Simultaneous Localization and Mapping for a Planetary Rover

Priyanka Deo  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213  
pdeo@andrew.cmu.edu

Prashant Sridhar  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213  
psridha1@andrew.cmu.edu

Abstract
Position estimation is the crux of autonomous robot navigation. Simultaneous Localization and Mapping (SLAM) is a method of estimating position while building a map and exploring an unknown environment. This research implements Extended Kalman Filter, Monocular SLAM (EKFMonoSLAM) that has been modified to better suit lunar applications. The algorithm was run on a lunar simulation that approximates images taken by a lunar rover. The unmodified EKFMonoSLAM algorithm has 36% error on the simulated dataset, while the modified algorithm improves this to 7% error. Compared with the Library for Visual Odometry 2 (LibVISO2) stereo SLAM algorithm, the modified EKFMonoSLAM algorithm calculates final position and distance traveled more accurately. This research shows that the modified EKFMonoSLAM algorithm is the best suited, of those compared, for lunar applications, and can be used for autonomous planetary rover navigation.

1 Introduction
1.1 Rover Navigation
A planetary rover may be tasked with traveling long distances to landmark locations. In order to achieve this task, the rover must accurately navigate on the planetary surface. If the rover loses communication with the Earth, it has two options: it may continue to move, which is risky if it cannot navigate autonomously, or it may remain stationary, which is inefficient. An on-board autonomous navigation system combined with SLAM, can guide the rover to its destination while avoiding hazards. The rover’s navigation system would also help operators on Earth control the robot by giving them a better understanding of the rover’s environment.

On the moon, a rover can use several methods for navigation. One such approach would be to use existing maps of the lunar surface. This approach is difficult because current data about lunar surface elevatio
n is unreliable over the whole surface of the moon. Current digital elevation maps (DEMs) of the lunar surface are based on LIDAR readings from the Lunar Reconnaissance Orbiter (LRO). However, current elevation readings are on the order of $10^1$ meters per pixel (the resolution at the equator is about 30m/pixel). This resolution of lunar maps is adequate for viewing from a distance; however, it obscures features that may be of importance to a viewer at a low altitude, such as a rover navigating the lunar surface. Although higher-resolution information does exist, this information is restricted to only a small subset of locations on the moon and is limiting especially if the rover must travel long distances.

Another approach for rover navigation is visual odometry. This approach has previously been implemented as a means for navigation on the moon. Visual odometry uses a sequence of images and tracks how one frame moves in comparison with the next in order to estimate how the rover has moved. Errors in visual odometry stem from incorrect feature detection and association, and errors in precise image correlation. For visual odometry algorithms that do not update calculations, errors in associating images propagate for subsequent calculations. Thus, even if an algorithm has low error for each correlation, over time, these errors aggregate.

Therefore, this research implements a SLAM algorithm, which takes a sequence of images and tracks features in the image, using that data to simultaneously figure out where the robot is and map the environment. The map is used to correct the estimation of where the robot is, and the location of the robot in the map is used to generate the map. This research shows that SLAM is an effective means for autonomous rover navigation.

1.2 Lunar Constraints

The lunar environment poses some challenges to the SLAM algorithm. The terrain appears homogenous in images, thus finding defining features of the environment is difficult. Because the bandwidth of Earth-to-Moon communications is limited, the latency of data traveling to Earth and back is too high, and communications with the Earth are unreliable, off-board navigation is not feasible. Therefore the rover must process the navigation algorithm on-board. This research modifies the EKF MonoSLAM algorithm to work within these constraints.

2 Methodology

SLAM algorithms, in general, have four major steps – prediction, data association, measurement update, and augmentation. In the prediction step, the algorithm predicts the new state vector and covariance matrix from the previous state and covariance.

EKFMonoSlam uses the following techniques to track and match features and to estimate pose.

Extended Kalman Filter – Given a model for the present state and sensor data, the filter updates the model and estimates position and orientation of the rover.

1-Point RANSAC – RANSAC is used to reject features which are outliers and probably correspond to noise.
Low and High Innovation features – EKFmonoSlam treats points close to the camera and on the horizon differently since features on the horizon have possibly infinite depth as parallax is not observed until there is a significant displacement.

2.1 Extended Kalman Filter

EKFmonoSLAM uses an Extended Kalman Filter in order to predict the new state and covariance. The Kalman filter builds a model of the rover and uses matched features to estimate pose. Given model composed of pose and velocity, the Kalman Filter first estimates the new position of the rover and then verifies this against matched features to correct for errors. This largely reduces covariance of the measure. The drawback of the Kalman filter is that it assumes the model varies from one time frame to the next in a linear fashion. The Extended Kalman filter overcomes this by allowing for the model to vary with as a differential function instead of a strictly linear one.

2.2 1-Point RANSAC

The 1-Point RANSAC algorithm is used to reject outlier features which have passed primary correlation matching. Corners in the images are used as features using a fast corner tracking algorithm. After each frame, a basic correlation matcher is run across the features and the missed features are not used for the model. Finally, the accepted matches are passed through RANSAC. RANSAC builds a model of expected position of the features using a random sample of the features assuming that they are inliers. Other features are then matched against this and if they correlate well, they are also labeled as inliers. Finally, the covariance of the model is calculated. This process is repeated several times and the model with the least covariance is chosen. Features which are not accepted by the 1-Point RANSAC are not used by the EKF to estimate pose.

2.3 Low and High Innovation features

EKFmonoSlam distinguishes between Low and High Innovation points in pose estimation. Low Innovation points are those that have correlated well with the RANSAC model while High Innovation features are those that did not correlate as well with the model but are still probably not outliers. Typically, Low Innovation points are those that are close by hence only a few centimeters of travel are required to notice a parallax. High Innovation points are those that occur close to the horizon and are incredibly far away. Such points require position change of several meters or even kilometers before parallax is noticed. Hence Low Innovation points are valuable for estimating displacement while High Innovation points are required for estimation of angular displacement.

3 Results

3.1 Original Results

The algorithm was run on a sequence of images from a lunar simulation. The path that was simulated was a circle with radius 15m. The algorithm found craters using the contrast provided by their shadows, and also detected and tracked features on the horizon. Fig. 1 shows the algorithm detecting and tracking features. Using the feature detection, the algorithm estimated how the camera moved and calculated the position of the rover. Fig. 2 is the plot of the
rover's position as calculated by EKFMonoSLAM. In Fig. 2, the deviation from ground truth is apparent. The estimation of the rover's position starts off accurate and slowly grows inaccurate. The final position is 49m off the expected answer. The total distance traveled was calculated to be 128m instead of the known 94m, and the percent error was 36%.

3.2 Feature Deletion Results

One of the more time intensive steps of SLAM is feature correlation. In order to improve the computational time, the number of features stored was decreased by removing features that did not correlate after a certain number of consecutive frames. Fig. 3 shows the outputs for setting the threshold for the number of consecutive frames to one, three, and five. The figure indicates that a threshold value of three is the most accurate. Table 1 shows the outputs of the accuracy for each threshold value tested by comparing the difference in distance between the calculated final position of the rover and the known final position. Again, a threshold value of three is shown to be the most accurate.

3.3 Results after Modifications

Removing features after they were not correlated after three frames optimized accuracy. It also decreased the runtime of the algorithm because each new feature detected was compared fewer times. The runtime improved from over 2.5 hours to 80 minutes. The final output was 20m off from the known position of the robot, which is more accurate than the original algorithm. The total distance traveled was calculated to be 101m instead of 94m, and the percent error was 7%.

3.4 Library for Visual Odometry 2 Results

For comparison, the Library for Visual Odometry 2 (LibVISO2) stereo SLAM algorithm was run on the same lunar simulation as EKFMonoSLAM. Fig. 4 shows the final position calculations of the algorithm. Unlike EKFMonoSLAM, LibVISO2 predicts that the rover traveled in a complete circle. However, its prediction is inaccurate because the rover keeps traveling beyond the start point. The final calculated position was 25m off from the known position of the robot. The total distance traveled was calculated to be 132m instead of the known 94m, and the percent error was 40%.

4 Conclusion

EKFMonoSLAM runs on the entire simulated data set. It is able to detect and track features between consecutive images. It is also able to accurately determine the movement of the robot using image correlation techniques to within 7% error. The modified implementation is two times faster than original implementation and improves accuracy from 36% to 7%. Also, the EKFMonoSLAM algorithm is also more accurate than the LibVISO2 algorithm, which has 40% error.

The accuracy of the algorithm is dependent on several factors including the simulated data. EKFMonoSLAM detects features based on contrast in the image; therefore, it picks up shadows from rocks and craters easily. However, as the rover moves, the shadows in the image move as well. Thus, between consecutive images, changes in feature position lead to errors in image correlation and pose estimation. Also, as the lunar simulation improves, the
results of the algorithm will change as well. Getting an accurate lunar simulation, especially accurate color and lighting details, will affect feature detection and the output of SLAM algorithms.

This research demonstrates that, with current lunar simulations, EKFMonoSLAM is an accurate means for rover localization. The algorithm is able to detect how the rover moves and track the distance that it has traveled. This research also demonstrates that the modified EKFMonoSLAM algorithm is more accurate than the LibVISO2 and original EKFMonoSLAM algorithms, and is faster than the original EKFMonoSLAM algorithm.

5 Future Work

5.1 Combining Left and Right Stereo Images

EKFMonoSLAM currently takes as an input one sequence of images for SLAM. A planetary rover, however, might have stereo vision, meaning that it has both a left and right set of images for each point in time. The algorithm does not support combining the data from both sets of images and using the aggregate data for SLAM. Combining data will lead to redundancy in calculations that should help with accurate feature tracking and correlation. For example, the algorithm can check consistency between feature correlation calculations between the right and left images. If the calculations are consistent, the probability that the feature was correctly correlated increases. If the calculations are inconsistent, the correlation is probably coincidental. However, since combining data from left and right cameras requires processing more images, the described algorithm will be more computationally expensive than one that runs only on the right or left set.

5.2 Testing Pieces of Algorithms on Data Set and Determining best methods

SLAM algorithms can have different ways of implementing the four general steps. For example, feature detection can be done using Speeded Up Robust Feature (SURF) detection or Scale-invariant feature transform (SIFT). Future work on determining the optimal SLAM algorithm for planetary data would include testing these smaller implementations on the same data set and comparing their output. One of the challenges of this task is determining an objective function for comparing outputs. Algorithms can output data that are incompatible and therefore difficult to compare, and the notion of what is “optimal” is not well defined. However, if this challenge is overcome, the “optimal” implementations of the parts of SLAM can be strung together, which should create a SLAM algorithm that is best suited for planetary data and applications.
Figures and Tables:

**Figure 1:** An image of EKF MonoSLAM output. It is the input image enhanced with ovals that represent features that have been detected. Red ovals are features that have been matched, while blue ovals are those features for which matches have not been found.
Figure 2: Final output of EKFMonoSLAM run without any modifications. The black line is the calculated trajectory of the robot, while the green circle is the known path. The mapping output has been suppressed for image clarity.
Figure 3: Graph of the output of the algorithm when setting the threshold value for the number of consecutive images that a feature can miss before it is deleted to one, three, and five compared with ground truth.
**Figure 4:** Final output of LibVIS02. The red x’s are the calculated position of the robot at each interval. The red line is the trajectory of the robot calculated using the position estimates.

<table>
<thead>
<tr>
<th>Threshold Value</th>
<th>Distance from rover’s calculated position to actual position (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.9</td>
</tr>
<tr>
<td>3</td>
<td>20.1</td>
</tr>
<tr>
<td>5</td>
<td>47.8</td>
</tr>
</tbody>
</table>

**Table 1:** Table used to compare effectiveness of threshold values for testing modifications to EKFMonoSLAM. For each threshold value, the resulting error in meters between calculated final position of the rover and the known final position was measured.