FastSLAM with Stereo Vision

Rounak Mehta Sudhanva Sreesha Frederick Wirth

***Abstract*— Simultaneous localization and mapping (SLAM) is one of the most fundamental problems in mobile robotics. It's necessary when the robot does not have access to a map of the environment and it does not know its own pose in the world. This paper outlines a method of using the Speeded Up Robust Feature (SURF) detector on stereo images to determine the key features of a map and the robot's position with respect to those features. Rather than using every single SURF feature detected in a pair stereo images, we only select the most distinguishable and stable features in order to increase the speed and accuracy of the data association of SLAM. We have tested our implementation on an outdoor environment and have found that results varied depending on the environment.**

I. INTRODUCTION

SLAM is one of the most fundamental problems in mobile robotics and it arises when the robot neither has access to a map of the environment nor does it know its own pose in the world. Instead, the robot must orient itself given its observations $z_{1:t}$ and motion commands $u_{1:t}$. In SLAM, the robot incrementally acquires a map of its environment while simultaneously localizing itself relative to this map.

Building a map of the environment is one of the most fundamental tasks for autonomous mobile robots. Building a map allows the robot to perform higher level tasks including exploration and path planning. Since it is imperative for the robot to build a map of the environment and to localize itself in that environment to accomplish true autonomy, SLAM has received significant attention over the past two decades. In our case, we are mapping a variety of outdoor environments so the map of the environment will be classified by the surrounding landmarks, such as trees, road signs, and other stationary objects.

There exists a variety of methods to solve the SLAM problem, such using an Extended Kalman Filter (EKF) or a particle filter, and they all have different strengths and weaknesses. We have decided to use FastSLAM as our underlying SLAM implementation, because of its ability to scale well with a large number of landmarks [5].

Every implementation of SLAM requires a method to obtain observations of the surrounding environment. Some techniques for obtaining observations from an environment are using LIDAR, sonar, or a camera. We have chosen to use a camera because we believe that Vision-Based SLAM has several advantages to other types of SLAM. Cameras are generally cheaper than LIDAR and sonar equipment and they can still provide 3D information about a scene. Vision-Based observations are also intuitive for people to understand because people visually map out an environment based on the points of interest that they see.

A. Previous Work

We based our method on a previous paper by A. Gil et Al. [3]. They were able to successfully perform FastSLAM using stereo cameras to obtain the observations. They found that their method was successful and they could perform loop closures properly. Their method is outlined below:

- Use SIFT feature detector by Lowe [4] to obtain features
- Reject features that are not distinguishable or not stable
- Use these features as the landmarks in localization
- Match new features to existing landmarks using nearest neighbor data association

The innovation of this paper comes from the rejection step since they will track visual landmarks over several frames and only select features that stay stable over those frame. They found that including the rejection step allowed them to use fewer particles and still achieve good results. The authors' biggest concern with their method is that using a feature detector is ineffective in areas with a low amount of features, such as blank walls or a long hallway. Although that is still a concern for our implementation, we have found that outdoor images tend to have a lot more distinguishable features, although there are some exceptions, such as a long path with a several similar looking objects, like trees.

B. The KITTI Vision Benchmark Suite

We used the KITTI Vision Benchmark Suite [2] as our primary dataset. This dataset was creating by driving an autonomous driving platform around several environments, including city, residential, road and campus environments. We chose this dataset because it is well documented and provides high quality odometry and stereo images. The dataset provides synced and rectified gray-scale stereo images, accompanying 3D GPU/IMU data, and the necessary tools to use this dataset.

C. Contribution

Our contribution to this topic is testing the method detailed by A. Gil et Al. [3] on an outdoor dataset. Due to the emerging research and development in the field of autonomous vehicles, we believed that this technique could be adapted to be used with a real time autonomous vehicle. Since autonomous vehicles are a real time system, we needed to focus on techniques that would improve the speed of this method. Some of the changes we have implemented to improve the speed of this method are using a SURF feature detector [1] instead of a SIFT feature detector and limiting the features used as our observations to only the "strongest" features.

Outdoor datasets tend to be more challenging than indoor datasets because they are generally less controlled. Outdoor datasets are less controlled because they tend to have variable lighting conditions, there is a potential for either large open areas or densely packed areas, and there is a greater variety in the types of environments to be examined. In addition to the extra challenges from using an outdoor dataset, there are also some concerns with using an automotive dataset. Our biggest concern is the possibility of similar looking environments. Since road structure, street signs, and street lamps tend to be standardized, it is possible that they will have very similar SURF features.

II. METHODS & IMPLEMENTATION DETAILS

A. Rao-Blackwellized Particle Filter SLAM

At the outset, the predominant approach to the SLAM problem was first introduced by Smith, Self, and Cheesemen [7]. In their seminal paper, they proposed using an Extended Kalman Filter (EKF) in order to incrementally estimate the robot pose and the locations of all the landmarks. Although this method is widely accepted in the field of robotics, it still suffers from a major pitfall: the time complexity involved in updating its map. Since the Kalman Filter keeps track of a combined covariance matrix which has a quadratic space complexity, the measurement update step is time quadratic in the number of landmarks N. The large time complexity limits the use of the EKF SLAM solution to cases with only a few hundred landmarks. It quickly fails to handle the landmarks efficiently in natural environments where potentially millions of landmarks are created.

Therefore, this section is intended to discuss the *particle filter* approach to the SLAM problem. First of all, particle filters are at the core of some of the most effective robotics algorithms today. However, particles filters scale exponentially because they suffer from the curse of high dimensionality. A straightforward approach of the particle filter is prone to failure because of the large number of variables used to represent the map for each particle.

We can apply a modified version of the particle filters, known as the *Rao-Blackwellized particle filters*, to the full SLAM problem. This version of the particle filter exploits the conditional independence between any two disjoint sets of features in the map, given the full robot pose. The dependencies arise only through the uncertainties of the robot pose, but we can estimate the location of all the features in the map, independently of each other, given the robot poses.

We decided to use the *FastSLAM 1.0*, as detailed by Montemerlo et Al. [5], as our underlying SLAM implementation. This solution to the SLAM problem recursively estimates the full posterior distribution over the robot pose and the landmark locations, and scales linearly with the number number of landmarks in the map. Since the individual map errors are conditionally independent, the mapping part of the solution is factored into separate problems, one for each landmark in the map. At the core of the algorithm, each of the landmarks are localized using separate low-dimensional EKFs.

Since the underlying algorithm, in our case, is implemented in linear time in the number of features, our implementation offers computational advantages over other SLAM algorithms. Additionally, its ability to maintain data associations for each particle significantly improves its robustness in dealing with data association problems.

There is a second version of the FastSLAM, known as *FastSLAM 2.0*, which addresses the problem when the accuracy of control is low relative to the accuracy of the robot's sensors. This version is mostly equivalent to the first version, but with one key difference: the proposal distribution now takes into account both the control, u_t, and the measurement, z_t, while sampling the robot pose from the previous time step to the current time step [6]. Since the control noise was not worse than the sensor noise in the dataset that we utilized while experimenting our algorithm, we decided to use FastSLAM 1.0 instead.

First of all, the set of M particles in FastSLAM 1.0 is described as follows:

$$\mathbf{Y_t} = \left\langle Y_t^{[1]}, Y_t^{[2]}, ..., Y_t^{[M]} \right\rangle \tag{1}$$

Each of these particles consists of a path estimate and a set of estimators of individual landmark locations with corresponding covariances. Likewise, the k^{th} particle is denoted:

$$Y_t^{[k]} = \left\langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \mu_{2,t}^{[k]}, \Sigma_{2,t}^{[k]}, ..., \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \right\rangle \tag{2}$$

In the above equation, the k^{th} particle is represented by the bracketed notation $[k]$, the estimate of the robot path is represented by $x_t^{[k]}$, and the estimated location and covariance of the n^{th} landmark is represented by $\mu_{n,t}^{[k]}$ and $\Sigma_{n,t}^{[k]}$, respectively.

The key mathematical insight on FastSLAM pertains to the fact that the full SLAM posterior, given by $p(y_{1:t}|u_{1:t}, z_{1:t}, c_{1:t})$ can be factored as follows:

$$\begin{aligned} p(y_{1:t}|u_{1:t}, z_{1:t}, c_{1:t}) &= p(x_{1:t}|u_{1:t}, z_{1:t}, c_{1:t}) \\ &\quad \times \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \end{aligned} \tag{3}$$

Moreover, computing the posterior over the particles and landmarks can be decomposed in $N + 1$ probabilities.

B. Motion Model

FastSLAM 1.0 uses the control, u_t, to sample the new robot pose, x_t, for each particle. Moreover, FastSLAM 1.0 samples the pose x_t of the k^{th} particle by drawing a sample from the motion posterior as follows:

$$x_t^{[k]} \sim p(x_t|x_{t-1}^{[k]}, u_t) \tag{4}$$

The posterior estimate of the k^{th} particle at time $t-1$ is given by $x_{t-1}^{[k]}$, at time t is given by $x_t^{[k]}$, and the control input at time t is given by u_t.

We used the velocity motion as detailed on page 124 in Probabilistic Robotics [8]. The motion model contains the translational velocity of the car, denoted by v_t at time t, and the rotational velocity, denoted by ω_t, also at time t. The KITTI dataset provided 3D GPS/IMU data which included location, speed, and acceleration. We used the forward velocity of the car and the angle around the upward axis as our v_t and ω_t respectively. Since the robot motion is subject to noise, the actual velocities differ from the measured ones. Therefore, we modeled this difference by a zero-centered Gaussian random variable with a finite variance. We denote these noisy velocities as $\hat{v_t}$ and $\hat{\omega_t}$. We measured the process noise by finding a section of the dataset where the car was remaining perfectly still. Since the car was not moving, we can model the car as a zero mean Gaussian and calculate the variance based on the difference in measurements.

The noisy motion model from the previous time step, $\mathbf{x}$, to the robot pose at the current time step, denoted by $\mathbf{x}'$ is as follows:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v_t}}{\hat{\omega_t}} sin(\theta) + \frac{\hat{v_t}}{\hat{\omega_t}} sin(\theta + \hat{\omega_t}\Delta t) \\ \frac{\hat{v_t}}{\hat{\omega_t}} cos(\theta) - \frac{\hat{v_t}}{\hat{\omega_t}} cos(\theta + \hat{\omega_t}\Delta t) \\ \hat{\omega_t}\Delta t \end{pmatrix} \quad (5)$$

Furthermore, we used the velocities in the x and y directions, available from the GPS, as the ground truth for the robot.

C. Visual Landmarks

The Speeded-Up Robust Features, as detailed by Bay et Al. in [1], is a scale and rotation invariant feature detector. This was originally developed for applications including object recognition, classification, registration, and 3D reconstruction. SURF features are represented as a 64-dimensional vector and, according to its authors, they are said to outperform existing methods based on repeatability, robustness and distinctiveness of the descriptors.

Given a pair of stereo images, denoted by (I_t^L, I_t^R), from the left and the right cameras at time t, we use the SURF feature extractor to obtain landmarks which correspond to points in the 3-dimensional camera space. We then convert these points to extract the locations of landmarks in the robot's reference frame and use these in our observation model.

D. Observation Model

The observation model consisted of three major steps. (1) At every time step, the camera images ran through an accumulator which tracked the most distinguishable and most stable features across five time steps. These features were outputted at the fifth time step into the next module. (2) Given those distinguishable and stable features, the module returned the 2D locations of those features in the robot's reference frame as observations into the correction step. (3) The correction step, also ran every fifth time step, used these observations to perform data association, update existing landmarks, initialize new landmarks, and resample particles.

1) Accumulator: The best SURF features to pick are the ones that are stable and distinguishable. Our definition of stable features are those that remain visible to the stereo cameras for five consecutive time steps. To ensure that features are stable, we implemented an accumulator module.

For the first time step, the accumulator stores only the 'k' strongest or most distinguishable SURF features extracted from the left camera's image. For time steps two through five, the accumulator compares the previously stored stable features with the features of the left camera's image from the current time step; it removes all of the previously stored stable features that do not match with any of the current features. Since this tracks only the stable SURF features, it reduces the amount of observations to, at most, a fifth of the total number of possible features. We want to avoid using unstable features because they can create additional landmarks that are difficult to associate with.

One of the major advantages of the accumulator is that, by limiting the number of features we observe, it decreases the overall overhead necessary to store landmarks.

2) Compute the depth map: Once we extract the most distinguishable and stable features, we need to compute the positions of these features in the robot's reference frame. The first step in this procedure is to compute a depth map based on the features. In order to compute the depth map, we first match the stable features between the left and the right stereo images.

The matched points are then used to calculate the disparity of the points, in pixel units, between the left and the right images. Given that the two stereo images have already been rectified, the matches should have the same y-pixel coordinate. We can calculate disparity using:

$$disparity_j = l_j - r_j \quad (6)$$

The l_j and r_j represent the x-coordinates of the j^{th} matched point from the left and right images, respectively. The features are then passed through a band-pass filter to reject features that are too far or too close to the stereo rig that they should be classified as mismatches. Once the image disparity has been computed, the depth of the image can be found by using the camera's intrinsic parameters: baseline distance, the distance between the centers of the left and the right cameras, denoted by B, and the focal length, denoted by f.

$$depth = \frac{B * f}{disparity} \quad (7)$$

3) Convert pixel coordinates to camera world coordinates: Once the depth of the feature is obtained, the location of the features, in pixel units, is combined with the depth to compute the position of the features in the camera's world reference frame.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = K^{-1} \times depth \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad (8)$$

where K is the camera calibration matrix provided by the makers of the KITTI dataset.

4) Convert camera world coordinates to robot coordinates: Once we compute the coordinates of the features in the camera's world reference frame, we use the homogeneous transformation matrix, provided by the makers of the KITTI dataset, to convert these to the robot's coordinate frame which is at the origin of the 3D GPS/IMU sensor. Since we are running the dataset in a 2D world to simplify the algorithm, we use a projection matrix, denoted by P_1, to project the 3D coordinates to 2D.

$$\begin{bmatrix} z_{t,x} \\ z_{t,y} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ({}^{r}_{c}H) \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{9}$$

Here, X represents the forward direction, Y represents the left direction, and Z represents the upward direction, all in the camera's world reference frame.

E. Update Step

The update/correction step iterates over all the particles performing data association, updating existing landmarks, and initializing new landmarks, using the observations from the observation model.

1) Map management: According to FastSLAM 1.0, the map M is represented as a collection of N landmarks, $M = (m_1, , m_2, ..., m_N)$. Each landmark is described as: $m_n = (\mu_n, \Sigma_n, d_n)$, where $\mu_n = (x_n, y_n)$ is a vector describing the position of the landmark in the global reference frame and Σ_n is the covariance matrix associated with the landmark position. We also include the 64-dimensional SURF descriptor d_n to help differentiate between landmarks.

2) Data association: While the robot moves through the environment, it must decide whether the observation $z_{t,k}$, the k^{th} observation at time step t, corresponds to a previously mapped landmark or to a new landmark.

The first part of the data association process is to compare the Mahalanobis distances of the actual observations to the expected observations for each landmark in each particle. If the Mahalanobis distance is under the 3σ bound, we add the observation to the particle's landmark's bin.

We then go through each of the bin's observations to compare the square of the Euclidean distance between the SURF descriptors of the observations and the landmark whose bin it is in. For each landmark bin we associate the landmark with the observation for which this value is the lowest and under a certain threshold. For the unmatched observations, we hypothesize that these are of new landmarks and that we will need to initialize them.

3) Update landmarks: FastSLAM 1.0 updates the posterior over the feature estimates, represented by the mean $\mu^{[k]}_{n,t-1}$ and the covariance $\Sigma^{[k]}_{n,t-1}$. The exact update equation depends on whether or not a feature m_n was said to have been observed by the data association layer. The landmarks that were not observed in the current time step are not modified in the map. However, the following measurement function was used in the observation model to update existing landmarks that were hypothesized to be seen by the data association layer.

$$h(\mu^{[k]}_{n,t-1}, x^{[k]}_t) = \begin{bmatrix} cos(\theta^{[k]}_t) & -sin(\theta^{[k]}_t) \\ sin(\theta^{[k]}_t) & cos(\theta^{[k]}_t) \end{bmatrix} \begin{bmatrix} \mu^{[k]}_{n,t-1,x} - x^{[k]}_t \\ \mu^{[k]}_{n,t-1,y} - y^{[k]}_t \end{bmatrix} \tag{10}$$

In order to update the covariance of the landmark position, the measurement noise had to be converted from the pixel units, which is what the noise was initialized in terms of, to the global reference frame. The covariance of the zero-mean Gaussian random variable that models the measurement noise in pixels is as follows:

$$\Sigma_p = \begin{bmatrix} \sigma^2_{x'} & 0 & 0 \\ 0 & \sigma^2_{y'} & 0 \\ 0 & 0 & \sigma^2_{disparity} \end{bmatrix} \tag{11}$$

After a series of transformations illustrated in 12, we compute the covariance in terms of the landmark location in the world.

$$\Sigma_w = ({}^{w}_{r}R)(P_1)({}^{r}_{c}R)(A)(\Sigma_p)[({}^{w}_{r}R)(P_1)({}^{r}_{c}R)(A)]^T \tag{12}$$

where

$$A = K^{-1} \times depth \times \begin{bmatrix} 1 & 0 & -\frac{x}{disparity} \\ 0 & 1 & -\frac{y}{disparity} \\ 0 & 0 & -\frac{1}{disparity} \end{bmatrix} \tag{13}$$

Once the noise covariance is estimated, we used the regular update step using the Jacobian of the measurement function to compute the innovation and the Kalman gain to eventually update the covariance matrix of the landmark location. One thing to note here is that the descriptor of the landmark is never modified.

4) Initialize new landmarks: In order to create a new landmark that was never seen before, we used the following inverse observation model.

$$h^{-1}(z_t, x^{[k]}_t) = \begin{pmatrix} x^{[k]}_t + z_{t,x}cos(\theta^{[k]}_t) - z_{t,y}sin(\theta^{[k]}_t) \\ y^{[k]}_t + z_{t,x}sin(\theta^{[k]}_t) + z_{t,y}cos(\theta^{[k]}_t) \end{pmatrix} \tag{14}$$

We then computed the covariance matrix of the landmark using same steps described in FastSLAM 1.0, using the inverse of the Jacobian of the inverse observation model with respect to the observation and the noise from 12. Lastly, the descriptor, d_n was set to the mean of the descriptors of the stable features seen across the last five time steps.

F. Resample

Once we have updated the positions of all of our particles, we give each particle an importance weight, and obtain a new set of particles by sampling from the previous set based on their importance weights. The importance weight

of each particle is assigned based on how confident we are in the particle's position. The benefit of resampling is that the next iteration of the particle filter will be more likely to use particles that we are more confident in. We use the $lowVarianceSampler$ as detailed on page 110 in Probabilistic Robotics [8].

III. RESULTS & ANALYSIS

A. Experimental Datasets

The KITTI Vision Benchmark Suite offers a wide variety of environments for us to test our implementation on. We tested a variety of these datasets and narrowed them down to five major ones that we wanted to test. In all of our plots, the blue line represents the ground truth, the red lines represent the path of the particles, and the green dots represent the landmarks for each particle. The units for the x and y axes are plotted in meters. All of these tests use 10 particles unless otherwise noted.

1) Stationary Car with Moving Landmarks: Test Case `2011_09_26_drive_0018` shows the car starts stationary at an intersection and then starts turning left. Number of particles used was set to 10. The output of this test is shown in Figure 1. We used this test because the car was stationary at the start. This allowed the car to be very confident about the stationary landmarks, such as street signs. Once the car started moving, it could localize itself based on those landmarks. This test also had several cars that drive alongside our car, and the accumulator successfully filters out the features from those cars, because they are not stable.

2) Moving Car with Stationary Landmarks: Test Case `2011_09_26_drive_0035` shows the car moving through a densely packed environment with buildings and stationary cars. The output of this test is shown in Figure 2. We used this test because the environment contains no moving objects. This allowed us to verify that we track stable landmarks even when the car is moving. The particles started to diverge at the beginning, but those that diverged were rejected during resampling and those that were closer to ground truth continued to exist.

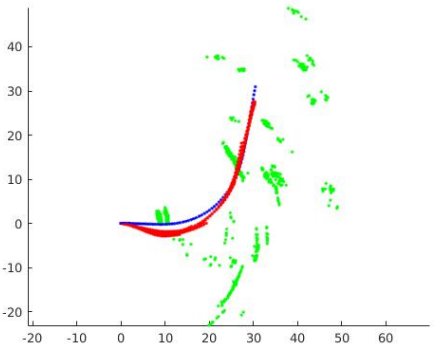

Fig. 1. This environment contained an open road with moving cars around our car. Our car was initially stationary, so it became very confident about the location of several stationary landmarks. KITTI Dataset: `2011_09_26_drive_0018`

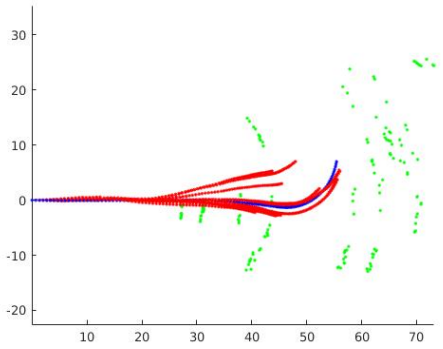

Fig. 2. This environment was densely packed with buildings and stationary cars. Our method could filter out the particles that diverged and resample to the proper particles. KITTI Dataset: `2011_09_26_drive_0035`

3) Moving Car with Moving Landmarks: Test Case `2011_09_26_drive_0091` shows the car driving down a street with multiple buildings on the right and a few points of interest on the left. Our method performs substantially well due to the presence of abundant landmarks having distinct features. The number of particles used in this test was 25. The output of this test is shown in Figure 3.

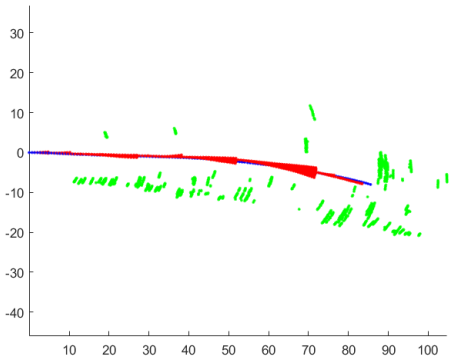

Fig. 3. This environment contained a road with buildings on the right and a few features on the left. KITTI Dataset: `2011_09_26_drive_0091`

4) Moving Car with Sparse Landmarks: Test Case `2011_09_26_drive_0028` shows the car driving down a road with dense foliage on the right and left. Likewise, the features are indistinguishable and we get much fewer features from the accumulator. Hence, the number of frames to track the stable SURF features for was reduced from five frames to two frames. The number of particles used in this test was 25. The output of this test is shown in Figure 4.

B. Number of Features

As the number of accumulator frames are reduced, the number of features registered increases significantly 5, without a significant effect on the error as seen in Figure 6.

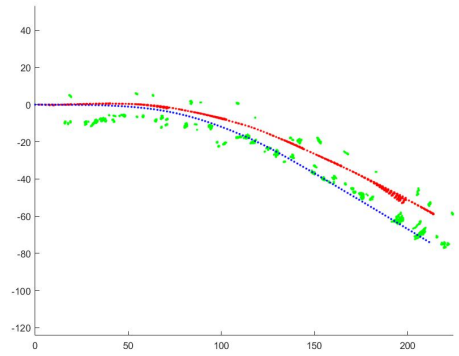

Fig. 4. This environment contained a road with dense foliage on both the right and left side. KITTI Dataset: `2011_09_26_drive_0028`

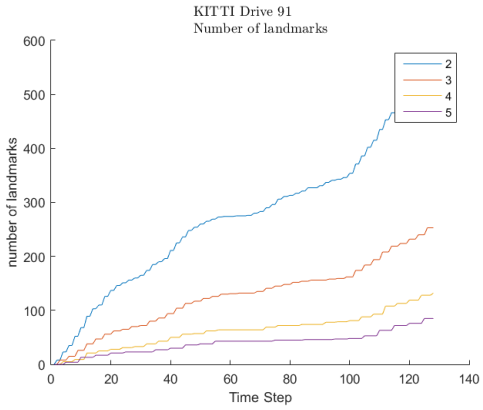

Fig. 5. The number of features increases significantly with small accumulator frame numbers. KITTI Dataset: `2011_09_26_drive_0091`

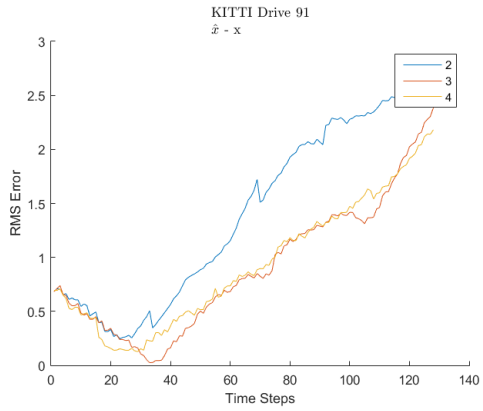

Fig. 6. Effect on RMS error due to number of accumulator frames. KITTI Dataset: `2011_09_26_drive_0091`

IV. Conclusion and Future Work

In this paper, we adapted the solution from [3] to work with an outdoor dataset. We found that this method is not robust enough to work with all types of outdoor environments. Based on the drives from the KITTI dataset, we found that our method worked best when traveling through closed areas, such as areas with a lot of surrounding cars and buildings, whereas our algorithm did not work well while traveling through open areas, such as suburban roads or highways. This is primarily due to the closed areas being feature rich. Our biggest concern with this implementation is that the KITTI dataset only contained open loop data, so we were unable to test this on a data set which has evident loop closures. Since the FastSLAM algorithm tends to only converge once a loop closure is spotted, we could not test the convergence functionality. This method is also infeasible in real time systems because the data association layer takes too long to complete.

One future improvement to this implementation is to ignore objects that are moving. One method is to use the information from the prediction step. We estimate how the pixels of stationary objects should move in the camera frame, then only use the pixels whose matches lie in this 3σ ellipse in the next time step. When driving down the street, there is a possibility for there to be objects such as cars, people or animals that are stationary while in our camera frame but move away at some later point. Our current implementation would initialize that object as a landmark, and then expect to see the object again if it ever goes back to that position. One way to solve this is to use a pre-trained convolutional neural network to create bounding boxes around cars and pedestrians and ignore the SURF descriptors within these.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. *SURF: Speeded-Up Robust Features*. 2006.

[2] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[3] A. Gil, O. Reinoso, O. M. Mozos, C. Stachniss, and W. Burgard. Improving data association in vision-based slam. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2076–2081, Oct 2006.

[4] David Lowe. Object recognition from local scale-invariant features. pages 1150–1157, 1999.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[7] Randall Smith, Matthew Self, and Peter Cheeseman. A stochastic map for uncertain spatial relationships, 1991.

[8] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.