Utilizing TartanVO

COLIN LEYDON AND EYADO OCHO[*]

1 ABSTRACT

The final project of our semester focused on Deep Learning-based Tracking for Mobile Systems. We used the well-known computer vision framework, the Tartan VO model, to evaluate the practical consequences. Our main experiment was running the model against the large-scale KiTTi dataset, which is well-known for being used in the training of autonomous cars, drones, and robotics for autonomous navigation.

The goal was to evaluate the effectiveness of the Tartan VO model by assessing how well it performed in actual situations. In addition to providing practical experience with applying Deep Learning approaches for tracking mobile systems, the study offered insightful information about the potential benefits and application of advanced models like as Tartan VO for robotic systems and autonomous cars.

Additional Key Words and Phrases: Visual Odometry, Deep Learning, Operating System, Visual Simultaneous Localization and Mapping

ACM Reference Format:
Colin Leydon and Eyado ocho. 2023. Utilizing TartanVO. 1, 1 (December 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

GitHub https://github.com/ochoeyado/ECE-535-Deep-Learning-Project

2 INTRODUCTION

As robots and vehicles become more autonomous and are trained to seamlessly navigate complex environments it requires these systems to have the ability of precise and efficient navigation. This is where Visual SLAM (Simultaneous Localization and Mapping) comes into play which is a cutting-edge technology that combines computer vision and robotics to enable machines to navigate and understand their surroundings in real-time. Unlike Visual Odometry, Visual SLAM not only tracks the movement of a camera but simultaneously constructs and updates a map of the environment. This simultaneous localization and mapping capability make Visual SLAM invaluable in scenarios where the environment is dynamic or previously unexplored.

For instance, in autonomous vehicles, Visual SLAM helps in precisely determining the vehicle's location while constantly updating a map of the changing road environment. Similarly, in robotics, Visual SLAM allows robots to navigate through unfamiliar spaces, avoiding obstacles and efficiently completing tasks. Augmented reality applications also use Visual SLAM to connect virtual objects in the real world precisely and accurately, making the overall user experience better.

Imagine an autonomous delivery robot tasked with navigating and delivering through an urban environment to deliver packages. In this scenario, the use of visual odometry (VO) and visual SLAM plays a crucial role which enables the robot to perceive its surroundings, make informed decisions, and autonomously reach its destination.

As the robot moves along its path, equipped with one or more cameras, the visual odometry system continuously captures images of its surroundings. With analysis of consecutive image frames, the system can track key visual features, such as distinctive points or objects, and estimate the robot's relative motion in the environment.

As the robots moves forward, the visual odometry system can detect changes in the positions of visual features, understanding the motion of nearby objects. Through calculations and algorithms, the system is able to get the robot's movement—both in terms of translation (forward/backward, left/right) and rotation (turning).

This real-time estimation of the robot's position and orientation which is known as pose enables it to build a map of its surroundings while also creating a dynamic representation of the environment. As the robot encounters intersections or obstacles, the visual odometry system helps it adapt its path and make navigation decisions on-the-fly.

Unlike traditional, geometry-based models, TartanVO does not follow the traditional systems. It uses a vast amount of datasets to fuel its deep neural network, allowing it to estimate camera motion with a very good leap in accuracy compared to the existing geometry methods.

Our study aimed to put TartanVO to the test using the TartanAir Evaluator on popular datasets like Kitti and EuRoC. We charted its average trajectory error and relative pose error. Both of them being key metrics for measuring SLAM accuracy. Though we ran into troubles due to outdated code we were able to run the tartan air evaluator and do an analysis on the different datasets.

3 LITERATURE REVIEW

Our project centers around the paper titled "TartanVO: A Generalizable Learning-based VO" authored by Wenshan Wang, Yaoyu Hu, and Sebastian Scherer.

TartanVO is a pioneering model in learning-based visual odometry (VO) models, showcasing remarkable versatility across various data sets and real-world scenarios. The model is able to demonstrate capabilities over traditional geometry-based methods through the use of the TartanAir SLAM dataset, which has diverse synthetic data in complex environments, enabling the model to achieve generalization capabilities.

The group was inspired by the evolving trend of incorporating unsupervised learning, prompted by challenges in data collection and the noticeable limitations in existing models' generalization. They were also motivated by systems which had already achieved heightened accuracy with auxiliary outputs like depth and optical flow,

[*]Both authors contributed equally to this research.

Author's address: Colin Leydon and Eyado ocho, cleydon@umass.edu, eocho@umass.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM XXXX-XXXX/2023/12-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

so they began creating the TartanVO model —a solution addressing both the lack of generalization and performance challenges.

TartanVO's significant contributions are its generalizability across data sets and real-world scenarios, coupled with superior performance in challenging environments. Notably, the model's ability to learn from real-world synthetic data streamlines training, enabling the generation of diverse scenes. This, in turn, enhances its applicability across various scenarios and aids the devices it supports.

TartanVO is a leading model in visual odometry, offering a solution that not only tackles existing limitations but also sets new standards for generalization and performance in complex and dynamic environments.

4 SYSTEM DESIGN

Our journey with TartanVO focused on understanding its performance across diverse datasets and real-world scenarios. We worked on the software side and planned on running the model itself on our laptops for a more comprehensive evaluation. Luckily, Tartan is a relatively not hardware-intensive program and the only requirement to run it is a Cuda-enabled GPU which is offered on certain Nvidia graphics cards. However, due to some challenges that will be explained later in the paper, we had to go through an alternative approach which was analyzing the already pre-calculated trajectories generated by TartanVO against its own ground truth data.

While this was not the original plan, this method still offered valuable insights into TartanVO's capabilities. By feeding various datasets through the **Tartanair evaluator**, we could assess its accuracy and generalizability. Comparing these results with the ground truth data provided a clear picture of its performance strengths and weaknesses. This also showed the robustness of Tartan and its use on a diverse range of datasets.

This new approach, allowed us to explore crucial aspects of TartanVO:

- **Data set Performance**: We could analyze how TartanVO's accuracy varied across different datasets, namely EuRoC and KiTTi helping us identify its strengths and potential biases towards specific environments or scenarios.
- **Error Analysis:** Evaluating the trajectories against ground truth data allowed us to get specific areas where TartanVO struggled and which dataset it would work better with.

While running the model directly on our laptops would have provided a more complete picture, this alternative path was still able to get us significant findings. Understanding how TartanVO behaves with different datasets and comparing its performance to its own ground truth is good enough to lay a valuable foundation for further research and improvement.

The next section will go deeper into the challenges that made us go through this approach and explore the specific findings from our analysis of TartanVO's performance.

5 CHALLENGES WITH IMPLEMENTATION

When attempting to run TartanVo we ran into a variety of challenges that all required a unique solution. Some of these challenges prevented us from running the model as intended forcing us to take a mathematical approach.

5.1 Docker

On the GitHub there was an explicit section about utilizing TartanVO on Docker, a container application that allows you to run programs outside of your operating system. Our first attempt in running Tartan was through this method as the "ReadMe" proposed this to be a quick and simple way to get started, this proved to be the contrary. This was both of our first times trying to use Docker and were a little confused about the process. Following the steps was unfruitful as attempting to get the Docker image did not return anything. We then saw through our research that you can search prebuilt Docker images through Docker itself. When searching this we appeared to find the correct image as it was the same name listed on the GitHub, but upon attempting to run the image nothing would happen. No matter what we attempted when executing the image, Docker would exit within a couple of seconds and there would be no form of output. We attempted various fixes but there seemed to be no remedy to the situation. It was for this reason we had to try other methods of attaining Tartan and its dependencies.

5.2 Boot-able Flash-drive

Seeing as we were unable to make Docker work we were forced to use the method of manual installation through Ubuntu. The problem with this was that neither of our machines was Ubuntu-based. Based on past experiences we knew that we could create a boot drive on a flash drive that is capable of running another operating system on your machine. In order to do this we installed an Ubuntu image and Rufus, which is an image-writing program for Windows. This allowed us to successfully boot our machine into Ubuntu and begin our attempts at running Tartan.

However, this resulted in a problem that we had overlooked, when running an operating system off of a flash drive you are given two options. These options consist of trying the OS or installing the OS. If you select the 'install' option then this operating system will write over your current OS partition on your hard drive in an irreversible manner. We knew this was not an option for us as we needed our computers for other activities, which left us with the option of 'try'. This option works perfectly for practicing in an Ubuntu environment since you can not cause damage to your own system as Ubuntu is self-contained on the flash drive. Unfortunately though, by using 'try' every time you launch Ubuntu, it is essentially a first-time set-up of the OS. The reason this is so crucial is because it meant nothing you did was saved each time you shut down your laptop.

The solution to this we found was utilizing a virtual machine(VM). Virtual machines are interesting as they allow you to essentially run an entirely new computer within your computer. As these virtual machines are self-contained there is no effect on your computer regardless of what you do inside one. We began by installing Virtual Box by Oracle to host our VM as that seemed to be the most commonly used host. We then went about installing the same Ubuntu image from before to this VM. This took a few attempts as we were new to VMs and did not fully understand how the allocation of resources worked until after further research. Upon successfully installing Ubuntu to the VM and booting it up we could begin our work on TartanVO and implementing it with our datasets.

5.3 Outdated Code

The primary work done on TartanVO was done around 2017 were most of the files were last modified. This can be seen both on GitHub and when viewing the Docker files provided by the research group. When attempting to run these files you will immediately be met with issues as the code was written in Python 2 rather than the now common Python 3. This means that when viewing these files the syntax will be completely different, causing the machine to not be able to interpret the files. We first discovered this issue after viewing the GitHub repository where the file syntax was clearly not the Python we have become accustomed to, when viewing the "ReadMe" files it became clear that Python 2 was listed in the needed software and libraries. The GitHub also linked to a variation that was meant for Python 3 but this also failed to run appropriately with the provided files.

There were also issues getting some of the necessary libraries. Upon viewing although CuPy is still being updated, it seems to be losing support as our research showed most people are switching to Nvidia Container Toolkit to utilize the Cuda ability on certain General Processing Units(GPU). When we were able to successfully get CuPy installed it seemed that the commands in the provided files were no longer up to date with the current version of the library, CuPy 12.3.0. It seems that Nvidia is moving from CuPy, RAPIDs, and cuDF towards one singular library controlled by them the Container Toolkit. We attempted to install this new library, but doing this would mean rewriting the code in a way that would work, but this exceed our knowledge of Tartan and of the Container Toolkit.

5.4 Externally-Managed-Enviorment

Upon our attempts to run the model in Linux, we continuously ran into an error message:" Externally-managed-environment". In an attempt to circumvent this, we tried using sudo installs, pip install, and pipx installs yet none of these solutions worked either the command would receive a new error message, or would still give the externally-managed environment. We attempted to look at solutions to this problem but most seemed to be out of our control. The main solution proposed by the terminal was to reach out to our operating system(OS) provider and have them clear the installation. As this was one of our first times using the Ubuntu OS we were unfamiliar with whether or not this was commonplace. We were also worried about attempting to force a system install, as our research online showed that if there were errors in the installation, it could break the entire operating system if it changes some necessary values in the system.

There was also another tip to try and use the pipx command rather than a standard pip command. The use case of pipx is to install commands for your scripts, rather than dependencies for your code. This is why we received an error as we were attempting to install the KiTTi tests which are in fact dependencies, not commands. We also attempted to utilize the sudo apt install but struggled to find a viable link that way. Pip is a command that can search the web for the packages you are attempting to install, as well as make sure you get the most up-to-date package, which is why it is such a useful command. When using the sudo method you are attempting to install directly to root with a known packet location from the internet. We attempted to modify the pip command, but doing this led to no packets being found as it was not a direct link to the package and version.

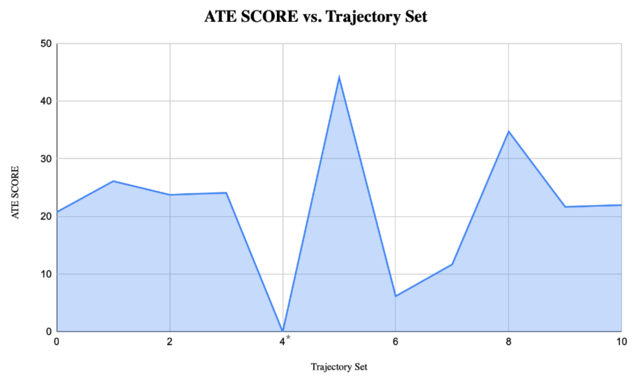


Fig. 1. Results of our KiTTi Tests

5.5 Out of Bounds

In order to address these issues and have a running version of our model we were forced to use an archive of these datasets and run the Tartan Evaluator Base to receive an output. This also came with its own set of issues. We were forced to modify some of the functions in the code as the libraries had since been updated and required new commands that would have a similar output. This included but was not limited to the shape() command and the as_dcm command which was updated to as_matrix. In order to figure out this error we had to add special error messages in the code to figure out where the error occurred. When we realized that this was the stem of our problems we moved forward onto our issue with the shape() command.

The model runs by comparing the ground_truth file to the ground_estimate_file. The evaluator file passes these to the base where they can then be properly compared, but an issue will arise if these matrices do not have the exact same coordinates. Luckily this was not an issue with the test files given, but this was a problem when running the KiTTi and EuRoC tests. These files varied in length some being only 400 lines were others were thousands of lines. Fortunately, these files all had the same number of columns, so the only thing that needed to be edited was their row quantities. We had to base all the lengths on our ground truth file which contained 734 lines. By manually adapting each file we were able to circumvent the error raised by the shape() function.

6 OUR IMPLEMENTATION

6.1 Average Trajectory Error

Due to the issues addressed above we were unable to run the entire model and were forced to take a mathematical approach. On running the model we were given a series of outputs including average trajectory error(ATE), relative pose error(RPE), and a matrix output. For our presentation of data, we graphed our trials against the average trajectory error versus the trial number. You can see our

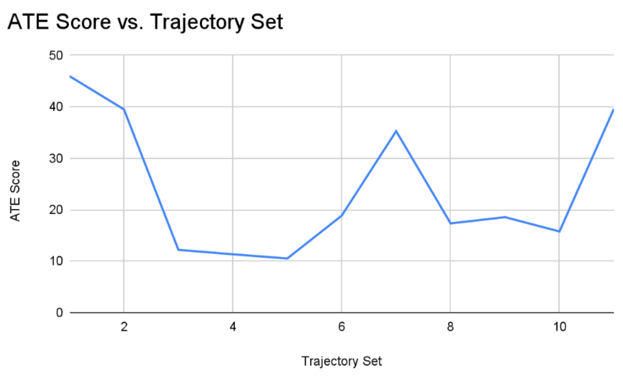


Fig. 2. Results of our EuRoC Test

results for the KiTTi test in Figure 1 and the EuRoC test in Figure 2. To calculate the ATE score you use the equation:

$$\text{ATE} = \frac{1}{N}\sum_{i=1}^{N}\left\|\text{pose\_ground\_truth}_i - \text{pose\_estimated}_i\right\|$$

Where 'N' is the total number of time stamps. Which was implemented in the code. The ATE score describes the average discrepancies between two trajectories in this case represented by paths. We believe that this was an accurate representation of the model and how the data can be interpreted.

6.2 Relative Pose Error

By running this code we also received the relative pose error, which is the difference in position and orientation of two trajectories. RPE is calculated by:

$$\text{RPE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left\|\text{pose\_ground\_truth}_i^{-1}\cdot\text{pose\_estimated}_i\right\|^2}$$

Where 'N' is the total number of timestamps. Similarly to our evaluation of ATE we graphed our results in RPE vs trial number, as seen in figures 3 through 6. RPE is a little different from ATE however as we are given two values the rotation error as well as the translation error.

- **Rotation Error:** refers to the difference in rotation between our ground estimate file and our ground_truth file often expressed in radians.
- **Translation Error:** refers to the difference in translation between our ground estimate file and our ground_truth file often expressed in meters.

We elected to graph the values for rotation and translation separately as we felt it was important to highlight each of these results.

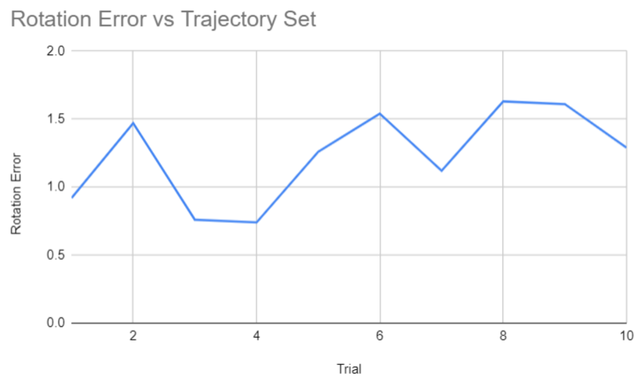


Fig. 3. EuRoC Rotation Error

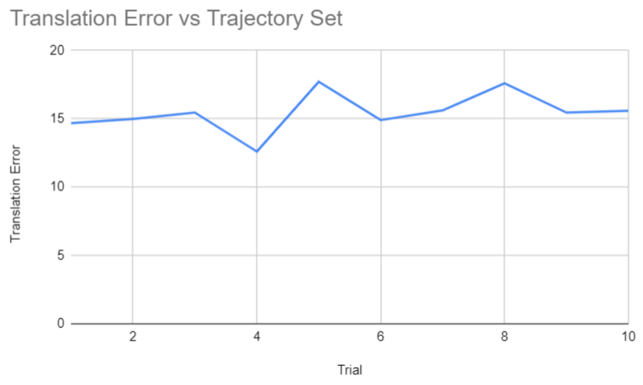


Fig. 4. EuRoC Translation Error

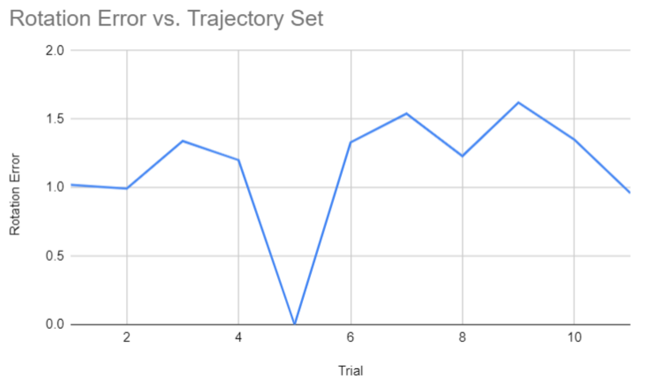


Fig. 5. KiTTi Rotation Error

7 EVALUATION

In the world of visual odometry (VO) and Simultaneous Localization and Mapping (SLAM), **Absolute Trajectory Score (ATE)** and **Relative Pose Error (RPE)** are two crucial metrics used to evaluate the accuracy and performance of these systems.

7.1 Absolute Trajectory Score(ATE)

The ATE score is how close the estimated path of the model is to the actual path the system took. It is like the overall distance between the planned route and the actual route driven. It is the average deviation of the estimated trajectory from the ground truth

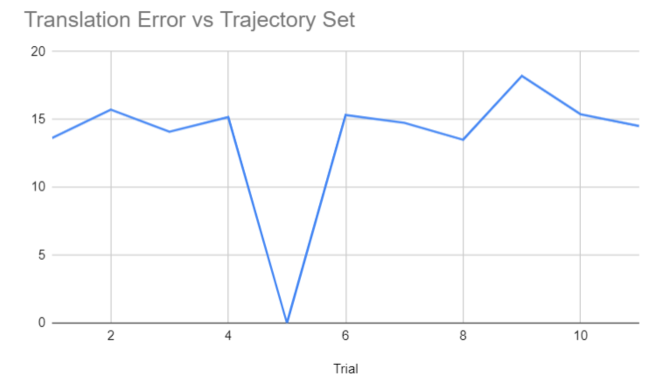
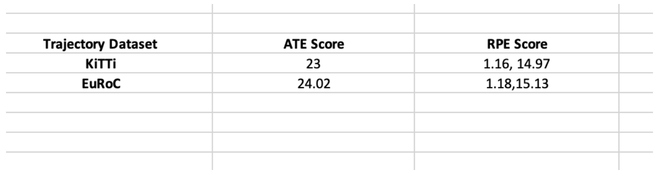


Fig. 6. KiTTi Translation Error



Trajectory Dataset	ATE Score	RPE Score
KiTTi	23	1.16, 14.97
EuRoC	24.02	1.18,15.13

Fig. 7. EuRoC vs KiTTi

trajectory. The average ATE score of TartanVO for the KiTTi dataset is 23.43 and using a scale of 0.3 it means the model has an average error of 0.7 meters per frame when considering the cumulative effect across the sequence. Compared to other established model such as ORB-SLAM3 and LOAM (Lidar Odometry and Mapping) which have ATE scores of 0.85 and 0.55 respectively it shows that TartanVO is a good model to use for deep learning tracking for mobile systems.

7.2 Relative Pose Error

This is how accurately the system estimates the pose (position and orientation) of the camera at each frame, compared to the previous frame. It is the measure of the average difference between the estimated and actual poses of the camera at each frame, relative to the previous frame. The Relative Pose of TartanVO was given in the rotation and translation errors and the average when run against the KiTTi dataset was 1.13 in the rotation error and 14.97 in the translation error this indicates that TartanVO's estimates of the camera's position and orientation between consecutive frames have an average error of 1.13 radians in the rotation error and 14.97 meters in the translation error. Similarly when running the ErUco tests rotion error equated to 1.18 radians on average, while the translation error averaged to 15.13 meters.

7.3 Comparison between EuRoC and KiTTi datasets

As said before the EuRoC and KiTTi datasets are two datasets that contain data to check for the effectiveness of a visual odometry model. The table above shows the ATE score and RPE score of the EuRoC and the KiTTi datasets run on the TartanVO model. The table shows that TartanVO performs slightly better on the KiTTi dataset having a smaller ATE score than when the EuRoC datasets are run.

7.4 Trade offs between ATE and RPE score

While both ATE and RPE are important to understanding a visual odometry model, there can be a trade-off between them. A system with a very low RPE might not necessarily have a low ATE, with a low RPE this model will have accurate measurements at each time-step but if this model has a high ATE then there will be small changes over time that will drastically skew the results. While a system with a low ATE might not have a low RPE. In this case, when viewing the results globally you will end with an accurate trajectory to the ground truth but when looking piece by piece you may notice large errors at various time-steps. All in all, it is very important when working with visual odometry to minimize errors in all forms, especially the ATE and RPE. Though depending on your use case it may be more important to minimize one form of error over the other. If you are worried about errors at each individual step then it is important to mitigate RPE. If you are more concerned with error over the entire event then focus on reducing ATE.

8 CONCLUSION

In this paper, we used TartanVO to check into our project which was Deep Learning in Mobile Tracking Systems. By testing the model with two prominent datasets in the field of visual odometry, to evaluate how good the system is in real-world scenarios. The outcomes of our experiments show that TartanVO's is a good and robust model for tracking systems. This discovery opens up for future research, such as including the model's overall accuracy and robustness and also there is an opportunity to extend the capabilities beyond pose estimation, which is getting a deeper understanding of objects in the environment and their attributes.

TartanVO demonstrated a very good performance across various datasets, highlighting its adaptability and its generalizing ability. This not only underscores the model's current capabilities but also positions it as a promising answer for the future of autonomous mobile tracking systems. As we continue to go through our findings, we can see that TartanVO and similar learning-based models represent a a huge shift in already existing models.

ACKNOWLEDGMENTS

To Professor Anwar Fatima, for explaining and helping us understand how to read, understand, and critique research papers.

To our teaching assistant Yasra Chandio for clearing up confusion and providing the necessary archive to run TartanVO.

REFERENCES

[1] Wang, W., Hu, Y., & Scherer, S. (2022). *TartanVO: A generalizable learning-based VO.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 13049-13058). Retrieved from https://arxiv.org/abs/2011.00359

[2] Wikipedia contributors. (2023, January 14). *Visual odometry.* In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Visual_odometry

[3] Computer Vision Group, TUM School of Computation, Information and Technology, Technical University of Munich. *Visual SLAM.* Retrieved from https://cvg.cit.tum.de/research/vslam

[4] Zhao, X., Agrawal, H., Batra, D., & Schwing, A. G. (2021). *The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation.* In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 16127-16136).

[5] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). *ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics*, 31(5), 1147-1163.

[6] Zhang, J., & Singh, S. (2014, July). *LOAM: Lidar odometry and mapping in real-time.* In *Robotics: Science and Systems* (Vol. 2, No. 9, pp. 1-9).