Simultaneous Localization and Mapping for Vehicles

Dylan Li
Harvard University
Cambridge MA, 02138
li_dylan@college.harvard.edu

Jonathan Hu
Harvard University
Cambridge MA, 02138
jonathanhu@college.harvard.edu

Abstract

Simultaneous localization and mapping (SLAM) is a computational problem of constructing 3D maps and localizing a camera within that 3D space. In this paper, we build an end to end SLAM pipeline for vehicles, implementing multiple functions that are natively implemented in open CV. In our implementation of SLAM, we use a Samsung Galaxy S10 Plus, whose intrinsic parameters are calculated at constant focal length using images derived from the April Calibration Board. We build on SLAM pipeline by taking pictures across multiple different scenes, and building a 3D scene reconstruction using the PnP algorithm.

1. Introduction

Simultaneous Localization and Mapping (SLAM) has been a central problem in robot perception and state estimation. SLAM builds a 3D representation of an environment and localizes a vehicle or camera center within that 3D representation. With that knowledge of 3D points, a robot or other agent in the space can make decisions on how to behave in that space. Visual SLAM systems usually use at least a single camera that moves through a space and reconstruct scene points based on the camera's movement through that space.

In general, SLAM can be considered a solved problem. The need to build and use a map of the environment has multiple use cases. Firstly, the map can inform path planning or provide a visual intuition for a human operator of a robotic system. The map also allows for the robot to limit the error in state estimation. By using a map and localizing its position within that map, it is not prone to sensor drift using traditional localization algorithms.

Our final project aims to implement a SLAM pipeline and to build a 3D mapping of Winthrop House, one of Harvard's historic undergraduate dorms (see figure 1).

1.1. Related Work

SLAM relies on both the trajectory of the platform and the location of all landmarks without the need for *a priori* knowledge of location.

Different papers offer different perspectives on the mapping and localization problem. In Probabilistic SLAM, at a particular point in time t, we define the following quantities [2]:

- x_t: the state vector describing the location of the camera at time t
- m_t: the vector describing the location of the ith point in 3D space
- $z_{i,t}$: observation taken from the vehicle of the ith landmark at time t

Probabilistic SLAM aims to use a Bayesian updating step to estimate state and reconstruct a scene:

$$P(x_t, \vec{m}|Z_{0:t}, U_{0:t}, x_0)$$

This probability distribution describes the joint posterior density given recorded observations. As a robot or camera moves through a scene, SLAM gives a sequential prediction and correction to the scene points and its location in space. A map $\vec{m}$ fuses the points from the camera, and the posterior density provides an estimate for the sequential locations of the camera in the 3D space.

Structure from Motion (SfM) is a related work that allows one to reconstruct a scene from unordered images. The pipeline for SfM begins with an iterative process of feature extraction and matching. As images are added to the pipeline, points from those images are triangulated out into space and outliers are filtered out.

However, the robustness and accuracy of SfM pipelines is a well-known problem. In their "Structure-from-Motion Revisited" paper, Schoenberger et al. propose three methods for increasing accuracy and robustness: 1) Next Best View Selection, 2) Robust and Efficient Triangulation, and 3) Redundant View Mining [7].

Figure 1. Courtyard of Winthrop House

With regards to performance and processing speed, recent works include Lim et al.'s (2016) contributions in "Real-time monocular image-based 6-dof localization", which implements an optimized SLAM system that can run at 30Hz on a laptop [4].

2. Methods

2.1. Implementation Strategy

Given the scope and complexity of implementing a SLAM pipeline, we first assembled a fully functional pipeline using pre-implemented functions found in the OpenCV library.

2.2. Overview

Using the intrinsic camera matrix K and a set of n sequential images to process, we want our system to output a 3D mapping (see figure 2) and the localization of the camera.

The preliminary step is to calibrate the camera. We use code from the April Robotics Lab at the University of Michigan [6] to calibrate and find the intrinsic camera matrix K, where:

$$K = \begin{bmatrix} f_x & s & x_o \\ 0 & f_y & y_o \\ 0 & 0 & 1 \end{bmatrix}$$

First, we build a core from the first two images and find the camera matrices P_0, P_1. With the two camera matrices, we can extract the camera centers and backproject to get 3D world coordinates. Building the core starts with matching keypoints in the first two images using the SIFT algorithm. The choice of the initial two images is extremely important. In "Structure-from-Motion Revisited", Schoenberger et al. (2016) explain that the reconstruction of 3D points might not be able to recover from a pair of images with sparse interest points; the reason is that a dense image provides increased redundancy [7].

Our code for matching descriptors between the two images comes from assignment 2, which upon observation seems to have been inspired by the OpenCV-Python Tutorials series on "Epipolar Geometry" [1].

The next step is to recover the fundamental matrix F, which relates 2D coordinates in one image to epipolar lines in the second image. We implemented the 8-point algorithm version and RANSAC for robustness against outliers, replacing the OpenCV library function findFundamentalMatrix() with parameters cv.RANSAC.

Next, we implemented our own version of the findEssentialMat() function. The implementation of this function first required us to normalize our matched points in two images. Then, we performed singular value decomposition on a set of 8 random points, applied the constraint that the fundamental matrix is singular, and ran RANSAC to find the fundamental matrix that maximized the number of inlier points. Then, using equation 9.12 from Hartley and Zisserman [3], we see that given the fundamental matrix F and the intrinsic camera matrix K, the essential matrix E can be computer from:

$$E = K^T F K$$

Here, we call the recoverPose() function. Given the essential matrix E and at least one 2D correspondence, the pose recovery function finds the rotation R and translation t matrices of the second camera, assuming that the first camera is situated at the 3D world origin $(0, 0, 0)$ and has no rotation R component. The recoverPose() uses the chirality test to determine which of the four poses is the right one. We rely on the OpenCV implementation for the recoverPose() function.

Now that we have the intrinsic camera matrix K, the rotation matrices R_0, R_1, and translation matrices t_0, t_1, we compute the camera matrices P_0, P_1.

$$P_i = K_i[R_i|t_i]$$

Finally, we replaced the OpenCV implementation of triangulate(), which backprojects the 2D points into 3D world coordinates. Again, this required using the cross product trick (multiplying $x'(Px)$). We also implement extracting the camera centers from the camera matrics using the property that the null space of camera matrices is the camera center [3].

Now, at the end of the process of building the core, we have two camera matrices, a set of 3D points and a set of camera centers (See figures 2 - 4).

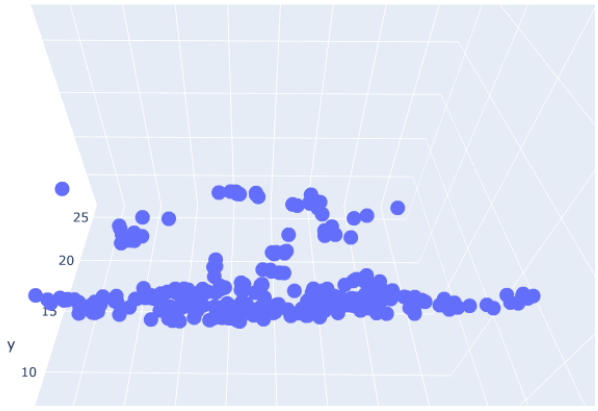

Figure 2. 3D Mapping from Core (Front View)

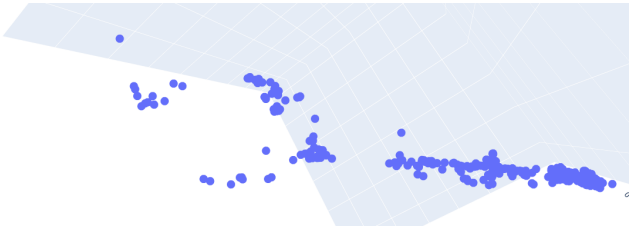

Figure 3. 3D Mapping from Core (Side View)

2.3. Generalizing to n sequential images from the initial core

We continue by adding on one image at a time.

We implement the data processing pipeline below. We run SIFT on each new image, which produces a the new set of approximately 50,000 descriptors. We filter this set against the descriptors from the previous image that correspond to the set of inliers, and in our demo we are left with 235 points. We run findFundamentalMatrix to filter out the outliers and are left with about 193 points.

This set of 193 points is what we feed to the next function: solvePnP. At this point our implementation ends, and we switch back to the OpenCV implementation of the Efficient Perspective-n-Point Camera Pose Estimation function called solvePnP [5].

We keep on adding one images, each time adding the new 3D points to our existing list of 3D points, and in the end we are able to generate a 3D mapping of the scene we walked through (see figure 2, 3, 4).

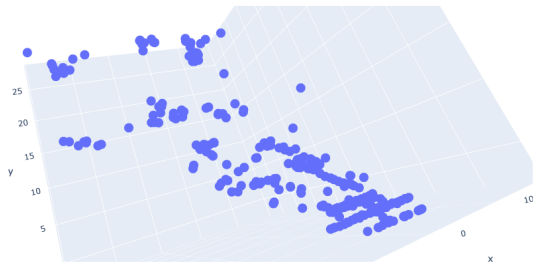

Figure 4. 3D Mapping from Core (Perspective View)

3. Experiments

3.1. Data

For this project, we initially wanted to calibrate, capture and analyze data from a camera mounted on a moving vehicle. In the end, we opted to approximate the motion and data from a moving vehicle by taking sequential pictures from a mobile phone while walking outdoors due to time constraints. The location of the photos is the courtyard of Winthrop House (32 Mill Street, Cambridge, MA 02138. See figure 1).

3.2. Camera Parameters

The following parameters and settings are provided for reproducibility of our experiments. The photos are taken using the Adobe Lightroom App.

Device: Samsung S10 Plus

Focus: 100%

Image Format: sRGB

Export Image Quality: High

3.3. Camera Intrinsic Parameters Estimation

To build our SLAM pipeline, we needed to estimate relative camera pose and the intrinsic camera parameters. We used an AprilBoard, for which we know the locations with respect to a coordinate system that is attached to the plane. The AprilBoard calibration package provided us with the following intrinsic camera matrix [6]:

$$\begin{bmatrix} f_x & s & x_o \\ 0 & f_y & y_o \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2785 & 0 & 1519 \\ 0 & 2790 & 1929 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Results

See figures 2 - 7.

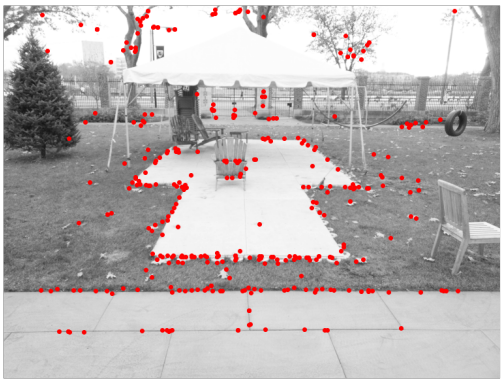

Figure 5. SIFT Interest Points in Core Image 0 after Fundamental RANSAC

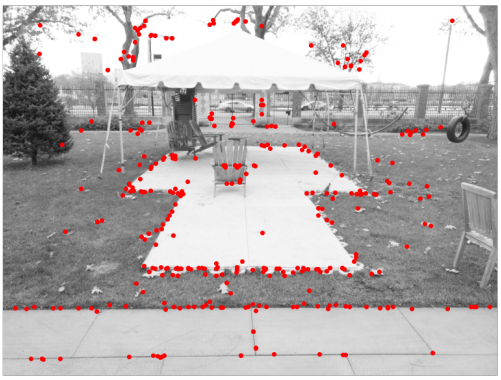

Figure 6. SIFT Interest Points in Core Image 1 after Fundamental RANSAC

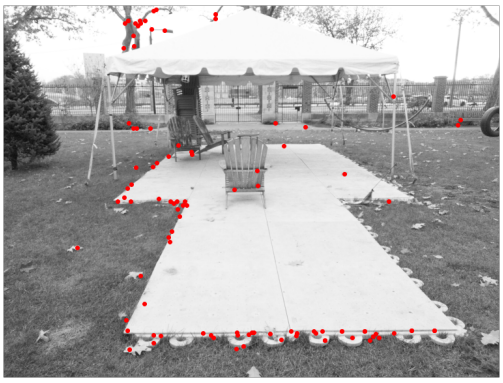

Figure 7. SIFT Interest Points in Image 2 After Filtering for Matches in Image 1

5. Discussion & Limitations

5.1. Limitations

The current processing bottleneck happens at the descriptor matching step. In Google Colab, it takes approximately 2 minutes to match descriptors of two images. For real-time SLAM applications such as autonomous driving, this processing time is an order of magnitudes too slow. Lim et al. (2016) discuss methods for increasing processing speed and are able to achieve running the SLAM system at 30Hz on a laptop in "Real-time monocular image-based 6-dof localization" [4].

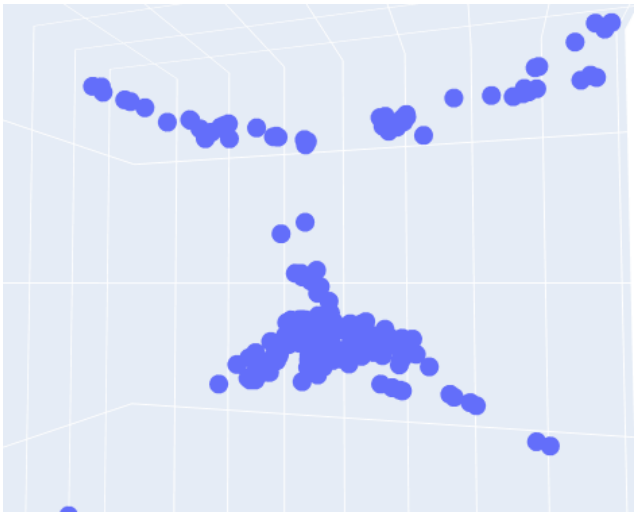

Figure 8. Final 3D Mapping (Front View)

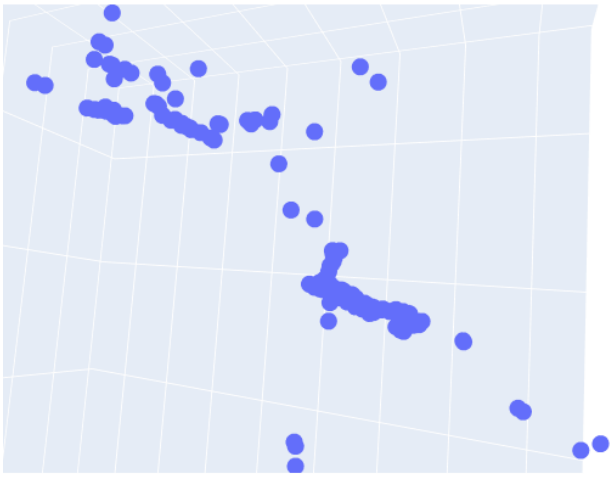

Figure 9. Final 3D Mapping (Side View)

5.2. Evaluation of Structure from Motion

By visual inspection of the inlier interest points in each image, we observe that the pair of core images is relatively sparse. There very few interest points on the the white tent, on the pine tree and the grass. Recall from section 2 the importance of picking a good initialization of core images. We observe that the number of inlier interest points in image 2

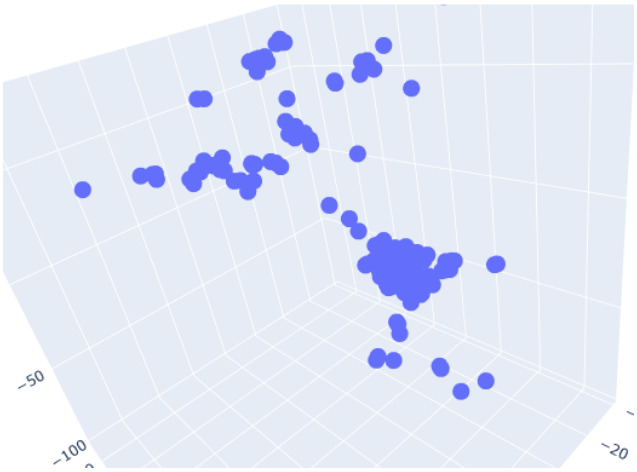

Figure 10. Final 3D Mapping (Perspective View)

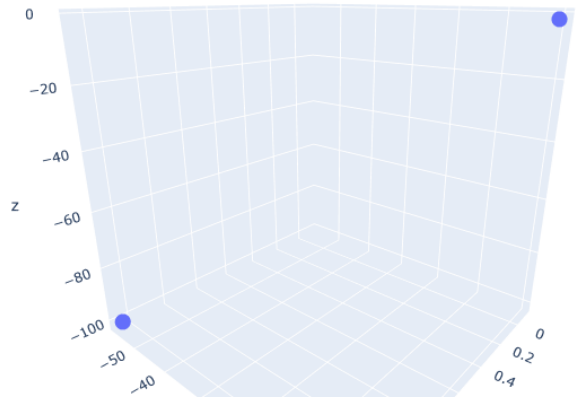

Figure 11. Camera Centers

is even more sparse than the core images. Indeed, as noted by Schloenberger et al., "the reconstruction may never recover from a bad initialization" [7]. Inspecting the images, we noticed that the type of scene we chose is intrinsically sparse in interest points because alot of the image is uniform in color and texture. Thus, walking through an urban neighborhood might produce an image with much more interest points, which is something we will consider in the future.

We also observe in the interactive 3D plot that the 3D mapping produced by our pipelines is quite inaccurate. For instance, 3D world points we expect to be in the same plane (i.e. interest points of the ground) are not (refer to Python notebook). Our camera centers may also be inaccurate with the below plot. We believe this may have been due to numerical instability, and when printing out our camera centers, many of them were centered the origin, as expected.

5.3. Future Directions

One limitation of our implementation of SLAM is the low number of inlier points after we compute the fundamental matrix with RANSAC. The low number of inlier points ultimately results in a sparse 3D map that may not be very representative of the 3D world around the camera. For the purposes of this research project, a low number of inlier points is acceptable. However, for real-life applications such as autonomous driving where the system depends on a dense 3D map to 1) detect the presence of an object or person and 2) identify if an object is a pedestrian, a car or a bike.

The robustness and accuracy of SfM pipelines is a well-known problem. Schoenberger et al. propose three methods for increasing accuracy and robustness: 1) Next Best View Selection, 2) Robust and Efficient Triangulation, and 3) Redundant View Mining [7]. These improvements can be implemented in future iterations of our SLAM pipeline.

6. Conclusion

In conclusion, the SLAM pipeline we implemented works, but the outputted 3D mapping suffers from inaccuracies and is not robust to the initialization of core images.

Reflecting on our implementation of the SLAM pipeline, we are fascinated by how starting with only a series of sequential images and the intrinsic parameters of our camera, we are able recover the structure of a scene and the our location within a scene by using projective geometry principles. While the inputs and outputs to the SLAM pipeline seem simple, its underpinnings rest on decades of hard by researchers in the fields of computer vision, projective geometry and systems performance.

References

[1] Epipolar geometry. *OpenCV*. 2

[2] H. F. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping. 2006. 1

[3] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, UK, 2nd ed. edition, 2003. 2

[4] H. Lim, S. N. Sinha, M. F. Cohen, M. Uyttendaele, and H. J. Kim. Real-time monocular image-based 6-dof localization. *The International journal of robotics research*, 34(4-5):476–492, 2015. 2, 4

[5] S. Mallick. Head pose estimation using opencv and dlib. *LearnOpenCV*, May 2021. 3

[6] A. Richardson, J. Strom, and E. Olson. AprilCal: Assisted and repeatable camera calibration. November 2013. 2, 3

[7] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 5