Implementing Localization and Mapping

Divya Thuremella, Samhita Karnati, Roopa Ramanujam

16 January 2018

1 Introduction

Simultaneous Localization And Mapping (SLAM) is an optimization problem that has been around for many years. The basic principle of the problem involves estimating the location of a vehicle (such as a robot) while simultaneously mapping its surroundings. This is usually referred to as a "chicken and egg" problem because accomplishing one of these tasks helps the other perform better. Estimating the position of the vehicle is made more accurate by incorporating information about its surroundings, and the positions of its surroundings are calculated more accurately when knowledge of the vehicle's position is more known with certainty. This is an important problem because it has many current real-world applications such as in autonomous vehicles and underwater and space exploration, where these vehicles need to chart accurate paths while mapping their surroundings using different sensors. Visual SLAM is an approach that is especially of interest because it is one of the most cost effective methods, since it only requires a camera. This project was inspired by the SLAM optimization problem, but implements localization and mapping separately without the feedback loop between them.

2 Related Work

Much work has been done in the interest of optimizing SLAM. In "Large-Scale Direct SLAM with Stereo Cameras," Engel, Stuckler, and Cremers proposed an approach that does not involve interest point based methods, but instead aligns stereo images based on pixels of high contrast. They tested the method on the KITTI and EuRoC data sets with excellent results. In "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," Mur-Artal and Tardos implemented a similar system for monocular, stereo, and RGB-D cameras, and evaluated it on the KITTI data set. Finally, in "Stereo Parallel Tracking and Mapping for robot localization," Pire et al. created a SLAM system with stereo that specifically focused on the parallel nature of the problem, in order to be able to work in a large-scale setting. They found that their method was highly accurate, tested with the KITTI dataset.

3 Datasets

For this project, we used two datasets: the Malaga Campus RT dataset and the KITTI raw dataset. We used these datasets on the recommendation of our adviser and because they both have both stereo images as well as ground truths that would allow for quantitative analysis.

1. Malaga dataset
   The Malaga Campus RT dataset includes stereo images (from left and right cameras) at 7.5 fps. Fig. 1 shows an example of the included stereo images.

Figure 1: *Left and right stereo images from the Malaga Campus RT dataset at timestamp 1226226123.*

   The images are from a drive down a relatively empty road (therefore the "world" objects around the car are stationary). It is important to note that the road that the car drives on has many twists and turns; in fact, the first thing the car does is take a left turn. This dataset also includes GPS ground truths that have associated covariance matrices, but these ground truths don't exist for all frames.

2. KITTI dataset
   The KITTI dataset we used is one of their raw data visual benchmarks. We downloaded the synced and rectified stereo images as well as their system's calibration information. Fig. 2 shows an example of the KITTI stereo images.

Figure 2: *Left and right stereo images from the KITTI dataset at timestamp 0000000001.*

Like the Malaga dataset, the KITTI dataset was taken on a drive where there were almost no other moving objects. One important difference is that the path of the car was relatively straight. Although the road itself exhibits a slight curve, there were no turns made by the car. The KITTI dataset also includes GPS ground truths (without covariances) and velodyne ground truths. The GPS ground truths exist for each frame taken.

4 System Design and Implementation

The goal of our project was to implement a system that performed localization and mapping given a sequence of sets of stereo images. See Appendix 1 for descriptions of each file that we wrote to make this system.

4.1 Extracting features

In order to extract features from the images, we used the MATLAB `detectSURFFeatures` function. After extracting SIFT features from the left and right frames of the same timestamp, we used the `matchFeatures` function to find the features that existed in both frames. Fig. 3 shows an example of matched left and right frame features.

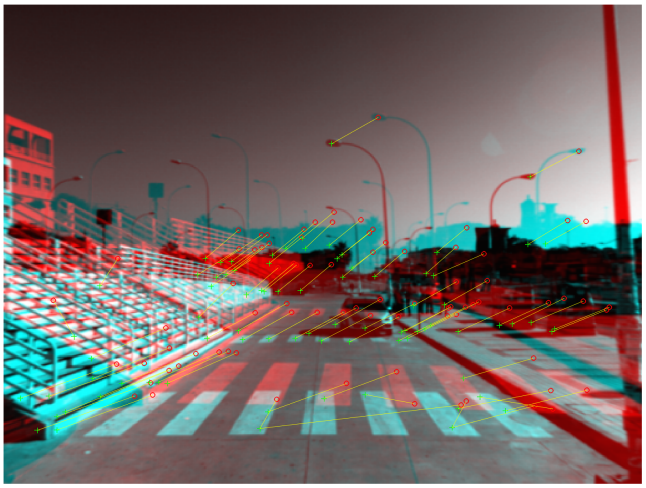

Figure 3: *Example of matched left and right feature points from the Malaga dataset. Blue indicates left and red indicates right. Green lines indicate which features were matched.*

We repeated the same steps to extract features from the subsequent set of stereo images in the consecutive frame.

Next, we matched feature points across consecutive frames, again using the `matchFeatures` function. Fig. 4 shows an example of feature points matched across two consecutive timesteps as seen from the left camera.

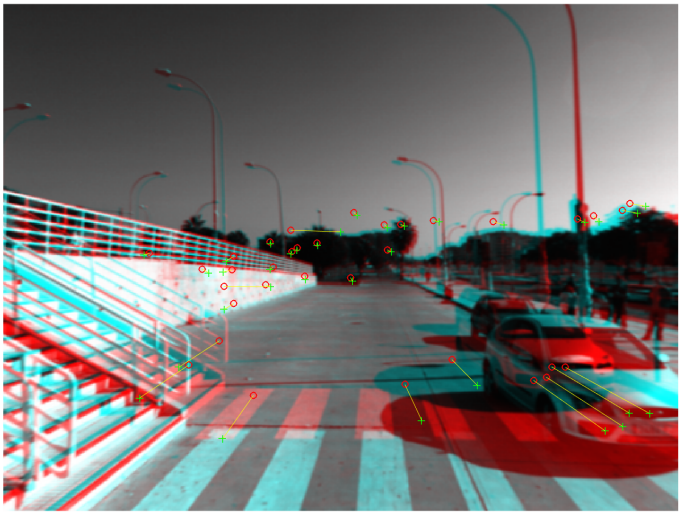

Figure 4: *Example of matched features across two consecutive timesteps.*

4.2 Going into 3D coordinates

The next task was to project these 2D image coordinates into 3D world coordinates. This involves calibrating the cameras in order to triangulate the 3D points. To do so, we first estimated the fundamental matrix F using the `estimateFundamentalMatrix` function, specifying that it use RANSAC. The input to this function was the set of points that were common between the left and right cameras, the number of iterations to be used in RANSAC, and the threshold to be used in RANSAC. Since we were using two frames, each with a set of points that were common between the left and right camera, we decided to calculate the fundamental matrix twice, once for each consecutive frame, and compare them to make sure we were getting a similar answer from both. At first, the two consecutive frames yielded two different fundamental matrices, which seemed like a problem. However, when we increased the number of iterations to be used in RANSAC, the fundamental matrices calculated by each consecutive frame looked similar to each other. Therefore, we concluded the RANSAC must have gone through enough iterations to be convergent. For threshold, we used the default value provided. Next, we specified the camera parameters (`cameraParameters` is an object type that we could specify in MATLAB) using the intrinsic matrices (K1 and K2) for each of the cameras. An intrinsic matrix K takes the form

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are the focal length of the camera, x_0 and y_0 are the principal point offsets, and s is the skew. All of these parameters for each dataset could be respectively found in the Malaga dataset paper and the KITTI calibration files.

Next, we calculated the relative camera pose (rotation matrix and translation vector) by using the function `relativeCameraPose`. It takes in the fundamental matrix, left camera parameters, right camera parameters, and the set of matched points between the two cameras. Finally, we used the `triangulate` function to determine the 3D projection of our 2D image coordinates. Fig. 5 shows an example of the output of the triangulate function.

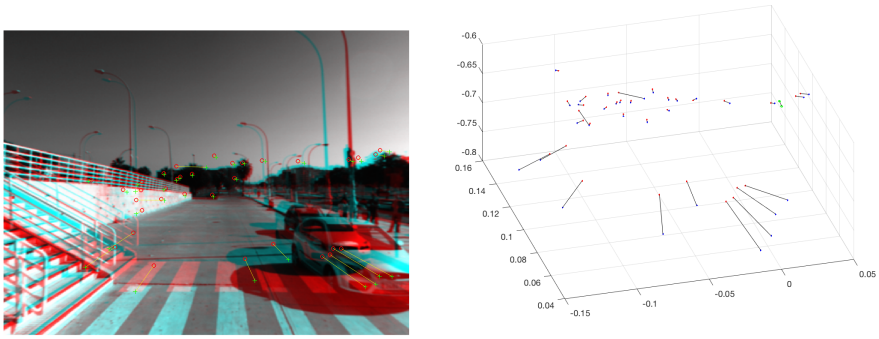

Figure 5: *The left image shows the same as Fig. 4 and the right image shows the same 2D image points plotted in 3D coordinates.*

4.3 Implementing RANSAC to calculate the 3D affine transformation

The next task was to determine the 3D affine transform from one 3D world coordinate to the next. After searching, we realized that MATLAB only has a 2D affine transform estimation function and that the 3D version only exists in the OpenCV library, which we had been unsuccessful in installing previously. Thus, we thought that this would be a great opportunity to write from scratch an algorithm that we had learned in class.

We created a function called `findAffine3D` that performs RANSAC to estimate the 3D affine transform. It takes in as parameters both sets of 3D points, the number of iterations to be performed in RANSAC, and the error threshold.

To calculate the 3D affine transform, we followed these steps:

1. For the number of iterations specified, we first randomly chose 4 pairs of points (4 3D points from timestep 1, and the corresponding 4 3D points from timestep 2) and converted them into homogenous coordinates by concatenating a fourth column of ones to the three-column vector of points.

2. Then, we calculated the transform that took the points from timestep 1 to timestep 2 by the equation $3Dpoints_1 * T = 3Dpoints_2$, where $3Dpoints_1$ and $3Dpoints_2$ are 4 corresponding 3D points (in homogenous coordinates) from timestep 1 and timestep 2 respectively, and T is the transform that takes the points from one timestep to the next. This equation can be solved by calculating $T = 3Dpoints_1^{-1} * 3Dpoints_2$.

3. Taking the transform calculated from the 4 corresponding points, we apply the transform to each point in the set of all 3D points for timestep 1, and calculate what we would estimate the corresponding 3D point in

timestep 2 to be according to the equation $3Dpoints_1 * T = 3Dpoints_2$. If the difference between our estimation of the corresponding 3D point in timestep 2 and the actual given point in timestep 2 is less than the specified threshold, we consider that point an inlier.

4. After having done the previous step for each 3D point given in the input, we calculate whether or not our inlier to total number of points ratio is greater than 50%. If it is, we exit the loop and save the transform as well as the indices of the inliers to the transform. If not, we continue until the loop finishes.

   (a) If the for loop finishes without having found more than 50% inliers, we simply take the transform (and the corresponding indices of its inliers) that had the maximum number of inliers.

   (b) If the loop finishes without having found any inliers at all for any of the transforms calculated, we double the threshhold and redo the above steps until we've found a transform that has some inliers.

5. After finding the transform, we refit the transform to its inliers by using the equations $3Dpoints_1 * T = 3Dpoints_2$ and $T = 3Dpoints_1^{-1} * 3Dpoints_2$ except this time, $3Dpoints_1$ and $3Dpoints_2$ are lists of all the inliers (more than 4 sets of points). Because there are more than 4 sets of points, solving this equation essentially calculates the transform with the best least squares fit (solves $min(3Dpoints_1 * T - 3Dpoints_2)$ with respect to T). If there are less than 4 sets of points, this equation will have a rank of less than 3 and therefore $3Dpoints_1$ won't be invertible. Therefore, for transforms with less than 4 inliers, we simple use the transform as is instead of refitting it.

4.4 Estimating the next location

With the 3D affine transformation calculated, the last step of our system was to calculate the next location. Since transform calculated is the transform that takes the 3D points from one timestep to the next, we simply have to apply the same transform to our current position to get what the next position will be from our current frame of reference.

4.5 Estimating map points

To get the location, we simply had to apply the given transform to our current location. Transforming the given map points to the appropriate coordinates, however, was a bit more tricky. Every map coordinate observed at timestep n was its location at timestep 1 multiplied by the transform matrices that have been compounded since timestep 1. Essentially, $3Dpoints_1 * T_1 = 3Dpoints_2$, $3Dpoints_1 * T_1 * T_2 = 3Dpoints_3$, $3Dpoints_1 * T_1 * T_2 * T_3 = 3Dpoints_4 ... 3Dpoints_1 * T_1 ... * T_n = 3Dpoints_n$. Therefore, an observation at time n has to by multiplied by $3Dpoints_1 = 3Dpoints_n * (T_n ... * T_1)^{-1}$ to figure

out what its coordinates are in the original coordinate system we started with (the coordinate system that the locations are calculated in, where the left camera at the first timestep is considered the origin). This was done efficiently by keeping track of the compounded T's that have occured so far, and multiplying it by the current T at each step of the iteration.

It should be noted that $T_1, T_2, ...T_n$ are always invertible because they are affine transforms. Affine transforms take the form

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

where R is a 3x3 rotation matrix, t is a 3x1 translation matrix, and 0 is a 1x3 matrix of 0's. Since rotation matrices are orthonormal, the first three columns of the transform span 3D space, so the first 3 columns have a rank of 3, and the 1 in the bottom right position forces the matrix to have a rank of 4, making it invertible because it's a 4x4 matrix.

4.6 System output

We decided that the most useful output of our system would be a graph that includes both the localization aspect (the estimated path of the car) and the mapping component (the 3D points that the car thought it saw at each location) on the same graph. We also wanted the option for this graph to include the ground truth. We wrote methods for each of the datasets that would do this mapping: `mapMalaga` and `mapKITTI`. Each of these functions had to make the necessary adjustments to the ground truths such that they were in relative coordinates. For example, the KITTI dataset reports ground truth locations in lattitude, longitude, and altitude. Thus, we had to get flat world coordinates (using the `lla2flat` function) for it to be a useful comparison against our calculated positions.

The dataset-specific map functions each call `map.m`, which is a function that we wrote to actually create the system output. This is the file that actually estimates the next location using the calculated transforms.

5 Evaluation Methods

5.1 Qualitative Evaluation

For the qualitative evaluation, we plotted the 3d world points in blue and our estimated trajectories in red. We expected to see the 3d points in blue to construct a structure similar to a road, since that is what both the KITTI and Malaga datasets are comprised of (driving down relatively empty roads). We also expected to see our trajectory as a red line heading down the middle of the blue points that formed the road. We also managed to plot the ground truth locations in green with our estimated locations in red and the estimated 3d

world points in blue. We predicted that while our estimated trajectory might not fall on the ground truth trajectory exactly, it would assume the general shape and path of the ground truth trajectory.

5.2 Quantitative Evaluation

One of the important quantities to evaluate in the system was translational drift, especially by varying conditions and observing the effect on the drift. The average translational drift was calculated by finding the difference between the ground truth location and the estimated location in each dimension (x, y, z), and then using the MATLAB norm function to get the magnitude of the differences in each dimension. This metric was mentioned in several papers, including the ones discussed above.

One of the key evaluations mentioned in "Large-Scale Direct SLAM with Stereo Cameras" was decreasing the image resolution and seeing how the calculated drift responded to the change. We followed the same method, decreasing the resolution of images by half from 1 (full resolution) to 1/8, and calculating the translational drift for each decreased resolution. The average drift, total drift, and RMSE of the drift were calculated for several runs of each resolution, and averaged out. This evaluation was performed using the KITTI dataset and the Malaga dataset.

Another quantitative evaluation method involved tracking the drift over periods of distances traveled to see how drift is affected by time. "Large-Scale Direct Slam" also does this and tracks the RMSE drift over a period of 800 meters. In the KITTI dataset, this was done over a period of 60 meters. In the Malaga dataset, this was done over a period of 80 meters. This was done several times for both datasets and then averaged out. We predicted that drift would get worse over time because since each position is calculated using the previous position, the errors in predicted position would compound and lead to the drift becoming worse.

6 Initial System Results

6.1 Qualitative Results

The graphs of our localization and mapping generally look like the graph in Fig. 6. Our output graphs encode the estimated location in with red points, and the 3D points in blue. Here, we can see that the blue points form a type of road that the cameras saw as the car was driving and the red position line goes in a relatively straight line, which is consistent with the KITTI car driving on an almost straight road.

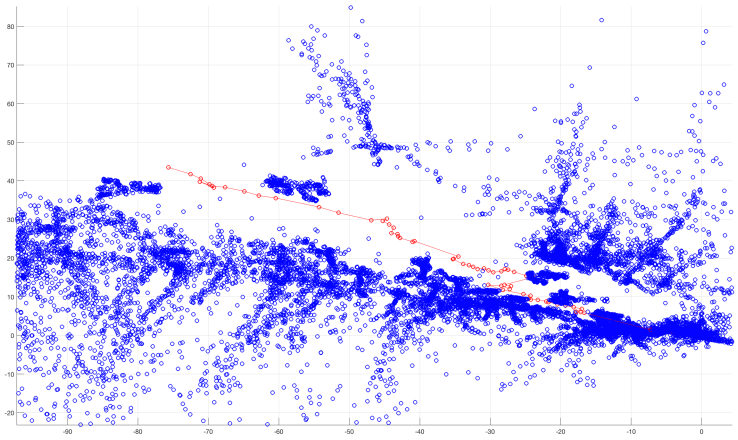

Figure 6: *Example output when system is run on all frames of the KITTI dataset.*

Fig. 7 shows an example of a not as good run of the system. As RANSAC uses random sampling, there is high variability in the output of the system. Since the 3D world points are plotted based on the estimated affine transform and the associated predicted position, a misstep in the position of the car also results in a cluttering of the 3D world points. In Fig. 7, we see what looks like many sets of parallel lines that make the overall road messy. This is because of such missteps in the location that caused the roads to be plotted at a variety of angles. Comparing the red line with the ground truth green line, we see that our predicted path is similar to the ground truth.

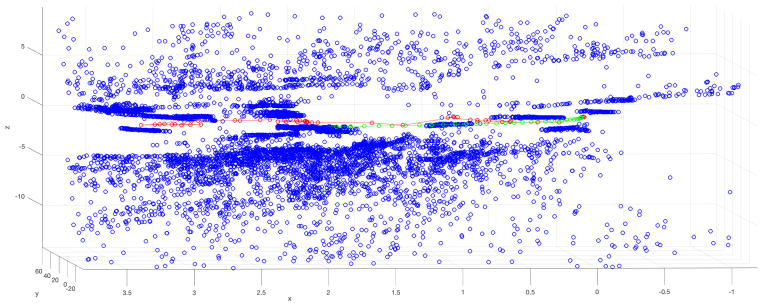

Figure 7: *Output when system was run on 50 frames from KITTI.*

With fewer frames used, it is easier to see the road and the straight line that the car takes. Fig. 8 is an example of the output we got when we ran the system with 20 points from a straight section of the Malaga dataset. The path of the

car is very straight and the road is pronounced. We predicted that drift would get worse over time, as we mentioned in the quantitative evaluation methods section, which is qualitatively confirmed when comparing outputs of few frames with those of many frames.

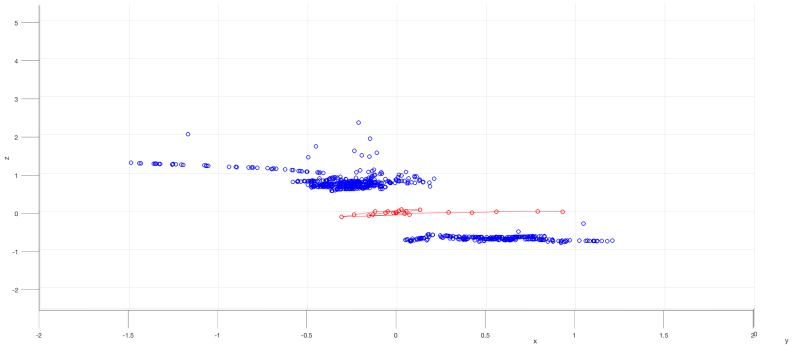

Figure 8: *20 points from Malaga dataset.*

When we first ran our system with the Malaga points, we got graphs that looked like the graph in Fig. 9. It is incredibly messy and looked like the algorithm had lots of difficulty determining what the locations should be. However, when we looked at the images it was using, we noticed that the very first thing the car does is take a ninety-degree left turn. Zooming into the graph (shown in the zoom box in Fig. 9), we see that the algorithm did in fact see this corner and plot it, but that after the turn, the location was extremely hard to determine.

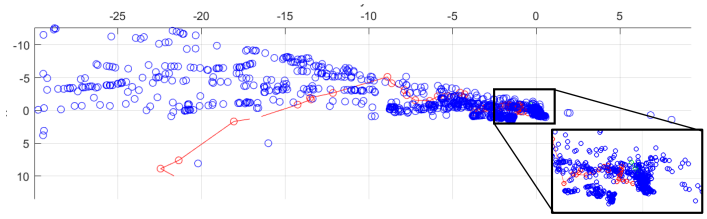

Figure 9: *Effect of corner on rest of algorithm, from Malaga dataset.*

We suspect this difficulty with turns is because the turns are rather fast and the parts of the images seen by the stereo videos in one time step has very little overlap with that seen by the stereo videos in the next timestep. Therefore even if many features were detected and matched between the stereo pair of images, there were very few common features between the stereo pair images in one timestep and the stereo pair images in the next timestep. Even when looking at these images by eye, it was difficult to see that they were from the same run sequence, and even knowing that they were from the same run sequence, it was

difficult to pinpoint where they matched up by eye. Therefore, we used straight line sections of the Malaga dataset for our quantitative evaluation. This also makes the dataset more comparable to the KITTI dataset.

6.2 Quantitative Results

The results of the image resolution evaluation on the KITTI dataset are shown below:

Image resolution	RMSE (dm)	Average Drift (dm)	Total Drift (dm)
1	3.502875	3.00805	85.44485
1/2	6.0457	5.0918	63.3016
1/4	10.44485	9.4622	293.0268
1/8	N/A	N/A	N/A

Figure 10: *Effect of decreasing image resolution on translational drift using KITTI dataset*

Image resolution (without Kalman)	RMSE (dm)	Average Drift (dm)	Total Drift (dm)
1	3.96878	3.48974	104.6689
1/2	4.2067	3.6185	110.1629
1/4	7.8649	6.9094	187.3821
1/8	N/A	N/A	N/A

Figure 11: *Effect of decreasing image resolution on translational drift using Malaga dataset*

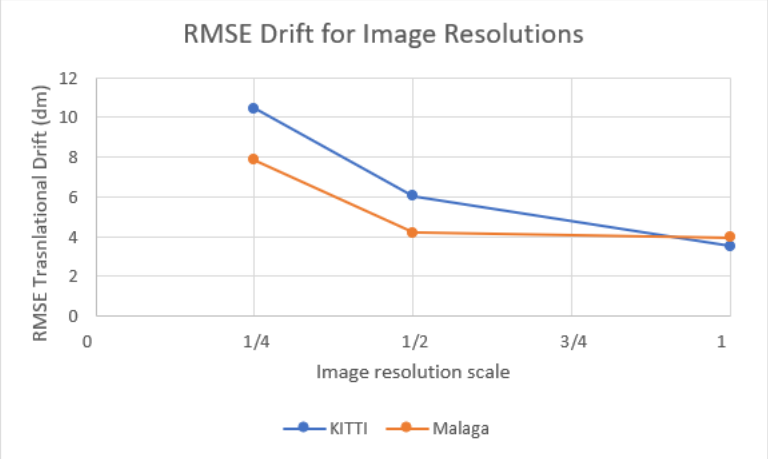

Figure 12: *Effect of decreasing image resolution on translational drift*

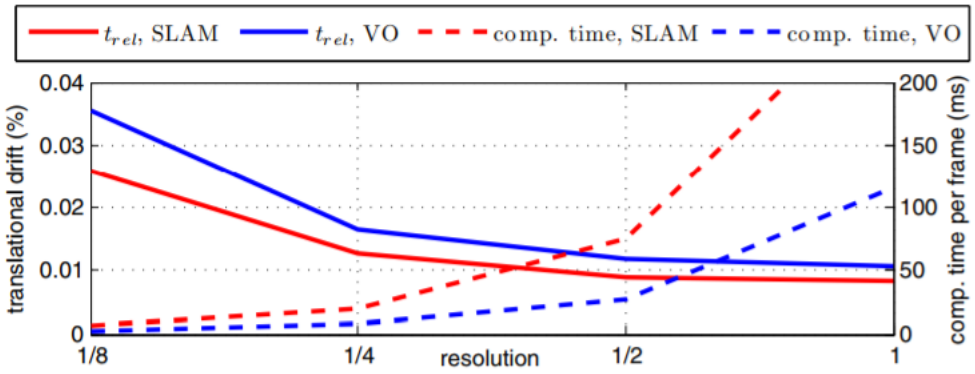

Figure 13: *Effect of decreasing image resolution on translational drift from "Large-Scale Direct SLAM with Stereo Cameras"*

Not surprisingly, the drift seemed to generally increase as the image resolution decreased. There was a general substantial increase in the RMSE and average drift as the image resolution decreased for both the Malaga and the KITTI datasets, which was to be expected. However, what was surprising was that the total drift reached an optimum low point at 1/2 image resolution, and that this was a consistent trend reflected in both the Malaga and The KITTI datasets. One possible reason for this trend may be that the feature detectors are finding noisy repetitive features like trees, and wrongly matching them to parts of other images that are just similar looking trees, not the same trees. Our hypothesis is that when the image resolution is decreased to 1/2, the noise

of the image is smoothed over, and the features detected don't have as much noise and are less repetitive, making them better features to detect. When the image resolution was decreased to 1/8, however, the map function didn't run at all, probably because the number of feature points detected was too low to triangulate and form a reasonable transformation hypothesis out of. It was also interesting to note the difference between the total drift in "Large-Scale Direct SLAM with Stereo Cameras" and our implementation - they had total drifts anywhere from 0.2 to 9.0 meters, while ours ended up averaging out to about 850 meters. This is probably because they implemented the update part of SLAM, while we implemented localization and mapping separately.

The RMSE of the drift and average drift increase when the image resolution is decreased, while the total drift reaches an optimum low point at 1/2 resolution. This is similar to the results seen in the KITTI dataset in the same evaluation above. When the image resolution was decreased to 1/8, the map function couldn't run at all and there were no results.

In figures 12 and 13, we can see that there is a similar trend of declining translational drift in the graphs as the image resolution increases from our algorithm with the KITTI and Malaga datasets, and in the graph from "Large-Scale Direct SLAM with Stereo Cameras." However, the paper mentioned that they were able to still achieve a reasonable drift even with an image scaled down to 1/8 its size and therefore found a useful method for preserving accuracy while decreasing computation time, which we were clearly not able to do as our drifts were relatively high.

Segment Length	Drift (dm)
0-10 m	1.7205
10 -20 m	5.14448
20-30 m	10.51936
30-40 m	14.09162
40-50 m	15.64072
50-60 m	25.17386

Figure 14: *Table representation of drift over segment length for KITTI dataset*

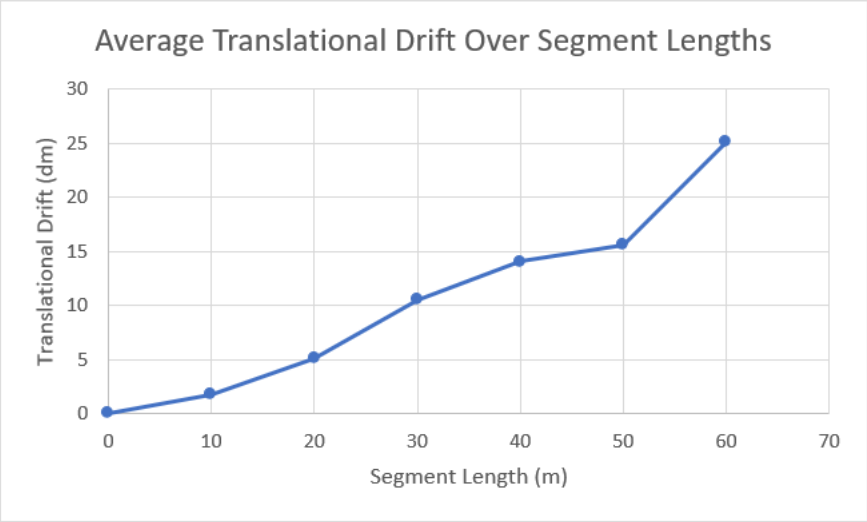

Figure 15: *Graph representation of drift over segment length for KITTI dataset*

Figures 14 and 15 show the effect on drift over segment lengths of the KITTI dataset. As was expected, the drift consistently increased with each segment length, with a slight leveling between 40 and 50 meters. This supported our hypothesis that since the error compounds by building off of its last erroneous hypothesis and adding extra error at each step, the drift towards the end of the run was significantly larger than the drift at the beginning of the run.

Segment (m)	Drift without Kalman (dm)
0-10	2.19248
10 -20	4.12088
20-30	8.71208
30-40	12.80334
40-50	16.71268
50-60	14.93972
60-70	22.61548
70-80	25.09528

Figure 16: *Table representation of drift over segment length for Malaga dataset*

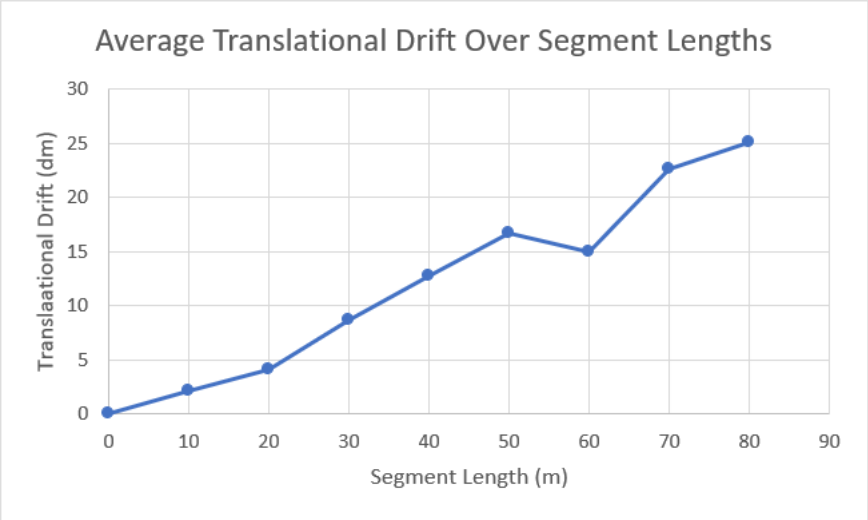

Figure 17: *Graph representation of drift over segment length for Malaga dataset*

Figures 16 and 17 show the effect on drift over segment lengths of the Malaga dataset. The drift over segment length for the Malaga dataset also increased consistently for the most part, except for a slight drop off between the 50 and 60 meter segment. This generally followed our hypothesis of the drift increasing due to compounded error, and makes sense in the context of our calculations.

7 Implementing a Kalman Filter

What the Kalman filter essentially does is that it combines two measurements taken two different ways with two different uncertainties into one measurement that usually has a lower uncertainty than either of the two measurements that make it up. The way it does this starts with two assumptions: that each method of measurement can be modeled by a gaussian. The first of these methods, called the prior, or $p(x)$, is the probability function of the measurement that we are calculating (our ground truth position). The mean of this probability function is the exact number that we have calculated through stereo odometry (because using maximum likelihood estimation, the likelihood of the probability function is maximized when the mean of that probability function equals the observed, or in our case, calculated, value), and its covariance, P, is a measure of how uncertain we are about our measurement/calculation. The second way of measurement, called the likelihood, or $p(z|x)$, is the the measurement z, and is related to the prior in some way. In our case, z is the ground truth, and it is related by $z = x + w$, where w is gaussian noise with zero mean, and covariance R and x is the true position of the car. Since we have the covariance matrix of

the ground truth measurements for the Malaga dataset, we equated R with the dataset's covariance matrix. Essentially, using these probabilities, what we are trying to find is x, the true position of the car. Therefore, the probability we are trying to calculate is $p(x|z) = p(z|x) * p(x)/p(z)$. This probability distribution will be a gaussian (since we assumed that $p(z|x)$ and $p(x)$ are gaussians and we can get $p(z)$ by integrating $p(z|x) * p(x)$), whose mean (which is its maximum and most probable value) and covariance we can calculate. By expanding these probabability distributions into their multidimensional gaussian representations (definition of a gaussian in terms of its mean and covariance), and using matrix manipulation to make the equation take a useful form (from the notes of Paul Newman, Oxford University), we can get the equation to look like this: $x|z_{mean} = x_{priormean} + W * v$ and $Cov_{x|z} = P - W * S * W^T$ where the innovation, v, is $v = z - x_{priormean}$, the innovation covariance, S, is $S = P + R$, and the Kalman Gain, W, is $W = P * S^{-1}$.

Linear Kalman Filter Equations

prediction:

$$\hat{\mathbf{x}}(k|k-1) = \mathbf{F}\hat{\mathbf{x}}(k-1|k-1)$$
$$\mathbf{P}(k|k-1) = \mathbf{F}\mathbf{P}(k-1|k-1)\mathbf{F}^T + \mathbf{Q}$$

update:

$$\hat{\mathbf{x}}(k|k) = \hat{\mathbf{x}}(k|k-1) + \mathbf{W}(k)\nu(k)$$
$$\mathbf{P}(k|k) = \mathbf{P}(k|k-1) - \mathbf{W}(k)\mathbf{S}\mathbf{W}(k)^T$$

where

$$\nu(k) = \mathbf{z}(k) - \hat{\mathbf{x}}(k|k-1)$$
$$\mathbf{S} = \mathbf{H}\mathbf{P}(k|k-1) + \mathbf{R}$$
$$\mathbf{W}(k) = \mathbf{P}(k|k-1)\,\mathbf{S}^{-1}$$

Figure 18: *Figure taken from online notes by Oxford University Professor Paul Newman.*

This means that at every frame, we have a prediction of where we think our location is based on multiplying our current location by the translation matrix.

According to the equations in the above figure, this translation matrix is represented by F.

As stated before, P is how certain we are about our measurement. However, at the very beginning of the algorithm, we are 100% certain about our location because we are defining the coordinate system by placing ourselves at (0,0,0) when we first start out. Therefore, at the beginning, P consists of all 0's.

The matrix Q in the above figure represents the uncertainty added to our own measurements at each iteration. This is a quantity that we can control. By making the elements in Q close to 0, we are saying that we are very certain about our measurements, and by making the elements of Q large, we are saying that we are very uncertain that our measurements are correct.

When there is ground truth data we update the prediction using the 'update' equations (derived in the above paragraphs). These equations take our predictions as the prior and multiply it by the ground truth 'likelihood' to get the best estimate of the true x. These equations are implemented in our `kalmanFilter.m` file

We had originally intended to use the Kalman Filter to incorporate both the stereo odometry measurements we had calculated, and the vehicle odometry that is calculate using the motion of the wheel, but we were unable to find datasets that contained both stereo video, and vehicular wheel odometry. Therefore, we performed our measurements using ground truth odometry instead of vehicular wheel odometry, just to show how the Kalman Filter would work. To make it fair, we made the values of our Q very close to the values of R, on the order of between 10^{-5} and 10^{-7} to force the Kalman Filter to incorporate our measurements as well as the ground truth measurements. If we had chosen reasonably large values for Q, the algorithm would ignore our measurements and be extremely biased by the ground truth measurements because they are much more accurate.

After implementing the Kalman filter, it was important to evaluate its effect on performance compared to the base implementation without it. This was done by calculating the translational drift several times and then averaged out, as discussed above, with the Kalman filter included and without. This evaluation was performed on the Malaga dataset.

	RMSE (dm)	Average Drift (dm)	Total Drift (dm)
Without Kalman	3.96878	3.48974	104.6689
With Kalman	3.89408	3.2315	97.3478

Figure 19: *Comparison of performance without Kalman Filter vs. with Kalman Filter*

Image resolution (with Kalman)	RMSE (dm)	Average Drift (dm)	Total Drift (dm)
1	3.89408	3.2315	97.3478
1/2	5.2372	4.3115	53.3657
1/4	8.3487	7.4365	230.5329
1/8	N/A	N/A	N/A

Figure 20: *Effect of decreasing the image resolution with Kalman filter*

Segment (m)	Drift with Kalman (dm)
0-10	2.159
10 -20	4.11566
20-30	8.954
30-40	13.00396
40-50	16.61484
50-60	14.69616
60-70	21.90662
70-80	23.34456

Figure 21: *Drift over segment length with Kalman filter*

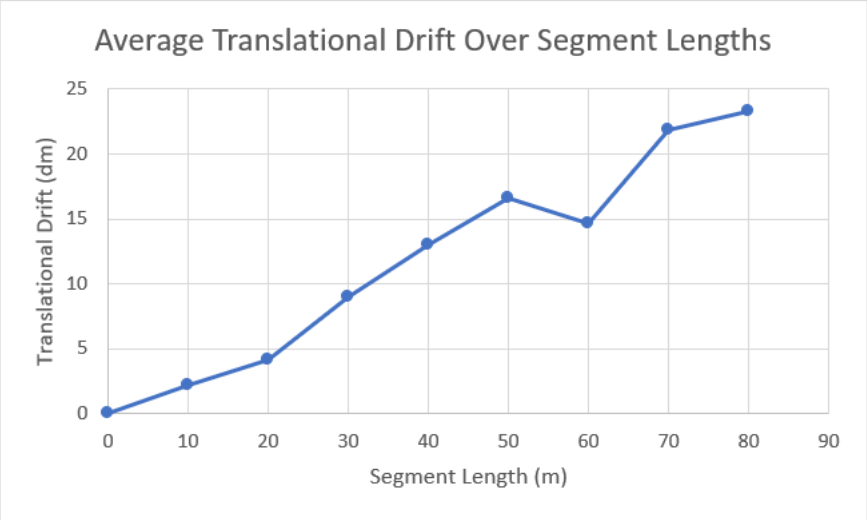

Figure 22: *Drift over segment length with Kalman filter*

Figure 18 shows how the algorithm performed with the Kalman filter and without it. Figure 19 shows the effect of decreasing the image resolution with the Kalman filter, and figures 20 and 21 show the effect on drift over segment length with the Kalman filter. It is hard to tell whether the implemented Kalman filter improved the algorithm performance or not. This wasn't an entirely unexpected result given the amount of confidence we gave to our measurements. The RMSE, average drift, and total drift all decreased slightly with the Kalman filter, but it is unclear whether this is due to the filter or due to other factors. With the image resolution, the drift did increase significantly, but it seemed to increase along the same scale as the implementation without the Kalman filter. Similarly, for the drift over segment length evaluation, the drift with the Kalman filter seemed to increase along the same pattern as the drift without the Kalman filter.

8 Using more Malaga dataset frames

As we mentioned in our initial qualitative analysis, turns were particularly difficult for our algorithm to handle given that there was very little overlap between frames taken one second apart as the car was turning. In the KITTI dataset, each frame had an associated ground truth. However, for the Malaga dataset, ground truths were given at most every second, and there were many stretches where no ground truth locations were provided. Thus, when we built our initial system, we decided to use the first frame from every second of the Malaga dataset so that we could quantitatively evaluate. However, we thought that it might be interesting to try and use our algorithm with more of the Malaga

dataset points and qualitatively evaluate the resulting graphs. Our hypothesis was that a higher frame rate would result in better pictures.

We used the first 207 frames from the Malaga dataset, which translated to about 30 seconds. The first 130 frames are when the car is making the first left turn and the remainder of the frames have the car driving in a straight line. Fig. 23 shows the result of inputting all 207 frames into our system.

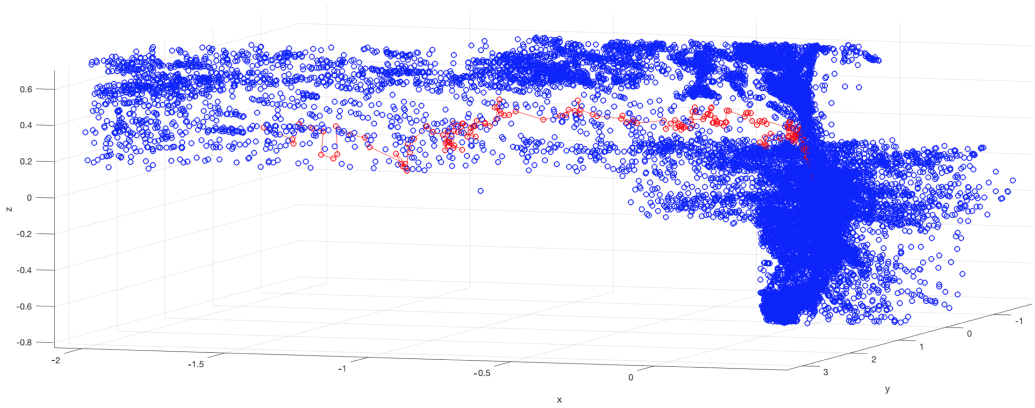

Figure 23: *Malaga dataset when all frames per second were used.*

The results are quite an improvement from the output graph of a turn with only one frame per second. We can see the car making a left turn and then maintaining a relatively straight path.

We also used frames 130 to 207 to see how more frames per second improved a estimating straight line motion, the output of which can be seen in Fig. 24.

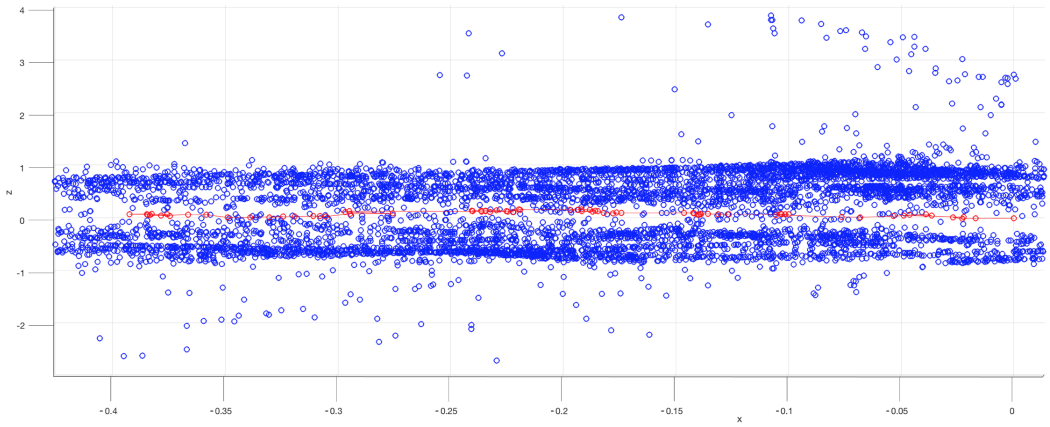

Figure 24: *Straight portion of the Malaga dataset when all frames per second were used.*

This figure is quite beautiful and we can very clearly see the car moving in a straight line as well as pretty definitive road boundaries. So, as expected, a higher frame rate, which allows for more overlap from frame to frame, visibly improves our system.

9 Challenges

One of the main problems we noticed while starting this project was the lack of open source 3D stereo implementations of SLAM. While there were a few of these open source projects, they were incredibly difficult to install on both MacOS and Windows due to tricky dependencies such as g2opy, Pangolin, or OpenCV. Although there were several 2D SLAM simulations, not many allowed for testing on real world datasets. This lack of implementations partly inspired us to implement a system ourselves without any starter code.

10 Conclusions and Future Work

In this project, we were successful in implementing a localization and mapping algorithm and tracking our progress against the ground truth. One of the main strengths of our system was that it worked really well with straight road sequences, as shown in the graphs of our evaluation. Furthermore, as expected, it worked much better when the time between frames was smaller. The Kalman Filter seemed to have a positive impact on the results by decreasing the drift, but we can't say conclusively that it would help given that we weren't able to implement it using vehicle wheel odometry as we had planned to. Another interesting conclusion we could make from our data was that halving the image resolution actually improved the results, as shown in both datasets.

However, our implementation also had a few flaws. One of the flaws is that decreasing the image resolution to 1/4 and 1/8 significantly degraded the performance to the point where the algorithm wouldn't even run on 1/8 resolution on any of the tested datasets, even with the Kalman Filter. However this flaw may be attributed to using built in SIFT feature extraction functions so it may be improved by using better feature extraction algorithms. Another flaw was that the drift was extremely high compared to the drift recorded in papers that implemented similar algorithms. One reason for this may be that the algorithms described in the papers included the feedback loop between localization and mapping, thereby decreasing the uncertainty of both functionalities.

One small issue with our algorithm, which might have led to lower accuracies, is the percentage of points that we considered 'inliers' in the RANSAC algorithm we implemented in `findAffine3D.m`. If we defined inliers to be a higher percentage of the total points, this might make our algorithm more accurate. Another aspect of our RANSAC implementation that we could have explored is increasing the number of iterations. When we came across points that didn't have any inliers, we changed the threshold, but maybe we could have changed the number of iterations as well and gotten potentially better results

For the future, we can try to implement a feedback loop between localization and mapping to fully implement SLAM. This will involve remembering past features somehow, and detecting when the car has made a loop, and therefore is much more complicated than the current scope of the project.

Another area of future work would be to properly implement the Kalman Filter the way we had planned, by using vehicle wheel odometry. We think it would be very interesting to see whether or not the vehicle wheel odometry (which is highly inaccurate in and of itself) will have a positive effect on our result.

11 Appendix 1: Description of submitted MATLAB files

1. `adjustMalagaGroundTruth.m`
   We have this function to correct for the fact that we are using the world coordinates' transform when we calculate our motion, however the world coordinates are moving in the opposite direction from us. Thus, our motion is in the -x direction (as can be seen in all of the plots) instead of the +x direction as the ground truth specifies. This function multiplies the x coordinate of the ground truth by -1 so that motions we are comparing have consistent directions.

2. `calculateTranslationalDrifts.m`
   This function has all the quantitative evaluation code. Here, we calculate

the drifts in x, y, z directions, the total drift, average drift over the entire sequence of frames, the RMSE of the drift in each direction, and the drift over various segment lengths.

3. `findAffine3D.m`
   This is the function that performs RANSAC to determine the 3D affine transformation between two sets of world points.

4. `getMalagaCovariances.m`
   Function to parse the Malaga ground truth text file to get covariance matrices for each available ground truth timestamp.

5. `getMalagaGroundTruths.m`
   Function to parse out ground truth locations from Malaga ground truth text file.

6. `kalmanFilter.m`
   This function can be called from `mapMalaga.m` instead of `map.m`. Like `map.m` it will output a graph of the estimated location of the car and the associated 3D points, however it uses the Malaga covariances in a Kalman filter to improve on the estimated positions before plotting.

7. `map.m` This function is called from `mapKitti.m` and `mapMalaga.m`. It uses the calculated transforms that are passed into it to determine the path of the car, and does the plotting for the output graphs.

8. `mapKitti.m`
   This function is the starting point for executing the system with frames from the KITTI dataset. It calls `stereo2transform.m` for each consecutive set of stereo frames to get the transform and then `map.m`, which gives the output graphs.

9. `mapMalaga.m`
   This function is the starting point for executing the system with frames from the KITI dataset. It takes in one parameter which is a boolean toggle for whether or not to use a Kalman filter. First, it calls `stereo2transform.m`. Then, it the value of the parameter is 0, then this function calls `map.m`. If the parameter is set to 1, this function calls `kalmanFilter.m`.

10. `mapMalagaHighFrameRate.m`
    We made this function to experiment with using all the available frames from the Malaga dataset to see how it would improve our localization and mapping output.

11. `plotMovingWorldPoints.m`
    This function enabled us to create plots like the right image in Fig. 5, as a sanity check to make sure that our 3D point triangulation made sense.

12. `stereo2transform.m`
    This function does most of the heavy lifting in our system. It does all the steps from retrieving the images specified by a parameter from the specified dataset to calling `findAffine3D.m` to get the tranform. It is this transform that it returns to its caller – either `mapMalaga.m` or `mapKITTI.m`.

12 Appendix 2: Links to papers, datasets, and used code

12.1 Papers

1. "Large-Scale Direct SLAM with Stereo Cameras"
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7353631
2. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras"
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7946260
3. "Stereo Parallel Tracking and Mapping for robot localization"
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7353546&tag=1
4. "A collection of outdoor robotic datasets with centimeter-accuracy ground truth"
   https://link.springer.com/content/pdf/10.1007%2Fs10514-009-9138-7.pdf
5. "Vision meets robotics: The KITTI Dataset"
   http://www.cvlibs.net/publications/Geiger2013IJRR.pdf

12.2 Datasets

1. Malaga campus RT dataset
   http://www.mrpt.org/downloads/dataset2009/README_campus_RT.html
2. KITTI dataset
   http://www.cvlibs.net/datasets/kitti/raw_data.php?type=city
   (Note: we used the first dataset on this page: `2011_09_26_drive_0001`)

12.3 Code

All the functions that we used in building our system (listed in Section 4: System Design and Implementation) are from the MATALB Computer Vision Toolbox (https://www.mathworks.com/products/computer-vision.html).

The one exception is the function `lla2flat`, which is from the MATLAB Aerospace Toolbox (https://www.mathworks.com/help/aerotbx/).