Höhere Technische Bundeslehranstalt
und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

Autonomous Car Mapping and Tracking

Diploma Documentation

School autonomous focus on Mobile Computing and Software Engineering

Performed in school year 2019/2020 by:

Alexander Voglsperger (AV), 5AHELS

Simon Moharitsch (SM), 5AHELS

Advisors:

Dipl. Ing. Müller Gerhard

April 13, 2020

Thema:
Autonomous Car Mapping and Tracking

Subtopics and Editor:

Implementing SLAMS and DeepTAM, Image Pre-Processing
Alexander Voglsperger, 5AHELS
Advisors: Dipl. Ing. Müller Gerhard

Implementing DeepTAM, Gathering Trainingdata
Simon Moharitsch, 5AHELS
Advisors: Dipl. Ing. Müller Gerhard

Projectpartner:

Designation:	Johannes Kepler University - Artificial Intelligence Lab
Address:	Altenberger Straße 69
ZIP, location:	4040 Linz, Austria
Contact person:	Dr. Nessler Bernhard
Phone:	+43 (0)732 2468 4539
E-Mail:	nessler@ml.jku.at

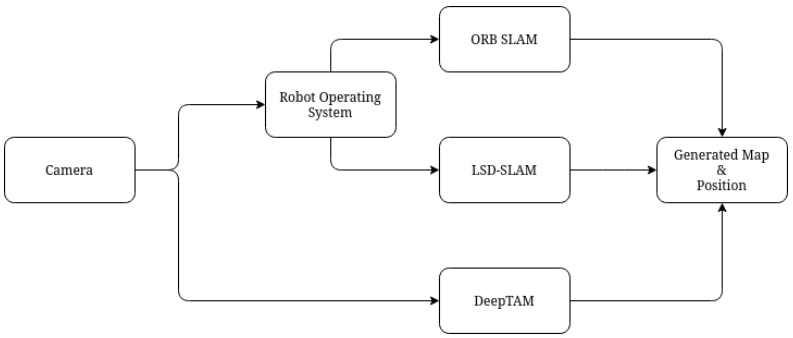
Höhere technische Bundeslehranstalt
und Bundesfachschule Braunau
Elektronik und Technische Informatik
School autonomous focus on Mobile Computing
and Software Engineering

DIPLOMA DOCUMENTATION

Author	Alexander Voglsperger, Simon Moharitsch
Vintage School year	5AHELS 2019/2020
Topic of the diploma documentation	Autonomous Car Mapping and Tracking
Cooperation-partner	Johannes Kepler University - Artificial Intelligence Lab
Task definition	The task of this work is to get depth information out of a video stream in real time. The considered field of application is an autonomous driving car that uses several cameras to orientate itself while driving. In a first approach the Robot Operating System (ROS) sends the video of a camera to two Simultaneous Localization And Mapping (SLAM) algorithms which gather depth information out of the images. This results in a point cloud which represents the detected surroundings. Another approach is DeepTAM, a method which implements artificial intelligence methods to estimate distances between objects in each frame of a video stream. As DeepTAM has not been updated for a while there are many compatibility issues with newer drivers, newer Tensor Flow framework and other required libraries. Thus the task is to fix the compatibility issue.
Realization	Simultaneous Localization And Mapping (SLAM) algorithms are implemented on the free available platform Robot Operating System (ROS). ORB-SLAM and LSD-SLAM are two specific SLAM methods. These two SLAMs have been modified to get executable program code on ROS. Several performance tests and a comparison between ORB-SLAM and LSD-SLAM are also part of this work.
Outcome	In this thesis the results of ORB-SLAM and the LSD-SLAM are compared. The ORB-SLAM is working, but will not produce a detailed map if prominent points are missing in the video. Prominent points are characterized by sharp edges or corners of an object. SLAM methods use these prominent points to get the depth information in a 2D video stream. The LSD-SLAM does not work as good as the ORB-SLAM when the camera only has a axial movement in the sense that it requires both axial and rotational movement to work and thus would generate an incorrect map. Otherwise the LSD-SLAM only works on cars when a wide-angle camera lens is used. DeepTAM works on Ubuntu 16.04. Since this is not up to date, most libraries need to be downgraded, which creates driver problems. To get DeepTAM installed on the current version of Ubuntu, the source code of DeepTAM has to be adapted.

Illustrative graph, photo (incl. explanation)

Structure of the flowchart:



Accessibility of diploma thesis

HTL Braunau archive, or
`https://diplomarbeiten.berufsbildendeschulen.at/`

Approval (date / signature)

Examiner	Head of College / Department

Statement

I declare in lieu of oath that I have written this diploma thesis independently and without outside help, have not used sources and aids other than those stated directly and have made the sources used verbatim and in terms of content taken as such recognizable.

Braunau/Inn, 13.04.2020	Alexander Voglsperger	_____
Location, Date	*Author*	*Signature*
Braunau/Inn, 13.04.2020	Simon Moharitsch	_____
Location, Date	*Author*	*Signature*

Contents

Abstract

An autonomous driving car has to orientate in the immediate vicinity. For this purpose the car needs a map and the own position in the map. Why is this such an important task when we already have a technology like Global Positioning System (GPS)? GPS provides routing data, but no live details what happens on the street, like an ongoing construction or a pedestrian crossing the street. To get this information the car has to collect visual data from its environment in real-time. This thesis examines two approaches of visual data processing to generate a map and locate the car inside this map. One approach is Simultaneous Localization And Mapping (SLAM), the other one uses artificial intelligence. The platform for ORB-SLAM and LSD-SLAM is the Robot Operating System (ROS), which handles the publish/subscriber based communication between the camera and the SLAM algorithms.
DeepTAM uses Deep Neural Networks which can learn to predict the distance between objects in an image. The prediction is based on training data. The accuracy of the result depends on the amount of training data. It is a system for keyframe-based dense camera tracking and depth map estimation that is entirely learned. This thesis shows how to implement the different algorithms in ROS and points out the pros and cons of each method compared to the others and gives an outlook into future methods.

Ein selbstfahrendes Auto muss sich in der unmittelbaren Umgebung orientieren. Dazu benötigt das Auto eine Karte der Umgebung und die eigene Position. Warum ist dies so eine wichtige Aufgabe, wenn wir bereits über Technologien wie das Global Positioning System (GPS) verfügen? GPS liefert zwar die Standortdaten, aber keine aktuellen Details darüber, was auf der Straße passiert, wie z.B. eine Baustelle oder ein Fußgänger der die Straße überquert. Um diese Informationen zu erhalten, muss das Auto visuelle Daten aus seiner Umgebung in Echtzeit sammeln. In dieser Arbeit werden zwei Ansätze der visuellen Datenverarbeitung untersucht, um eine Karte zu erstellen und das Auto in dieser Karte zu lokalisieren. Ein Ansatz ist Simultaneous Localization And Mapping (SLAM), der Andere verwendet eine künstliche Intelligenz. Die Plattform für den ORB-SLAM und den LSD-SLAM ist das Robot Operating System (ROS), das die Publisher/Subscriber-basierte Kommunikation zwischen der Kamera und den SLAM-Algorithmen übernimmt. Der DeepTAM verwendet ein Deep Neural Network, welches trainiert wurde den Abstand zwischen Objekten in einem Bild vorherzusagen. Die Genauigkeit der Vorhersage beruht auf der Anzahl der Trainingsdaten. Der DeepTAM verwendet ein angelerntes System welches mit Keyframe-basiertem Dense Camera Tracking und Tiefenschätzung eine Karte generiert. Diese Arbeit zeigt die Funktionsweise der verschiedenen Algorithmen, beleuchtet deren Vor- und Nachteile und gibt einen Ausblick auf zukünftige Methoden.

Summary

This paper starts with the introduction into Simultaneous Localization and Mapping (SLAM) in general and the ORB-SLAM and LSD-SLAM in more detail.
Both methods need camera data. Therefore, the Basler camera is explained in chapter 5. This image data needs to be collected and transferred to the SLAM methods. For this task the Robot Operating System (ROS) is used. Chapter 2 explains ROS, which provides the platform for the different SLAM methods. ROS is a publish/subscribe based system. In chapter 6 a more detailed look into the ORB-SLAM is taken to make it easier to understand the concept of the LSD-SLAM in chapter 7. The introduction into ORB-SLAM and LSD-SLAM includes how the algorithm work, what data they require and in which way they can be used. Both algorithms provide the current position and a pointcloud which can be transferred into ROS which has a tool called Rviz 2.4.4 to visualize the pointcloud with the current position in it.
In contrast to the first two methods, DeepTAM works with Artificial Intelligence. Therefore, Artificial Neural Networks (ANN) are needed which are explained in chapter 3. This chapter explains the components and the most important types of Artificial Neural Networks for DeepTAM: Feed Forward Neural Networks and Convolution Neural Networks. The next chapter is Deep Learning. There are three different ways Deep Learning can be implemented: Supervised, Semi-Supervised and Unsupervised. This all leads to chapter 8 where DeepTAM itself is explained.
Chapter 9 deals with the project workflow where the used hardware and software is listed, setup, and how the errors where fixed during the implementation.

1 SLAM^{AV}

1.1 What is SLAM?

SLAM is an acronym and stands for **S**imultaneous **L**ocalisation **A**nd **M**apping. *"SLAM is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time navigating the environment using the map."* [1]
This problem is thus a chicken-and-egg problem because neither the map or location are known, and have to be estimated at the same time. Cameras, ultrasonic sensors and laser radar (Lidar) sensors are most commonly used for fetching 2D images or points in a 3-Dimensional space of the robot's surroundings [2].
There are several algorithms out there, which try to solve this problem using algorithms and some even deep learning. Mostly they achieve an approximate map, which is done in a reasonable time span. Many popular SLAM-algorithms use methods that include *particle filters*, *extended Kalman filter* and *co-variance intersection*[1] [3].

1.2 Application

The biggest selling point for using SLAM implementations is pretty simple. Many places where autonomous robots may be required do not have good enough maps that are up-to-date , if they exist at all. There might also be in an environment where positioning for instance GPS cannot be used properly because the environment faces frequent changes, e.g parked cars or passengers [4]. If SLAMs were not be used then someone would have to go to the place and make a map. This would delay the mission and add to the costs.
With a robot, that is capable of using a SLAM method to detect and locate itself in the unknown surroundings this wouldn't be an issue. The robot could go in, generate a map that updates itself and use it to navigate around.
Existing approaches that are used are in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers and newer domestic robots [5].

1.3 History

The fundamental research in SLAMs was done in the research by R.C. Smith and P. Cheeseman who worked on the representation and estimation of spatial uncertainty in 1986 [1]. Another major work in this area was done by a research group with the head being *Hough F. Durrant-Whyte*. Durrant-Wyhte and his group showed in their paper [3] that the answer to SLAMs lies in the nearly infinite amount of data that can be used. This lead to the mo-

tivation of finding algorithms which are trackable and approximate in a time realistic manner.

1.4 Existing Methods

There exits a big variety of SLAM methods, that try to achieve the same goal using different approaches [6]. Most known or popular are the following:

- EKF SLAM
  Utilizes the extended Kalman filter. The algorithm uses the likely-hood for data association. It was the favored SLAM from 1990 to the early 2000s until Fast SLAM was introduced [7].
- Fast SLAM
  Works recursively so it scales logarithmically to the scale of the landmark. It can handle much bigger landmarks than the EKF-SLAM ever could without requiring as much computing power [7].
- ORB-SLAM2
  It's a real-time SLAM library designed for Monocular, Stereo and RGB-Depth (RGB-D) cameras. It can detect loops if it already was at an area before and locate the camera in real-time. It uses camera trajectory and sparse 3D reconstruction to get information out of the image sequence [8].
- DVO-SLAM
  Implements a *dense visual SLAM* system for RGB-D cameras. It's based on *Dense Visual Odometry* and was extended to include frame-to-key matching with loop closure to older key-frames [9].
- RGB-D SLAM
  Utilizes the depth information of a RGB-D camera, e.g., Microsoft Kinect or Intel Real-Sense Cameras [10].
- LSD-SLAM
  It's a direct monocular SLAM. It tracks the *direct image alignment* and estimates geometry in form of *semi-dense depth map* instead of relying on keypoints [11].

2 Robot Operating System[AV]

2.1 What is the Robot Operating System?

The Robot Operating System[1], which is also known as ROS™is a flexible framework for writing software that gets utilized on robots. It was founded by Willow Garage in 2012 and gets primarily maintained by the Open Source Robot Foundation (OSRF) [12]. In Europe the project gets coordinated by the Fraunhofer IPA in form of the ROS Industrial Consortium Europe [13]. ROS is a middle-ware which is not an operating system but provides services that handles passing messages between processes, package management and hardware abstraction. *"It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."* [14]

Figure 2.1: "Nine dots"™ROS logo

2.2 Design

The processes of ROS are represented in nodes which are in a graph structure. Everything gets managed by a single process called *ROS Master*, to whom all other nodes register on startup, tell what they publish or on which topic they want to subscribe on. But instead of sending all of the messages over the master, the master sets up a peer-to-peer connection between the nodes which then looks like figure 2.2. Services are a bit different since a node directly connects to another node using the service instead of a publicly available topic. This decentralized architecture is helpful as many robots consists of many computer hardware which is connected via a network and are likely to transfer big messages [15].
For example the publisher and subscriber nodes in figure 2.2 first register at the ROS master. Afterwards the publisher node publishes data like a image on a specific topic like *image*. When a node wants to subscribe to a certain topic the ROS master first checks if another node is publishing on the topic. If that is not the case the ROS master will return an error. If the topic gets published the master creates a connection between the publisher and the subscriber. The messages that are able to contain pretty much every datatype are set over such a connection until. It is important to keep in mind that the publisher node only starts

[1]ROS and/or the "nine dots" ROS logo and/or any other ROS trademarks are a trademark of Open Robotics

sending messages when another node actually subscribes an thus saves bandwidth when a publisher node has no subscribers.

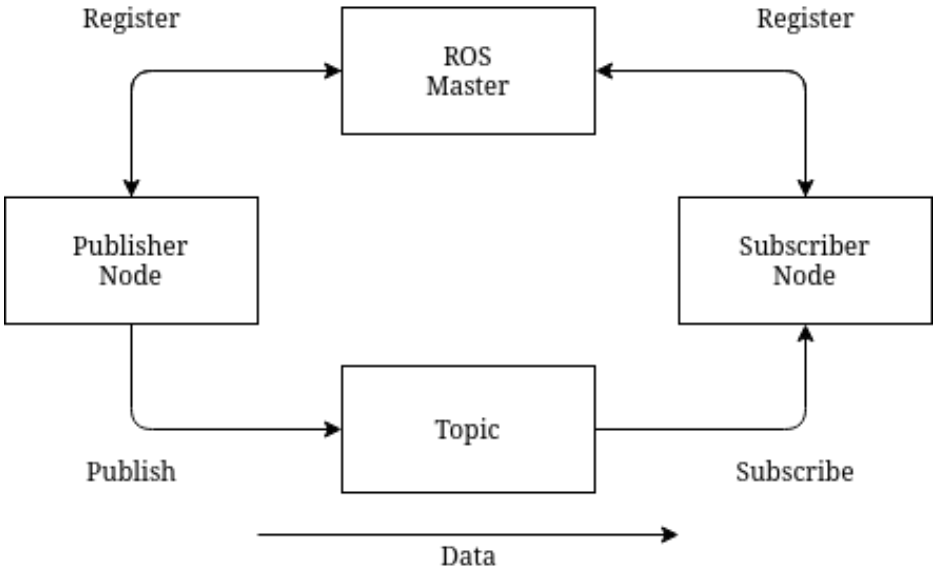


Figure 2.2: ROS structure

2.2.1 Topics

It is based on a topic system, where a topic acts like a bus over which nodes send and receive messages. Each topic must be unique in its name, which is usually set by the developer. The process of publishing and subscribing is handles anonymously so that no node knows which nodes are sending and receiving messages on a certain topic. When a node wants to subscribe onto a specific topic it communicates with the *ROS master*. If another node publishes on this topic the master connects the two nodes with each other in a peer-to-peer design.

2.2.2 Nodes

A node, which represents a single running process, can provide data using a matching topic and publishes it to the system, where theoretically every other node can subscribe to it, to get the data.

2.2.3 Services

Nodes can also advertise services which contain a single action. For example a node that publishes a video might contain a service which only returns the first, current or last frame. They are used for commands that are not used frequently or don't process much data.

2.2.4 Parameter server

A parameter server is like a database where all nodes have static or semi-static access to information. The data can be settings or fixed variables like distance or weight.

2.3 Licenses and OS

The language-independent tools and the main client libraries have been released under the BSD license and as such they are open source software for commercial and research use. The majority of 3rd party packages are released under several other open-source licenses.
The ROS libraries are geared toward a UNIX-System which is mainly due to their dependence on a large collection of open source software and libraries. For example *Ubuntu* is in the list of supported operating systems, while others like *Fedora, Mac OS and Windows* are "experimental" and are mainly supported by the community [16].

2.4 Tools

One of the core functionalities that ROS provides are the tools which allow the developers to visualize 2D and 3D data, record data, easily navigating ROS packages, creating complex scripts that configure and setup processes. This tool simplifies the workflow and provides solution for common robotic development.

2.4.1 Rosbag

Rosbag is a tool that can be used via the command line to record, playback and store ROS message data. When starting a recording all the published data gets stored in a so called bag-file. This makes it possible to record all published topics and then artificially run them later again. By doing this the recorded messages get published into the system, as if they were live. It's very handy if you need data for later development or to test nodes on different scenarios that come across later.

2.4.2 RQt

RQt provides a graphical overview of the ROS computation graph. It shows the nodes and how they are connected to each other. It also shows if a node is even subscribing to a topic or publishes something. Other than that it can be used to subscribe to different topics and show them directly in RQt.

2.4.3 CatKin

Catkin is the newer ROS build system, which compiles the files in the source folder. It is based on CMake and is cross-platform and language-independent as most other ROS tools.

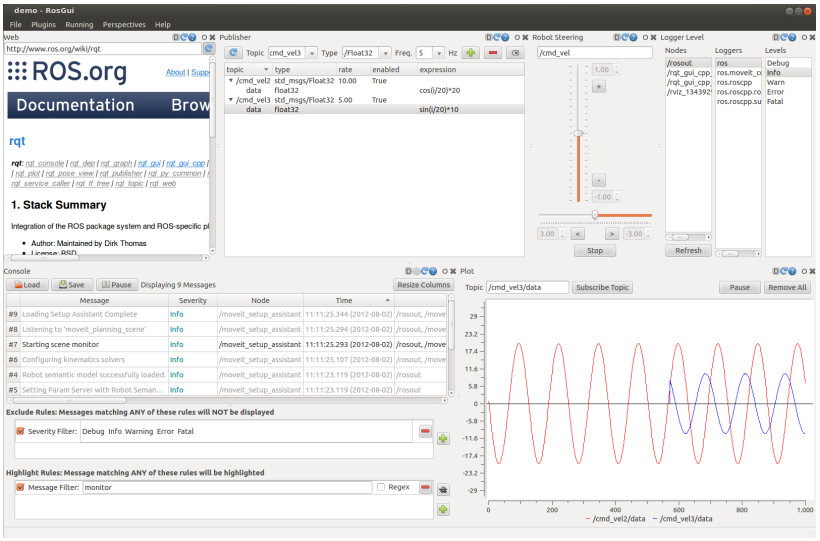


Figure 2.3: RQt interface
Src: `https://wiki.ros.org/RQt`

2.4.4 Rviz

A visualizer for three-dimensional position-data where robots, environments and sensor information can be visualized. It is highly customizable and is able to display many different types of incoming data. The typical interface on startup looks similar to figure 2.4. On the left are the subscribed topics and on the right are settings for displaying the incoming information.

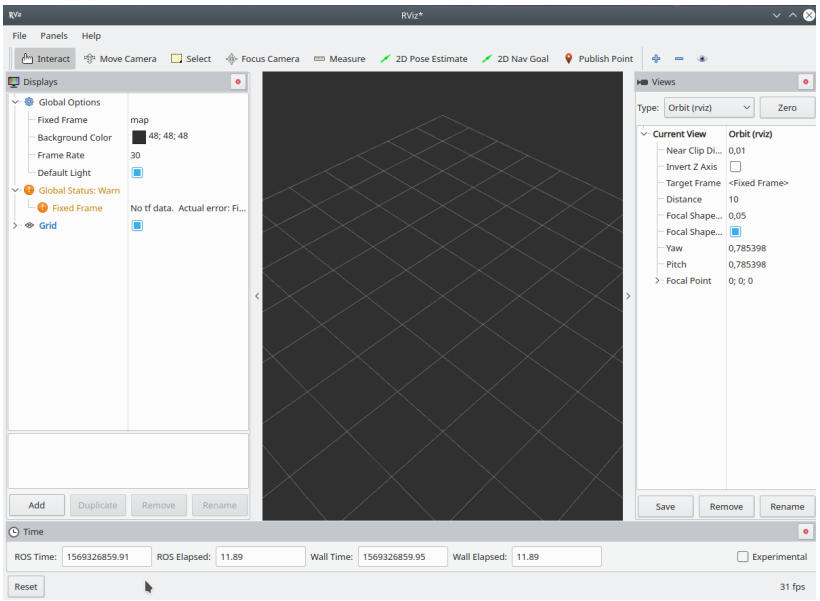


Figure 2.4: Rviz interface

2.4.5 Roslaunch

Roslaunch is a tool for launching multiple ROS nodes and setting parameters on startup. It can be used to launch nodes locally or remotely on a server. The configuration for a start script is written in a launch file using XML. In these files it is easy to make a automated startup and configuration process to be executed with one command. It is also possible to execute launch files in other launch files to chain them together and therefore start a complete system in the correct order with a single launch file.

3 Artificial Neural Networks[SM]

3.1 What is a Artificial Neural Networks?

Artificial **N**eural **N**etworks (ANN) are inspired by biological neural networks that constitute animal brains. Important to notice is that they are not faithful models of biologic neural or cognitive phenomena. In fact most of these models are more closely related to mathematical and/or statistical models (For Example: clustering algorithms). Clustering algorithms task is to group a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.

3.2 Areas of Application

ANN are viable computational models for a wide variety of problems, including pattern classification, speech synthesis and recognition, adaptive interfaces between human and complex physical system, function approximation, associative memory, clustering, forecasting and prediction, combinatorial optimization, nonlinear system modeling, and control [17]

3.3 Components of an ANN

Simplified a ANN consists of neurons, connection and the weight associated with them, the propagation function, a loss function and a bias. The following topics will give you an introduction what these components represent and how they interact.

3.3.1 Neurons

Neurons are elementary units in an ANN. A neuron gets one ore more inputs and depending on the value of the inputs the output is set to a value between 0 and 1. A neuron can get its inputs from other neurons or, if its at the beginning, from the source of the data that needs to be processed. Depending on the application the ANN is needed for they are placed in different structures.

3.3.2 Connection and weights

Neurons are connected to neurons in the next layer. It depends on the structure which neurons are connected with which neurons in the next layer. The weights characterize how

important a connection between neurons is.

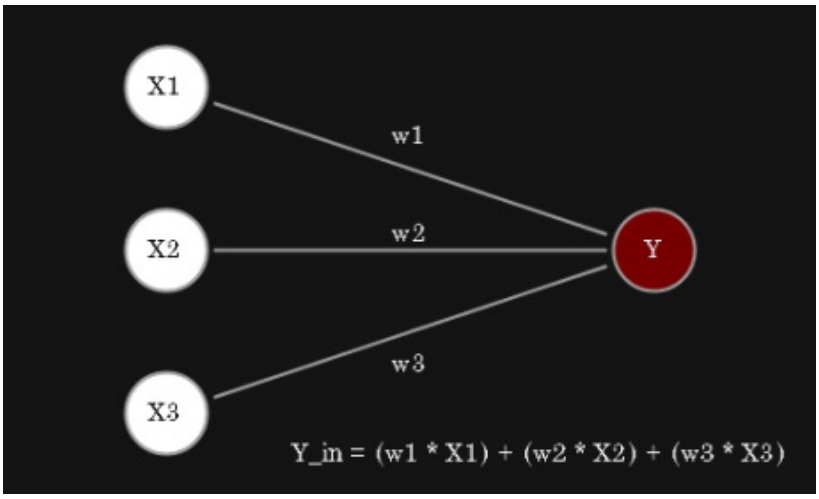


Figure 3.1: Simple Picture to explain Weights
Source: https://tinyurl.com/smw9bk4

For example:
A neuron, we call it Y for this example, has three neurons connected to it as inputs. These neurons are called X1, X2 and X3. The weight of the connection of X1 has a bigger weight than the connections X2 and X3. That means the output of Y depends more on the input of X1 because the input from Y is made up from the three input neurons times their weights summed up.

3.3.3 Propagation function, activation function and Bias

This is a function which takes the Inputs of a neuron, the weight of these connections and the bias and adds them up. The resulting value is the processed by the activation function which sets the output. One of the most common activation function is the sigmoid function because it is not a step function which means the output does not change instantaneously. That is important for the training algorithm because with that the output of a neuron can not only be 1 or 0. It can be any number between 1 and 0 and give an answer for how certain it is. Therefore the training will be easier because it is easier to find out in which directions and how much the weights should be changed. The bias is a Neuron which has no Inputs. A bias is used to shift the decision boundary to the left or right.

3.4 Organization

A artificial neural network can be organized in many different ways.

3.4.1 Feed Forward ANN

The following picture demonstrates a feed forward ANN .There are a variable number of hidden layers depending on the purpose of the neural network. Nothing in the hidden layer

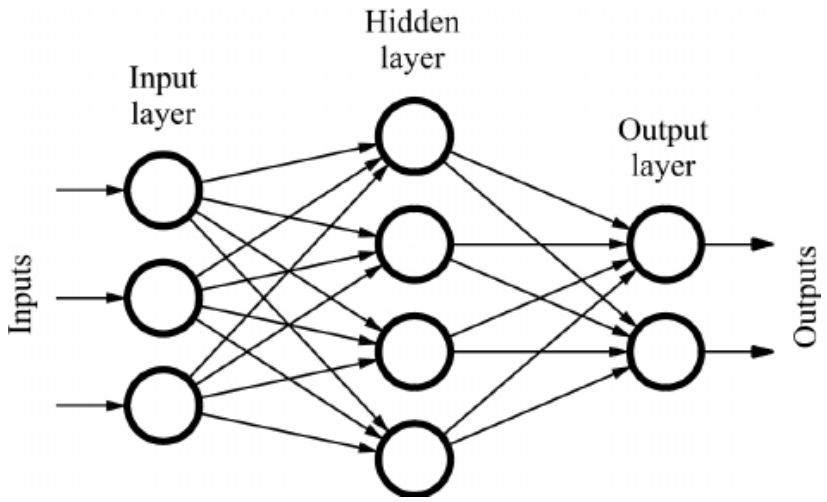


Figure 3.2: Feed forward network

is visible.

3.4.2 CNN

Convolution Neural Network (CNN) is the most important Network organization for this task. It is based on the human visual cortex and are optimal for image and video recognition. The components of a CNN are a series of convolution and sub-sampling layers followed by a fully connected layer and a normalizing layer.

How a CNN works will be explained in the following example. The same example like in "Review of Deep Learning Algorithms and Architectures"[18] will be used

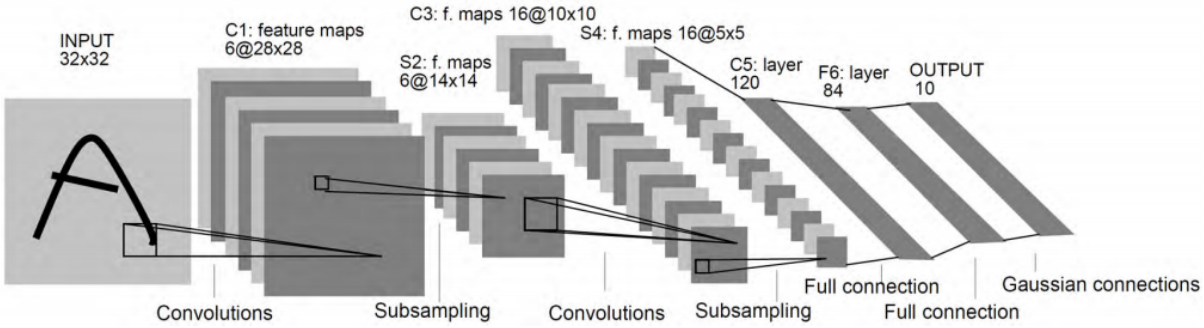


Figure 3.3: 7-layer architecture of CNN for character recognition
[18, Fig. 4.]

Progressively more refined feature extraction at every layer is performed by the series of multiple convolution layers. This process is moving from input to output layer. After the convolution layers there are fully connected layers that perform classification. There is also the possibility of putting Sub-sampling or pooling layers between each convolution layer. The input of a CNN is a 2D n*n pixelated image. Each layer consists of filters or kernels

(groups of 2D neurons). In most neural networks neurons in each feature extraction layer are connected to all neurons in the adjacent layers. But in a CNN they are only connected to the spatially mapped fixed sized and partially overlapping neurons in the previous layer's input image or feature map.

3.5 Encoder-Decoder-Based Architecture

Encoder-Decoder-Based networks are networks, which try to reconstruct their own input. You construct the network so that it reduces the input size by using one or more hidden layers, until it reaches a reasonably small hidden layer in the middle. As a result your data has been compressed (encoded) into a few variables. From this hidden representation the network tries to reconstruct (decode) the input again.

In order to do a good job at reconstructing the input the network has to learn a good representation of the data in the middle hidden layer. This can be useful for dimensionality reduction, or for generating new "synthetic" data from a given hidden representation. This is especially important considering that we are trying to construct a 3D Map out of the Camera Input.

4 Deep Learning^{SM}

4.1 What is Deep Learning?

4.2 Supervised Learning

4.3 Semi-Supervised Learning

4.4 Unsupervised Learning

4.5 Applications

5 Basler Camera[AV]

One method that the AADC 2018 car uses to observe its surroundings is a camera. Without the car, another camera system has to be used for recording images. The camera on the AADC 2018 car is a *daA1280-54uc (S-Mount) - Basler dart* by the Basler AG [19].

5.1 Information

The Camera is a 1.2 Megapixel CMOS camera that uses a USB 3.0 interface. It can do 54 FPS and features a global shutter which is very help full [20].

Figure 5.1: Basler Camera
Src: `https://tinyurl.com/ss4mwzq`

5.2 Prerequisites

To use a Basler Camera in general it is required to have the so-called *pylonSDK* installed which can be downloaded from the official Basler homepage [21].

5.3 Running pylon-ros-camera node

The added node that fetches the video from the camera and then publishes it is called *pylon-ros-camera* and is developed by the Basler AG since October 1st 2019 [22]. Before a similar node was developed by the Magazino GmbH.[23].

5.3.1 Compiling node

1. Cloning Pylon Ros Camera
   First clone the Github Repository by the [22] Basler AG into a workspace.
2. Cloning DragAndBot_common
   The DragAndBot_common repository contains *DragAndBot* messages and services that the pylon-ROS-camera node requires. So clone the *DragandBot_common* repository [24] into the same workspace as the Pylon-Ros-Camera.
3. Installing ROS dependencies
   To install the necessary dependencies for the ROS nodes using the command [**?**].

```
sudo sh -c 'echo "yaml https://raw.githubusercontent.com/basler/pylon-ros-
    camera/master/pylon_camera/rosdep/pylon_sdk.yaml" > /etc/ros/rosdep/
    sources.list.d/30-pylon_camera.list' && rosdep update && sudo rosdep
    install --from-paths . --ignore-src --rosdistro=$ROS_DISTRO -y
```

Listing 5.1: Installing dependencies for Pylon-Ros-Camera

4. Compiling and Sourcing
   To compile the nodes run *catkin build* or *caktin_make* in the root directory of the workspace.
   To use the node when compiling has finished it is required to enter *./devel/setup.bash* when being in the root directory.

5.3.2 Using Pylon-Ros-Camera Node

Running the node is as simple as launching it by using

```
roslaunch pylon_camera pylon_camera_node.launch
```

This command published the video feed, which other nodes then can subscribe to.

6 ORB-SLAM2^{AV}

6.1 What is ORB-SLAM?

ORB-SLAM2 is a versatile, real-time SLAM implementation which uses Mono-, Stereo- and RGB-D cameras. It's designed to generate a 3D map from prominent points in the picture and keypoints. It features loop closing, re-localization and a reusable map [25]. It works in a wide variety of use cases. The image that the SLAM uses can be gathered from small hand-held cameras, cameras on drones for top-down mapping and cameras on cars for self driving. ORB-SLAM2 is based on ORB-SLAM and was inter alia developed by Raúl Mur-Artal who already worked on ORB-SLAM.

Figure 6.1: ORB-SLAM Example image
Src: `https://tinyurl.com/ruvnj39`

6.2 How does the ORB-SLAM work?

The SLAM is made out of multiple blocks which all handle a different type of operation. These operations range from extracting the keypoints to localization.

6.2.1 Extracting Keypoints

The SLAM uses a feature-based method. This means that it extracts features on prominent keypoints throughout the image input. These feature information is then distributed to all operations which handle them independently from the camera type. [25]

This is how finding these keypoints works on different camera types:

- Stereo Image
  For a stereo camera setup the keypoints get extracted from the image using a survival of the fittest process, were the most prominent points stay until a threshold is reached. This is done for both images separately and then the left keypoints are searched on the right image. Then the found points get compared to the original ones that where found on the right side
- RGB-D Image
  On a RGB-D camera keypoints get extracted using prominent keypoints and then calculating the approximate position using the depth information from the information from the depth sensor.
- Monocular Image
  On a Monocular image the approximate position gets triangulated by using multiple images. The disadvantage is that they don't provide a scale information and only do rotational and transnational movement estimations.

6.2.2 Loop-closing and Bundle Adjustments

Loop-closing and bundle adjustments are performed in two steps. First the loop-closing will happen when the system detects overlapping environments where the system changes scaling to reconnect certain parts as scale drifting will occur on monocular cameras.
Second step is the bundle adjustment, which gets executed after a successful loop-closing, where the system tries to optimize all keypoints and keyframes using the Levenberg-Marquardt method (alternative to Gauss-Newton method) [26]. This optimization like seen in 6.2 tries to adjust the map in a way it makes sense again. Also the camera orientation and position will be optimized to compensate errors in tracking. All the bundle adjustment is done in a separate thread since this is a more computational task.
When finished, the updated and optimized keyframes and keypoints get merged into the original keyframes and keypoints [25].

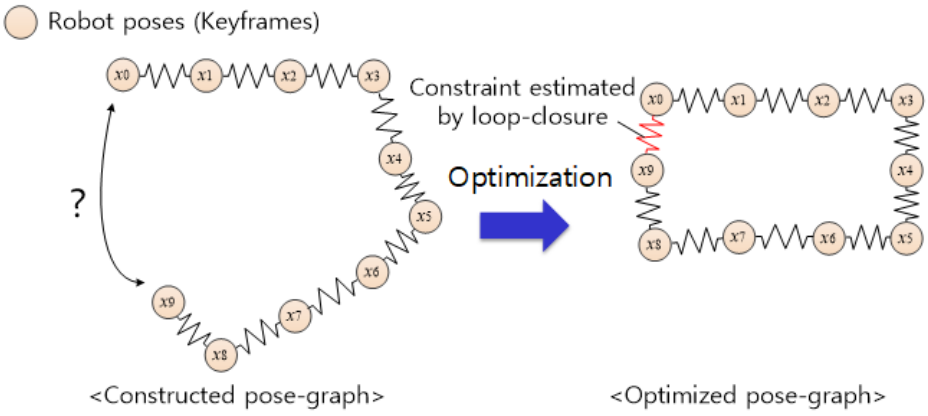


Figure 6.2: Loop-Closing example
Src: `https://tinyurl.com/to4678g`

6.2.3 Localization

When an area has been mapped well in the past the *Localization Mode* can be turned on which deactivates the Local Mapping and the Loop Closing thread and this saving computing power. Locating is done by continuously comparing the previous points with the current points of the image. This works when an area is unmapped but drifting might add up. Matching the current points with the one on the map will ensure that it is drift-free [25]. There are a few points that are pretty important to keep in mind to achieve good results. The points can be viewed at 9.7.

6.2.4 Input/Output

Input Data: rectified Monochrome/Color images
Output Data: Rough 3D map with pixel-points and image with current prominent key-points

7 LSD-SLAM^{AV}

7.1 What is LSD-SLAM?

LSD-SLAM stands for **L**arge-**S**cale **D**irect Monocular SLAM and is a fairly new real-time monocular SLAM that is fully direct-driven instead of relying on keypoint/keyfeature [11]. The algorithm works on the image intensity for tracking and mapping at the same time. This method allows building large-scale maps on normal processors that are consistent when comparing it to current state-of-the-art algorithms.

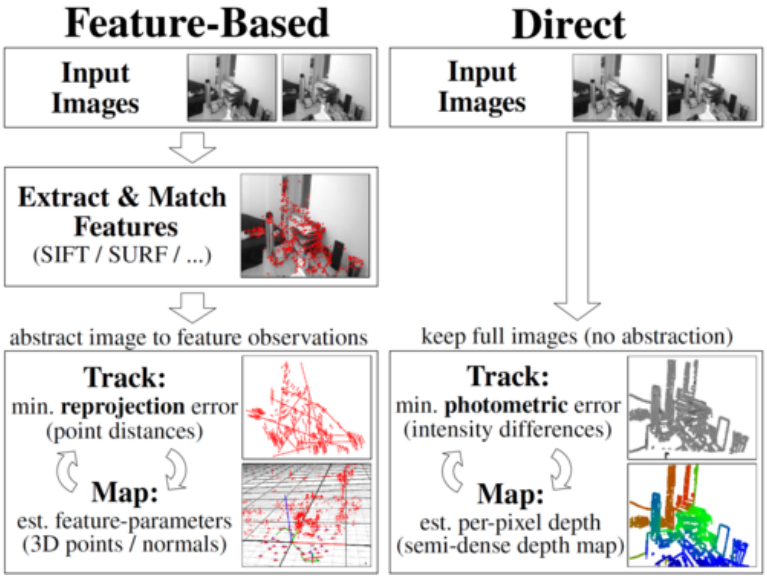


Figure 7.1: Feature-Based and Direct Difference
Src: `https://tinyurl.com/ycmjhb9d`

7.2 Difference Feature-Based and Direct

- Feature-based
  A feature-based SLAM (e.g. ORB-SLAM 6) looks for distinctive points in the image and then uses only these keypoints to process the information. When there are only a few prominent keypoints (e.g indoor, tunnels) the result will not be that accurate.
- Direct
  Direct based SLAMs use all information that's provided in the image. This does not

only include distinctive points but also edges and sometimes surfaces and thus makes it possible to create a more accurate and denser 3D map [11].

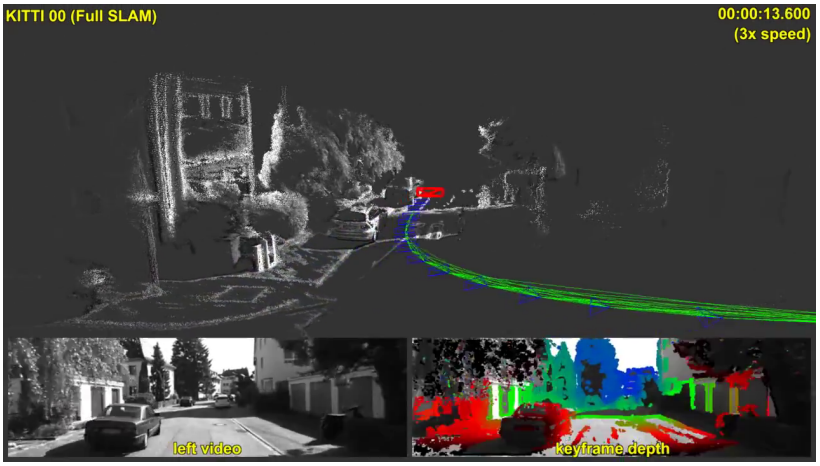


Figure 7.2: LSD-SLAM Example image
Src: `https://tinyurl.com/qkuamyy`

7.3 How does the LSD-SLAM work?

7.3.1 Components that make up the LSD-SLAM

- Tracker
  The *Tracking* component uses the current frame in relation to the last frame to continuously track the camera movement.
- Depth Map Estimation
  Depth map estimation is done using tracked frames to refine or replace the current frame. Depth information is calculated by filtering over a small per-pixel baseline. If the camera has moved too far or the image has changed too much a new keyframe is initialized [11].
- Map Optimization
  When a keyframe gets replaced as a tracking reference, the refinement process stops and it gets included in the 3D depth map. *Map optimization* then starts working to detect loop-closures or scale-drifts. This is done by a similarity transformation to frames that where taken nearby.

7.3.2 Depth Map Estimation

New keyframes are created when there where no frames before or the camera has moved/rotated so far that the set threshold has been exceeded. When this happens the new latest frame is chosen to become the new keyframe and the keypoints from the previous keyframe get projected onto the new one. The process is followed by scaling to fit the needs of the Direct Image Alignment. When that is done the keyframe replaces the previous ones and gets used to track the subsequent frames.

Not every frame results in a new keyframe. Frames that don't make it to a new keyframe are used to improve and refine the current keyframe. The refinement is done by using a small stereo comparison for regions in an image where the expected use for advancement is higher. The result of this comparison then gets merged into the existing 3D point cloud to add potentially new information or refining existing pixels [11].

7.3.3 Map optimization

Scaling, rotation and movement is not always perfect, which results in drift. Even if the drift effect is little it adds up, which might result in a very distorted map [11]. The Pose-Graph-Optimization is an optimization algorithm which aims to fix these drifts with pretty good results. The advantages of the pose-graph-optimization are that it is fast and vulnerable for poor initialization estimates [27].

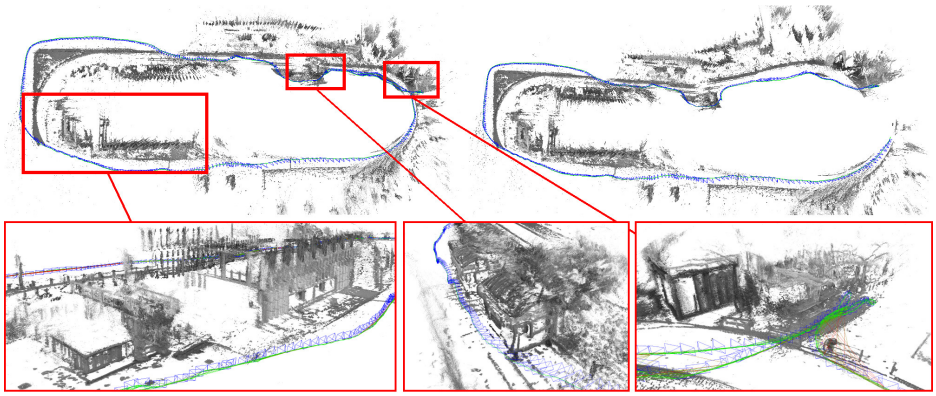


Figure 7.3: LSD-SLAM Pose-Graph-Optimization (left after loop-closure, right before loop-closure)
Src: `https://tinyurl.com/qkuamyy`

7.3.4 Input/Output

Input Data: rectified monocular image, camera info
Output Data: image with probability colored points, 3D point cloud

8 DeepTAM[SM]

This Chapter is based on the work of:"DeepTAM: Deep Tracking and Mapping with Convolutional Neural Networks".[28]

8.1 What is DeepTAM?

DeepTAM provides a keyframe-based dense camera tracking and depth map estimation system that is entirely learned. The idea of DeepTAM is based on DTAM [29].The generic idea is: drift-free camera tracking via a dense depth map towards a keyframe and aggregation of depth over time. But the way to implement this concept is different. In the DeepTAM deep networks are used for tracking and mapping.These networks learn only from data. It also processes more than two images for the 6 DOF egomotion and depth estimation. With that it can avoid the drift of the use of keyframes and as more keyframes come in it can refine the depth map.

8.2 Tracking

The main objective is to estimate a 4 x 4 transformation matrix T. This matrix maps a point in the keyframe coordinate system to the coordinate system of the current camera frame. DeepTAM uses an more efficient way. It generates a virtual keyframe and tries to predict the increment instead of trying to estimate T. For more details see [28].

8.2.1 Network Architecture

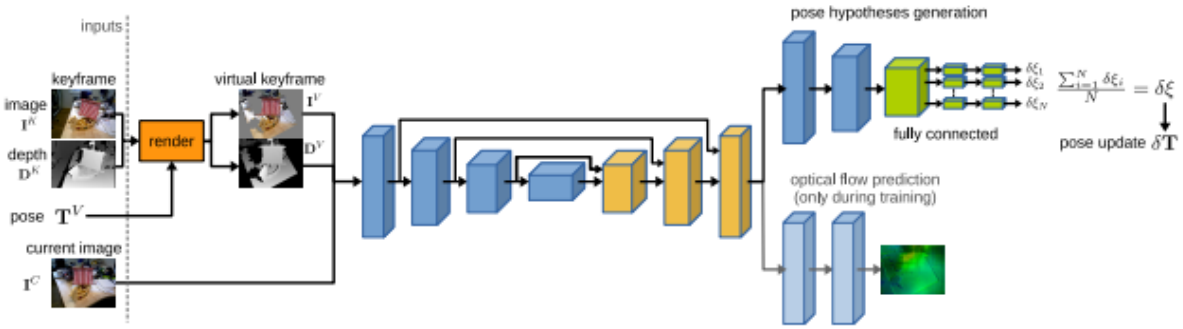


Figure 8.1: Schematic of DeepTAM
Src: http://lmb.informatik.uni-freiburg.de/Publications/2019/ZUB19a

To estimate the 6 DOF pose between a keyframe and an image the encoder-decoder-based architecture is used. To estimate camera motion you have to relate the keyframe to the current image. Because of that DeepTAM uses optical flow as an supportive task. With this optical flow the network is ensured to take advantage of the relationship between both frames. It uses two network branches for predicting the pose. One is the optical flow prediction and the other is the pose hypotheses generation. This improves the accuracy for the pose prediction.

8.3 Mapping

DeepTAM computes a set of depth maps every keyframe. For good quality depth maps, information will be accumulated in a cost volume. From this cost volume the depth map will be extracted by means of a convolutional neural network.

Normally the cost volume is taken as data term and because of that a depth map can be obtained by searching for the minimum cost. Using this method, because of the noise in the cost volume, there must be various optimization techniques and sophisticated regularization terms included to extract the depth in a robust manner.DeepTAM instead has a network which is trained to use the matching cost information in the cost volume and simultaneously combine it with the image-based scene priors to obtain more accurate and more robust depth estimates.

The accuracy is limited by the number of depth labels for cost-volume-based methods. That is why there is an adaptive narrow band strategy used to keep number of labels constant while increase the sampling density. The cost volume for the narrow band recomputes for a small selection of frames and searches again for a better depth estimate. The narrow band requires a good initialization and regularization to keep the band in the right place but allows to recover more details in the depth map.

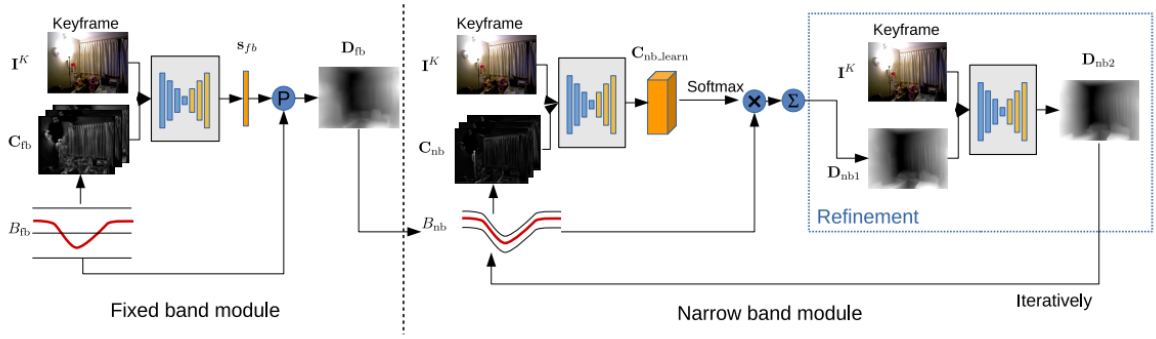


Figure 8.2: Mapping Networks Overview
Src: http://lmb.informatik.uni-freiburg.de/Publications/2019/ZUB19a

8.3.1 Network Architecture

The network is trained to predict the keyframe inverse depth from the keyframe image and the cost volume which is computed from a set of images and camera poses. The keyframe inverse depth is represented as inverse depth, which enables a more precise representation with closer distance. There is also a coarse-to-fine strategy along the depth axis applied. Mapping is divided into a fixed band module and a narrow band module. The narrow band cost volume centers at the current depth estimation and accumulates information in a small band close to the estimate, while the fixed band module builds a cost volume with depth labels evenly spaced in the whole depth range.

Between the minimum and maximum depth label the fixed band module regresses an interpolation factor as output. Because of this the network cannot reason about the absolute scale of the scenes, which makes the network more flexible and generalize better. The fixed band contains a set of fronto-parallel planes as depth labels. Conversely the narrow band contains discrete labels which are individual for each pixel.The prediction of interpolation factors is not useful since the network in the narrow band module has no knowledge of the band's shape. The narrow band is not provided with the band shape, because the network tends to ignore the cost information in the cost. This makes the depth regularization difficult. Therefore there is another network appended, which focuses on this problem.

8.3.2 Training

In about 8 days in total the training of the mapping network can be acomplished on a NVIDIA GTX 1080Ti.

9 Workflow

9.1 Used Hardware[AV]

For video capturing a *Raspberry Pi 3B+* of the Raspberry Pi Foundation with a *Pi Camera V2* or a *Pi Wide Angle Lens Camera* are used. The Raspberry Pi sends the video feed to a separate more power full PC over WiFi using a Python3 script 9.4.2 since the scripts that are available use RTSP (Real Time Streaming Protocol) had a few problems.
For running the DeepTAM a *Lenovo Think-Station S20* is used since it requires a dedicated graphics card. Since the DeepTAM is a bit tricky to get it working a *Lenovo W550s* was used as a secondary machine to not loose any progress. For more intense work a server access at the Johannes Kepler University was supplied to work on their system which contains Nvidia Quadro and Nvidia GTX graphics cards.
As the work is based around implementing it on the Audi Autonomous Driving Cup (AADC) car a cheaper remote controlled model car was borrowed to test the algorithms in an live environment that is not based on a dataset. The AADC car was not used since it has very much power and is pretty expensive.

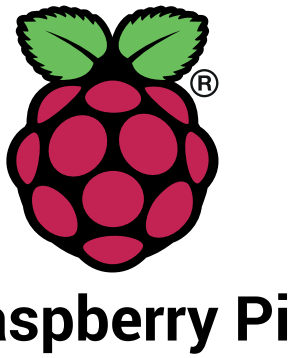


Figure 9.1: Raspberry Pi Logo™
Src: `https://tinyurl.com/njb3a7o`

9.1.1 Raspberry Pi

The Raspberry Pi™[1] is running Raspbian Buster since it is a well optimized version of Debian for the mini computer and only required to be able to execute a Python script to send the raw video feed over http to the the processing device.

[1] Raspberry Pi is a trademark

9.1.2 PC

The Think-Station and the laptop are running Kubuntu 18.04, which is basically Ubuntu but has a GUI that's a more like Windows and is supported until May 2023.
The Think-Station has a eight core Intel Xeon CPU, a GTX 1660TI and 12GB of RAM inside.
The Laptop has a four core Intel i7 and 8GB of RAM built in.

9.2 Used Software[AV]

ADTF
At first Ubuntu 16.04 with Automotive Data and Time-Triggered Framework (ADTF) was used since it's the recommended environment by the AADC car manufacturer DigitalWerk. There were many compatibility ans stability issues and it is very difficult to get into the whole system as it's not very beginner friendly. After trying to get the basics of ADTF working it was clear that switching to ROS might be better. The main problems with ADTF are, that ADTF isn't running very stable, requires certain packages to be in a non-standard folders and not having them in the regular location and it is very difficult to work with when using it for the first time.

ROS
Running ROS Melodic on Kubuntu 18.04 was pretty straight forward. The instructions on the ROS website [30] are very clear and can be directly copied without issues. The principle of the workspace is also easy to understand. In the source folder the modules get put in and when compiling the modules automatically generates a setup file to use them. Usage is very easy as the framework already does a lot in the background and using nodes is nearly always setting input and output with a few parameters.

Python[2]
Python[31] is a object-oriented, high-level programming language. *"Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together."* [32] Next to that the community features a wide range of already written packages which makes it even more simple to create programs.

Figure 9.2: Python Software Foundation logo™
Src: `https://tinyurl.com/tfxfbrt`

[2] "Python" and the Python logos are trademarks or registered trademarks of the Python Software Foundation

9.3 Setup[AV]

As the PC and laptop are not the best idea to run around with, a Raspberry Pi is used instead to stream the video over WiFi to the PC/laptop which is connected to the router over LAN. This makes the camera setup very portable as the pi, camera and powerbank are packed together and only weighs around 500g. The PC can be placed somewhere where and just processing the received video signal.

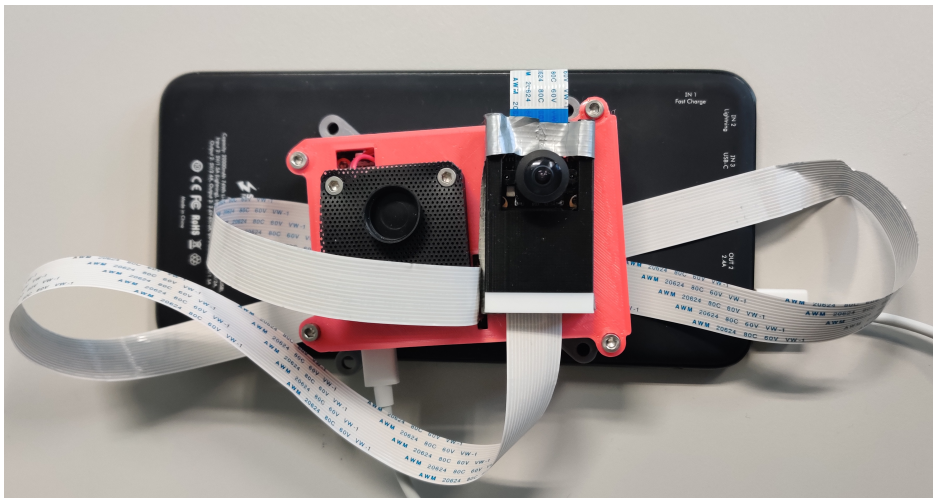


Figure 9.3: Stream setup with Raspberry Pi, camera and powerbank

9.4 Streaming video from Pi to PC[AV]

9.4.1 Enabling Camera

To use a camera on a Raspberry Pi the camera serial interface (CSI) needs to be enabled first. This can be done in the built-in tool called *raspi-config*. In this tool as seen in figure 9.4 under the subsection called *Interfacing Options* there is an option with the name *Camera*. When the interface is activated a restart is required.

9.4.2 Python Script

In listing 9.1 at first a *piCamera* instance with the name *cam* is created. As parameters the resolution gets set to *1280x720* pixels and the frame rate is set to *30* frames per second (FPS). If needed the image can be rotated, e.g the camera is mounted upside down. When starting the camera a output and format are expected. For the output a separate class is used which sets how and when a new frame can be published. The *mjpg* video codec is chosen as a pack for getting MPEG-streams already exists in ROS and it's not power hungry when running it on the Raspberry Pi as the hardware supports it [33].
After the camera "recording" has started successfully the server is started to make the stream

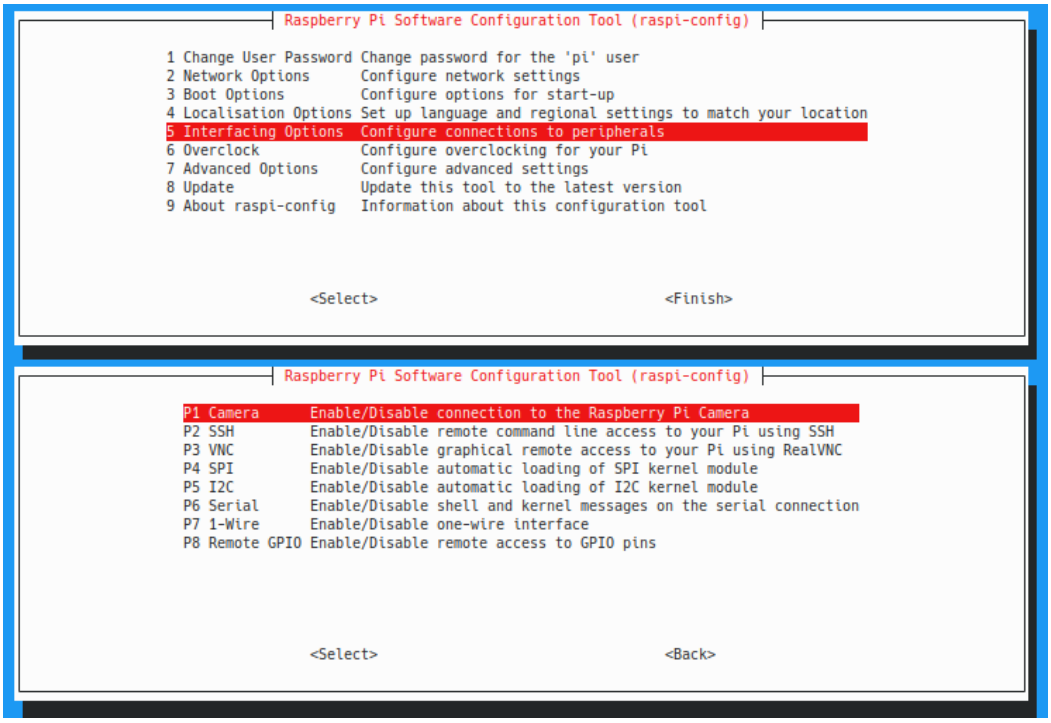


Figure 9.4: Activating camerainterface in raspi-config

accessible to other devices. The server runs until the user closes the script using *CTRL + C*. After closing the server the *finally* block gets called, where the camera "recording" is stopped so that other programs can use the camera again.

```
#Only resolution and framerate in Constructorparameters
with picamera.PiCamera(resolution=RESOLUTION, framerate=FPS) as cam:
    output = streamingOutput()
    # Rotation if needed
    cam.rotation = ROTATION
    #start stream
    cam.start_recording(output, format='mjpeg')

    try:
        #IP, Port
        hostAndPort = ('',PORT) # '' as IP automatically get's ip
        server = streamingServer(hostAndPort, streamingHandler)
        server.serve_forever() # Serves as long as script is running
    finally:
        cam.stop_recording() # Stops stream to make camera accessable from other
            apps again
```

Listing 9.1: Main Function of Camera Feed

The streamingHandler that is shown in listing 9.2 handles the actions that are taken when client connects to the Raspberry Pi. At the beginning it checks if the client is requesting the */stream.mjpg* file. If the client is not requesting that specific file a 404 Not Found Error is returned. But if the correct file is requested at first a 200 OK code is sent. In addition to the status code headers are sent, to not use cache. After sending the HTTP OK to the client a permanent loop is started which always waits until a new image from the camera

is ready and then sends it to the client as a JPEG image. The loop ensures that the client receives the latest image and such creates a video. Should the client drop the connection an exception is raised which causes the script to drop out of the loop and stops the handler for that specific client until the client connects again.

```
class streamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        # Check if user is requesting stream
        if self.path == '/stream.mjpg':
            # yes --> Send Stream
            # send header for beginning streaming
            self.send_response(200) # HTTP OK
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-replace;
                boundary=FRAME')
            self.end_headers()

            # start Stream
            try:
                while True:
                    # Send new image when available
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    # Header for sending image
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type', 'image/jpeg')
                    self.send_header('Content-Length', len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                print('Removed streaming client %s: %s', self.client_address,
                    str(e))
        else:
            # User requested something else --> Send 404 Page
            self.send_error(404)
            self.end_headers()
```

Listing 9.2: Streaming Handler of CamStream

The behavior when a new image from the pi camera is ready to send is defined in the snipped 9.3. At the beginning it initializes itself with basically no image. *piCamera* class constantly writes into this, as it's defined as the output. When a new image is ready the *picamera* class sends "\xff\xd8" as binary data to the output class to notify it. The output class then cuts the buffer so it only contains the current image and sets it in the *frame* variable. To let everybody else know that a new image is ready it sends out a notification.

```
class streamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

```

```
7     def write(self, buf):
8         if buf.startswith(b'\xff\xd8'):
9             # New frame available to get to buffer --> Notify all clients
10            self.buffer.truncate()
11            with self.condition:
12                self.frame = self.buffer.getvalue()
13                self.condition.notify_all()
14            self.buffer.seek(0)
15        return self.buffer.write(buf)
```

Listing 9.3: Streaming Output of CamStream

9.5 Receiving images on PC and Laptop[AV]

For receiving the stream on the PC or Laptop the existing Video-Stream-OpenCV Node is used. Video-Stream-OpenCV is designed to publish videos in the ROS network which are received from different sources, e.g. USB-cameras, video-files, network cameras and video-streams [34].

9.5.1 MJPG-Stream receiver

To automate the startup procedure of the node a roslaunch file is used. The lauchfile automatically starts the node and passes the required parameters.
The launchfile shown in 9.4 published the received image stream on a topic called *camera*. The video stream provider are the Raspberry Pi's IP-address, port and */stream.mjpg* directory. 30 FPS are used since they provide a good balance between amount of traffic and amount of detail in the movement.

```
1   <!-- launch video stream -->
2   <include file="$(find video_stream_opencv)/launch/camera.launch" >
3      <!-- node name and ros graph name -->
4      <arg name="camera_name" value="camera" />
5      <!-- url of the video stream -->
6      <arg name="video_stream_provider" value="http://192.168.2.109:5000/stream.
           mjpg" />
7      <!-- set camera fps to (probably does nothing on a mjpeg stream)-->
8      <arg name="set_camera_fps" value="30"/>
9        <!-- set buffer queue size of frame capturing to -->
10       <arg name="buffer_queue_size" value="100" />
11     <!-- throttling the querying of frames to -->
12     <arg name="fps" value="30" />
```

Listing 9.4: MJPG-Stream receiver Launch file

9.6 Cameras[AV]

Nearly every camera has some kind of distortion where the proportions of the image are different to the real world. This is especially noticeable on wide angle lenses which can capture a bigger part of the environment while sitting in the same spot. This can be seen in

figure 9.5 that the normal camera only captures a small portion compared to the wide angle lens camera but the wide angle lens creates distortions when getting to the edges.

Figure 9.5: **Left Image:** normal Raspberry Pi Camera. **Right Image:** wide angle lens camera

9.6.1 Calibration

To get rid of the distortions on a wide angle lens camera calibration is needed, which is a mask that gets applied on the image to remove these distortions and rectify it. For calibration the *camera calibration*-node is used. Calibration is done by moving and rotating a checkerboard, which is used since it has a good contrast between the tiles and the size of a tile is known and always the same. The node recognizes the checkerboard and calculates the distortion-factors from a series of pictures that have been taken.
The command for starting the node is the following, where amount of tiles, size of tiles in millimeter and camera are set:

```
rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.026 --no-
    service-check image:=/camera/image_raw camera:=/camera
```

Listing 9.5: Start Calibration Node

This command opens a window which shows the live image feed from the camera and highlights edges on the checkerboard which is shown in figure 9.6. These highlighted edges are points which are used to calculate the distortion parameters using an algorithm that was developed by OpenCV [35].

When the node has enough reference images the computing of the parameters can start. The duration of the calculations depends on the CPU and how many images have been taken. But most likely it will take around 5 Minutes until the computing is finished. The output data contains two formats of the same data which will look like something like shown in figure 9.6. The output files are normally compatible with all applications without any issues.

```
image_width: 1280
```

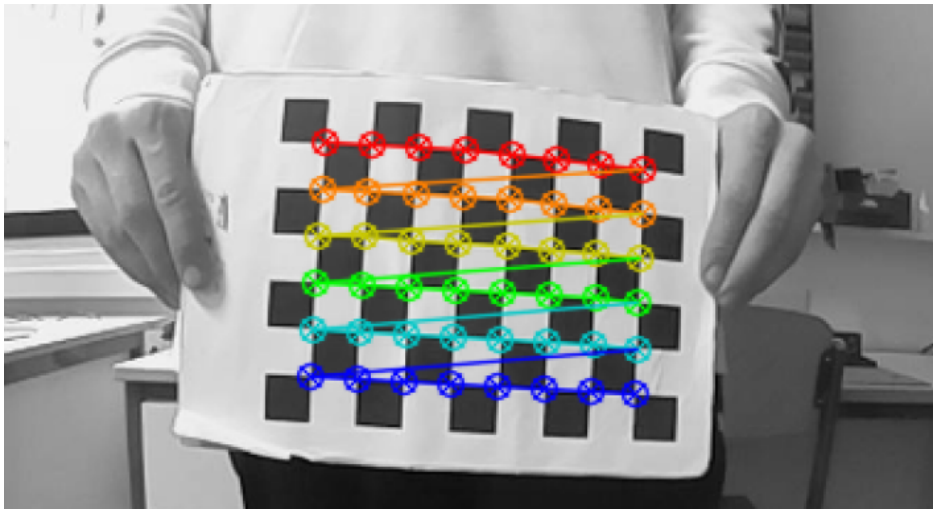


Figure 9.6: Camera calibration

```
image_height: 720
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [ 600.43227,    0.     ,  650.29343,
            0.     ,  605.883  ,  253.24984,
            0.     ,    0.     ,    1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.253310, 0.043398, 0.005101, -0.003291, 0.000000]
```

Listing 9.6: Calibration file

9.7 Things to keep in mind

These points are helpful to achieve good results.

- Initialization
  Avoid movement and too much rotation when the system is initializing.
- Rotation
  Rotation only will not generate any depth information and might even generate wrong points.
- Low texture environment
  Since the ORB-SLAM needs prominent points it won't work properly when there are none or very little points that the algorithm can lock on, e.g. a clear white hallway.
- Many/Big moving objects
  Many or big moving objects generate much data that is not stationary as the algorithm "thinks" that the vehicle itself is moving.

9.8 Running Monocular ORB-SLAM2[AV]

Integrating ORB-SLAM2 6 is pretty straight forward since there is already a ROS version of it which is very well maintained by **Lennart Haller** on behav of *appliedAI-Initiative* on GitHub [36]. It just needs to be cloned into the *src* folder of the ROS-workspace and compiled using the *catkin build* command.When building the packs is finished the generated setup-file has to be sourced. The file is located under *workspace/devel/setup.sh* and can be sourced using the *source setup.sh* command. The SOURCE-command is often used to set up variables or commands that are not loaded in the terminal by default. After sourcing the setup all new types of commands that are added by the nodes are available.

9.8.1 Calibration file

Before using the ORB-SLAM the calibration file needs to be created in *src/orb_slam2/config/config.yaml*. For a baseline the config from another mono-camera can be copied. The calibration part looks like shown in 9.9 and thus the config that the calibration tool generated needs to be converted manually. In the camera_matrix the values are the following:

```
1       [camera.fx, 0, camera.cx, 0, camera.fy, camera.cy, 0, 0, 1]
```

Listing 9.7: Matrix listing

The distortion_coefficients need to be mapped like shown in listing 9.8.

```
1       [camera.k1, camera.k2, camera.p1, camera.p2, 0]
```

Listing 9.8: Matrix listing

Additionally the width, height and FPS of the incoming video have to be set to use the correct amount of frames and resolution.

```
1  # Camera calibration and distortion parameters (OpenCV)
2  Camera.fx: 600.43227
3  Camera.fy: 605.883
4  Camera.cx: 560.29343
5  Camera.cy: 253.24984
6
7  # Camera distortion parameters (OpenCV)
8  Camera.k1: -0.253310
9  Camera.k2: 0.043398
10 Camera.p1: 0.005101
11 Camera.p2: -0.00329
12 Camera.k3: 0.0
13
14 Camera.width: 1280
15 Camera.height: 720
```

Listing 9.9: ORB-SLAM2 config calibration

9.8.2 Launching ORB-SLAM2

To load the new config file a new launch file is the best way to tell the SLAM to use this config instead of the default parameters. The launch files for ORB-SLAM2 sit in *src/orb_slam2/ros/launch* where an existing one can be duplicated. The new launch-file should look like the snipped 9.10 except in line **13** the wrong config is specified. Additionally in the config other settings can be set. For example if the node should publish a pointcloud or position, only do tracking. It is also possible to load an existing map which has been created before and might be used to add additional information or do tracking only on it. When everything is configured it is now possible to launch the ORB-SLAM2 using *roslaunch orb_slam2_ros orbslam.launch*.

```
1  <launch>
2    <node name="orb_slam2_mono" pkg="orb_slam2_ros"
3        type="orb_slam2_ros_mono" output="screen">
4
5         <param name="publish_pointcloud" type="bool" value="true" />
6         <param name="publish_pose" type="bool" value="true" />
7         <param name="localize_only" type="bool" value="false" />
8         <param name="reset_map" type="bool" value="false" />
9
10        <!-- static parameters -->
11        <param name="load_map" type="bool" value="false" />
12        <param name="map_file" type="string" value="map.bin" />
13        <param name="settings_file" type="string" value="$(find orb_slam2_ros)/
              orb_slam2/config/config.yaml" />
14        <param name="voc_file" type="string" value="$(find orb_slam2_ros)/
              orb_slam2/Vocabulary/ORBvoc.txt" />
15
16        <param name="pointcloud_frame_id" type="string" value="map" />
17        <param name="camera_frame_id" type="string" value="camera_link" />
18        <param name="min_num_kf_in_map" type="int" value="5" />
19   </node>
20 </launch>
```

Listing 9.10: ORB-SLAM2 launch file

When starting RQt 2.4.2 and opening the Node Graph it should show, like seen in figure 9.7 that the camera-node is sending *image_raw* to the *orb_slam2_mono*-node. If that is not the case it is most likely due to some spelling error or a *remap* line in the launch-file of the ORB-SLAM2.

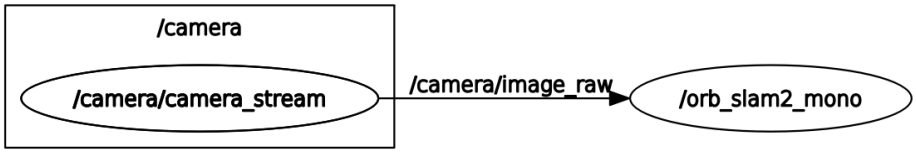


Figure 9.7: Node Graph with Stream and ORB-SLAM

9.9 Running Stereo ORB-SLAM2[AV]

As explained in the ORB-SLAM chapter 6 there are some problems with running it with a single-camera setup. The main problem is that on a monocular image the scale information is not provided an thus drifting occurs. A way to bypass this problem is by using two cameras looking in the same direction but with a slight offset. The setup with two cameras then works a bit more like eyes where the depth perception is easier when having both eyes open compared to only having one open.

9.9.1 Hardware setup

Since stereo cameras and synchronization boards are not cheap, two identical Raspberry Pi's with the same camera are used for the testing environment. The two Raspberry Pi's have the same SD card type, operating system and packages to prevent any timing difference as much as possible. There are solutions that use a special board with a circuit for synchronizing the cameras or even boards that already have two cameras mounted. The problem is that they are pretty expensive. The Cameras are taped on a stick of wood with a distance of around six centimeters to mimic the average distance between the human eyes.

9.9.2 Calibration

Calibration in this testing environment is pretty straight forward as the cameras used are the normal Pi Cameras v2 which basically have nearly no distortion. Just for being on the safe side calibration was done on both cameras and the values are only slightly different. So the format that was entered in the config was the same as on the monocular SLAM calibration 9.8.1. If cameras with distortion are used there are more values that are required a bit further down in the config.

9.9.3 Software setup

The same Python script as already explained in section 9.4.2 is used on both devices to make the camera feed available on the local network. To receive these streams two MJPG-Stream receivers are required. Note that the camera name must be unique, e.g they may be named LCAMERA 9.11 and RCAMERA 9.12 to know which is the left and right camera. Also required is to set the stream receiver to the correct IP-address since it is important later. Re-building isn't necessary since only the config and launchfile change, which are not compiled but instead loaded every time the SLAM starts.

```
1       <arg name="camera_name" value="lcamera" />
```

Listing 9.11: Left camera stream launch file

```
1       <arg name="camera_name" value="rcamera" />
```

Listing 9.12: Right camera stream launch file

Since the two video streams are still on a topic that the SLAM doesn't subscribe on it is necessary to remap the two streams to the correct topics. This can be done in a launchfile for the stereo ORB-SLAM. To do this it is only required to add two lines which are shown in listing 9.13 at line **5** and **6** before all other settings for the SLAM get set. This remaps the nodes that triesw to subscribe to e.g */image_right/image_color_rect* to the correct topic that is publishing the image named */rcamera/image_raw*

```
1  <launch>
2    <node name="orb_slam2_stereo" pkg="orb_slam2_ros"
3        type="orb_slam2_ros_stereo" output="screen">
4
5         <remap from="image_left/image_color_rect" to="/lcamera/image_raw" />
6         <remap from="image_right/image_color_rect" to="/rcamera/image_raw" />
7
8         <param name="publish_pointcloud" type="bool" value="true" />
9         <param name="publish_pose" type="bool" value="true" />
10        <param name="localize_only" type="bool" value="false" />
11        <param name="reset_map" type="bool" value="false" />
12
13        <!-- static parameters -->
14        <param name="load_map" type="bool" value="false" />
15        <param name="map_file" type="string" value="map.bin" />
16        <param name="settings_file" type="string" value="$(find orb_slam2_ros)/
              orb_slam2/config/config.yaml1l" />
17        <param name="voc_file" type="string" value="$(find orb_slam2_ros)/
              orb_slam2/Vocabulary/ORBvoc.txt" />
18
19        <param name="pointcloud_frame_id" type="string" value="map" />
20        <param name="camera_frame_id" type="string" value="camera_link" />
21        <param name="min_num_kf_in_map" type="int" value="5" />
22   </node>
23 </launch>
```

Listing 9.13: ORB-SLAM2 stereo launch file

9.9.4 Launching

When starting the stereoscopic launch for the SLAM using the command 9.9.4 ,it should indicate in RQt 2.4.2 that the ORB-SLAM subscribed onto both video streams like shown in figure 9.8. If only one is connected it might be caused by a misspelled word in the remap line or when naming the cameras in the stream launch file. Additionally when opening the *Image View* and selecting /ORB_SLAM2_STEREO/IMAGE_IMAGE as the source the image of the left camera should be displayed.

```
1     roslaunch orb_slam2_ros stereo-slam.launch
```

Note that the SLAM will freak out when the cameras are not "looking" at the same picture or are at different angles.

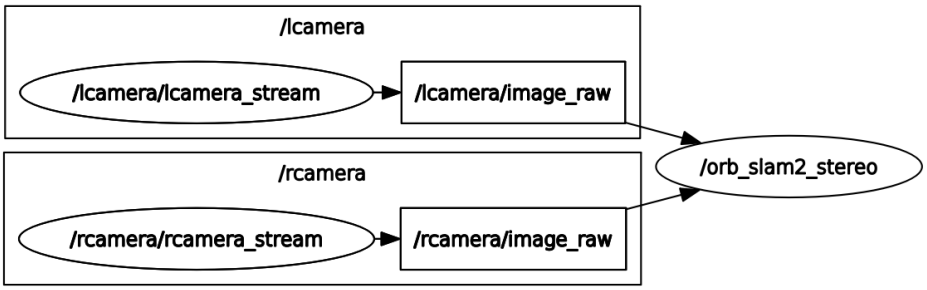


Figure 9.8: Node Graph with Stereo ORB-SLAM2

9.10 Result with ORB-SLAM

The result when running the ORB-SLAM is a 3D map that consist of many dots that are at specific points in a space. To get the map out of the ROS node it is necessary to call a service 2.2.3. Depending on the used camera type the command looks like one shown in listing 9.14.

```
rosservice call /orb_slam2_rgbd/save_map map.bin
rosservice call /orb_slam2_stereo/save_map map.bin
rosservice call /orb_slam2_mono/save_map map.bin
```

Listing 9.14: Saving map from ORB-SLAM

When viewing the map in a programm like RViz 2.4.4 it should look similarly to figure 9.9 .

Figure 9.9: ORB-SLAM Top Down
Src: `https://tinyurl.com/swoouaa`

9.11 Running LSD-SLAM[AV]

Running the LSD-SLAM is not as straight forward as it is on the ORB-SLAM 9.8. The implementation of the LSD-SLAM is made to use it with ROS out of the box but the original version [37] by the Computer Vision Group at the Technical University of Munich was not updated in the past 5 years and such had some problems when running it on a newer version of ROS and Ubuntu. Therefor an more up-to-date version by Kevin George was used where he added support for newer versions of libraries and fixed some bugs. Even though the fork by Kevin George [38] was updated 2 Years ago there were still some bugs and other problems. So a fork was created with the fixes to address the problems that where in Kevin George's Repository. The fork includes bug fixes to run LSD-SLAM on ROS-Melodic and Ubuntu 18.04 pretty much out of the box.
The new fork [39] contains implementation to use EIGEN v3.2.5 that doesn't need to be installed, a special version of General Graph Optimization (G2O) and a CSPARSE-directory path fix.

9.11.1 Errors when building from source original source

This list shows typical errors when building the LSD-SLAM from source and how to fix or at least bypass them.

Cannot bind non_const lvalue to rvalue

When receiving an error like shown in listing 9.11.1 this is most likely due to a datatype error where the functions expects a DOUBLE but a FLOAT is defined. to fix this, it is only necessary to change *float x,y,z;* to *double x,y,z;* in the two files in the error.

```
1 PointCloudViewer.h:136:26: error: cannot bind non-const lvalue reference of type
      'qreal& {aka double&}' to an rvalue of type 'qreal {aka double}'
2    frame.getPosition(x,y,z);
3
4 PointCloudViewer.cpp:327:44: error: cannot bind non-const lvalue reference of
     type 'qreal& {aka double&}' to an rvalue of type 'qreal {aka double}'
5         camera()->frame()->getPosition(x,y,z);
```

no such function found: g2o::OptimizationAlgorithmLevenberg and *base_vertex.h not found*

This error occurs due to a change in newer G2O versions. To fix this, it is required to install the *g2o c++03* version. The fix itself for this error can be found at G2O 9.11.2.

/usr/bin/ld: cannot find -lcsparse

The error shown in 9.11.1 most likely accuress when catkin is looking for files that are in a different folder that expected by it.

```
[ 91%] Linking CXX shared library /home/alex/lsd-slam_ws/devel/lib/liblsdslam.so
/usr/bin/ld: cannot find -lcsparse
collect2: error: ld returned 1 exit status
```

To fix it the paths where the required files are, have to be included in the *CMakeList.txt* file which can be found in the source folder. Add the block shown in **??** at the top of the file and replace CSPARSE and CXSPARSE with ${CSPARSE_INCLUDE_DIR} at the line where *target_link_libraries* is adding the links to libraries.

```
find_path(CSPARSE_INCLUDE_DIR NAMES cs.h
  PATHS
  /usr/include/suitesparse
  /usr/include
  /opt/local/include
  /usr/local/include
  /sw/include
  /usr/include/ufsparse
  /opt/local/include/ufsparse
  /usr/local/include/ufsparse
  /sw/include/ufsparse
  PATH_SUFFIXES
  suitesparse
  )
```

liblsdslam.so: Warning: undefined reference »..«

This error is a batch of errors that occur at the same time. It is mainly that references cannot be found by the system. The error might look something like shown below and is very easy to fix. The problem is that the *libsuitesparse-dev* package is missing. When installing the package and then compiling G2O it should work again.

```
.../liblsdslam.so: Warning: undefined reference cs_di_post
.../liblsdslam.so: Warning: undefined reference cs_di_etree
.../liblsdslam.so: Warning: undefined reference cs_di_sfree
```

double free or corruption

```
double free or corruption (out)
Aborted (core dumped)
```

The error shown in listing is due to changes in newer versions of Eigen. To fix this the easiest way is to download the version 3.2.5 from the Eigen Bitbucket website [40] and uncompress it somewhere where it can stay during the build process. In the *CMakeList.txt* of the *lsd_slam_core* add the code like shown in listing 9.15 and replace PATH with the just unpacked folder path.

```
set(EIGEN3_INCLUDE_DIR "PATH")
include_directories(
```

```
3   include
4   ${EIGEN3_INCLUDE_DIR}
```

Listing 9.15: CMakeList in lsd_slam_core with fix for Eigen

g2oTypeSim3Sophus.h Errors

Another error that likes to come up multiple times in the same file. These errors will look something like show below but in a much bigger scale.

```
1 lsd_slam/lsd_slam_core/src/GlobalMapping/g2oTypeSim3Sophus.h:96:33: error:
      template argument 1 is invalid
2    Eigen::Map<const g2o::Vector7d> v(m);
```

This error occurs because of some changed in EIGEN and is a relativly easy fix. To fix it it is only necessary to change the code show below from *line 1* to *line 2* in the *g2oTypeSim3Sophus.h* [41].

```
1 Eigen::Map<const g2o::Vector7d> v(m);
2 Eigen::Map<const Eigen::Matrix<double, 7 ,1>> v(m);
```

Listing 9.16: Fix for g2oTypeSim3Sophus.h

Couldn not find package configuration file provided by Eigen

When receiving a error message that states something similiar to the message below it is most likely because of a naming difference in the *CMakeList.txt*.

```
1 lsd_slam/lsd_slam_viewer/CMakeLists.txt:30 (find_package):
2   By not providing "FindEigen.cmake" in CMAKE_MODULE_PATH this project has
3   asked CMake to find a package configuration file provided by "Eigen", but
4   CMake did not find one.
```

The fix is to rename *Eigen* to *Eigen3* in the *CMakeList.txt* in the stated path.

g2o No matching function for call

When getting this error it is because of a non-compatible version of G2O. To fix this it is required to uninstall all other versions of G2O and then install a working version like explained at 9.11.2.

ros/ros.h: No such file or directory

If an error message like shown occurs it is because the catkin directories are not included by default.

```
1 lsd_slam/lsd_slam_core/src/IOWrapper/ROS/ROSImageStreamThread.h:27:10: fatal
      error: ros/ros.h: No such file or directory
```

To fix it it is required to add *$catkin_INCLUDE_DIRS* to the *include_directories* in the CMakeList-File [42]. It should look like the listing below afterwards.

```
include_directories(
  include
  ${EIGEN3_INCLUDE_DIR}
  ${PROJECT_SOURCE_DIR}/src
  ${PROJECT_SOURCE_DIR}/thirdparty/Sophus
  ${CSPARSE_INCLUDE_DIR} #Has been set by SuiteParse
  ${CHOLMOD_INCLUDE_DIR} #Has been set by SuiteParse
  ${OpenCV_INCLUDE_DIRS}
  ${catkin_INCLUDE_DIRS}
)
```

Listing 9.17: Fix for ros/ros.h: No such file or directory

CV_GRAY2RGB & CV_RGB2GRAY not declared

If the following error comes up it is due to a change in the naming scheme.

```
error: 'CV_GRAY2RGB' was not declared in this scope
   cv::cvtColor(keyFrameImage8u, res, CV_GRAY2RGB);
```

The fix is just to replace *cv_* with *vc::COLOR_* and should look like e.g.

```
cv::COLOR_GRAY2RGB
```

Listing 9.18: Example fix for ĆV_GRAY2RGB& ĆV_RGB2GRAYńot declared

9.11.2 Installing LSD-SLAM from fixed repository

To run the LSD-SLAM a few packages are required. They can be installed using

```
    apt install ros-melodic-cv-bridge liblapack-dev libblas-dev freeglut3-dev
        libqglviewer-dev-qt4 libsuitesparse-dev libx11-dev
```

Listing 9.19: Installing prerequisites for LSD-SLAM

and then creating a link for *libQGLViewer* to the file the LSD-SLAM is looking for. To create the softlink this command is used wich basically creates a link just without the -QT4 at the end.

```
    sudo ln -s /usr/lib/x86\_64-linux-gnu/libQGLViewer-qt4.so /usr/lib/x86\_64-
        linux-gnu/libQGLViewer.so
```

Listing 9.20: Creating softlink for libQGLViewer.so

G2O

As the LSD-SLAM requires a special version of the G2O framework all other versions that have been installed on ROS need to be removed completely. For this first the *ros-melodic-g2o* package needs to be purged and the leftovers need to be deleted to not conflict with the special version that is going to be installed later using.

```
rm -r /usr/local/lib/libg2o* /usr/local/include/g2o /usr/local/lib/g2o /usr/
    local/bin/g2o*
```

Listing 9.21: Removing leftover from G2O

To install the correct version of G2O it needs to be cloned from GitHub [43] which creates a g2o folder in which a *build*-folder has to be created. In the *build* folder two commands have to be executed in order to install the framework.

```
cmake ..
sudo make install
```

Eigen

When done it is also necessary to download a specific version of the Eigen-library which has to be in a location where it is not likely to be moved as the full path is required. The Path should look a bit like this */home/alex/Projects/LSD-SLAM/eigen-eigen-bdd17ee3b1b3/*. This path needs to be set in the *CMakeList.txt* in the *lsd_slam_core* folder which was contained in the LSD-SLAM repository. In the file at line **106** there should be a placeholder like shown in 9.22, which has to be replaced with the path of Eigen.

```
set(EIGEN3_INCLUDE_DIR "{PATH TO EIGEN THAT YOU JUT DOWNLOADED}")
```

Listing 9.22: CMakeList Eigen Directory

Building LSD-SLAM

When these steps are done there is only one last command to be executed in the root directory of the workspace. To compile the LSD-SLAM in the *src* folder the command is

```
catkin_make
```

which may take some time.

9.11.3 Things to keep in mind

These points are helpful to get good results and some to keep in mind when running the SLAM. [37].

- Resolution
  A recommended resolution for the video input is 640x480 pixel. Higher resolutions are possible but require more computing power and changes in hard-coded values.
- Framerate
  At least 30 FPS or more should be delivered to the SLAM to get good results when moving.
- Global shutter
  The camera should have a global shutter where the the image is taken all at once. Rolling shutters which scan the image from top to bottom [**?**]. This might lead to changes while the scan is running and thus generating false information. This wrong information then leads to inferior results.
- Field of View
  A camera lens that has a wide field of view (FOV) is recommended to get a bigger area that the camera can capture and thus is able to fetch more information. A lens with 130° or more is recommended.
- Camera Movement
  It is a very important aspect. Typically the LSD-SLAM is designed to move more sideways but when having a wide FOV it is also possible to have forwards and backwards motion. Rotating the camera without moving it will not work since there is not enough camera translation.
- Different results when running multiple times
  Since the LSD-SLAM is designed to be non-deterministic the results will not be the same all the time. E.g when running a dataset the map will most likely be similar but not exactly the same. This is due to parallelisation and some slight changes in keyframe which have a big impact on everything that follows in the processing order and thus creates different results.

9.12 Launching

When running the LSD-SLAM it is necessary to specify what the sources are. The command should look similar to command 9.12 and for debugging purpose it is also possible to show the debug image that the SLAM is working on using the command. 9.12.

```
1    rosrun lsd_slam_core live_slam image:=/camera/image_raw camera_info:=/camera
         /camera_info
```

```
1    rosrun lsd_slam_viewer viewer
```

When looking in RQt 2.4.2 it shows that the LSD-SLAM has connected to the camera and the viewer connected to the LSD-SLAM. The node graph most likely looks like shown in figure 9.10.

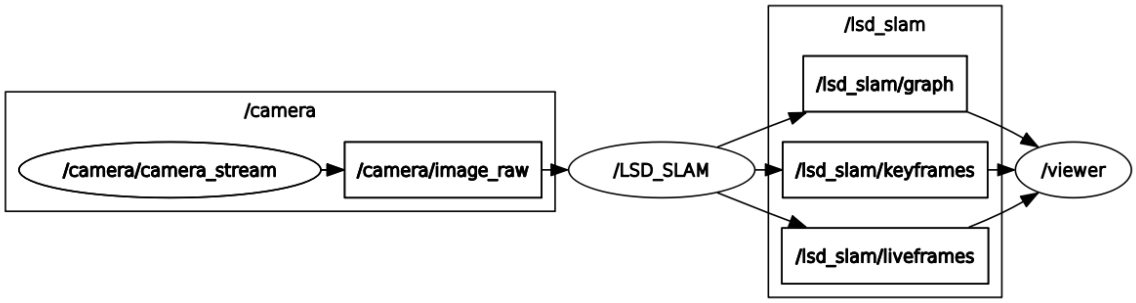


Figure 9.10: Node Graph with Stream and LSD-SLAM

9.13 Results with LSD-SLAM

The result when running the LSD-SLAM is a 3D map which contains many dots in space which represent a edges, surfaces and other details. Compared to the ORB-SLAM results 9.10 the LSD-SLAM produces a more detailed map in terms of amount of points and how they are laid out. When looking into a visualization programm like RViz 2.4.4 it the map should look similar to 9.11 which shows the more detailed map due to more information gathered by the SLAM.

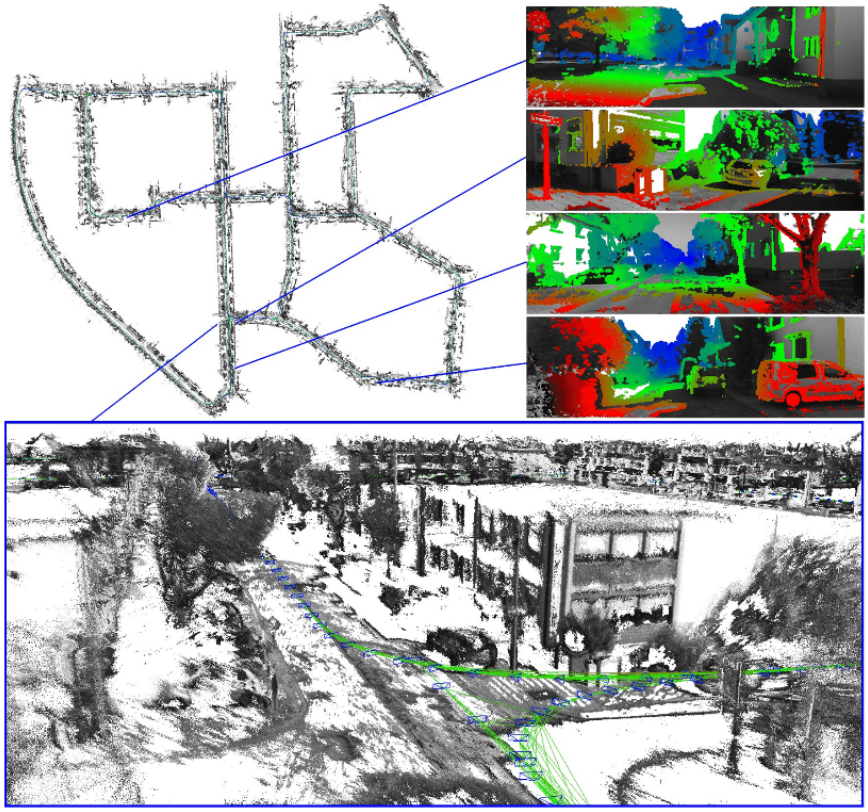


Figure 9.11: LSD-SLAM Topdown and Detailed View
Source: https://tinyurl.com/uepquna

9.14 DeepTAM[SM]

In the following section the process of getting DeepTAM to run will be explained. Also the Problems of our workflow will be listed.

9.14.1 Errors when building from source original source

The first attempt was to work with Ubuntu 16.04 and Python3, following the instructions given by Uni-Freiburg. At first a virtual environment manager needed to be installed with pip3(a package manager for Python3). Here pew is used and installed. Afterwards a new environment is created and entered. These are the commands:

```
1    pip3 install pew
2    pew new deeptam
3    pew in deeptam
```

The next step was to install tensorflow(TF), minieigen and scikit-image, also with pip3. Here the first error occurred. The error is that during the installation of minieigen Eigen/Core cannot be found. This problem can be worked around with the following commands.

```
1    sudo apt-get install libeigen3-dev
2    sudo apt-get install python3-minieigen
3    sudo apt-get install libboost-all-dev
```

After these commands minieigen can be installed with pip3. The next step is to clone and build lmbspecialops. lmbspecialops is a collection of tensorflow ops. The ops focus on networks for predicting depth and camera motion but can also be useful for other tasks.

9.14.2 Errors when building with Script

10 Fazit und Persönliche Erfahrungen

10.1 Fazit

Zusammenfassung der Projektergebnisse. Besondere Erkenntnisse. Beurteilung des Lösungswegs. Eventuelle Alternativen und möglicher Erweiterungen.

10.2 Persönliche Erfahrungen

Hier (und nur hier) darf aus der Ich-Perspektive geschrieben werden.

10.3 Ausblick

Glossary

AADC Audi Autonomous Driving Cup. 13, 24, 25

ADTF Automotive Data and Time-Triggered Framework. 25

ANN Artificial Neural Network. xi

CSI Camera Serial Interface. 26

FOV Field of View. 42

FPS Frames per Second. 13, 26, 29, 32, 42

G2O General Graph Optimization. 37–39, 41, 51

GPS Global Positioning System. x, 1

Lidar Laser radar. 1

OSRF Open Source Robotics Foundation. 3

RGB-D Red Green Blue - Depth. 2, 15, 16

ROS Robot Operating System. iv, x, xi, 3–5, 7, 14, 26, 29, 32, 36, 37, 41, 50

RTSP Real-Time Streaming Protocol. 24

SLAM Simultaneous Localization and Mapping. iv, x, xi, 1, 2, 15, 18, 31, 33–36, 41–43

TF TensorFlow. 44

Bibliography

[1] S. Riisgaard and M. R. Blas, "Slam for Dummies." https://tinyurl.com/y32jtecm.

[2] S. Prabhu, "Introduction to slam (simultaneous localisation and mapping)." ARreverie, 2019. https://tinyurl.com/y5an9jq9.

[3] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): part ii," *IEEE Robotics Automation Magazine*, vol. 13, pp. 108–117, Sep. 2006.

[4] S. Martin, "What is simultaneous localization and mapping?." Techapeek, 2019. https://tinyurl.com/y5yajv5u.

[5] V. O. M. Team, "What is visual slam technology and what is it used for?," 2018. https://www.visiononline.org/blog-article.cfm/What-is-Visual-SLAM-Technology-and-What-is-it-Used-For/99.

[6] C. Stachniss, U. Frese, and G. Grisetti, "OpenSLAM Website." Openslam, 2019. https://openslam-org.github.io/.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598, AAAI, 2002.

[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct 2015.

[9] F. Steinbruecker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*, 2011.

[10] F. Endrees, J. Hess, and N. Engelhard, "Rgb-d slam." ROS Wiki, 2019. https://tinyurl.com/yynyqwg2.

[11] J. Engel, T. Schops, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision (ECCV)*, September 2014. https://vision.in.tum.de/research/vslam/lsdslam?redirect=1.

[12] O. Foundation, "Osr foundation homepage." Website, 2019. https://www.osrfoundation.org/.

[13] "Ros industrial consortium europe," 2020. https://rosindustrial.org/ric-eu.

[14] ROS, "About ros." ROS, 2019. https://www.ros.org/about-ros/.

[15] C. Robotics, "Ros 101: Intro to the robot operating system." Robohub, 2014. https://tinyurl.com/y4q6paak.

[16] ROS, "Is ros for me?," 2020. https://www.ros.org/is-ros-for-me/.

[17] M. H. Hassoun, *Fundamentals of Artifical Neural Networks*. The MIT Press, 1995.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[19] Basler-AG, "daa1280-54uc (s-mount) - basler dart." https://www.baslerweb.com/de/produkte/kameras/flaechenkameras/dart/daa1280-54uc-s-mount/.

[20] Basler, "daa1280-54uc (s-mount) - basler dart," 2020. https://www.baslerweb.com/de/produkte/kameras/flaechenkameras/dart/daa1280-54uc-s-mount/.

[21] Basler, "Pylonsdk download page." https://www.baslerweb.com/de/vertrieb-support/downloads/downloads-software/.

[22] B. AG, "Pylon ros camera," 2020. https://github.com/basler/pylon-ros-camera.

[23] M. GmbH, "Pylon camera," 2019. https://wiki.ros.org/pylon_camera.

[24] W. Siebert, "Dragandbot common," 2019. https://github.com/dragandbot/dragandbot_common.

[25] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, Oct 2017.

[26] E. Weisstein, "Levenberg-marquardt method," 2020. http://mathworuld.wolfram.com/Levenberg-MarquardtMethod.html.

[27] E. Olson, J. Leonard, and S. Teller, "Fast iterative optimization of pose graphs with poor initial estimates," pp. 2262–2269, 2006.

[28] H. Zhou, B. Ummenhofer, and T.Brox, "Deeptam: Deep tracking and mapping with convolutional neural networks," *International Journal of Computer Vision*, 2019.

[29] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 International Conference on Computer Vision*, pp. 2320–2327, Nov 2011.

[30] ROS, "Ros melodic installation instructions." https://wiki.ros.org/melodic/Installation.

[31] P. S. Foundation, "Python homepage." https://www.python.org/.

[32] "What is python? executive summary." https://www.python.org/doc/essays/blurb/.

[33] R. P. Foundation, "Raspberry pi 3b+ product brief." https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+3+Model+B+plus+Product+Brief+PRINT&DIGITAL.pdf.

[34] R. D. Group, "video stream opencv," 2020. https://github.com/ros-drivers/video_stream_opencv.

[35] O. D. Team, "Camera calibration with opencv," 2020. https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[36] L. Haller, "Orb-slam 2 ros," 2019. https://github.com/appliedAI-Initiative/orb_slam_2_ros.

[37] T. U. M. Vision, "Lsd-slam," 2015. https://github.com/tum-vision/lsd_slam.

[38] K. George, "Lsd-slam fork," 2018. https://github.com/kevin-george/lsd_slam.

[39] A. Voglsperger, "Lsd-slam fork," 2019. https://github.com/MrMinemeet/lsd_slam.

[40] Eigen, "Eigen releases list," 2015. https://bitbucket.org/eigen/eigen/downloads/?tab=tags.

[41] T. Han, "tum-vision/lsd_slam - issue #313," 2018. https://github.com/tum-vision/lsd_slam/issues/313.

[42] F. Facundo, "fatal error: ros/ros.h: No such file or directory," 2018. https://answers.ros.org/question/283665/fatal-error-rosrosh-no-such-file-or-directory/?answer=283669#post-id-283669.

[43] R. K. Felix Endres, "G2o fork," 2016. https://github.com/felixendres/g2o/tree/c++03.

List of Figures

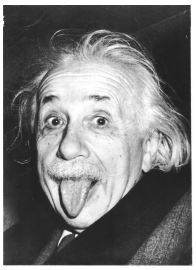
Listings

Authors

Alexander Voglsperger

Birthday, Place of birth:	25.03.2001, Ried im Innkreis
School education:	Volksschule Aurolzmünster Informatik Hauptschule Aurolzmünster HTL Braunau
Internship:	Team7 Natürlich Wohnen GmbH, 4 Weeks, IT Krankenhaus Ried im Innkreis, 4 Weeks, IT Johannes Kepler University - AI Lab, 4 Weeks, Mapping and Tracking on self-driving car
Address:	Forchtenau 196 4971, Aurolzmünster Österreich
E-Mail:	alexander.voglsperger@gmail.com

Simon Moharitsch



Birthday, Place of birth:	01.01.1970, Braunau am Inn
School education:	Volksschule Hauptschule HTL
Internship:	Firmenname, Zeit, Tätigkeit
Address:	Strasse Nummer PLZ, Ort Österreich
E-Mail:	max@mustermann.com