RBE/CS549: Project 4 - Classical Visual-Inertial Odometry

Sanya Gulati sgulati@wpi.edu
Siyuan Huang shuang4@wpi.edu
Yijia Wu ywu21@wpi.edu

I. Introduction

In this project, we implemented the seven functions of stereo Multi-State Constraint Kalman Filter (MSCKF)[1]. As this paper is an extension of the original MSCKF paper[2], we have taken it as a reference.

II. Phase 1: stereo MSCKF

A. Initialize gravity and bias

The gravity and gyroscope bias are initialized with the first 200 messages from IMU when the robot is placed statically. The gyroscope bias b_g is initialized as the average of these 200 gyroscope readings. The gravity g is initialized as $[0, 0, g_{norm}]$, and g_norm is the norm of the first 200 accelerometer readings. The orientation is initialized as the quaternion from $-g$ to g_{norm}.

B. Batch imu processing

When we receive a new feature, we want to first process all the IMU messages received prior to the new feature's time stamp. For every IMU message within the IMU message buffer that's received prior to the feature time stamp, we update our state estimation using the process model.

C. Process model

We model the IMU system model as a continuous-time in equation 1.

$$\begin{aligned} &{}^I_G\dot{\bar{q}}(t) = \frac{1}{2}\Omega(\omega(t)){}^I_G\bar{q}(t), \\ &\dot{b}_g(t) = n_{wg}(t), \\ &{}^G\dot{v}_I(t) = {}^G a(t), \\ &\dot{b}_a(t) = n_{wa}(t), \\ &{}^G\dot{(p)}_I(t) = {}^G v_I(t) \end{aligned} \tag{1}$$

Where ${}^I_G\bar{q}(t)$ is the unit quaternion describing the rotation from global frame {G} to IMU frame {I}. $\omega(t) = [\omega_x, \omega_y, \omega_z]^T$ is the rotational velocity in IMU frame, and

$$\Omega(\omega) = \begin{bmatrix} -\lfloor\omega_\times\rfloor & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{2}$$

$$\lfloor\omega_\times\rfloor = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3}$$

The gyroscope and accelerometer measurements, ω_m and a_m can be written as

$$\omega_m = \omega + b_g + n_g \tag{4}$$

$$a_m = {}^I_G R({}^G a - {}^G g + 2\lfloor\omega_{G\times}\rfloor {}^G v_I + 2\lfloor\omega_{G\times}\rfloor^2 {}^G p_I) + b_a + n_a \tag{5}$$

where ${}^I_G R$ is the rotation matrix calculated from quaternion ${}^I_G\bar{q}$ Then apply expectation operator on equation 1 we obtain the equations for propagating IMU state estimates:

$$\begin{aligned} &{}^I_G\dot{\hat{\bar{q}}}(t) = \frac{1}{2}\Omega(\hat{\omega}^I_G)\hat{\bar{q}}, \\ &\dot{\hat{b}}_g = 0_{3\times1}, \\ &{}^G\dot{\hat{v}}_I = C^T_{\hat{q}}\hat{a} - 2\lfloor\omega_{G\times}\rfloor {}^G\hat{v}_I + \lfloor\omega_{G\times}\rfloor^2 {}^G\hat{p}_I + {}^G g, \\ &\dot{\hat{b}}_a = 0_{3\times1}, \\ &{}^G\dot{\hat{p}}_I = {}^G\hat{v}_I \end{aligned} \tag{6}$$

The linearized continuous-time model for IMU error-state is:

$$\dot{\widetilde{X}}_{IMU} = F\widetilde{X} + Gn_{IMU} \tag{7}$$

Where F and G are

$$F = \begin{bmatrix} \lfloor\hat{\omega}_\times\rfloor & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^I_G\hat{q})^T\lfloor\hat{a}_\times\rfloor & 0_{3\times3} & 0_{3\times3} & -C({}^I_G\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{8}$$

$$G = \begin{bmatrix} -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C({}^I_G\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{9}$$

D. Predict new state

Every time we received a new IMU measurement, we use 4^{th} order Runge-Kutta numerical integration to propagate our estimation for the next state.

E. State augmentation

In the state_augmentation function, we update the camera position ${}^{G}p_C$ and orientation ${}^{C}_{G}q$ with the last IMU state and augment the state covariance matrix P.

The pose of the camera can be computed as,

$$ {}^{G}p_C = {}^{G}p_I + C({}^{C}_{G}q)^T \, {}^{I}p_C \tag{10} $$

$$ {}^{C}_{G}q = {}^{C}_{I}q \otimes {}^{I}_{G}q \tag{11} $$

$$ J = \begin{bmatrix} J_1 & O_{6\times 6N} \end{bmatrix} \tag{12} $$

$$ J_1 = \begin{bmatrix} C({}^{I}_{C}q) & 0_{3\times 9} & 0_{3\times 3} & I_3 & 0_{3\times 3} \\ \lfloor C({}^{I}_{G}q)^T \, {}^{I}p_C \times \rfloor & 0_{3\times 9} & I_3 & 0_{3\times 3} & I_3 \end{bmatrix} \tag{13} $$

$$ P_{k|k} = \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix} P_{K|K} \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix}^T \tag{14} $$

F. Add feature observations

This function can update the detected feature in the latest frame to the feature map. Each feature will be added to the feature map with its feature ID and current state ID. The feature that was collected with state ID i and feature ID j is represented as

$$ Z_i^j = \begin{bmatrix} u_{i,1}^j & v_{i,1}^j & u_{i,2}^j & v_{i,2}^j \end{bmatrix}^T \tag{15} $$

1 represents the left camera, and 2 represents the right camera.

G. Measurement update

The measurement_update function takes measurement matrix H and residual matrix r as input to compute the Kalman gain K, and then updates the IMU state X_{IMU}, camera state, and state covariance matrix P.

The measurement matrix H is a matrix with block rows $H^(j)$, $j = 1...L$. L is the number of all detected features. If the number of row (feature) is larger than the number of state X components (21), we employ QR decomposition for the matrix H. With the reduced mode of numpy.linalg.qr function, we can directly get Q and T_H.

$$ H = \begin{bmatrix} Q & Q_2 \end{bmatrix} \begin{bmatrix} T_H \\ O \end{bmatrix} \tag{16} $$

The matrix Q can then be used to compute residual r_n as

$$ r_n = Q^T r = T_H \widetilde{X} + n_n \tag{17} $$

R_n is the covariance matrix of the noise vector n_n. σ_{im}^2 is the noise of observation. q is the number of rows of matrix Q.

$$ R_n = \sigma_{im}^2 I_{q\times q} \tag{18} $$

The Kalman gain K can be computed with the following equation

$$ K = PT_H^T(T_H P T_H^T + R_n)^(- 1) \tag{19} $$

However, because the matrix inverse computation is unstable, we can change it as solving the $Ax = b$ problem, x is K^T, A is $S = T_H P T_H^T + R_n$, and b is $T_H P$. A is S because $S = S^T$, both matrix P and R_n are symmetric.

$$ SK^T = T_H P \tag{20} $$

After getting the Kalman gain K, we can use it to compute the correction for state ΔX as

$$ \Delta X = K r_n \tag{21} $$

Finally, the state covariance matrix P can be updated with

$$ P_{k+1|k+1} = (I_{k\times k} - KT_H)P_{k|k} \tag{22} $$

H. Trajectory error evaluation

In this section, we plot the error between Ground Truth and Estimate Trajectory with the rpg_trajectory_evaluation toolbox[3]. The absolute median trajectory error (ATE) is 0.06910338087405558 m and the root mean square translation error (RMSE) is 0.081715762810781 m.

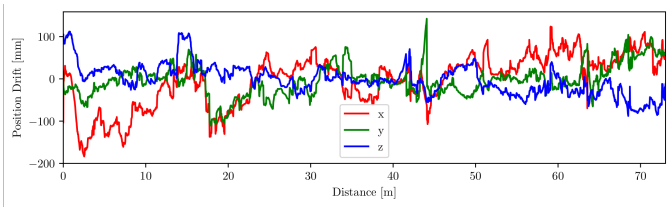

Fig. 1: Translation Error

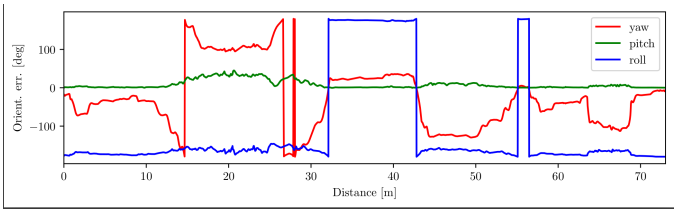

Fig. 2: Rotation Error

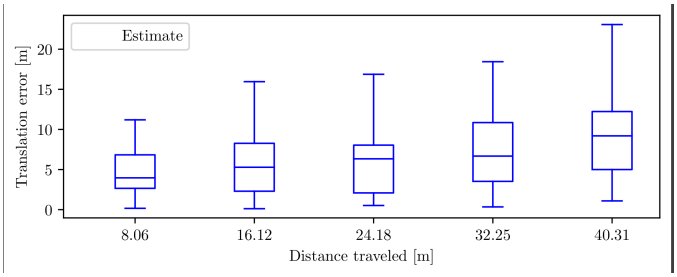

Fig. 3: Relative Translation Error

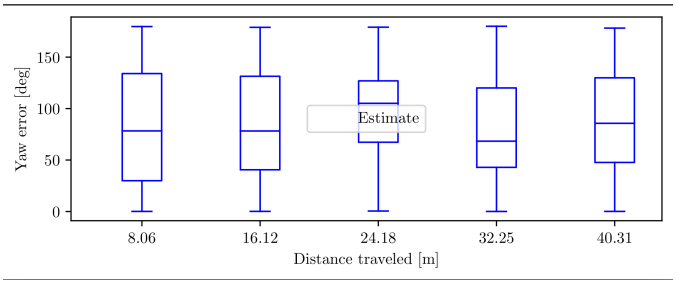

Fig. 4: Relative Yaw Error

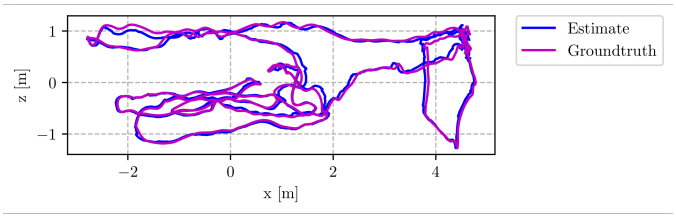

Fig. 5: Trajectory Side View

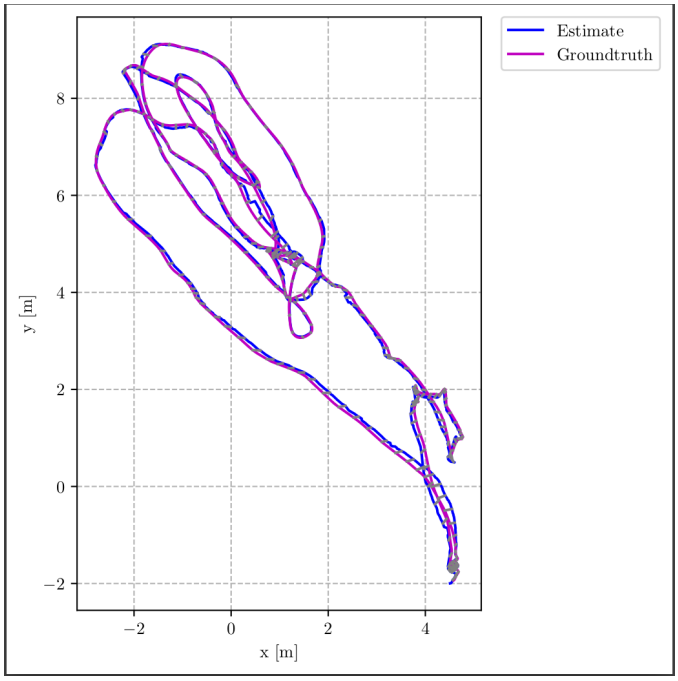

Fig. 6: Trajectory Top View

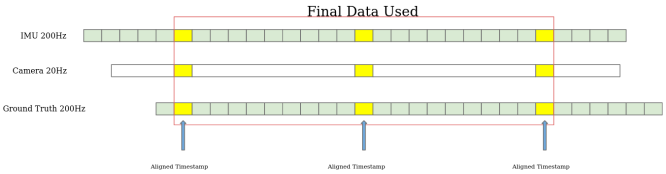

Fig. 7: Data Preprocessing for time stamp alignment

III. Phase 2: Deep VIO

In this phase, we explored using deep learning approach to solve Visual Inertial Odometry. We first preprocessed the dataset for time stamp alignment and then used the processed dataset to train Visual-only odometry network and Inertial-only odometry network. We also proposed a network architecture for fusing visual and inertial signals to obtain odometry. However, because of time constraints and unsolved issues in our VO and IO network, we didn't get a chance to actually train the VIO network. We also evaluated our odometry prediction result with the ground truth using *rpg_trajectory_evaluation* toolbox similarly as we have done for Phase 1.

A. Data preprocessing

The dataset we used to train and test our models is the Machine Hall dataset in the EUROC dataset. In this dataset, camera frames are given at 20Hz, IMU frames are given at 200Hz, and the ground truth captured by the VICON motion capture system is provided at 200Hz. We found that all three sources of the data are time-stamped but not aligned. In order to use the dataset, we had to find the common beginning and common end time stamps to crop out and align the data for later use. Figure 7 is an image illustration of the process.

Another crucial data preprocessing step that we missed and didn't realize earlier was that all the positions in the ground truth data are in the world frame. In order to use the ground truth data, we have to convert it into a body frame for relative poses. We believe that this is the major reason why our networks failed. Unfortunately, we didn't realize this earlier thus we didn't have time to fix this in our submission.

B. Deep VO architecture

We designed the architecture as shown in Fig. 8 based on the pose estimator network in [5]. The input is the stack of two images in time (t-1) and t. The network architecture starts with 10 2D convolution layers, follows by a max pool layer and some fully connected linear layers which finally output the estimated frame-to-frame position and orientation. The quaternion estimation is normalized to be a unit quaternion.

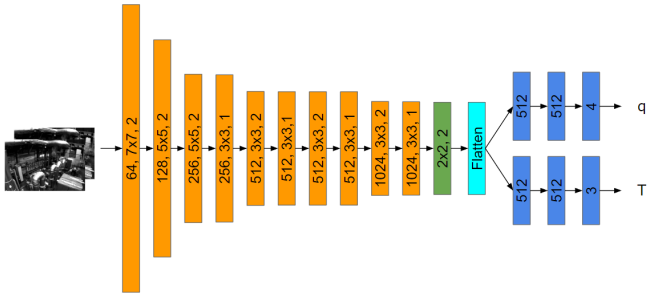

Fig. 8: Deep VO architecture

C. Deep IO architecture

We used Temporal Convolution based network architecture to design our Inertial odometry network. As shown in Figure 9, given 200 frames of IMU data, which contains 6 channels linear and angular accelerations, we use our network to predict

the relative pose between the last 100 frames as 7 channels outputs, 3 for translation and 4 for orientation in quaternion. The input data first goes through 8 connected 1D convolution layers with 265 channels and a kernel size of 2. The output is flattened and fed to two more fully connected layers to get the final 7×1 desired output.

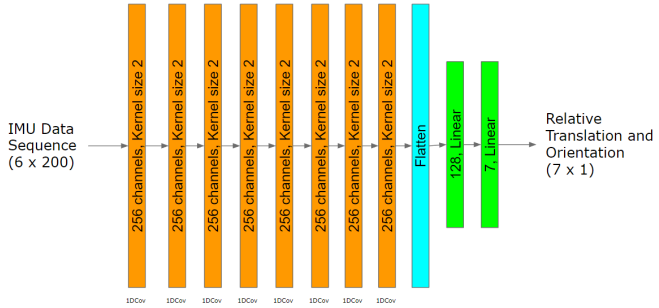

Fig. 9: Deep IO architecture

D. Deep VIO architecture

The Deep VIO architecture takes flattened outputs from VO and IO as inputs, concatenates them, and similar to the architecture of IO, we add two linear layers, and finally predict the final output, the orientation (q), and the translation (T).

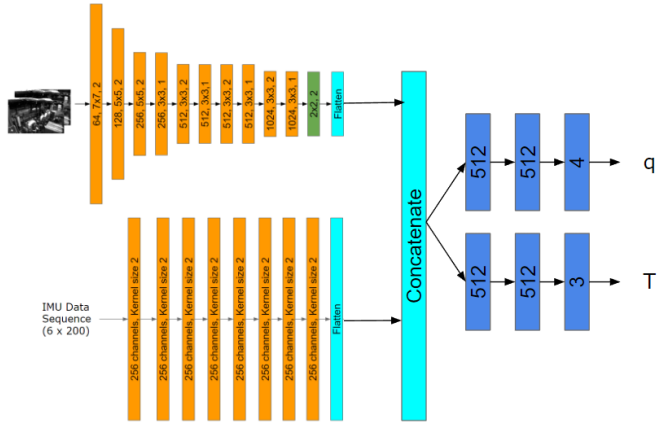

Fig. 10: Deep VIO architecture

E. Loss function

The loss function was chosen to be a weighted combination of position error L_p and orientation error L_q of the frame-to-frame pose estimation.

The position error L_p is the L2 norm of the estimated position p^* and ground truth position p.

$$L_p = ||p - p^*||_2 \tag{23}$$

The orientation error L_q is calculated as

$$L_q = \sum ||1 - ||\boldsymbol{q} \cdot \boldsymbol{q}^*||| \tag{24}$$

q is the ground truth quaternion. q^* is the normalized estimated quaternion. If they are the same, the error should be zero.

Finally, they are combined with a scale factor β because the value of position error is normally larger than the value of orientation error. This value change between dataset. We chose β=150 according to the loss value ratio of our training set.

$$L = L_p + \beta L_q \tag{25}$$

F. Training and testing pipeline

We have implemented, trained, and tested the Deep VO and IO architecture. The training dataset contains the trajectory of Machine Hall 02-05 with both stereo camera, IMU reading, and ground truth trajectory from Vicon. The networks were trained with a batch size of 32, Adam optimizer with β_1=0.9, β_2=0.999, and an initial learning rate of 0.0001 with 0.5 decay rate per 300 iterations (almost an epoch). After training, we use the saved model to test the Machine Hall 01 dataset. The result is shown in the next section.

G. Trajectory Error Evaluation

The Error between Ground Truth and Estimate Trajectory using the rpg_trajectory_evaluation toolbox[3] is shown below. The absolute median trajectory error (ATE) is 123.4665558045848 m and the root mean square translation error (RMSE) is 128.34377138779902 m. The average frame-to-frame translation error is 0.0011 m and the average quaternion error is 1.8320e-05. However, though the frame-to-frame error is small when they were accumulated, the final absolute error is too large.

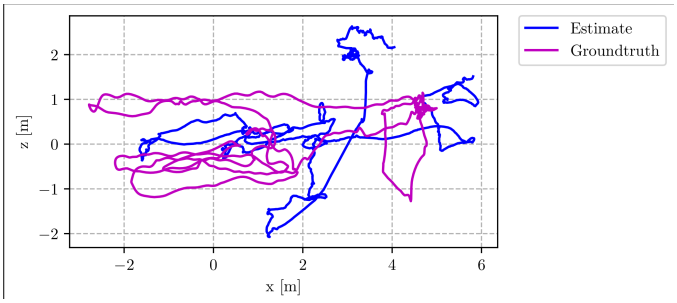

Fig. 11: Trajectory Side View

REFERENCES

[1] K. Sun et al., "Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight." arXiv, 2017. doi: 10.48550/ARXIV.1712.00036.

[2] A. I. Mourikis and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation," Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, Apr. 2007. doi: 10.1109/robot.2007.364024.

[3] Z. Zhang and D. Scaramuzza, "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Oct. 2018. doi: 10.1109/iros.2018.8593941.

[4] Weber, D., Gühmann, C., Seel, T. (2020). Neural Networks Versus Conventional Filters for Inertial-Sensor-based Attitude Estimation. arXiv. https://doi.org/10.48550/ARXIV.2005.06897

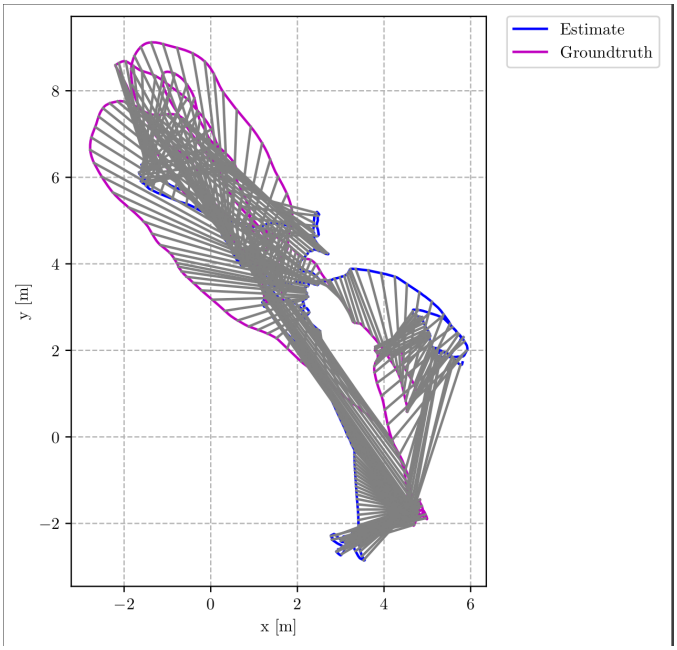

Fig. 12: Trajectory Top View

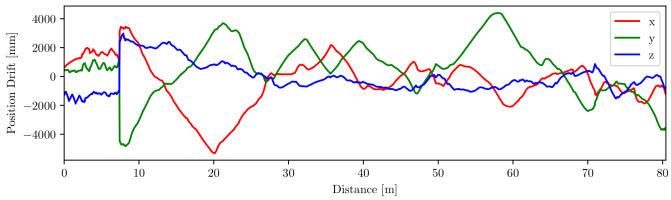

Fig. 13: Translation Error

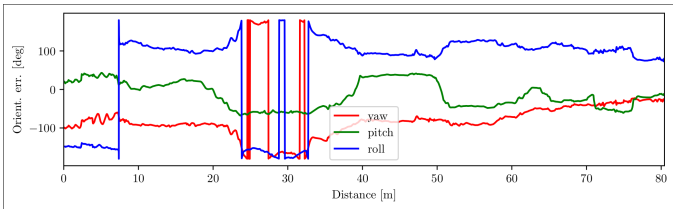

Fig. 14: Rotation Error

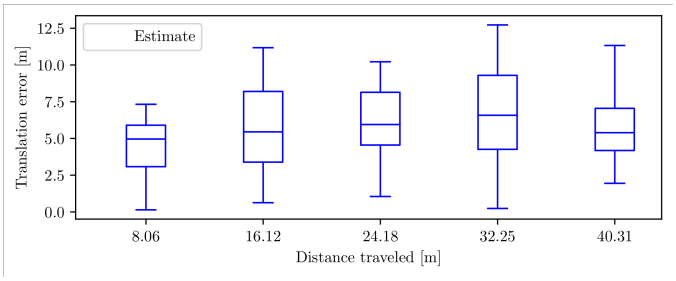

Fig. 15: Relative Translation Error

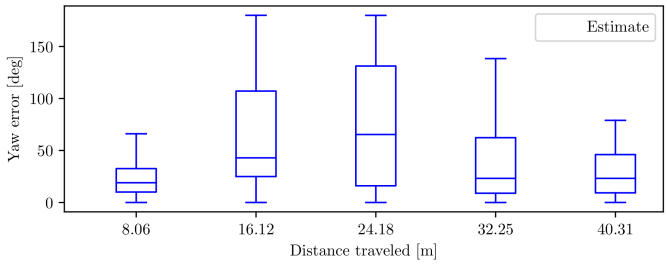

Fig. 16: Relative Yaw Error

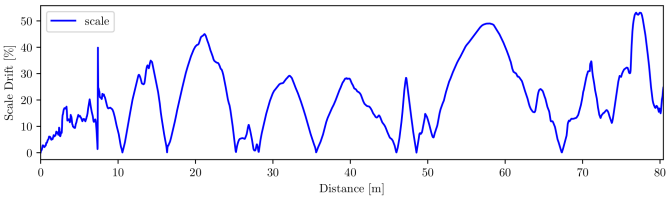

Fig. 17: Scale Error

[5] R. Li, S. Wang, Z. Long, and D. Gu, "UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning." arXiv, 2017. doi: 10.48550/ARXIV.1709.06841.