MonoSLAM via 1-point RANSAC with EKF

TJ Melanson

Paper ID melanson

Abstract

The purpose of this project was to build a MonoSLAM algorithm capable of mapping both the camera location and that of objects in the environment without help from GPS or other locationing devices. This paper goes over the creation of an EKF-SLAM algorithm, as well as how to use 1-point RANSAC to create point-cloud imaging from sets of a single image.

1. Introduction

Simultaneous Location and Mapping (SLAM) consists of being able to correctly map an environment while at the same time determining one's location within the map. Because this would allow a robot to be ”truly autonomous” (as it would not need outside help for navigation), SLAM has been considered a ”holy grail” problem for mobile robotics[1]. This is because, until the past couple decades, the compounded error of estimating position from distance, and vice versa, was thought to become unbounded. That is, the combined use of position and mapping data would make both estimates become unreliable.

2. Problem Statement

The algorithm relies on this probability distribution:

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(m|X_{0:k}, Z_{0:k})P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0).$$

Where $X_{0:k}$ is the overall trajectory being followed instead of a single pose.

The SLAM algorithm consists of several steps:

1. Feature detection via SIFT/FLANN
2. Feature extraction via RANSAC
3. Updating physical and object locations via the Extended Kalman Filter

First, there must be a scale-invariant way to determine common points between the pictures.

Although most traditional SLAM algorithms use Joint Compatibility Branch-and-Bound (JCBB) to determine the best fit for the data, they usually are unnecessarily intensive and slow due to JCBB's exponential runtime. Therefore, we use a RANSAC algorithm, which determines the best curve fit from a randomly generated algorithm and a stochastic pre-conditioning of previous lines.

The Extended Kalman filter is used to make sure the compounded error from the position and velocity estimates converge to the smallest value. In order to do so, it takes in the closeness of a given measurement to a predicted value and thereby determines the ”weight” the measurement will have in the calculation of the final value (this will be referred to as the gain).

3. Technical Content

First, the SLAM algorithm needs an estimation of where the location of the camera currently is. This is done by interpolating the kinematic data of the device.

In order to determine a single point, one can also find the essential matrix E. This can be done by first finding the fundamental matrix F via point correspondence, then using the formula

$$E = K^T F K$$

to determine E. From there, the position of the camera, if not rotated, is the essential matrix converted from the cross-product matrix t_x.

Secondly, this estimate must be calibrated via external image points. From the current view and past locations, the camera must determine the location of several relevant world points to orient itself. However, calculations of an object based on a single correspondence point are often inaccurate and lead to inaccurate estimations. Therefore, SLAM algorithms go through a tree of stored in order to determine the most likely location of a given point. Most SLAM code uses Joint Compatibility Branch-and-Bound. This algorithm is an extension of the Nearest Neighbor approach, which greedily chooses the location in the previous level closest to that of the current estimation. Instead, JCBB determines the best fit for both ends.

In this project, JCBB was discarded in favor of Random Sample Consensus, or RANSAC. Rather than check each member of a branch, which takes exponential time, RANSAC relies on sampling the data and updating the line of best fit until convergence.

The two samples are then taken through an Extended Kalman Filter. The filter approximates the position of the robot, x_k, as well as each detected landmark, z_k, as follows:

$$x_k = f(x_{k-1}) + w_k$$
$$z_k = h(x_k) + v_k$$

where $f(x)$ is the approximated velocity described above, $h(x_k)$ is the projected geometry of the point z_k via RANSAC, and w_k and v_k are Gaussian distributions about the means of $f(x_k)$ and $h(x_k)$ respectively (i.e. the source of error). From there, the Kalman filter "predicts" the actual x_k, as well as the observation prediction P_k using the variance Q_k:

$$\hat{x_k} = f(\hat{x_{k-1}})$$
$$P_k = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1}$$

where F_{k-1} is the Jacobian of $f(x_k)$. Next, in order to merge the current guess with the rest, G_k, the "gain" which determines the weight of each point, is found:

$$G_k = P_k H_k (H_k P_k H_k^T + R)^{-1}$$

where R is the covariance of the system. From there, the difference between the expected versus given observation is factored into x_k and P_k updated:

$$\hat{x_k} = \hat{x_k} + G_k(z_k - h(x_k))$$
$$P_k = (I - G_k H_k) P_k$$

This process is repeated until convergence. This will greatly improve the overall error of a system.

4. Experimental Setup and Results

Although this was originally developed for a mobile device, it was considered better to use and test still images first before trying the actual device. This was to focus more on the computer vision side of the project rather than the application side. To work with the device, however, the camera matrix needed to be extracted from the system. For that, a Matlab toolbox was used with phone camera images of the checkerboard box. The resulting camera matrix was recorded and used for all subsequent images.

As for the testing, the camera took several photos of objects in a room. Although the exact dimensions of the objects were unknown (approximately 2 meters from the camera), the displacement of the camera was recorded. The camera took photos at an origin, 300 millimeters to the left, 300 millimeters to the right, 250 millimeters in front, 250 millimeters behind. It would be up to the camera to check the locations recorded by the object.

The testing process first consisted of several manually recorded points. The results were then extrapolated from similar points on two images (each with the measurements shown above). Once reasonable points were attained from the set, the experiment moved on to using SIFT and FLANN to automatically determine common features between the images.

A working EKF filter, though not implemented in the final system, was also implemented, as well as methods to compute the required Jacobians H_k and J_k (mentioned above).

5. Conclusions

Overall, although the tests separately worked, the experiment was not able to gather together well. The results were similar to that of the cited papers in terms of accuracy at each step in the process.

References

http : //www.vision.caltech.edu/publications/SoattoIJCV1997.pdf
http : //home.wlu.edu/ levys/kalman$_t$utorial/
http : //ieeexplore.ieee.org/stamp/stamp.jsp?tp = arnumber = 976019
http : //webdiis.unizar.es/ jcivera/papers/civera$_e$tal$_j$fr10.pdf