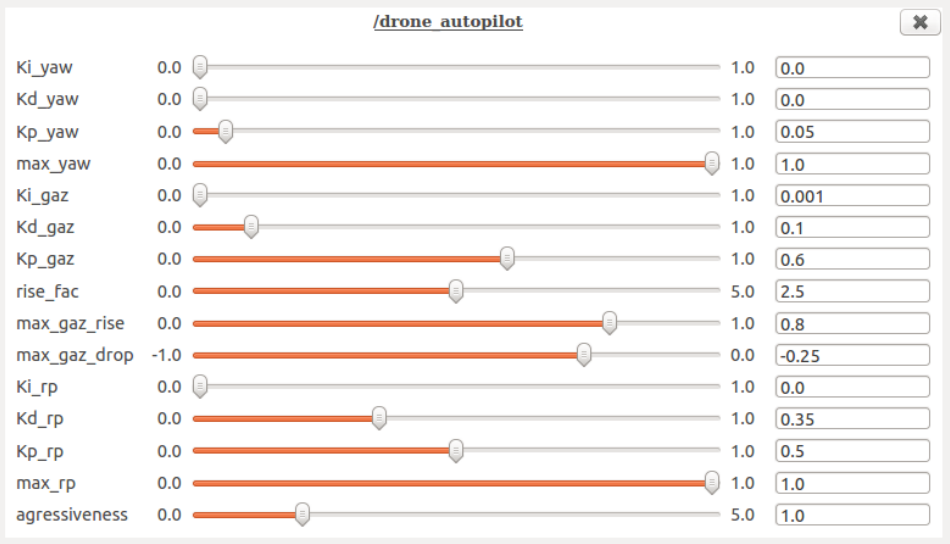


Figure 3.8.: Dynamically reconfigurable parameters of the autopilot from the `tum_ardrone` package

Control signals are sent to the drone at 100 Hz. These signals define each the desired roll, pitch, yaw rotational speed and vertical velocity: $\left(\bar{\Phi}, \bar{\Theta}, \bar{\dot{\Psi}}, \bar{\dot{z}}\right)$. The control signals are fractions of the maximum values that are imposed by `tum_ardrone`. These maximum values are 18° for roll and pitch, $2\,\mathrm{m\,s^{-1}}$ for vertical speed and $90\,^\circ\,\mathrm{s^{-1}}$ for yaw rotational speed.

Controller gains were determined experimentally using an external motion capture system (more information in [24]) and can be dynamically reconfigured (Figure 3.8). The resulting

optimal PID control gains are:

$$\begin{pmatrix} \bar{\Phi} \\ \bar{\Theta} \\ \dot{\bar{\Psi}} \\ \bar{\dot{z}} \end{pmatrix} = \begin{pmatrix} R(\Psi) \begin{bmatrix} 0.5(\hat{x}-x)+0.32\dot{x}+0 \\ 0.5(\hat{y}-y)+0.32\dot{y}+0 \end{bmatrix} \\ 0.02(\hat{\Psi}-\Psi)+0+0 \\ 0.6(\hat{z}-z)+0.2\dot{z}+0.01\int(\hat{z}-z) \end{pmatrix} \quad \text{with } R(\Psi) \text{ a planar rotation by } \Psi \tag{3.14}$$

It can be noted that the the vertical velocity controller is the only controller with an integral component and that the yaw rotational speed has a purely proportional controller.

In practice, only the aggressiveness parameter is reduced. A higher value for the aggressiveness caused the quadcopter to loose stability due to an overshoot of the control commands in pitch and roll. A value of 0.5 has proven to work fine.

Some remarks regarding the control of the drone:

- The parameters that are shown in Figure 3.8 only affect the autopilot. Manual control with a keyboard or joystick make use of the default control parameters of `ardrone_autonomy`,
- `ardrone_autonomy` should be launched with the launch file that is provided by `tum_ardrone`. Changing parameters of `ardrone_autonomy` such as the maximum vertical velocity or maximum yaw rotational speed will break the autopilot functionality of `tum_ardrone`,
- Depending on the last settings that were used on the Parrot AR.Drone, it is possible that the maximum vertical height should be manually added in the launch file that starts `ardrone_autonomy`.

4. Monocular SLAM algorithms

Different monocular SLAM algorithms exist nowadays. One of the technological developments that increased the worldwide interest into this field is undeniably the rise of small robots such as the AR.Drone. These SLAM algorithms can be subdivided into two major categories. The first category is based on image features while the second category is based on a direct approach that uses pixel intensities as the principal source of information. In what follows a general definition of pose will be given followed by an explanation of the SLAM principles and a more in depth description of the used algorithms.

4.1. Pose

Nowadays different SLAM algorithms are based on the successful approach of probabilistic robotics. Probabilistic robotics takes into account the uncertainty of calculations and observations and propagates them through the system. To make such calculations feasible appropriate formulations of states and other variables are needed. Within SLAM the position and the orientation of an object needs to be described. The two combined form the pose of an object. Pose-algebra formalises these poses and makes it possible to do calculations [1, 10].

To determine the position and orientation of an object different formulations exist but the ones most used are rigid body transforms or the combination of a position vector and a quaternion. A 3D body transform $\mathbf{G} \in SE(3)$ is defined by [1] as

$$\mathbf{G} = \begin{pmatrix} \mathbf{R} & t \\ 0 & 1 \end{pmatrix} \text{ with } \mathbf{R} \in SO(3) \text{ and } t \in \mathbb{R}^3. \tag{4.1}$$

These transforms are thus formulated in $SE(3)$ or a special euclidean group of order three. The $SO(3)$ in Equation (4.1) denotes a special orthogonal group (SO) of order three and is in practice given by a 3×3 matrix that respects the properties of a SO. This transform can also implicitly be used to indicate the pose of an object in the considered absolute world. In this case it is the transform to the *world frame*.

A SLAM system receives continuously new data that may add new information on existing estimated poses. Therefore in most slam systems the poses are interconnected in a graph, the so called pose-graph [1, 3, 4, 29]. Different techniques exist to optimize these graphs and to achieve loop closure. In most systems a minimal representation of the rigid body transform is used by performing some kind of mapping.

Monocular SLAM has difficulties with changing scale of the perceived scene. The implications are twofold. Firstly estimating absolute distance is inaccurate and secondly scale tends to drift. To overcome the second problem the factor scale is added to the pose. This resulting

formulation is called a similarity transformation (SIM). A 3D similarity transformation, $\mathbf{S} \in SIM(3)$, in defined in [1] by

$$\mathbf{S} = \begin{pmatrix} s\mathbf{R} & t \\ 0 & 1 \end{pmatrix} \text{ with } \mathbf{R} \in SO(3), t \in \mathbb{R}^3 \text{ and } s \in \mathbb{R}^+.$$

4.2. Principles behind SLAM

Before covering the SLAM algorithms separately, the global differences between them should be explained. Basically, two different families exist within the world of SLAM algorithms.

The first category is based on features and uses Bundle adjustment (BA) to optimise the camera localization and create a sparse geometrical reconstruction of the scene from a set of images[30].

Bundle adjustment is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter (camera pose and/or calibration) estimates. Optimal means that the parameter estimates are found by minimizing some cost function that quantifies the model fitting error, and jointly that the solution is simultaneously optimal with respect to both structure and camera variations[30].

Example algorithms in this category are PTAM and ORB-SLAM. Feature-based methods are able to match features from a wide baseline, thanks to the viewpoint invariance of the features. In the context of SfM, [31] pointed out the benefits of feature-based against direct methods. [4] confirmed this statements with experimental tests.

The second category makes use of direct pixel intensity information to create a semi-dense depthmaps. Because of the latter, this category may be more useful for other tasks than just camera localization. However, these systems have their own limitations. The most important limitation might be the assumption of the surface reflectance model that is used. In order to reduce depth uncertainty, wide baseline observations are required. However, direct SLAM methods generally match pixels from a narrow baseline because otherwise the reflectance model would be violated and many erroneous correspondences would appear. Another disadvantage of direct methods is the fact that they are general very computational demanding. This affects the way in which the 3D point cloud is generated. For example the DTAM package [32] incrementally expands a single map and LSD-SLAM reduces the map optimization to a pose graph optimization, discarding all sensor measurements.

As it is stated in [4], the future of Monocular SLAM should incorporate the best of both approaches. However, because such a framework does not exist yet, this thesis chose to use two different SLAM algorithms for different purposes. A feature-based algorithm, PTAM, is used for an improved state estimation and pose tracking and a direct method, LSD-SLAM, provides the 3D pointcloud for the 3D detection algorithms.

4.3. LSD-SLAM

The 3D information that the authors use in this thesis, is derived from the Large-Scale Direct Monocular SLAM package. Although the algorithm is readily available for ROS, multiple modifications are made and additional features are implemented in order to integrate the algorithm in this thesis.

4.3.1. Description

One of the latest SLAM algorithms that belongs to the direct category is LSD-SLAM (published in 2014). The algorithm was developed by Technische Universität München (TUM), [1, 24, 33] and excels in resource usage as it functions directly on smart-phones[34].

The algorithm works with a direct approach and maps to subsequent images by matching pixel intensities. As described in [1] a classic variational formulation requires a lot of computing power. This kind of computing power is nowadays only found in state of the art graphics processing units (GPUs). Therefore they choose to apply a semi-dense filtering. The drawback is that the resulting depth-images are also semi-dense and processing such images is not always easy because some algorithms demand continuity in the data. A sample run containing a staircase is shown in Figure 4.1.

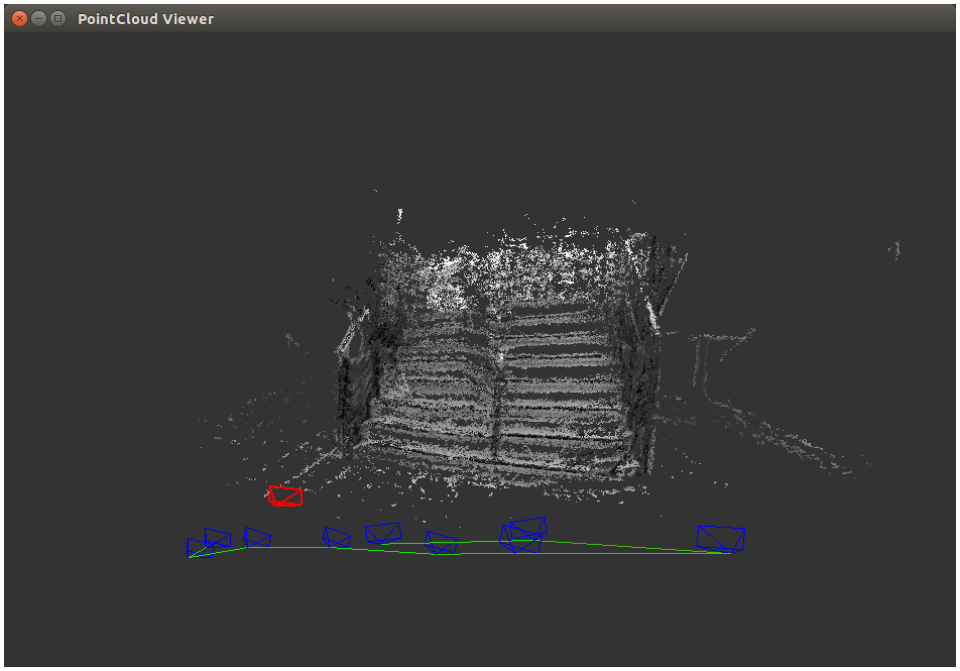


Figure 4.1.: LSD-SLAM demo on staircase

LSD-SLAM relies heavily on the notion of frames. A frame is a pose that may be associated with a depth image. Keyframes (KFs) are associated with a depth image that is built up by the algorithm by tracking camera images and their estimated poses, $\xi \in SE(3)$, and incrementally refining the result. Liveframes (LFs) only have a pose. The poses in both the KFs and LFs are estimated by incrementally optimising the photogrammetric error between an image and the current or last KF. An overview of the algorithm is shown in Figure 4.2.

The map optimisation in Figure 4.2 uses pose-graph optimisation based on Lie-algebra implemented in the Sophus library. A simple explanation of Lie algebra is not easy but the principle is understandable. A graph of discrete poses formulated in a Lie algebra makes the graph a continues structure usable for optimisation techniques. The special euclidean group (SE) of order three can be mapped to a Lie algebra resulting in a group denoted by $\mathfrak{se}(3)$ that respects the laws within a Lie algebra and is in practice a minimal representation. This minimal representation can be written as $\mathfrak{se}(3) \in \mathbb{R}^6$ and is used as such in the implementation to copy the parameters from one similarity transform instance to another.

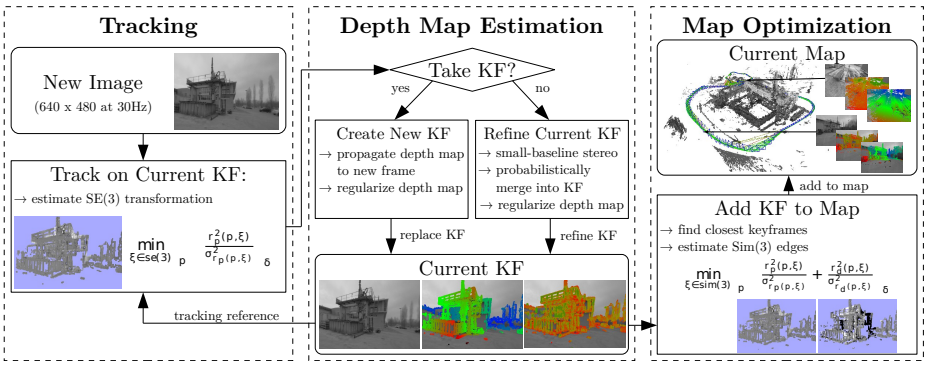


Figure 4.2.: Overview of the LSD-SLAM algorithm [1]

The pose in a Lie-algebra, $\xi \in \mathfrak{se}(3)$, is mapped to $SE(3)$ by the exponential map $\mathbf{G} = \exp_{\mathfrak{se}(3)}(\xi)$. The inverse mapping is done by $\xi = \log_{SE(3)}(\mathbf{G})$. The transformation moving a point from frame i to frame j is written as ξ_{ji} and pose concatenation is done using the concatenation operator $\circ$: $\mathfrak{se}(3) \circ \mathfrak{se}(3) \to \mathfrak{se}(3)$ defined in [1] as

$$\xi_{ki} := \xi_{kj} \circ \xi_{ji} := \log_{SE(3)} \left(\exp_{\mathfrak{se}(3)}(\xi_{kj}) \cdot (\exp_{\mathfrak{se}(3)}(\xi_{ji}) \right).$$

The former pose-graph optimisation does not allow for large loop closure. Therefore LSD-SLAM has an extra feature based SLAM on board, namely openFABMAP [29]. This framework shows similarities with PTAM and ORB-SLAM, both discussed in the following sections.

The algorithm, like many published, has a major drawback when combined with ROS, the complete point-cloud can not be sent. This is because the TCP back-end from ROS would be saturated by the continuous updates provide by the pose-graph optimisation. This is partly solved in LSD-SLAM by sending the KFs and afterwards only updates on the pose of the KFs. There is to the knowledge of the authors not a single point-cloud processing module currently available that can process this formulation of point-cloud data. As discussed in section 5.3 a module that can handle this formulation is created in this thesis.

A second problem is that LSD-SLAM can lose track of the environment. This is a problem because the program then needs to be restarted. The problem is further induced by using a camera that is actually not recommended. The authors of LSD-SLAM recommend a high framerate, at least 30 FPS, global shutter camera ([34]) while the camera on the Parrot Ar.Drone is a low framerate, rolling shutter model as shown in Table 1.1 in section 1.1. This is counteracted by damping the controls of the drone and make it move slower.

A good calibration and image rectification is also paramount to obtain the best results possible. The calibration process is described in section 2.3.3 and the influence of the rectification is visualized in figure 4.3. To start LSD-SLAM with the rectified image stream, the following commands should be used:

```
# Create the /ardrone/front/image_rect topic
ROS_NAMESPACE=/ardrone/front/ rosrun image_proc image_proc
# Start LSD-SLAM
rosrun lsd_slam_core live_slam image:=/ardrone/front/image_rect
    camera_info:=/ardrone/front/camera_info
```

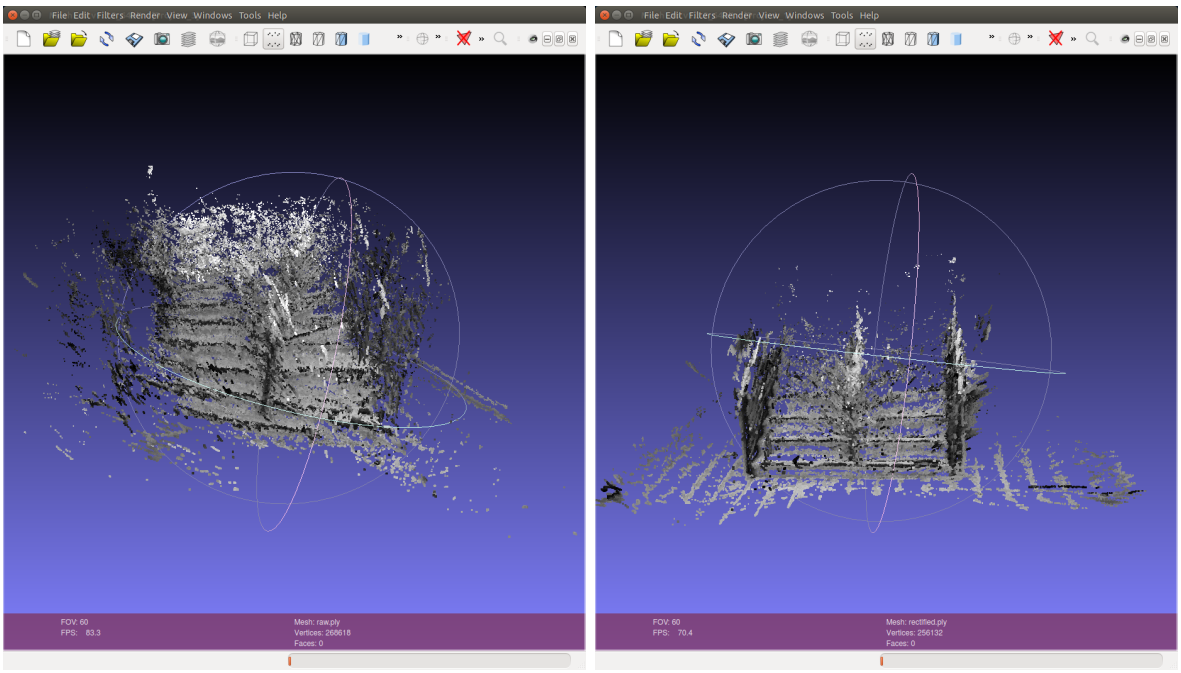


Figure 4.3.: Influence of image rectification on the performance of LSD-SLAM

4.3.2. Parameter tuning

The parameters used by LSD-SLAM can dynamically be (re)configured via ROS. This is because ROS provides a callback architecture for configuration messages. These messages can be generated via a graphical user interface (GUI) using `rqt`. The configuration dialogue for LSD-SLAM can be found in Figure 4.4.

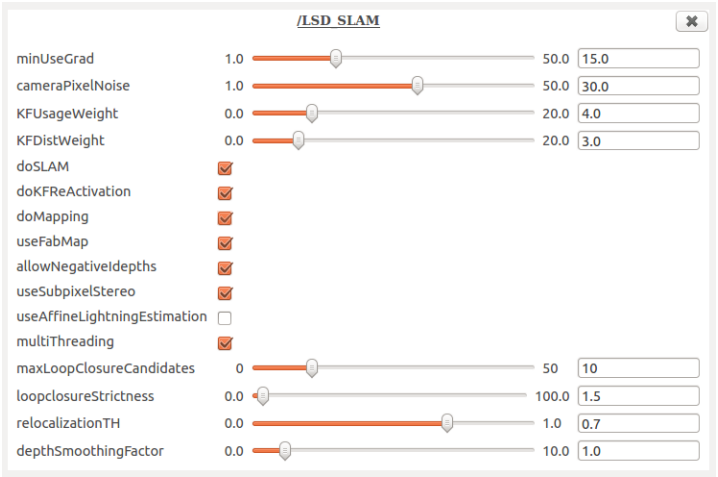


Figure 4.4.: Dynamically reconfigurable parameters of LSD-SLAM

To optimise LSD-SLAM for this thesis three parameters were changed as suggested by the readme of the `LSD_SLAM` package: *minUseGrad*, *cameraPixelNoise*, and *depthSmoothingFactor*. Firstly, optimising was done by determining maximum and minimum working values per parameter. Secondly, these parameters are set to a central value based on the minimum

and maximum values of the previous step. To test[1] and optimize the new configuration of parameters, these two steps are repeated four times.

The *cameraPixelNoise* indicates to LSD-SLAM how noisy the images are produced by the camera. LSD-SLAM can better estimate the error of the pose between two subsequent frames. In Figure 4.5, the influence of the *cameraPixelNoise* parameter is illustrated.

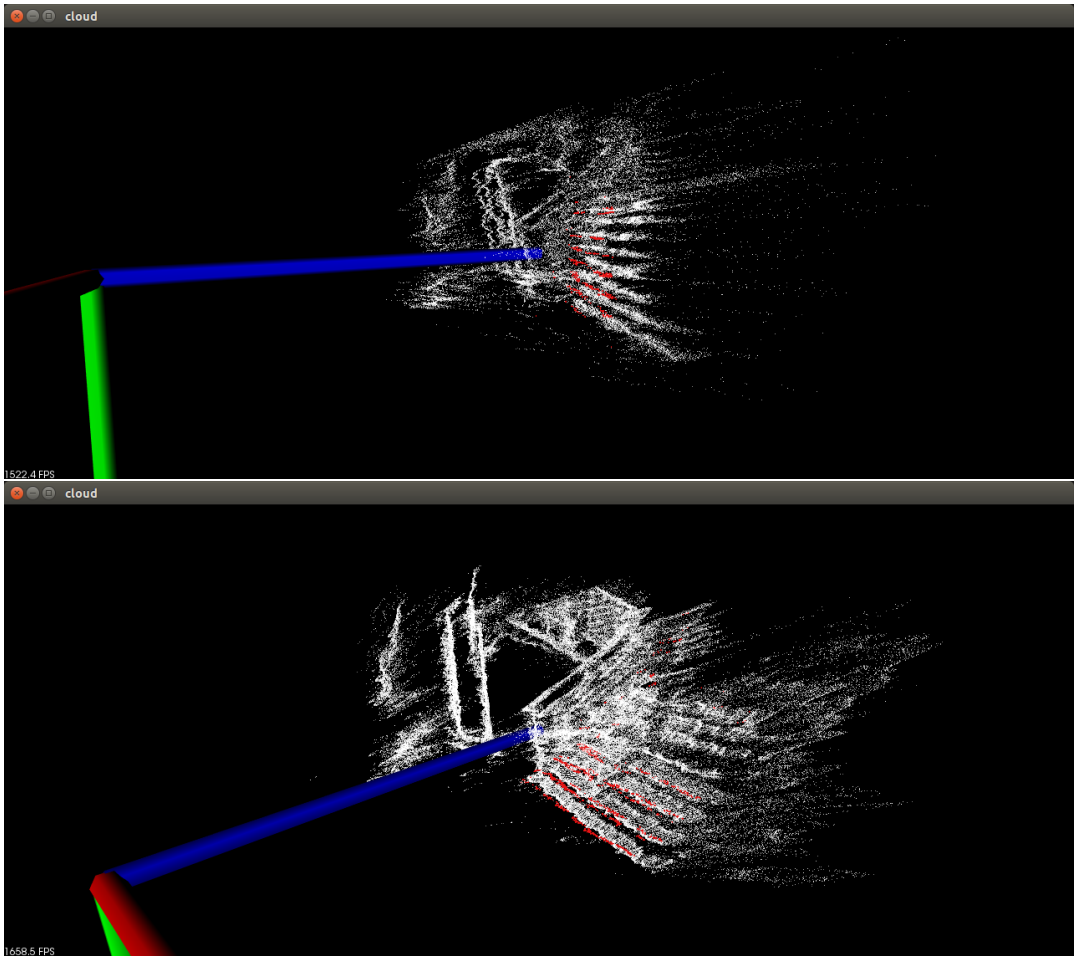


Figure 4.5.: Influence of the *cameraPixelNoise* parameter on LSD-SLAM. The point cloud on top is generated with a low value (4) for *cameraPixelNoise* while the picture on the bottom has a higher value (30)

The *minUseGrad* parameter defines the minimum amount of difference between the pixels that should be taken into account. The *depthSmoothingFactor* also influences the quality of the point cloud a little bit but not enough to make a good decision about its value. Therefore only the first two parameters are changed in this thesis. The new values are set to 15 and 30 for respectively the *minUseGrad* and the *cameraPixelNoise* parameters.

4.3.3. Modifications

To be able to use LSD-SLAM with the developed detection software, some modifications are made to the LSD-SLAM package. These can be found in section B.5.

[1] The evaluation of the parameters is done manually and is based on the visualization of the generated point-cloud

- The debug window is removed together with some functionalities that make use of OpenCV 2.x. This because the developers chose to use OpenCV 3 because of its performance improvement and some new functionalities.
- An automatic re-initialization after "TRACKING LOST" is also added after a short delay.
- A reset service is added (**resetlsd**) to flush and restart LSD-SLAM

4.4. PTAM

Because of the obligation to use the Parrot AR.Drone 2.0, the choice to use `tum_ardrone` as an EKF was obvious. Consequently, PTAM is used to track the drone's camera movements. The way PTAM is implemented in `tum_ardrone` is not changed in this thesis.

PTAM is one of the first ground braking algorithms that boosted the development of mobile near real time SLAM algorithms [3]. It is a feature based SLAM algorithm that does not require prior knowledge of the environment. PTAM uses two separate threads: the first thread tracks the camera movements while the second thread performs the mapping. By splitting these two operations, computationally expensive batch optimisation techniques (bundle adjustment) can be used. The result is a detailed map with a lot of landmarks that can be tracked at frame-rate.

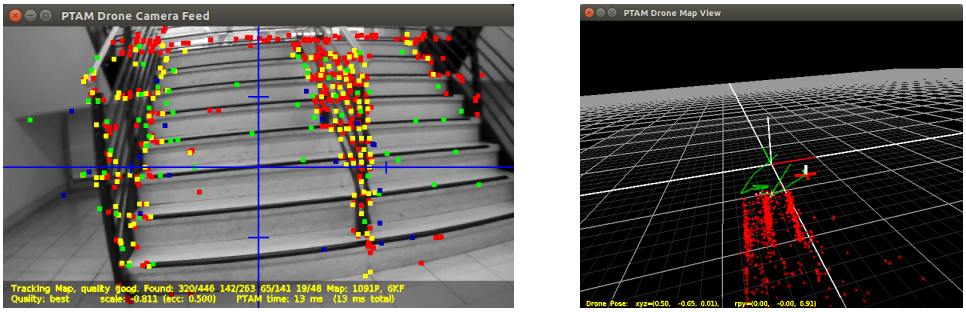


Figure 4.6.: PTAM demo on staircase on the left, with the generated pointcloud and estimated position on the right

PTAM can be summarized in the following points:

- Tracking and mapping run in two parallel threads
- Mapping is based on keyframes which are processed using bundle adjustment
- Map initialization is done manualy from a stereo pair. This procedure normally requires human intervention but an attempt[2] is made to automate this in `tum_ardrone`.
- A relatively large number (within the category of feature-based SLAM) of points are mapped to a pointcloud

[2]The word attempt is used because manual initialization of PTAM is still faster and works better.

When using PTAM, one has to be aware of the shortcomings of the algorithm such as the difficult initialization method, the lack of a loop closing mechanism, the low invariance to viewpoint of its re-localization method and its restriction to small scenes. However, beside those shortcomings PTAM remains to this day one of the best performing BA-based SLAM algorithms.

More information about the initialization, tracking and mapping can be found in [3].

4.5. ORB-SLAM

In the quest to use the best available SLAM algorithms for this thesis, ORB-SLAM was taken into consideration to use as a base algorithm for the EKF. But because of the complexity and the fact that there was already an EKF with PTAM readily available for the Parrot AR.Drone, ORB-SLAM is not used in this thesis. Nonetheless it is interesting to mention that this algorithm has proven its superiority over PTAM, especially because of the automatic initialization and loop closure mechanism. Consequently, it may be a good idea to implement it for future work.

An overview of the working and advantages of ORB-SLAM are provided in this section.

ORB-SLAM is a relatively new algorithm that is first published in February 2015. As the paper [4] states, it is a Monocular SLAM system that operates in real time, in small and large, indoor and outdoor environments, with the capability of wide baseline loop closing and relocalization, and including full automatic initialization. The survival of the fittest approach to select the points and keyframes of the reconstruction generates a compact and trackable map that only grows if the scene content changes, enhancing lifelong operation. One can say

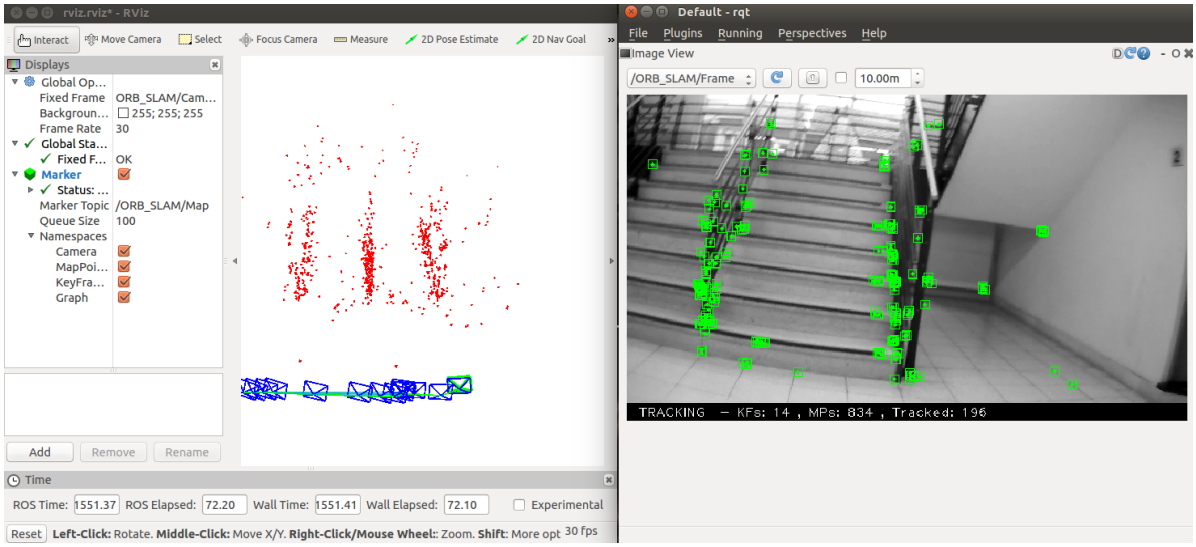


Figure 4.7.: ORB-SLAM demo on staircase

that it is a successor to PTAM. Because it is based on the same principle: BA. The results that are published on their website[3] are quite impressive.

ORB-SLAM incorporates, as shown in Figure 4.8, three threads that run in parallel: a tracking thread, local mapping thread and a loop closing thread. The choice of using ORB features

[3] http://webdiis.unizar.es/~raulmur/orbslam/

is not a coincidence because these features are used in the three threads. ORB features are oriented multi-scale FAST corners with a 256 bits descriptor associated. They are fast to compute and match, while being invariant to rotation and scale (within a certain range). These features are also invariant to the viewpoint and improves the performance of BA.

ORB-SLAM has the following properties:

- Use of ORB features versus FAST corners in PTAM
- Real time operation in large environments
- Loop closing mechanism based on the optimization of a pose graph
- Real time camera relocalization with high invariance to viewpoint
- New automatic and robust initialization procedure that works on planar and non-planar scenes.
- Survival of the fittest approach to map point and keyframe selection

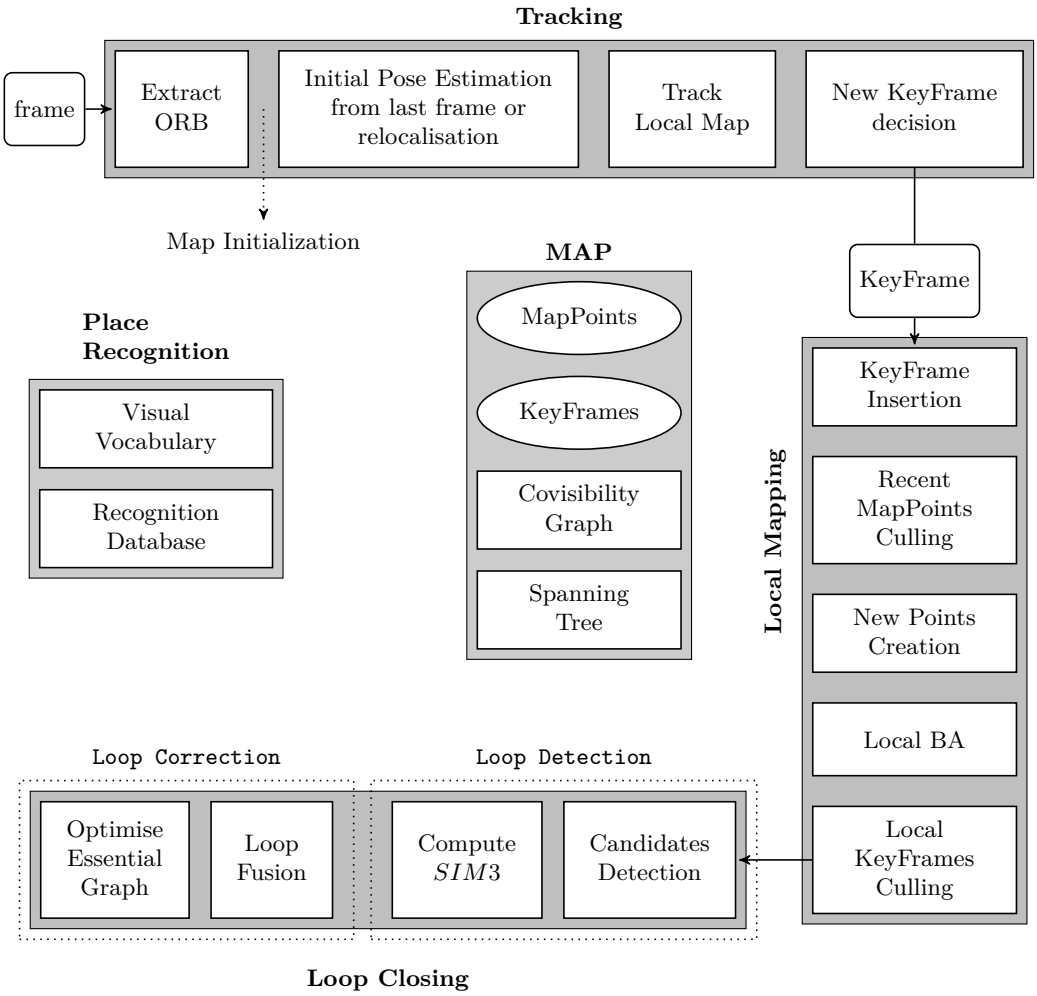


Figure 4.8.: ORB-SLAM system overview, showing all the steps performed by the tracking, local mapping and loop closing threads. The main components of the place recognition module and the map are also shown.

4.6. Scale estimation for monocular SLAM

As mentioned in [1], monocular SLAM is inherently scale-ambivalent because the absolute scale of the environment is not directly observable. Over long trajectories this leads to scale-drift. Threshold-based outlier rejection can also drop too many points if the scale is badly estimated. Different approaches exist to solve this scale-issue.

LSD-SLAM solves this in a newly developed direct, scale-drift aware image alignment technique by using the inherent correlation between scene depth and tracking accuracy. This mechanism is further elaborated in [1].

The scale estimation of LSD-SLAM is used to compose the 3D pointcloud and will therefore be used in the detection algorithm. However, the EKF makes use of a different estimated scale that combines the output of PTAM with sensor data of the quadcopter. The method to derive this scale is explained in the remainder of this section.

The developers of the `tum_ardrone` created their own method for the scale $\lambda \in \mathbb{R}^+$ estimation (more information can be found in Section IV-A of [26] or Section 4 of [24]). A closed-form solution determines the scale λ by using a derived maximum likelihood estimator. The assumption that is made is the presence of noisy measurements of absolute distances or velocities. This is the case for metric sensor such as the air pressure sensor and altimeter.

4.6.1. Problem formulation

The quadcopter regularly performs two different types of measurements (x_i, y_i):

1. $x_i \in \mathbb{R}^d$ - the d-dimensional distance according to PTAM
2. $y_i \in \mathbb{R}^d$ - the absolute distance/velocity provided by the metric sensors.

In the case of the altimeter, d equals one. Both x_i and y_i measure movements of the drone but with different scales. The relation between the two types of data can be approximated by a linear equation: $x_i \approx \lambda y_i$.

Under the assumption that the measurement noise is isotropic and Gaussian white noise, the following equations can be written:

$$\begin{aligned} x_i &\sim \mathcal{N}(\lambda\mu_i, \sigma_x^2 I_{d\times d}) \\ y_i &\sim \mathcal{N}(\mu_i, \sigma_y^2 I_{d\times d}) \end{aligned} \tag{4.2}$$

with

- $\mu_i \in \mathbb{R}^d$ true distances
- $\sigma_x^2, \sigma_y^2 \in \mathbb{R}^+$ variances of the measurement errors

Assume that a set of samples is gathered $\{(x_1, y_1), ..., (x_n, y_n)\}$ before we go to the next step.

4.6.2. Naïve estimation strategies

Possible methods to naively estimate λ are the arithmetic average, geometric average or median of $\frac{||x_i||}{||y_i||}$. Unfortunately these methods fail to estimate λ accurately. This is illustrated in Equation (4.3) with two sample pairs: $(x_1, y_1) = (1, 0.5)$ and $(x_2, y_2) = (1, 1.5)$. y_1 and y_2 deviate symmetrically from the true value and x_i has no measurement noise.

$$\left.\begin{array}{ll} \text{Geometric mean:} & \left(\prod_{i=1}^{n} \frac{||x_i||}{||y_i||}\right)^{\frac{1}{n}} \\ \text{Arithmetic mean:} & \frac{1}{n}\left(\sum_{i=1}^{n} \frac{||x_i||}{||y_i||}\right) \end{array}\right\} \Rightarrow \text{Example:} \begin{array}{l} \left(\frac{1}{0.5} \cdot \frac{1}{1.5}\right)^{\frac{1}{2}} \approx 1.15 \\ \frac{1}{2}\left(\frac{1}{0.5} + \frac{1}{1.5}\right) \approx 1.3 \end{array} \tag{4.3}$$

An alternative is to take the sum of squared differences (SSD) between the absolute measurements and the re-scaled measurements:

$$\lambda_y^* := \arg\min_{\lambda} \sum_{i=1}^{n} ||x_i - \lambda y_i||^2 = \frac{\sum_i x_i^T y_i}{\sum_i y_i^T y_i} \tag{4.4}$$

$$\lambda_x^* := \left(\arg\min_{\lambda} \sum_{i=1}^{n} ||x_i - \lambda y_i||^2\right)^{-1} = \frac{\sum_i x_i^T x_i}{\sum_i x_i^T y_i} \tag{4.5}$$

All the methods in this section do not converge to the correct value of λ for $n \to \infty$, this is visualized in Figure 4.9.

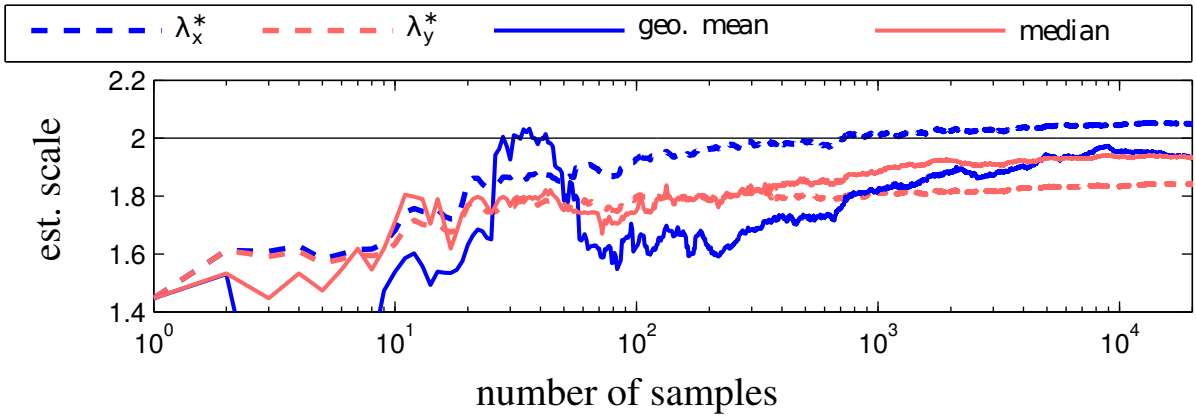


Figure 4.9.: Scale estimation in function of the number of samples available with parameters: $\lambda = 2$, $\sigma_x = 0.3$, $\sigma_y = 0.3$ and $\mu_i \sim \mathcal{N}(0, 1)$. There is clearly no convergence. Figure taken from [24]

4.6.3. Maximum likelihood estimator

The idea behind an ML estimator is to choose an unknown value such that the probability to observe the measured data is maximised. Typically, the negative log-likelihood is minimised:

$$\mathcal{L}(\mu_1, ..., \mu_n, \lambda) \propto \frac{1}{2} \sum_{i=1}^{n} \left(\frac{||x_i - \lambda\mu_i||^2}{\sigma_x^2} + \frac{||y_i - \mu_i||^2}{\sigma_y^2}\right) \tag{4.6}$$

After minimising over μ_1, ..., μ_n and then over λ, Equation (4.6) has a global minimum in:

$$\mu_i^* = \frac{\lambda^* \sigma_y^2 x_i + \sigma_x^2 y_i}{\lambda^{*2} \sigma_y^2 + \sigma_x^2}$$
$$\lambda^* = \frac{s_{xx} - s_{yy} + sign(s_{xx})\sqrt{(s_{xx} - s_{yy})^2 + 4s_{xy}^2}}{2\sigma_x^{-1}\sigma_y s_{xy}} \quad \text{with} \quad \begin{aligned} s_{xx} &= \sigma_y^2 \sum_{i=1}^{n} x_i^T x_i \\ s_{yy} &= \sigma_x^2 \sum_{i=1}^{n} y_i^T y_i \\ s_{xy} &= \sigma_y \sigma_x \sum_{i=1}^{n} x_i^T y_i \end{aligned} \tag{4.7}$$

This solution gives an ML estimator of λ, assuming that σ_x and σ_y are known and that $s_{xy} > 0$ (which holds for a large sample set) and $\lambda > 0$. Three interesting properties appear from this formula:

1. $\lambda* \in \left[\lambda_x^*, \lambda_y^*\right]$
2. For $\sigma_x^2 \to 0 : \lambda* \to \lambda_x^*$
3. For $\sigma_y^2 \to 0 : \lambda* \to \lambda_y^*$

In other words, $\lambda*$ interpolates between the two extreme cases when one of the measurements sources is noise-free. More information about the derivation can be found in [35].

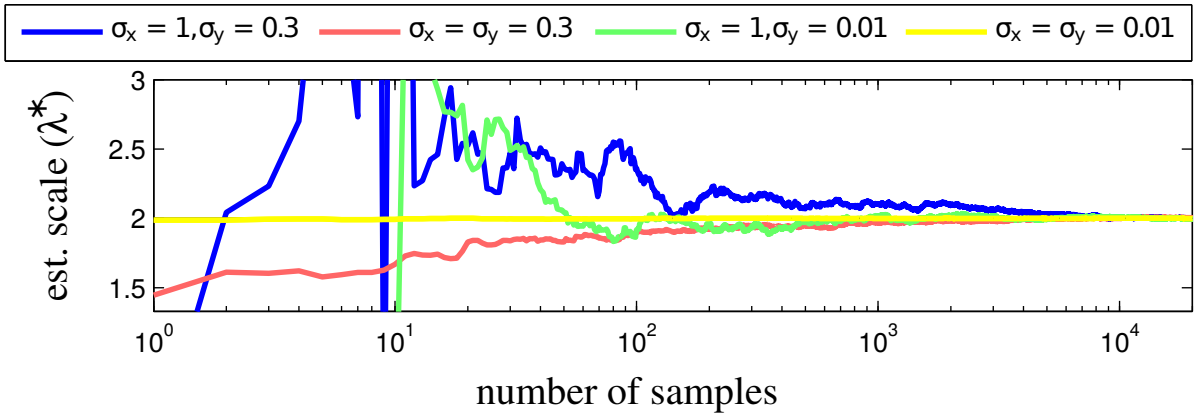


Figure 4.10.: Scale estimation with different noise levels using a maximum likelihood estimator. The parameters are the same as in Figure 4.9. Figure taken from [24]

In order to calculate λ^*, a sample set should be generated and the measurement variances should be calculated.

4.6.4. Sample set

In subsection 4.6.1 under the title problem formulation, the position of a set of samples $\{(x_1, y_1), ..., (x_n, y_n)\}$ is assumed. This section explains how these sample pairs are obtained.

The input data is a metric sensor (e.g. pressure sensor) operating at 200 Hz and visual altitude measurements with a varying update speed. The pairs are created in two steps (illustrated in Figure 4.11):

1. At time t_i, a visual altitude measurement $a_v(t_i) \in \mathbb{R}$ is taken together with a corresponding metric altitude measurement $\bar{a}_m(t_i) \in \mathbb{R}$. Because the metric measurements are more frequently updated, an average value is taken. No values are used more than once to preserve statistical independence.

2. Over a timespan of k frames, the distance travelled is computed (Equation (4.8)). In practice, k is chosen between 30 and 60 but in theory it should be in function of the speed of the drone. If the speed is high, k can be chosen relatively small. However when the drone is moving very slowly, the value for k can be taken larger than 60.

$$\begin{aligned} x_i &= a_v(t_i) - a_v(t_{i-k}) \\ y_i &= \bar{a}_m(t_i) - \bar{a}_m(t_{i-k}) \end{aligned} \tag{4.8}$$

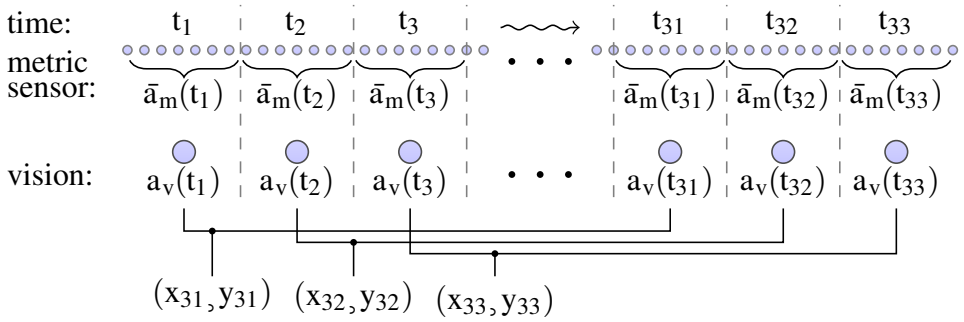


Figure 4.11.: Computation of the sample set for the estimation of $\lambda*$. Figure taken from [24]

4.6.5. Measurement variances

In order to calculate $\lambda*$ (in Equation (4.7)), σ_m^2 (metric sensor) and σ_v^2 (vision estimates) have to be known. These variances can be estimated from data under the following assumptions:

- Consecutive measurements are independent and statistically identical distributed,
- Measurements are taken at regular time intervals,
- The speed of the drone is constant over three consecutive measurements given by

$$\begin{aligned} a_v(t_{i-1}) &\sim \mathcal{N}(a-b, \sigma_v^2) \\ a_v(t_i) &\sim \mathcal{N}(a, \sigma_v^2) \\ a_v(t_{i+1}) &\sim \mathcal{N}(a+b, \sigma_v^2) \end{aligned} \quad \text{with} \quad \begin{cases} a & \text{true hight} \\ b & \text{true velocity} \end{cases} .$$

If the additivity property is used of the normal distribution, the following function is used to estimate σ_v^2:

$$(a_v(t_{i-1}) - 2a_v(t_i) + a_v(t_{i+1})) \sim \mathcal{N}(0, \sigma_v^2 + 4\sigma_v^2 + \sigma_v^2) \tag{4.9}$$

The formula for estimating σ_v^2 from a set of n measurements $\{a_v(t_1), ..., a_v(t_n)\}$:

$$\sigma_v^{2*} = \frac{1}{6}\frac{1}{n-3}\sum_{i=2}^{n-1}(a_v(t_{i-1}) - 2a_v(t_i) + a_v(t_{i+1}))^2 \tag{4.10}$$

The variance of the visual altitude measurements can then be given by $\sigma_x^2 = 2\sigma_v^2$. The calculation of σ_m^2 is analogous. An important note that is added to [24] emphasizes that the assumption of a constant pose estimation variance is questionable. Especially for a visual

SLAM system given the fact that its error may change in-flight. However, the order of accuracy of a visual SLAM system is several orders of magnitude lower than that of an altitude pressure sensor for example.

5. Staircase detection

5.1. Overview of the algorithms

Detection of objects in general and staircases has already been subjects of many papers like [36–40] . Most of them are using depth data to detect or improve detection while visually exploiting the parallel nature of the stairs [40] or using common object detection techniques [37] as described in subsection 5.2.4.

This paper is largely based on the implementation of the algorithm in [36] where 2D edge detection is combined with 3D curvature analysis. The depth images are not obtained directly but aggregated via LSD-SLAM (section 4.3) and put together by a wrapper designed by the authors that is discussed in section 5.3.

Next to 3D analysis and edge detection, visual object detection is applied on the images as described partly in [37]. Because the relatively long process times both processes are paralysed and synchronised. A general overview of how the wrapper integrates in ROS is given in Figure 5.1.

5.2. 2D image processing

Because processing and analysing in 3D is computational intensive and leave a lot to be desired, 2D image processing is used in many projects alike this thesis [36, 37]. This thesis uses edge detection and object detection. Firstly, a general overview of computer vision techniques used in similar applications is given. Secondly, both techniques and their implementation details will be elaborated in this section.

5.2.1. Overview

Computer vision is a fast evolving science and it is a popular topic. In the last decades several publications where made on analysing images and object detection. Pre-processing, filtering, and analysing images is a task that is nowadays doable in an automated way with computer vision techniques. Detection and extraction of well known specific objects is also feasible. Segmentation of unknown objects somehow works but detection and extraction of unknown or ill described objects is difficult.

A classic image processing algorithm consist in several stages that are generally sequential. A first step is filtering the incoming image. All sensors suffer from noise and so does a camera. Noise is mostly a disturbing factor in the calculations and must be removed. The general drawback of removing noise is that sharp edges are also blurred.

The second stage is feature detection and extraction. This is the calculation of derived values from the image and keeping the strongest ones. These features also need to be described in a

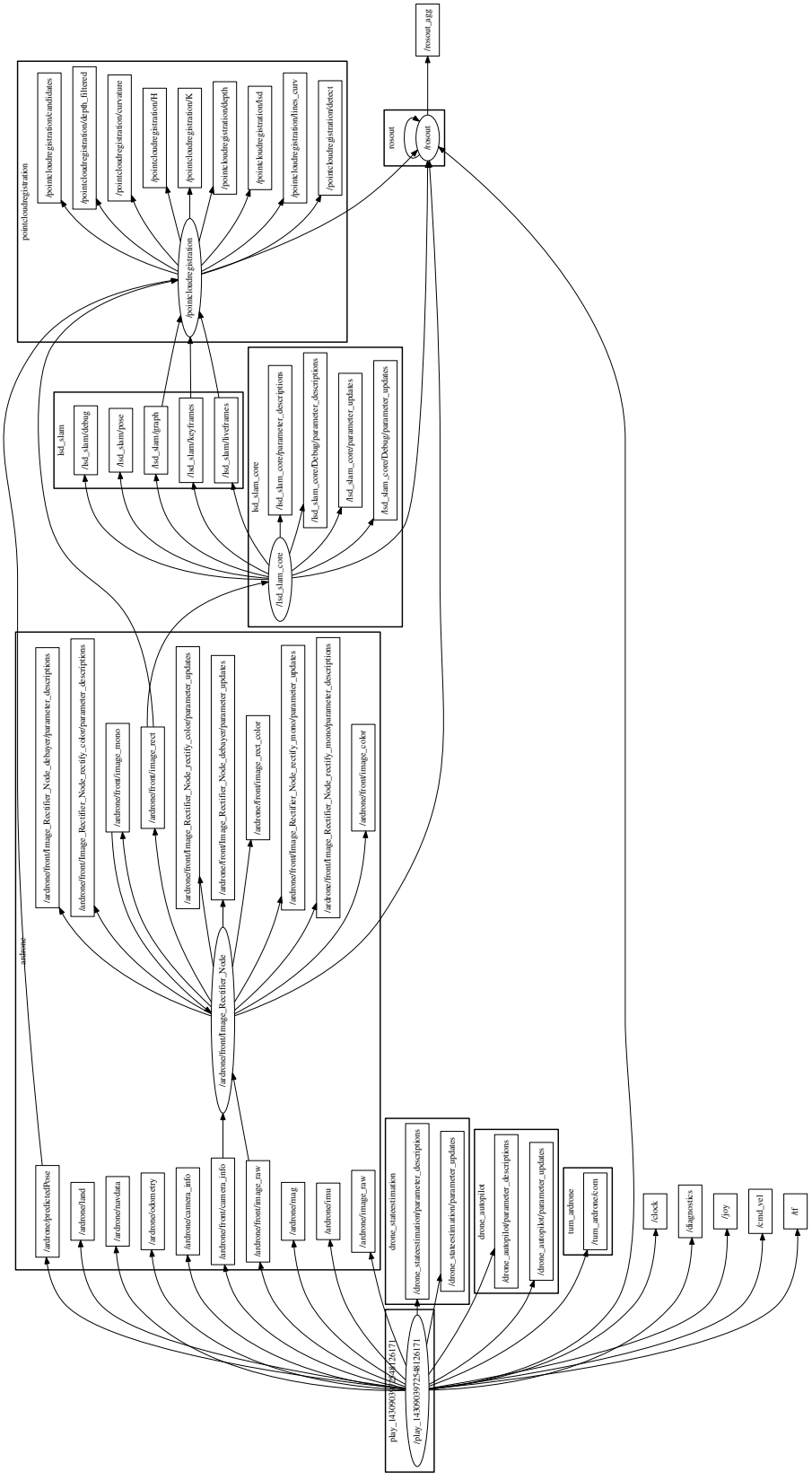


Figure 5.1.: Overview of the detection framework generated using `rqt-graph`. The node `/pointcloudregistration` is the wrapper developed in this thesis

general way so they can be used in a classification stage or clustering stage. The most popular features in recent works are the so called HAAR-like features based on the HAAR-wavelet [41] or derivatives of corner detectors and descriptors like scale-invariant feature transform (SIFT), Features from accelerated segment test (FAST) and Oriented FAST and Rotated BRIEF (ORB). A good overview of practical implementations and a brief explanation of the background theory can be found on [42].

The main advantage of HAAR-like features is that they are fast to implement using an image filter. They are also very suitable for multi-scale applications. The general disadvantage is that they are sensitive to rotation. This can also be advantageous as sometimes the orientation is known or important.

Corner detectors and descriptors have proved themselves useful in matching between subsequent images, an application is shown in Figure 5.2. An extension of these techniques is used in different SLAM systems discussed in chapter 4.

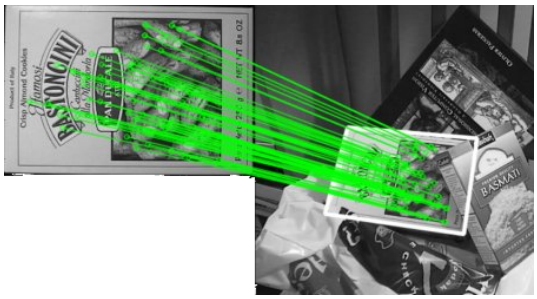


Figure 5.2.: Matching using SIFT [42]

5.2.2. Line detection

Classic method

Line detection has always been a topic of interest in computer vision as many man-made objects ar distinguishable by the appearance of straight lines. Line detection is only feature detection with the characteristics of the line a possible descriptor. Line detection is however not adequate for detecting complete objects and thus, in such applications lines need to be grouped or linked in a certain way to be meaningful. The further exploitation of the detected lines is given in subsection 5.2.3 and subsection 5.2.5.

A classic scheme to detect lines is the combination of an edge detector with a variant of the Hough transform. The major drawback of this scheme is that both algorithms need parameter tuning adapted to the scene and camera.

The automatic thresholding for the edge detector can be solved using a Canny edge detector combined with Otsu threshold as described in [43]. This method only determines the high threshold while the lower threshold is chosen relatively to the higher threshold. Nevertheless this method is promising and implemented in OpenCV via the threshold functions.

The Hough transform is a relatively old tool and a good description can be found in [44]. A maybe bigger problem than the parameter optimisation is that the method does not permit determining the begin point and the end point of the lines found. The method is also not that robust and it is computational intensive. The probabilistic Hough transform described in [45] addresses these problems and generally reduces runtime.

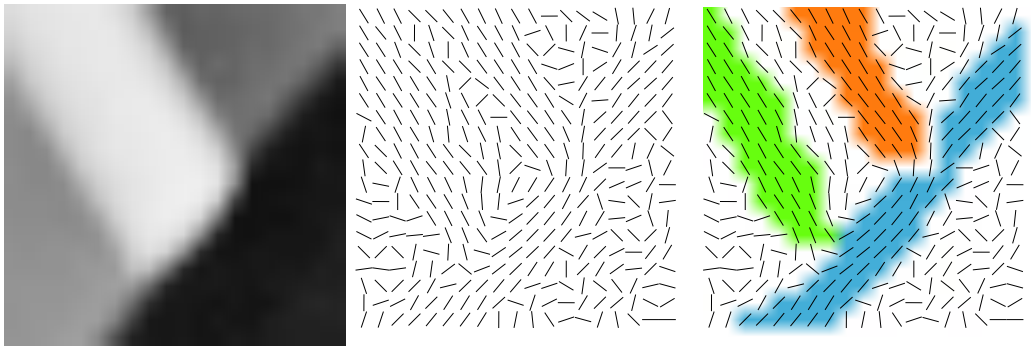
Line segment detector

The improvements take the major drawbacks away from the classic setup but the scheme does not support region growing. Therefore another method is used in this thesis, the line segment detector described in [46]. The method is designed to work on digital images without parameter tuning and the implementation in OpenCV provides the begin and the end points of the detected lines.

As the classic method lines are sought where the gradient of the grayscale image is large. To formalise this, level-lines that are perpendicular to the orientation of the gradient are introduced. Each point gets a unitary level-line that is under an angle, the level-line angle (LLA). The detector starts with determining the level-line field in an image by computing the LLA. Secondly the image is segmented in different support regions for lines each within a tolerance angle τ. The process is shown in Figure 5.3. With the support regions a rectangle is associated for further analysing. Derived from τ the initial precision p is defined as

$$p = \frac{\tau}{\pi}$$

Later in the algorithm τ is refined and the precision, p, of every detected line is given to the user as output.



(a) Image (b) level-line field (c) line support regions

Figure 5.3.: Line support regions,[46]

The Line Segment Detector (LSD) is based on a contrario validation approach where the algorithm tests if such rectangle could exist in noise and how much aligned points it would contain. From this notion the Number of False Alarms (NFA) is defined. This in the number of rectangles that have a sufficient number of aligned points to be as rare as the tested rectangle under the a contrario model. A rectangle is thus meaningful if the NFA is low. Within the algorithm a threshold $\varepsilon \in \mathbb{N}_0^+$ unknown to the end-user is determined and when a rectangle has NFA $< \varepsilon$ that rectangle is called ε-meaningful. This threshold is not meant to be adapted as stated in [46].

The rectangles found are then further optimised by reducing the width of the rectangle and the precision p. The rectangle with the smallest NFA is kept. An overview of the algorithm is given in Listing 5.1 and explained in the following paragraphs.

To avoid artefacts by compression and scaling of the digital image, the image is always scaled. The default value is $S = 80\,\%$. The scaling is done by Gaussian sub-sampling with a kernel size of $\sigma = \Sigma/S$. The value of Σ is set to 0.6 which is a balance between avoiding aliasing and blurring.

Listing 5.1: line segment detector algorithm [46]

```
1  input: An image I.
2  output: A list out of rectangles.
3  I_S ← ScaleImage(I, S, σ = Σ/S)
4  ( LLA, |∇I_S|, OrderedListPixels ) ← Gradient(I_S)
5  Status ← { USED,     pixels with |∇I_S| ≤ ρ
            { NOT USED, otherwise
6  foreach pixel P ∈ OrderedListPixels do
7          if Status(P) = NOT USED then
8                  region ← RegionGrow(P, τ )
9                  rect ← Rectangle(region)
10                 while AlignedPixelDensity(rect, τ ) < D do
11                         region ← CutRegion(region)
12                         rect ← Rectangle(region)
13                 end
14                 rect ← ImproveRectangle(rect)
15                 nfa ← NFA(rect)
16                 if nfa ≤ ε then
17                         Add rect → out
18                 end
19         end
20 end
```

The gradient, ∇I_S, of the scaled image is calculated by applying a 2×2 filter. Thresholding is done by ρ given in Equation (5.1). The threshold is dependant on the quantisation noise. The upper bound of the quantisation noise is given by q. The value of q should be $q = 1$ in a standard 8 bit image but is set to a more conservative value of $q = 2$ by [46].

$$\rho = \frac{q}{\sin \tau} \tag{5.1}$$

To create the regions, region growing is done upon a seed, a pixel from the ordered list of unused pixels. Pixels are added to the region if the LLA is within the tolerance τ. A rectangle that encloses this region is determined.

The region growing causes pixels to be added that are protrusions to the wanted line. To eliminate these protrusions the region is cut and the rectangle refitted. This process is repeated until the region fills up a sufficient part, D, of the rectangle.

To further enhance the result within the scope of this thesis only nearly horizontal lines are kept. To define the horizon, the roll angle that is estimated by the EKF is used. The result of this algorithm is shown in Figure 5.4.

5.2.3. Filter out parallel lines

Once the lines are withheld that fall within a detection rectangle from the object detector, only the parallel lines are kept. To do this, Random sample consensus (RANSAC) [47, 48] is

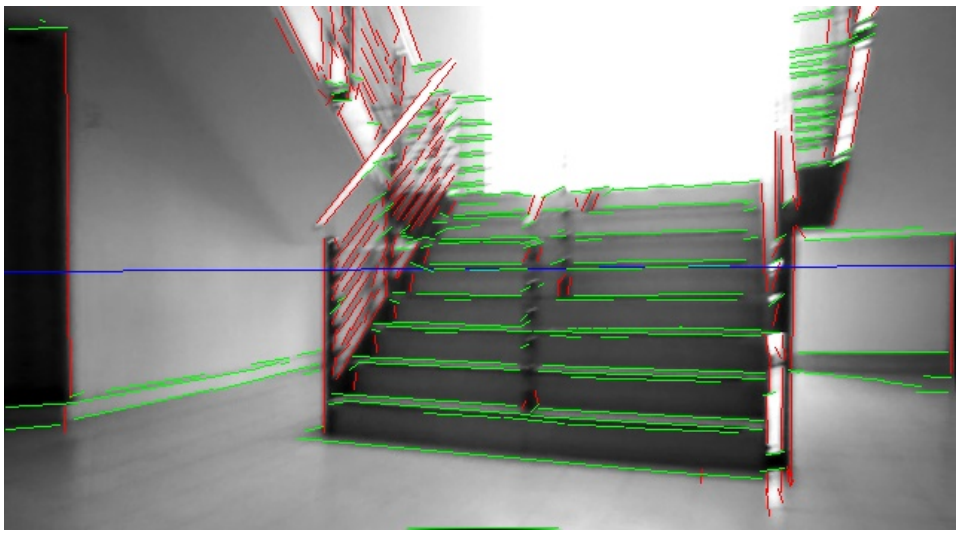


Figure 5.4.: LSD applied on environment with a staircase. The blue line is the estimated horizon, the green lines are within tolerance (8°) and the red lines were the detected by the LSD but are not within tolerance.

used to determine the model slope of the lines. By using RANSAC, the outliers are not taken into account to determine the slope of the detected lines that originate from the stairs.

RANSAC is a robust estimation technique that selects randomly a minimum amount of points to estimate the model parameters of the data. The algorithm is executed N times and can be broken down in five different steps per iteration:

1. Make a random selection of the minimum amount of points required to calculate the model parameters
2. Solve the parameters of the model
3. Determine the amount of points that satisfy the model (inliers) within tolerance ϵ.
4. If # inliers$> \tau$ with $\tau \in \mathbb{N}_0$ the threshold, the model parameters are re-estimated using all the inliers.
5. If this model is better then the previous best one, the previous best model is updated.

The model in the particular case of determining parallel lines is the slope and the MSE coupled to this slope. An iteration is than given by:

1. Select a line randomly
2. Calculate the slope of the line
3. Add lines to a test set if their slope is within tolerance $\epsilon \in \mathbb{R}$
4. If there are enough lines in this test set calculate the mean slope
5. Calculate the MSE with respect to the mean slope of the test set and if this MSE is lower than the MSE of the model update the model

N is calculated in Equation (5.2) and chosen high enough to ensure a high probability p (>0.99) that at a certain moment, a set of random samples does not include an outlier. Let $P\{\text{all points are inliers}\} = u$ and $P\{\text{observe outlier}\} = v = 1 - u$.

$$1 - p = (1 - u^m)^N \Leftrightarrow N = \frac{\log(1-p)}{\log(1-(1-v)^m)} \tag{5.2}$$

with

- $1 - v$ approximated by $\frac{\#\text{inliers}}{\#\text{points in data}}$
- m is the minimum required amount of points to calculate the model parameters
- $N \in \mathbb{N}_0$ the estimated number of iterations needed

This only works if the camera is looking directly to the stair. A fallback is provided using vanishing point detection. A vanishing point is the intersection of the projections of a set parallel lines. An example where vanishing point detection was forced is shown in Figure 5.5. The model in this case is the location of the vanishing point and the MSE coupled to this point. An iteration is now given by:

1. Select two lines randomly
2. Calculate the intersection of both lines
3. Add lines to a test set if one of their intersections with the random selected line lays close enough
4. If there are enough lines in this test set calculate the mean of the intersections of all the lines in the test set
5. Calculate the MSE with respect to the mean intersection of the set and if this MSE is lower than the MSE of the model update the model

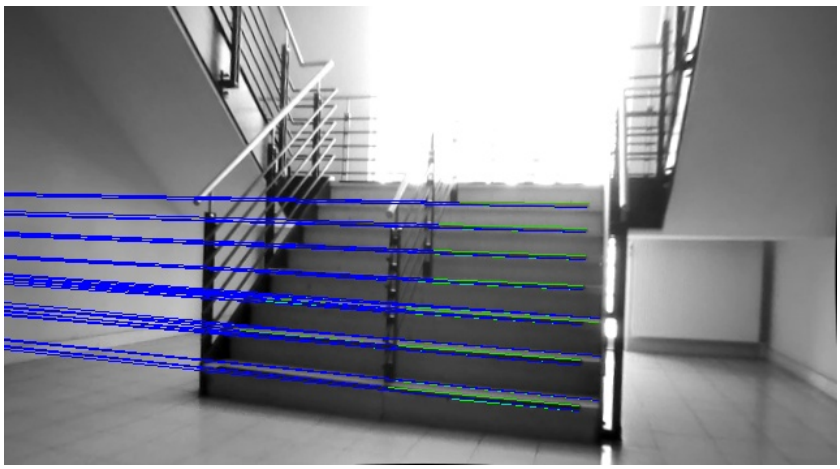


Figure 5.5.: Vanishing point detection

5.2.4. Object detection

Object detection can be done in several ways were some are more generally applicable than others. Object detection matches the extracted descriptors with known data and decides if they actually match. The criteria for matching can be an euclidean range in a hyperspace or binary like with support vector machine (SVM). Because the dimension of such a hyperspace can be large and mostly the dimension is reduced by using techniques like principal component analyses (PCA). The major drawback of PCA is that the usage is limited to supervised learning. This is however a minor problem in this context as current object detection algorithms are practically always based on supervised learning. In what follows some supervised learning techniques will be discussed.

A popular technique is using SVM. This classifier is trained with a number of examples. The feature space dimension can be reduced by applying PCA. SVM is a binary classifier. A hyperplane divides the future space in two volumes. The calculation of this hyperplane is an optimisation problem which is solved during the training sequence. Sometimes feature-descriptors are non-linear and can not directly be used in an SVM. This is solved by applying a kernel which maps the non-linear space to a (higher dimension) linear space [49].

A computational efficient technique is cascaded classification. This technique uses multiple simple classifiers to obtain a good performing classifier. The gain is found in the fact that a lot of candidates are already rejected in the early stages and do not need processing in the later more computational intensive stages. A graphical representation is given in Figure 5.6.

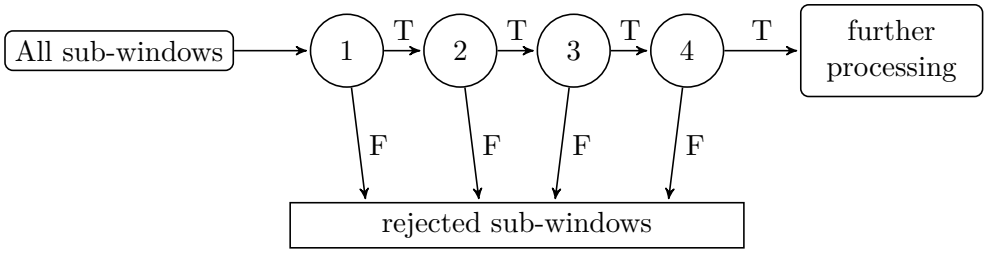


Figure 5.6.: Schematic depiction of a the detection cascade [50].

Since the first usage of cascaded classifier by [50] a lot of improvements have been proposed. A non-exhaustive overview and a new proposition can be found in [51]. In our implementation the HAAR-like features are used and the trainer is based on the ADABOOST algorithm that is already integrated in OpenCV.

5.2.5. Combining object detection and line segment detection

In order to combine this object detection algorithm with the former LSD algorithm a line segment - rectangle collision detection algorithm is used. LSD produces horizontal line segments and the object detector produces rectangles in the regions of interest. The object detector is used as the primary detection mechanism. All the horizontal lines that are (partially) detected within the rectangular regions are withheld while the others are thrown away. The line segments are defined by two points: $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$. The rectangle on its turn is defined by two opposite corner points: $(X_{\min}, Y_{\min})$ and $(X_{\max}, Y_{\max})$. The goal now is to verify if the lines fall within the rectangles. In image processing there already exist different algorithms, also called 2D clipping algorithms, such as the Liang–Barsky algorithm or Cohen Sutherland

clipping algorithm [52] to address such a problem. A simplified version of the latter was used in this thesis.

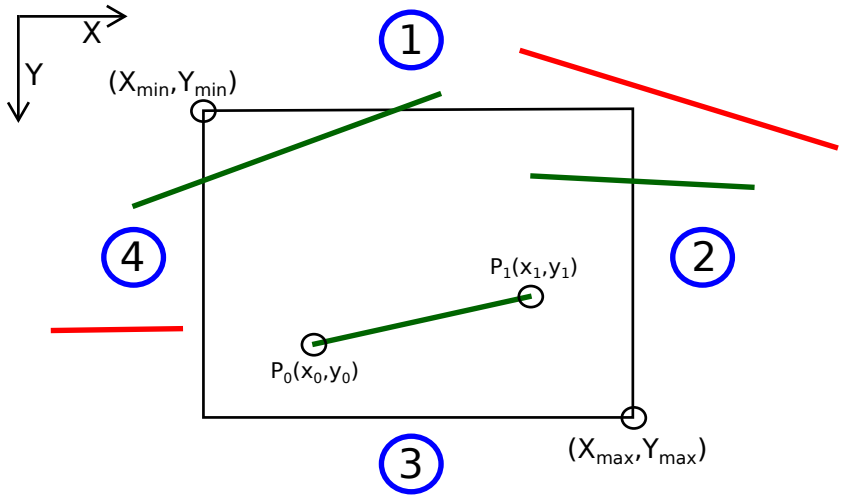


Figure 5.7.: Line-rectangle intersection principle. The green lines are withheld. The red lines are rejected

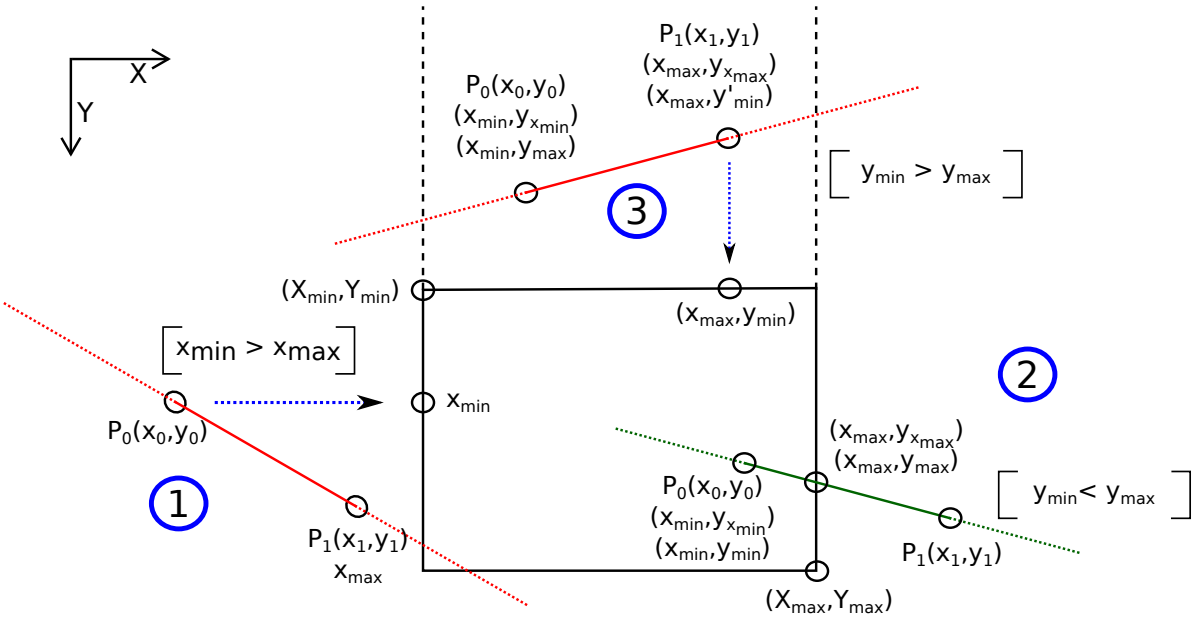


Figure 5.8.: Line-rectangle intersection examples. Case 1 is rejected because $x_{\min}$ is greater than $x_{\max}$. Case 2 satisfies all the conditions and is therefore withheld. Case 3 doesn't satisfy the last condition because $y_{\min}$ is greater than $y_{\max}$.

The implemented algorithm is a boolean function which returns true if the line segment falls within the rectangle and returns false in all other cases:

1. If P_0 and P_1 are both on the right side (region 2 in Figure 5.7) or the left side (region 4) of the rectangle, false is returned. In order to do so, $x_{\min}$ and $x_{\max}$ are defined:

$$\begin{cases} x_{\min} = \max(\min(x_0, x_1), X_{\min}) \\ x_{\max} = \min(\max(x_0, x_1), X_{\max}) \end{cases} \tag{5.3}$$

If $x_{\min} > x_{\max}$, false is returned. This is illustrated in case 1 in Figure 5.8. Case 2 and 3 satisfy this condition.

2. Now $dx = x_2 - x_1$ is calculated. If the line segment is not vertical i.e. $|dx| > 0$, the intersection point is calculated in $x_{\min}$ and $x_{\max}$:

$$\begin{cases} a = \frac{y_2 - y_1}{dx} \\ b = y_1 - ax_1 \end{cases} \Rightarrow \begin{cases} y_{\min} = \max(\min(ax_{\min} + b, ax_{\max} + b), Y_{\min}) \\ y_{\max} = \min(\max(ax_{\min} + b, ax_{\max} + b), Y_{\max}) \end{cases} \tag{5.4}$$

   If $y_{\min} > y_{\max}$, false is returned. This is illustrated in case 3 in Figure 5.8

3. If condition 1 and 2 are satisfied, the line will be withheld. This is the case for 2 in Figure 5.8.

5.3. 3D point-cloud processing

Using LSD-SLAM (section 4.3) 3D data is generated. This data is semi dense and can not directly be fed in existing algorithms as discussed in section 4.3. The discontinuity makes processing hard because mathematical operations like derivatives only perform well on continuous data. The final goal is to determine where in 3D curbs are likely to be found. This is done via the curvature index, a measure further explained in subsection 5.3.3.

5.3.1. Collect 3D data

The analysis proposed in [36] is done on a depth image. Real-time depth images are not available from LSD-SLAM and are thus generated. A first step in this endeavour is to collect all the 3D data generated by LSD-SLAM and put it into a point-cloud.

The 3D data generated by LSD-SLAM is an assembly of KFs linked together by the pose-graph. The pose-graph contains the pose of every KF and links them together. The pose-graph is regularly updated by LSD-SLAM using pose-graph optimization and large loop closure as discussed in section 4.3. Regenerating the resulting cloud on every update would be a computational intensive task. Because of this, the complete point-cloud is never aggregated.

To get the depth image the 3D data must firstly be transformed to the camera frame. The transformation to the camera frame is done from each KF separately. The current camera frame is given by the data in the published LFs. Because not every keyframe (section 4.3) contributes to the current scene, only the KFs closest to the current LF are transformed, to save computing time.

Transformation is done using pose algebra and similarity transforms as described in section 4.1. In this calculation there is no difference between the $SIM(3)$ representation and the $SE(3)$ representation. In practice the matrix representation of these entities is used to perform the calculations. In Listing 5.2 the basic algorithm is described with ξ_{KF} the pose of a KF, ξ_{LF} the pose of the current LF, P_{KF} the points described in the local coordinates of the KF, and P_{LF} the points described in the considered LF and stored in a point-cloud.

Listing 5.2: Pose transformation. The concatenation operator $\circ$, has been defined in section 4.3.

```
1 Initiliaze empty point-cloud, P_Σ
2 foreach(KF in graph)
3 {
4         get KF pose, ξ_KF
5         get transformation from KF to LF, ξ_KF-LF = ξ_LF^-1 ∘ ξ_KF
6         transform KF to pointcloud in LF, P_LF = ξ_KF-LF · P_KF
7         check if the points P_LF is not to far nor to close to be
              interesting and throw those points away
8         aggregate pointcloud,  P_Σ = P_Σ + P_LF
9 }
```

5.3.2. Project to depth-image

To get the depth image there is still a planar projection needed. The projection is derived from the pinhole camera model[10] and is already described in subsection 2.3.1.

When a projection of the point cloud in the camera frame is provided and $z \neq 0$, Equation (2.4) gives the projection from points described in the pose of the LF, $\{(x, y, z)\}$, to the camera plane, $\{(u, v)\}$, by

$$u = x \cdot f_x/z + c_x,$$
$$v = y \cdot f_y/z + c_y.$$

The condition $z \neq 0$ is always verified because points too close to the camera are not considered as shown in Listing 5.2.

Because the result needs to fit within the image a check must be performed if the result lays within the image boundaries. This is not only a physical problem but would also cause a problem in the implementation because the program would try to write to addresses that do not exist.

Listing 5.3: Projecting the pointcloud

```
1 foreach( point(x,y,z) in P_Σ)
2 {
3         u = round(x*fx/z + cx)
4         v = round(y*fy/z + cy)
5         if(u,v within boundaries)
6                 depthimage(u,v)=min(z, depthimage(u,v))
7 }
```

The pointcloud is rather dense at certain locations and some pixels really get a lot of projections. To deal with this only the smallest distance is withheld and the resolution of the generated depth image is augmented. Afterwards an erode is applied to make the image more

dense. The erode function e with a rectangular kernel applied to an image $I \in \mathbb{R}+$ is given by

$$\mathsf{e} : \mathbb{R}^{w \times h} \to \mathbb{R}^{w \times h} : I_{i,j} \mapsto \mathsf{e}(I_{i,j}) = \min_{m=i-\mathbf{w}/2+0.5}^{i+\mathbf{w}/2+0.5} \left(\min_{n=j-\mathbf{h}/2+0.5}^{j+\mathbf{h}/2+0.5} (I_{m,n}) \right)$$

with

- $I_{m,n} = \begin{cases} I_{i,j} & \text{if } m \in [0,w] \text{ and } n \in [0,h] \\ 0 & \text{otherwise} \end{cases}$
- $w \in \mathbb{N}_0^+$ the width of the image
- $h \in \mathbb{N}_0^+$ the height of the image
- $\mathbf{w} = 2 \cdot k + 1, k \in \mathbb{N}_0^+$ the width of the erode kernel
- $\mathbf{h} = 2 \cdot l + 1, l \in \mathbb{N}_0^+$ the height of the erode kernel

The only pixels that are updated in this thesis are the ones that did not had a value before. This is done to minimize the addition of new information that may disturb the original data. The image is afterwards resized to the original size for further processing. The result is visible in Figure 5.9.

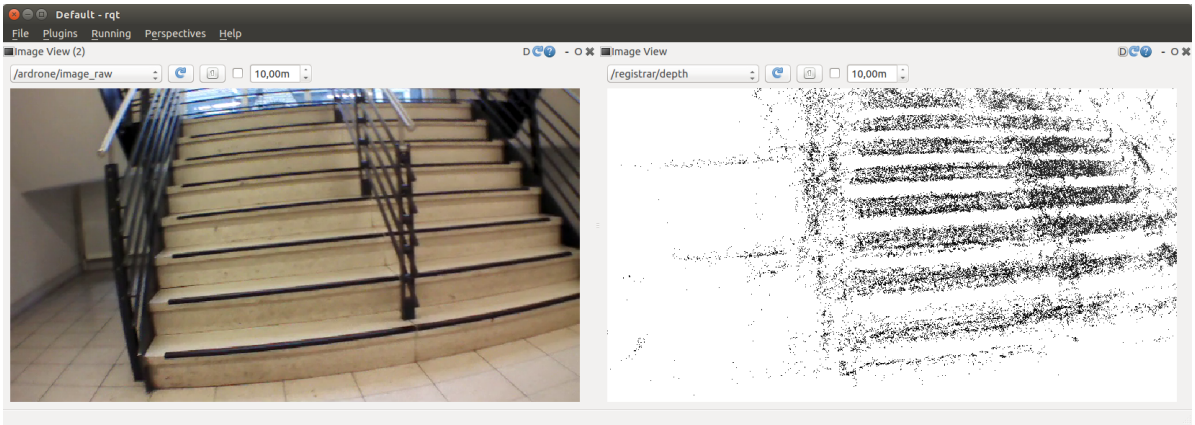


Figure 5.9.: Near realtime rendering of a depth image in the current camera pose.

5.3.3. 3D curvature calculation

Curvature describes how much a geometric object deviates from being flat. Applied to surfaces this can be used to do a quantitative analysis how much and in which direction a surface is bend. Two main ways are used: the principal curvature [53] and the Gaussian and mean curvature.

Any point on a surface has a normal vector. Every plane containing this vector is called a normal plane. The curvature of the section between the surface and a normal plane is called the normal curvature. The measure chosen is generally the inverse radius of the osculating circle. The two principal curvatures (κ_1, κ_2) are the minimum and maximum normal curvature.

Gaussian curvature K and mean curvature H are related to the principal curvatures by

$$K = \kappa_1 \cdot \kappa_2$$
$$H = \frac{\kappa_1 + \kappa_2}{2}$$

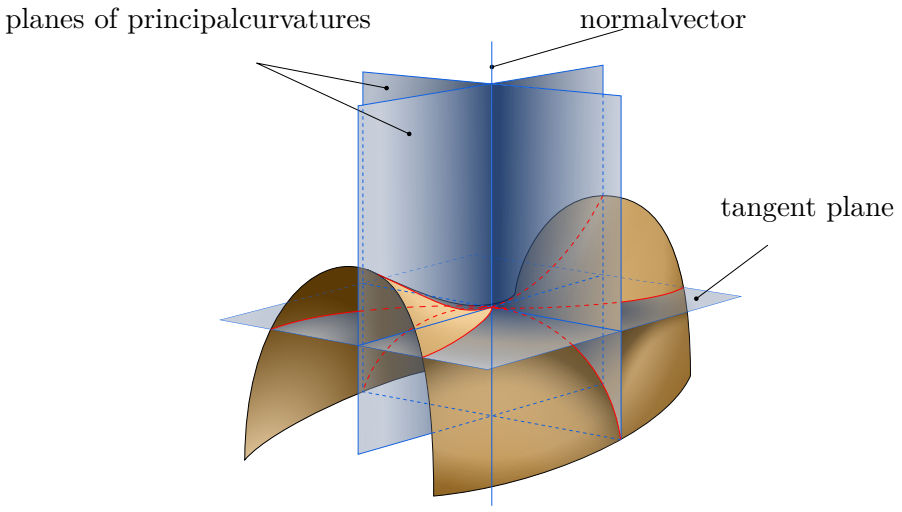


Figure 5.10.: Visualisation principal curvature [53]

These measures are also used in a technique called HK-segmentation. This technique tries to classify surface patches based on how they curved.

Table 5.1.: surface patches classification scheme [54].

K	H	Local shape class
0	0	plane
0	+	concave cylindrical
0	-	convex cylindrical
+	+	concave elliptic
+	-	convex elliptic
-	any	hyperbolic

The curvature index indicates how much the surface is curved in a point without taken into account the direction. It is calculated using the algorithm explained in [36] and the formulas in [54]. The general idea in [36] is to weigh the detected edges with a curvature derived from a depth image. Consider a surface α described by

$$\alpha : h = f(x, y)$$

with $x, y \in \mathbb{R}$ the ordinates of points on the surface and $h \in \mathbb{R}$ the height of each point. On this surface the spatial derivatives of first and second order with respect to x and y are denoted by h_x, h_y, h_{xx}, h_{yy}, and h_{xy}. The curvature index, CI, of such a surface is determined in [36] by

$$CI = \begin{cases} \dfrac{H^2 - K}{H^2 - \epsilon} & K \geq 0 \\ \dfrac{H^2}{H^2 - K - \epsilon} & K < 0 \end{cases}$$

with

- H the mean curvature given in [54] by

$$2H = \frac{(1 + h_x^2)h_{yy} - 2h_x h_y h_{xy} + (1 + h_y^2)h_{xx}}{(1 + h_x^2 + h_y^2)^{\frac{3}{2}}}$$

- K is the Gaussian curvature given in [54] by

$$K = \frac{h_{xx}h_{yy} - h_{xy}^2}{(1 + h_x^2 + h_y^2)^2}$$

- $\epsilon \in \mathbb{R}_0^+$ a small constant set to $\epsilon = 1$ in [36]

These formulas are particularly interesting because they only need the derivatives of the surface and this is numerically easy to do. A depth-image can considered to be such a surface and consequently this permits to calculate the curvature in every point. The drawback however is that the data is far from continuous and the approximation of the derivative by differentiating is unstable. This is resolved in [36] by applying Gaussian smoothing. However, this reforms the original data in the same way as with 2D noise removal.

5.4. Combining 2D and 3D detection

5.4.1. 3D line fitting

The detected stairs have only a meaning if they can be located in 3D. Therefore, the line-segments are pointwise backprojected on the depth image. On these 3D points a robust fitting is done using built in functions from PCL. The default *SACMODEL_LINE* is used in combination with the built-in RANSAC. The inliers are kept for further processing. A discussion on RANSAC can be found in subsection 5.2.3.

Per point that is retrieved in 3D the curvature is looked up. Afterwards the mean weighted curvature (subsection 5.3.3) is determined. However, this measure was not usable because the clutter in the depth image caused curvature to be detected everywhere as shown in Figure 5.11.

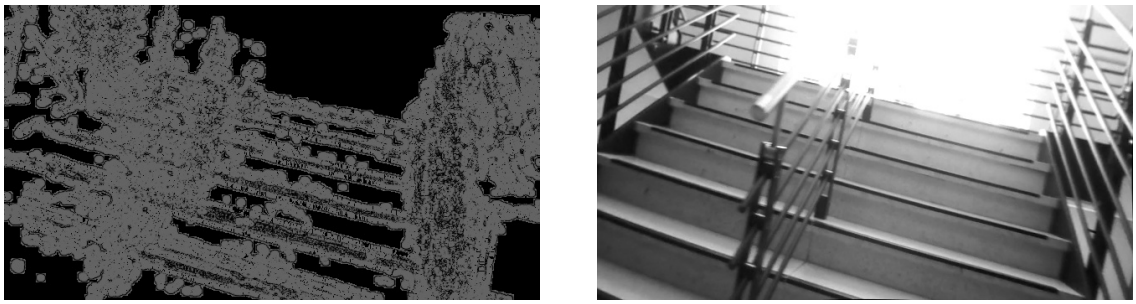


Figure 5.11.: Curvature index on the left with the corresponding rectified visual image on the right

5.4.2. 3D plane fitting and plane angle

The build-in *SACMODEL_PLANE* model from PCL is used to define the detected plane. The four coefficients of the plane are its Hessian Normal form: $[n_x, n_y, n_z, d]$. A standard plane equation can be written as $ax + by + cz + d = 0$. The normal vector is defined as follows:

$$\hat{n} = (n_x, n_y, n_z) \Rightarrow \begin{cases} n_x = \frac{a}{\sqrt{a^2+b^2+c^2}} \\ n_y = \frac{b}{\sqrt{a^2+b^2+c^2}} \\ n_z = \frac{c}{\sqrt{a^2+b^2+c^2}} \end{cases} \tag{5.5}$$

To determine the dihedral angle between two intersecting planes, the dot product is calculated between the normalized vectors of the two planes:

$$\begin{aligned} a_1x + b_1y + c_1z + d_1 = 0 \\ a_2x + b_2y + c_2z + d_2 = 0 \end{aligned} \Rightarrow \cos\theta = \hat{n}_1 \cdot \hat{n}_2 = \frac{a_1a_2 + b_1b_2 + c_1c_2}{\sqrt{a_1^2 + b_1^2 + c_1^2}\sqrt{a_2^2 + b_2^2 + c_2^2}} \tag{5.6}$$

5.4.3. Temporal consistency by agglomerative clustering

To take a decision on the where to find a staircase clustering is used. Clustering is an unsupervised machine learning technique that groups relevant entities together. This can be based on any metric or derived value.

When using a clustering mechanism, the number of clusters that are needed can be set beforehand (K-mean clustering) or the clusters can be determined via a type of linkage (hierarchical clustering). Because the amount of clusters is a priori unknown, this thesis implements the latter.

The different detections in this thesis are clustered together based on the euclidean distance in space as metric. To separate two sets from each other it must be described how they are linked. Different criteria exist but in this work the centroid linkage clustering is used:

$$\|c_s - c_t\| \qquad \text{with } c_s \text{ and } c_t \text{ the centers of cluster } s \text{ and cluster } t$$

Hierarchical clustering tries to establish a hierarchy between the different sets based on their metric and linkage. To create such an hierarchy two approaches exist: top-down and bottom-up. The top-down approach tries to iteratively divide the complete data set until every point has its proper set. The bottom-up approach iteratively groups different entities until all points are in one set. In this thesis the bottom-up approach is used, this is also called agglomerative clustering. An illustration of the principle can be found in Figure 5.12.

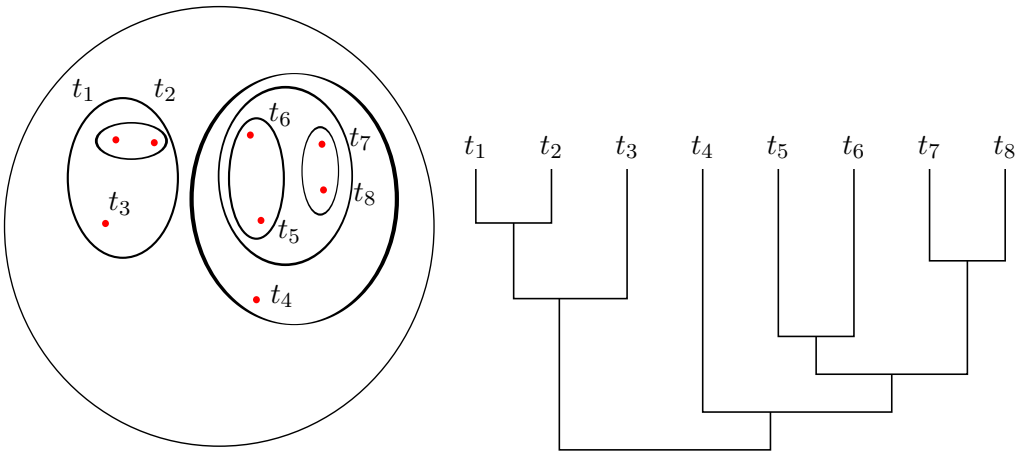


Figure 5.12.: Hierarchical clustering with representation of the data points t_i left and a bottom-up clustering right. [55]

The clustering mechanism that is implemented in this thesis takes two arguments: type of cluster distance and the minimum distance between two clusters. These parameters are respectively set to the centroid cluster distance with a minimum of one meter in between two seperate clusters.

5.5. Staircase projection model

In the scope of this thesis a 2D projection model was made of a staircase. However, some assumptions have to be made in order to simplify the model. A frontal view together with a point projection is assumed. No yaw or roll have been taken into account.

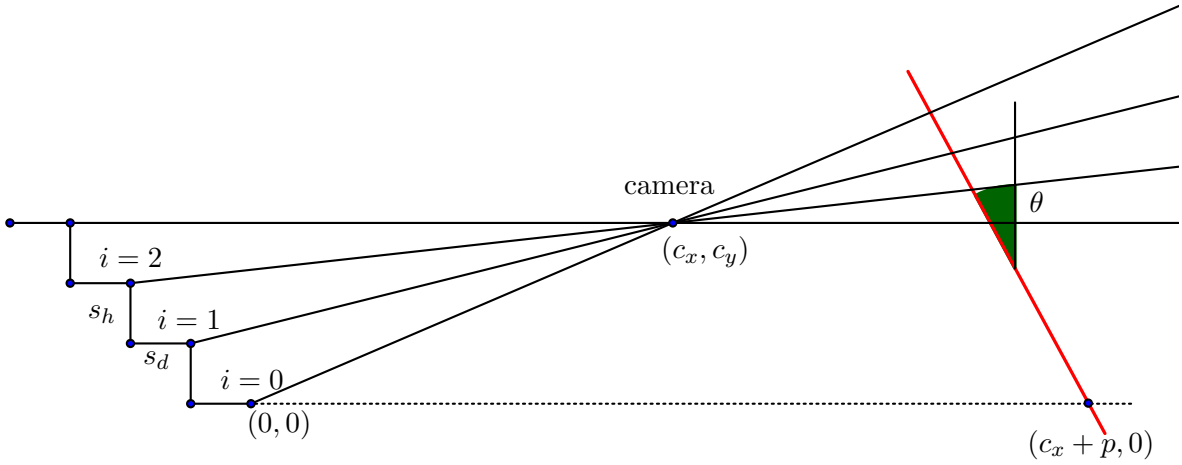


Figure 5.13.: Simplified staircase projection model

The beginning of the staircase is situated at the origin $(0, 0)$. The camera at coordinates (c_x, c_y) [m] and the projection screen has a pitch angle of θ. The dimensions of the staircase are characterized by the couple (s_d, s_h) and are given in meters.

For the derivation of the analytic model a sixth parameter p was used. The distance at which the projection screen intersects the x-axis is $c_x + p$.

5.5.1. Derivation

First, the lines that start at the beginning of each stair (i) and go through the camera's position can be modelled with the following formula:

$$r_i(x) = \frac{c_y - is_h}{is_d + c_x}(x - c_x) + c_y = \frac{c_y - is_h}{is_d + c_x}x + \frac{c_y is_d + c_x is_h}{is_d + c_x} \tag{5.7}$$

Secondly we have the equation of the projection screen:

$$P(x) = -\frac{1}{\tan\theta}(x - (c_x + p)) = -\frac{1}{\tan\theta}x + \frac{c_x + p}{\tan\theta} \tag{5.8}$$

If one wants to determine the intersection point of two crossing lines, the following formula can be used:

$$\left.\begin{aligned} y &= ax + b \\ y &= cx + d \end{aligned}\right\} \Rightarrow \text{Intersection point:} \quad \left(\frac{b - d}{c - a}, \frac{bc - da}{c - a}\right) \tag{5.9}$$

Formula (5.9) applied to Equation (5.7) and Equation (5.8) results in the following coordinates:

$$\begin{aligned} P_{x_i} &= \frac{(c_x+p)(is_d+c_x)-\tan\theta(is_h c_x+is_d c_y)}{\tan\theta(c_y-is_h)+is_d+c_x} \\ P_{y_i} &= \frac{is_d c_y+c_x c_y+pc_y-pis_h}{is_d+c_x+\tan\theta(c_y-is_h)} \end{aligned} \tag{5.10}$$

The relevant information that can be used is the euclidean distance (Equation (5.11)) between the intersection points. This distance corresponds to the observed distance between the projected lines of the staircase in a 2D image. However, absolute distances are not useful because these are a priori unknown in a 2D image. Therefore, the ratio of two successive distances is used to characterize a staircase (Equation (5.12)). The derived model is verified in Matlab and shown in Figure 5.14.

$$d_{P_i,P_{i+1}} = \sqrt{\frac{(p - c_y \tan\theta)^2(\tan^2\theta + 1)(c_y s_d + c_x s_h)^2}{(c_x + s_d i + c_y \tan\theta - s_h i \tan\theta)^2(c_x + s_d + s_d i + c_y \tan\theta - s_h \tan\theta - s_h i \tan\theta)^2}} \tag{5.11}$$

$$\frac{d_{P_i,P_{i+1}}}{d_{P_{i+1},P_{i+2}}} = \frac{c_x \cos\theta + 2s_d \cos\theta + c_y \sin\theta - 2s_h \sin\theta + s_d i \cos\theta - s_h i \sin\theta}{c_x \cos\theta + c_y \sin\theta + s_d i \cos\theta - s_h i \sin\theta} \tag{5.12}$$

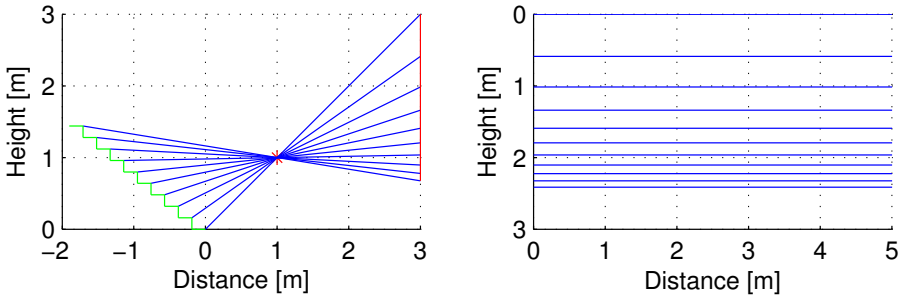


Figure 5.14.: Matlab model verification. On the left figure, the projection from the staircase (green) to the projection screen (red) is illustrated. On the right figure, the projected lines of an image are simulated.

5.5.2. Simplification

In other words, Equation (5.12) is a function $f(c_x, c_y, s_d, s_h, i, \theta)$ that depends on six variables. However by smartly estimating some parameters, an approximative function can be derived that only depends on two parameters: $\hat{f}(c_x, i)$:

- c_y can be measured by an altimeter (assuming the surface is flat which is a reasonable assumption for indoor environments).
- s_d and s_h are parameters that does not vary a lot. Therefore they can be approximated by a default value (e.g. 19cm and 16cm).
- θ corresponds to the pitch af the quadcopter and can be measured with the IMU.

5.5.3. Application

This model could be exploited by using template matching. Template matching slides candidate samples over the image with and indicates a match if the similarity is above a certain

threshold. However, there are a multitude of degrees of freedom like scale and camera orientation that need estimation or robust detection by creating samples for each value. Multiscale detection can be done using image pyramids while orientation may be solved by using techniques that estimate the pose of the camera like applied in PTAM and ORB-SLAM (chapter 4).

6. Autonomous guidance

Because of the high complexity of this thesis and the colossal amount of different packages that are used, a GUI is implemented to centralize all the software in one place. This chapter will elaborate the integration of all the different software packages, describe the implementation of the autonomous guidance and cover the functionalities of the GUI and how to use them.

6.1. Basis of the GUI

The GUI is completely written in Python and uses tkinter for the visualization. It uses the ROS library for python, `rospy`, and therefore acts as a rosnode. To launch the GUI, one should enter the following command: *rosrun ardrone_controller ardrone_gui_controller.py*.

Although the GUI is completely developed from scratch, a small portion of the code (controller class) is based on some existing scripts that were written by Mike Hamer and can be downloaded from the robohub.org website[1].

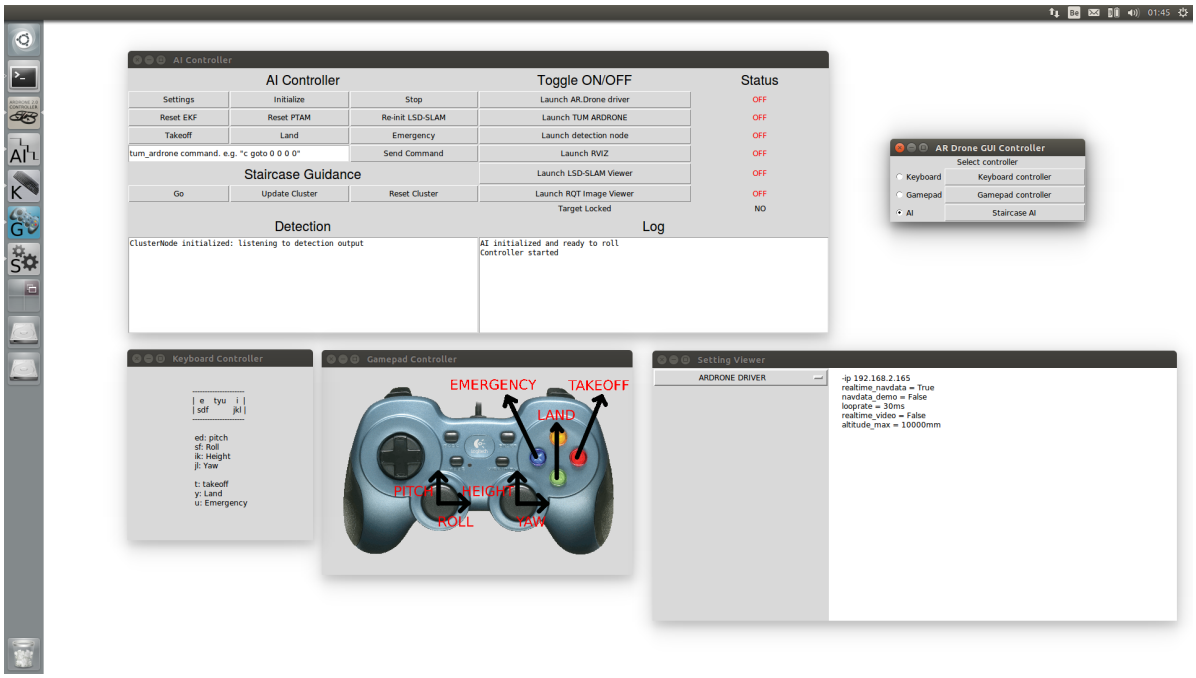


Figure 6.1.: Overview of the different GUI windows. In the top-left corner, the AI controller is shown. In the top-right the main window is opened. On the bottom from left to right there is the keyboard controller window, the gamepad controller window and the setting viewer window.

[1] `http://robohub.org/up-and-flying-with-the-ar-drone-and-ros-joystick-control/`

The GUI consists of three principal controllers: the keyboard controller, the gamepad controller, and the artificial intelligence (AI) controller. Each controller can be started by clicking on their respective buttons in the main window (Figure 6.2). Be careful to click the radio button next to it in order to activate the controller. Otherwise the commands will not be forwarded by the build-in controller of the GUI.

6.2. Controllers

The first controller is a window that allows the user to directly manoeuvre the quadcopter. This controller directly interfaces with the `ardrone_autonomy` packages via a controller class that is integrated in the GUI. The window should be focused during the key presses in order to control the drone. Information about the control keys is displayed on the window itself (Figure 6.3).

The second controller uses a gamepad (logitech, ps3, xbox) to control the quadcopter. The software was tested with a logitech gamepad and may require some additional configuration for other models. The principles are similar to that of the keyboard controller. The `joynode` in ROS is started and the commands of the gamepad are forwarded to the AR.Drone by the internal controller of the GUI.

An important condition in order for the first two controllers to work is the fact that the `ardrone_autonomy` package should be running. This can be done by clicking the AR.Drone driver button in the AI controller or by manually starting it with a launch file or just from command line.

The third and last controller is the AI controller and is the core application that is used to control all the software that is integrated in this thesis. The main task is to fly the drone to the detected points which represent the top of a detected staircase. Because the detection algorithms are not 100 percent accurate, a supervised autonomous guidance is implemented meaning that the quadcopter will signal the user if a detection has taken place and waits for the consent of the user to fly to this point. The control commands are not handled by the build-in controller this time but by `tum_ardrone` package. More information about the use of the AI controller will be provided in the next sections.

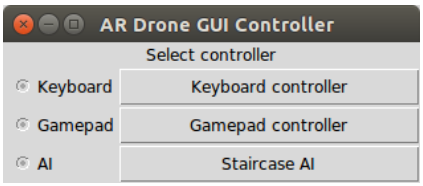


Figure 6.2.: The main window of the GUI from which the keyboard controller, gamepad controller or AI controller can be started.

6.3. Software integration

6.3.1. Software launcher

The entire upper-right part of the AI controller (Figure 6.4) is dedicated to the launch of all the necessary software packages for this thesis. A list is given below with all the ROS nodes

Figure 6.3.: Keyboard controller on the left and gamepad controller on the right

that are started by clicking each button.

- Launch AR.Drone driver:
  `ardrone_autonomy` - `ardrone_driver`
- Launch TUM ARDRONE:
  `tum_ardrone` - `drone_stateestimation`,
  `tum_ardrone` - `drone_autopilot`,
  `tum_ardrone` - `drone_gui`,
  `tf` - `static_transform_publisher`
- Launch detection node:
  `image_proc` - `image_proc`,
  `lsd_slam_core` - `live_slam`,
  `pointcloudregistration` - `registrar`
- Launch RVIZ:
  `rviz` - `rviz`
- Launch LSD-SLAM viewer:
  `lsd_slam_viewer` - `viewer`
- Launch RQT Image Viewer:
  `rqt_image_view` - `rqt_image_view`

The `pointcloudregistration` package contains the developed software for the staircase detection.

Each ROS node has its own parameters. Some can be dynamically reconfigured, some can only be changed upon start and others are hard coded in the source files. An overview of most of these parameters can be acquired by clicking on the *Settings* button in the AI controller. This opens up a window where the parameters can be asked for each individual package (Figure 6.5).

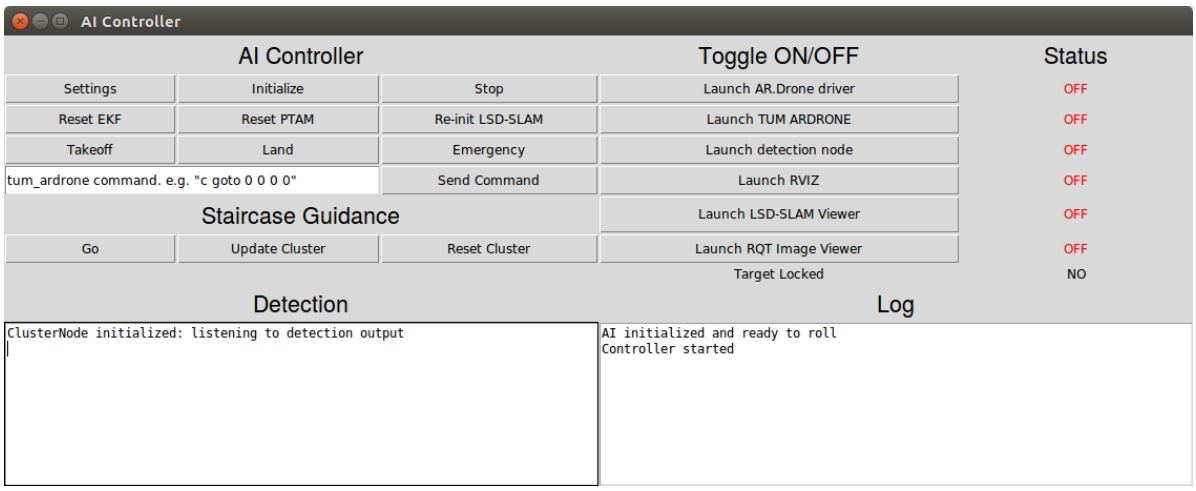


Figure 6.4.: AI controller from which everything is started and monitored

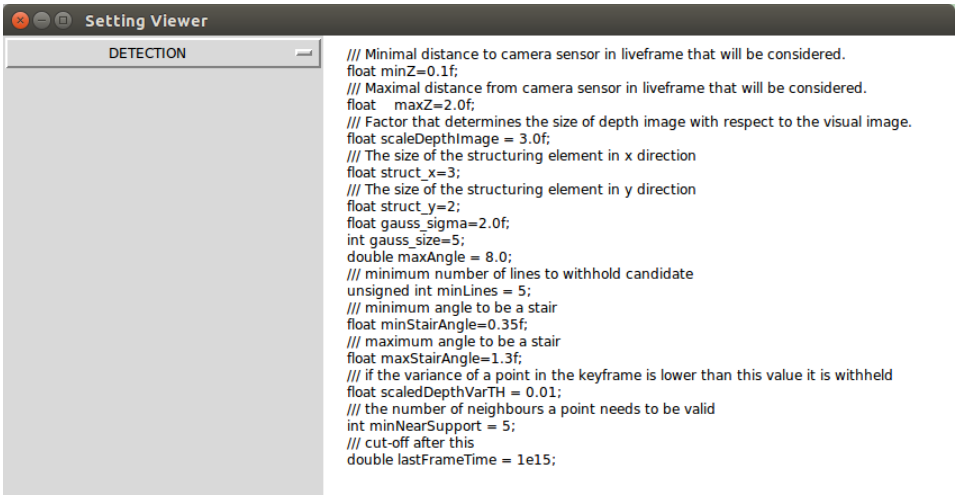


Figure 6.5.: A Setting window that can be started from the AI controller and gives an overview of all the parameters that are explicitly changed and added in all the different packages.

6.3.2. Clustering

The GUI clusters from start-up all the candidates that are generated by the detection node. The mechanism of agglomerative clustering, which is implemented, is explained in subsection 5.4.3. The points are projected to the reference coordinate system (*/map*) and collected in an array of points. If the required amount of points is reached, the clustering mechanism is run. An additional module in the code creates corresponding markers that are used by RVIZ to visualize the process (Figure 6.6).

6.4. Usage

The usage of the keyboard controller and the gamepad controller is already described in section 6.2. Therefore this section will primarily focus on the usage of the AI controller. Especially the initialization process has proven to be quite tricky and needs some explanation.

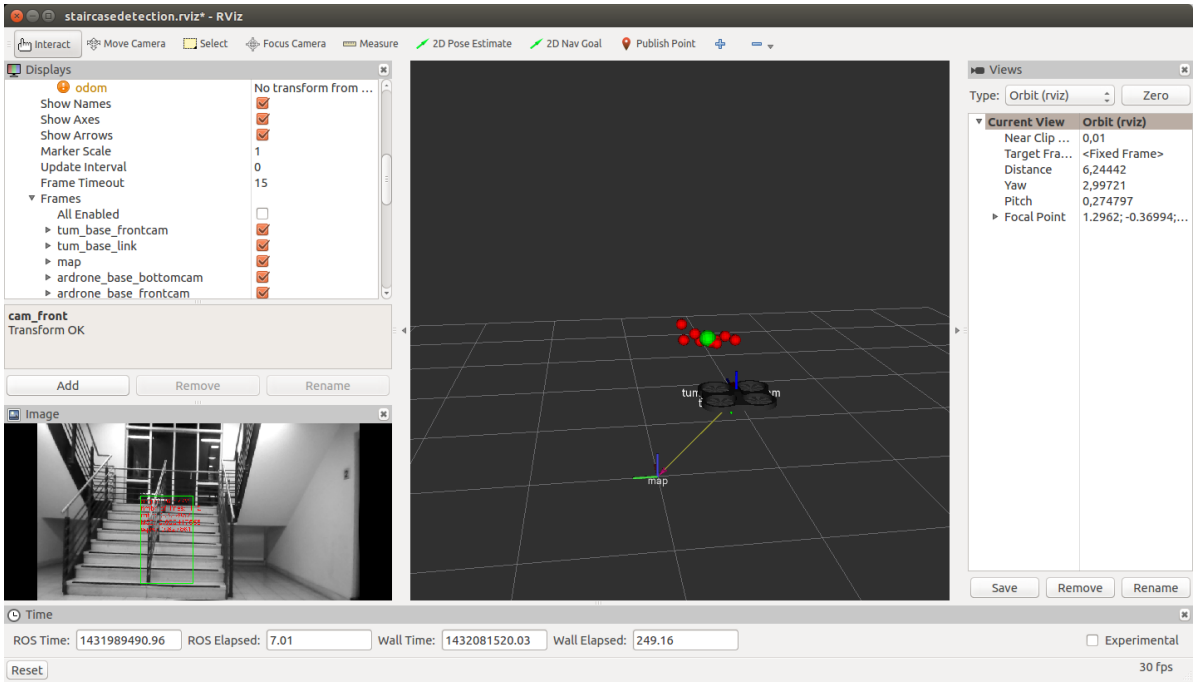


Figure 6.6.: Detection visualization in RVIZ. The red dots are individual detected points while the green dot is the final result after clustering.

6.4.1. tum_ardrone related usage

As already mentioned several times in this thesis, the `tum_ardrone` package is used to estimate the drone's position and to control it. The most crucial step in order to use this package is the initialization process of PTAM.

Initialization process

A good PTAM initialization consists of the following steps:

- Drone takes off and hoovers at approximate one meter above the ground while facing a planar scene such as a wall. The more texture the better.
- The first keyframe is taken
- One second later the second keyframe is taken (the small drift is enough to create a small viewpoint difference)
- The drone should now go up and down over a range of minimum one meter. This should increase the accuracy of the scale estimation.

An important factor is the accuracy of the scale estimation (shown on the bottom of the left image of Figure 4.6). This should be equal to one in order to correctly fly the quadcopter.

Despite the efforts of the `tum_ardrone` team to automate this initialization, this process does not always succeed. Moreover the same method performs differently on different platforms[2].

[2]During tests with `tum_ardrone`, different computers showed distinctive behaviour. It is the authors believe that the performance of the platform impacts the operation of PTAM which is not necessarily better if the platform has more performance.

Because of the difficult initialization process, the user has three different options. First of all the user can manually initialize PTAM[3] by using the `tum_ardrone` GUI. The second option is to use to auto-initialization scripts of `tum_ardrone` (e.g. initDemo.txt), which can also be selected in `tum_ardrone`. Make sure the drone is not in the *Emergency* state because this will block the quadcopter from taking off and consequently PTAM will crash because two keyframes will be taken on exactly the same location. A third and last method is implemented in the GUI that is developed for this thesis. This method is called by pushing on the *Initialize* button and uses a combination of manual commands to the `ardrone_autonomy` package and the auto-initialization function of `tum_ardrone`.

This is implemented in the code with the initialization button. The callback function behind this button resets the EKF of `tum_ardrone` and sends an initialization command to the `drone_autopilot` node. This command is explained in the documentation of `tum_ardrone`.

If the visual tracking is lost and cannot be recovered, it is possible to reset PTAM by pushing the *Reset PTAM* button.

EKF

The EKF within `tum_ardrone` can be reset by pressing the *Reset EKF* button in the GUI. This brings the drone to the origin of the reference coordinate system and flushes all previous data. This can be usefull to remove for example initial drift before taking off.

If commands are send to the drone and one wants to reset the command queue and stop all ongoing manoeuvres, the *Stop* button can be pressed.

Send commands to tum_ardrone

The GUI offers the possibility to send manually commands to the `drone_autopilot` node. More documentation about the commands can be found in [19]. Some interesting commands are:

- c setReference $POSE$ (Sets reference point from which the goto commands are interpreted)
- c goto x y z w (w the yaw angle in degrees, coordinates are in meters)

6.4.2. State control

To control the state of the quadcopter, direct access to the `ardrone_autonomy` package is provided for the *Takeoff*, *Land* and *Emergency* buttons.

[3] Has proven to be the fastest and most reliable way

6.4.3. LSD-SLAM initialization

The detection algorithm that is used in this work uses LSD-SLAM as a source of 3D information. A good initialization of LSD-SLAM is therefore paramount to obtain a valuable detection.

The function behind the *Re-init LSD-SLAM* button does just that. It calls the reset service of LSD-SLAM, that was newly developed for this thesis, and sends a lateral flying pattern to the `tum_ardrone` autopilot to improve the quality of the generated pointcloud.

6.4.4. Cluster control

As it was previously explained, once the cluster algorithm has enough points it calculates the central point of the biggest cluster. This point is withheld until the user presses the *Update Cluster* button which will recalculate the central point from the latest set of points. The *Reset Cluster* button totally flushes all points and restarts collecting new points.

The *Go* button sends the drone to the central point that is selected by the clustering algorithm.

7. Methodology

This thesis is made by two authors and involves a setup of different softwares (chapter 2). To make such things workable and prevent possible conflicts during development, the authors chose to use certain techniques and tools.

7.1. Version management and collaboration

Version management is important if one wants to keep track of the changes he made or the changes made to a platform/software he is using. This is particularly handy if one wants to follow the progress of his/her colleagues. A gitlab(section 2.8) server was set up to manage both the program software code and the LaTeXcode of this thesis. Because the code is pushed to a server and stays on the computer of the clients, a backup is automatically generated.

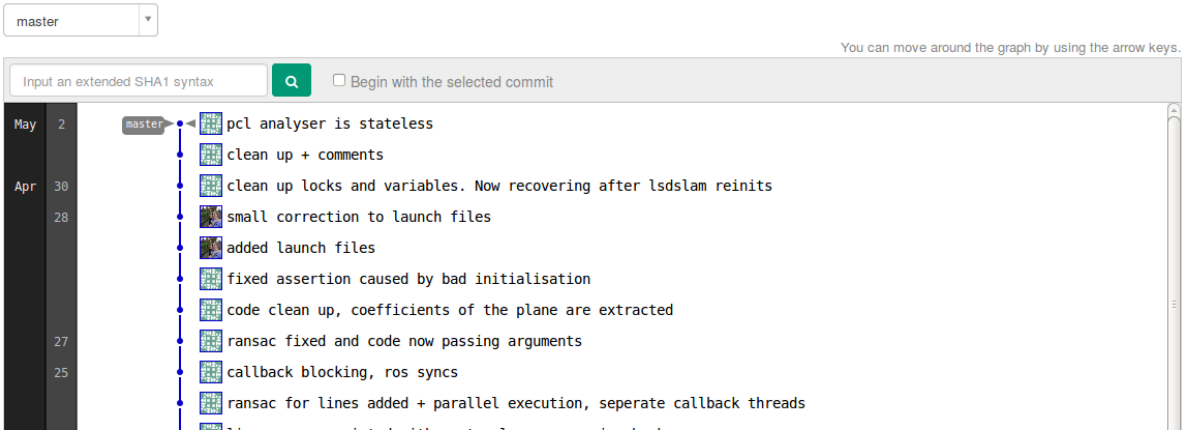


Figure 7.1.: Gitlab overview of all the commits

As already mentioned in section 2.8, the wiki pages of Gitlab are used to exchange ideas and tutorials. A full list of install instructions is published on the wiki pages. This improves productivity when a full reinstall is required or a new computer has to be configured to work with the same set-up.

7.2. Software

Multiple compatible software packages already exist for ROS. Therefore, it was preferred to use existing *standards* or tools whenever possible. This was not always easy because even the community does not always respect their own standards or supports them. The most prevalent example was the incompatibility of rqt with the default output format for depth images. This output format is also used with any value derived from the depth images. The incompatibility

is because rqt can not handle 16 bit images properly and reads the image as if it were an 8 bit image. The effect is shown in Figure 7.2. This can be solved by using rviz or by saving the image with the `image_proc` package and viewing them afterwards with another viewer.

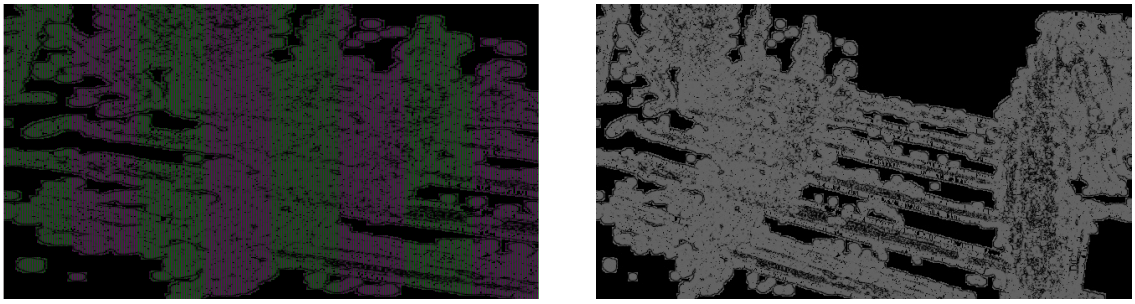


Figure 7.2.: rqt with 16 bit images. On the left the image as shown in rqt and on the right the image as it is supposed to be.

To ensure the results in this thesis are reproducible all changes made to existing software is kept track off. The important changes or documented throughout this thesis and in Appendix B. The procedure to install every dependency is put in an install script that can be found in Appendix C. These install scripts pull the needed repositories from a server connected via VPN. In other set-ups different links to the repositories need to be changed.

To further facilitate the usage of the results in this thesis the interface of the different classes within the main detection package is documented. The documentation is generated by doxygen and adapted for usage within this document. The result can be found in Appendix A.

The software in this thesis compiles against the C++ 11 to make the code somewhat future proof and to take advantage of compiler optimisations that are available for this standard. This was not straight forward as PCL is not by default compiled against C++ 11 and runtime errors occurred. This can be solved by compiling PCL from source. In this thesis the latest release of the 1.7 branch (1.7.2) was chosen to not break compatibility with the former version included by the system (1.7.1).

There was a maximum of standard, platform agnostic C++functions used. In cases where this is not possible the platform independent BOOST-library was used. This library will convert the commands only to platform specific code upon compilation.

To stay within the near real-time constraints, parallel processing is used by using async spinners in ROS and the Intel TBB library in the detection code. The async spinners start up a separate thread for the message handling in ROS and the execution of the callback. Two spinners are started within `pointcloudregistration`: a spinner handling the KFs and the pose-graph updates, and a spinner handling the LFs and the images to initiate the detection. Intel TBB is used to process multiple candidates in parallel per frame. The usage of multiple cores by `pointcloudregistration` can be seen in Figure 1.2.

7.3. Algorithm development and testing

To test and debug the implementation of the algorithms publisher with intermediary data are created. These publishers become only active when there is a subscriber on them. An overview of all the publishers is given in Figure 7.3. The functionality of each publisher is as follows:

- `~/depth_filtered` a 16 bit grayscale image that represents the depth images filtered by using Gaussian smoothing.
- `~/K` a 16 bit grayscale image that represents the Gaussian curvature.
- `~/lines_curv` an RGB image that depicts the lines that are used for back-projection. The color indicates the mean curvature index.

- `~/H` a 16 bit grayscale image that represents the Gaussian curvature.
- `~/candidates` a RGB image that shows a rectangle corresponding to the 2D object detection. result with extra information about the 3D detection.
- `~/curvature` a 16 bit grayscale image that represents the curvature index;
- `~/depth` a 16 bit grayscale image that represents the unfiltered depth image.
- `~/lsd` a RGB image with the lines detected by the line segment detector. The lines that are horizontal with respect to the data from the EKF are green the others are red.
- `~/target` a topic on which a single point per candidate is posted. This is the highest points of the points that are inliers in the detected plane.
- `~/detect` an image with the output of the cascade classifier.

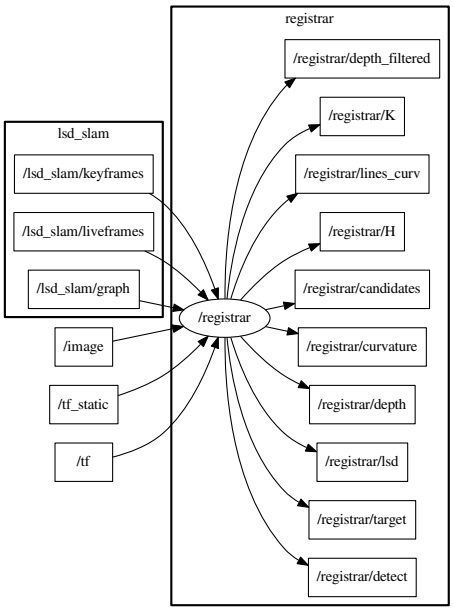
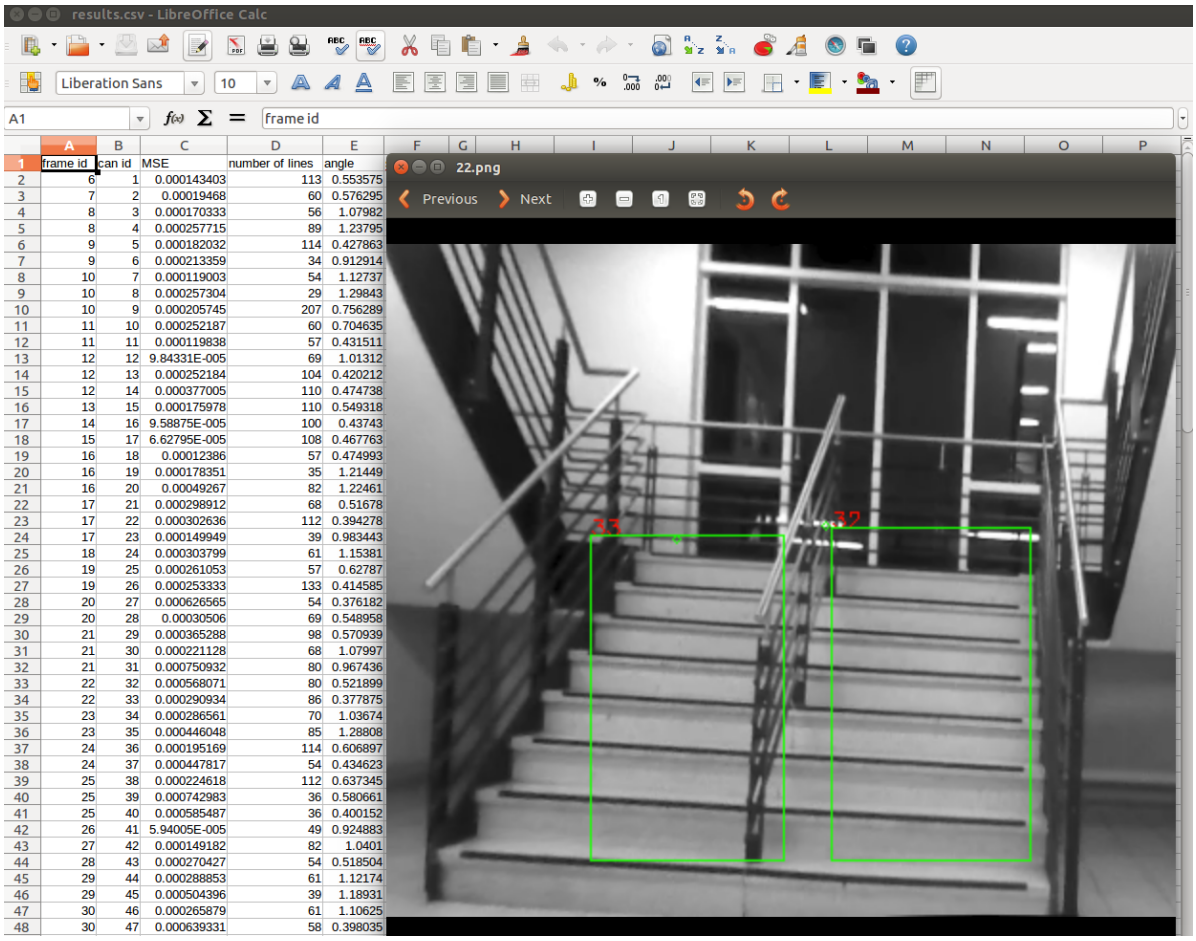


Figure 7.3.: Overview of different debug topics

To verify the results more in detail the `image_view` can be used to save the images from the topics mentioned above. Another way is the performance monitor that was built-in. This provides an overview of the metrics in a csv-file while per detection cycle the image is saved

with an unique id. If there were candidates the rectangle from the object detector is added and the 3D point from the `~/target` topic is projected on it. Every candidate has also an unique id during a test-run. The result is shown in Figure 7.4.



	A	B	C	D	E
1	frame id	can id	MSE	number of lines	angle
2	6	1	0.000143403	113	0.553575
3	7	2	0.00019468	60	0.576295
4	8	3	0.000170333	56	1.07982
5	8	4	0.000257715	89	1.23795
6	9	5	0.000182032	114	0.427863
7	9	6	0.000213359	34	0.912914
8	10	7	0.000119003	54	1.12737
9	10	8	0.000257304	29	1.29843
10	10	9	0.000205745	207	0.756289
11	11	10	0.000252187	60	0.704635
12	11	11	0.000119838	57	0.431511
13	12	12	9.84331E-005	69	1.01312
14	12	13	0.000252184	104	0.420212
15	12	14	0.000377005	110	0.474738
16	13	15	0.000175978	110	0.549318
17	14	16	9.58875E-005	100	0.43743
18	15	17	6.62795E-005	108	0.467763
19	16	18	0.00012386	57	0.474993
20	16	19	0.000178351	35	1.21449
21	16	20	0.00049267	82	1.22461
22	17	21	0.000298912	68	0.51678
23	17	22	0.000302636	112	0.394278
24	17	23	0.000149949	39	0.983443
25	18	24	0.000303799	61	1.15381
26	19	25	0.000261053	57	0.62787
27	19	26	0.000253333	133	0.414585
28	20	27	0.000626565	54	0.376182
29	20	28	0.00030506	69	0.548958
30	21	29	0.000365288	98	0.570939
31	21	30	0.000221128	68	1.07997
32	21	31	0.000750932	80	0.967436
33	22	32	0.000568071	80	0.521899
34	22	33	0.000290934	86	0.377875
35	23	34	0.000286561	70	1.03674
36	23	35	0.000446048	85	1.28808
37	24	36	0.000195169	114	0.606897
38	24	37	0.000447817	54	0.434623
39	25	38	0.000224618	112	0.637345
40	25	39	0.000742983	36	0.580661
41	25	40	0.000585487	36	0.400152
42	26	41	5.94005E-005	49	0.924883
43	27	42	0.000149182	82	1.0401
44	28	43	0.000270427	54	0.518504
45	29	44	0.000288853	61	1.12174
46	29	45	0.000504396	39	1.18931
47	30	46	0.000265879	61	1.10625
48	30	47	0.000639331	58	0.398035

Figure 7.4.: Performance monitor

To test if the decision based on angle would be an addition, a MatLab script was written. The script looks at the depth image in two different zones. The angle with respect to the ground plane is then calculated assuming the camera was perpendicular with respect to the ground. The code can be found in Appendix D. Two example results are listed in Figure 7.5.

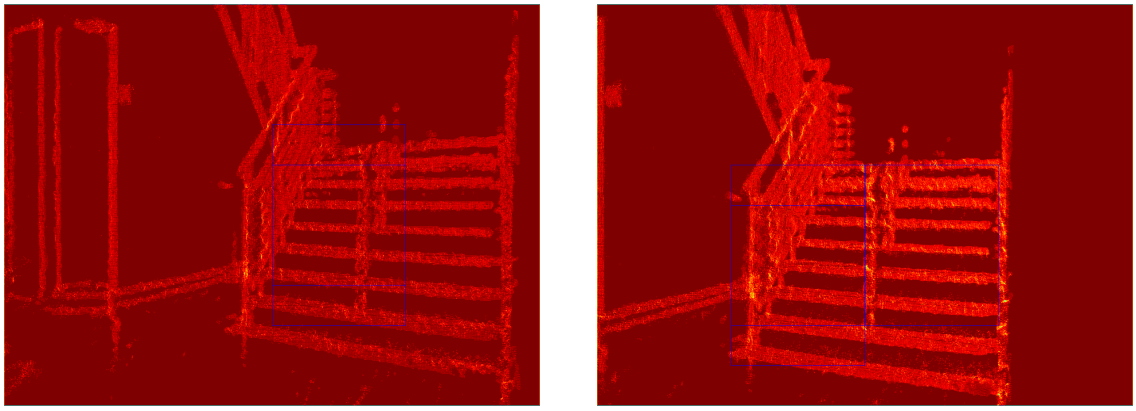


Figure 7.5.: Matlab results

8. Future work

This thesis is certainly not complete and improvements are possibly in many directions.

8.1. Computer architecture

One possible improvement is the usage of different computer architectures. The main problem with current General-Purpose computing on Graphics Processing Units (GPGPU) is that, although the fact that they are performant, the data needs to be copied to the Video random-access memory (VRAM) before processing (Figure 8.1).

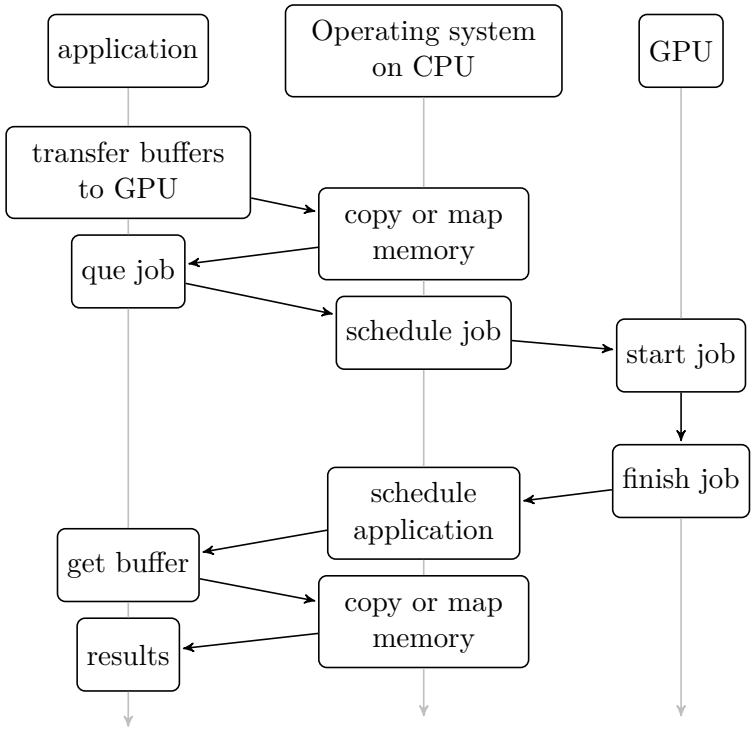


Figure 8.1.: Using the GPU without Heterogeneous System Architecture (HSA)[56]. Zero-copy possible with unified system memory.

The problem of needing to copy everything around to do GPU processing is already addressed in the latest game consoles like the PS4 and Xbox One using a so called HSA (Figure 8.2). Chip-makers and platform developers are also investing in these architectures. This is however not the single solution towards massive parallelism but the one that is likely to be the most affordable and supported today and in the near future.

When commercialising such systems or optimizing them for high performance capabilities the usage of Field programmable gate arrays (FPGAs) for doing fixed preprocessing is recom-

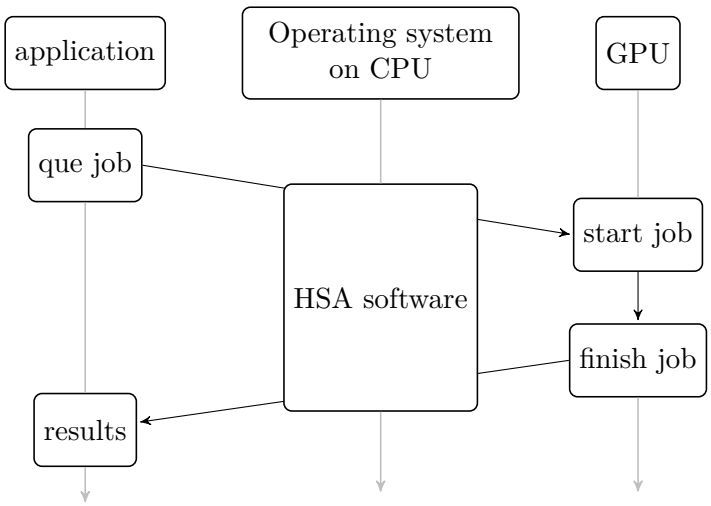


Figure 8.2.: Using the GPU with HSA[56]. Zero-copy possible with unified system memory.

mended. It is however not always trivial to adapt existing algorithms to a FPGA implementation. The performance gain by using FPGAs lies in the specific instructions that are preprogrammed in stead of the general instructions that are in place in central processing units (CPUs) and GPUs.

Further improvement is not only in using specialised processors but also by using existing techniques as Streaming SIMD Extensions (SSE) on X86 machines and NEON™ on Acorn RISC Machine (ARM®) machines. Both techniques speed up arithmetic operations by paralysing instructions and using specific parts of CPU. The downside is that programming is sometimes cumbersome and version dependant. These techniques are already applied within some off the used libraries like OpenCV, Eigen and PCL. But some interconnecting code does not support new processor optimisations.

8.2. Platform

While the Parrot AR Drone is practically the single commercial available drone that has such an elaborate SDK, a lot of documentation on the hardware is still missing. Notably the accuracy of the onboard accelerometer. Another pain point is the limited expandability and upgradeability of the platform. Changes that involve extra sensors generally involving adding an extra communicating channel while the bandwidth of the added sensor does not necessitate this at all.

The onboard camera should be a global shutter type for alike implementations with a higher framerate. Installing a stereocamera or a time-of-flight (ToF) camera omits the need of a framework the distillates depth images out of a mono-ocular camera feed. These camera systems tends to add some weight and therefore reduce the additional payload however less processing is required thus less energy is consumed by the processing unit. Another option is the usage of algorithms that make use of GPUs to calculate a more dense output.

Not only the sensors and *avionics* on the drone need to be analysed but also the size and durability of the platform. While the parrot drone is small compared to most professional outdoor drones it is quite large for an indoor drone. This impacts the flight performance

because several aerodynamic affects arise from the fact that the drone is relatively to its size close to obstacles. The most noticeable effect is when the drone comes to close to the wall it gets sucked towards it and crashes. However, smaller drones are inherently more instable because their inertial moments are much smaller. This puts more real-time constraints on the flight controller.

8.3. Algorithms

8.3.1. SLAM improvements

This thesis relies heavily on existing slam algorithms that have redundant capabilities. The best example is the pose estimation from PTAM and from LSD-SLAM. An ideal setup would only include one instance of pose estimation and exchange the information between each other. A cooperative mechanism would also result in a more robust tracking in LSD-SLAM as position is kept by the EKF and temporary loss in visual tracking should not mean complete loss of pose.

ORB-SLAM is very promising as discussed in section 4.5. Certainly the auto-initialising capability would simplify the controller. Also full integration should be the goal such that LSD-SLAM or a similar replacement could profit from ORB-SLAM to obtain large loop-closure to solve the aforementioned tracking lost problem.

8.3.2. EKF

The Wifi channel delay estimation in the state estimation algorithm can still be improved and may have a non-negligible influence on the performance of the EKF. This issue could be overcome by putting the EKF on-board. This is certainly possible with the processing power in today's embedded platforms even including the packages used for visual odometry by feature based SLAM.

Estimation of the yaw angle could be further improved by adding the magnetometer to the package.

8.3.3. convolutional neural network

Logistic regression is a linear projection of an input vector onto a set of hyperplanes. Each hyperplane represents a class. This input vector can be a sequence of raw pixel values or a vector with feature descriptors. A multilayer perception (MLP) combines logistic regression with an initial non-linear mapping as shown in Figure 8.3. A convolutional neural network (CNN) is a by biology inspired MLP. A CNN is commonly sparsely connected and has a multitude of hidden stages.

The idea to mimic the functioning of the human brain to solve vision problems is far from new. Initial results left a lot to be desired. This was mainly caused by the limited training-data available. The exploitation of big data in the ICT world has also opened new opportunities for machine learning. Big data can easily generate training data that are several orders of magnitude bigger. A recent effort in this is imagenet [1]. This is a platform where a lot

[1] `http://www.image-net.org/`

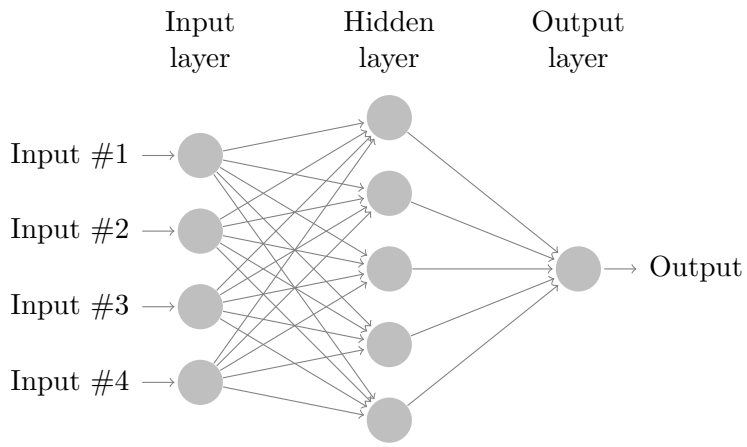


Figure 8.3.: multilayer perception[62]

of digital images are classified and/or annotated by humans via for example the Amazon Mechanical Turk marketplace[2]. The data on this site is freely available via an application program interface (API).

8.3.4. Robust fitting and probabilistic robotics

In this work RANSAC is used to do robust fitting. However, in recent years, a multitude of alternatives have arisen with specific use cases. The study of the different techniques for the different cases of robust fitting applied in this thesis fell out of the scope of this work. However performance could be improved using other techniques but also deteriorate the outcome.

In practice a lot of measurements and estimations have uncertainties. Probabilistic robotics aims to propagate these uncertainties throughout the system to have a better idea of the uncertainty when a final decision needs to be made. Robust fitting could play a role in this as some techniques are appropriate to not only predict a model but also to estimate an uncertainty of this model.

Next to taking into account the uncertainty of robust estimations the uncertainty of the data generated by LSD-SLAM should also be propagated to the detection algorithm. Also further research on the variances of the behaviour of the AR.Drone should be done. Now the magnetometer and the altimeter are not taken into account. This because the EKF was initially created for the Parrot AR.Drone 1.0 and consequently the study has not yet been made and their input would be useless for a state observer like an EKF.

8.4. Legal issues and moral development

National and international regulations are evolving and reacting to the development of UASs. A lot of regulations apply physical constraints to drone development regarding to size and weight but also limit the use cases. Autonomous flight and decision taking are not well regulated or in most cases prohibited. Thus when developing a platform one should take into account all these legal constraints or the result could be legally grounded afterwards. However,

[2] `https://www.mturk.com/mturk/welcome`

a lot of interest groups have been formed and there are several international fora where all the stakeholders discuss different legal matter.

The legal framework is not always complete and development is continuously pushing its borders. It is therefore important that the ones pushing these borders stay morally responsible. This not only because of their own work but also to keep a healthy community.

8.5. Human interface and multirobot set-up

Current development in certain niches is still limited to one drone operated by a specialised operator. However, to maximise the utility of UASs, non specialised operators should also be able to operate the platform. It is therefore of great importance that a solid and understandable human interface is built on the platform.

Another interesting topic is multirobot set-ups. These set-ups combine information of multiple drones and/or human actors and provide the fusioned data back to every actor. To make a UAS usable in such a case, communication between the actors is essential. There are non the less several use cases where communication channels are very limited in bandwidth or impossible to rely upon. It is therefore important to make the UAS partially stand-alone in such cases and to provide a robust framework for the communication that can happen.

Existing frameworks and standards like ROS, MOOS, and, JAUS, that provide communication are in most cases too verbose and they would saturate the communication channel in no-time. Another issue is the safety of such frameworks and standards. Most communication schemes are based on old strategies that do not take into account the security aspect. This is not only a problem in robot to robot communication but also in human-robot communication. A sound approach requires an authentication and key-exchange framework where every interaction is bilaterally authenticated. The overhead of this scheme should however be limited due to the aforementioned constraints on the communication channel.

Conclusion

By writing this thesis, it became clear to the authors that the research field of computer vision is a well established but still very flourishing one. Everyday new breakthroughs are published and experience is exchanged. By writing this thesis the authors hope that their experience can contribute to the development of future algorithms and development in general. Therefore their recommendations and findings are collected in this chapter.

A lot of existing algorithms have been used in this thesis in the attempt to detect as good as possible staircases. Because a lot of the algorithms already exist, the difficulty lies not so much in the development of new methods but rather in the successful fusion of existing ones. This is what this work tried to establish.

State estimation is a very important part in this work. Results can be very good with good feedback to external disturbances but generally the initialisation is cumbersome and a drift occurs while initialisation is done. This limits usability indoors as the confined spaces do not permit such a drift. Nevertheless, the EKF has proven its practical usability and enforces yet again its solid reputation.

The authors still believe that a unique combination of multiple algorithms is the key to success. The unique combination that is developed in this work shows great potential but requires more testing in different scenarios and especially on different platforms. The Parrot AR.Drone is a good drone for development but is not well suited for vision processing. Low frame rate and rolling shutter causes the SLAM algorithms to perform bad and this affects tremendously the quality of the 3D information. It is the crucial part of information that is needed to make a decision.

Despite the state of the art SLAM algorithms that are used in this thesis, detection is still not always successful. The biggest source of errors is the 3D information that contains too much noise. To improve the software in this thesis, new ways of generating a denser pointcloud should be looked at as discussed in section 8.2. GPU processing was not taken into account for this thesis because on-board computing was kept in mind. However more and more modern mobile platforms have an integrated GPU unit and may be capable to do the GPU-based calculations on-board. The google Tango project[3] for example shows great potential in this field.

Because the internal working of the AR.Drone is not accessible, an off-board EKF is used in this thesis. Obviously as discussed in subsection 8.3.2 it should be possible to implement the EKF and controller on-board. This eliminates the cumbersome delay handling and will improve the accuracy and stability. Especially when the computer was under full load during detection, the controller often became unstable because of the large delays. A new implementation would solve this issue.

[3] `https://www.google.com/atap/project-tango/`

The approaches discussed in chapter 7 have proven their use during the making of this thesis. Even when working alone, version management is greatly recommended as it gives the developer a way to reflect on the changes made. The usage of platform independent open tools ensures future expandability.

This thesis is a stepping stone towards a working real-time detection of staircases. However, a more general approach is recommended to be able to deal with a multitude of obstacles. The same framework could than also be used to do positive identification of objects nearby.

The code of this thesis is available on github[4] and visual results are posted on a youtube channel[5].

[4] `https://github.com/olivierdm/thesis2015`
[5] `https://www.youtube.com/channel/UCPZkTI9SgHYo5-yVoVRwC-g`

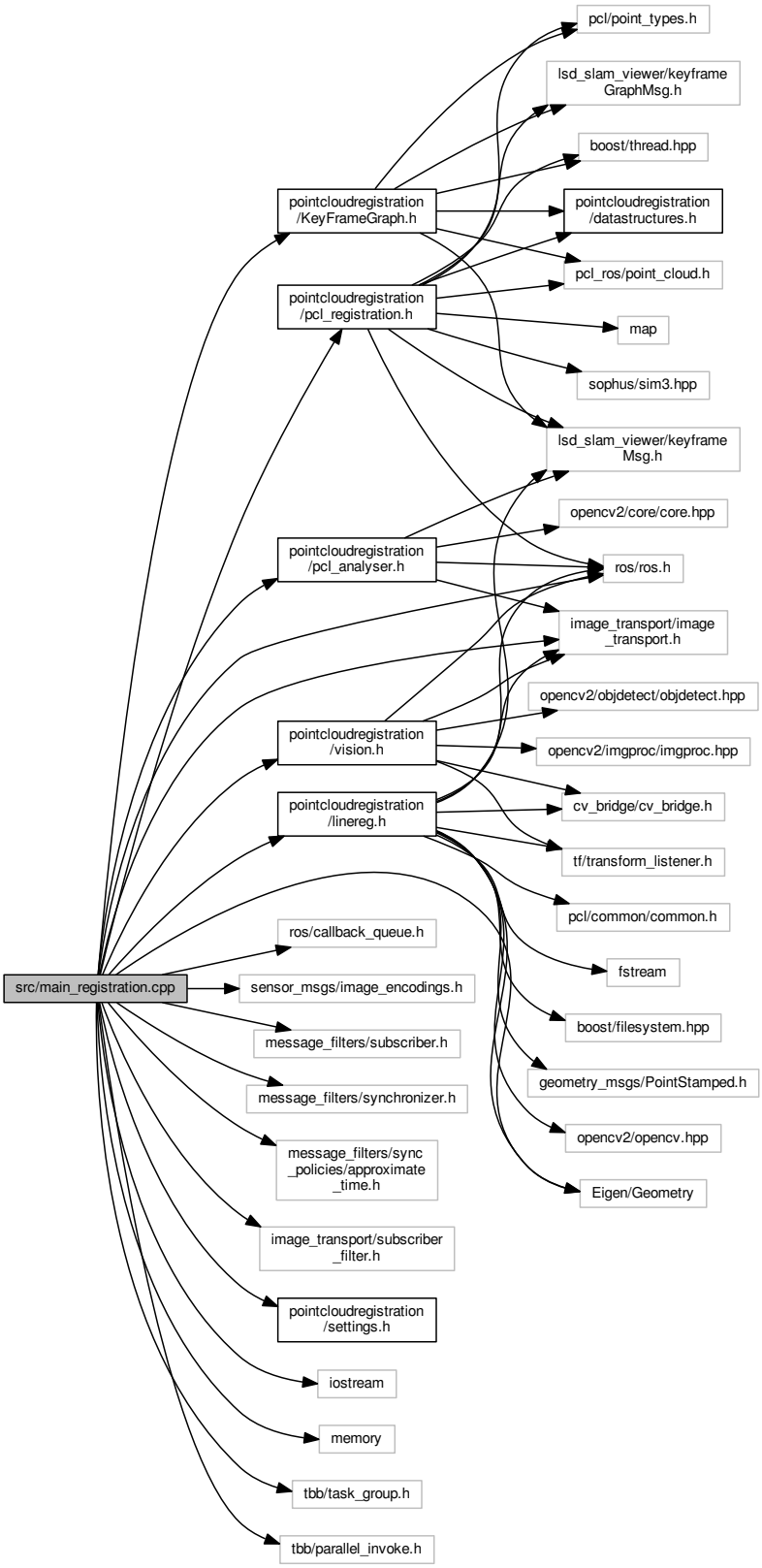
Appendices

A. Code documentation

A.1. src/main_registration.cpp File Reference

```
#include "ros/ros.h"
#include <ros/callback_queue.h>
#include <sensor_msgs/image_encodings.h>
#include <image_transport/image_transport.h>
#include <message_filters/subscriber.h>
#include <message_filters/synchronizer.h>
#include <message_filters/sync_policies/approximate_time.h>
#include <image_transport/subscriber_filter.h>
#include "pointcloudregistration/pcl_registration.h"
#include "pointcloudregistration/pcl_analyser.h"
#include "pointcloudregistration/KeyFrameGraph.h"
#include "pointcloudregistration/vision.h"
#include "pointcloudregistration/linereg.h"
#include "pointcloudregistration/settings.h"
#include <Eigen/Geometry>
#include <iostream>
#include <memory>
#include "tbb/task_group.h"
#include "tbb/parallel_invoke.h"
```

Include dependency graph for main_registration.cpp:



Functions

- void frameCb (lsd_slam_viewer::keyframeMsgConstPtr msg)
- void graphCb (lsd_slam_viewer::keyframeGraphMsgConstPtr msg)
- void callback (const sensor_msgs::ImageConstPtr &imgMsg, const lsd_slam_viewer::keyframeMsgConstPtr &frameMsg)
- int main (int argc, char **argv)

A.1.1. Function Documentation

void callback (const sensor_msgs::ImageConstPtr & *imgMsg*, const lsd_slam_viewer::keyframeMsgConstPtr & *frameMsg*)

Process the incoming data and detect stairs
Parameters

in	*imgMsg*	ros image msg from the ardrone front camera
in	*frameMsg*	lsdslam liveframe

Initialize variables that will be passed by reference.
Retrieve the image from the image message.
Get the keyframes from the graph and pass the data to the PCL_analyser. Image processing is done in parallel
Combine the data and attempt detection

void frameCb (lsd_slam_viewer::keyframeMsgConstPtr *msg*)

Handles the liveframes and the keyframes to construct the keyframe graph.
Parameters

in	*msg*	liveframe message or keyframe message

void graphCb (lsd_slam_viewer::keyframeGraphMsgConstPtr *msg*)

Handles graph messages to update the constraints and pose of the keyframes already in the graph.
Parameters

in	*msg*	keyframe graph message

int main (int *argc*, char ** *argv*)

Initialize the main program.
Parameters

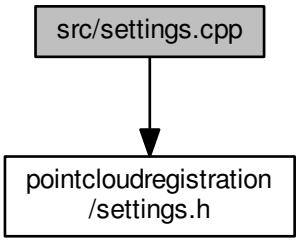
in	*argc*	number of arguments passed
in	*argv*	parameters passed to the main function, can contain parameters for ros initialisation

Initialise ros, this needs to be done before adding subscribers.
Initialise classes for handling the input.
Initialize queu and handler for lsdslam messages.
Initialize classes for processing.

Initialize queu and handler for processing

A.2. src/settings.cpp File Reference

`#include "pointcloudregistration/settings.h"`
Include dependency graph for settings.cpp:



Variables

- float minZ =0.1f

  Minimal distance to camera sensor in liveframe that will be considered.
- float maxZ =2.0f

  Maximal distance from camera sensor in liveframe that will be considered.
- float scaleDepthImage = 3.0f

  Factor that determines the size of depth image with respect to the visual image.
- float struct_x =3

  The size of the structuring element in x direction.
- float struct_y =2

  The size of the structuring element in y direction.
- unsigned int minLines = 5

  minimum number of lines to withhold candidate
- float minStairAngle =0.35f

  minimum angle to be a stair
- float maxStairAngle =1.3f

  maximum angle to be a stair
- float scaledDepthVarTH = 0.01

  if the variance of a point in the keyframe is lower than this value it is withheld
- int minNearSupport = 5

  the number of neighbours a point needs to be valid
- double TH =160000.0

  distance threshold ransac vanishing points
- double parTH =0.1

  rico threshold ransac parallel lines

- float eps =0.0001f

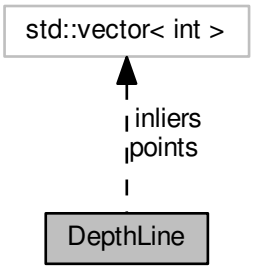
  tolerance used to calc maximum number of iterations
- double lastFrameTime = 1e15

  cut-off after this
- bool perfmon =true

  analyse performance

A.3. DepthLine Struct Reference

Collaboration diagram for DepthLine:

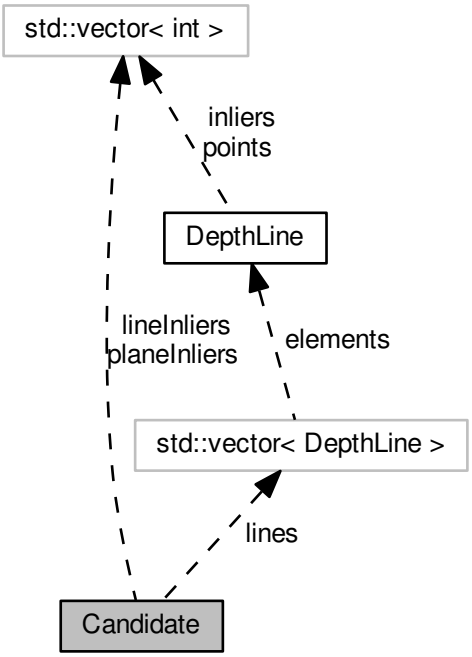


The documentation for this struct was generated from the following file:

- src/linereg.cpp

A.4. Candidate Struct Reference

Collaboration diagram for Candidate:



The documentation for this struct was generated from the following file:

- src/linereg.cpp

A.5. GraphFramePose Struct Reference

Contains the pose of keyframe identified by id.
`#include <include/pointcloudregistration/datastructures.h>`

Public Attributes

- int id

  keyframe identifier

A.5.1. Detailed Description

Contains the pose of keyframe identified by id.
The documentation for this struct was generated from the following file:

- include/pointcloudregistration/datastructures.h

A.6. InputPointDense Struct Reference

Struct used for casting the incoming point cloud data.
`#include <include/pointcloudregistration/datastructures.h>`

Public Attributes

- float idepth

  stores the inverse depth
- float idepth_var

  stores the variance of the inverse depth
- unsigned char color [4]

  array for casting color in RGBA

A.6.1. Detailed Description

Struct used for casting the incoming point cloud data.
The documentation for this struct was generated from the following file:

- include/pointcloudregistration/datastructures.h

A.7. framedist Struct Reference

Public Member Functions

- framedist (std::shared_ptr< KeyFrame > &key, Sophus::Sim3f &camToWorld)
- bool operator< (const framedist &rhs) const

  distance between the keyframe and the liveframe

Public Attributes

- Eigen::Matrix4f soph

  The homogenious coordinates of a point in meter in front of the camera center.

A.7.1. Constructor & Destructor Documentation

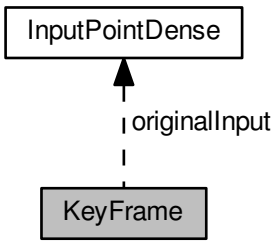
framedist::framedist (std::shared_ptr< KeyFrame > & *key*, Sophus::Sim3f & *camToWorld*) `[inline]`

Default constructor. Initializes the distance between the liveframe and this keyframe.
The documentation for this struct was generated from the following file:

- src/pcl_analyser.cpp

A.8. KeyFrame Class Reference

Collaboration diagram for KeyFrame:



Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW KeyFrame ()

  KeyFrame class.
- ∼KeyFrame ()
- void setFrom (lsd_slam_viewer::keyframeMsgConstPtr msg)
- PointCloud::Ptr getPCL ()

Public Attributes

- Sophus::Sim3f camToWorld

  camera pose, may be updated by kf-graph.

Private Member Functions

- void refreshPCL ()

Private Attributes

- float fx

  focal length in x direction
- float fy

  focal length in y direction
- float cx

  first ordinate of the camera's principal point
- float cy

  second ordinate of th camera's principal point
- int width

  image width
- int height

  image height

- PointCloud::Ptr cloud

  pointcloud data in pcl format, stays always in the local coordinates of the keyframe
- InputPointDense * originalInput

  datastructure for casting the incomming message

A.8.1. Constructor & Destructor Documentation

KeyFrame::KeyFrame ( )

KeyFrame class.
This class is used to store the received keyframes and their attributes. Default constructor, solely used to initialize private members.

KeyFrame::~KeyFrame ( )

Default destructor that clears the inputdata.

A.8.2. Member Function Documentation

PointCloud::Ptr KeyFrame::getPCL ( )

Passes back a pointer to the local cloud and locks the cloud for internal use.
The cloudMutex is locked and the pointer is passed. This method has a blocking behauviour if lock is optained by other local method. KeyFrame::release() can be used to unlock the lock.

void KeyFrame::refreshPCL ( ) [private]

Renders or rerenders the pcl cloud from the incoming data.
Checks if the local parameters still correspond to the global parameters. If not the pcl cloud is cleared and the input data is rendered in the pcl cloud.

void KeyFrame::setFrom (lsd_slam_viewer::keyframeMsgConstPtr *msg*)

Copies the data from the incoming message.
Copies the camera pose and camera parameters. Then it checks if the input is correctly sized and casts the pointcloud data.
Parameters

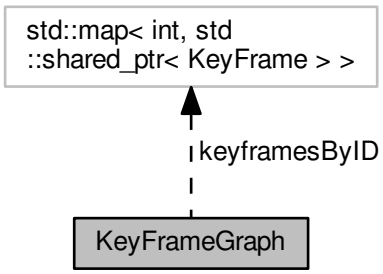
in	*msg*	keyframe message

The documentation for this class was generated from the following files:

- include/pointcloudregistration/KeyFrame.h
- src/KeyFrame.cpp

A.9. KeyFrameGraph Class Reference

Collaboration diagram for KeyFrameGraph:



Public Member Functions

- KeyFrameGraph ()
- virtual ~KeyFrameGraph ()
- void addMsg (lsd_slam_viewer::keyframeMsgConstPtr)
- void addGraphMsg (lsd_slam_viewer::keyframeGraphMsgConstPtr)
- std::vector< std::shared_ptr
  < KeyFrame > > getFrames ()

A.9.1. Constructor & Destructor Documentation

KeyFrameGraph::KeyFrameGraph ( )

default constructor

KeyFrameGraph::~KeyFrameGraph ( ) [virtual]

The default destructor cleans up the keyframes in the graph

A.9.2. Member Function Documentation

void KeyFrameGraph::addGraphMsg (lsd_slam_viewer::keyframeGraphMsgConstPtr *msg*)

Updates the graph constraints and updates the newly calculated poses.
Parameters

in	*msg*	keyframe graph message

void KeyFrameGraph::addMsg (lsd_slam_viewer::keyframeMsgConstPtr *msg*)

Adds a new keyframe to the graph
Parameters

in	*msg*	keyframe message

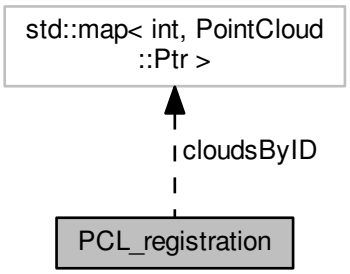
std::vector< std::shared_ptr< KeyFrame > > KeyFrameGraph::getFrames ( )

Returns an array of pointers to the current set of keyframes.
The documentation for this class was generated from the following files:

- include/pointcloudregistration/KeyFrameGraph.h
- src/KeyFrameGraph.cpp

A.10. PCL_registration Class Reference

Collaboration diagram for PCL_registration:



Public Member Functions

- PCL_registration (std::shared_ptr< KeyFrameGraph > &)
- virtual ~PCL_registration ()
- bool addFrameMsg (lsd_slam_viewer::keyframeMsgConstPtr)
- void addGraphMsg (lsd_slam_viewer::keyframeGraphMsgConstPtr)
- void drawPlane (const pcl::PointCloud< pcl::PointXYZ >::Ptr &, const Eigen::Affine3f &)

Private Member Functions

- void visualiserThread ()
- void eraseClouds ()

A.10.1. Constructor & Destructor Documentation

PCL_registration::PCL_registration (std::shared_ptr< KeyFrameGraph > & *keyGraph*)

default constructor, initialize visualiser thread

PCL_registration::~PCL_registration ( ) `[virtual]`

default destructor, shuts down the visualiser thread.

A.10.2. Member Function Documentation

bool PCL_registration::addFrameMsg (lsd_slam_viewer::keyframeMsgConstPtr *msg*)

Add Keyframe or check if lsdslam is ok with liveframe. Returns
Returns true if jump detected.

void PCL_registration::addGraphMsg (lsd_slam_viewer::keyframeGraphMsgConstPtr *msg*)

Pass the graph message to the keyframe graph.

void PCL_registration::drawPlane (const pcl::PointCloud< pcl::PointXYZ >::Ptr & *plane*, const Eigen::Affine3f & *pose*)

Add plane with plane inliers within the EKF pose.

void PCL_registration::eraseClouds ( ) [private]

erase clouds in the viewer

void PCL_registration::visualiserThread ( ) [private]

function that initialises viewer and shows updates
The documentation for this class was generated from the following files:

- include/pointcloudregistration/pcl_registration.h
- src/pcl_registration.cpp

A.11. PCL_analyser Class Reference

Public Member Functions

- void operator() (const lsd_slam_viewer::keyframeMsgConstPtr &, std::vector< std-::shared_ptr< KeyFrame >> &, cv::UMat &, cv::UMat &, cv::UMat &)

Private Member Functions

- void calcCurvature (const cv::UMat &, const float &, const float &, const int &, cv::UMat &, cv::UMat &, cv::UMat &)
- void getDepthImage (std::vector< framedist > &, const float &, const float &, const float &, const float &, cv::UMat &)
- void filterDepth (cv::UMat &, const int &, cv::UMat &)
- void writeHist (const float &, const float &, const int &, const cv::UMat &)

A.11.1. Member Function Documentation

void PCL_analyser::calcCurvature (const cv::UMat & *filt*, const float & *fx*, const float & *fy*, const int & *my_scaleDepthImage*, cv::UMat & *H*, cv::UMat & *K*, cv::UMat & *CI*) [private]

Calculate the curvature using the constructed and filtered depth image.

Parameters

`in`	*filt*	filtered depth image
`in`	*fx*	focal length in x direction
`in`	*fy*	focal length in y direction
`in`	*my_scaleDepth-Image*	the scaling applied when projecting the image
`out`	*H*	mean curvature
`out`	*K*	Gaussian curvature
`out`	*CI*	image containing pixelwise curvature index

Create kernels for calculating the different derivatives with convolutive filters
Apply the filter kernels to the image.
Normalize derivatives to get uniform dx en dy.
calculate K

1+hx^2 + hy^2

hxx∗hyy-hxy^2

calculate H

1+hx^2

2∗hx∗hy∗hxy

1+hy^2

calculate CI

H^2 - K

H-eps

H-eps-K

void PCL_analyser::filterDepth (cv::UMat & *depthImg*, const int & *my_scaleDepthImage*, cv::UMat & *filt*) `[private]`

applies required filtering and resizes the image back to the original size.
Parameters

`in`	*depthImg*	The depth image generated by combining different keyframes.
`in`	*my_scaleDepth-Image*	the scaling applied when projecting the image
`out`	*filt*	the filtered depth image

Gaussian smoothing is applied to make the derivatis stable.

void PCL_analyser::getDepthImage (std::vector< framedist > & *mykeyframes*, const float & *fx*, const float & *fy*, const float & *cx*, const float & *cy*, cv::UMat & *depthImgu*) [private]

Generates a depth image from the received keyframes.
Parameters

in	*keyframes*	all the currently captured keyframes
in	*fx*	focal length in x direction
in	*fy*	focal length in y direction
in	*cx*	first ordinate of the camera's principal point
in	*cy*	second ordinate of the camera's principal point
out	*depthImg*	the generated depth image

The function gets the currently received keyframes from the keyFrameGraph instance and transforms them to coordinates in the liveframe axes. This set of accumulated points is then projected using the camera parameters to get a depth image that corresponds with the image captured by camera.
Get the keyframes closest to the liveframe.
Get each keyframe in liveframe pose.
Project the accumulated cloud to a 2D image.

void PCL_analyser::operator() (const lsd_slam_viewer::keyframeMsgConstPtr & *msg*, std::vector< std::shared_ptr< KeyFrame >> & *keyframes*, cv::UMat & *depthImg*, cv::UMat & *H*, cv::UMat & *CI*)

Get the depth image and the curvature in the passed liveframe.
Parameters

in	*msg*	keyframe message
in	*keyframes*	vector containing smart pointers to the keyframes
in	*filt*	filtered depth image
in	*H*	mean curvature
out	*CI*	image containing pixelwise curvature index

Get camera parameters from the liveframe.
Initialize the vector that permits sorting the keyframes by distance to the camera.

void PCL_analyser::writeHist (const float & *min*, const float & *max*, const int & *bins*, const cv::UMat & *CI*) [private]

Writes histogram to csv file for debug purposes.
Parameters

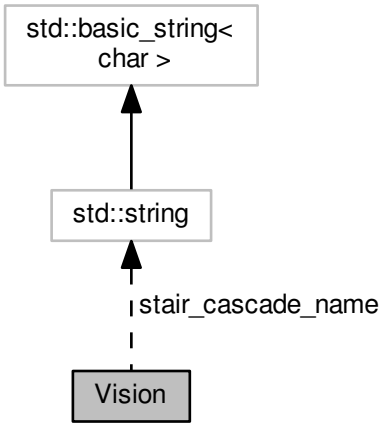
in	*min*	the minimal value in the histogram
in	*max*	the maximum value in the histogram
in	*bins*	the number of bins in the histogram
in	*CI*	image describing the pixelwise curvature

The documentation for this class was generated from the following files:

- include/pointcloudregistration/pcl_analyser.h
- src/pcl_analyser.cpp

A.12. Vision Class Reference

Collaboration diagram for Vision:



Public Member Functions

- Vision ()
- void operator() (const cv_bridge::CvImagePtr &, std::vector< cv::Rect > &, std-::vector< cv::Vec4f > &)

Private Member Functions

- void getLines (const cv_bridge::CvImagePtr &, cv::Mat &, std::vector< cv::Vec4f > &)
- void detect (const cv_bridge::CvImagePtr &, cv::Mat &, std::vector< cv::Rect > &)

A.12.1. Constructor & Destructor Documentation

Vision::Vision ( )

default constructor, sets up the cascade classifier

A.12.2. Member Function Documentation

void Vision::detect (const cv_bridge::CvImagePtr & *cv_input_ptr*, cv::Mat & *inputGray*, std::vector< cv::Rect > & *rectangles*) [private]

Detect stairs using a cascade classifier.
Parameters

in	*cv_input_ptr*	cv_bridge image coming from the front camera
in	*inputGray*	grayscale version of the input image
out	*rectangles*	the zones detected by the object detector

Detect rectangles.
Draw rectangles on the output image.

void Vision::getLines (const cv_bridge::CvImagePtr & *cv_input_ptr*, cv::Mat & *inputGray*, std::vector< cv::Vec4f > & *lines*) [private]

Detects lines using the line segment detector implemented in OpenCV.
Parameters

in	*cv_input_ptr*	cv_bridge image coming from the front camera
in	*inputGray*	grayscale version of the input image
out	*lines*	vector containing the lines

Detect using lsd.
Get transform
acess roll pitch and yaw
Draw lines on the output image.

void Vision::operator() (const cv_bridge::CvImagePtr & *cv_input_ptr*, std::vector< cv::Rect > & *rectangles*, std::vector< cv::Vec4f > & *lines*)

Accepts the data of an image and copies the neccesary data to the adequate private variables.
Parameters

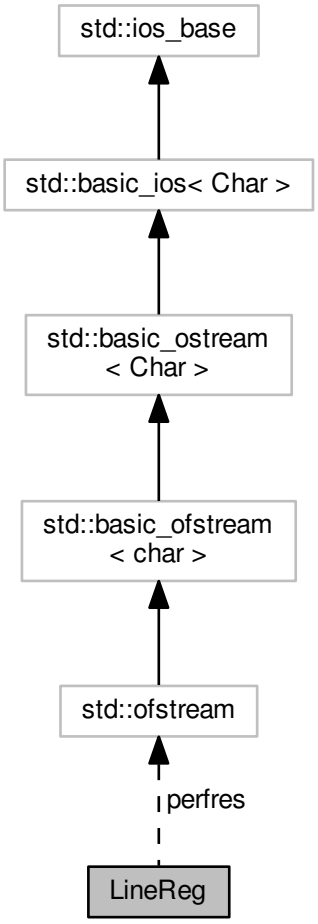
in	*cv_input_ptr*	cv_bridge image coming from the front camera
out	*rectangles*	the zones detected by the object detector
out	*lines*	vector containing the lines

Convert input to grayscale
Detect lines
Perform object detection
The documentation for this class was generated from the following files:

- include/pointcloudregistration/vision.h
- src/vision.cpp

A.13. LineReg Class Reference

Collaboration diagram for LineReg:



Public Member Functions

- bool operator() (cv::UMat, cv::UMat, cv::UMat, std::vector< cv::Rect >, std::vector< cv::Vec4f >, cv_bridge::CvImagePtr, const lsd_slam_viewer::keyframeMsgConstPtr, pcl::PointCloud< pcl::PointXYZ >::Ptr &, Eigen::Affine3f &)
- bool SegmentIntersectRectangle (cv::Rect &, cv::Vec4f &)

Private Member Functions

- void getParallelLines (Candidate &, std::vector< DepthLine > &)
- void get3DLines (Candidate &, const cv::Mat &, const cv::Mat &, const cv::Mat &, float &, float &, float &, float &)
- void getPlane (Candidate &, cv::Mat &, cv::Mat &, const Eigen::Affine3d &)

A.13.1. Member Function Documentation

void LineReg::get3DLines (Candidate & *can*, const cv::Mat & *depthImg*, const cv::Mat & *curv_weight*, const cv::Mat & *meanCurvature*, float & *fxi*, float & *fyi*, float & *cxi*, float & *cyi*) [private]

get 3D points from lines and depthImage.
Parameters

in	*can*	candidate with initialised rectangle
in	*depthImg*	filtered depth image
in	*curv_weight*	weighted curvature
in	*meanCurvature*	mean curvature
in	*fxi*	inverse focal length in x direction
in	*fyi*	inverse focal length in y direction
in	*cxi*	inverse first ordinate of the camera's principal point
in	*cyi*	inverse second ordinate of the camera's principal point

set line properties
created RandomSampleConsensus object and compute the line model for each line

void LineReg::getParallelLines (Candidate & *can*, std::vector< DepthLine > & *depthLines*) [private]

Get the lines that are parallel with each other
get lines that intersect with rectangle
calculate parameters of each line
get maybeinliers from set
shift set every iteration
get model of maybeinliers
add maybeinliers to alsoinliers
add elements from set to alsoinliers if intersection with one of the maybeinliers is close enough to the model
check if there are enough inliers
get all intersections
get the mean intersection

void LineReg::getPlane (Candidate & *can*, cv::Mat & *linesImg*, cv::Mat & *canImg*, const Eigen::Affine3d & *trans*) [private]

Do robust estimation from the plane that virtually lays on the staircase
Parameters

in	*can*	candidate containing 3D-lines
out	*linesImg*	output image containing lines with curvature
out	*canImg*	output image containing the candidates with extra info
in	*trans*	rigid body transform between camera and ekf

bool LineReg::operator() (cv::UMat *depthImg*, cv::UMat *H*, cv::UMat *CI*, std::vector< cv::Rect > *rectangles*, std::vector< cv::Vec4f > *lines*, cv_bridge::CvImagePtr *cv_input_ptr*, const lsd_slam_viewer::keyframeMsgConstPtr *frameMsg*, pcl::PointCloud< pcl::PointXYZ >::Ptr & *planeCloud*, Eigen::Affine3f & *pose*)

Accepts the data and fusions it to expell candidates
Parameters

in	*depthImg*	a filtered version of the rendered depth image
in	*H*	matrix with the mean curvature
in	*CI*	matrix containing the curvature in each point
in	*rectangles*	vector containing all the candidates
in	*lines*	set of lines that are detected on the visual image
in	*cv_input_ptr*	the colored input image
in	*frameMsg*	lsdslame liveframe

Returns
True if there are candidates detected, false if not Get inverse camera parameters

initialize output images
construct transformation matrix

bool LineReg::SegmentIntersectRectangle (cv::Rect & *rectangle*, cv::Vec4f & *line*)

Check if a line intersects with a rectangle.
Parameters

in	*rectangle*	a rectangle defined by origin and size
in	*line*	line defined by begin and endpoint

Find min and max X for the segment
Find the intersection of the segment's and rectangle's x-projections
If their projections do not intersect return false
Find corresponding min and max Y for min and max X we found before
Find the intersection of the segment's and rectangle's y-projections
If Y-projections do not intersect return false
The documentation for this class was generated from the following files:

- include/pointcloudregistration/linereg.h
- src/linereg.cpp

Code index

B. Software changes

B.1. tum_ardrone

```
diff --git a/CMakeLists.txt b/CMakeLists.txt
index 723ec9d..f0252db 100644
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -145,7 +145,8 @@ add_executable(drone_stateestimation ${STATEESTIMATION_SOURCE_FILES
    } ${STATEESTI
 set_target_properties(drone_stateestimation PROPERTIES COMPILE_FLAGS "-D_LINUX -
    D_REENTRANT -Wall  -O3 -march=nocona -msse3")
 target_link_libraries(drone_stateestimation ${PTAM_LIBRARIES} ${catkin_LIBRARIES})
 add_dependencies(drone_stateestimation thirdparty ${PROJECT_NAME}_gencpp ${
    PROJECT_NAME}_gencfg)
-
+# Install Rules
+install(TARGETS drone_stateestimation RUNTIME DESTINATION ${
    CATKIN_PACKAGE_BIN_DESTINATION})
 # ------------------------- autopilot & KI -----------------------------------------
 # set header ans source files
 set(AUTOPILOT_SOURCE_FILES
@@ -169,7 +170,8 @@ set(AUTOPILOT_HEADER_FILES
 add_executable(drone_autopilot ${AUTOPILOT_SOURCE_FILES} ${AUTOPILOT_HEADER_FILES})
 target_link_libraries(drone_autopilot ${catkin_LIBRARIES})
 add_dependencies(drone_autopilot thirdparty ${PROJECT_NAME}_gencpp ${PROJECT_NAME}
    _gencfg)
-
+# Install Rules
+install(TARGETS drone_autopilot RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
 # ------------------------------ GUI -----------------------------------------------
 # set header ans source files
 set(GUI_SOURCE_FILES
@@ -208,4 +210,9 @@ add_executable(drone_gui ${GUI_SOURCE_FILES} ${
    GUI_RESOURCE_FILES_CPP} ${GUI_UI_
 target_link_libraries(drone_gui ${QT_LIBRARIES} cvd ${catkin_LIBRARIES})
 add_dependencies(drone_gui thirdparty ${PROJECT_NAME}_gencpp ${PROJECT_NAME}_gencfg)

-
+# Install Rules
+install(TARGETS drone_gui RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
+install(DIRECTORY launch/
+    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/launch
+    PATTERN ".git" EXCLUDE
+)
diff --git a/camcalib/ardrone2_default (copy).txt.bak b/camcalib/ardrone2_default (copy
    ).txt.bak
new file mode 100644
index 0000000..0f3a518
--- /dev/null
+++ b/camcalib/ardrone2_default (copy).txt.bak
@@ -0,0 +1 @@
+0.771557 1.368560 0.552779 0.444056 1.156010
diff --git a/camcalib/ardrone2_default.txt b/camcalib/ardrone2_default.txt
index 0f3a518..da69462 100644
--- a/camcalib/ardrone2_default.txt
+++ b/camcalib/ardrone2_default.txt
@@ -1 +1 @@
-0.771557 1.368560 0.552779 0.444056 1.156010
+0.766313 1.41334 0.529179 0.517397 1.15136
\ No newline at end of file
diff --git a/launch/ardrone_driver.launch b/launch/ardrone_driver.launch
```

```
index 0dec2f1..021e54f 100644
--- a/launch/ardrone_driver.launch
+++ b/launch/ardrone_driver.launch
@@ -1,9 +1,10 @@
 <launch>
-  <arg name="droneip" default="192.168.1.1" />
+  <arg name="droneip" default="192.168.2.165" />
   <node name="ardrone_driver" pkg="ardrone_autonomy" type="ardrone_driver" output="
       screen" args="-ip $(arg droneip)">
     <param name="navdata_demo" value="False" />
     <param name="realtime_navdata" value="True" />
     <param name="realtime_video" value="True" />
     <param name="looprate" value="30" />
+    <param name="altitude_max" value="10000" />
   </node>
 </launch>
diff --git a/src/UINode/PingThread.cpp b/src/UINode/PingThread.cpp
index 157d44b..aaad0fb 100644
--- a/src/UINode/PingThread.cpp
+++ b/src/UINode/PingThread.cpp
@@ -25,6 +25,11 @@
 #include <stdio.h>
 #include "RosThread.h"
 #include "tum_ardrone_gui.h"
+#include <fstream>
+#include <opencv2/opencv.hpp>
+#include "opencv2/core/utility.hpp"
+#include <opencv2/imgproc/imgproc.hpp>
+

 PingThread::PingThread()
 {
@@ -77,8 +82,8 @@ void PingThread::run()
 {
        std::cout << "Starting PING Thread" << std::endl;

-       sprintf(pingCommand20000,"ping -c 1 -s 20000 -w 1 192.168.1.1");
-       sprintf(pingCommand500,"ping -c 1 -s 500 -w 1 192.168.1.1");
+       sprintf(pingCommand20000,"ping -c 1 -s 20000 -w 1 192.168.2.165");
+       sprintf(pingCommand500,"ping -c 1 -s 500 -w 1 192.168.2.165");
        ros::Rate r(2.0);
        FILE *p;

@@ -111,6 +116,13 @@ void PingThread::run()
                        double res20000 = parsePingResult(line2);

                        std::cout << "new ping values: 500->" << res500 << " 20000->"
                            << res20000 << std::endl;
+
+                       /* Write ping values to csv file */
+                       std::ofstream myfile;
+                       myfile.open ("histogram.csv",std::ofstream::out | std::ofstream
    ::app);
+                       myfile << res500 << ", " << res20000 << std::endl;
+                       myfile.close();
+                       //*/

                        // clip between 10 and 1000.
                        res500 = std::min(1000.0,std::max(10.0,res500));
diff --git a/src/UINode/tum_ardrone_gui.cpp b/src/UINode/tum_ardrone_gui.cpp
index 84a97a1..6ae6ec1 100644
--- a/src/UINode/tum_ardrone_gui.cpp
+++ b/src/UINode/tum_ardrone_gui.cpp
@@ -319,26 +319,28 @@ int tum_ardrone_gui::mapKey(int k)
 {
        switch(k)
        {
-               case 74: //j
+               // http://api.kde.org/qyoto-api/class_qt_core_1_1_qt.html#
    a632272b1f891fdbe394baf8babe399fa
+               case 83: //s
```

```
                        return 0;
-               case 75: //k
+               case 68: //d
                        return 1;
-               case 76: //l
+               case 70: //f
                        return 2;
-               case 73: //i
+               case 69: //e
                        return 3;
-               case 85: //u
+               case 74: //j
                        return 4;
-               case 79: //o
+               case 76: //l
                        return 5;
-               case 81: //q
+               case 73: //i
                        return 6;
-               case 65: //a
+               case 75: //k
                        return 7;
        }
        return -1;
 }

+
 void tum_ardrone_gui::keyReleaseEvent( QKeyEvent * key)
 {
        if(currentControlSource == CONTROL_KB)
@@ -376,10 +378,10 @@ void tum_ardrone_gui::keyPressEvent( QKeyEvent * key)
                                rosThread->sendControlToDrone(calcKBControl());
                }

-               else if(key->key() == 83)       // s
+               else if(key->key() == 84)       // t
                        rosThread->sendTakeoff();

-               else if(key->key() == 68)       // d
+               else if(key->key() == 89)       // y
                        rosThread->sendLand();
        }

@@ -405,14 +407,14 @@ ControlCommand tum_ardrone_gui::calcKBControl()

        ControlCommand c;

-       if(isPressed[0]) c.roll = -sensRP; // j
-       if(isPressed[1]) c.pitch = sensRP; // k
-       if(isPressed[2]) c.roll = sensRP; // l
-       if(isPressed[3]) c.pitch = -sensRP; // i
-       if(isPressed[4]) c.yaw = -sensYaw; // u
-       if(isPressed[5]) c.yaw = sensYaw; // o
-       if(isPressed[6]) c.gaz = sensRP; // q
-       if(isPressed[7]) c.gaz = -sensRP; // a
+       if(isPressed[0]) c.roll = -sensRP; // s
+       if(isPressed[1]) c.pitch = sensRP; // d
+       if(isPressed[2]) c.roll = sensRP; // f
+       if(isPressed[3]) c.pitch = -sensRP; // e
+       if(isPressed[4]) c.yaw = -sensYaw; // j
+       if(isPressed[5]) c.yaw = sensYaw; // l
+       if(isPressed[6]) c.gaz = sensRP; // i
+       if(isPressed[7]) c.gaz = -sensRP; // k

        return c;
 }
diff --git a/src/stateestimation/MapView.cpp b/src/stateestimation/MapView.cpp
index 2f43b30..a779034 100644
--- a/src/stateestimation/MapView.cpp
+++ b/src/stateestimation/MapView.cpp
@@ -27,6 +27,7 @@
```

```
 #include "DroneKalmanFilter.h"
 #include "PTAMWrapper.h"
 #include "EstimationNode.h"
+#include <tf/transform_broadcaster.h>

 pthread_mutex_t MapView::trailPointsVec_CS = PTHREAD_MUTEX_INITIALIZER; //
     pthread_mutex_lock( &cs_mutex );

@@ -95,7 +96,11 @@ void MapView::Render()
         lastFramePoseSpeed = filter->getCurrentPoseSpeedAsVec();        // Note: this
             is maybe an old pose, but max. one frame old = 50ms = not noticable.
         pthread_mutex_unlock(&filter->filter_CS);

-
+        tf::TransformBroadcaster br_ardrone;
+        tf::Transform transform_ardrone;
+    transform_ardrone.setOrigin( tf::Vector3(lastFramePoseSpeed[1], -
    lastFramePoseSpeed[0], lastFramePoseSpeed[2]) );
+    transform_ardrone.setRotation( tf::createQuaternionFromRPY(lastFramePoseSpeed
    [3]/180*3.141592, -lastFramePoseSpeed[4]/180*3.141592, -lastFramePoseSpeed
    [5]/180*3.141592) );
+    br_ardrone.sendTransform(tf::StampedTransform(transform_ardrone, ros::Time::now(),
     "map", "tum_base_link"));


         if(clearTrail)
```

B.2. ardrone_autonomy

```
diff --git a/.gitignore b/.gitignore
index 76e32af..3f167ad 100644
--- a/.gitignore
+++ b/.gitignore
@@ -2,6 +2,7 @@
 *.slo
 *.lo
 *.o
+*.*~

 # Project Files
 *.sublime-workspace
diff --git a/include/ardrone_autonomy/video.h b/include/ardrone_autonomy/video.h
index 76ad841..bcae543 100644
--- a/include/ardrone_autonomy/video.h
+++ b/include/ardrone_autonomy/video.h
@@ -56,7 +56,7 @@ ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
    OF THE POSSI

 // NO PIP, Both camera streams provide the same reseloution: Simple!
 #define D2_STREAM_WIDTH 640
-#define D2_STREAM_HEIGHT 360
+#define D2_STREAM_HEIGHT 352

 extern video_com_multisocket_config_t icc;
 extern const vp_api_stage_funcs_t vp_stages_export_funcs;
diff --git a/src/ardrone_driver.cpp b/src/ardrone_driver.cpp
index e60b29f..50186ce 100644
--- a/src/ardrone_driver.cpp
+++ b/src/ardrone_driver.cpp
@@ -499,7 +499,7 @@ void ARDroneDriver::PublishVideo()
     image_msg.step = D2_STREAM_WIDTH * 3;
     image_msg.data.resize(D2_STREAM_WIDTH * D2_STREAM_HEIGHT * 3);
     if (!realtime_video) vp_os_mutex_lock(&video_lock);
-    std::copy(buffer, buffer + (D2_STREAM_WIDTH * D2_STREAM_HEIGHT * 3), image_msg.
    data.begin());
+    std::copy(buffer + 4*3*D2_STREAM_WIDTH, buffer + 4*3*D2_STREAM_WIDTH + (
    D2_STREAM_WIDTH * D2_STREAM_HEIGHT * 3), image_msg.data.begin());
     if (!realtime_video) vp_os_mutex_unlock(&video_lock);
     // We only put the width and height in here.
```

B.3. rqt_image_view

```
diff --git a/rqt_image_view/src/rqt_image_view/image_view.cpp b/rqt_image_view/src/
    rqt_image_view/image_view.cpp
index e68c224..8392baf 100644
--- a/rqt_image_view/src/rqt_image_view/image_view.cpp
+++ b/rqt_image_view/src/rqt_image_view/image_view.cpp
@@ -282,7 +282,7 @@ void ImageView::callbackImage(const sensor_msgs::Image::ConstPtr&
    msg)
          conversion_mat_ = cv_ptr->image;
        } else if (msg->encoding == "8UC1") {
          // convert gray to rgb
-         cv::cvtColor(cv_ptr->image, conversion_mat_, CV_GRAY2RGB);
+         cv::cvtColor(cv_ptr->image, conversion_mat_, cv::COLOR_GRAY2RGB);
        } else if (msg->encoding == "16UC1" || msg->encoding == "32FC1") {
          // scale / quantify
          double min = 0;
@@ -300,7 +300,7 @@ void ImageView::callbackImage(const sensor_msgs::Image::ConstPtr&
    msg)
          }
          cv::Mat img_scaled_8u;
          cv::Mat(cv_ptr->image-min).convertTo(img_scaled_8u, CV_8UC1, 255. / (max - min
              ));
-         cv::cvtColor(img_scaled_8u, conversion_mat_, CV_GRAY2RGB);
+         cv::cvtColor(img_scaled_8u, conversion_mat_, cv::COLOR_GRAY2RGB);
        } else {
          qWarning("ImageView.callback_image() could not convert image from '%s' to '
              rgb8' (%s)", msg->encoding.c_str(), e.what());
          ui_.image_frame->setImage(QImage());
diff --git a/rqt_marble/CMakeLists.txt b/rqt_marble/CMakeLists.txt
deleted file mode 100644
index 6442f44..0000000
diff --git a/rqt_marble/include/rqt_marble/bridge_ros_marble.h b/rqt_marble/include/
    rqt_marble/bridge_ros_marble.h
deleted file mode 100644
index 3a8c616..0000000
diff --git a/rqt_marble/include/rqt_marble/marble_plugin.h b/rqt_marble/include/
    rqt_marble/marble_plugin.h
deleted file mode 100644
index e329a36..0000000
diff --git a/rqt_marble/msg/RouteGps.msg b/rqt_marble/msg/RouteGps.msg
deleted file mode 100644
index af0fd5e..0000000
diff --git a/rqt_marble/package.xml b/rqt_marble/package.xml
deleted file mode 100644
index 96b94cf..0000000
diff --git a/rqt_marble/plugin.xml b/rqt_marble/plugin.xml
deleted file mode 100644
index 8256d92..0000000
diff --git a/rqt_marble/resource/marble_plugin.ui b/rqt_marble/resource/marble_plugin.
    ui
deleted file mode 100644
index eb2ef7b..0000000
diff --git a/rqt_marble/scripts/rqt_marble b/rqt_marble/scripts/rqt_marble
deleted file mode 100755
index 88f810b..0000000
diff --git a/rqt_marble/setup.py b/rqt_marble/setup.py
deleted file mode 100644
index 811cff2..0000000
diff --git a/rqt_marble/src/rqt_marble/bridge_ros_marble.cpp b/rqt_marble/src/
    rqt_marble/bridge_ros_marble.cpp
deleted file mode 100644
index bb49473..0000000
diff --git a/rqt_marble/src/rqt_marble/marble_plugin.cpp b/rqt_marble/src/rqt_marble/
    marble_plugin.cpp
deleted file mode 100644
index b1013ce..0000000
diff --git a/rqt_marble/test/test_bridge_ros_marble.cpp b/rqt_marble/test/
    test_bridge_ros_marble.cpp
deleted file mode 100644
```

```
index 5c7c478..0000000
diff --git a/rqt_marble/test/utest.test b/rqt_marble/test/utest.test
deleted file mode 100644
index 553522a..0000000
```

B.4. ORB-SLAM

```
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..48cfb84
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,5 @@
+build/
+Thirdparty/
+bin/
+ORBvoc.yml
+KeyFrameTrajectory.txt
diff --git a/ExampleFuerte.launch b/ExampleFuerte.launch
deleted file mode 100644
index 48af8f6..0000000
diff --git a/ExampleGroovyHydro.launch b/ExampleGroovyHydro.launch
index 9c9be87..1adf62d 100644
--- a/ExampleGroovyHydro.launch
+++ b/ExampleGroovyHydro.launch
@@ -1,6 +1,6 @@
 <launch>

-   <node pkg="image_view" type="image_view" name="image_view" respawn="false" output="
    log">
+   <node pkg="rqt_image_view" type="rqt_image_view" name="rqt_image_view" respawn="
    false" output="log">
        <remap from="/image" to="/ORB_SLAM/Frame" />
        <param name="autosize" value="true"/>
        </node>
diff --git a/ExampleGroovyHydro.launch~ b/ExampleGroovyHydro.launch~
new file mode 100644
index 0000000..9c9be87
--- /dev/null
+++ b/ExampleGroovyHydro.launch~
@@ -0,0 +1,14 @@
+<launch>
+
+   <node pkg="image_view" type="image_view" name="image_view" respawn="false" output="
    log">
+       <remap from="/image" to="/ORB_SLAM/Frame" />
+       <param name="autosize" value="true"/>
+       </node>
+
+       <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ORB_SLAM)/Data/rviz.
    rviz" output="log">
+   </node>
+
+       <node pkg="ORB_SLAM" type="ORB_SLAM" name="ORB_SLAM"  args="Data/ORBvoc.yml
    Data/Settings.yaml" cwd="node" output="screen">
+   </node>
+
+</launch>
diff --git a/manifest.xml b/manifest.xml
index c98fa97..a01ffd8 100644
--- a/manifest.xml
+++ b/manifest.xml
@@ -9,7 +9,6 @@
   <depend package="roscpp"/>
   <depend package="tf"/>
   <depend package="sensor_msgs"/>
-  <depend package="opencv2"/>
   <depend package="image_transport"/>
   <depend package="cv_bridge"/>
```

```
diff --git a/src/FramePublisher.cc b/src/FramePublisher.cc
index f84da10..a1cef0c 100644
--- a/src/FramePublisher.cc
+++ b/src/FramePublisher.cc
@@ -156,7 +156,7 @@ void FramePublisher::DrawTextInfo(cv::Mat &im, int nState, cv::Mat
    &imText)
 {
     stringstream s;
     if(nState==Tracking::NO_IMAGES_YET)
-        s << "WAITING FOR IMAGES. (Topic: /camera/image_raw)";
+        s << "WAITING FOR IMAGES. (Topic: /ardrone/front/image_raw)";
     else if(nState==Tracking::NOT_INITIALIZED)
         s << " NOT INITIALIZED ";
     else if(nState==Tracking::INITIALIZING)
diff --git a/src/ORBextractor.cc b/src/ORBextractor.cc
index 0b334cf..987424a 100644
--- a/src/ORBextractor.cc
+++ b/src/ORBextractor.cc
@@ -57,6 +57,7 @@

 #include <opencv2/core/core.hpp>
 #include <opencv2/highgui/highgui.hpp>
+#include <opencv2/opencv.hpp>
 #include <vector>

 #include "ORBextractor.h"
diff --git a/src/Tracking.cc b/src/Tracking.cc
index 6af7400..204da4a 100644
--- a/src/Tracking.cc
+++ b/src/Tracking.cc
@@ -160,7 +160,7 @@ void Tracking::SetKeyFrameDatabase(KeyFrameDatabase *pKFDB)
 void Tracking::Run()
 {
     ros::NodeHandle nodeHandler;
-    ros::Subscriber sub = nodeHandler.subscribe("/camera/image_raw", 1, &Tracking::
    GrabImage, this);
+    ros::Subscriber sub = nodeHandler.subscribe("/ardrone/front/image_raw", 1, &
    Tracking::GrabImage, this);

     ros::spin();
 }
```

B.5. LSD-SLAM

```
diff --git a/.gitignore b/.gitignore
index 9793fb9..8805ab8 100644
--- a/.gitignore
+++ b/.gitignore
@@ -25,8 +25,8 @@ lsd_slam_viewer/.pydevproject
 lsd_slam_viewer/thirdparty/Sophus/CMakeFiles/
 lsd_slam_viewer/thirdparty/Sophus/Makefile
 lsd_slam_viewer/thirdparty/Sophus/SophusConfig.cmake
-
+*build_codeblocks*
 cmake_install.cmake
 *.cfgc
-
-
+*.*~
+.qglviewer.xml
diff --git a/lsd_slam_core/CMakeLists.txt b/lsd_slam_core/CMakeLists.txt
index 7b4f8f2..af78294 100644
--- a/lsd_slam_core/CMakeLists.txt
+++ b/lsd_slam_core/CMakeLists.txt
@@ -7,7 +7,7 @@ project(lsd_slam_core)
 #  Release : w/o debug symbols, w/ optimization
 #  RelWithDebInfo : w/ debug symbols, w/ optimization
 #  MinSizeRel : w/o debug symbols, w/ optimization, stripped binaries
-set(CMAKE_BUILD_TYPE Release)
+set(CMAKE_BUILD_TYPE RelWithDebInfo)
```

```
 find_package(catkin REQUIRED COMPONENTS
   cv_bridge
@@ -15,24 +15,35 @@ find_package(catkin REQUIRED COMPONENTS
   sensor_msgs
   image_transport
   roscpp
+  rospy
   rosbag
+  cmake_modules
+  message_generation
+  std_msgs
 )

 find_package(Eigen REQUIRED)
+add_definitions(${EIGEN_DEFINITIONS})
 find_package(X11 REQUIRED)
+find_package(OpenCV REQUIRED)
+if (OpenCV_VERSION VERSION_EQUAL "3")
+ add_definitions("-DOPENCV3=1")
+endif()
+ MESSAGE(STATUS "using Opencv version " ${OpenCV_VERSION})
 include(cmake/FindG2O.cmake)
 include(cmake/FindSuiteParse.cmake)

 message("-- CHOLMOD_INCLUDE_DIR : " ${CHOLMOD_INCLUDE_DIR})
 message("-- CSPARSE_INCLUDE_DIR : " ${CSPARSE_INCLUDE_DIR})
 message("-- G2O_INCLUDE_DIR : " ${G2O_INCLUDE_DIR})
-
-# FabMap
-# uncomment this part to enable fabmap
-#add_subdirectory(${PROJECT_SOURCE_DIR}/thirdparty/openFabMap)
-#include_directories(${PROJECT_SOURCE_DIR}/thirdparty/openFabMap/include)
-#add_definitions("-DHAVE_FABMAP")
-#set(FABMAP_LIB openFABMAP )
+add_service_files(
+ FILES
+ ResetLSD.srv
+)
+generate_messages(
+  DEPENDENCIES
+  std_msgs
+)

 generate_dynamic_reconfigure_options(
   cfg/LSDDebugParams.cfg
@@ -42,9 +53,17 @@ generate_dynamic_reconfigure_options(
   LIBRARIES lsdslam
   DEPENDS Eigen SuiteSparse
-  CATKIN_DEPENDS libg2o
+  CATKIN_DEPENDS libg2o
+  message_runtime
 )

+# FabMap
+# uncomment this part to enable fabmap
+add_subdirectory(${PROJECT_SOURCE_DIR}/thirdparty/openFabMap)
+include_directories(${PROJECT_SOURCE_DIR}/thirdparty/openFabMap/include)
+add_definitions("-DHAVE_FABMAP")
+set(FABMAP_LIB openFABMAP )
+
 # SSE flags
 add_definitions("-DUSE_ROS")
 add_definitions("-DENABLE_SSE")
@@ -87,7 +106,7 @@ set(SOURCE_FILES

 include_directories(
   include
-  ${EIGEN3_INCLUDE_DIR}
+  ${EIGEN_INCLUDE_DIRS}
```

```
    ${PROJECT_SOURCE_DIR}/src
    ${PROJECT_SOURCE_DIR}/thirdparty/Sophus
    ${CSPARSE_INCLUDE_DIR} #Has been set by SuiteParse
@@ -97,7 +116,7 @@ include_directories(

 # build shared library.
 add_library(lsdslam SHARED ${SOURCE_FILES})
-target_link_libraries(lsdslam ${FABMAP_LIB} ${G2O_LIBRARIES} ${catkin_LIBRARIES}
    csparse cxsparse )
+target_link_libraries(lsdslam ${FABMAP_LIB} ${G2O_LIBRARIES} ${catkin_LIBRARIES}
    csparse cxsparse X11)
 #rosbuild_link_boost(lsdslam thread)


@@ -110,6 +129,9 @@ target_link_libraries(live_slam lsdslam ${catkin_LIBRARIES} ${
    G2O_LIBRARIES})
 add_executable(dataset src/main_on_images.cpp)
 target_link_libraries(dataset lsdslam ${catkin_LIBRARIES} ${G2O_LIBRARIES})

-# TODO add INSTALL
-
-
+## Mark executables and/or libraries for installation
+install(TARGETS live_slam lsdslam dataset
+#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
+   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
+   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
+ )
diff --git a/lsd_slam_core/cfg/LSDParams.cfg b/lsd_slam_core/cfg/LSDParams.cfg
index 4c2acfb..e24dd45 100755
--- a/lsd_slam_core/cfg/LSDParams.cfg
+++ b/lsd_slam_core/cfg/LSDParams.cfg
@@ -18,7 +18,7 @@ gen.add("KFDistWeight", double_t, 0, "Determines how often Keyframes
    are taken,
 gen.add("doSLAM", bool_t, 0, "Toggle Global Mapping Component on/off. Only takes
     effect after a reset.", True)
 gen.add("doKFReActivation", bool_t, 0, "Toggle Keyframe Re-Activation on/off: If close
     to an existing keyframe, re-activate it instead of creating a new one. If false,
     Map will continually grow even if the camera moves in a relatively constrained
     area; If false, the number of keyframes will not grow arbitrarily.", True)
 gen.add("doMapping", bool_t, 0, "Toggle entire Keyframe Creating / Update module on/
     off: If false, only the Tracking Component stays active, which will prevent rapid
     motion or moving objects from corrupting the map.", True)
-gen.add("useFabMap", bool_t, 0, "Use OpenFABMAP to find large loop-closures. Only
    takes effect after a reset, and requires LSD-SLAM to be compiled with FabMap.",
    False)
+gen.add("useFabMap", bool_t, 0, "Use OpenFABMAP to find large loop-closures. Only
    takes effect after a reset, and requires LSD-SLAM to be compiled with FabMap.",
    True)


 gen.add("allowNegativeIdepths", bool_t, 0, "Allow idepth to be (slightle) negative, to
     avoid introducing a bias for far-away points.", True)
diff --git a/lsd_slam_core/package.xml b/lsd_slam_core/package.xml
index 4bc245b..bd9ced9 100644
--- a/lsd_slam_core/package.xml
+++ b/lsd_slam_core/package.xml
@@ -21,6 +21,7 @@
   <build_depend>eigen</build_depend>
   <build_depend>suitesparse</build_depend>
   <build_depend>libg2o</build_depend>
+  <build_depend>message_generation</build_depend>

   <run_depend>cv_bridge</run_depend>
   <run_depend>dynamic_reconfigure</run_depend>
@@ -28,6 +29,7 @@
   <run_depend>roscpp</run_depend>
   <run_depend>lsd_slam_viewer</run_depend>
   <run_depend>rosbag</run_depend>
+  <run_depend>message_runtime</run_depend>
   <run_depend>eigen</run_depend>
```

```
   <run_depend>suitesparse</run_depend>
   <run_depend>libg2o</run_depend>
diff --git a/lsd_slam_core/src/DepthEstimation/DepthMap.cpp b/lsd_slam_core/src/
    DepthEstimation/DepthMap.cpp
index 831b8aa..85200d3 100644
--- a/lsd_slam_core/src/DepthEstimation/DepthMap.cpp
+++ b/lsd_slam_core/src/DepthEstimation/DepthMap.cpp
@@ -1111,13 +1111,21 @@ void DepthMap::updateKeyframe(std::deque< std::shared_ptr<Frame
    >> referenceFram
        {
                cv::Mat keyFrameImage(activeKeyFrame->height(), activeKeyFrame->width()
                    , CV_32F, const_cast<float*>(activeKeyFrameImageData));
                keyFrameImage.convertTo(debugImageHypothesisHandling, CV_8UC1);
-               cv::cvtColor(debugImageHypothesisHandling, debugImageHypothesisHandling
    , CV_GRAY2RGB);
+#if OPENCV3
+       cv::cvtColor(debugImageHypothesisHandling, debugImageHypothesisHandling, cv::
    COLOR_GRAY2RGB);
+#else
+       cv::cvtColor(debugImageHypothesisHandling, debugImageHypothesisHandling,
    CV_GRAY2RGB);
+#endif

                cv::Mat oldest_refImage(oldest_referenceFrame->height(),
                    oldest_referenceFrame->width(), CV_32F, const_cast<float*>(
                    oldest_referenceFrame->image(0)));
                cv::Mat newest_refImage(newest_referenceFrame->height(),
                    newest_referenceFrame->width(), CV_32F, const_cast<float*>(
                    newest_referenceFrame->image(0)));
                cv::Mat rfimg = 0.5f*oldest_refImage + 0.5f*newest_refImage;
                rfimg.convertTo(debugImageStereoLines, CV_8UC1);
-               cv::cvtColor(debugImageStereoLines, debugImageStereoLines, CV_GRAY2RGB)
    ;
+#if OPENCV3
+       cv::cvtColor(debugImageStereoLines, debugImageStereoLines, cv::COLOR_GRAY2RGB);
+#else
+       cv::cvtColor(debugImageStereoLines, debugImageStereoLines, CV_GRAY2RGB);
+#endif
        }

        struct timeval tv_start, tv_end;
@@ -1238,7 +1246,11 @@ void DepthMap::createKeyFrame(Frame* new_keyframe)
        {
                cv::Mat keyFrameImage(new_keyframe->height(), new_keyframe->width(),
                    CV_32F, const_cast<float*>(new_keyframe->image(0)));
                keyFrameImage.convertTo(debugImageHypothesisPropagation, CV_8UC1);
-               cv::cvtColor(debugImageHypothesisPropagation,
    debugImageHypothesisPropagation, CV_GRAY2RGB);
+#if OPENCV3
+cv::cvtColor(debugImageHypothesisPropagation, debugImageHypothesisPropagation, cv::
    COLOR_GRAY2RGB);
+#else
+ cv::cvtColor(debugImageHypothesisPropagation, debugImageHypothesisPropagation,
    CV_GRAY2RGB);
+#endif
        }


@@ -1403,7 +1415,11 @@ int DepthMap::debugPlotDepthMap()

        cv::Mat keyFrameImage(activeKeyFrame->height(), activeKeyFrame->width(), CV_32F
            , const_cast<float*>(activeKeyFrameImageData));
        keyFrameImage.convertTo(debugImageDepth, CV_8UC1);
+#if OPENCV3
+       cv::cvtColor(debugImageDepth, debugImageDepth, cv::COLOR_GRAY2RGB);
+#else
        cv::cvtColor(debugImageDepth, debugImageDepth, CV_GRAY2RGB);
+#endif

        // debug plot & publish sparse version?
        int refID = referenceFrameByID_offset;
```

```
diff --git a/lsd_slam_core/src/GlobalMapping/FabMap.cpp b/lsd_slam_core/src/
    GlobalMapping/FabMap.cpp
index 7e267f3..27f453d 100644
--- a/lsd_slam_core/src/GlobalMapping/FabMap.cpp
+++ b/lsd_slam_core/src/GlobalMapping/FabMap.cpp
@@ -23,7 +23,7 @@

 #include <fstream>
 #include <opencv2/core/core.hpp>
-#include <opencv2/nonfree/features2d.hpp>
+#include <opencv2/features2d.hpp>
 #include "openfabmap.hpp"

 #include "util/settings.h"
@@ -36,7 +36,6 @@ namespace lsd_slam
 FabMap::FabMap()
 {
         valid = false;
-
         std::string fabmapTrainDataPath = packagePath + "thirdparty/openFabMap/
             trainingdata/StLuciaShortTraindata.yml";
         std::string vocabPath = packagePath + "thirdparty/openFabMap/trainingdata/
             StLuciaShortVocabulary.yml";
         std::string chowliutreePath = packagePath + "thirdparty/openFabMap/trainingdata
             /StLuciaShortTree.yml";
@@ -78,7 +77,7 @@ FabMap::FabMap()
         int options = 0;
         options |= of2::FabMap::SAMPLED;
         options |= of2::FabMap::CHOW_LIU;
-        fabMap = new of2::FabMap2(clTree, 0.39, 0, options);
+        fabMap = cv::makePtr<of2::FabMap2>(clTree, 0.39, 0, options);
         //add the training data for use with the sampling method
         fabMap->addTraining(fabmapTrainData);

@@ -91,12 +90,12 @@ FabMap::FabMap()
 //      //fabMap = new of2::FabMapFBO(clTree, 0.39, 0, options, 3000, 1e-6, 1e-6, 512,
     9);

         // Create detector & extractor
-        detector = new cv::StarFeatureDetector(32, 10, 18, 18, 20);
-        cv::Ptr<cv::DescriptorExtractor> extractor = new cv::SURF(1000, 4, 2, false,
    true); // new cv::SIFT();
+        detector = cv::xfeatures2d::StarDetector::create(32, 10, 18, 18, 20);
+        cv::Ptr<cv::xfeatures2d::SURF> extractor = cv::xfeatures2d::SURF::create(1000,
    4, 2, false, true); // new cv::SIFT();

         //use a FLANN matcher to generate bag-of-words representations
         cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create("
             FlannBased"); // alternative: "BruteForce"
-        bide = new cv::BOWImgDescriptorExtractor(extractor, matcher);
+        bide = cv::makePtr<cv::BOWImgDescriptorExtractor>(extractor, matcher);
         bide->setVocabulary(vocabulary);

         printConfusionMatrix = false;
diff --git a/lsd_slam_core/src/GlobalMapping/FabMap.h b/lsd_slam_core/src/GlobalMapping
    /FabMap.h
index 15e58e1..44f4d02 100644
--- a/lsd_slam_core/src/GlobalMapping/FabMap.h
+++ b/lsd_slam_core/src/GlobalMapping/FabMap.h
@@ -21,13 +21,15 @@
 #ifdef HAVE_FABMAP
 #pragma once
 #include <opencv2/core/core.hpp>
+#include <opencv2/features2d.hpp>
+#include <opencv2/xfeatures2d.hpp>

 namespace of2 {
         class FabMap;
 }
 namespace cv {
-        class FeatureDetector;
```

```
-       class BOWImgDescriptorExtractor;
+
+       //class BOWImgDescriptorExtractor;
 }


diff --git a/lsd_slam_core/src/GlobalMapping/TrackableKeyFrameSearch.cpp b/
    lsd_slam_core/src/GlobalMapping/TrackableKeyFrameSearch.cpp
index dddc86f..e4ad4ea 100644
--- a/lsd_slam_core/src/GlobalMapping/TrackableKeyFrameSearch.cpp
+++ b/lsd_slam_core/src/GlobalMapping/TrackableKeyFrameSearch.cpp
@@ -30,7 +30,7 @@ namespace lsd_slam


 TrackableKeyFrameSearch::TrackableKeyFrameSearch(KeyFrameGraph* graph, int w, int h,
     Eigen::Matrix3f K)
-: graph(graph)
+: fabMap(),graph(graph)
 {
       tracker = new SE3Tracker(w,h,K);

diff --git a/lsd_slam_core/src/IOWrapper/ImageDisplay.h b/lsd_slam_core/src/IOWrapper/
    ImageDisplay.h
index e844645..fc8d856 100644
--- a/lsd_slam_core/src/IOWrapper/ImageDisplay.h
+++ b/lsd_slam_core/src/IOWrapper/ImageDisplay.h
@@ -48,7 +48,11 @@ inline void displayImage(const char* windowName, const float* image,
     int width,
       cv::Mat floatWrapper(height, width, CV_32F, const_cast<float*>(image));
       cv::Mat tempImage(height, width, CV_8UC1);
       floatWrapper.convertTo(tempImage, CV_8UC1);
-      cv::cvtColor(tempImage, tempImage, CV_GRAY2RGB);
+#if OPENCV3
+ cv::cvtColor(tempImage, tempImage, cv::COLOR_GRAY2RGB);
+#else
+ cv::cvtColor(tempImage, tempImage, CV_GRAY2RGB);
+#endif
       displayImage(windowName, tempImage);
 }

diff --git a/lsd_slam_core/src/IOWrapper/OpenCV/ImageDisplay_OpenCV.cpp b/lsd_slam_core
    /src/IOWrapper/OpenCV/ImageDisplay_OpenCV.cpp
index 3c83086..6e09089 100644
--- a/lsd_slam_core/src/IOWrapper/OpenCV/ImageDisplay_OpenCV.cpp
+++ b/lsd_slam_core/src/IOWrapper/OpenCV/ImageDisplay_OpenCV.cpp
@@ -70,6 +70,7 @@ void displayThreadLoop()
                                 }
                         }
                         cv::imshow(displayQueue.back().name, displayQueue.back().img);
+                        cv::waitKey(1);
                         displayQueue.pop_back();
                 }
         }
diff --git a/lsd_slam_core/src/IOWrapper/Output3DWrapper.h b/lsd_slam_core/src/
    IOWrapper/Output3DWrapper.h
index 75d5bdb..6e9ce4a 100644
--- a/lsd_slam_core/src/IOWrapper/Output3DWrapper.h
+++ b/lsd_slam_core/src/IOWrapper/Output3DWrapper.h
@@ -46,9 +46,6 @@ public:
       virtual ~Output3DWrapper() {};


-
-
-
       virtual void publishKeyframeGraph(KeyFrameGraph* graph) {};

       // publishes a keyframe. if that frame already existis, it is overwritten,
           otherwise it is added.
diff --git a/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.cpp b/lsd_slam_core/src
    /IOWrapper/ROS/ROSOutput3DWrapper.cpp
```

```
index 78faa88..7edd95f 100644
--- a/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.cpp
+++ b/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.cpp
@@ -21,6 +21,7 @@
 #include "ROSOutput3DWrapper.h"
 #include "util/SophusUtil.h"
 #include <ros/ros.h>
+#include <ros/time.h>
 #include "util/settings.h"


@@ -57,6 +58,7 @@ ROSOutput3DWrapper::ROSOutput3DWrapper(int width, int height)

         pose_channel = nh_.resolveName("lsd_slam/pose");
         pose_publisher = nh_.advertise<geometry_msgs::PoseStamped>(pose_channel,1);
+        service = nh_.advertiseService("resetlsd", &lsd_slam::ROSOutput3DWrapper::reset
    , this);


         publishLvl=0;
@@ -65,7 +67,19 @@ ROSOutput3DWrapper::ROSOutput3DWrapper(int width, int height)
 ROSOutput3DWrapper::~ROSOutput3DWrapper()
 {
 }
-
+bool ROSOutput3DWrapper::reset(lsd_slam_core::ResetLSD::Request &req, lsd_slam_core::
    ResetLSD::Response &res)
+{
+        if(req.request)
+        {
+                fullResetRequested = true;
+                res.reset=true;
+                return true;
+        }else{
+                res.reset=false;
+                return true;
+        }
+
+}

 void ROSOutput3DWrapper::publishKeyframe(Frame* f)
 {
@@ -118,6 +132,9 @@ void ROSOutput3DWrapper::publishTrackedFrame(Frame* kf)

         fMsg.id = kf->id();
         fMsg.time = kf->timestamp();
+        ros::Time stamp;
+        stamp.fromSec(kf->timestamp());
+        fMsg.header.stamp=stamp;
         fMsg.isKeyframe = false;


diff --git a/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.h b/lsd_slam_core/src/
   IOWrapper/ROS/ROSOutput3DWrapper.h
index 9c5cab8..264c7d1 100644
--- a/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.h
+++ b/lsd_slam_core/src/IOWrapper/ROS/ROSOutput3DWrapper.h
@@ -22,7 +22,7 @@

 #include <ros/ros.h>
 #include "IOWrapper/Output3DWrapper.h"
-
+#include "lsd_slam_core/ResetLSD.h"

 namespace lsd_slam
 {
@@ -82,7 +82,7 @@ public:

 private:
         int width, height;
-
```

```
+       bool reset(lsd_slam_core::ResetLSD::Request&, lsd_slam_core::ResetLSD::Response
    &);
        std::string liveframe_channel;
        ros::Publisher liveframe_publisher;

@@ -94,7 +94,7 @@ private:

        std::string debugInfo_channel;
        ros::Publisher debugInfo_publisher;
-
+       ros::ServiceServer service;

        std::string pose_channel;
        ros::Publisher pose_publisher;
diff --git a/lsd_slam_core/src/LiveSLAMWrapper.cpp b/lsd_slam_core/src/LiveSLAMWrapper.
    cpp
index 45423d6..d7f6f65 100644
--- a/lsd_slam_core/src/LiveSLAMWrapper.cpp
+++ b/lsd_slam_core/src/LiveSLAMWrapper.cpp
@@ -118,7 +118,11 @@ void LiveSLAMWrapper::newImageCallback(const cv::Mat& img,
    Timestamp imgTime)
        if (img.channels() == 1)
                grayImg = img;
        else
-               cvtColor(img, grayImg, CV_RGB2GRAY);
+#if OPENCV3
+       cvtColor(img, grayImg, cv::COLOR_RGB2GRAY);
+#else
+       cvtColor(img, grayImg, CV_RGB2GRAY);
+#endif


        // Assert that we work with 8 bit images
diff --git a/lsd_slam_core/src/SlamSystem.cpp b/lsd_slam_core/src/SlamSystem.cpp
index d1e1dbf..babd80f 100644
--- a/lsd_slam_core/src/SlamSystem.cpp
+++ b/lsd_slam_core/src/SlamSystem.cpp
@@ -35,6 +35,7 @@
 #include <g2o/core/robust_kernel_impl.h>
 #include "DataStructures/FrameMemory.h"
 #include "deque"
+#include <boost/thread/thread.hpp>

 // for mkdir
 #include <sys/types.h>
@@ -963,6 +964,9 @@ void SlamSystem::trackFrame(uchar* image, unsigned int frameID,
    bool blockUntilM
                unmappedTrackedFramesMutex.unlock();

                manualTrackingLossIndicated = false;
+               std::cout << "Sleeping for 2 seconds before re-initialization" << std::
    endl;
+               boost::this_thread::sleep( boost::posix_time::seconds(2) );
+               fullResetRequested = true;
                return;
        }

diff --git a/lsd_slam_core/src/main_on_images.cpp b/lsd_slam_core/src/main_on_images.
    cpp
index 1aa0686..e83624e 100644
--- a/lsd_slam_core/src/main_on_images.cpp
+++ b/lsd_slam_core/src/main_on_images.cpp
@@ -221,7 +221,7 @@ int main( int argc, char** argv )

        for(unsigned int i=0;i<files.size();i++)
        {
-               cv::Mat imageDist = cv::imread(files[i], CV_LOAD_IMAGE_GRAYSCALE);
+               cv::Mat imageDist = cv::imread(files[i], 0);

                if(imageDist.rows != h_inp || imageDist.cols != w_inp)
                {
```

```
diff --git a/lsd_slam_core/src/util/globalFuncs.cpp b/lsd_slam_core/src/util/
    globalFuncs.cpp
index c02ca08..e4dd205 100644
--- a/lsd_slam_core/src/util/globalFuncs.cpp
+++ b/lsd_slam_core/src/util/globalFuncs.cpp
@@ -49,11 +49,15 @@ void printMessageOnCVImage(cv::Mat &image, std::string line1,std::
    string line2)
                for(int y=image.rows-30; y<image.rows;y++)
                        image.at<cv::Vec3b>(y,x) *= 0.5;

-       cv::putText(image, line2, cvPoint(10,image.rows-5),
-           CV_FONT_HERSHEY_SIMPLEX, 0.4, cv::Scalar(200,200,250), 1, 8);
-
-       cv::putText(image, line1, cvPoint(10,image.rows-18),
-           CV_FONT_HERSHEY_SIMPLEX, 0.4, cv::Scalar(200,200,250), 1, 8);
+#if OPENCV3
+       cv::putText(image, line2, cv::Point(10,image.rows-5), cv::FONT_HERSHEY_SIMPLEX,
     0.4, cv::Scalar(200,200,250), 1, 8);
+
+       cv::putText(image, line1, cv::Point(10,image.rows-18), cv::FONT_HERSHEY_SIMPLEX
    , 0.4, cv::Scalar(200,200,250), 1, 8);
+#else
+       cv::putText(image, line2, cvPoint(10,image.rows-5), CV_FONT_HERSHEY_SIMPLEX,
    0.4, cv::Scalar(200,200,250), 1, 8);
+
+       cv::putText(image, line1, cvPoint(10,image.rows-18), CV_FONT_HERSHEY_SIMPLEX,
    0.4, cv::Scalar(200,200,250), 1, 8);
+#endif
 }


@@ -71,7 +75,11 @@ cv::Mat getDepthRainbowPlot(const float* idepth, const float*
    idepthVar, const f
                cv::Mat keyFrameImage(height, width, CV_32F, const_cast<float*>(gray));
                cv::Mat keyFrameImage8u;
                keyFrameImage.convertTo(keyFrameImage8u, CV_8UC1);
+#if OPENCV3
+               cv::cvtColor(keyFrameImage8u, res, cv::COLOR_GRAY2RGB);
+#else
                cv::cvtColor(keyFrameImage8u, res, CV_GRAY2RGB);
+#endif
        }
        else
                fillCvMat(&res,cv::Vec3b(255,170,168));
@@ -136,7 +144,11 @@ cv::Mat getVarRedGreenPlot(const float* idepthVar, const float*
    gray, int width,
                cv::Mat keyFrameImage(height, width, CV_32F, const_cast<float*>(gray));
                cv::Mat keyFrameImage8u;
                keyFrameImage.convertTo(keyFrameImage8u, CV_8UC1);
+#if OPENCV3
+               cv::cvtColor(keyFrameImage8u, res, cv::COLOR_GRAY2RGB);
+#else
                cv::cvtColor(keyFrameImage8u, res, CV_GRAY2RGB);
+#endif
        }
        else
                fillCvMat(&res,cv::Vec3b(255,170,168));
diff --git a/lsd_slam_core/src/util/settings.cpp b/lsd_slam_core/src/util/settings.cpp
index 8696551..e819e56 100644
--- a/lsd_slam_core/src/util/settings.cpp
+++ b/lsd_slam_core/src/util/settings.cpp
@@ -35,7 +35,7 @@ bool autoRunWithinFrame = true;
 int debugDisplay = 0;

 bool onSceenInfoDisplay = true;
-bool displayDepthMap = true;
+bool displayDepthMap = false;
 bool dumpMap = false;
 bool doFullReConstraintTrack = false;

diff --git a/lsd_slam_core/srv/ResetLSD.srv b/lsd_slam_core/srv/ResetLSD.srv
```

```
new file mode 100644
index 0000000..1bbd58f
--- /dev/null
+++ b/lsd_slam_core/srv/ResetLSD.srv
@@ -0,0 +1,4 @@
+
+bool request
+---
+bool reset
diff --git a/lsd_slam_core/thirdparty/Sophus/CMakeLists.txt b/lsd_slam_core/thirdparty/
    Sophus/CMakeLists.txt
deleted file mode 100644
index 49a5657..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/FindEigen3.cmake b/lsd_slam_core/
    thirdparty/Sophus/FindEigen3.cmake
deleted file mode 100644
index 9c546a0..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/README b/lsd_slam_core/thirdparty/Sophus/
    README
deleted file mode 100644
index 05213ac..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/SophusConfig.cmake.in b/lsd_slam_core/
    thirdparty/Sophus/SophusConfig.cmake.in
deleted file mode 100644
index 24ba84f..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/cmake_modules/FindEigen3.cmake b/
    lsd_slam_core/thirdparty/Sophus/cmake_modules/FindEigen3.cmake
deleted file mode 100644
index 469c77d..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/rxso3.hpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/rxso3.hpp
deleted file mode 100644
index a0262a0..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/se2.hpp b/lsd_slam_core/thirdparty/
    Sophus/sophus/se2.hpp
deleted file mode 100644
index 32f3ff2..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/se3.hpp b/lsd_slam_core/thirdparty/
    Sophus/sophus/se3.hpp
deleted file mode 100644
index d2a20cc..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/sim3.hpp b/lsd_slam_core/thirdparty
    /Sophus/sophus/sim3.hpp
deleted file mode 100644
index 97087c3..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/so2.hpp b/lsd_slam_core/thirdparty/
    Sophus/sophus/so2.hpp
deleted file mode 100644
index d2e8160..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/so3.hpp b/lsd_slam_core/thirdparty/
    Sophus/sophus/so3.hpp
deleted file mode 100644
index 88a362e..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/sophus.hpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/sophus.hpp
deleted file mode 100644
index 76e142e..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_rxso3.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_rxso3.cpp
deleted file mode 100644
index 46134d3..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_se2.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_se2.cpp
deleted file mode 100644
index 1d9d293..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_se3.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_se3.cpp
deleted file mode 100644
index 4d9313b..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_sim3.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_sim3.cpp
```

```
deleted file mode 100644
index 97c2c8c..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_so2.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_so2.cpp
deleted file mode 100644
index 1685b2b..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/test_so3.cpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/test_so3.cpp
deleted file mode 100644
index e961e1c..0000000
diff --git a/lsd_slam_core/thirdparty/Sophus/sophus/tests.hpp b/lsd_slam_core/
    thirdparty/Sophus/sophus/tests.hpp
deleted file mode 100644
index 0ec3c4a..0000000
diff --git a/lsd_slam_core/thirdparty/openFabMap/CMakeLists.txt b/lsd_slam_core/
    thirdparty/openFabMap/CMakeLists.txt
index aa390f2..1daec3e 100644
--- a/lsd_slam_core/thirdparty/openFabMap/CMakeLists.txt
+++ b/lsd_slam_core/thirdparty/openFabMap/CMakeLists.txt
@@ -64,7 +64,15 @@ IF(OPENCV2_FOUND)
                 TARGET_LINK_LIBRARIES(openFABMAP opencv_highgui opencv_core
                     opencv_features2d opencv_imgproc opencv_nonfree)
         ENDIF(WIN32)
 ENDIF(OPENCV2_FOUND)
-
+#install targets for library and trainingdata
+install(TARGETS openFABMAP
+    LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
+ )
+ install(DIRECTORY trainingdata/
+    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/thirdparty/openFabMap/trainingdata/
+    FILES_MATCHING PATTERN "*.yml"
+#    PATTERN ".svn" EXCLUDE
+ )

 ############## openFABMAPcli executable ###################
 # if(OPENCV2_FOUND)
diff --git a/lsd_slam_core/thirdparty/openFabMap/FindOpenCV.cmake b/lsd_slam_core/
    thirdparty/openFabMap/FindOpenCV.cmake
deleted file mode 100644
index 1fa1a78..0000000
diff --git a/lsd_slam_viewer/CMakeLists.txt b/lsd_slam_viewer/CMakeLists.txt
index f0c2512..3daca0b 100644
--- a/lsd_slam_viewer/CMakeLists.txt
+++ b/lsd_slam_viewer/CMakeLists.txt
@@ -1,4 +1,4 @@
-cmake_minimum_required(VERSION 2.4.6)
+cmake_minimum_required(VERSION 2.8.7)
 project(lsd_slam_viewer)

 # Set the build type. Options are:
@@ -7,20 +7,19 @@ project(lsd_slam_viewer)
 #  Release : w/o debug symbols, w/ optimization
 #  RelWithDebInfo : w/ debug symbols, w/ optimization
 #  MinSizeRel : w/o debug symbols, w/ optimization, stripped binaries
-set(CMAKE_BUILD_TYPE Release)
-
-ADD_SUBDIRECTORY(${PROJECT_SOURCE_DIR}/thirdparty/Sophus)
-
-set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${PROJECT_SOURCE_DIR}/cmake)
+set(CMAKE_BUILD_TYPE RelWithDebInfo)
+set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${PROJECT_SOURCE_DIR}/cmake )

 find_package(catkin REQUIRED COMPONENTS
   cv_bridge
   dynamic_reconfigure
+  std_msgs
   sensor_msgs
   roscpp
   rosbag
   message_generation
```

```
    roslib
+   cmake_modules
  )

  find_package(OpenGL REQUIRED)
@@ -31,23 +30,32 @@ find_package(Eigen REQUIRED)
  find_package(OpenCV REQUIRED)
  find_package(Boost REQUIRED COMPONENTS thread)

-include_directories(${QGLVIEWER_INCLUDE_DIR}
+include_directories(${EIGEN_INCLUDE_DIRS}
+                    ${QGLVIEWER_INCLUDE_DIR}
                     ${catkin_INCLUDE_DIRS}
                     ${EIGEN_INCLUDE_DIR}
                     ${QT_INCLUDES} )

+add_definitions(${EIGEN_DEFINITIONS})
+
  # SSE flags
  set(CMAKE_CXX_FLAGS
     "${CMAKE_CXX_FLAGS} -march=native -Wall -std=c++0x"
  )

  add_message_files(DIRECTORY msg FILES keyframeMsg.msg keyframeGraphMsg.msg)
-generate_messages(DEPENDENCIES)
+generate_messages(DEPENDENCIES std_msgs sensor_msgs)

  generate_dynamic_reconfigure_options(
    cfg/LSDSLAMViewerParams.cfg
  )

+catkin_package(
+  DEPENDS opencv
+  CATKIN_DEPENDS
+  cv_bridge
+message_runtime
+)

  # Sources files
  set(SOURCE_FILES
@@ -85,3 +93,14 @@ target_link_libraries(viewer ${QGLViewer_LIBRARIES}
  #                            GL glut GLU
  #                     )

+## Mark executables and/or libraries for installation
+ install(TARGETS viewer
+#    ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
+    LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
+    RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
+ )
+ install(DIRECTORY src/
+    DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
+    FILES_MATCHING PATTERN "*.h"
+#    PATTERN ".svn" EXCLUDE
+ )
diff --git a/lsd_slam_viewer/cmake/FindEigen3.cmake b/lsd_slam_viewer/cmake/FindEigen3.
    cmake
deleted file mode 100644
index 9c546a0..0000000
diff --git a/lsd_slam_viewer/msg/keyframeMsg.msg b/lsd_slam_viewer/msg/keyframeMsg.msg
index 004a166..621b97f 100644
--- a/lsd_slam_viewer/msg/keyframeMsg.msg
+++ b/lsd_slam_viewer/msg/keyframeMsg.msg
@@ -1,3 +1,4 @@
+Header header
  int32 id
  float64 time
  bool isKeyframe
diff --git a/lsd_slam_viewer/package.xml b/lsd_slam_viewer/package.xml
index b411459..c212889 100644
--- a/lsd_slam_viewer/package.xml
```

```
+++ b/lsd_slam_viewer/package.xml
@@ -19,7 +19,7 @@
   <build_depend>roslib</build_depend>
   <build_depend>rosbag</build_depend>
   <build_depend>message_generation</build_depend>
-
+  <run_depend>message_runtime</run_depend>
   <run_depend>cv_bridge</run_depend>
   <run_depend>dynamic_reconfigure</run_depend>
   <run_depend>sensor_msgs</run_depend>
diff --git a/lsd_slam_viewer/raw.ply b/lsd_slam_viewer/raw.ply
new file mode 100644
index 0000000..4cc3350
Binary files /dev/null and b/lsd_slam_viewer/raw.ply differ
diff --git a/lsd_slam_viewer/rectified.ply b/lsd_slam_viewer/rectified.ply
new file mode 100644
index 0000000..aeb8b1d
Binary files /dev/null and b/lsd_slam_viewer/rectified.ply differ
diff --git a/lsd_slam_viewer/src/KeyFrameDisplay.cpp b/lsd_slam_viewer/src/
    KeyFrameDisplay.cpp
index 0ba66a2..e10791a 100644
--- a/lsd_slam_viewer/src/KeyFrameDisplay.cpp
+++ b/lsd_slam_viewer/src/KeyFrameDisplay.cpp
@@ -90,7 +90,7 @@ void KeyFrameDisplay::setFrom(lsd_slam_viewer::keyframeMsgConstPtr
    msg)
        {
                if(msg->pointcloud.size() != 0)
                {
-                       printf("WARNING: PC with points, but number of points not right
    ! (is %zu, should be %u*%dx%d=%u)\n",
+                       printf("WARNING: PC with points, but number of points not right
    ! (is %zu, should be %zu*%dx%d=%zu)\n",
                                        msg->pointcloud.size(), sizeof(InputPointDense)
                                            , width, height, width*height*sizeof(
                                            InputPointDense));
                }
        }
diff --git a/lsd_slam_viewer/src/KeyFrameGraphDisplay.cpp b/lsd_slam_viewer/src/
    KeyFrameGraphDisplay.cpp
index 6fce35f..3b7e36b 100644
--- a/lsd_slam_viewer/src/KeyFrameGraphDisplay.cpp
+++ b/lsd_slam_viewer/src/KeyFrameGraphDisplay.cpp
@@ -47,7 +47,7 @@ void KeyFrameGraphDisplay::draw()

        // draw keyframes
        float color[3] = {0,0,1};
-       for(unsigned int i=0;i<keyframes.size();i++)
+       for(std::size_t i=0;i<keyframes.size();i++)
        {
                if(showKFCameras)
                        keyframes[i]->drawCam(lineTesselation, color);
@@ -63,7 +63,7 @@ void KeyFrameGraphDisplay::draw()
                printf("Flushing Pointcloud to %s!\n", (ros::package::getPath("
                    lsd_slam_viewer")+"/pc_tmp.ply").c_str());
                std::ofstream f((ros::package::getPath("lsd_slam_viewer")+"/pc_tmp.ply"
                    ).c_str());
                int numpts = 0;
-               for(unsigned int i=0;i<keyframes.size();i++)
+               for(std::size_t i=0;i<keyframes.size();i++)
                {
                        if((int)i > cutFirstNKf)
                                numpts += keyframes[i]->flushPC(&f);
@@ -87,7 +87,11 @@ void KeyFrameGraphDisplay::draw()
                f2.close();
                f3.close();

-               system(("rm "+ros::package::getPath("lsd_slam_viewer")+"/pc_tmp.ply").
    c_str());
+               int systemRet =system(("rm "+ros::package::getPath("lsd_slam_viewer")+"
    /pc_tmp.ply").c_str());
+               if(systemRet==-1)
```

```
+               {
+                       printf("Removing temp file failed\n");
+               }
                flushPointcloud = false;
                printf("Done Flushing Pointcloud with %d points!\n", numpts);

@@ -98,7 +102,7 @@ void KeyFrameGraphDisplay::draw()
        {
                int totalPoint = 0;
                int visPoints = 0;
-               for(unsigned int i=0;i<keyframes.size();i++)
+               for(std::size_t i=0;i<keyframes.size();i++)
                {
                        totalPoint += keyframes[i]->totalPoints;
                        visPoints += keyframes[i]->displayedPoints;
@@ -117,7 +121,7 @@ void KeyFrameGraphDisplay::draw()
                // draw constraints
                glLineWidth(lineTesselation);
                glBegin(GL_LINES);
-               for(unsigned int i=0;i<constraints.size();i++)
+               for(std::size_t i=0;i<constraints.size();i++)
                {
                        if(constraints[i].from == 0 || constraints[i].to == 0)
                                continue;
@@ -162,7 +166,7 @@ void KeyFrameGraphDisplay::addGraphMsg(lsd_slam_viewer::
    keyframeGraphMsgConstPtr
        constraints.resize(msg->numConstraints);
        assert(msg->constraintsData.size() == sizeof(GraphConstraint)*msg->
            numConstraints);
        GraphConstraint* constraintsIn = (GraphConstraint*)msg->constraintsData.data();
-       for(int i=0;i<msg->numConstraints;i++)
+       for(std::size_t i=0;i<msg->numConstraints;i++)
        {
                constraints[i].err = constraintsIn[i].err;
                constraints[i].from = 0;
diff --git a/lsd_slam_viewer/src/KeyFrameGraphDisplay.h b/lsd_slam_viewer/src/
    KeyFrameGraphDisplay.h
index 7872f52..7af4b27 100644
--- a/lsd_slam_viewer/src/KeyFrameGraphDisplay.h
+++ b/lsd_slam_viewer/src/KeyFrameGraphDisplay.h
@@ -60,8 +60,8 @@ public:

        void draw();

-       void addMsg(lsd_slam_viewer::keyframeMsgConstPtr msg);
-       void addGraphMsg(lsd_slam_viewer::keyframeGraphMsgConstPtr msg);
+       void addMsg(lsd_slam_viewer::keyframeMsgConstPtr);
+       void addGraphMsg(lsd_slam_viewer::keyframeGraphMsgConstPtr);



diff --git a/lsd_slam_viewer/src/main_viewer.cpp b/lsd_slam_viewer/src/main_viewer.cpp
index 9d0ceff..91ea627 100644
--- a/lsd_slam_viewer/src/main_viewer.cpp
+++ b/lsd_slam_viewer/src/main_viewer.cpp
@@ -88,8 +88,7 @@ void rosThreadLoop( int argc, char** argv )

        //glutInit(&argc, argv);

-       ros::init(argc, argv, "viewer");
-       ROS_INFO("lsd_slam_viewer started");
+

        dynamic_reconfigure::Server<lsd_slam_viewer::LSDSLAMViewerParamsConfig> srv;
        srv.setCallback(dynConfCb);
@@ -113,7 +112,7 @@ void rosThreadLoop( int argc, char** argv )

 void rosFileLoop( int argc, char** argv )
 {
-       ros::init(argc, argv, "viewer");
+
```

```
        dynamic_reconfigure::Server<lsd_slam_viewer::LSDSLAMViewerParamsConfig> srv;
        srv.setCallback(dynConfCb);

@@ -173,6 +172,9 @@ int main( int argc, char** argv )

        boost::thread rosThread;

+       ros::init(argc, argv, "viewer");
+       ROS_INFO("lsd_slam_viewer started");
+
        if(argc > 1)
        {
                rosThread = boost::thread(rosFileLoop, argc, argv);
diff --git a/lsd_slam_viewer/thirdparty/Sophus/CMakeLists.txt b/lsd_slam_viewer/
    thirdparty/Sophus/CMakeLists.txt
deleted file mode 100644
index 49a5657..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/FindEigen3.cmake b/lsd_slam_viewer/
    thirdparty/Sophus/FindEigen3.cmake
deleted file mode 100644
index 9c546a0..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/README b/lsd_slam_viewer/thirdparty/
    Sophus/README
deleted file mode 100644
index 05213ac..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/SophusConfig.cmake.in b/lsd_slam_viewer/
    thirdparty/Sophus/SophusConfig.cmake.in
deleted file mode 100644
index 24ba84f..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/cmake_modules/FindEigen3.cmake b/
    lsd_slam_viewer/thirdparty/Sophus/cmake_modules/FindEigen3.cmake
deleted file mode 100644
index 469c77d..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/rxso3.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/rxso3.hpp
deleted file mode 100644
index a0262a0..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/se2.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/se2.hpp
deleted file mode 100644
index 32f3ff2..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/se3.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/se3.hpp
deleted file mode 100644
index d2a20cc..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/sim3.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/sim3.hpp
deleted file mode 100644
index 97087c3..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/so2.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/so2.hpp
deleted file mode 100644
index d2e8160..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/so3.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/so3.hpp
deleted file mode 100644
index 88a362e..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/sophus.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/sophus.hpp
deleted file mode 100644
index 76e142e..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_rxso3.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_rxso3.cpp
deleted file mode 100644
index 46134d3..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_se2.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_se2.cpp
deleted file mode 100644
index 1d9d293..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_se3.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_se3.cpp
```

```
deleted file mode 100644
index 4d9313b..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_sim3.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_sim3.cpp
deleted file mode 100644
index 97c2c8c..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_so2.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_so2.cpp
deleted file mode 100644
index 1685b2b..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/test_so3.cpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/test_so3.cpp
deleted file mode 100644
index e961e1c..0000000
diff --git a/lsd_slam_viewer/thirdparty/Sophus/sophus/tests.hpp b/lsd_slam_viewer/
    thirdparty/Sophus/sophus/tests.hpp
deleted file mode 100644
index 0ec3c4a..0000000
```

C. Install scripts

C.1. OpenCV

```
####################
# OpenCV3 Intallation
####################
#install build essentials
sudo apt-get install build-essential cmake git libgtk2.0-dev pkg-config libpython3-dev
    python3-sphinx python3-numpy -y
#install codecs and video handling
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libxine-dev
    libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev libtbb-dev libqt4
    -dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev
    libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-utils unzip -y
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev
    libtiff-dev libjasper-dev libdc1394-22-dev -y
# install opengl libraries
sudo apt-get install mesa-common-dev -y
sudo apt-get install libglu1-mesa-dev -y
# install opencl libraries
sudo apt-get install ocl-icd-opencl-dev -y


cd ~/Downloads
git clone https://github.com/Itseez/opencv.git
# Remove OpenCV:
# sudo rm /usr/local/lib/libopencv*
# sudo rm /usr/local/bin/opencv*
# sudo rm -rf /usr/local/share/OpenCV
# sudo rm -rf /usr/local/include/opencv*
git clone https://github.com/Itseez/opencv_contrib.git

rm -r clBLAS*
rm -r clFFT*

wget https://github.com/clMathLibraries/clBLAS/releases/download/v2.4/clBLAS-2.4.0-
    Linux-x64.tar.gz
tar -xvf clBLAS*
sudo cp -r clBLAS*/bin/* /usr/local/bin/.
sudo cp -r clBLAS*/lib64/* /usr/local/lib/.
sudo cp -r clBLAS*/include/* /usr/local/include/.

wget https://github.com/clMathLibraries/clFFT/releases/download/v2.2/clFFT-2.2.0-Linux-
    x64.tar.gz
tar -xvf clFFT*
sudo cp -r clFFT*/bin/* /usr/local/bin/.
sudo cp -r clFFT*/lib64/* /usr/local/lib/.
sudo cp -r clFFT*/include/* /usr/local/include/.

cd opencv
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -DOPENCV_EXTRA_MODULES_PATH=~/Downloads/
    opencv_contrib/modules -DBUILD_opencv_adas=off -DBUILD_opencv_bgsegm=OFF -
    DBUILD_opencv_bioinspired=OFF -DBUILD_opencv_matlab=OFF -DBUILD_opencv_saliency=OFF
     -DBUILD_opencv_ccalib=OFF -DBUILD_opencv_datasets=OFF -DBUILD_opencv_face=OFF -
    DBUILD_opencv_line_descriptor=OFF -DBUILD_opencv_reg=OFF -DBUILD_opencv_optflow=OFF
     -DBUILD_opencv_rgbd=off -DBUILD_opencv_rgbd=OFF -DBUILD_opencv_surface_matching=
    OFF -DBUILD_opencv_text=OFF -DBUILD_opencv_ximgproc=OFF -DBUILD_opencv_xobjdetect=
    OFF -D BUILD_opencv_java=OFF -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
    WITH_V4L=ON -D WITH_QT=ON -D WITH_OPENGL=ON -D BUILD_TESTS=OFF -D BUILD_PERF_TESTS=
```

```
    OFF -D WITH_CUDA=OFF ..
make -j8
sudo make install
#detect newly installed libraries
sudo ldconfig
```

C.2. ROS

```
# Guide to install ROS INDIGO + ARDRONE_AUTONOMY package

# 1. Setup Sources list
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/
    sources.list.d/ros-latest.list'

# 2. Set up pub key of ros repo
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key
     add -

# 3. Get info from repo
sudo apt-get update

# 4.a Install ROS indigo dependencies
sudo apt-get install build-essential daemontools libasound2-dev libavahi-client-dev
    libavahi-common-dev libcaca-dev libdbus-1-dev libiw-dev libpulse-dev libsdl1.2-dev
    libslang2-dev libudev-dev libqglviewer-dev freeglut3-dev libogre-1.9-dev libopenni-
    dev libflann-dev libvtk5-dev -y

# 4.b Some additional media codex
sudo apt-get install gstreamer0.10-plugins-ugly libxine1-ffmpeg gxine mencoder
    libdvdread4 totem-mozilla icedax tagtool easytag id3tool lame nautilus-script-audio
    -convert libmad0 mpg321 libavcodec-extra -y

# 4.c Install ROS indigo
sudo apt-get install ros-indigo-actionlib-msgs ros-indigo-actionlib-tutorials ros-
    indigo-angles ros-indigo-bond ros-indigo-bond-core ros-indigo-bondcpp ros-indigo-
    bondpy ros-indigo-camera-info-manager ros-indigo-catkin ros-indigo-class-loader
    ros-indigo-cmake-modules ros-indigo-collada-parser ros-indigo-collada-urdf ros-
    indigo-common-msgs ros-indigo-common-tutorials ros-indigo-control-msgs ros-indigo
    -cpp-common ros-indigo-diagnostic-aggregator ros-indigo-diagnostic-analysis ros-
    indigo-diagnostic-common-diagnostics ros-indigo-diagnostic-msgs ros-indigo-
    diagnostic-updater ros-indigo-diagnostics ros-indigo-dynamic-reconfigure ros-indigo
    -eigen-conversions ros-indigo-eigen-stl-containers ros-indigo-executive-smach ros-
    indigo-filters ros-indigo-gencpp ros-indigo-genlisp ros-indigo-genmsg ros-indigo-
    genpy ros-indigo-geometric-shapes ros-indigo-geometry ros-indigo-geometry-msgs
    ros-indigo-geometry-tutorials ros-indigo-image-transport ros-indigo-interactive-
    marker-tutorials ros-indigo-interactive-markers ros-indigo-joint-state-publisher
    ros-indigo-kdl-conversions ros-indigo-kdl-parser ros-indigo-laser-geometry ros-
    indigo-libg2o ros-indigo-map-msgs ros-indigo-media-export ros-indigo-message-
    filters ros-indigo-message-generation ros-indigo-message-runtime ros-indigo-mk ros-
    indigo-nav-msgs ros-indigo-nodelet ros-indigo-nodelet-core ros-indigo-nodelet-topic
    -tools ros-indigo-nodelet-tutorial-math ros-indigo-octomap ros-indigo-orocos-kdl
    ros-indigo-pluginlib ros-indigo-pluginlib-tutorials ros-indigo-python-orocos-kdl
    ros-indigo-python-qt-binding ros-indigo-qt-dotgraph ros-indigo-qt-gui ros-indigo-qt
    -gui-cpp ros-indigo-qt-gui-py-common ros-indigo-random-numbers ros-indigo-resource
    -retriever ros-indigo-robot ros-indigo-robot-model ros-indigo-robot-state-publisher
     ros-indigo-ros ros-indigo-ros-base ros-indigo-ros-comm ros-indigo-ros-core ros-
    indigo-ros-tutorials ros-indigo-rosbag ros-indigo-rosbag-migration-rule ros-indigo
    -rosbag-storage ros-indigo-rosbash ros-indigo-rosboost-cfg ros-indigo-rosbuild ros
    -indigo-rosclean ros-indigo-rosconsole ros-indigo-rosconsole-bridge ros-indigo-
    roscpp ros-indigo-roscpp-core ros-indigo-roscpp-serialization ros-indigo-roscpp-
    traits ros-indigo-roscpp-tutorials ros-indigo-roscreate -y
sudo apt-get install ros-indigo-rosgraph ros-indigo-rosgraph-msgs ros-indigo-roslang
    ros-indigo-roslaunch ros-indigo-roslib ros-indigo-roslint ros-indigo-roslisp ros-
    indigo-roslz4 ros-indigo-rosmake ros-indigo-rosmaster ros-indigo-rosmsg ros-indigo-
    rosnode ros-indigo-rosout ros-indigo-rospack ros-indigo-rosparam ros-indigo-rospy
    ros-indigo-rospy-tutorials ros-indigo-rosservice ros-indigo-rostest ros-indigo-
    rostime ros-indigo-rostopic ros-indigo-rosunit ros-indigo-roswtf ros-indigo-rqt-
    action ros-indigo-rqt-bag ros-indigo-rqt-bag-plugins ros-indigo-rqt-console ros-
    indigo-rqt-dep ros-indigo-rqt-graph ros-indigo-rqt-gui ros-indigo-rqt-gui-cpp ros-
    indigo-rqt-gui-py ros-indigo-rqt-launch ros-indigo-rqt-logger-level ros-indigo-rqt-
```

```
    moveit ros-indigo-rqt-msg ros-indigo-rqt-nav-view ros-indigo-rqt-plot ros-indigo-
    rqt-pose-view ros-indigo-rqt-publisher ros-indigo-rqt-py-common ros-indigo-rqt-py-
    console ros-indigo-rqt-reconfigure ros-indigo-rqt-robot-dashboard ros-indigo-rqt-
    robot-monitor ros-indigo-rqt-robot-steering ros-indigo-rqt-runtime-monitor ros-
    indigo-rqt-service-caller ros-indigo-rqt-shell ros-indigo-rqt-srv ros-indigo-rqt-tf
    -tree ros-indigo-rqt-top ros-indigo-rqt-topic ros-indigo-rqt-web ros-indigo-self-
    test ros-indigo-sensor-msgs ros-indigo-shape-msgs ros-indigo-shape-tools ros-indigo
    -smach ros-indigo-smach-msgs ros-indigo-smach-ros ros-indigo-smclib ros-indigo-std-
    msgs ros-indigo-std-srvs ros-indigo-stereo-msgs ros-indigo-tf ros-indigo-tf-
    conversions ros-indigo-tf2 ros-indigo-tf2-msgs ros-indigo-tf2-py ros-indigo-tf2-ros
     ros-indigo-topic-tools ros-indigo-trajectory-msgs ros-indigo-turtle-actionlib ros-
    indigo-turtle-tf ros-indigo-turtle-tf2 ros-indigo-turtlesim ros-indigo-urdf ros-
    indigo-urdf-parser-plugin ros-indigo-urdf-tutorial ros-indigo-visualization-marker-
    tutorials ros-indigo-visualization-msgs ros-indigo-xacro ros-indigo-xmlrpcpp sbcl
    shiboken sip-dev tango-icon-theme uuid-dev ros-indigo-tf2-geometry-msgs ros-indigo-
    tf2-sensor-msgs -y

# 5. Initialize ros dependency list
sudo rosdep init
rosdep update

# 6. Add ROS env vars to all bash sessions
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

# 7. Install rosinstall
sudo apt-get install python-rosinstall -y

# 8. Install PCL from source
cd ~/Downloads
rm -rf pcl-*
wget https://github.com/PointCloudLibrary/pcl/archive/pcl-1.7.2.tar.gz
tar -xvf pcl-*
cd pcl-pcl*
gedit CMakeLists.txt
# Add the following line to CmakeLists.txt
# set(CMAKE_CXX_FLAGS  "${CMAKE_CXX_FLAGS}  -std=c++11" )
mkdir release
cd release
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j8
sudo make install

# 9. Install ROS packages
USERNAME=olivier
PASSWORD=kakkerlak

# Install SOPHUS in seperate workspace
mkdir -p ~/3rd_party_workspace/src
cd ~/3rd_party_workspace/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/sophus.git -b indigo
cd ~/3rd_party_workspace
catkin_make_isolated -DCMAKE_INSTALL_PREFIX=install_isolated --install
echo 'source ~/3rd_party_workspace/install_isolated/setup.bash' >> ~/.bashrc

# Create new workspace
mkdir -p ~/ros/src
cd ~/ros/src
catkin_init_workspace
echo 'source ~/ros/install/setup.bash' >> ~/.bashrc

bash
USERNAME=myusername
PASSWORD=myPaSSWord

# Install all the packages but build in order
cd ~/ros/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/vision_opencv.git
cd ~/ros
catkin_make install

cd ~/ros/src
```

```
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/ardrone_autonomy.git -b
    indigo-devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone https://github.com/ros-perception/image_pipeline.git -b indigo
cd ~/ros
catkin_make install

cd ~/Downloads
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/rqt_common_plugins.git
cp -r ~/Downloads/rqt_common_plugins/rqt_image_view ~/ros/src/.
cd ~/ros
catkin_make install

cd ~/ros/src
git clone https://github.com/ros-perception/pcl_msgs -b indigo-devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone https://github.com/ros-perception/pcl_conversions.git -b indigo-devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone https://github.com/ros-perception/perception_pcl.git -b indigo-devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/rviz.git -b indigo-devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/lsd_slam.git -b catkin
cd ~/ros
catkin_make install

cd ~/ros/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/tum_ardrone.git -b indigo
    -devel
cd ~/ros
catkin_make install

cd ~/ros/src
git clone http://$USERNAME:$PASSWORD@192.168.65.18:165/common/pointcloudregistration.
    git
cd ~/ros
catkin_make install


#There may be interrupts that ask if you want to install extra packages (yes auto
    installs it)
#must be done without --pkg switch to generate setup.bash and etc. files
yes | catkin_make install
# 9. Add new ROS packages to all bash sessions
echo "source ~/ros/install/setup.bash" >> ~/.bashrc
bash
# 10. Install ardrone_autonomy and lsd_slam

yes | catkin_make install --pkg lsd_slam_viewer
yes | catkin_make install --pkg lsd_slam_core
yes | catkin_make install
```

D. Matlab code

D.1. 3D angle detection

```
close all; clear all; clc;
plot=4;

%set(0,'DefaultFigureWindowStyle','docked');

figure(4);clf;
colormap('Autumn')

files = dir('DepthImages/*.png');
imgSize = size(imread(['DepthImages/',files(1).name]));

iSteps = 10;
jSteps = 4;

di = floor(imgSize(1)/iSteps);
dj = floor(imgSize(2)/jSteps);

fx=555.497124*5;
cx=326.380454*5;
fy=562.999371*5;
cy=197.357375*5;

camToWorld = @(u,v,d)[(u/fx-cx/fx)*d,(v/fy-cy/fy)*d];

for k=1:length(files)

dI = double(imread(['DepthImages/',files(k).name]))/(2^16-1)*2;

% ERODE
%se = strel('rectangle',[5,5]);
%dIE = imerode(depthImg,se);

% Gaussian filter
filter = fspecial('gaussian',[5,5],2);
dIF = imfilter(dI,filter,'same');

% Select patch
patchPanel = [];
for j=1:jSteps
    for i=1:iSteps
        patch=dIF(di*(i-1)+1:di*i,dj*(j-1)+1:dj*j);
        if  sum(sum(patch<1.999))/numel(patch) > 0.5 % Density check
            meanPatch = mean(patch(patch<1.999));
        else
            meanPatch = 2;
        end
        cooPatch = camToWorld(dj*(2*j-1)/2,di*(2*i-1)/2,meanPatch);
        patchPanel = [patchPanel; cooPatch, meanPatch];
    end
end
clf;
imagesc(dIF);
hold on;
dAngle=4;
detected = [];
for j=1:jSteps-1
    % cut off 20% of the image (starting from the top)
    for i=floor(iSteps*0.2):iSteps-dAngle
```

```
        angle = atand((patchPanel(i+dAngle+iSteps*(j-1),2)-patchPanel(i+iSteps*(j-1),2)
            )/(patchPanel(i+dAngle+iSteps*(j-1),3)-patchPanel(i+iSteps*(j-1),3)));
        if angle > 30 && angle < 89
                line([(j-1)*dj+1,j*dj],[(i-1)*di+1,(i-1)*di+1])
                line([(j-1)*dj+1,j*dj],[(i+dAngle-1)*di,(i+dAngle-1)*di])
                line([(j-1)*dj+1,(j-1)*dj+1],[(i-1)*di+1,(i+dAngle-1)*di])
                line([j*dj,j*dj],[(i-1)*di+1,(i+dAngle-1)*di])
                detected = [detected; i,j,dAngle,angle];
        end
    end
end
detected

pause(0.05);
end
```

D.2. Staircase theoretical model verification

```
function model01(cameraHeight,cameraDepth,stairDepth,stairHeight,projectDistance,
    projectAngle)
%
% Example
%   model01(1,1,0.19,0.16,0,10/180*pi)
%
% function projection/distance/relativeDistance
% Arguments:
%   a = Distance between beginning staircase and camera (horizontal)
%   b = Height of camera
%   c = Horizontal distance between projection screen and camera
%   d = depth of staircase
%   h = height of staircase
%   i = step of the staircase (starting from 1)
%

if(nargin==0)
    cameraHeight = 1;
    cameraDepth = 1;
    stairDepth = 0.19;
    stairHeight = 0.16;
    projectDistance = 2;
    projectAngle = 0;%45/180*pi; % Radians
end

projection = @(a,b,c,d,h,i,t) [((a+c)*(i*d+a)-tan(t)*(i*h*a+i*d*b))/(tan(t)*(b-i*h)+i*d
    +a), ...
(i*d*b+a*b+c*b-c*i*h)/(i*d+a+tan(t)*(b-i*h))];

distance = @(a,b,c,d,h,i,t) (((c - b*tan(t))^2*(tan(t)^2 + 1)*(b*d + a*h)^2)/...
    ((a + d*i + b*tan(t) - h*i*tan(t))^2*(a + d + d*i + b*tan(t) - h*tan(t) - h*i*tan(t
        ))^2))^(1/2);

relativeDistance = @(a,b,d,h,i,t) (a*cos(t) + 2*d*cos(t) + b*sin(t) - 2*h*sin(t) + d*i*
    cos(t) - ...
    h*i*sin(t))/(a*cos(t) + b*sin(t) + d*i*cos(t) - h*i*sin(t));

amountOfStairs = 10;


figure(1); clf;
subplot(1,2,1);
hold on; axis equal;
plot(cameraDepth,cameraHeight,'r*');
points=[];
for i=1:amountOfStairs
    points = [points; 0-stairDepth*(i-1), 0+stairHeight*(i-1); ...
              0-stairDepth*i, 0+stairHeight*(i-1)];
    projectedPoints(i,1:2) = projection(cameraDepth, cameraHeight, ...
                  projectDistance, stairDepth, stairHeight,i-1,projectAngle);
    line([points(2*i-1,1), projectedPoints(i,1)], ...
         [points(2*i-1,2), projectedPoints(i,2)]);
```

```
end
for i=1:size(points,1)-1
    line(points(i:(i+1),1),points(i:(i+1),2),'Color','green');
end

line(projectedPoints([1,end],1),projectedPoints([1,end],2),'Color','red');

ylimit = get(gca,'YLim');
xlabel('Distance [m]')
ylabel('Height [m]')
grid on
axis([-2,3,0,3])
% Projection in plane
subplot(1,2,2); hold on; axis equal
frameLines = 0;
line([0,10],frameLines(1)*ones(1,2));
for i=1:amountOfStairs
    frameLines(i+1) = frameLines(i) + ...
        distance(cameraDepth, cameraHeight, projectDistance, stairDepth, stairHeight,i
            -1,projectAngle);
    line([0,10],frameLines(i+1)*ones(1,2));
end
set(gca,'YDir','reverse');
%set(gca,'YLim',ylimit);
xlabel('Distance [m]')
ylabel('Height [m]')
grid on
axis([0,5,0,3])


end
```

E. Camera calibration files

E.1. ROS camera calibration file

ardrone_front.yaml

```
image_width: 640
image_height: 352
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [559.987134, 0.000000, 325.025852, 0.000000, 560.595505, 207.532958, 0.000000,
      0.000000, 1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.513762, 0.286297, -0.002095, -0.002213, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [456.938599, 0.000000, 323.962136, 0.000000, 0.000000, 526.611023, 210.075920,
      0.000000, 0.000000, 0.000000, 1.000000, 0.000000]
```

E.2. PTAM calibration file

ardrone2_default.txt

```
0.766313 1.41334 0.529179 0.517397 1.15136
```

E.3. PTAM camera calibrator launch file

cameracalibrator.launch

```
<launch>
    <node launch-prefix="gdb -ex run --args" name="cameracalibrator" pkg="ptam" type="
        cameracalibrator" clear_params="true" output="screen">
            <remap from="image" to="/ardrone/front/image_mono" />
            <remap from="pose" to="pose"/>
            <rosparam file="$(find ptam)/PtamFixParams.yaml"/>
    </node>
</launch>
```

Bibliography

[1] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *Computer Vision–ECCV 2014*. Springer, 2014, pp. 834–849.

[2] Young Hoon Lee, Tung-Sing Leung, and G Medioni. "Real-time staircase detection from a wearable stereo system". In: *21th International Conference On Pattern Recognition*. IEEE. 2012, pp. 3770–3773.

[3] Georg Klein and David Murray. "Parallel Tracking and Mapping for Small AR Workspaces". In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. Nara, Japan, Nov. 2007.

[4] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *arXiv preprint arXiv:1502.00956* (2015).

[5] Gary Bradski. In: *Dr. Dobb's Journal of Software Tools* (2000).

[6] Mani Monajjemi. *ardrone_autonomy*. 2013. URL: `https://github.com/AutonomyLab/ardrone_autonomy` (visited on 03/30/2015).

[7] Jason M O'Kane. *A Gentle Introduction to ROS*. Jason M. O'Kane, 2014.

[8] *Camera Calibration and 3D Reconstruction*. English. OpenCV. Feb. 25, 2015. URL: `http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html` (visited on 03/20/2015).

[9] Markus Achtelik, Stephan Weiss, and Simon Lynen. *ethzasl_ptam*. 2014. URL: `http://wiki.ros.org/ethzasl_ptam` (visited on 04/16/2015).

[10] Peter Corke. *Robotcs, vision and control: fundamental algorithms in MATLAB*. Vol. 73. Springer, 2011.

[11] James Bowman and Patrick Mihelich. *image_pipeline*. 2015. URL: `http://wiki.ros.org/camera_calibration` (visited on 04/14/2015).

[12] Duane C Brown. "Decentering distortion of lenses". In: *Photometric Engineering* 32.3 (1966), pp. 444–462.

[13] Kurt Konolige, Patrick Mihelich, and Jeremy Leibs. *image_proc*. 2015. URL: `http://wiki.ros.org/image_proc` (visited on 04/14/2015).

[14] Johannes Meyer and Stefan Kohlbrecher. *hector_quadrotor*. 2014. URL: `http://wiki.ros.org/hector_quadrotor` (visited on 03/30/2015).

[15] Hongrong Huang and Jürgen Sturm. *tum_simulator*. 2013. URL: `http://wiki.ros.org/tum_simulator` (visited on 03/30/2015).

[16] David Gossow, Dave Hershberger, and Josh Faust. *rviz*. 2015. URL: `http://wiki.ros.org/rvi` (visited on 03/30/2015).

[17] Wim Meeussen. *robot_pose_ekf*. 2015. URL: http://wiki.ros.org/robot_pose_ekf (visited on 03/30/2015).

[18] Tom Moore. *robot_localization*. 2014. URL: http://wiki.ros.org/robot_localization (visited on 03/30/2015).

[19] Jakob Engel, Jürgen Sturm, and Daniel Cremers. *tum_ardrone*. 2013. URL: http://wiki.ros.org/tum_ardrone (visited on 03/30/2015).

[20] *Hacker guide*. 2013. URL: http://www.nodecopter.com/hack (visited on 03/30/2015).

[21] Xavier Neyt. *State observers*. slideshow. Dec. 22, 2011.

[22] Arnaud Doucet and Adam M Johansen. "A tutorial on particle filtering and smoothing: Fifteen years later". In: *Handbook of Nonlinear Filtering* 12 (2009), pp. 656–704.

[23] Andreas Svensson. *Particle Filter Explained without Equations*. Oct. 8, 2013. URL: https://youtu.be/aUkBa1zMKv4 (visited on 10/12/2014).

[24] Jakob Engel, Jürgen Sturm, and Daniel Cremers. "Scale-aware navigation of a low-cost quadrocopter with a monocular camera". In: *Robotics and Autonomous Systems* (2014).

[25] Jakob Engel, Jürgen Sturm, and Daniel Cremers. "Camera-based navigation of a low-cost quadrocopter". In: *Intelligent Robots and Systems (IROS)*. IEEE,RSJ. 2012, pp. 2815–2821.

[26] Jakob Engel, Jürgen Sturm, and Daniel Cremers. "Accurate figure flying with a quadrocopter using onboard visual and inertial sensing". In: *IMU* 320 (2012), p. 240.

[27] Sarah Y Tang. "Vision-Based Control for Autonomous Quadrotor UAVs". In: (2013).

[28] Yue Sun. "Modeling, identification and control of a quad-rotor drone using low-resolution sensing". In: (2012).

[29] Arren Glover et al. "OpenFABMAP: An Open Source Toolbox for Appearance-based Loop Closure Detection". In: *The International Conference on Robotics and Automation* (2011).

[30] Bill Triggs et al. "Bundle adjustment—a modern synthesis". In: *Vision algorithms: theory and practice*. Springer, 2000, pp. 298–372.

[31] Philip HS Torr and Andrew Zisserman. "Feature based methods for structure and motion estimation". In: *Vision Algorithms: Theory and Practice*. Springer, 2000, pp. 278–294.

[32] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. "DTAM: Dense tracking and mapping in real-time". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2320–2327.

[33] Jakob Engel, Jürgen Sturm, and Daniel Cremers. "Semi-Dense Visual Odometry for a Monocular Camera". In: *IEEE International Conference on Computer Vision (ICCV)*. Sydney, Australia, Dec. 2013.

[34] *tum-vision/lsd_slam · GitHub*. English. TUM. Dec. 11, 2014. URL: https://github.com/tum-vision/lsd_slaml (visited on 12/15/2014).

[35] Jakob Engel. "Autonomous camera-based navigation of a quadrocopter". PhD thesis. Master's thesis, Technical University Munich, 2011.

[36] Xiaoye Lu and Roberto Manduchi. "Detection and Localization of Curbs and Stairways Using Stereo Vision". In: *International Conference on Robotics and Automation (ICRA)* 4 (2005), p. 4648.

[37] Young Hoon Lee, Tung-Sing Leung, and Gérard Medioni. "Real-time staircase detection from a wearable stereo system". Eng. In: *International Conference on Pattern Recognition* (Nov. 11, 2012), pp. 3770–3773. ISSN: 1051-4651.

[38] Jeffrey A Delmerico et al. "Ascending stairway modeling from dense depth imagery for traversability analysis". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (2013), pp. 2283–2290.

[39] Alejandro Pérez-Yus, Gonzalo López-Nicolás, and Jose J Guerrero. "Detection and Modelling of Staircases Using a Wearable Depth Sensor". In: *Assistive Computer Vision and Robotics* (2014).

[40] Stephen Se and Michael Brady. "Vision-based detection of staircases". In: *Fourth Asian Conference on Computer Vision ACCV* 1 (2000), pp. 535–540.

[41] Alfréd Haar. "Zur Theorie der orthogonalen Funktionensysteme. (Erste Mitteilung)." ger. In: *Mathematische Annalen* 69 (1910), pp. 331–371. URL: http://eudml.org/doc/158469.

[42] *Feature Detection and Description*. Eng. OpenCV. Nov. 10, 2014. URL: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html (visited on 04/02/2015).

[43] Mei Fang, Guangxue Yue, and QingCang Yu. "The study on an application of otsu method in canny operator". In: *International Symposium on Information Processing (ISIP)* (2009), pp. 109–112.

[44] Robert Fisher et al. *HYPERMEDIA IMAGE PROCESSING REFERENCE. Image Transforms - Hough Transform*. English. 2003. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm (visited on 10/10/2014).

[45] Jiri Matas, Charles Galambos, and Josef Kittler. "Robust detection of lines using the progressive probabilistic hough transform". In: *Computer Vision and Image Understanding* 78.1 (2000), pp. 119–137.

[46] R Grompone Von Gioi et al. "LSD: a line segment detector". In: *Image Processing On Line* 2.3 (2012), p. 5.

[47] Konstantinos G Derpanis. "Overview of the RANSAC Algorithm". In: *York University* 68 (2010).

[48] Marco Zuliani. "RANSAC for Dummies". In: *With examples using the RANSAC toolbox for Matlab and more* (2009).

[49] Alexander Statnikov, Constantin F. Aliferis, and Douglas P. Hardin. *A Gentle Introduction to Support Vector Machines in Biomedicine: Theory and methods*. A Gentle Introduction to Support Vector Machines in Biomedicine. World Scientific, 2011. ISBN: 9789814324380. URL: https://books.google.be/books?id=UNY8O7-OgOwC.

[50] Paul Viola and Michael Jones. "Robust real-time object detection". In: *International Journal of Computer Vision* 4 (2001), pp. 34–47.

[51] Mohammad Saberian and Nuno Vasconcelos. "Boosting Algorithms for Detector Cascade Learning". In: *Journal of Machine Learning Research* 15 (2014), pp. 2569–2605. URL: http://jmlr.org/papers/v15/saberian14a.html.

[52] James D Foley et al. *Introduction to computer graphics*. Vol. 55. Addison-Wesley Reading, 1994.

[53] *Principal curvature*. English. wikipedia. Oct. 24, 2014. URL: http://en.wikipedia.org/wiki/Principal_curvature (visited on 03/15/2015).

[54] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998. ISBN: 9780132611084. URL: https://books.google.be/books?id=jfAeAQAAIAAJ (visited on 03/02/2015).

[55] Thanh N Tran, Ron Wehrens, and Lutgarde MC Buydens. "Clustering multispectral images: a tutorial". In: *Chemometrics and Intelligent Laboratory Systems* 77.1 (2005), pp. 3–17.

[56] ScotXW. *File:HSA – using the GPU with HSA.svg*. July 9, 2014. URL: http://commons.wikimedia.org/wiki/File:HSA_%E2%80%93_using_the_GPU_with_HSA.svg (visited on 04/02/2015).

[57] Armin Hornung et al. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013). DOI: 10.1007/s10514-012-9321-0. URL: http://octomap.github.com.

[58] Christopher Kemp. "Visual control of a miniature quad-rotor helicopter". PhD thesis. University of Cambridge, 2006. URL: http://ecse.monash.edu/staff/twd/Research/Kemp-thesis.pdf.

[59] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast Semi-Direct Monocular Visual Odometry". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014.

[60] Yi Ma. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. springer, 2004.

[61] Johan Thelin. *Foundations of Qt development*. Vol. 7. Springer, 2007.

[62] Kjell Magne Fauske. *Example: Neural network*. Dec. 7, 2006. URL: http://www.texample.net/tikz/examples/neural-network/ (visited on 05/05/2015).