Vehicle Localization Using Low Cost, Mobile Device-Based Sensor Suite with Google Street View

Anchal Sinha Alexander Swerdlow Anuj Agrawal Connor Roberts Puneet Nayyar

***Abstract*—In this paper, we present a method to use data recorded from a smartphone in tandem with Google Street View imagery to perform localization in an urban environment. We query for nearby panoramas, create virtual views using backprojection, and perform robust feature matching with our smartphone camera frames. We then run a 3D point reconstruction and subsequently perform a local bundle adjustment both with and without fixed camera poses. We fuse this location estimate with a separate Visual-Interial Odometry module to improve the robustness of our final localization estimate. Our method is tested on an urban environment along a 1 km test track and accuracy is calculated by computing reprojection errors from our bundle adjustment and the distance between our estimated location and ground truth location which is a set of hand labeled points along our trajectory. Our results showed the efficacy of this approach in providing lane-level accuracy across our test region, along with producing low reprojection error for the bundle adjustment process.**

I. Introduction

In recent years, many cars have released with SAE Level 2 autonomous driving features. The price tag for a car with these new features is still out of reach of many people, including those who may need them most. One of the major contributors to the price of these vehicles is the cost of their sensors. Currently, a full suite of sensors for a level 2 autonomous vehicle can cost anywhere from $8,000-10,000 [1]. As we begin to progress towards higher levels of autonomy, the suite of sensors needed will only increase in complexity, and in turn, price. Additionally, it has been shown that traditionally GPS is not accurate enough to be used as a localization method for autonomous driving and this inaccuracy only increases exponentially in an urban environment. With our approach we aim to create a level of accuracy that is comparable to that of much more expensive sensors at a fraction of the cost.

In this paper we propose a way to exploit this 'urban jungle' and improve greatly upon the accuracy of a traditional GPS using sensors found on a single smartphone. Our method is based around using Google Street View's panoramic images and comparing them to those being taken on board with our monocular smartphone camera. We chose to use Street View because it contains data from a large portion of the world and this data gives us both a high resolution image with accurate associated location metadata. Additionally, anyone around the world with an internet connection can access Street View. We also log measurements from the IMU and Apple's ARKit Framework simultaneously with the images recorded from our smartphone.

Our approach involves extracting features from our recorded frames, as well as from the panoramas taken from Google Street View. Once we have these features, we calculate the relative pose between our frame and the surrounding panoramas. We then use the poses in tandem with the coordinates of each panorama provided in the metadata to triangulate our estimated position. Additionally, we aim to use the IMU measurements alongside our image data to compute our Visual-Inertial Odometry module and combine it with our estimated location in order to reduce noise.

II. Related Work

Previous work exists on visual global localization, including work specifically on the Google Street View dataset. Agarwal et al. [9] used optical-flow to create tracks of 3D points and performs a semi-global bundle adjustment on these tracks. More recently, Yu et al. [8] took a more limited approach by only performing a local bundle adjustment on the current set of images, not taking advantage of any odometry data. This approach showed centimeter-level accuracy, however it also made use of depth data that we found unreliable in many areas, and two roof-mounted sideways cameras that faces street facades. Our approach builds off of this previous work from [9] and [8] but uses only a single front-facing phone camera and combines additional odometry data directly from the phone.

III. Method

In this section, we describe our pipeline for global localization. The pipeline was implemented in an offline manner, but at each timestep we only look at current inputs and the estimated vehicle pose, discarding all other past inputs. Although we do perform bundle adjustment, our approach does not involve SLAM in its typical formulation since we do not perform mapping. Instead, we seek to only compute a global localization through feature matching and local bundle adjustment in combination with Visual-Inertial Odometry. The diagram in Figure 1 illustrates the important steps of our pipeline.

A. Google Street View Data

Google Street View is a platform that contains millions of panoramas and depth maps, collected from vehicles driving throughout the world. Each panorama is a 16384x8192 equirectangular image with a 360° horizontal and 180° vertical field-of-view (FoV), and each depth map is a 512x256 image with the same horizontal and vertical FoV. The panoramas are

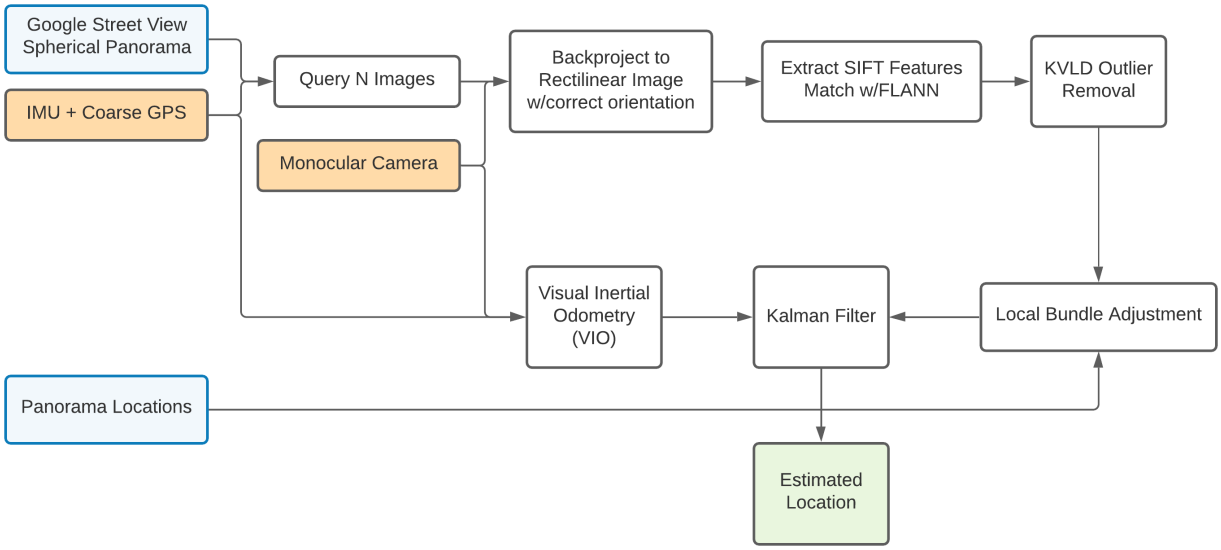


Fig. 1. Diagram illustrating our system pipeline

tagged with precise accurate latitude and longitude coordinates which is calculated by first fusing sensor data from GPS, wheel encoders, and an IMU, and further optimizing the trajectory with known road network and intersection maps [7].

The depth map captures data from laser scans or using optical flow methods, and stores only the dominant scene features such as buildings and roads [7]. However, as seen in Figure 2, in dense urban environments where we need complete representation of static scene elements, the depth map is unreliable in generating that data.

Fig. 2. Google Street View Panorama (RGB and Depth)

To perform feature extraction and matching on the panoramas, we must generate rectilinear images from the equirectangular panoramas. Google provides an API to pull rectilinear images at specified panorama locations given the heading and pitch angles. Instead, we store a database of panoramas we intend to query as equirectangular projections of the full 360° field-of-view, and extract the rectilinear regions of the panorama with the precise vehicle heading, making our method more robust and fast. We create a "virtual camera" and perform a back-projection technique to project the panorama into a standard rectilinear view. Provided a heading and field of view, we can compute the corresponding pixel coordinates on the rectilinear image from the panorama. With the center elevation and azimuthal angles of the image being (θ_0, ϕ_0) and the angular distance from rectilinear pixel coordinates (x, y) to panorama coordinates (θ, ϕ) defined as c, the transformation equations described in (1) are used to generate rectilinear images with a defined focal length and FoV.

$$
\begin{aligned}
x &= \frac{cos(\theta)sin(\phi - \phi_0)}{cos(c)} \\
y &= \frac{cos(\theta_0)sin(\theta) - sin(\theta_1)cos(\theta)cos(\phi - \phi_0)}{cos(c)} \\
cos(c) &= sin(\theta_0)sin(\theta) + cos(\theta_0)cos(\theta)cos(\phi - \phi_0)
\end{aligned}
\tag{1}
$$

This virtual camera has the same camera intrinsics as our vehicle camera, maintaining the same focal length, horizontal and vertical field-of-view, and resolution, although we assume the principal point of the camera is the image center. Consistency in the camera intrinsics will simplify future computations for 3D point estimation and pose estimation knowing that the focal lengths are the same.

B. Feature Matching

For each frame in our video stream, we find nearby Street View panoramas using a KD-tree. For our image frame and

one of these nearby panoramas, we use SIFT to extract key points in both images, shown in 3 [2]. With these SIFT key points, we then use FLANN based matching to find matching key points between our image and the panorama, similarly shown in 3 [3]. From this example set of matches, we notice no clear set of matching features between the two images. Clearly, a large portion of the computed matches in this example are incorrect and as a result should be discarded. These incorrect matches, referred to onwards as outliers, negatively affects our ability to find the essential matrix that describes the epipolar geometry between our captured image and the nearby panorama, which leads to inaccurate localization.

Fig. 3. From left to right: captured frame and nearby panorama image, camera frame and panorama SIFT key points, camera frame and panorama FLANN matches.

To effectively address the outlier issue, we apply a filtering technique to our FLANN matches using second order descriptors known as Virtual Line Descriptors (VLD) [4]. To define a VLD, for each pair of key points in an image we compute a number of separate SIFT-like descriptors for the regions in the image between the two points, shown in 4. Together, this collection of descriptors act as our second order descriptor or VLD. With this definition, we can then also compute the difference between two VLDs as the pairwise difference between corresponding disks in their gradient histograms as well as main orientations [4].

Now, we consider a pair of matches in our image and the panorama, giving us two points, or one VLD, in each of the two images. Then, we find the difference between these two VLDs, and if this difference is below a maximum threshold we consider this pair of matches to be consistent [4].

The K-VLD filtering algorithm uses this idea of consistent matches, and for any primary match between our image and the panorama, we look one by one at all the nearby matches, and for each of these, check if the pair of matches is consistent [4]. If there are at least K consistent matches for our primary match, it satisfies the K-VLD consistent constraint and is kept as a reliable match [4]. This process is repeated for all matches between the two images until we have only K-VLD consistent matches remaining [4].

We return to our previously computed FLANN matches in 3 with all of the outliers which we need to remove. Using the K-VLD algorithm, we can find all of the consistent VLDs, shown in 5 where these VLDs are highlighted. We then keep all matches from these consistent VLDs, leaving a much smaller set of matches, but now with almost all outliers removed, shown in 5.

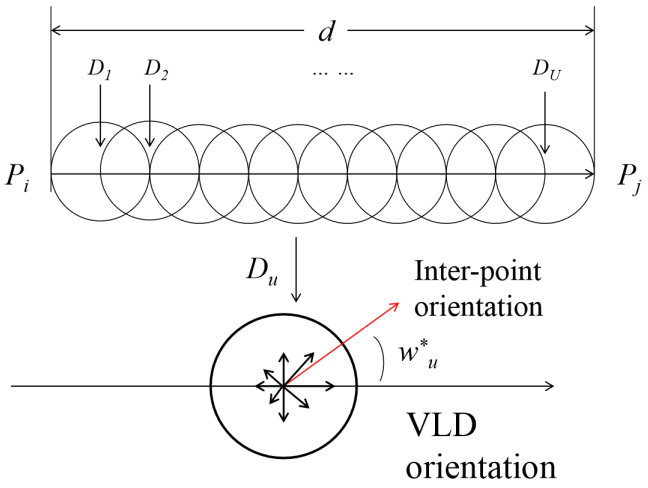


Fig. 4. Visualisation of a Virtual Line Descriptor [4]

Fig. 5. Top: Camera frame and panorama with consistent VLDs highlighted. Bottom: K-VLD filtered matches

The main benefit of K-VLD filtering for our application is its property of removing ambiguous matches [4]. These types of matches are common in our dataset, which features large numbers of ambiguous features such as building windows. Additionally, this method removes false matches near epipolar lines, which again is well suited for our dataset with the majority of detected features being located along or near epipolar lines [4]. We perform SIFT feature extraction, FLANN matching, and K-VLD filtering using a fork of openMVG [25].

C. Visual Pose Estimation

In order to estimate the pose of the vehicle, we tested two similar but distinct strategies, both based on a local bundle adjustment. The first method relies upon known fixed panorama locations and uses a limited local bundle adjustment to estimate 3D points common between all panorama images. Then, using features common with the phone camera, we use PnP to estimate the phone camera relative to a reference panorama. The second method involves a standard local bundle adjustment procedure where we allow the known panorama locations as well as the vehicle pose to be adjusted subject to some constraints and use the final vehicle pose from this computation. We chose these two approaches because we sought to analyze whether allowing for more flexibility during bundle adjustment w.r.t camera poses would produce more accurate localization.

D. Fixed-pose Local Bundle Adjustment

With a set of panoramas P and their corresponding features F and matches M with the vehicle frame V, we will take the subset of features

$$F^* = \{F_P \cap F_V\}$$

that are present in all panoramas P and vehicle frame V. This will give us a set M^* containing pairs of features, where for each feature in F_V we will have a corresponding feature F_P in each of the panoramas, as seen in Figure 11. We can then perform a nonlinear optimization using SciPy to estimate the 3D points for each feature F_V relative to the selected panorama by minimizing the following loss function. Unlike a typical bundle adjustment where we allow the camera poses to be adjusted, we fix these poses and only adjust the 3D point estimations while minimizing the error defined in (2).

$$F(x, y) = \sum_{ij} e_{ij}(x, y) \tag{2}$$

where:

- x = A vector of 6DOF Street View camera poses
- y = A vector of 3D points associated with each feature
- e_{ij} = Error function between predicted 3D points $z_{ij}(x, y)$ and measured point $\hat{z}_{ij} = (\theta_{ij}, \phi_{ij})$
- z_{ij} = The elevation and azimuthal angles from camera pose x to 3D feature point in the 2D camera frame

With the image coordinates and 3D global coordinates of F_V, we can perform a Perspective-n-Point pose estimation to best estimate the pose of the camera relative to the 3D points to best minimize the reprojection error. The overall problem of 3D point estimation can be seen in Figure 7, where we estimate the positions of the 3D object points when each of the corresponding feature points in each camera view is known.

E. Standard Local Bundle Adjustment

As before, we have a set of panoramas P and their corresponding features F and matches M with the vehicle frame V. Instead of taking the intersection of features

$$F^* = \{F_P \cap F_V\}$$

that are present in all panoramas P, we instead require only that a feature is present in at least n panoramas. This will give us a set M^* containing pairs of features, where for each feature in F_V we will have a corresponding feature F_P in at least n panoramas.

To get the best results from the local bundle adjustment, we need to initialize the solver with an estimate of all $|M^*|$ 3D points. Since each of our $|M^*|$ 3D points is present in at least n panoramas, there is not one correct initialization. Instead, we use the the 8-pt algorithm for each $(f_v, f_p), f_v \in F_V, f_p \in F_p$ pair to approximate the essential matrix. We then recover the relative pose transformation between the cameras and perform 3D reconstruction by triangulation. We then retrieve only the 3D points that are common to at least n images.

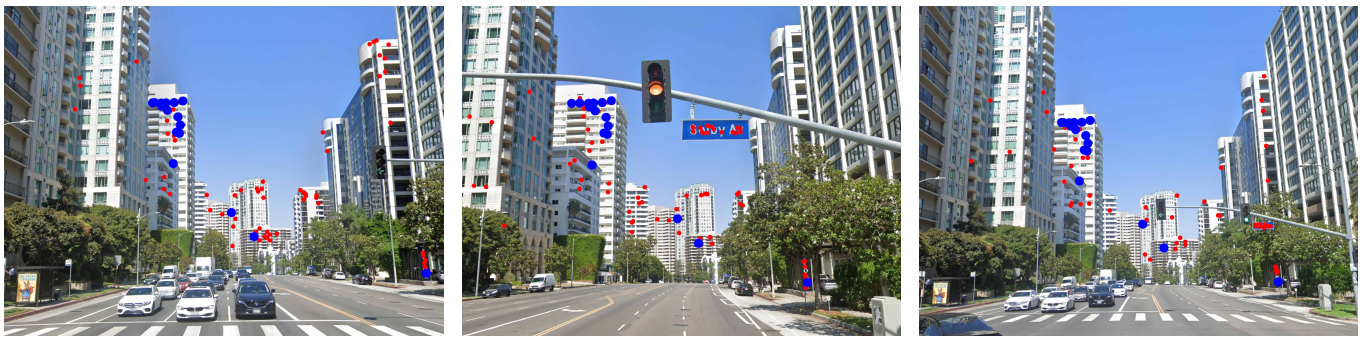
Thus we have at least n estimates of any given 3D point through our approximate reconstruction method. However, these points are in the reference frame of the respective panoramas so we must transform these coordinates into a global frame which we define as the reference frame of the panorama with highest number of feature matches. Given at least n estimates of any given 3D point we pick the point with the median L2 norm as a simple form of outlier rejection.

We chose to compare two common nonlinear solvers used to for graph optimization, g2o [5] and Ceres [6]. A graph is a natural representation of the bundle adjustment problem since it allows us to relate quantities we wish to estimate such as camera poses and 3D points, with our measurements as edges that relate the quantities. Specifically, we construct a graph such that the edges represent the projection of landmark 3D points onto our camera frames. A visual representation of how the factor graph represents an objective function is seen in figure 8 courtesy of [5].

We create n vertices in our factor graph G corresponding to our n panorama locations and, for Ceres, we fix the allowable translation and rotation of these poses to some upper and lower bound. With more data on the accuracy of these panorama poses, future work may incorporate a Gaussian prior term in the error function that represents our certainty of the panorama poses and penalizes significant adjustments. This would better model our assumption of the system but we were unable to incorporate it into the Ceres solver through the Python bindings used for testing.

Next, we add an additional vertex corresponding to our phone camera pose which we initialize from our Kalman Filter or, for the first frame, our GPS coordinates. This vertex is entirely unconstrained.

For each of our $|M^*|$ 3D points, we add a vertex to our factor graph G with the estimated position from our 3D reconstruction and attach at least $n+1$ edges to this point. At



(a) (b) (c)

Fig. 6. Features present in all panorama views. Red key points indicate features unique to that frame and blue key points indicate features found in all frames.

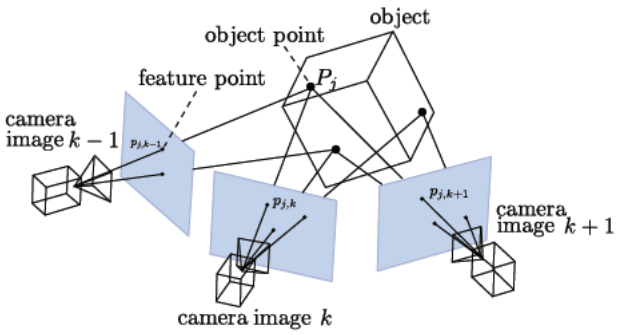
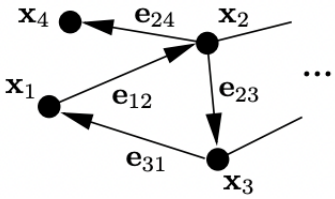


Fig. 7. Multi-view 3D Point Estimation. Object points are estimated knowing that a corresponding feature point is in each camera view. Courtesy of [24]



$$\mathbf{F}(\mathbf{x}) = \mathbf{e}_{12}^\top \, \mathbf{\Omega}_{12} \, \mathbf{e}_{12} + \mathbf{e}_{23}^\top \, \mathbf{\Omega}_{23} \, \mathbf{e}_{23} + \mathbf{e}_{31}^\top \, \mathbf{\Omega}_{31} \, \mathbf{e}_{31} + \mathbf{e}_{24}^\top \, \mathbf{\Omega}_{24} \, \mathbf{e}_{24} + \dots$$

Fig. 8. Relationship between objective function and factor graph courtesy of [5]

least n edges are between panorama locations and the point with an additional edge connecting the smartphone camera frame. These edges contain the observed (x, y) pixel that corresponds to the 3D point as well as the uncertainty of the measurement. The objective that we seek to minimize is the reprojection error of these 3D points onto the least $n+1$ cameras in their respective locations, and we do so by adjusting the positions of the 3D points and the 6-DoF poses of the cameras.

As suggested in [8] we use the Tukey Biweight function as in (3) with $c = 3$ to improve convergence of our optimization problem in cases where outliers remain past the K-VLD step. We replace the squared error we are trying to minimize with the Tukey Biweight function and use g2o and Ceres to solve the graph optimization problem.

$$\ell(r) = \begin{cases} \frac{c^2}{6}\left(1 - \left[1 - \left(\frac{r}{c}\right)^2\right]^3\right) & \text{if } |r| \le c, \\ \frac{c^2}{6} & \text{otherwise.} \end{cases} \tag{3}$$

F. Sensor Fusion

Our estimate of the vehicle's position using Google Street View might suffer from various points of failure such as occlusion of static features, lighting and weather condition and inaccurate street view data. The dependability of autonomous systems of one just one such position estimate can result in serious consequences. We address this problem by implementing a multi-sensor fusion algorithm. The recent trend in high quality and low cost smartphones has led to multiple sensors being incorporated into a small form factor. We exploit these sensors to derive a better estimate of the vehicle's state. We explored two such methods:

1) Visual Odometry and AR Poses: Visual odometry is estimating the motion of the vehicle using visual cues from the cameras attached to it. We deployed a featured based monocular odometry. Our pipeline for visual odometry is as shown in Fig. 5. We use the FAST corner detection algorithm [14] [15] to detect features in each frame It due to it's fast computational efficiency. Each of these corner pixels are tracked using the optical flow bases KLT tracker [16] in the successive frame It+1. By selecting points using RANSAC from pairs of corresponding points in successive frames we estimate the essential matrix E as shown in Eqn. 4 using the Nister five-point algorithm [17]. Furthermore, decomposing the essential matrix using SVD and exploiting rotation matrix constraints gives the rotation and translation matrix between the two consecutive frames as shown in Eqn. 5 and 6.

$$y_1^T E y_2 = 0 \tag{4}$$

$$E = U \Sigma V^T \tag{5}$$

$$[t] = VW\Sigma V^T, \; R = UW^{-1}V^T \tag{6}$$

The net pose with respect to the first frame is given by

$$R_{pos} = RR_{pos}, \; T_{pos} = t_{pos} + \mu t R_{pos} \tag{7}$$

Translation in Monocular visual odometry suffers from scale factor issues that need to be estimated using external sensors. We use AR poses as obtained from the Augmented Reality Toolkit present in iPhone. The relative scale is evaluated by calculating the Frobenius norm between the transnational vectors of the two frames.

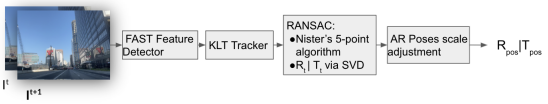


Fig. 9. Visual odometry pipeline with AR pose scale

2) INS and Estimated GPS: Initial Navigation system (INS) is a state estimation system that uses data from a IMU to continuously calculate the attitude of the system by dead reckoning (Position, Velocity and Orientation). IMU consists of a accelerometer, gyroscope and magnetometer that measures the linear acceleration, angular velocity and magnetic field of the Vehicle respectively for all 3 directions. We used a strapdown INS that integrates angular rate in attitude estimation and uses the attitude data to transform the IMU acceleration into latitude, longitude and heading. A detailed explanation and equations used for INS are given here [19] [22]. The output trajectory is prone to gyroscope and accelerometer bias and random noise that gets added during the dead reckoning process to cause a drift from the actual trajectory. We used an Extended Kalman Filter that inputs high-frequency IMU data and predicts the state of the system based on the INS one time stamp ahead. It also updates the error covariance of the EKF filter. For each low-frequency GPS data we update filter state and error covariance by computing the Kalman gain. The state for the system is given as: $x = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 & pos_N & pos_E & V_N & V_E \end{bmatrix}^T$ And the state space equation as:

$x_{t|t-1} = f(\hat{x}_{k-1|k-1}, u_k) =$

$$\begin{bmatrix} q_0 - q_1\frac{(\Delta\theta_x - \Delta\theta bias_x)}{2} - q_2\frac{(\Delta\theta_y - \Delta\theta bias_y)}{2} - q_3\frac{(\Delta\theta_z - \Delta\theta bias_z)}{2} \\ q_1 + q_0\frac{(\Delta\theta_x - \Delta\theta bias_x)}{2} - q_3\frac{(\Delta\theta_y - \Delta\theta bias_y)}{2} + q_2\frac{(\Delta\theta_z - \Delta\theta bias_z)}{2} \\ q_2 + q_3\frac{(\Delta\theta_x - \Delta\theta bias_x)}{2} - q_0\frac{(\Delta\theta_y - \Delta\theta bias_y)}{2} - q_1\frac{(\Delta\theta_z - \Delta\theta bias_z)}{2} \\ q_3 - q_2\frac{(\Delta\theta_x - \Delta\theta bias_x)}{2} + q_1\frac{(\Delta\theta_y - \Delta\theta bias_y)}{2} + q_0\frac{(\Delta\theta_z - \Delta\theta bias_z)}{2} \\ pos_N + (\Delta t)(V_N) \\ pos_E + (\Delta t)(V_E) \\ (V_N + (\Delta t)(g_N) + (\Delta V_x - \Delta V bias_x)(q_0^2 + q_1^2 - q_2^2 - q_3^2) \\ -2(\Delta V_y - \Delta V bias_y)(q_0q_3 - q_1q_2) + 2(\Delta V_z - \Delta V bias_z)(q_0q_2 + q_1q_3)) \\ (V_E + (\Delta t)(g_E) + (\Delta V_y - \Delta V bias_y)(q_0^2 - q_1^2 + q_2^2 - q_3^2) \\ -2(\Delta V_x - \Delta V bias_x)(q_0q_3 + q_1q_2) + 2(\Delta V_z - \Delta V bias_z)(q_0q_1 - q_2q_3)) \end{bmatrix} \quad (8)$$

Where:

u_k = is controlled by accelerometer and gyroscope data that has been converted to delta velocity and delta angle through trapezoidal integration

q_i : Parts of the orientation quaternion

pos_i : Position in local NED frame

$\Delta\theta bias_i$: Bias in the integrated gyroscope reading

$\Delta vbias_i$: Bias in the integrated accelerometer reading

$\Delta\theta_i$: integrated gyroscope reading

Δv_i : integrated accelerometer reading

Please refer to [24] for more information on Extended Kalman Filter and [21] for implementation details.

IV. Testing Methodology

Since we aim to show that our system performs well with the vast Google Street View dataset, we cannot test with standard autonomous driving datasets such as KITTI. This complicates direct comparisons to alternate methods and requires us to source our own ground truth. A common source of ground truth for global localization tasks is the use of an RTK-GPS system which are capable of centimeter-level accuracy in ideal conditions. However, we were not able to acquire an RTK system and instead rely on a manually annotated trajectory on top of a satellite map. For our data collection, we drove in a single lane with a front-facing camera and later approximated the camera trajectory with a piecewise linear segments. While it is difficult to ascertain the exact accuracy of this method, we expect the ground-truth trajectory to be within 1m of the true trajectory at any given point due to the width of the lane and position of the camera.

In order to evaluate our method of localization, we compare each of the estimated global locations represented by a latitude and longitude using the WGS-84 projection to our ground truth trajectory. We find the minimum distance from that point to the piecewise linear trajectory and then calculate the RMSE of a sequence of estimated points. Our estimated trajectory is formulated as a sequence of points, $T_{est} = \{(x_0, y_0), \ldots, (x_n, y_n)\}$ and our ground truth trajectory is similarly: $T_{truth} = \{(x_0, y_0), \ldots, (x_m, y_m)\}$.

We define our error function for any $p \in T_{est}$ in (9)

$$\text{err}(p) = \|\underset{i,j}{\text{argmin}}(\text{dist}(i, j, p)) - p\|_2 \quad (9)$$

where $0 \leq i < j \leq m$, $i{+}1 = j$, and $\text{dist}(i, j, p))$ represents the minimum distance from p to the line defined by the points i, j. We finally define the RMSE in (10):

$$RMSE(T_{est}) = \sqrt{\frac{1}{|T_{est}|}\sum_{p \in T_{est}} \text{err}(p)} \quad (10)$$

This method does not capture time information and thus would not tell us whether our estimated points are ahead or behind the ground truth trajectory in time. Similarly, since we do not have time information for our ground truth, we cannot plot the error over time and instead choose to plot over frames. This means our RMSE is dependent on factors such as vehicle speed, unexpected stops, etc. A possible approach might be to use the coarse GPS data to create a plot of error versus position but we chose to not implement this since it adds additional sources of error into our measurement that we cannot account.

Another unexpected source of error came from the panorama poses themselves, as provided by the Google Street View API. Despite the high-accuracy expected given the laser-range data and RTK-GPS used by Google Street View cars [7], we found significant deviations in some locations by comparing the reported coordinates to known locations on a

map. While several satellite imagery sources showed similar deviations, it is difficult to determine whether the reported coordinates are erroneous or the map coordinates themselves but regardless, this error is incorporated when we measure the estimated trajectory deviation from ground truth.

V. Results

We chose to implement our pose graph optimization in both g2o and Ceres. We compare their performance given our setup in I over a varying subsets of matched features. As previously described, this value determines how many queried panoramas a given feature must be present in to be considered for bundle adjustment. We find that the g2o solver outperforms Ceres for all of the tested feature subsets, although Ceres appears to be much more robust to the varying features, producing similar results over the different subsets. For our fixed nonlinear BA, we find that taking the intersection of all 4 panoramas produces the best results.

Our primary metric as previously described is $RMSE(T_{est})$. We tested over a 1km track in Westwood, CA that saw variable traffic density, vehicle speed, and building density and distance. Over the entire track, we achieved an $RMSE(T_{est}) = 1.529m$ using our fixed pose bundle adjustment.

Looking at a plot of $\mathrm{err}(p)$ in Figure 10 we see significant variability in the results that appear to be due to the varying environmental conditions. We see frames 5800-6800 have a low error variance which corresponds to a period in which the vehicle is stopped at a light.

TABLE I
RMSE and Std Dev over a 100m segment

Min Common Panoramas	1	2	3	4
g2o RMSE	3.015	2.092	2.631	2.26
Ceres RMSE	3.596	3.596	3.596	3.596
SciPy RMSE	2.858	2.042	1.893	1.729
g2o Std Dev	2.505	1.718	2.22	1.842
Ceres Std Dev	2.287	2.287	2.287	2.287
SciPy Std Dev	2.109	3.005	1.845	2.803

A plot of the trajectory generated by IMU dead reckoning, panoramic GPS and INS + GPS integration is shown in Figure 10. From the plot we can see a significant bias in the gyroscope and accelerometer sensor that causes a drift in the trajectory. The INS + GPS trajectory near the panoramic GPS trajectory but at instances show vast deviation from the road when plotted and analysed visually. There is a scope of improvement in the trajectory generation by filtering out erroneous IMU data, initial calibration of the INS system and better filter tuning based on sensor characteristics.

VI. Future Work

There are several aspects of our system that we believe may be significantly improved upon. One area which we did not focus on in this paper is implementing our pipeline directly on a mobile phone running in real-time, however many of the components of our pipeline are already present in many mobile devices. Visual-Inertial Odometry and Extended Kalman Filters are, for example, present on most iOS devices with ARKit and run in real-time. Feature extraction and matching has been demonstrated to run in real-time at 30fps on mobile phones [13] and newer implementations of SIFT and nearest-neighbor matching on the GPU has shown these both of these steps can be performed in less than 4ms combined [10]. Performing real-time local bundle adjustment is more challenging, however although implementations have been demonstrated to run in 500ms on a mobile phone [12] and 230ms on a GPU [11]. Finally, while we found VLDs to perform robust outlier rejection for our pipeline, we are unaware of any work to accelerate K-VLD filtering or perform it on a mobile device. Future work may examine the performance of K-VLD filtering on mobile devices and the potential to perform these computations on a remote server and transmit the final results to the client mobile device.

Another future continuation of this work involves combining the odometry estimates of the position from monocular camera and AR poses along with the positional estimate with Google Street View and IMU to have a robust end to end localization. This involves a Visual Inertial Odometry System that combines GPS, IMU and VO data using a Kalman Filter. For additional reading please refer to [22] where the authors have implemented Loosely Coupled Error State Kalman Filter algorithms for autonomous ground vehicles tested on online dataset and [23] where the authors have developed an Multi-state Constrained Extended Kalman Filter.

VII. Conclusion

Our initial motivation for this project was that the sensor suite typically required for autonomous navigation is expensive and inaccessible for the vast majority of the population with motor vehicles. To address this issue, we built a system for accurate localization that is not gated by cost and would be easily accessible to anyone with a car. Using only a monocular camera, IMU, and GPS from a smartphone, along with panoramas retrieved from Google Street View, our method is able to compute a localization and trajectories with visual inertial odometry. Importantly, our method is able to localize with lane level accuracy, which is better than GPS alone, especially in dense city environments with large buildings or structures that obstruct RF signals. Additionally, with advancement in smartphone sensor technology and general smartphone availability, our method will only become more accurate and accessible.

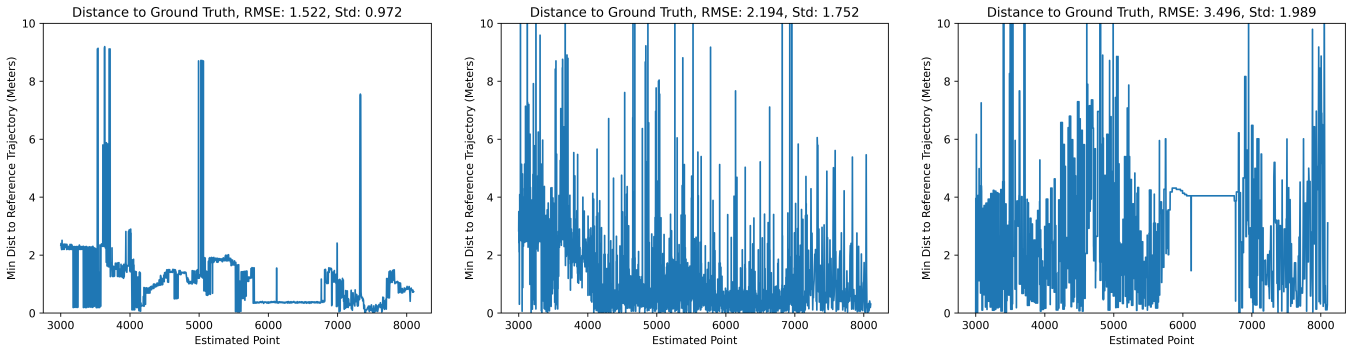
References

[1] S. LeVine, "What it really costs to turn a car into a self-driving vehicle," Quartz, 05-Mar-2017. [Online]. Available: https://qz.com/924212/what-it-really-costs-to-turn-a-car-into-a-self-driving-vehicle/.

[2] D. G. Lowe, "Distinctive image features from scale-invariant key points," International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[3] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, 2009.

[4] Z. Liu and R. Marlet, "Virtual line descriptor and semi-local graph matching method for reliable feature correspondence," Procedings of the British Machine Vision Conference 2012, 2012.



(a) SciPy, fixed pose (b) g2o, variable pose (c) Ceres, variable pose

Fig. 10. Localization error per frame over a 1 km test track

[5] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in Proc. IEEE Int. Conf. Robot. Autom., Shanghai, China, May 2011

[6] S. Agarwal and K. Mierle. Ceres Solver: Tutorial & Reference. Google Inc

[7] Google Street View: Capturing the world at street level

[8] Li Yu. Absolute Localization by Mono-camera for a Vehicle in Urban Area using Street View. Auto- matic. Université Paris sciences et lettres, 2018.

[9] P. Agarwal, W. Burgard, and L. Spinello, "Metric localization using google street view," in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015, pp. 3111–3118

[10] M. Björkman, N. Bergström and D. Kragic, "Detecting, segmenting and tracking unknown objects using multi-label MRF inference", CVIU, 118, pp. 111-127, January 2014.

[11] Fixstars, "A CUDA implementation of Bundle Adjustment", 2021. Available: https://github.com/fixstars/cuda-bundle-adjustment

[12] P. O. Fasogbon, "Depth from Small Motion using Rank-1 Initialization", arXiv [cs.CV]. 2019.

[13] G. Hall, "GPU accelerated feature algorithms for mobile devices", March 2014

[14] Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[15] Rosten, Edward, and Tom Drummond. "Fusing points and lines for high performance tracking." Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. Vol. 2. Ieee, 2005.

[16] Shi, Jianbo. "Good features to track." 1994 Proceedings of IEEE conference on computer vision and pattern recognition. IEEE, 1994.

[17] Nistér, David. "An efficient solution to the five-point relative pose problem." IEEE transactions on pattern analysis and machine intelligence 26.6 (2004): 756-770.

[18] Savage, Paul G. "Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms." Journal of guidance, control, and dynamics 21.1 (1998): 19-28.

[19] Savage, Paul G. "Strapdown inertial navigation integration algorithm design part 2: Velocity and position algorithms." Journal of Guidance, Control, and dynamics 21.2 (1998): 208-221.

[20] Thrun, Sebastian. "Probabilistic robotics." Communications of the ACM 45.3 (2002): 52-57.

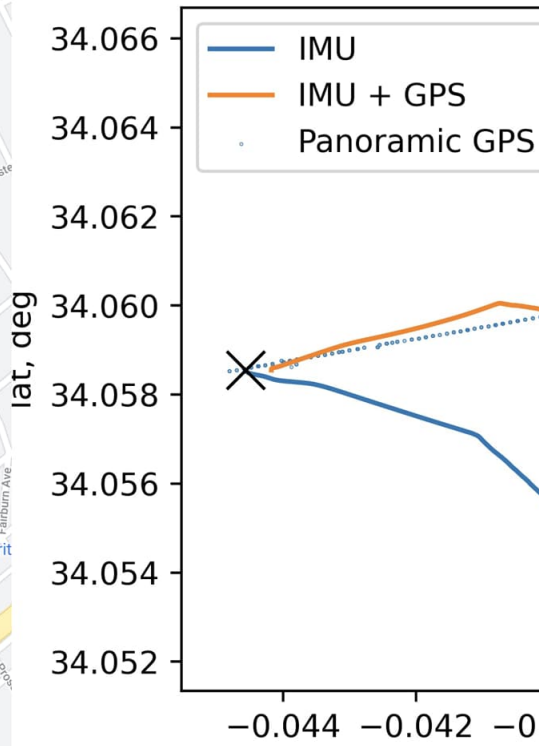
[21] pyINS, https://pyins.readthedocs.io/en/latest/index.html

[22] Burusa, Akshay Kumar. "Visual-Inertial Odometry for Autonomous Ground Vehicles." (2017).

[23] Mourikis, Anastasios I., and Stergios I. Roumeliotis. "A multi-state constraint Kalman filter for vision-aided inertial navigation." Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, 2007.

[24] "sfm — openMVG library", Openmvg.readthedocs.io, 2021. [Online]. https://openmvg.readthedocs.io/en/latest/openMVG/sfm/sfm/. [Accessed: 01- Dec- 2021].

[25] P. Moulon, P. Monasse, R. Perrot, en R. Marlet, "OpenMVG: Open multiple view geometry", in International Workshop on Reproducible Research in Pattern Recognition, 2016, bll 60–74.



(a)

Fig. 11. Comparative plot of sensor fusion algorithms