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

February 25, 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 Schoolyear 5AHELS 2019/2020

Topic of the diploma documentation Autonomous Car Mapping and Tracking

Cooperation-partner Johannes Kepler University - Artificial Intelligence Lab

Taskdefinition A camera delivers a sequence of 2D pictures of the environment in front of a car. Only sing these pictures the programm should generate a 3D map. Since the pictures don't contain any depth information a SLAM (Simultaneous Localization and Mapping) should be applied.

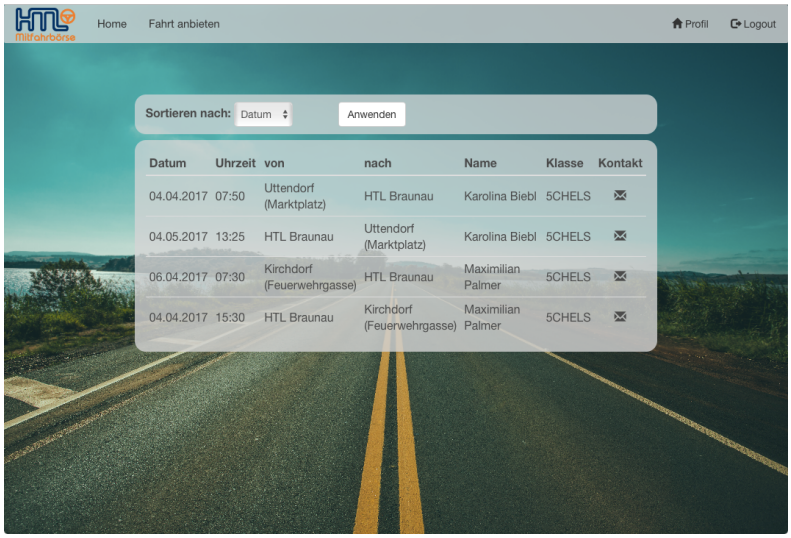
Realization As a foundation ROS was used because it is freely available and has an active community supporting the project. The ORB SLAM and LSD SLAM have already been implemented in ROS as nodes and can be used with changing a few things to get it working. DeepTAM is fairly new and hasn't been implemented into ROS yet.

Outcome Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

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

Illustrative graph, photo (incl. explanation)

Landing page of HTL-carpooling:



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, 25.02.2020	Alexander Voglsperger	_____
Location, Date	*Author*	*Signature*

Braunau/Inn, 25.02.2020	Simon Moharitsch	_____
Location, Date	*Author*	*Signature*

Contents

Abstract

Im Vorwort teilt der Bearbeiter dem Leser wichtige Tatsachen mit, die Erklärungen zu seiner Arbeit beinhalten – z.B. die Motivation für die Bearbeitung des Themas oder besondere Schwierigkeiten bei der Bearbeitung und/oder Materialbeschaffung.

Hier können auch Mitteilungen persönlicher Natur enthalten sein – z.B. Dank an Institutionen/Personen für die geleistete Unterstützung.

Summary

Die *Zusammenfassung* oder auch *Kurzfassung* soll den Inhalt der Diplomarbeit auf maximal einer halben Seite zusammenfassen.

Dieses Dokument dient als Vorlage und Beschreibung für die Dokumentation der Diplomarbeit. Es werden Hinweise zur Erstellung einer guten Dokumentation gegeben. Dies betrifft welchen Inhalt die Arbeit haben soll genauso wie welche Regeln eingehalten werden müssen und mit welchen technischen Mitteln das Dokument erstellt werden kann.

Beim Inhalt dieser Arbeit wurden alle grundlegenden Qualitätsregeln eingehalten und kann daher als Musterlösung gesehen werden. Zum Erstellen wurde das Textsatzsystem LaTeXverwendet. Es ist vorgesehen, dass der LaTeXQuelltext dieses Dokuments als Ausgangspunkt für die eigene Dokumentation verwendet wird.

Dieses Dokument sollte unbedingt aufmerksam gelesen werden ehe mit der eigenen Arbeit begonnen wird.

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 don't have good enough maps that are up-to-date , if the exist at all or it might be in an environment where positioning for instance GPS can't be used properly because the environment faces frequent changes, e.g parked cars or passengers [4]. If SLAMs weren't 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 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 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, 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 manage hardware abstraction, low-level device control, message-passing between processes and package management. *"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]

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 2.1. 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].

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 it's 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.

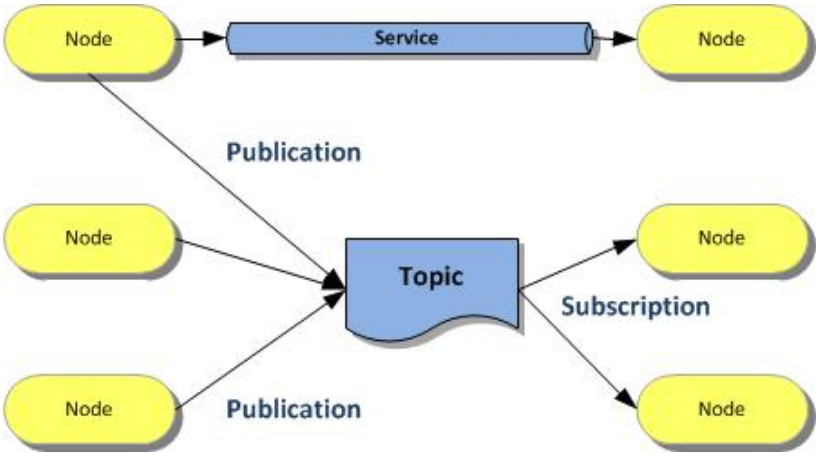


Figure 2.1: ROS structure
Source: `https://tinyurl.com/tkf2smq`

2.2.2 Nodes

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

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 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. The col gets stored in a file called bag, where it records the messages as they come in. 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 they where 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.

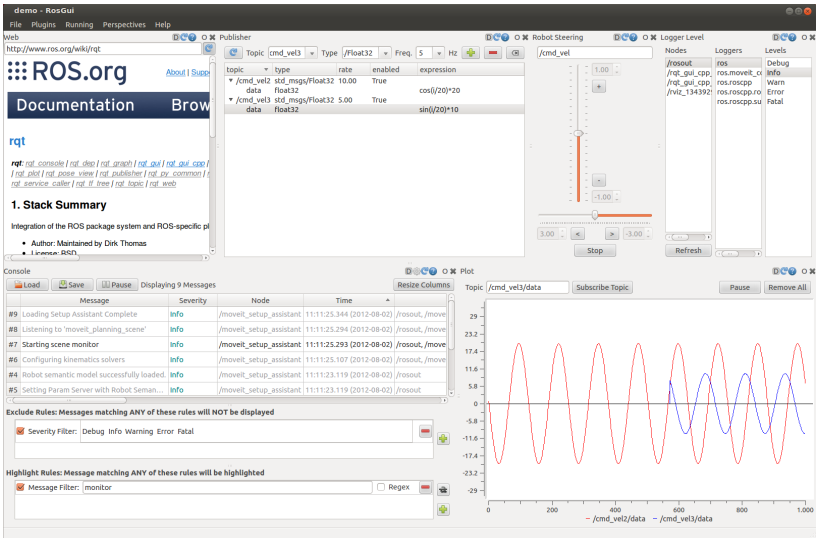


Figure 2.2: RQt interface
Source: `https://wiki.ros.org/RQt`

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.

2.4.4 Rviz

A visualizer for three-dimensional position-data where robots, environments and sensor information can be visualized. It is highly customise able with display many types of visualisation and plugin support. The typical interface on startup looks similar to figure 2.3. On the left are the subscribed topics and on the right are settings for displaying the incoming information.

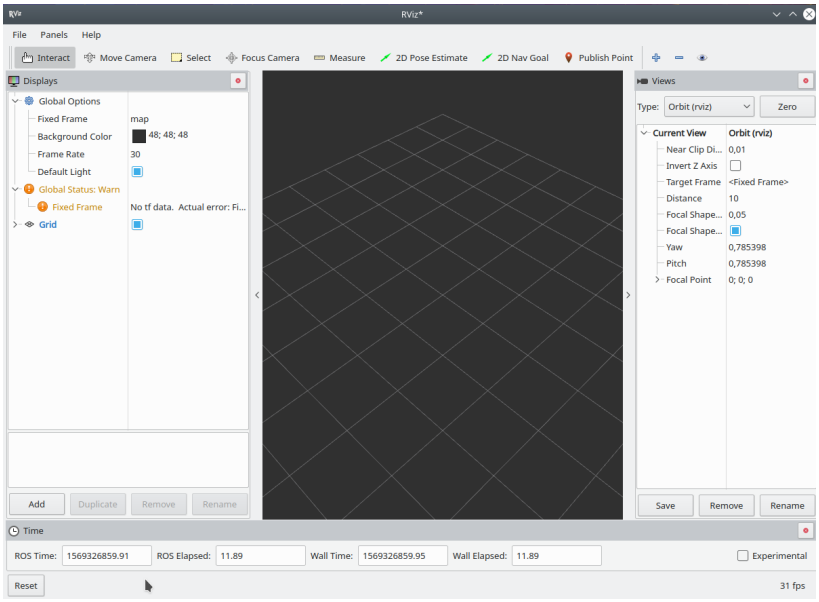


Figure 2.3: Rviz interface

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's easy to make a automated startup and configuration process to be executed with one command. It's possible to execute launch files in other launch files to chain them together.

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). 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 three main components(neurons, connection and the weight associated with them) the propagation function and a bias. In the following topics I will give you a short summary what these components are and afterward I will explain how they work together.

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. 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 Type of ANN they are placed in different structures. In most cases the output of a Neuron is a number between 0 and 1.

3.3.2 Connection and weights

These Neurons are Connected. Which neurons are connected with others depends on the structure. The weights characterize how important a connection between neurons is.

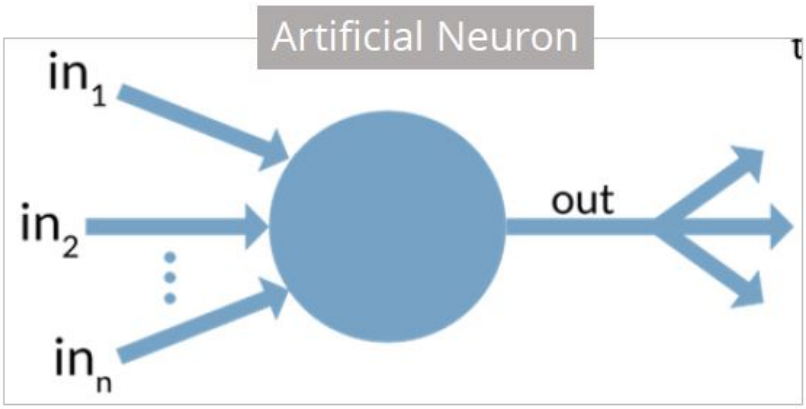


Figure 3.1: General view of Neurons
Source: https://tinyurl.com/yyfthk7c

For example:
A neuron, we call it base for this example, has two neurons connected to it as inputs. The weight of the connection of the first neuron has a bigger weight than the second connection. That means the output of the base depends more on the input of the first neuron

3.3.3 Propagation function and activation function

This is a function witch 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 wich means the output doesn't change instantaneously. Thats important for the training algorithmus.

3.3.4 Bias

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 is visible.

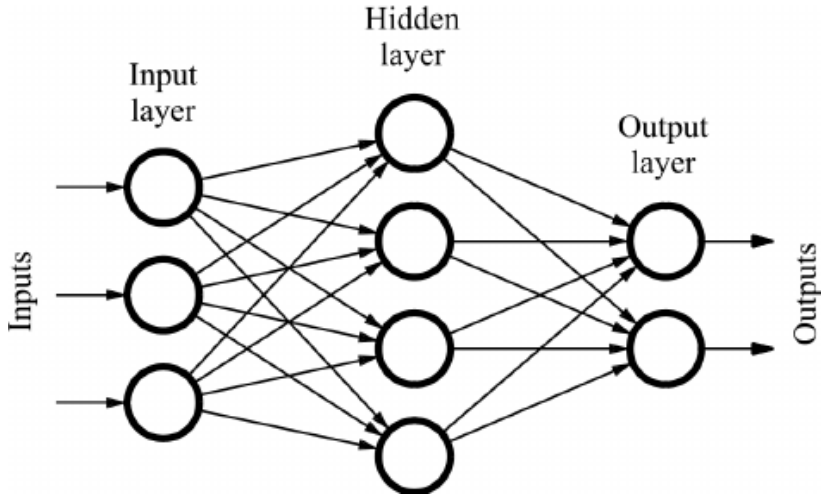


Figure 3.2: Feed forward network

3.4.2 CNN

Convolution Neural Network (CNN) is the most important Network organization for our task. It is based on the human visual cortex and perfect 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, 19] will be used

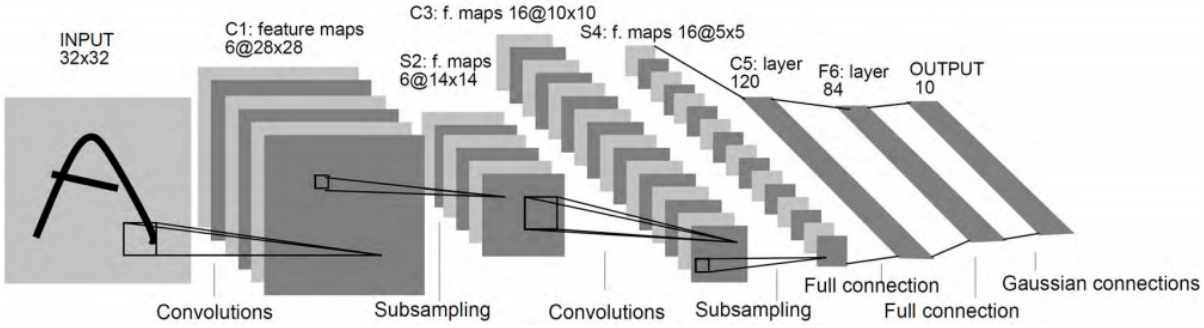


Figure 3.3: 7-layer architecture of CNN for character recognition
[18, 19, Fig. 4.]

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

4 ORB-SLAM2^{AV}

4.1 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 [20]. It works in a wide variety of use cases. The SLAM can be used on a small hand-held camera or drones up to self driving cars. ORB-SLAM2 is based on ORB-SLAM and was inter alia developed by Raúl Mur-Artal who already worked on ORB-SLAM.

Figure 4.1: ORB-SLAM Example image
Source: `https://tinyurl.com/ruvnj39`

4.2 How does the ORB-SLAM work?

4.2.1 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 independent from the camera type. [20]

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

- Stereo Image
  For a stereo camera setup the keypoints get extracted 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.

4.2.2 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).[21] 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 heavier task.
When finished, the updated and optimized keyframes and keypoints get merged into the original keyframes and keypoints [20].

4.2.3 Localization

When a 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 [20].

4.2.4 Input/Output

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

5 LSD-SLAM^{AV}

5.1 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.

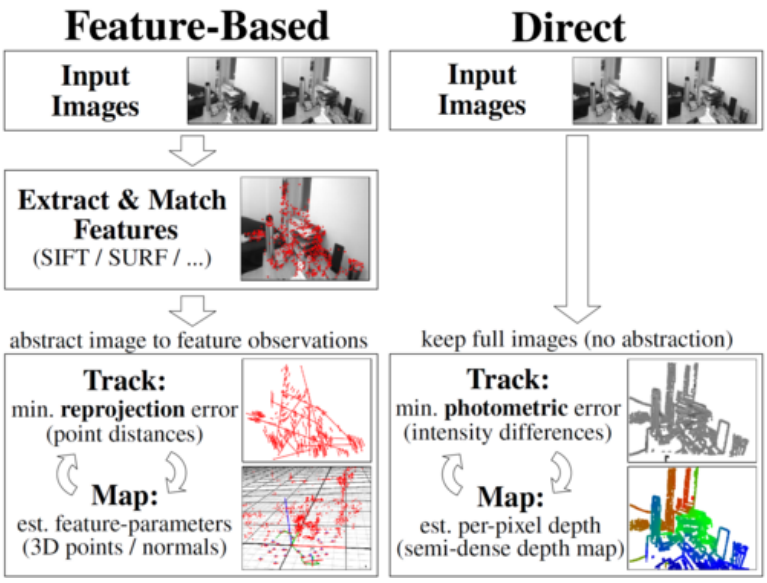


Figure 5.1: Feature-Based and Direct Difference
Source: `https://tinyurl.com/ycmjhb9d`

5.2 Difference Feature-Based and Direct

- Feature-based
  A feature-based SLAM (e.g. ORB-SLAM 4) 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].

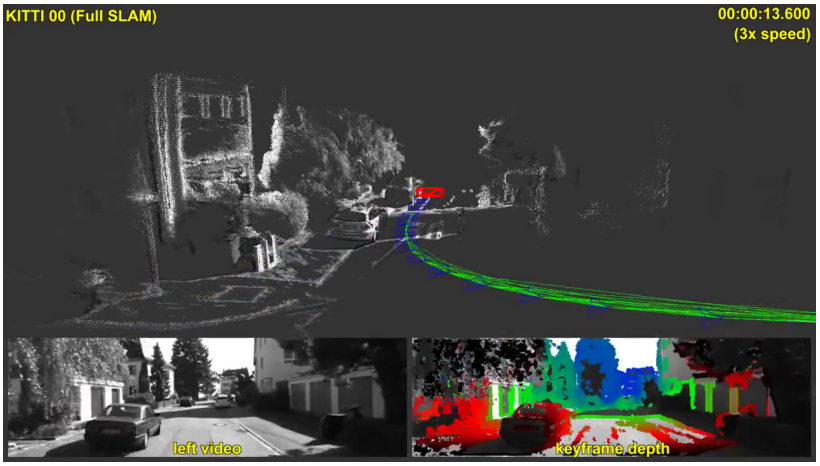


Figure 5.2: LSD-SLAM Example image
Source: `https://tinyurl.com/qkuamyy`

5.3 How does the LSD-SLAM work?

5.3.1 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.

5.3.2 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].

5.3.3 Map optimization

Scaling, rotation and movement isn't always perfect, which results in drift. Even if the drift effect is little it adds up, which might result in some very off map [11]. The Pose-Graph-Optimization is a optimization algorithm which aims to fix these drifts with pretty good results. The advantages of the pose-graph-optimization are that it's fast and vulnerable for poor initialization estimates [22].

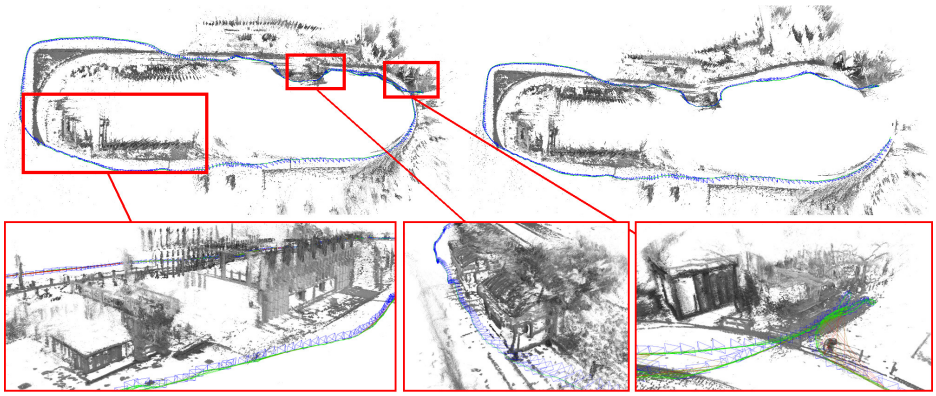


Figure 5.3: LSD-SLAM Pose-Graph-Optimization (left after loop-closure, right before loop-closure)
Source: `https://tinyurl.com/qkuamyy`

5.3.4 Input/Output

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

6 DeepTAM^{SM}

This Chapter is based on the work of:"DeepTAM: Deep Tracking and Mapping with Convolutional Neural Networks".[23]

6.1 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 [24].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.

6.2 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 [23].

6.2.1 Network Architecture

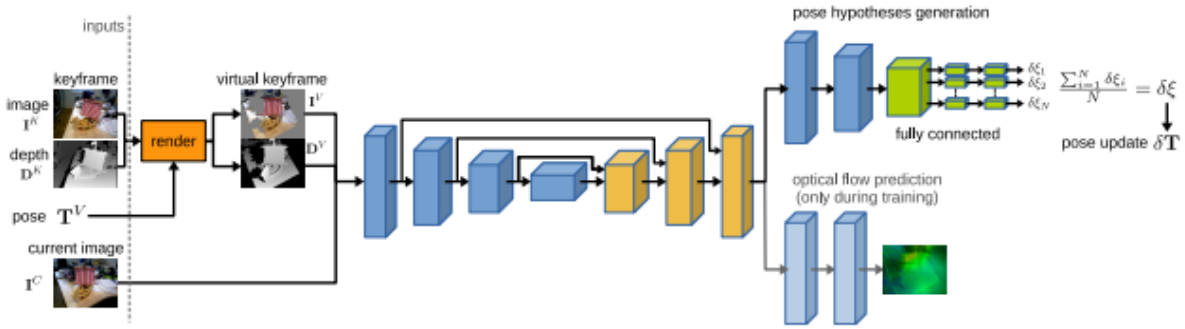


Figure 6.1: Schematic of DeepTAM
Source: http://lmb.informatik.uni-freiburg.de/Publications/2019/ZUB19a

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.

6.3 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.

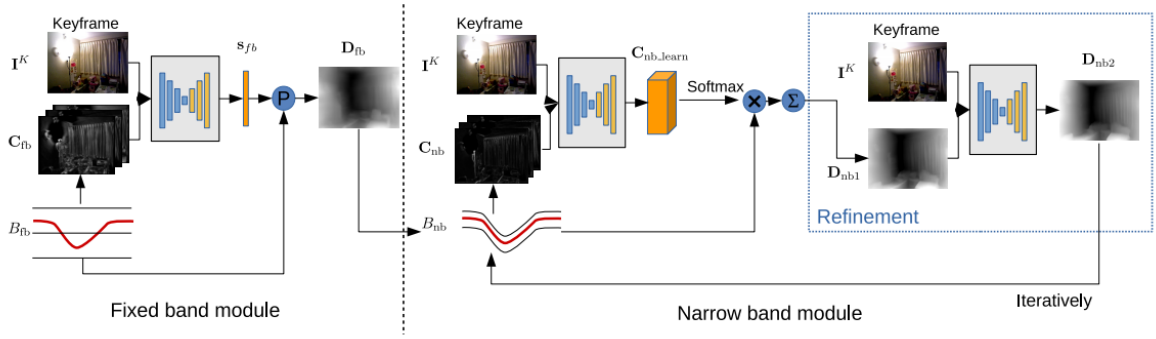


Figure 6.2: Mapping Networks Overview
Source: http://lmb.informatik.uni-freiburg.de/Publications/2019/ZUB19a

6.3.1 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.

6.3.2 Training

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

7 Workflow

7.1 Used Hardware[AV]

For video capturing a *Raspberry Pi 3B+* 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.
For processing a *Lenovo Think-Station S20* or a *Lenovo W550s* are used depending on the amount of processing power is required. For more intense work a server access at the Johannes Kepler University was supplied to work on their system.
As the work is based around implementing it on the Audi Autonomous Driving Cup (AADC) car a remote controlled model car was borrowed for a few weeks.

7.2 Used Software[AV]

7.2.1 Raspberry Pi

The Raspberry Pi is running Raspbian Buster since it is well optimized 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.

7.2.2 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.

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 at the 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 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.

7.3 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 are connected to the router over LAN. This makes the camera setup very portable as the pi, camera and powerbank are packed together and don't have much weight. The PC can sit somewhere where and just processing the received video signal.

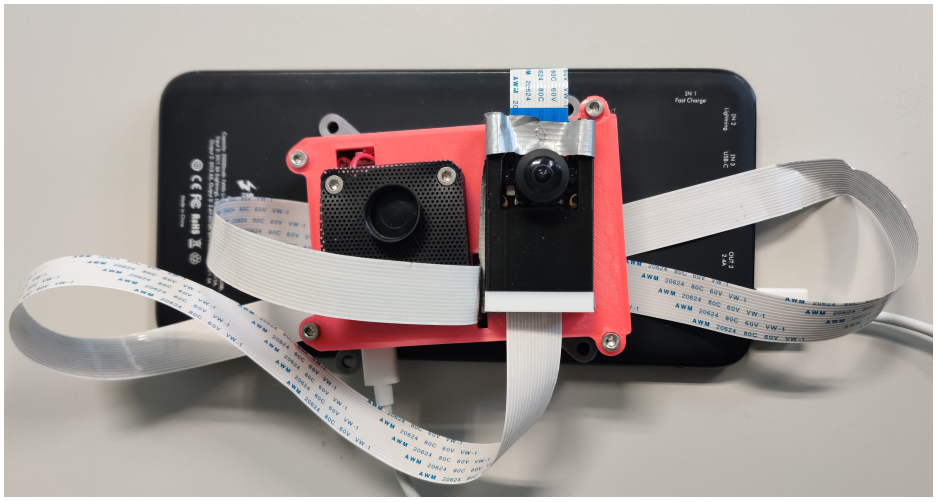


Figure 7.1: Stream setup with Raspberry Pi, camera and powerbank

7.4 Streaming video from Pi to PC[AV]

7.4.1 Enabling Camera

To use a camera on a Raspberry Pi the interface needs to be enabled first. This can be done in the built-in tool called *raspi-config*. In this tool under the subsection called *Interfacing Options* there is a option with the name *Camera*. When this is done the camera can be used after a restart.

7.4.2 Python Script

In the code snipped 7.1 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 and for the format 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.
After the camera "recording" has started successfully the server is started to make the stream 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 7.1: Main Function of Camera Feed

The streamingHandler that is shown in snipped 7.2 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. In addition to the status code headers are send, which tell the client 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 an JPEG image. The loop ensures that the always client receives the latest image and so creates a video. Should the client drop the connection an exception is raised which causes the loop to stop and end 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')
```

```
10              self.send_header('Pragma', 'no-cache')
11              self.send_header('Content-Type', 'multipart/x-mixed-replace;
                    boundary=FRAME')
12              self.end_headers()
13
14              # start Stream
15              try:
16                  while True:
17                      # Send new image when available
18                      with output.condition:
19                          output.condition.wait()
20                          frame = output.frame
21                      # Header for sending image
22                      self.wfile.write(b'--FRAME\r\n')
23                      self.send_header('Content-Type', 'image/jpeg')
24                      self.send_header('Content-Length', len(frame))
25                      self.end_headers()
26                      self.wfile.write(frame)
27                      self.wfile.write(b'\r\n')
28              except Exception as e:
29                  print('Removed streaming client %s: %s', self.client_address,
                        str(e))
30          else:
31              # User requested something else --> Send 404 Page
32              self.send_error(404)
33              self.end_headers()
```

Listing 7.2: Streaming Handler of CamStream

The behavior when a new image from the pi camera is ready to send is defined in the snipped 7.3. 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 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.

```
1  class streamingOutput(object):
2      def __init__(self):
3          self.frame = None
4          self.buffer = io.BytesIO()
5          self.condition = Condition()
6
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 7.3: Streaming Output of CamStream

7.5 Receiving images on PC and Laptop[AV]

For receiving the images on the PC or Laptop an existing ROS-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 [25].

7.5.1 MJPG-Stream receiver

To automate the startup procedure of the node a roslaunch file is used to start the node is written for and automatically sets the required parameters.
The launchfile shown in 7.4 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 7.4: MJPG-Stream receiver Launch file

7.6 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 7.2 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.

7.6.1 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 is used since it has 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.

Figure 7.2: **Left Image:** normal Raspberry Pi Camera. **Right Image:** wide angle lens camera

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 7.5: 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 7.3. These highlighted edges are points which are used to calculate the distortion parameters using an algorithm that was developed by OpenCV [26].

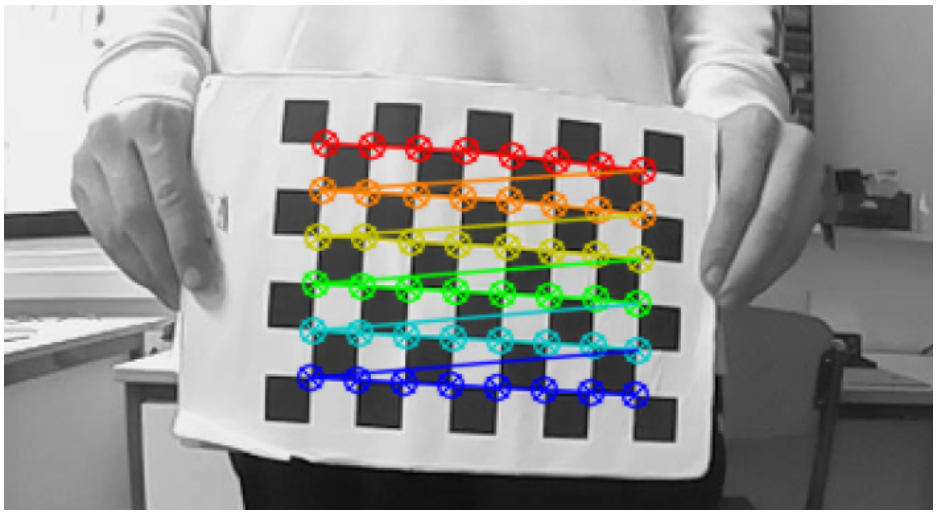


Figure 7.3: Camera calibration

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 something like shown in figure

7.6. The output files are normally compatible with all applications without any issues.

```
image_width: 1280
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 7.6: Calibration file

7.7 Running Monocular ORB-SLAM2[AV]

Integrating ORB-SLAM2 4 is pretty straight forward since there is already ROS version of it which is very well maintained by **Lennart Haller** on behav of *appliedAI-Initiative* on GitHub [27]. It just needs to be cloned into the *src* folder of the ROS-workspace and complied 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 sources using the *source setup.sh* command. After sourcing the setup all new types of commands that are added by the nodes are available.

7.7.1 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 7.9 and thus the config that the calibration tool generated needs to be converted manually. In the camera_matrix the values are the following:

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

Listing 7.7: Matrix listing

The distortion_coefficients need to be mapped like shown in listing 7.8.

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

Listing 7.8: 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.

```
# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 600.43227
Camera.fy: 605.883
```

```
Camera.cx: 560.29343
Camera.cy: 253.24984

# Camera distortion parameters (OpenCV)
Camera.k1: -0.253310
Camera.k2: 0.043398
Camera.p1: 0.005101
Camera.p2: -0.00329
Camera.k3: 0.0

Camera.width: 1280
Camera.height: 720
```

Listing 7.9: ORB-SLAM2 config calibration

7.7.2 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 7.10 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*.

```
<launch>
  <node name="orb_slam2_mono" pkg="orb_slam2_ros"
      type="orb_slam2_ros_mono" output="screen">

       <param name="publish_pointcloud" type="bool" value="true" />
       <param name="publish_pose" type="bool" value="true" />
       <param name="localize_only" type="bool" value="false" />
       <param name="reset_map" type="bool" value="false" />

       <!-- static parameters -->
       <param name="load_map" type="bool" value="false" />
       <param name="map_file" type="string" value="map.bin" />
       <param name="settings_file" type="string" value="$(find orb_slam2_ros)/
           orb_slam2/config/config.yaml" />
       <param name="voc_file" type="string" value="$(find orb_slam2_ros)/
           orb_slam2/Vocabulary/ORBvoc.txt" />

       <param name="pointcloud_frame_id" type="string" value="map" />
       <param name="camera_frame_id" type="string" value="camera_link" />
       <param name="min_num_kf_in_map" type="int" value="5" />
  </node>
</launch>
```

Listing 7.10: ORB-SLAM2 launch file

When starting RQt 2.4.2 and opening the Node Graph it should show, like seen in figure 7.4 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.

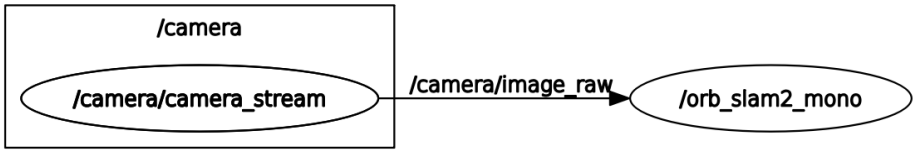


Figure 7.4: Node Graph with Stream and ORB-SLAM

7.8 Running Stereo ORB-SLAM2[AV]

As explained in the ORB-SLAM chapter 4 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. This then is works a bit more like eyes where the depth perception is easier when having both eyes open compared to only having one open.

7.8.1 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. 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.

7.8.2 Calibration

Calibration in this testing environment is pretty straight forward as the cameras used are the normal Pi Cameras v2 which basically have little to no distortion. Just for being on the save side calibration was done on both cameras and the values only are slightly different. So the format that was entered in the config was the same as on the monocular SLAM calibration 7.7.1. If cameras with distortion are used there are more values that are required a bit further down in the config.

7.8.3 Software setup

The same Python script as already explained in section 7.4.2 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. The thing to note here is that the camera name must be unique, e.g they may be named LCAMERA 7.11 and RCAMERA 7.12 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 7.11: Left camera stream launch file

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

Listing 7.12: 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 7.13 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 7.13: ORB-SLAM2 stereo launch file

7.8.4 Launching

When starting the stereoscopic launch for the SLAM using the command 7.8.4 ,it should indicate in RQt 2.4.2 that the ORB-SLAM subscribed onto both video streams like shown in figure 7.5. 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.

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

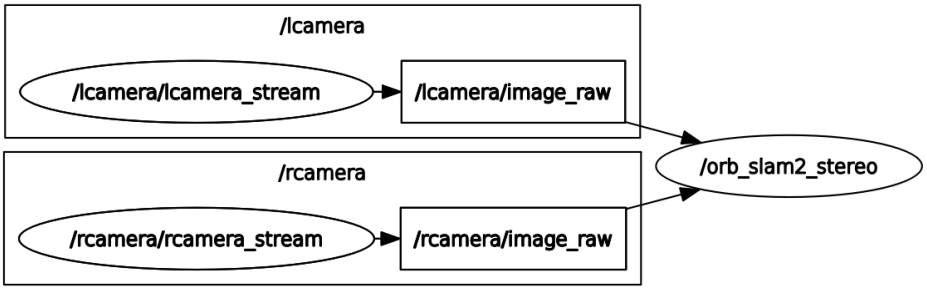


Figure 7.5: Node Graph with Stereo ORB-SLAM2

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

7.9 Running LSD-SLAM[AV]

Running the LSD-SLAM is not as straight forward as it is on the ORB-SLAM 7.7. The implementation of the LSD-SLAM is made to use it with ROS out of the box but the original version [28] 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 [29] 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 [30] 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.

7.9.1 Installing LSD-SLAM

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
```

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
```

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*
```

To install the correct version of G2O it needs to be cloned from `https://github.com/felixendres/g2o.git` 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
```

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 7.14, which has to be replaced with the path of EIGEN.

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

Listing 7.14: CMakeList Eigen Directory

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.

7.10 DeepTAM

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

7.10.1 Ubuntu 16.04

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 for these tasks.

```
pip3 install pew
pew new deeptam
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.

```
sudo apt-get install libeigen3-dev
sudo apt install python3-minieigen
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.

8 Fazit und Persönliche Erfahrungen

8.1 Fazit

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

8.2 Persönliche Erfahrungen

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

Glossary

AADC Audi Autonomous Driving Cup. 19

ADTF Automotive Data and Time-Triggered Framework. 19

FPS Frames per Second. 21, 23, 25

G2O General Graph Optimization. 29, 30

Lidar Laser radar. 1

OSRF Open Source Robotics Foundation. 3

RGB-D Red Green Blue - Depth. 2, 11, 12

ROS Robot Operating System. 3–6, 21, 23, 25, 29, 30, 36

SLAM Simultaneous Localization and Mapping. 1, 2, 11, 13, 26–29

TF TensorFlow. 31

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] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

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

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

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

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

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

List of Figures

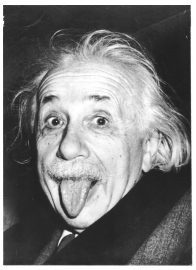
Listings

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