ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ROBOTICS &
PERCEPTION
GROUP

Mini Project Report

# Vision Algorithms for Mobile Robotics

**Autumn Term 2021**

**Authors:**
Elias Asikainen
Robin Frauenfelder
Pascal Lieberherr
Silvan Löw

# Contents

# Chapter 1

# Introduction

This report summarizes the overall work for the mini project of the course *Vision Algorithms for Mobile Robotics*. It presents how the visual odometry pipeline was implemented including the milestones that we achieved, how we tuned the parameters of the different parts and, last but not least, the results for the different data sets. As an additional feature, we recorded our own data set on Rosie, an autonomous robot designed for weeding on sugar beet fields. The robot was developed by the focus project Rowesys at the Robotic Systems Lab (RSL).

# Chapter 2

# Implementation

We implement the visual odometry pipeline using *MatLab*. In our implementation, we make use of the following toolboxes: *Computer Vision Toolbox*, *Image Processing Toolbox*, *Statistics and Machine Learning Toolbox*. The exact functions are listed in appendix A.

## 2.1 Initialization

The goal of the initialization is to get the initial pose estimate of the camera based on two frames. Particularly, we want $T_{C0C1}$, which is the transformation matrix from camera frame 1 to camera frame 0. This is achieved by extracting corresponding features of two sufficiently distant frames. Additionally, the corresponding features and their respective landmarks are used to initialize the continuous operation. Below, the main steps of the initialization are described together with the most important observations and findings.

- **Detect key points** In the first step we detect the key points by extracting Harris features. This happens for each of the two sufficiently distant frames. Here we rely on the Matlab function *detectHarrisFeatures()*. After the Harris Features are detected, the best features have to be extracted. At first try, we did this by selecting the strongest 200. A Harris feature is strong when the Harris corner response function for the feature has a high value. However, we achieve are more precise initial pose estimation $T_{C0C1}$ by selecting 200 uniformly distributed Harris features. Especially the z dimension of the translation vector ($t_z$) in $T_{C0C1}$ is improved a lot. The 200 strongest ones usually were not really distributed all across the image and most of them were also at a similar depth. We think that this lead to a bad estimate of $t_z$.

- **Describe key points** Here, each keypoint gets a descriptor which describes the keypoint. We used the Matlab function extractFeature() with the "block method". This means the descriptor of each keypoint is a patch of the pixels around the keypoint.

- **Match keypoints** After each keypoint has a descriptor in the two images, they need to be matched. The keypoints are matched with KLT.

- **Relative pose** Now we have two frames with corresponding keypoints. This allows us to estimate the fundamental matrix F. Note that the fundamental Matrix encodes $T_{C0C1}$. F is obtained using the Matlab function *estimateFundamentalMatrix()* and *relativeCameraPose()* is used to obtain $T_{C0C1}$ from F.

- **Triangulation** Having corresponding keypoints and $T_{C0C1}$, we are now able to triangulate the landmarks. We used the Matlab function *triangulate()*.
  Now we have the keypoints (2D), the landmarks (3D) and the initial pose with $T_{C1C2}$. This is all we need to start with the continuous operation.
  Since Matlab uses different conventions for the projection equation it took us some time to get familiar with it. In general, getting familiar with the Matlab conventions was the drawback of making use of Matlab functions instead of the lecture exercise functions. We go for Matlab function because we think they are better in performance.

## 2.2 Continuous Operation

During the continuous operation, we want to track the existing landmarks, add new landmarks and use those to find the pose of the camera. This is done by repeatedly calling the following three functions:

- **Process frame** The *processFrame* function gets called once a new frame appears. It first calls the *estimatePose* function and the *triangulateNewLandmarks* function. In the end, it removes impossible landmarks with a negative Z direction in the camera frame.

- **Estimate pose** The *estimatePose* function first tracks the existing keypoints with KLT and removes all non-tracked keypoints and landmarks. After that, it performs RANSAC, where it removes all outliers from the tracked keypoints and landmarks and simultaneously estimates the pose of the camera.

  The most important tuning parameters in this function are the parameters of the RANSAC, since they are not only a filter for the qualiy of our tracked keypoints and landmarks, but RANSAC is also responsible for a accurate estimation of the pose of the camera, with which new landmarks will get triangulated. So if the estimation of the camera pose is bad, all new keypoints will be wrongly triangulated and lead to an even worse estimation of the camera pose.

- **Triangulate new landmarks** Initial landmarks will disappear from the camera's field of view once the vehicle has moved too far. An accurate pose estimate can only be maintained if the VO Pipeline keeps triangulating new landmarks.

  In our implementation, we mainly follow the recommendations of the project statement. Each new frame, we check for new potential landmarks by detecting characteristic points using the Harris corner detector. These characteristic points are then tracked over subsequent frames. Every iteration, the 3D locations of those candidate landmarks are triangulated using the vehicle's current pose and the pose in which they have first been detected. Once this triangulation becomes accurate enough (verified by checking the bearing angle of the intersecting rays), these candidates are added to the regular landmarks used to determine the vehicle's pose.

  To prevent the routine from selecting candidates that exist already either as candidate or proper keypoints, we check the distance of new Harris corners to already existing keypoints. Only if their distance exceeds a certain threshold, they are considered in future iterations.

  Further, we try to select candidate keypoints that are as uniformly distributed over the image as possible, making the eventual pose estimate more accurate. However, we also include exceptionally strong corners as candidate keypoints

even if they are close together, since they can be tracked accurately and might result in an exact triangulation.

The most important parameters to be tuned in this part of the code are the bearing angle at which new landmarks are considered to be triangulated accurately, the distance candidate keypoints need to have from existing ones, as well as the ratio between strong Harris corners and uniformly distributed corners that are added to the candidate list.

## 2.3 Interesting Additional Feature

Our interesting additional feature is the VO pipeline running on our own recorded data set. The data set has been recorded on an agricultural autonomous mobile robot called Rosie, which is designed to autonomously remove the weed in a sugar beet field. The robot is equipped with a RealSense T265 stereo camera. Even though it is a stereo camera, we only used the images of one camera, since our VO pipeline is monocular. In fig. 2.1 the camera intrinsics and the distortion model parameters are shown. The Kannala-Brandt distortion model applies to the fish-eye lens of the T265. With the self written python script *undistFisheye.py*, we undistorted the images using the OpenCV function *cv2.fisheye.undistortImage()*. After we undistorted the images they could be fed into the VO pipeline. The camera did not need to be calibrated since its precalibration can be assumed to be accurate enough [1].

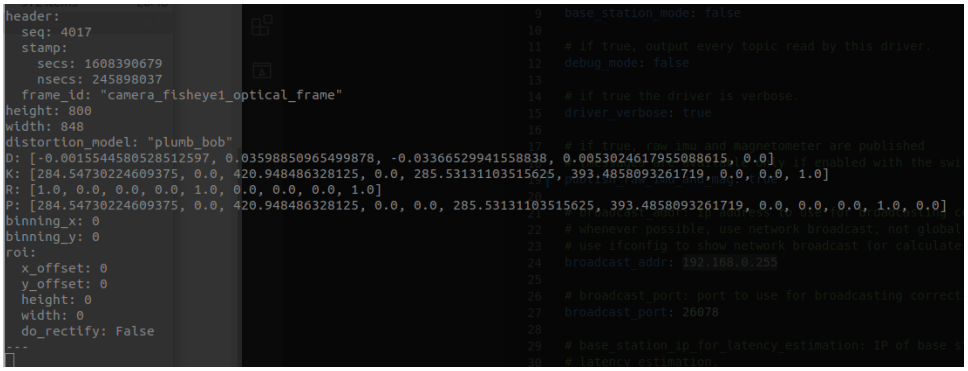


Figure 2.1: K: Camera intrinsic, D: Distortion parameters for Kannala-Brandt distortion model

[1] https://github.com/IntelRealSense/librealsense/issues/4247

Figure 2.2: The autonomous mobile robot Rosie

# Chapter 3

# Results

## 3.1 Parking Data Set

The parking data set was the easiest data set of all. For that reason, we used it primarily to test if our visual odometry pipeline was working correctly. The camera is moving in a straight line, the lightening is consistent and there are no moving object in the frame.
The most challenging part of the data set was the fact that sometimes cars pass by pretty close to the camera, which means that many landmarks are hidden and new landmarks need to be added quickly. Other than that, the trajectory is very simple. This leads to the fact that errors are easy to spot. Overall, tuning of the important parameters leads to a pretty good performance of our pipeline locally as well as globally for this data set.
The results (and the video) were obtained on a 2019 Macbook Pro running Mac OS Big Sur with a 2.4 GHz 8-Core Intel Core i9 processor and 32 GB of RAM.

## 3.2 KITTI Data Set

The KITTI data set includes multiple different scenarios and is the largest of the four that was used. It was recorded from a moving platform (VW station wagon) using high quality grayscale cameras by driving in Karlsruhe. It records various scenarios of street and traffic. However, the part of KITTI that we used only includes driving on small city streets with only little traffic.
Since it is the largest of our data sets with varying conditions it also is the most challenging for a visual odometry pipeline. There are varying lighting conditions and the movement of the platform is not always constant. Sometimes, it comes to a full stop. There are also some moving components in the images, like pedestrians and cyclists.
Since this is the only data set where the platform stops for a few frames, this caused an additional challenge for our visual odometry pipeline. The stop, which happens just before the fourth curve, caused our pipeline to crash, since it found multiple possible solutions for the triangulation in between the two stationary frames. To overcome this issue we reused our old triangulation results, if multiple solutions got found, and thus could fix this issue from occurring. With this fix we still have minor inaccuracies in scenarios like this. Overall the performance of our pipeline on this data set is good and from a global perspective we see that the trajectory overlaps when the same street is driven twice with a small error margin. Our pipeline also handles the few moving objects that are passed in this data set well and there are only slight to no inaccuracies caused by them.

The results for the KITTI data set where generated on a 2015 Macbook Pro running Mac OS Big Sur with 3.1 GHz Dual-Core Intel Core i7 processor and 16 GB of RAM.

## 3.3 Malaga Data Set

The Malaga data set was recorded on a public road in the Spanish coastal city. Pose estimation turns out to be relatively easy on the straights, since there is lots of distinctive structure containing features that can easily be tracked. There is also only moderate traffic, which means that there are not many moving cars could distract the pipeline.
The challenging parts of the data set are the two 180 degree curves. Further, the limited dynamic range of the camera can be a problem in the second curve, where there is significant glare. Additionally, large amounts of drift could become especially obvious on this data set, as the vehicle passes the same street segments twice.
Our visual odometry pipeline performs very well on this data set and estimates the vehicle's pose in a plausible way at all times. The two roads that are passed by the vehicle appear very parallel on the estimated trajectory and the curves have constant curvatures in the two roundabouts. The loop is closed well, only the distance on the second straight is estimated to be lower than the first one, which becomes obvious when the car reenters the first straight early. When the first straight is driven down the second time, the trajectory starts to differ a bit angular wise.
Minor discontinuities (where the trajectory is not completely smooth) can only rarely be found. The biggest problems arise in the the part where the sun shines straight into the camera, but they never become significant enough to hurt the global trajectory in any way. The results (and the video) were obtained on a Lenovo ThinkPad 490 running Ubuntu 18.04 with an Intel Core i7 4-Core and 24 GB of RAM.

## 3.4 Rosie Data Set (Interesting Additional Feature)

The Rosie data set was recorded in the garage of the ETH building called CLA. There the robot was steered with a game pad along the aisle including a 90° turn at the beginning.
When we first recorded the data set the wheels of the robot were in the field of view of the camera. Please note that due to the fish-eye lens the field of view was around 180°. We replaced the camera such that there are no parts of the robot in the field of view of the camera. Another way would have been to set a ROI (region of interest) in our VO pipeline such that the wheels are outside of the ROI.
The main challenge with the Rosie data set was the low texture of the scene. As one can see the floor and the walls are made out of concrete and only a few objects like cars, containers and doors added some texture to the scene. Therefore, one can observe that the detected features often change from frame to frame which makes it hard to reliably track many features from frame to frame.
The pose estimation for the straight path after the 90° turn at the beginning is locally and globally accurate. Whereas the turn at the beginning imposes difficulties for the local pose estimation. Namely, little jumps occur at the beginning of the turn. We think that this is due to the low texture and the fact that it's hard to reliably track features. Nevertheless, the local jumps are small such that the global curve can be estimated well.

The results for the Rosie data set where generated on a Lenovo ThinkPad P1 running Ubuntu 18 with a 2.6GHz 6-Core Intel i7 processor and 16GB of RAM.

## 3.5 Video recordings

The video recordings for each data set are available under the following links:

- Parking: `https://youtu.be/ZszFLWTXbAg`
- KITTI: `https://youtu.be/yq8YdIr0Iig`
- MALAGA: `https://youtu.be/hNJwbfJbpI4`
- Rosie: `https://youtu.be/-2u6_rH_cnc`

# Appendix A

# Appendix

## A.1 Used MatLab Functions

- `vision.PointTracker`
- `estimateFundamentalMatrix`
- `relativeCameraPose`
- `cameraPoseToExtrinsics`
- `cameraMatrix`
- `triangulate`
- `detectHarrisFeatures`
- `selectStrongest`
- `selectUniform`

## A.2 Used Python Functions

- `cv2.fisheye.undistortImage()`