CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# **Visual SLAM in an automotive context:** Implementing ORB-SLAM2 in the OpenDLV framework

Bachelor of Science Thesis in Software Engineering and Management

Linus Eiderström Swahn
Pontus Pohl

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



**Visual SLAM in an automotive context:**
Implementing ORB-SLAM2 inside the OpenDLV framework

Linus. Eiderström Swahn
Pontus. Pohl



Supervisor: Christian. Berger
Examiner: Piergiuseppe. Mallozzi

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

*Abstract*— **Simultaneous Localization and Mapping (SLAM) is a technique frequently used in the area of self-driving cars for mapping and odometry. SLAM has traditionally been performed using laser based range finders of the light detection and ranging (LIDAR) types. Due to the high cost of these sensors there is currently a trend of implementing visually-based SLAM systems using cameras as sensory input. This thesis explores the possibility of integrating a visual-SLAM component into an automotive framework as well as how this visual-SLAM compares to LIDAR based SLAM techniques. Using a state of the art visual SLAM algorithm, ORB-SLAM2, we implement and evaluate a modern visual-SLAM solution within the OpenDLV framework by performing a Design Science Research (DSR) study with the goal of implementing a microservice containing the ORB-SLAM2 algorithm inside of OpenDLV. The software artifact resulting from the DSR study is then evaluated using the evaluation methodology included in the KITTI visual odometry benchmark. Based on the results from this evaluation we conclude that the ORB-SLAM2 algorithm can successfully be integrated in the OpenDLV framework and that it is a possible replacement for LIDAR-based SLAM.**

## I. Introduction

Self-driving vehicles is a technical domain that could be seen as a frontier for the automotive industry. Several major car manufacturers are spending effort and engineers to further develop the techniques and implementations to fulfill this vision[1]. Besides the interest from car manufacturers, the academic community is also very interested in the problems and potential solutions that the area of autonomous vehicles promises. The Chalmers Revere laboratory is a research laboratory at Chalmers university of technology that is dedicated to the study and research regarding self-driving cars. This thesis will be conducted in collaboration with Chalmers Revere.

The area of mobile robotics can be perceived as a precursor to the current research area of autonomous vehicles since many of the problem areas are shared between the application of intelligent vehicles and that of autonomous robots. Specifically the concept of mapping and autonomous driving are areas that research first began in the mobile robotic community but is now also of interest to the domain of autonomous vehicles[2]. In order to solve these problems, many different techniques have been researched and tested[3]. With the development of more advanced sensory hardware such as three dimensional laser based range finders and stereo vision, the possibilities for more advanced techniques for solving these problems have become available[3]. Simultaneous localization and mapping (SLAM) is a technique that allows robots or autonomous vehicles to simultaneously localize themselves in regards to the surrounding environment as well as build a map of the world they navigate in[3].

Among laser based range finders the current state of the art hardware used are of the light detection and ranging (LIDAR) type. Multiple-beam scanning LIDAR devices are the most common type of LIDAR used in autonomous driving and one major manufacturer of such sensors is a company called Velodyne, that makes several different models[3]. A LIDAR works by rotating a laser based range finder, or several vertical ones in the case of multiple-beam LIDARS, at high speeds and measuring the distance from the LIDAR to obstacles surrounding the LIDAR using the time of flight of the laser beam[3]. Even though Velodyne LIDARS have decreased in price recently[4], the more advanced models still carry a price tag upwards of 75,000$[5]. In comparison, stereo cameras that can be used for visual SLAM carry a price tag upwards of 500$[6].

Two solutions that represent the current state of the art when it comes to LIDAR-based SLAM are IMLS-SLAM developed by Jean-Emmanuel Deschaud[7] and LOAM by Ji Zhang and Sanjiv Singh[8]. IMLS-SLAM uses a scan-to-model matching framework in order to get the trajectory the LIDAR has traveled between scans. The LOAM solution relies on two algorithms running in parallel. One algorithm performs high frequency, low fidelity odometry estimation while the other algorithm performs at much lower frequency but does a fine matching and registration of the point cloud the LIDAR outputs. LOAM and ILMS are ranked second and third respectively on the KITTI Odometry benchmark leader boards[9].

A topic that has become more and more popular in recent years is that of Visual SLAM[3]. Visual SLAM is a collection of methods for implementing the SLAM technique using image-based input from Monocular, Stereo or RGB-D cameras as opposed to that of laser based range finders. The area of Visual SLAM, while being more technically challenging than traditional SLAM also shows promise in that it can provide odometry information which is a key component of modern SLAM systems[3]. In the field of autonomous vehicles the area of visual SLAM has become an highly researched topic with several different techniques and implementations being developed[2]. A common focus of the advancements being done in the automotive visual SLAM area is the problem of minimizing the location drift that can occur with SLAM systems[3], [10].

ORB-SLAM2 is an open-source visual SLAM algorithm that uses a key frame based approach[11], which utilizes visual input from monocular or stereo cameras as input and outputs the camera pose for each frame along with a map containing points describing the world the camera travels through. Furthermore it utilizes a technique called loop closing, that enables it to effectively adjust it's trajectory and have a low error margin on estimated positions. Maintaining an accurate trajectory even after several kilometers without drift is one of the big challenges of visual SLAM [2]. ORB-SLAM2 has been tested on standard desktop hardware in the form of an Intel Core i7-4790 desktop computer with 16Gb RAM, meaning it doesn't require specialized computational hardware but instead can function on hardware easily accessible to anyone[11].

The Kitty Vision Benchmark Suite is a useful resource when developing SLAM algorithms[9]. It provides camera, LIDAR and ground truth data as well as an evaluation methodology that can be used when developing SLAM solutions. The data is compiled from test drives conducted by a group of researchers based in Germany[12]. The dataset is used as a benchmark by several researchers focused on the

SLAM problem[9]. In this thesis, the Kitti Vision Benchmark Suite will be used during the development and evaluation of the implementation.

With the high cost of LIDAR sensors when compared to cameras, visual SLAM solutions has the potential to provide autonomous vehicle systems with a low cost replacement or complement for some of the problems that a SLAM system solves[3]. With this in consideration there is an interesting possibility of integrating a visual SLAM algorithm in the OpenDLV framework being developed at Chalmers Revere laboratory. OpenDLV is an open source framework for autonomous cars. It provides a microservice based architecture that is designed to be run on autonomous vehicle platforms[13]. In this thesis we aim to investigate how well a vision-based implementation of SLAM based on the ORB-SLAM2 algorithm could be integrated into the OpenDLV framework and in what way such an implementation can benefit the automotive framework compared to the traditional LIDAR-based SLAM approach.

Our hypothesis is that by integrating the ORB-SLAM2 algorithm in the OpenDLV framework we will be able to evaluate it's suitability to be used in a real world autonomous car platform. By doing this we will contribute to the body of research by evaluating this approach to solve visual SLAM in a real world automotive context. By being the first to implement the ORB-SLAM2 algorithm inside of an automotive platform like OpenDLV our work will benefit future students, researchers, and upcoming projects using the OpenDLV framework as well as provide additional evaluation of ORB-SLAM2 as a visual SLAM algorithm. In addition, we will provide our implementation as open source using GPL-3.0[14] license and make it available on Github at `https://github.com/chalmers-revere/opendlv-perception-vision-orbslam2`.

## II. Related Work

The Springer Handbook of Robotics by Bruno Siciliano *et al*: provides a comprehensive coverage on the field of robotics. In particular there is a chapter covering the basic theory, history and current work that has been done on the SLAM topic. In our case this book provides domain overview and historical background as well as a reference for our own research[3].

Focusing more specifically on the area of autonomous cars, Guillaume Bresson, Zayed Alsayed, Li Yu and Sébastien Glaser has written the article: *Simultaneous Localization And Mapping: A Survey of Current Trends in Autonomous Driving*[2] which provides a background and an overview of the current state of technology regarding SLAM and visual SLAM in the area of autonomous cars specifically. This paper gives background to our problem domain and will help motivate why this thesis should be conducted.

Prior research in the domain of visually based SLAM has been conducted by R. Mur-Artal and J. D. Tardos with their papers on ORB-SLAM and ORB-SLAM2. ORB2-SLAM is a key-frame based visual SLAM algorithm basing its feature extraction on the computationally efficient ORB (Oriented Fast and Rotated Briefs)[15]. ORB-SLAM is able to operate in real-time without using any other processing power than what is available in consumer grade CPU's by using these efficient ORB features. The ORB-SLAM algorithm, performing visual SLAM using a monocular camera is explained in further detail in the the paper *ORB-SLAM: a Versatile and Accurate Monocular SLAM System* by R. Mur-Artal and J. D. Tardos[11]. The ORB-SLAM2 algorithm improves on the efficient SLAM solution introduced in ORB-SLAM by introducing the usage of either a stereoscopic or RGB-D camera as well. Using stereoscopic images as input for the algorithm fixes the issue with inconsistent scaling that occurs in monocular SLAM solutions[3]. The ORB-SLAM2 algorithm is further described in the article *ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras* by R. Mur-Artal and J. D. Tardos[11]. Since this thesis focuses on adapting the ORB-SLAM2 algorithm for use inside of OpenDLV these two papers provides the technical background for understanding the algorithm.

Because the motivation for this thesis is related to the comparison between visual and LIDAR-based SLAM, ILMS and LOAM has been selected as two LIDAR-based SLAM solutions used as a reference for the evaluation of the work being done in this thesis. The article *"IMLS-SLAM: scan-to-model matching based on 3D data"* by Jean-Emmanuel Deschaud[7] provides background, implementation and results for the ILMS algorithm. The article *"Low-drift and real-time lidar odometry and mapping"* by Ji Zhang and Sanjiv Singh[8] describes the LOAM SLAM solution in great detail, including the results of their work.

When it comes to evaluating SLAM solutions, the KITTI Vision Benchmark Suite provides both sensor data to use as input as well as an evaluation methodology for enabling relevant comparisons between different SLAM algorithms and implementations. We have opted to use this method for SLAM evaluation based on its popularity[2] and prevalence of comparable results, particularly in the form of the two LIDAR-based SLAM algorithms described in the previous paragraph. The evaluation methodology is described in greater detail in the article *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite* by Andreas Geiger, Philip Lenz and Raquel Urtasun[16].

Our proposed solution will be implemented as a microservice inside of OpenDLV, an open-source framework for autonomous cars being developed at Chalmers Revere lab. The OpenDLV framework will provide the overall structure and context where our proposed solution can function and will allow us to test our solution in a real autonomous driving scenario. The OpenDLV framework is described in more detail in the article *Containerized Development and Microservices for Self-Driving Vehicles: Experiences & Best Practices* by Christian Berger, Bjornborg Nguyen, and Ola Benderius[13].

## III. Methodology

Our research questions will be focused on the utility a camera-based visual SLAM solution can provide in the context of an automotive framework as well as how it

compares to LIDAR-based SLAM solutions. The algorithm that will be used for this comparison is the ORB-SLAM2 algorithm described in the introduction and the automotive framework will be the OpenDLV framework also described in the introduction.

(RQ I) To what extents can the ORB-SLAM2 algorithm be used as a visual SLAM component inside of the OpenDLV framework?

(RQ II) To what extents can the ORB-SLAM2 algorithm be considered a replacement to a LIDAR-based SLAM solution in the OpenDLV framework?

In order to answer our research questions, we will conduct a design science study with the goal of implementing the ORB-SLAM2 algorithm inside the OpenDLV framework as well as implementing the necessary infrastructure and visualization to properly evaluate and use this new component inside the framework. The reason for choosing design science as our research methodology is because of the strengths the methodology has when it comes to applying research to a real world problem[17].

The design science methodology will be constructed according to the six activities for Design Science Research(DSR) study[18]. These six steps will be described in further detail below.

### A. Identification and motivation of problem

The first of the step of DSR covers the identification of the research problem and the motivation for carrying out the research being conducted.

In this thesis we have identified that there exists a large interest in the research community towards implementing visually-based SLAM solutions as a complement to the traditional LIDAR-based SLAM solutions. Among these visual SLAM solutions we have identified ORB-SLAM2 as a suitable candidate to further evaluate. Whilst the ORB-SLAM2 algorithm has been tested on an automotive dataset such as KITTI, we have not found any prior evaluation using a real automotive framework which is our motivation for doing such an evaluation. Furthermore, the autonomous vehicle framework OpenDLV currently lacks a visually-based SLAM component, which is why it's of relevance to implement ORB-SLAM2 in this framework. This implementation will provide the means of further evaluating the suitability of the ORB-SLAM2 algorithm in the automotive context as well as further extending the functionality of the OpenDLV framework.

### B. Objectives of solution

The second of the six steps describes what the implemented software artifacts objectives are in regards to the problem defined in the previous step.

In this thesis we aim to provide a software artifact containing the ORB-SLAM2 algorithm and visualization adapted to work within the OpenDLV framework. This artifact will consist of a microservice containing the ORB-SLAM2 algorithm and a microservice containing visualization for the output generated by the first microservice. The first microservice will be self contained and receive image data over the OpenDLV framework from either recorded data sets or a live camera feed. The microservice will output consisting of a camera trajectory as well as a map of points describing the world around the vehicle. This data will be outputted over the OpenDLV conference system for consumption in other connected microservices containing visualizing components or potential driving logic. The second microservice will be a consumer of the output of the first microservice and allow visualization of the trajectory as well as map points delivered from the first microservice.

The first objective of this artifact is to provide a functional implementation of the ORB-SLAM2 algorithm contained in a OpenDLV microservice. A second objective is to enable us to compare the output of this implementation with that of the original unmodified ORB-SLAM2 algorithm as well as other SLAM solutions.

These objectives are motivated by the problem found in step one. It's further motivated by the fact that such an implementation can be of future use to students and researchers working with the OpenDLV framework.

In addition to implementing the two microservices we will also define a new message set so that OpenDLV can handle the internal communication of the type of camera trajectory and map data outputted by the ORB-SLAM2 algorithm.

### C. Design and development

The third of the steps is the design and implementation of the software artifact that can be used to answer the research questions defined earlier in the thesis.

More in-depth details regarding the software artifact and its implementation will be described in detail in the next section.

The implementation of the ORB-SLAM2 algorithm inside of OpenDLV, has been a collaborative effort together with Marcus Andersson and Martin Baerveldt, two master students at Chalmers University of Technology who will also use this implementation in their master thesis.

### D. Demonstration of solution

The fourth step of the DSR methodology should demonstrate the software artifact.

In order to demonstrate the utility of the implemented software artifact the solution will be using the KITTI odometry data set as input and the resulting camera trajectory will be collected for each sequence. The sequences that will be used are the training data set that includes the ground truth data. Using the Software Development Kit (SDK) provided by the KITTI benchmark, the translational error can then be extracted. This will then be the basis of comparison between this implementation and the original ORB-SLAM2 implementation as well as the two LIDAR-based SLAM solutions, thus demonstrating the utility of the software artifact. In addition to providing the raw results, the visualization provided will also be used to provide a visual demonstration of the output generated by the ORB-SLAM2 algorithm.

Because of the usage of docker containers within the OpenDLV framework[13] an already built solution of the software artifact can easily be run for demonstration purposes by any interested party with the only prerequisite being that they have docker and docker-compose installed locally. Further details can be found at `https://github.com/chalmers-revere/opendlv-perception-vision-orbslam2`.

### *E. Evaluation of solution*

The fifth step is the evaluation of the software artifact and it's output.

The evaluation of the software artifact will be made using dynamic analysis which is an analytical design evaluation method. We will use a combination of performance and accuracy metrics in order to evaluate how well our implementation of ORB-SLAM2 compares to the original implementation. We will also use the accuracy metric in order to compare this visual SLAM implementation with that of LIDAR-based SLAM solutions. The performance metrics that will be used for evaluation is the average CPU an memory. The amount of translational drift will be used as an accuracy metric during evaluation.

One of the motivations of using the ORB-SLAM2 as a visual SLAM component inside OpenDLV was the stated efficiancy of the algorithm and its abillity to run on consumer-grade hardware as described in the related works section. In order to properly evaluate the software artifact based on the objective of providing a functioning implementation of ORB-SLAM2 inside of OpenDLV, performance is one of the metrics of intereset to compare between our implementation and the original. Since visual SLAM solutions demands more computational power than traditional SLAM[3] its important for us to evaluate wether or not our implementation is as computational efficient as the original solution. To provide comparable results between our implementation and the original we will utilize the KITTY training dataset[12] to gather metrics on the system load when the microservice runs. By running both algorithms on all sequences in the training set we can measure the average cpu-load and the average memory-load for each sequence for both algorithms. Average and median values will be calculated for the complete set of cpu averages and memory averages recorded. The data used for this evaluation will be carried out on a consumer desktop computer with the following specifications: quad-core Intel i5-7600K CPU @ 3.80GHz, 16 GB DDR4 @ 2933MHZ. In order to capture the required data, the linux utility ps will be used which is a wrapper application around the Linux pseudo file system[19] allowing access to kernel states. When the algorithm finishes processing a dataset, the ps utility calculates the average CPU load by summing together the cpu time spent in user and kernel space, as well as CPU time spent waiting for child processes. This value is then divided by the total time that the process has been running.

$CPU\% = S/T.$

Finally to achieve a value that represents the fraction of the total available CPU resources, the value is divided by number of CPU-cores. Memory usage is computed by calculating how much memory the process has allocated divided by the total amount of memory.

Since one of the main features of SLAM is the ability to measure odometry this is also the focus area of our evaluation in order to answer the question whether or not Visual SLAM can provide a replacement technology to LIDAR-based SLAM. As discussed in the introduction, one of the most important properties of a SLAM implementation is the amount of drift that occurs. Since the drift accumulates with time, it's of the utmost importance to have as little translational drift as possible. The KITTI Vision Benchmark Suite provides a odometry benchmark that we will use for evaluating the drift in our implementation. The KITTI Vision Benchmark Suite provides both camera and LIDAR datasets as well as ground truth which allows different algorithms and implementations to be evaluated on equal grounds. The odometry benchmark also includes an evaluation methodology that enables us to compare this implementation with that of other SLAM solutions, including LIDAR-based ones[9]. We will also be able to compare this implementation with the original ORB-SLAM2 algorithm.

The KITTI team bases their evaluation methodology on the work done by Kummerle at al.[20]. Their proposed evaluation is based on measuring translational and rotational errors on a pose by pose basis. The contrasting evaluation would instead be based on just comparing the end pose with the ground truth. This contrasting evaluation is flawed in the sense that an algorithm might perform flawlessly for 99% of frames, but if a minor rotational error occurs early in a dataset, the end result can be very different from the ground truth even though technically only minor drift occurred.

The evaluation method used by the KITTI team extends the evaluation method proposed by Kummerle at al. by separating rotational and translational errors into two separate metrics[16]. In order to get comparable results for our implementation of ORB-SLAM2 we have used the SDK that KITTI provides alongside the datasets in their benchmark. This SDK contains code that outputs these translational and rotational errors as well as plots and graphs illustrating the difference with the ground truth. This tool is the same as the one used to evaluate the algorithms residing on the official KITTI Odometry leader boards[9].

The data that we will base this evaluation on is the training data comprised of sequence 00 to 10 of the Odometry benchmark challenge. The reason for using this data is that it contains ground truth data thus allowing us to calculate the translational error. There also exists other SLAM solutions that have published their results on this dataset thus allowing us to compare different solutions with each other. Since we want to make a comparison on a sequence by sequence basis in some cases, we have calculated the averages for each sequence and will provide them alongside comparable data from other algorithms in the result section.

The OpenDLV framework doesn't currently have an implementation of a LIDAR-based SLAM solution. This means that in order to make a comparison with LIDAR-based

SLAM we will have to use other SLAM implementations and have them act as a replacement in order to draw conclusions on our work. The algorithms wee have chosen to compare this implementation of ORB-SLAM2 with is LOAM[8] and IMLS-SLAM++[7]. The reason for these in particular is that they currently rank number two and three respectively on the KITTI leader boards, meaning they could be seen as state of the art LIDAR-based SLAM implementations. Lastly they both have published their results with the training data set, thus allowing us to make a direct comparison with our results.

### F. Communication of findings

The final step of the DSR method is how the findings of the research is communicated with the rest of the world.

This thesis will be available online after the date of approval. As mentioned in the introduction the software artifacts produced as a part of this thesis will be available online and under the open source license GPL-V3.

## IV. The design and implementation of the software artifact

OpenDLV is a microservice based framework. Because of this, the ORB-SLAM2 algorithm as well as the visualizer have been implemented inside of two self contained components called *opendlv-perception-vision-orbslam2*[21] and *orb-slam-vehicle-viewer*[22] respectively. OpenDLV is based on a data transport library called *libcluon*[23] which enables a common protocol definition for all software in the distributed chain. In our case this means that we can utilize a common communication protocol to exchange messages between the ORB-SLAM2 algorithm process, and the visualizer process.

### A. ORB-SLAM2

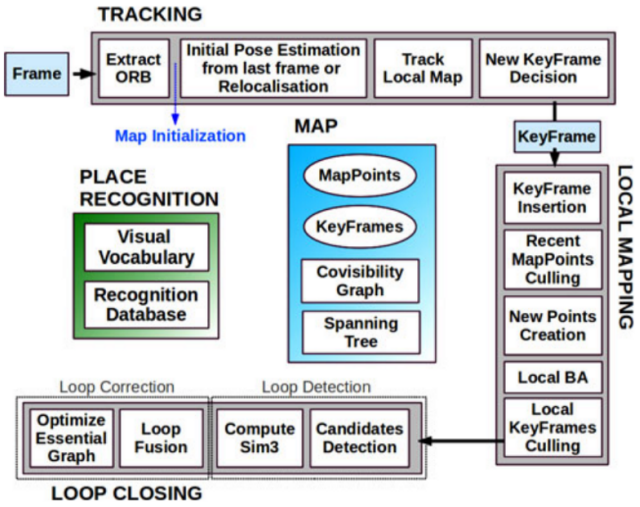
The source code for the ORB-SLAM2 based component can be found at: `https://github.com/chalmers-revere/opendlv-perception-vision-orbslam2`.

The ORB-SLAM2 algorithm itself contains several different components all responsible for extracting 3d-information from 2d pictures. In figure 1 an overview of the ORB2-slam algorithm components is visible.

The algorithm starts by reading an image delivered by the OpenDLV conference. The images sent by OpenDLV could be either be recorded live by the system or replayed from a saved recording. The images are handled using the Mat object from OpenCV which is essentially a Matrix with rows and columns containing pixel data.

In summary, the ORB-SLAM2 algorithm works by using three parallel threads. One thread for tracking features in the current frame and matching them to existing features in previous frames. The second thread manages the local map by updating it with new map points as well as performs local bundle adjustment, readjusting the local map. The third thread applies loop closing and can also launch a fourth thread, which adjusts the entire map. From all frames processed, certain frames are selected to be key-frames which is frames that are useful for localization. The key-frames a connected

Fig. 1. ORB2-slam components




in a weighted graph that links the entire set of key frames, where frames that share observations are linked. This enables efficient graph optimization to be applied to various parts or the entire map. For further reading about ORB2-SLAM, see *ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras* [11].

The output of the algorithm consists of a camera pose as well as a map containing map points. Each frame calculated by the algorithm contains the camera pose for the camera at that frame. The camera pose is represented by an extrinsic matrix[24]. The extrinsic matrix is composed of the camera rotation R and the translation:

$$\begin{bmatrix} R & | & t \end{bmatrix} = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

In order to transform the extrinsic camera matrix into real world position, we can use the following relationship:

$C = -R^t T$.

We can therefore get the world position of the camera at each frame by taking the negative transposed rotation and multiply it with the camera transform.

Each map point has world position expressed as a x, y and z coordinate. For each frame processed by the algorithm, the camera world position and every new map point is sent in an OpenDLV envelop to the conference for consumption in other components. Since the ORB-SLAM2 algorithm constantly updates the map points and trajectory points, the whole map and trajectory is sent every other second to guarantee that they both stay up to date.

The changes made to the ORB-SLAM2 algorithm when implementing it in the OpenDLV framework can be summarized with the following points:

- In order to conform to the OpenDLV coding standards, the use of raw pointers in the original ORB-SLAM2 code base has been completely replaced with shared_ptr or unique_ptr from the standard c++ library where each type is warranted.

- Since OpenDLV has a strong emphasis on code clarity and follows a different code naming standard for its code base than the original implementation, relevant parts of the algorithm has been refactored naming wise to introduce what we believe to be a clearer purpose of certain parts of the code base.
- The OpenDLV microservices is built on containerization using docker. This means that third-party dependencies has to be handled differently than in the original implementation and work was conducted on getting all functionality to work running both natively and inside the final docker environment. Especially when implementing the loop closing functionality we had to iterate the final docker image several times because of incompatibilities with the g2o library inside the docker environment.

The inputs and outputs of the algorithm has been completely rewritten to work inside the OpenDLV infrastructure. Currently the ORB-SLAM2 OpenDLV microservice can be run either using images sent over the OpenDLV conference system, meaning it can use prerecorded image sets that are played back inside the OpenDLV ecosystem, or by live images sent from a camera component running in OpenDLV. The other way the microservice can be serviced with images is by reading an image set from the file system. This last method has been used extensively for debugging and testing purposes during development using the KITTI dataset. The ORB-SLAM2 OpenDLV container outputs the results of the algorithm using a new message set containing map points and camera trajectory. For each frame the last changes are sent and every other second the complete map and trajectory is sent. This is done to compensate for the fact that ORB-SLAM2 constantly updates previous map points and trajectory according to new data it encounters. The camera trajectory is transformed from camera space to world space according the equation described above.

Since the original code base for the ORB-SLAM2 algorithm is very large and complex, the following steps were taken during the implementation in order to successfully adapt the algorithm for OpenDLV:

- The algorithm was divided into different parts that could be worked on in parallel. Mainly functionality was divided between orb extraction, tracking, data structures, mapping and loop closing.
- Each section was worked on until functional and then tested together with other implemented parts of the algorithm.
- As soon as the first iteration of the software solution that could produce output was completed, the solution was evaluated.
- By evaluating and implementing improvements the software artifact evolved until the final version was used for capturing the results of this thesis.

### B. Visualizer

The source code for the visualizer can be found at: `https://github.com/guding/orb-slam-vehicle-viewer`.

The visualizer is a web based component implemented in javascript using the 3d library three.js[25] to visualize three dimensional data from the algorithm. Other open-source libraries used includes Twitter-Bootstrap, font-awesome and jquery for visual components and scripting, as well as yarn, gulp and babel for build-tasks. The visualizer displays the results of the ORB-SLAM2 algorithm by receiving each message sent from the first microservice and parsing it. Each message contains arrays of map points and trajectory points as well as the current index that each array should start at. The reason for this message structure is that since a map and trajectory can potentially contain several thousands of points, the data needs to be split between several messages. Internally, the visualizer stores map points and trajectory points in arrays. Since the message sent from the ORB-SLAM2 microservice also contains array indexes these arrays can quickly be filled and rewritten by new messages containing updated trajectories and maps.

In order to visualize the output of the ORB-SLAM2 algorithm a new message set had to be defined in OpenDLV containing the camera pose as well as map points. Using this output the visualizer displays the map points created by the ORB-SLAM2 algorithm along with the camera position at every frame of input.

The OpenDLV visualizer can be seen in figure 2. The visualizer consists of a map displaying the camera trajectory as red dots and map points as white dots. In addition to the map, the status of the connection with the OpenDLV conference is displayed as well as the amount of messages sent per second. When running the whole system the map is updated in real time. The camera is controlled by keyboard input. The camera can be zoomed using the W and S keys. A and D moves the camera sideways. Q and E rotates the camera.

Fig. 2. OpenDLV visualizer


## V. Results

### A. Visual odometry

Shown in table I on page 7 is the output of the evaluation of our ORB2-SLAM implementation. Alongside the results of our implementation is the results from the original ORB2 algorithm[11] and two additional LIDAR-based SLAM

TABLE I

TRANSLATION ERROR FOR INDIVIDUAL SEQUENCES - LOWER IS BETTER

| Sequence | Loop closing | LOAM | IMLS | ORB-SLAM2 Original | **ORB-SLAM2 OpenDLV** |
|---|---|---|---|---|---|
| 00 | Yes | 0.78% | 0.50% | 0.71% | **0.78%** |
| 01 | No | 1.43% | 0.82% | 1.40% | **2.83%** |
| 02 | Yes | 0.92% | 0.53% | 0.76% | **0.81%** |
| 03 | No | 0.86% | 0.68% | 0.79% | **1.29%** |
| 04 | No | 0.71% | 0.33% | 0.44% | **2.31%** |
| 05 | Yes | 0.57% | 0.32% | 0.39% | **0.40%** |
| 06 | Yes | 0.65% | 0.33% | 0.49% | **0.72%** |
| 07 | Yes | 0.63% | 0.33% | 0.51% | **0.54%** |
| 08 | Yes | 1.12% | 0.80% | 1.03% | **1.13%** |
| 09 | No | 0.77% | 0.55% | 0.87% | **1.34%** |
| 10 | No | 0.79% | 0.53% | 0.64% | **0.71%** |
| overall | No | n/a | n/a | 0.96% | **1.66%** |
| | Yes | n/a | n/a | 0.73% | **0.80%** |
| | Both | n/a | 0.55% | 0.77% | **0.94%** |

TABLE II

TRANSLATIONAL ERROR DIFFERENCES WITH ORIGINAL IMPLEMENTATION

| Sequence | Loop closing | ORB-SLAM2 Original | ORB-SLAM2 OpenDLV | **Difference** |
|---|---|---|---|---|
| 00 | Yes | 0.71% | 0.78% | **0.07%** |
| 01 | No | 1.40% | 2.83% | **1.43%** |
| 02 | Yes | 0.76% | 0.81% | **0.05%** |
| 03 | No | 0.79% | 1.29% | **0.50%** |
| 04 | No | 0.44% | 2.31% | **1.87%** |
| 05 | Yes | 0.39% | 0.40% | **0.01%** |
| 06 | Yes | 0.49% | 0.72% | **0.23%** |
| 07 | Yes | 0.51% | 0.54% | **0.03%** |
| 08 | Yes | 1.03% | 1.13% | **0.10%** |
| 09 | No | 0.87% | 1.34% | **0.47%** |
| 10 | No | 0.64% | 0.71% | **0.07%** |
| overall | No | 0.96% | 1.66% | **0.70%** |
| | Yes | 0.73% | 0.80% | **0.07%** |
| | Both | 0.77% | 0.94% | **0.17%** |

algorithms[7][8]. Represented in table I is the amount of translation error when compared to the ground truth data provided by the KITTI datasets. In addition to the results for each sequence, the overall translational drift for the training data set is also available for all implementations except LOAM that didn't publish that information in their paper.

Included in the table is also the information if the sequence contains loop closing possibilities or not.

The first notable observation when looking at the results, is the comparison between our implementation and the original ORB2 SLAM algorithm. Our implementation is performing a bit worse across all datasets but is still very close to the original, which indicates that we have managed to to implement the algorithm without too much loss in accuracy. Overall, the original implementation is **0.17%** better than our implementation. When comparing datasets with or without loop closing, the original implementation is **0.07%** better with loop closing and **0.70%** better without.

The difference in drift between the original implementation and our implementation can be more properly visualized in figure 3 and 4. As explained in section IV, the output from the algorithm is the camera pose at each frame of input. By

Fig. 3. KITTI-dataset Sequence 00 trajectories

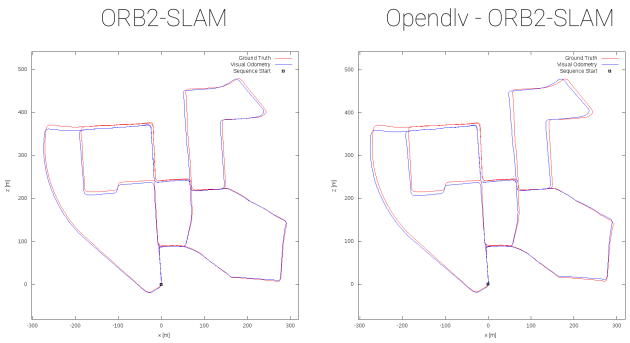
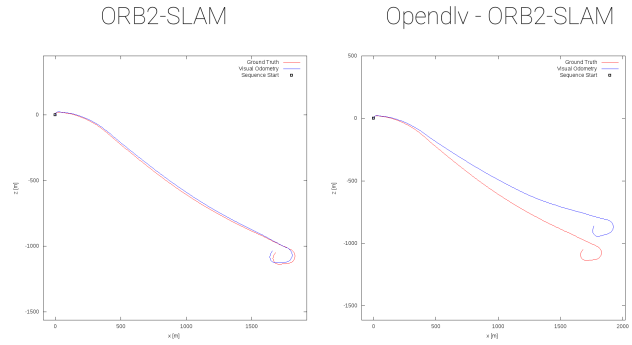



Fig. 4. KITTI-dataset Sequence 01 trajectories




applying the transformation also described in section IV on the camera pose, we get the real world position. Both figure 3 and 4 have drawn the position at each frame, creating the trajectory that the vehicle has traveled. Also included in each figure is the ground truth provided by the KITTI data set which means we can visualize the drift in position that occurs between the SLAM algorithm and where the vehicle actually is. In both figures the output of our artifact is shown on the right and the output from the original ORB2-SLAM is shown on the left. As we can see there is a smaller difference between the implementations in figure 3, which shows the trajectories for dataset sequence 00 that contains loop-closing compared to figure 4 that shows a sequence without loop closing.

Secondly, both the original ORB2-SLAM as well as our implementation performs very similarly to the LIDAR based algorithms. The best performing algorithm IMLS-SLAM[7] is ahead by a fraction on most datasets. When it comes to LOAM it gets out performed by the original ORB-SLAM2 while our implementation is sometimes better and sometimes worse.

### *B. performance*

Table III and IV shows output from the performance evaluation of our implementation of the ORB2 algorithm and the original implementation. Our implementation reaches a CPU usage average of 38,05% across the datasets with a

TABLE III

CPU USAGE - LOWER IS BETTER

| Sequence | ORB2(Original) | ORB2(opendlv) | **Difference** |
|---|---|---|---|
| 00 | 30.75% | 38.75% | **8.00%** |
| 01 | 43.00% | 41.00% | **-2.00%** |
| 02 | 31.75% | 37.25% | **5.50%** |
| 03 | 30.00% | 37.50% | **7.50%** |
| 04 | 37.25% | 37.00% | **-0.25%** |
| 05 | 30.25% | 40.75% | **10.50%** |
| 06 | 37.50% | 39.50% | **2.00%** |
| 07 | 26.75% | 37.50% | **10.75%** |
| 08 | 28.75% | 36.50% | **7.75%** |
| 09 | 30.00% | 36.25% | **6.25%** |
| 10 | 26.75% | 36.50% | **9.75%** |
| average | **32.07%** % | **38.05%** | **5.98%** |
| median | **30.25%** % | **37.50%** | **7.50%** |

TABLE IV

MEMORY USAGE - LOWER IS BETTER

| Sequence | ORB2(Original) | ORB2(opendlv) | Difference |
|---|---|---|---|
| 00 | 19.40% | 20.10% | **0.70%** |
| 01 | 10.50% | 7.30% | **-3.2%** |
| 02 | 24.50% | 23.60% | **-0.90%** |
| 03 | 4.30% | 4.00% | **-0.30%** |
| 04 | 3.80% | 3.20% | **-0.60%** |
| 05 | 11.10% | 11.90% | **0.80%** |
| 06 | 7.40% | 6.70% | **-0.70%** |
| 07 | 5.20% | 5.60% | **0.40%** |
| 08 | 11.10% | 10.30% | **-0.80%** |
| 09 | 6.90% | 8.00% | **1.10%** |
| 10 | 5.00% | 4.70% | **-0.30%** |
| average | **9.92%** % | **9.58%** | **-0.35%** |
| median | **7.40%** % | **7.30%** | **-0.30%** |

maximum reached level of 41,00% on dataset 1. The memory consumption is on average 9,58% with a maximum usage of 23,60% on dataset 2. The percentages are of course relative to the specification of the executing computer which in this case had a QUAD-CORE Intel i5-7600K CPU @ 3.80GHz and 16 GB DDR4 @ 2933MHZ memory. At the time of writing, this is a reasonable specification for customer-grade desktop computer, but there are hardware available in the market with much higher performance, for a low price.

## VI. DISCUSSION

In this section we will discuss the answers to our two research questions based on the results described in the previous section and relating it to existing work.

### A. Research Question 1:

Our first research question is: *To what extents can the ORB-SLAM2 algorithm be used as a visual SLAM component inside of the OpenDLV framework?*. In order to properly answer this question we have investigated both the resource consumption the ORB-SLAM2 algorithm generates when implemented inside of an OpenDLV microservice, as well as how accurate this implementation is compared to the original ORB-SLAM2 implementation.

As seen in the results table I, the accuracy of the OpenDLV implementation is worse than the original ORB-SLAM2 algorithm across all sequences. The difference is however quite small in the case of sequences where loop closing occurs, and a little bit higher in sequences where it doesn't occur. There could several reasons for these differences. The ORB-SLAM2 algorithm is a rather large and complex piece of software. It is possible that some minor mistake was made when implementing the OpenDLV adaption of the algorithm. The most notable difference between our implementation and the original is on the sequences that contain no loop-closing. The algorithm optimizes and recalculates the map and trajectories when a loop-closing occurs and on datasets that contain a loop closing we perform almost as well as the original implementation. On datasets without a loop closing event however, the difference from the original implementation is greater. This phenomenon is something we would have liked to investigate more, but we consider it out of the scope for this thesis. We do believe that it is a candidate for further research to investigate this as it is currently related to the largest difference in accuracy.

When it comes to computational performance we notice overall higher cpu usage when testing our implementation, compared to the original implementation. This is most likely related to the use of smart pointers[26] compared to the raw pointers used in the original implementation. Smart pointers introduces protection against memory leaks at the expense of requiring additional processing power. Overall our implementation still stays within the scope of consumer grade hardware which means that it can run in real time on reasonably priced hardware. Memory consumption is almost the same for both implementations with a slightly higher consumption by the original implementation. This could also be tied to the use of smart pointers versus raw pointers. We would have liked to investigate the performance further using a more specialized tool such as valgrind, but time constraints prevents us from including such evaluations in this thesis.

As mentioned by Bresson et al. the full SLAM problem can be difficult to handle in real time, and to be usable in an autonomous vehicle, a SLAM algorithm should preferably be able to run in real time. One of the features of the ORB2-SLAM algorithm[11] is it's use of the ORB(Oriented FAST and rotated BRIEF) technique[15], designed to run efficiently on standard CPUs, for fast and efficient feature extraction. ORB-SLAM2 performs several computations on structures that are derived from ORBs that are extracted from each frame, but central to the algorithms performance is the ORB extraction and its efficiency. Based on our results, the OpenDLV implementation seems to follow the low performance requirements of the original implementation.

Because of the similar accuracy to the original ORB-SLAM2 implementation along with the reasonable processing requirements, we argue that our implementation of the ORB-SLAM2 algorithm can be used as a visually-based SLAM component within the OpenDLV framework.

### B. Research Question 2:

Our second research question is: *To what extents can the ORB-SLAM2 algorithm be considered a replacement to a*

*LIDAR-based SLAM solution in the OpenDLV framework?*. We have investigated the accuracy of the ORB-SLAM2 OpenDLV implementation and how it compares to LIDAR-based SLAM implementations, in order to answer this question.

Based on the results, we can see that the OpenDLV implementation of ORB-SLAM2 performs worse overall compared to IMLS-SLAM. When compared to LOAM it performs better on some sequences and worse on others. In particular, all sequences without loop-closing, except sequence 10 the OpenDLV implementation shows worse translational drift than the LIDAR-based solutions. Worth discussing is also how well the original ORB-SLAM2 implementation compares to the LIDAR-based solutions. The original ORB-SLAM2 implementation outperforms LOAM but not IMLS on all sequences. This leads us to believe that the ORB-SLAM2 algorithm itself has the potential to be a replacement, or at least a complement to LIDAR-based SLAM in the OpenDLV framework, even though our implementation isn't a perfect representation of the possible accuracy. It is worth mentioning that neither visual nor LIDAR-based SLAM solutions can ever represent the "truth" when it comes to vehicle trajectory. They are techniques that both just represent an estimation of what the real trajectory is. In that regard the answer to research question 2 becomes dependent on if there ever is a "minimal" level of acceptable drift. Bresson et al. discusses the accuracy performance of SLAM solutions in autonomous driving and they argue that accuracy should be kept within 20 centimeters at all times[2]. With that value in mind, none of the solutions mentioned in this thesis achieves an average drift that low[11]. Bresson et al. further argues that it's necessary to have techniques for dealing with previous knowledge in the SLAM solutions that deals with the large scale environments encountered by autonomous vehicles. Loop closing is mentioned as one such technique.

Based on this reasoning we argue that an ORB-SLAM2-based SLAM implementation could work as a replacement to a LIDAR-based one in the OpenDLV framework. This conclusion is based on the fact that even though the accuracy isn't better than LIDAR-based solutions, it comes very close, with the potential to come even closer based on the comparison with the original ORB-SLAM2 implementation. It is further based on the fact that the ORB-SLAM2 algorithm contains the important loop closing feature identified in the previous research done on the subject.

## VII. Threats to validity

### A. Generalizability validity

One potential threat to generalizability validity is the fact that our results are limited to the use of the KITTI odometry dataset. The KITTI dataset is currently the main tool for evaluating the positioning accuracy according to Bresson et al.[2] and covers a wide range of traffic scenarios and road conditions. There might however be other variables or factors affecting the comparisons we make between SLAM solutions that are not captured by the KITTI dataset. For example, one possibility is the effect of different lightning conditions since LIDAR-based rangefinders are less light sensitive to their measuring than cameras are. We do believe that using the KITTI dataset is sufficient for the scope of this thesis but we acknowledge the need to reproduce our results using other datasets in the future.

Since one of the criteria when choosing comparable LIDAR-based SLAM solutions was that they had results for the KITTI dataset, this represents another threat to generalizability validity. There might be other, better suited solutions, that have not been validated using KITTI that we could have used for comparison instead. With the popularity of KITTI we do however argue that this potential threat is small.

Another potential validity threat we have found is that OpenDLV currently lacks an implementation of a LIDAR-based SLAM solution. This means that the comparison we perform in this thesis becomes more hypothetical than we would have wanted. It would have benefited our results if we could have had for example performance comparisons between LIDAR and visual SLAM solutions inside OpenDLV alongside the accuracy comparison, but we feel that for the scope of this thesis our current evaluation is enough.

### B. Construction validity

In order to minimize threats to construction validity we have designed our methodology so that we minimize the risk of our thesis having biased results. This is mainly achieved by using the KITTI benchmark and evaluation methodology when we compare SLAM solutions. We opted to use the KITTI benchmark because of its popularity both among visual and LIDAR-based SLAM solutions.

### C. Internal validity

We have strived to be as transparent as possible when describing our methodology that has led to the results we have presented in this thesis. By using methodology and tools based on previous research that are available to everyone we are confident that our results can be reproduced in a consistent manner by other students or researchers and that they are in fact not related to us personally. By publishing our finished software artifact as open source we also provide the means to exactly replicate everything done in this thesis by any interested party.

## VIII. Conclusion

In this thesis, we have investigated the possibility of using the ORB-SLAM2 algorithm as a visually-based SLAM solution inside the OpenDLV framework. We also investigated the potential the ORB-SLAM2 algorithm has as a replacement for LIDAR-based SLAM solutions in the context of OpenDLV. In order to perform this investigation, we conducted a DSR study with the goal of implementing the ORB-SLAM2 within the OpenDLV framework and we evaluated our software artifact using performance and accuracy metrics. Based on the results and our discussion we have concluded that the ORB-SLAM2 algorithm can be successfully used as a visually-based SLAM solution within the OpenDLV framework. We also concluded that the ORB-SLAM2 algorithm has the

potential to replace LIDAR-based SLAM within the OpenDLV framework.

During the evaluation and discussion around the results achieved in this thesis, we have identified several areas where there exists the potential for future research to be conducted.

First of all we have found that there is an opportunity to implement a LIDAR-based SLAM solution inside the OpenDLV framework, which can be used to further confirm or reject the results found in this thesis. Based on a study of prior research done on the subject such a LIDAR-based solution should include techniques similar to loop closing in order to properly conform to the requirements discussed by Bresson et al.[2]. Such an implementation could be used in conjunction with the software artifact produced by this thesis to further evaluate the SLAM area within the context of the OpenDLV framework.

Secondly we have identified different areas where the the research done in this thesis could be extended. Following on points made in the section on validity threats, the evaluation methodology used in this thesis should be applied to other datasets to address the the generalizability threat related to the use of only the KITTI dataset. With the inclusion of a LIDAR-based SLAM solution in the OpenDLV framework there is also the possibility of recording a new dataset containing exactly the parameters and variables of interest. With existence of an autonomous vehicle platform to record such a data collection at Chalmers Revere, we believe this to be an interesting basis for future work. Such a research project could also utilize the work done by Y. Hang and C. Berger for identifying the kind of data to collect[27].

Lastly we also suggest that research should be conducted by implementing a mapping feature inside of the OpenDLV framework, responsible for storing and loading maps to be used in conjunction with any SLAM solution LIDAR or visually-based. This seems like a logical next step in regards to SLAM functionality inside the OpenDLV framework.

## REFERENCES


[1] Permit holders (testing with a driver). https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/permit. Accessed: 2018-03-05.

[2] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[3] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[4] Velodyne cuts vlp-16 lidar price to $4k. https://www.spar3d.com/news/lidar/velodyne-cuts-vlp-16-lidar-price-4k/, Jan 2018. Accessed: 2018-03-24.

[5] Lidar costs $75,000 per car. if the price doesn't drop to a few hundred bucks, driverless cars won't go mass market. http://www.latimes.com/business/la-fi-hy-ouster-lidar-20171211-htmlstory.html, Dec 2017. Accessed: 2018-04-01.

[6] Stereo labs. https://www.stereolabs.com/. Accessed: 2018-04-01.

[7] J.-E. Deschaud. IMLS-SLAM: scan-to-model matching based on 3D data. *ArXiv e-prints*, February 2018.

[8] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.

[9] The kitti vision benchmark suite. http://www.cvlibs.net/datasets/kitti/eval_odometry.php. Accessed: 2018-03-15.

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

[11] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[13] Christian Berger, Bjornborg Nguyen, and Ola Benderius. Containerized development and microservices for self-driving vehicles: Experiences & best practices. pages 7–12. IEEE, April 2017.

[14] Gnu general public license. https://www.gnu.org/licenses/gpl-3.0.en.html, 2016.

[15] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[16] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[18] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[19] /proc. http://man7.org/linux/man-pages/man5/proc.5.html, September 2017. Accessed: 2018-04-15.

[20] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27(4):387, 2009.

[21] Chalmers revere orb-slam2. https://github.com/chalmers-revere/opendlv-perception-vision-orbslam2. Accessed: 2018-03-01.

[22] Chalmers revere vehicle viewer. https://github.com/chalmers-revere/opendlv-vehicle-view. Accessed: 2018-03-01.

[23] chrberger/libcluon. https://github.com/chrberger/libcluon. Accessed: 2018-03-01.

[24] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2017.

[25] Three.js javascript 3d library. https://threejs.org/. Accessed: 2018-03-24.

[26] Nicolai M Josuttis. *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012.

[27] Hang Yin and Christian Berger. When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–8. IEEE, 2017.