

UKRAINIAN CATHOLIC UNIVERSITY

FACULTY OF APPLIED SCIENCES

COMPUTER SCIENCE PROGRAMME

---

# Simultaneous Localization and Mapping

Linear Algebra final project report

---

*Authors:*

Bohdan HLOVATSKYI, Mykhailo PASICHNYK, Stefan-Yuriy  
MALYK

May 2022



APPLIED  
SCIENCES  
FACULTY ●

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Problem setting</b>	<b>3</b>
2.1	Problem formulation . . . . .	3
2.2	Camera model . . . . .	3
2.3	Feature detection . . . . .	5
2.4	Harris Corner Detector . . . . .	6
2.5	BRIEF . . . . .	7
2.6	Epipolar geometry . . . . .	7
2.7	RANSAC . . . . .	8
2.8	Triangulation . . . . .	9
<b>3</b>	<b>Overview of approaches</b>	<b>10</b>
<b>4</b>	<b>Implementation</b>	<b>12</b>
<b>5</b>	<b>Experiments</b>	<b>13</b>
<b>6</b>	<b>Limitations</b>	<b>14</b>
<b>7</b>	<b>Conclusions</b>	<b>15</b>

## Abstract

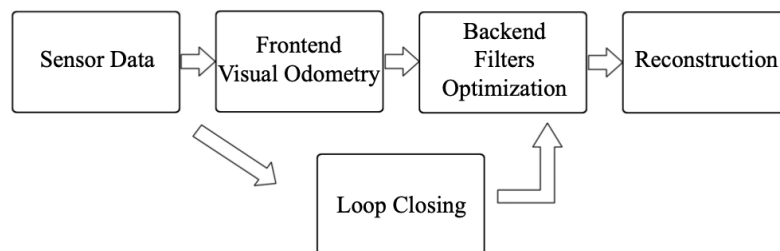
SLAM or Simultaneous Localization and Mapping - the problem of constructing a map of an unknown environment, simultaneously tracking the object's location. SLAM algorithms utilize the data obtained from sensors, such as video camera, LIDAR etc to construct a 3-D model of the nearby objects and estimate the relative location of the tracking device itself. They are used in variety of fields: autonomous driving, robotics, augmented reality and much more. The algorithms that rely on video data are referred to as Visual SLAM. They obtain information by means of Computer Vision and different techniques of Signal Processing. In the following work we review available approaches along with the implementation of the chosen one by means of Python and as our Linear Algebra course work. Source code: [1].

## 1 Introduction

Visual SLAM is a family of algorithms that solve the problem of Simultaneous Localization and Mapping by means of Computer Vision. Depending on the type of input video stream there are different types of Visual SLAM:

- Monocular SLAM. Such SLAM takes as an input video stream from a single RGB camera. The simplicity of the setup results in low costs, although implies some limitations, one being particularly significant. Since a single frame is essentially a projection of a real-world scene to an image plane, we can construct environment map only to some scale factor. In other words, we cannot obtain the exact depth of featured points in an environment from a single image source, only approximate one using multiple images and translation movement.
- Stereo SLAM. Input video stream comes from two cameras. Cameras are positioned on some distance, called baseline, value of which is known beforehand. This allows to get the more accurate depth of the featured points. The limitation is that calibration and computation process is difficult usually requiring additional hardware to generate real-time outputs.
- RGBD. Specific kind of SLAM that gets the input data from RGBD camera - the one that included depth of objects within an image. The depth is computed by emitting light and calculating how much does it took for it to return back. However, this technology is relatively new. As a result the range is limited making it not suitable for outdoor use and price is higher, compared to approaches described above.

The pipeline of SLAM algorithms will usually look in the following way:



In this work we are mainly interested in Frontend Visual Odometry, goal of which is to track the object that produces the source data and build non-optimized 3-D cloud map of the environment. Backend optimization and loop closing are powerful techniques to optimize the SLAM and get rid of the noise that brings a lot of inaccuracy in the SLAM algorithm. Those algorithm, such as Kalman and Particle filters, are beyond the scope of Computer vision and are related mainly to signal processing, thus won't be focused on here.

## 2 Problem setting

### 2.1 Problem formulation

With the given setting, we are interested mainly in two parts:

1. Modeling the object's motion. In an abstract setting this can be modeled as  $x_k = f(x_{k-1}, u_k, w_k)$ , where  $x_k$  denotes a current position of an object,  $x_{k-1}$  - its previous position,  $u_k$  - change of its state and  $w_k$  - noise.
2. Gather information about the landmarks of a nearby environment. The observed data  $z_{k,j}$  will then be described by this equation:  $z_{k,j} = f(y_j, x_k, v_{k,j})$ . Here  $y_j$  - landmark point,  $x_k$  - its coordinates,  $v_{k,j}$  - noise of the observation.

These two equation summarize the SLAM. For the Euclidean space the motion model can be expressed as the so-called special Euclidean Group:

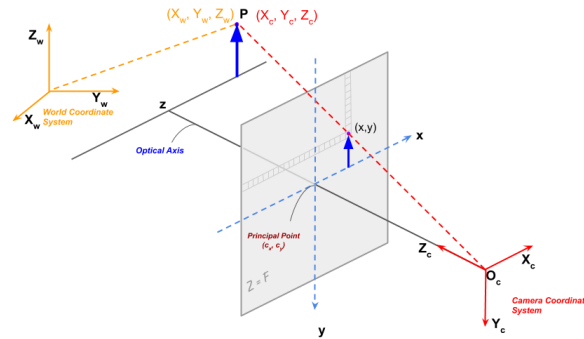
$$SE(3) = \{T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \in \mathbf{R}^{4 \times 4}\},$$

where  $R$  - rotation matrix,  $t$  - translation vector. It allows to express state  $a_1$  as

$$a_1 = Ra_0 + t.$$

Important note here is that we associate the state of an object with the location of camera that produces some input data.

### 2.2 Camera model



Any point in space can be represented by three coordinate systems:

- The World Coordinate System - point  $\mathbf{P} = (X_w, Y_w, Z_w)$ . For example if our object is in a room we can treat floor as  $x - y$  axis and a wall as  $z$  axis

- The Camera Coordinate System -  $(X_c, Y_c, Z_c)$  the same object in 3D space but from camera, where  $X_c$  points to the right,  $Y_c$  down and  $Z_c$  towards the camera
- The point projection to the plane (image itself)

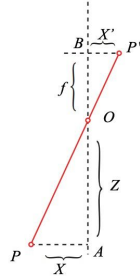
The world and camera coordinate systems are related by the rotation matrix  $\mathbf{R}$  and translation vector  $\mathbf{t}$ . Thus

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathbf{t}$$

We can rewrite statement above to make it more compact and call such  $[\mathbf{R}|\mathbf{t}]$  matrix the **Extrinsic matrix**

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Now that we have our point  $\mathbf{P}$  in the camera coordinate system we can project it onto the plane.



Our projected point  $(x, y)$  can be found using following equations:

$$x = f \frac{X_c}{Z_c}, y = f \frac{Y_c}{Z_c}$$

where  $f$  is the focal length of the camera. One important thing to mention is that the image  $\mathbf{P}$  going through camera's optical center would be inverted. However, since usage of such camera would be inconvenient camera's software performs some transformation. Thus, this step could be omitted and we can stick to the equations above. Next we construct such a matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} Y_c \\ X_c \\ Z_c \end{bmatrix}$$

It would be referred as the **Intrinsic Matrix**. Another important observation to make is that the optical center of the camera could be shifted from the origin of the camera coordinate system. Thus we need to modify the Intrinsic matrix as such:

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & f \end{bmatrix}$$

Combining all of the above we can get coordinates of the object on our image  $(u, v)$ :

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} Y_c \\ X_c \\ Z_c \end{bmatrix}$$

where

$$u = \frac{u'}{w'}, v = \frac{v'}{w'}$$

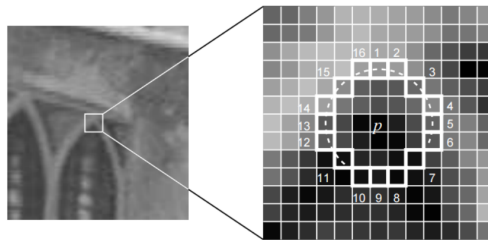
## 2.3 Feature detection

Firstly let define what a feature is. A feature is a some pattern in the image which has the following proprieties:

1. **Repeatability**: The same feature can be found in different images.
2. **Distinctiveness**: Different features have different expressions.
3. **Efficiency**: In the same image, the number of feature points should be far smaller than the number of pixels.
4. **Locality**: The feature is only related to a small image area.

Historically there were 3 most popular approaches in algorithms of feature detection:

1. **SIFT** (Scale-Invariant Feature Transform) : is one of the most classic. It is fully consider the changes in illumination, scale, and rotation during the image transformation, but is so calculating expensive, so rarely used in SLAM.
2. **FAST** : is extremely fast to compute. To understand its drawbacks, one should understand its algorithm in general. It selects a pixel  $p$  in the image, assuming its brightness as  $I_p$ . Then it sets a threshold  $T$  (for example, 20% of  $I_p$ ). Afterwards it selects the 16 pixels on a circle with a radius of 3 and a center  $p$ . If there are consecutive  $N$  points on the selected circle whose brightness is greater than  $I_p + T$  or less than  $I_p - T$ , then the central pixel  $p$  can be considered a feature point.  $N$  usually takes 12, which is FAST-12. Also  $N$  could be 9 / 11 / etc, with respective names of FAST.



[Fig 1] : Illustration of of FAST algorithm

Clearly, such algorithm has an problem, that the detector does not have descriptors (fixed pattern, which try to find). Also, as it has fixed the radius of the circle = 3, there is a scaling problem.

3. **ORB** (Oriented FAST and Rotated BRIEF) : it solves both mentioned problem of FAST, by adding the description of scale and rotation. In addition, it uses the extremely fast binary descriptor BRIEF. The scale invariance is achieved by the image pyramid and detect corner points on each layer. In terms of rotation, we calculate the gray centroid of the image near the feature point. The so-called centroid refers to the gray value of the image block as the center of weight. Then connect the geometric center  $O$  and the centroid  $C$  of the image block to get a direction vector  $\overrightarrow{OC}$ , so the direction of the feature point can be defined as:  $\theta = \arctan(\frac{m_{01}}{m_{10}})$ .

Due to the consideration of rotation and scaling, the ORB performs well under translation, rotation, and scaling. Meanwhile, the combination of FAST and BRIEF is very efficient, which makes ORB features very popular in real-time SLAM. Despite this, our tests showed that it is not enough for the requirement of being real-time in our implementation as we developed our SLAM via Python, thus we choose to use combination of improved BRIEF taken from the ORB algorithm and simple feature detection algorithm - Harris corner detection.

## 2.4 Harris Corner Detector

It is a naive, though powerful, feature detector, which has a great feature - being extremely computationally efficient - it can be represented only as 5 convolutions at image, which can be very efficiently vectorized. First of all, it is recommended to apply Gaussian blur so to make algorithm less error prone. Then one should calculate change of intensity for each pixel:

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2,$$

where  $w(x, y)$  is a window function - step, Gaussian etc. Using first order Taylor series approximation

$$f(x + u, y + v) = f(x, y) + u f_x(x, y) + v f_y(x, y).$$

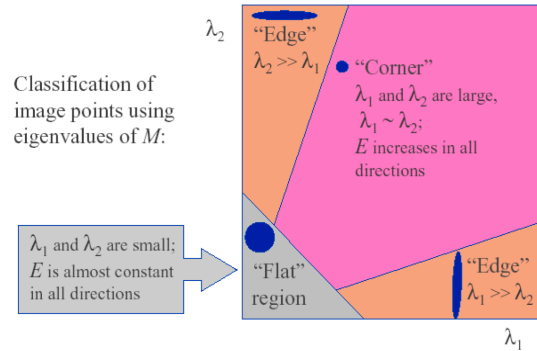
Which in turn transforms into

$$[u, v] \left( \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

Then one can calculate response of each pixel as

$$R = \det M - k(\text{trace} M),$$

which in turn is reduced to analysis of eigenvalues.



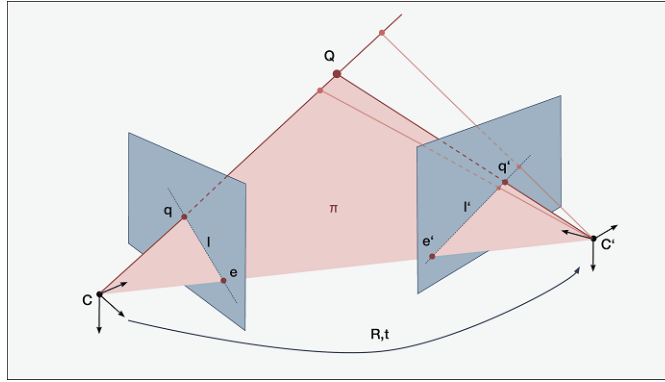
In the end, having determined the pixels where there might exist some corners, one should threshold the ones with lower response, which can be taken as synonym for confidence. Afterwards, it is recommended to apply non-maximum suppression, though we decided to omit this part as it brings additional computations, which do not influence our algorithm much.

## 2.5 BRIEF

BRIEF is a binary descriptor. Its description vector consists of many zeros and ones, which encode the size relationship between two random pixels near the key point (such as  $p$  and  $q$ ): If  $p$  is greater than  $q$ , then take 1, otherwise take 0. If we take 128 such  $p, q$  pairs, we will finally get a 128-dimensional vector consisting of 0s and 1s. The BRIEF implements the comparison of randomly selected points, which is very fast. Since it expresses in binary, it is also very convenient to store and suitable for real-time image matching. The original BRIEF descriptor does not have rotation invariance, so it is easy to get lost when the image is rotated. The ORB calculates the direction of the key points in the FAST feature point extraction stage. The direction information can be used to calculate the Steer BRIEF feature after the rotation so that the ORB descriptor has better rotation invariance.

## 2.6 Epipolar geometry

The object motion can be found from the correspondence between two consecutive video frames.



According to the camera model, each of the landmarks  $Q$  will be projected onto 2-D plane of pixels as  $q, q'$ . Motion, described by the rotation matrix  $R$  and transformation vector  $t$  can be reproduced than via Epipolar constraint.

Pixel points  $q, q'$  are described by the following equations:

$$\begin{cases} s_1 p_1 = K P \\ s_2 p_2 = K(RP + t) \end{cases} \quad (1)$$

As was discussed in the setting, due to the limitations of the approach, we can represent the points only to some constant factor. Moreover, it is worth mentioning that  $p_1, p_2$  will be the points matched by the feature matcher.

To find the coordinates on the normalized plane of two pixels, we can simply take the inverse of the matrix representing a camera:  $x_{1,2} = K^{-1}p_{1,2}$ . Then the equation above can



be rewritten as  $x_2 = Rx_1 + t$ . This can be further transformed into

$$p_2^T K^{-T} t \times R K^{-1} p_1 = 0.$$

This equation is referred to as epipolar constraint. For the sake of simplicity, the notions of Fundamental and Essential matrices are introduced:

$$\begin{cases} E = t \times R \\ F = K^{-T} t \times R K^{-1} \\ x_2^T E x_1 = 0 \\ p_2^T F p_1 = 0 \end{cases} \quad (2)$$

It is clear that usage of Essential matrix will make the calculations more accurate, as it uses precomputed calibration parameters for the camera. The process of camera pose estimation can then be summarized as:

1. Find  $E$  or  $F$  from the known pixel positions.
2. Find  $R, t$  based on those matrices.

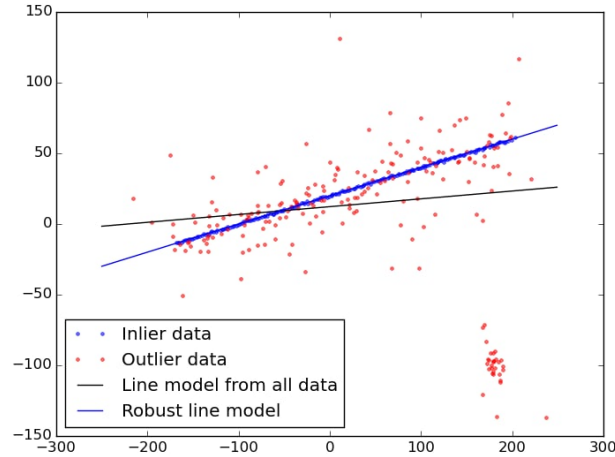
In practice, the Essential matrix is found via Eight-Point algorithm and then  $R, t$  are found via SVD and triangulation to get the result that would satisfy the constraints given by the known projections of landmarks. Let SVD decomposition of  $\mathbf{E}$  be:

$$E = U \Sigma V^T$$

$$\begin{cases} R = U R_{Z(\pm \frac{\pi}{2})} V^T \\ t = U R_{Z(\pm \frac{\pi}{2})} \Sigma U^T \end{cases} \quad (3)$$

## 2.7 RANSAC

Even if we do linear regression and apply least square solution for estimating parameters, than we will get not such result which we want, and the reason is outliers.



If we can identify outliers, then we can get much better approximation, and thus better model. And for this reason we use **RANSAC** [?] - **RAN**dom **SA**mple **C**onsensus algorithm. It has following steps:

1. Sample (randomly)  $k$  points to fit the model.  $k$  should be not least, than min number of points to fit model and less than number of all points, which we have. In case of line  $k \in [2, n)$ , in case of essential matrix:  $k \in [8, n)$ .
2. Calculate model parameters using chosen simple.
3. Score of our model by the fraction of inliers, as those points, which have distance not more than preset **threshold**.
4. Repeat previous steps  $m$  times and chose the model with the best score, calculate in step (3). In each step we can stop when we get needed number of inliers, or if we get needed probability of correct model. If we need the probability  $p$  and  $w = \frac{\text{num\_of\_inliers}}{\text{number\_of\_all\_points}}$ , than we get following:

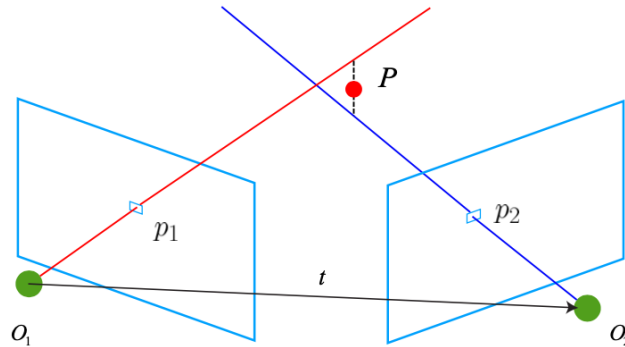
$$1 - p = (1 - w)^m \Rightarrow m = \frac{\ln(1 - p)}{\ln(1 - w)}$$

5. After previous steps, we re-estimate parameters for model, by taking into account all inliers, which we detect in the best model.

This approach work for any model and helps to detect and do not take into account outliers. In our case we use this algorithm for fitting Essential matrix.

## 2.8 Triangulation

The second goal of SLAM is to estimate the relative positions of the feature points. Knowing the relative position of the camera and the relative positions of those featurer points, we can reconstruct their global coordinates and construct the feature map of the environment.



From the epipolar constraint:  $s_2x_2 = s_1Rx_1 + t$ . Here  $x_{1,2}$  - normalised coordinates (coordinates in the pixels' plane) of two feature points. Knowing  $R, t$ , we can now calculate  $s_{1,2}$  that will represent the 'depth' of a pixel.

$$s_2 x_2 \times x_2 = 0 = s_1 x_2 \times R x_1 + x_2 \times t, \quad (4)$$

which can be solved for  $s_1$ . Geometrically, we seek to find a 3D point on the ray  $O_1 p_1$  to make its projection close to  $p_2$ .

### 3 Overview of approaches

As was mentioned earlier, there exist different types of SLAM algorithms that depend on different types of the input video stream. Though as we are focusing particularly on the visual SLAM in this work, we would consider different approaches to the SLAM problem from the perspective of Computer Vision:

- The most popular implementation of visual SLAM is OpenVSLAM available open-source. It utilises notions of global and local key frame maps. For the local map it finds points via triangulation and for the global one uses pose-graph optimization. It also contain a global optimization module, which optimizes matching against global map. [8]
- Interesting to mention is the approach proposed by the ORB-SLAM [3]. The project combines some of the fastest and most efficient algorithms resulting in being one of the most complete and reliable solutions. Some of the notable features and advantages are listed below:
  - Same features are used for multiple tasks e.g tracking, mapping, localization, loop-closing. This approach allows real-time performance without using additional hardware.
  - Could be operated in large environments thanks to a covisibility graph, which consists of keyframes as nodes and edges between nodes that have as least 15 map points
  - Real time loop is performed with an Essential Graph, a subgraph of covisibility graph that has at least 100 matching points
  - The authors use bag-of-words representation of encoded descriptors for each of the feature points in order to optimise the process of feature matching against the stored feature map.

It has also a logical continuation as ORB-SLAM 2, which introduced ability to work with stereo and RGB-D input data. [4]

- Speaking of the camera calibration, the classical approach to it is using a chess board photos from different perspective in order to estimate the intrinsic parameters. However the limitation of this approach is the fact that you cannot recover the intrinsic parameters having the video without direct access to the camera itself. Therefore it is worth mention some of the work focusing on the camera calibration using planar scenes. [6]. It uses homography-based self-calibration to obtain required parameters. Authors prove that this type of calibration is as accurate as classical approach. Moreover planar scenes could be give more robust result than when using an arbitrary scene.

- The research in the SLAM field is being actively maintained as technology have seen some success and made big technological companies interested. The problem is that many researches implement their own SLAM structure making it hard to test particular parts of the implementation, compare trade offs of different approaches and testing results. Thus the need of generalized SLAM frameworks emerged to ease research work and ideas exchange. The good example of such framework is GSLAM expressed in the cited paper. [9]. It provides open-source, plugin based approach alongside with universal benchmarking solution.

## 4 Implementation

The algorithm was implemented in Python mainly via `pyopencv`. The visualizer of the SLAM was written via `OpeGL`. For the feature extraction the Harris corner detector and BRIEF corner descriptor were used due to its computational efficiency. Feature matching was done in KNN manner because limitation on the number of features was only several hundreds. Because on the validation and test videos we had camera intrinsics, we used Essential matrix to recover rotation and transformation between two frames. The last step was to conduct triangulation so to get the 3-D coordinates of feature points. Demonstration of the algorithm can be found on Github or YouTube.

**Data:** stream of video frames

*init()* /\* make all the necessary initialization of visualizer and SLAM engine \*/ ;

$prev \leftarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$

*currentFrame*  $\leftarrow$  None;

*prevKeyPoints*  $\leftarrow$  None;

**while** true **do**

*cur*  $\leftarrow$  *getFrame()*;

*keyPoints*  $\leftarrow$  *getFeatures(currentFrame)*;

**if** *prevKeyPoints* is None **then**

*prevKeyPoints*  $\leftarrow$  *keyPoints*;

**continue**;

**else**

*matches*  $\leftarrow$  *getMatches(keyPoints, prevKeyPoints)*;

*E*  $\leftarrow$  *recoverEssentialMatrix(method = RANSAC, matches)*;

*R, t*  $\leftarrow$  *recoverPose(E)*;

*points*  $\leftarrow$  *trinagulate(matches, prevFrame, cur)*;

        /\* store all the necessary info \*/;

*poses.append([R, t])*;

*pointMap.extend(points)*;

        ;

        /\* update all the values \*/;

*tmp*  $\leftarrow$  *cur*;

*cur*  $\leftarrow$  *prev@cur*;

*prev*  $\leftarrow$  *tmp*;

*prevKeyPoints*  $\leftarrow$  *keyPoints*;

**end**

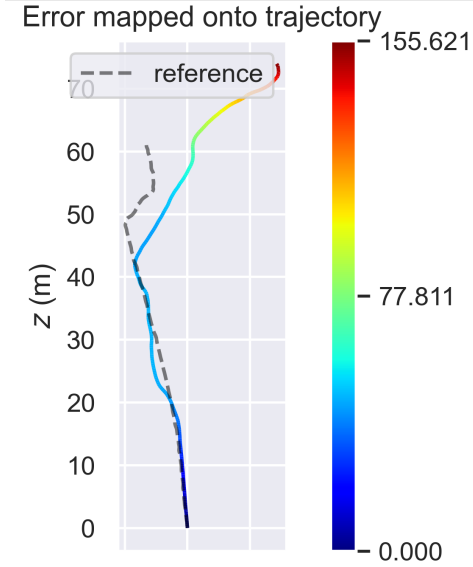
**end**

**Algorithm 1:** General algorithm of SLAM

## 5 Experiments

Experimentation was conducted on publicly available Kitty dataset [10]. It was collected via a standard station wagon with two high-resolution color and grayscale video cameras and is widely used in testing of different SLAM approaches. Accurate ground truth is provided by a Velodyne laser scanner and a GPS localization system. As we are working only with monocular SLAM we took only the video from the first camera to conduct testing. Moreover, the dataset also provided calibration data for cameras, which took out the need of researching of obtaining camera calibration from a given video source, which is completely another topic.

Evaluation was conducted on a small subset of the provided dataset, due to lack of computational resources, to be more specific - **2011-09-26-drive-0095'**. To measure metrics, we used open-source library for SLAM researchers - **evo** [11]. It contains tools to measure absolute and relative errors, whether matrices that describe poses conform to SE(3) restriction and many more.



We mainly focused on **RPE** - relative pose error, which allows to investigate the local consistency of a SLAM trajectory, which is the most interesting metric for us as our SLAM is not stable on long distances because of the lack of backend optimizers, which result in a huge absolute error.

RPE compares poses along estimated and reference trajectory. It is based on the delta pose difference [12]:

$$E_{i,j} = (P_{ref,i}^{-1} * P_{ref,j})^{-1} * (P_{est,i}^{-1} * P_{est,j})^{-1} \in SE(3); \quad (5)$$

$$RPE_{i,j} = \|E_{i,j} - I_{4 \times 4}\|_F \quad (6)$$

It is easy to notice the drift that the system suffers from - the reason for that is that the error accumulates constantly and we could not get rid of it without some non-linear optimizer, which was not in the focus of this project.

To see the drift even more clearly, we would decompose the position into a set of (x, y) coordinates:

Figure 1: Estimated y coordinate (in meters) against ground truth over iterations.

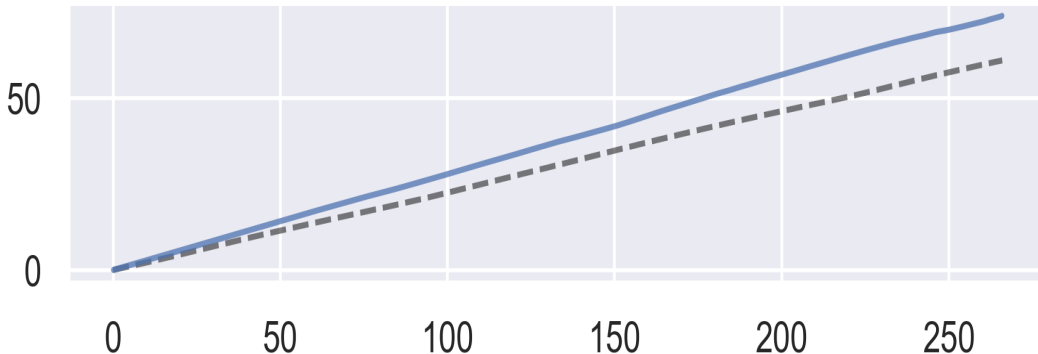
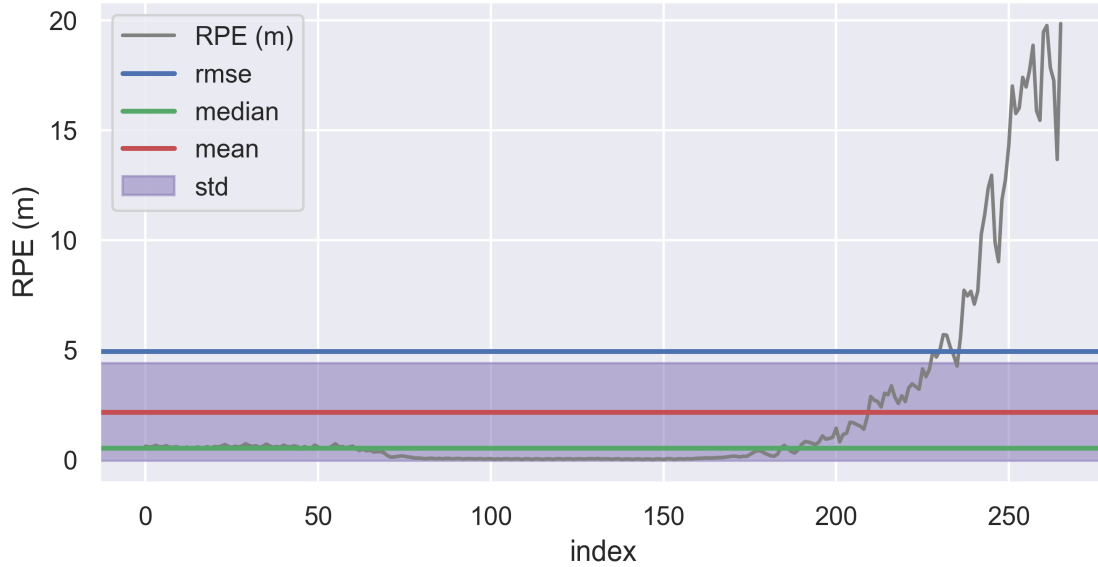
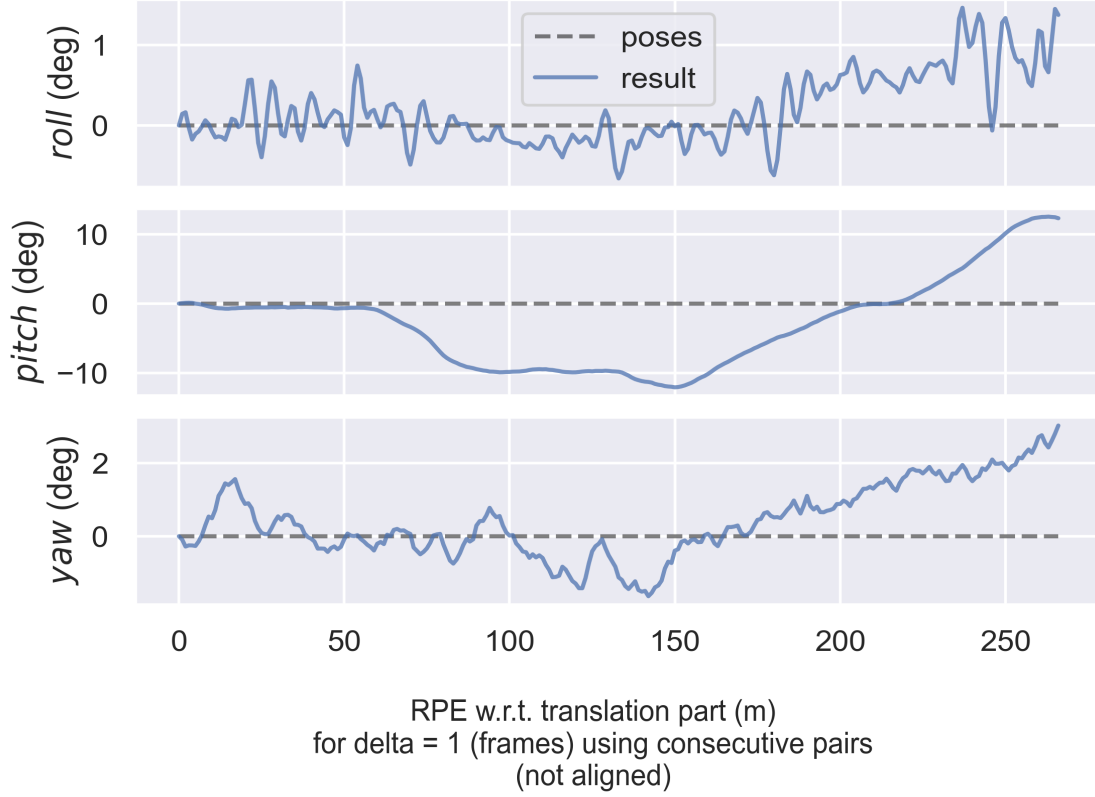


Figure 2: Relative error of estimated rotations.



## 6 Limitations

Monocular SLAM is the simplest SLAM possible, thus it possesses many limitations. The most prominent from them are the following:

1. Main limitation of monocular SLAM is the fact that it can determine translation only to some constant factor, as while we project world onto image plane we lose the information about depth of objects that we see. To overcome this problem, one may use different type of camera - RGBD (measures depth as well) or different type of algorithm - Stereo SLAM - the one that takes images simultaneously captured from two or more cameras.

2. Triangulation is only possible if the enough amount of translation is present. If the amount of translation is small, pixel's change would result in big depth uncertainty. On the other hand, if translation is too big, the algorithm may fail to match the features. This problem will be especially seen in cases of so-called pure rotation, when there is no translation. Though in not so extreme cases, there are means to make this problem less visible - delayed triangulation etc.
3. Assumption about static sight of view brings a lot of limitations as well. For instance, the algorithm cannot be applied to autopilots as the moving object would make it think that the camera has moved.

## 7 Conclusions

In this work we developed a Monocular SLAM engine which allows us to track object given input video source from a single camera. It gives adequate results, speaking in terms of relative pose error, in short terms, though suffers from error accumulation over time. Basing on reviewed literature, back end optimizers, such as Extended Kalman Filter or Bundle Adjustment, which utilizes non-linear optimizers to make the error of projections of discovered features points in world coordinates as small as possible, could greatly improve the overall quality of SLAM and get rid of those accumulated errors. Sadly, this part is more connected with signal processing and non-linear optimization rather than with Linear Algebra thus we did not focus on that in this work. Speaking about the further work, we would propose to implement one of those optimizers to make SLAM more accurate. We also see this project as a great foundation for further research for us - combined with deep learning, 3-D map and poses obtained by SLAM could greatly influence the field of reinforcement learning for robots.



## References

- [1] <https://github.com/bohдахllovatskyi/OhISee>
- [2] [http://www.cvlibs.net/datasets/kitti/eval\\_odometry.php](http://www.cvlibs.net/datasets/kitti/eval_odometry.php)
- [3] <http://webdiis.unizar.es/~raulmur/MurMontielTardosTR015.pdf>
- [4] <https://128.84.21.199/pdf/1610.06475.pdf>
- [5] <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf>
- [6] <https://users.aalto.fi/~kannaljl/publications/wacv2016.pdf>
- [7] <https://github.com/gaoxiang12/slambook-en>
- [8] <https://arxiv.org/pdf/1910.01122.pdf>
- [9] <https://arxiv.org/pdf/1902.07995.pdf>
- [10] <http://www.cvlibs.net/datasets/kitti/>
- [11] <https://github.com/MichaelGrupp/evo/>
- [12] <http://www2.informatik.uni-freiburg.de/~stachnis/pdf/kuemmerle09auro.pdf>