Implementation of Visual SLAM Using Single Handheld Camera

Wonhui Kim
wonhui@umich.edu

Rajat Mittal
mittalr@umich.edu

***Abstract*— The goal of this project is to solve a simultaneous localization and mapping (SLAM) problem using visual cues captured by a handheld monocular camera. We use EKF SLAM as our base algorithm. Through our project, we aim to address the added complexity of performing camera motion in 3D space, initializing, measuring, matching and detecting features from a camera in real time and managing the maximum number of features that can be detected at a given time in the image map.**

I. INTRODUCTION

SLAM in mobile robotics is mainly performed using information from sensors like LIDAR and SONAR which are expensive and not easily accessible. Visual SLAM which only requires a handheld camera is emerging as a widely researched topic with a variety of applications in the realm of augmented reality, humanoid motion, computer vision.

In this project, we aim to execute SLAM algorithm by taking inputs from camera as a sensor. Various techniques exist in literature. For the scope of this project, we use a single hand-held camera which is considered as a freely moving body and can perform random motions in 3D environment.

We follow the overall framework of the EKF SLAM algorithm. Various components of the EKF SLAM algorithm are modified to adapt to the problem we aim to address and to process the visual data. Feature matching using SURF descriptors [1] is performed to obtain new measurements at every time step. Detection and initialization of new features/landmarks is done using Shi and Tomasi Detector [2] and inverse depth parametrization [3] respectively. Our implementation is mainly based on the MonoSLAM proposed in [4] with some adjustments. We present in the sections below the implementation details and results of our algorithm.

II. PREVIOUS WORK

A. Previous Work

We mainly refer to the seminal work done by Davison *et al.* [4]. It is one of the most significant contributions to the work of visual SLAM and uses random motion of a single hand-held camera to build a real-time sparse map of features in the world coordinates.

Davison *et al.* [4] initialize new features using XYZ parametrization which lacks linearity for low parallax features with high depth uncertainty. To overcome the shortcomings of initialization using XYZ parametrization, Civera *et al.* [3] suggest feature initialization using inverse depth parametrization as it is able to overcome such linearity issues.

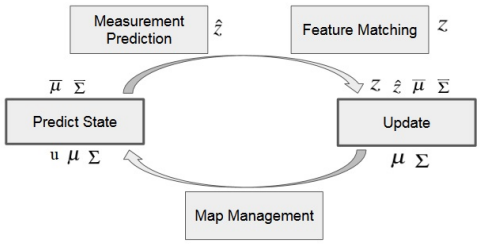

Fig. 1. System Overview

B. Contribution

We tried to experiment and improve upon the feature detection method suggested by Davison *et al.* [4]. We suggest detecting 100-200 salient points in the entire image first and then checking for overlap with existing features.

To improve upon the feature matching, we suggest three solutions: 1) periodically updating the image templates of the landmarks in the map, and 2) only considering salient pixel points around $\hat{z}$ as a potential measurement. In addition, 3) we attempt to skip the template warping step by using SURF descriptor.

III. TECHNICAL DETAILS

A. System Overview

As in the standard SLAM framework, our system repeats prediction and update steps, with several supplementary components including measurement prediction, feature matching, and map management. Fig. 1 shows the overview of the system.

B. State Representation

In our framework, we consider the following state vector:

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}^T & \mathbf{m}^T \end{bmatrix}^T \tag{1}$$

Here $\mathbf{x}$ corresponds to state vector of camera and $\mathbf{m}$ contains information about landmarks in the scene.

More specifically, the camera state vector is defined as a 13-dimensional vector as follows:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}^{WT} & \mathbf{q}^{WCT} & \mathbf{v}^{WT} & \omega^{CT} \end{bmatrix}^T \tag{2}$$

where $\mathbf{r}$ is a 3D position vector of the camera, $\mathbf{q}$ represents the quaternion of camera orientation, and $\mathbf{v}$ and ω defines linear and angular velocities relative to world frame W and the camera frame C.

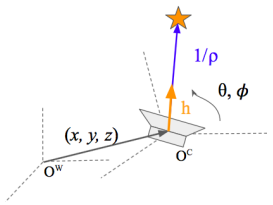

Fig. 2. Inverse depth parametrization encodes the position of camera center when the landmark is first observed, and the relative orientation of the landmark to the camera center in terms of azimuthal and elevation angles and inverse depth.

The map consists of a sparse set of landmarks in a 3D world. Inverse depth parametrization [3] represents each landmark as a 6-dimensional vector as described in Fig. 2.

$$\mathbf{m}_i = \begin{bmatrix} x_i & y_i & z_i & \theta_i & \phi_i & \rho_i \end{bmatrix}^T \tag{3}$$

C. EKF Prediction

1) Control Input: The control input to the camera motion mainly consists of linear and angular velocity and can be defined as.

$$\mathbf{u} = \begin{bmatrix} \mathbf{V}^W & \Omega^R \end{bmatrix}^T = \begin{bmatrix} \mathbf{a}^W \Delta t & \alpha^R \Delta t \end{bmatrix}^T \tag{4}$$

where $\mathbf{a}^W$ and α are unknown linear and angular accelerations, which can be considered as zero-mean Gaussian random processes.

Prediction model is used to predict the state of the camera. Since the camera motion does not influence the landmarks, only the camera state $\mathbf{x}$ of the state vector in (1) is considered for the update.

Since the camera is freely moving, we assume that no specific input commands are provided to it. Hence, we consider the input as a zero matrix. Usually, the process model is stated as $\mathbf{x}_t = f(\mathbf{u}_t, \mathbf{x}_{t-1}) + \varepsilon$ where ε is the process noise with covariance with $\varepsilon \sim \mathcal{N}(0, R)$. For our system, it can be described by the following state transition function

$$\mathbf{x}_t = f(\mathbf{u}_t, \mathbf{x}_{t-1}) = \begin{bmatrix} \mathbf{r}^W_{t-1} + (\mathbf{v}^W_{t-1} + \mathbf{V}^W)\Delta t \\ \mathbf{q}^{WR}_{t-1} \times \mathbf{q}((\omega^R_{t-1} + \Omega^R)\Delta t) \\ \mathbf{v}^W_{t-1} + \mathbf{V}^W \\ \omega^R_{t-1} + \Omega^R \end{bmatrix} \tag{5}$$

where $\mathbf{q}((\omega^R_{t-1} + \Omega^R)\Delta t)$ denotes the quaternion defined by the angle-axis rotation vector $(\omega^R_{t-1} + \Omega^R)\Delta t$.

2) Update Equations: Given the state estimates μ_{t-1} and Σ_{t-1}, and the control input $\mathbf{u}_t$, we need to make predictions during the prediction step. We can follow the same equations as of general EKF SLAM.

$$\overline{\mu}_t = g(\mathbf{u}_t, \mu_{t-1}) \tag{6}$$

$$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \tag{7}$$

The function $g(\cdot)$ is the state transition function for the complete augmented state vector $\mathbf{y}$ defined in (1). To evaluate (6), we need to compute the quaternion $\mathbf{q}(\cdot)$ as in (5).

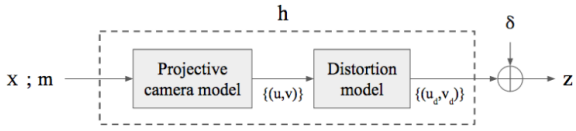

Fig. 3. The measurement model of the MonoSLAM system.

The matrices G_t in (7) requires the computation of Jacobians. In addition, given P_n, the covariance of the control input $\mathbf{u}$, R_t is $\frac{\partial \mathbf{y}}{\partial \mathbf{u}} P_n \frac{\partial \mathbf{y}}{\partial \mathbf{u}}^T$ evaluated at $u = u_t$. We skip the detailed computations here.

D. EKF Update

1) Measurement Model: In general, the measurement model is given as following:

$$z = h(\mathbf{x}; \mathbf{m}) + \delta \tag{8}$$

For the MonoSLAM, this measurement model corresponds to the mapping from camera pose and landmarks to the measurement, which is the pixel coordinate in the image frame. The measurement model is described in Fig. 3. It shows that given the camera pose and the landmark locations in the map, camera model returns the projection of landmarks in 2D image frame. For the images that involve distortion by the camera lens, we consider radial distortion model. We follow the same update equations as the standard EKF SLAM framework:

$$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1} \tag{9}$$

$$\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t)) \tag{10}$$

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t \tag{11}$$

To update the state estimates μ and Σ, EKF SLAM requires to compute the difference between the real measurement z and the expected measurement $\hat{z}$. This is called innovation, which quantifies how state estimates $\overline{\mu}$ and $\overline{\Sigma}$ should be corrected.

2) Measurement Prediction: We can compute the expected measurement by using the measurement model function h and the camera pose defined by $\overline{\mu}$. This is simply to project landmarks in the map onto the image plane determined by new camera pose. After predicting the measurements, we load the new frame and observe the landmarks, which will be mentioned in the next section.

3) Feature Matching: The feature matching step aims to obtain the measurements z by observing the landmarks in the map. To reduce the computational overhead, it searches over the local region around $\hat{z}$ to find the best match to the image template stored when the landmark is initialized. Fig. 4 depicts the way feature matching works.

The goal is to find the most likely pixel location in the new frame that corresponds to the particular landmark. The most likely location is determined as the one with maximum correlation with the initial template of the landmark. Note that corresponding to every landmark in the map there is an image template stored from the frame at which it is initialized.

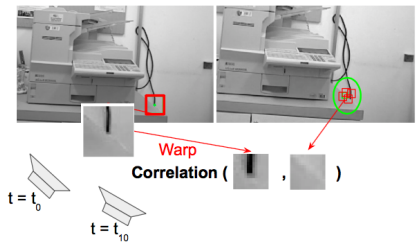

Fig. 4. Left: the initial frame that the landmark is first observed. A small patch around the detected feature point is stored. Right: Here we hope to observe the landmark again in the later frame, given the predicted location by zhat.

Fig. 5. Given the 95% confidence region represented as orange ellipse, we only consider the salient pixel points for feature matching, instead of considering every point inside the orange region. This significantly reduces the running time. (1.2024 sec per iteration on average)

Davison *et al.* [4] compute the correlation between two image templates directly from their pixel values. The appearance of the landmark may vary due to the camera movement, so the template of the landmark is warped considering the change in camera pose before computing the correlation. Warping the image template, however, involves errors especially when the landmark is tracked for a long time period with large appearance change. Moreover, computing correlation for all the pixel points around $\hat{z}$ is not efficient.

To avoid those issues, we suggest three solutions: 1) periodically updating the image templates of the landmarks, 2) only considering salient pixel points around $\hat{z}$ as a potential measurement, and 3) skipping the template warping step by using SURF descriptor.

When the landmark is initialized after its first observation, a small image template is stored in the map which is not updated until it is deleted. In case where the landmark experiences severe appearance changes while it remains in the map for a long while, the initial template is not reliable which may result in small correlation even at the correct measurement location. We observed that when the camera motion is along depth direction or when rotating around an axis, this issue arises. To fix the problem of degrading initial template, we update the template after every 50 frames.

As another contribution in feature matching step, we focus on salient pixel points when obtaining the measurement z, instead of considering all the pixels within the region of 95% confidence interval as in [4]. In Fig. 5, the orange ellipse represents the 95% confidence region around the predicted measurement $\hat{z}$. Let's say there are N pixels in the ellipse. While the approach of Davison *et al.* generates N image templates and computes correlations for each of them, we detect salient points within the ellipse and generate templates only around those points. We use corner detector of Shi *et al.* [2], and our approach usually gives 10-100 times smaller number of patches for each ellipse while preserving the saliency of the templates.

Fig. 6. Green points represent the previously matched features, and the points with circle represents the newly detected features. We can observe that the newly detected features lie away from the previously matched features.

When computing the correlation between two image templates, we use SURF descriptor [1] instead of using raw pixel values to increase the robustness of the descriptor. When using the SURF, we simply skipped the template warping.

E. Map Management

Map management is essential as it allows the algorithm to keep track of existing landmarks, initialize new ones and delete unreliable landmarks from the map. It consists of three major components:

1) Feature Deletion: Sometimes the newly detected features are not properly initialized. It means that the feature detected is not a strong feature, and hence does not represent any salient point. Also the feature may be detected at an image frame when the camera image is blurry. As a result, the template initialized along with the feature is blurry and not entirely representative of its surroundings. These features have a higher tendency to not get matched in the feature matching step and hence do not contribute to the update equation. To reduce the computational cost, it is better to delete such unreliable features that do not get matched after repeated matching attempts. A landmark is deleted from the map if the following criteria holds:

$$\frac{No.\ of\ times\ matched}{Total\ no.\ of\ matching\ attempts} < Threshold \qquad (12)$$

We use 0.5 as threshold.

2) Feature Detection: For a map to contain enough information about the environment, we need to keep the total number of features that are matched in a frame above a certain threshold. Map management function counts the number of matched features at every frame, and detects additional landmarks if the number is less than the threshold.

Initially 100-200 strongest salient points are detected in the image using Shi and Tomasi feature detector [2], and we pick one of them at random. If no other feature points lie around this point, the feature point is initialized. Otherwise, the feature point is discarded, and the procedure is repeated with another point. This is to avoid clutter and prevent two feature points from being very close to each other. Fig. 6 describes the example results of feature detection. Note that the new

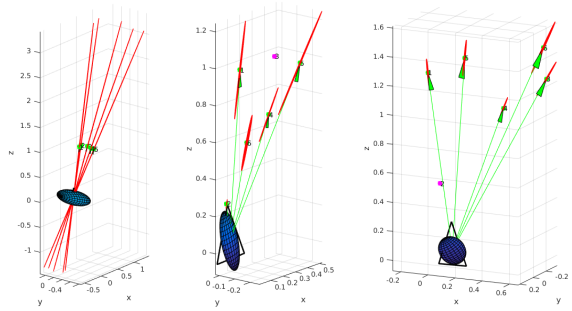

Fig. 7. Visualization of 3D map. The uncertainty ellipsoid around each landmark is shown in red, and the uncertainty ellipsoid of the camera position in blue. We can see that the uncertainty gets reduced with increasing time steps.

features are initialized away from the features matched in the previous time step. It may still be initialized close to the features which are present the state vector but were not matched in the previous time step.

3) Feature Initialization: After a new feature is detected in the image, it must be initialized in a 3D map. One option is to initialize the feature using XYZ parametrization, which encodes each landmark as a 3-dimensional XYZ coordinate. Due to the lack of linearity for low parallax feature, however, detected features need to be observed for a certain time frame until the uncertainty in depth is reduced before they can be added to the state vector. Civera *et al.* [3] proposed using inverse depth parametrization for initialization as inverse depth is linear and hence suitable for features at both high and low parallax, and can be immediately added to the state vector. The state of a feature using inverse depth parametrization is defined in eq. (3). For the initialization, Civera *et al.* [3] suggest assigning 0.1 as initial value to ρ.

The azimuth angle θ_i and elevation angle ϕ_i can be obtained from the direction vector h joining the camera center to the feature location.

Due to restriction of space, we do not show here the Jacobian computations. They can be read and understood in more detail from [5].

IV. EXPERIMENTS

A. Implementation

We are implementing the visual SLAM in MATLAB which provides a better environment for debugging the program and visualizations. To detect features and extract descriptors from image, we use functions from MATLAB Computer Vision toolbox. For some part of low-level functions such as computing Jacobians, we refer the calculations shown in [5] and the existing function codes available along with [4] .

B. Feature Detection Methods

We tested two methods for feature detection. Fig. 8 shows the output image with the method suggested by Davison *et al.* [4]. According to the method, a random bounding box of 80×80 pixel is created such that none of the existing

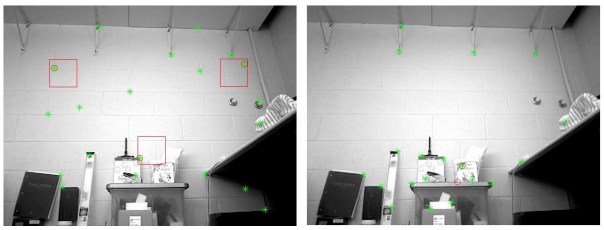

Fig. 8. Left: Feature detection using method suggested by [4]. The red box represents a random region of 80×80 pixels inside which a new feature is initialized. Right: Feature detection using our method. Green points represent features matched from the previous time step, and red circles show newly detected feature points.

matched features lie within this bounding box. A new feature is then detected and initialized within this bounding box. Through experiments, we observed that the features detected within this bounding box had high chances of not having any salient points, as the position of the box is randomly initialized. This is especially true when the image has several regions where no noticeable feature can be detected.

We also tested with our own method, the output figure for which can be seen in Fig. 8. Our feature detection method detects salient points better than the method suggested in [4].

C. Results

We have created our own dataset which has relatively less degrees of freedom in camera motion. The images are acquired with an ASUS Xtion Pro Live camera at VGA resolution (640x480) and full speed (30Hz) where the intrinsic camera parameters are known.

When capturing the sequences, we guide the camera motion along a flat plane in the horizontal or vertical direction with known length. However, as the camera is moved by hand, it is difficult to perfectly stabilize the camera motion or to move it with a constant velocity. Hence, the collected data is quite noisy. The objective of performing experiments on our dataset is to check if our system works properly in the real scene with freely moving handheld camera, and also to explore any unusual behavior of our system. Each data sequence presented below has 600-800 frames, which are converted to grayscale images when loaded at each iteration.

For the experiments with our dataset, we kept the total number of landmarks in the scene to at least 12, and set the image patch size as 49x49 for feature matching step.

1) Dataset 1. Translation with constant depth: This image sequence captures translational motion of the camera with constant depth, which mainly consists of nearby features within 1.5 meter. The camera first moves along the horizontal direction by about 70 cm, and then changes the direction to go downward by 60 cm. Fig. 9 shows the resulting trajectory with our SLAM system which is L-shaped.

2) Dataset 2. Translation with varying depth: We use the same environment as used in dataset 1. Here the sequence captures the camera translation motion along the depth direction (zooming) as well as translation along the horizontal direction. The camera is moved about 60cm along

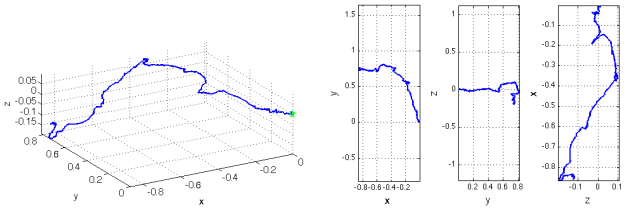

Fig. 9. The resulting trajectory with dataset 1 which involves translational camera motion with constant depth. Z-axis of the top plot is heading towards the depth direction, and in the plot corresponds to 10cm.

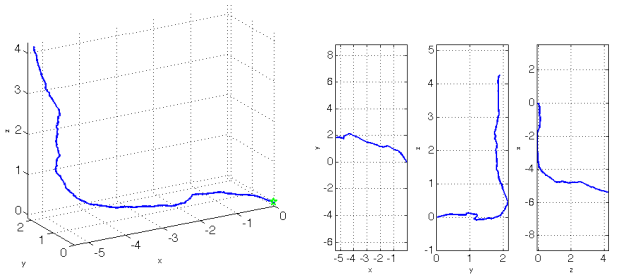

Fig. 10. The resulting trajectory with dataset 2 which involves translational camera motion with varying depth. Z-axis of the top plot is heading towards the depth direction, and one unit in the plot corresponds to 10cm.

the horizontal direction, and then moved about 50cm in the depth direction towards the objects.

Moving the camera along the depth direction changes the scale of the landmarks. This will affect the feature matching step because we use templates with fixed size. This image sequence is intended to observe the behavior when the landmark appearance experiences change in scale. Since the camera moves with small linear velocity in our dataset, feature matching works properly even with varying scale. The resulting trajectory can be seen in Fig. 10.

3) Dataset 3. In-plane rotation: We captured this sequence to check the behavior of our system when the camera has in-plane rotation. The first half includes the clockwise rotation of the camera, and then it rotates back counter-clockwise towards the initial position. The camera center has almost zero translation. Since the components in the scene do not change for this sequence, most observed landmarks are re-observed for long time periods.

Fig. 12 shows how the camera pose estimate is drifted away from origin during the entire sequence. Ideally it should stay at the origin, but it shows that the camera pose estimate drifts apart from the origin by about 0.8cm in the worst case. We think that 0.8cm of deviation occurs due to: 1) the camera lens which is not exactly aligned with the rotation axis, and also 2) our data acquisition process which is relatively unstable and hence susceptible to noise. Fig. 11 shows the resulting estimates of the linear and angular velocities.

This dataset is mainly designed to validate the periodic template updating scheme which we have suggested as an alternative update method and which has been described in detail in the feature matching section. Fig. 12 compares the resulting trajectories, one after including the periodic

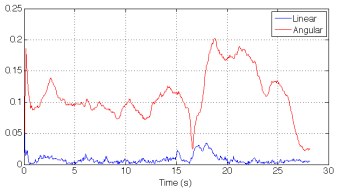

Fig. 11. The norm of linear and angular velocity estimates across the frames. Since the dataset 3 involves a camera-centered in-plane rotation, it makes sense to have almost zero linear velocity and non-zero constant angular velocity. The peak around 17s is the point where the camera starts rotating back towards the initial pose.

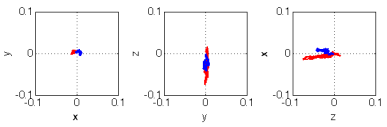

Fig. 12. The resulting trajectory with dataset 3, projected onto XY, YZ, and ZX planes. The blue corresponds to the result with applying the periodic template update that we suggested, and the red one is from not using it. 1cm corresponds to the 0.1 of the plot. We can observe that the estimated camera state stays near the initial position, and periodically updating the template gives more desirable results.

template update step, and the other where the map is not updated periodically.

4) Dataset 4. Rotation with varying azimuth angle: We created this dataset to validate our system when the camera motion involves rotation in azimuthal orientation. The camera motion has almost zero translation. This sequence is intended to cover changes in landmark appearance. Fig. 13 shows the resulting trajectory and Fig. 14 shows the estimated velocities.

5) Dataset 5: More challenging camera motion: For quantitative analysis, we applied our framework to the dataset with ground-truth trajectories. The sequence of copyroom scene[6] consists of relatively close landmark points and includes complex camera motion. The velocity is not constant which contradicts the underlying assumption of our system. Fig. 15 shows how the linear velocity of the camera changes rapidly over time. One of the major reasons for poor performance of our system on this sequence is the landmarks at close distance. This is critical because close landmarks moves relatively by a large amount even with a small camera movement, which can significantly increase the uncertainty. Some previous work already reports similar problems. As a strategy to solve this issue, Civera *et al.* [7] use partial update scheme, which updates the state estimates using only distant landmarks and then picks part of close landmarks with low uncertainty to make additional updates.

We can see that our state estimates show a large difference especially around where the ground-truth trajectory shows rapid movement.

6) Dataset 6: Challenging camera motion: We test the sequence of desk scene from TUM dataset [8] which also provides us with ground truth data. The camera motion trajectory is very challenging as it involves random motion in all three directions. The comparison of our trajectory with

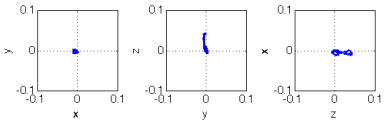

Fig. 13. The resulting trajectory with dataset 4, projected onto XY, YZ, and ZX planes. The estimated camera state stays around the origin.

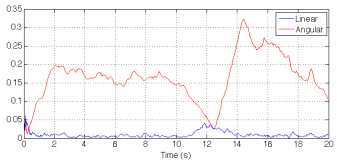

Fig. 14. The norm of linear and angular velocity estimates across the frames. Since the dataset 4 involves a camera-centered azimuthal rotation, it makes sense to have almost zero linear velocity and non-zero constant angular velocity. The peak around 12.5s is the point where the camera starts rotating back towards the initial pose.

that of ground truth can be seen in figure 17.

As the camera does not have information about its initial orientation, the X, Y and Z axis initially assigned by our MonoSLAM algorithm varies from that of the dataset. Hence the trajectories in the images are not completely aligned. As we can see, our algorithm is able to follow a trajectory similar to the ground truth trajectory until 200 iterations. Beyond that the trajectory diverges from the ground truth trajectory. As seen in the dataset images, we believe that the cause of this divergence is that after first 200 iterations the motion gets more complex and the camera rotates as well as translates along Y and Z axis.

V. CONCLUSION

In this project, we implement MonoSLAM which is one of the fundamental visual SLAM algorithms. We successfully implemented and integrated various modules of MonoSLAM such as prediction, measurement, feature matching, map management. Through experiments and further reading, we figured out that the major limitation of MonoSLAM framework comes from feature mismatch. Hence we particularly focused on feature detection and feature matching modules with some variations, to further optimise the framework.

To validate our SLAM implementation, we built our own dataset by capturing sequences of simple motions with a handheld camera. As shown through comparisons with ground truth data, we were able to achieve relatively accurate results with simpler camera motions. Integrating vision with SLAM is a major area of research today and paves way for several exciting applications such as augmented reality, kinect mapping, autonomous vehicles.

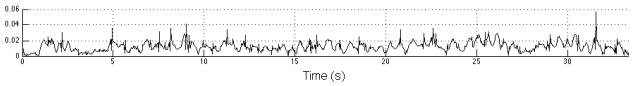

Fig. 15. The linear velocity of camera for the ground-truth copyroom dataset. We can see that it involves rapid changes in motion.

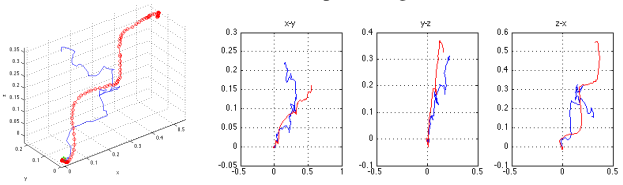

Fig. 16. The resulting trajectory from copyroom dataset. Red plot shows the ground truth trajectory and the blue shows the result of applying our system.

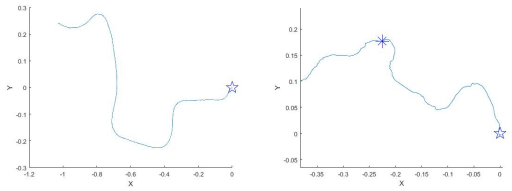

Fig. 17. Left: Ground truth trajectory along the XY axis. Right: Actual MonoSLAM trajectory along XY axis. 200th iteration denoted by *. After this camera motion estimate starts diverging from ground truth.

REFERENCES

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[2] Jianbo Shi and Carlo Tomasi. “Good features to track”. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on*. IEEE. 1994, pp. 593–600.

[3] Javier Civera, Andrew J Davison, and JM Martinez Montiel. “Inverse depth parametrization for monocular SLAM”. In: *Robotics, IEEE Transactions on* 24.5 (2008), pp. 932–945.

[4] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. “MonoSLAM: Real-time single camera SLAM”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.6 (2007), pp. 1052–1067.

[5] Sven Albrecht. “An Analysis of Visual Mono-SLAM”. PhD thesis. Citeseer, 2009.

[6] Qian-Yi Zhou and Vladlen Koltun. “Dense scene reconstruction with points of interest”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 112.

[7] Javier Civera, Oscar G Grasa, Andrew J Davison, and JMM Montiel. “1-Point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry”. In: *Journal of Field Robotics* 27.5 (2010), pp. 609–631.

[8] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. 2012.