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

Abstract

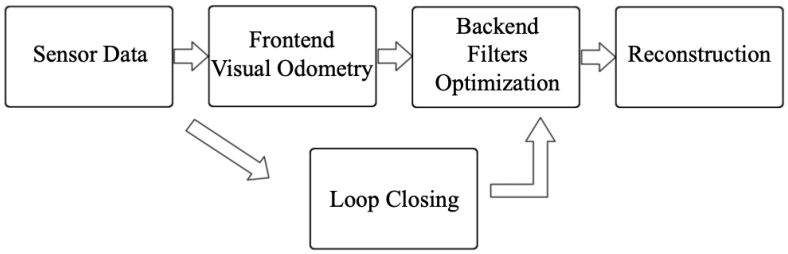
SLAM or Simultaneous Localization and Mapping - problem of constructing a map of unknown environment simultaneously tracking the object's location. Such algorithms utilize the data obtained from sensors, such as video camera, lidar etc to construct a 3-D model of the nearby objects. They are used in variety of fields: autonomous driving and robots, augmented reality and much more. The algorithms that rely on video data are referred to as Visual SLAM. They obtain the valuable information by means of Computer Vision and different techniques of Signal Processing. We implemented such algorithm by means of Python and pyopencv 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. This is the simplest type of SLAM, though it has a lot of limitations, the most important of which is the fact that 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 single image source.
- Stereo SLAM. Input video stream comes from two cameras. This allows to get the exact depth of the featured points.
- RGBD. Specific kind of SLAM that gets the input data from RGBD camera - the one that included depth of objects within an image. There are different technologies for such cameras though they are not particularly important for the algorithm itself.

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 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. Collecting 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:

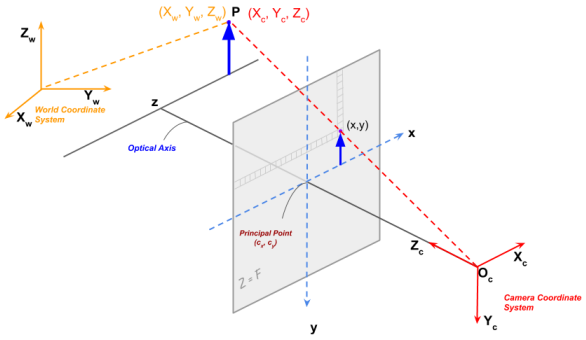
$$SE(3) = \{T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \in \mathbf{R}^{4x4}\},$$

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 threat 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_w down and Z_w towards the camera

- The point projection to the plane (image itself)

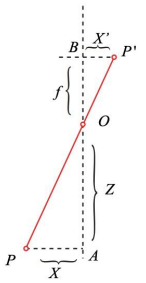
The world and camera coordinate systems are related by the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$. Thus

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

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

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

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 though 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):

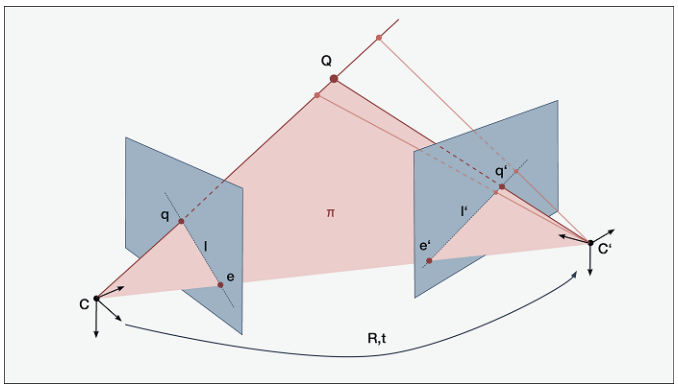
$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} Y_c \\ X_c \\ Z_c \end{bmatrix}$$

where

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

2.3 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:

$$\begin{cases} s_1 p_1 = KP \\ s_2 p_2 = K(RP + t) \end{cases}. \tag{1}$$

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

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

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

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

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:

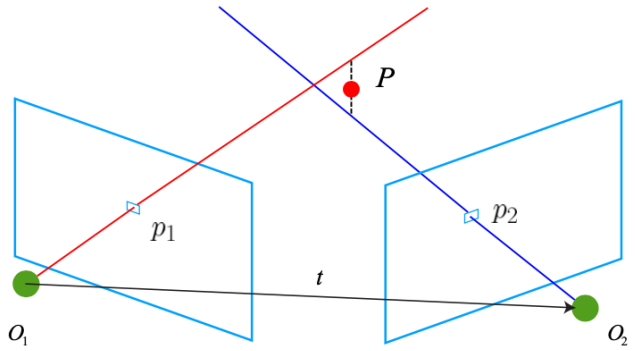
1. Find E or F from the known pixel positions.
2. Find R, t based on those matrices.

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.

$$\begin{cases} R = UR_{Z(\pm pi/2)}V^T \\ t = UR_{Z(\pm pi/2)}\Sigma U^T \end{cases} . \tag{3}$$

2.4 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 featuer points, we can reconstruct their global coordinates and construct the feature map of the environment.



From the epipolar constraint: $s_2 x_2 = s_1 R x_1 + t$. 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.

$$s_2 x_2 \times x_2 = 0 = s_1 x_2 \times R x_1 + x_2 \times t, \tag{4}$$

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

4 Implementation

Data: stream of video frames
initialization;
calibrate camera;
while *true* **do**
    get frame;
    get the features;
    match the features between current and previous frame;
    find essential matrix;
    find rotation matrix and transformation;
    store the movement and point map;
    update previous frame;
end

Algorithm 1: General algorithm of SLAM

The algorithm was implemented in Python via pyopencv. The visualizer of the SLAM was written via OpegGL. For the feature extraction the ORB was used due to its computational efficiency. Feature matching was done in brute-force manner because limitation on the number of features was only several hundreds. Fundamental matrix was then constructed as we lack the camera parameters for the test video.

5 Experiments

6 Limitations

There are two main limitations of such approach:

1. 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.

2. 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

References

[1] `https://github.com/bohdanhlovatskyi/OhISee`