RBE/CS 549 Project 4
Deep and Un-Deep Visual Inertial Odometry

Blake Bruell
Worcester Polytechnic Institute
babruell@wpi.edu

Cole Parks
Worcester Polytechnic Institute
cparks@wpi.edu

***Abstract*—In this project, we implement a visual-inertial odometry system that fuses data from IMUs and cameras to estimate the motion of a robot using. In Phase 1 we implement the stereo Multi-State Constraint Kalman Filter (MSCKF) to estimate the state of the robot, using stereo cameras, evaluated on the EuRoC dataset.**

I. Phase 1

A. Initializing Gravity and Bias

A six-degree-of-freedom IMU, used to measure rotation and acceleration using a gyroscope and accelerometer, necessitates calibration to correct bias in the sensors. This is done by calculating the mean of a set of stationary readings to determine the bias, since the only force acting on the accelerometer is gravity and there should be no torques acting on the gyroscope. The gyro bias is then subtracted from all subsequent gyroscope readings, and the gravity is initialized as [0,0, -gravity_norm]. This estimation of gravity is not perfect, so the gravity vector is updated during the filter process.

B. Batch IMU Processing

Since the features and the IMU data do not come in at the same rate, we want to batch IMU messages. In practice, when feature is received, all IMU messages in the IMU message buffer which have a timestamp prior to the feature's timestamp are processed using the IMU process model. This is ensures that the IMU state is updated to the time of the feature observation.

C. Process Model

Batch processing is responsible for processing IMU messages and updating the IMU state within a specified time frame. It iterates through the IMU messages, discarding those already processed and stopping at the time bound. For each unprocessed message, it applies the process model to update the IMU state based on angular velocity and linear acceleration measurements. The function updates the IMU state's timestamp and ID, and removes processed messages from the buffer. This function is crucial for accurate state propagation and synchronization between the IMU and Visual Odometry components, contributing to reliable sensor fusion and localization.

Mathematically, the process model can be described as follows [1]:

$$\begin{aligned}
{}^{I}_{G}\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\Omega(\hat{\boldsymbol{\omega}}(t))\,{}^{I}_{G}\hat{\mathbf{q}},\\
{}^{G}\dot{\hat{\mathbf{v}}} &= C({}^{I}_{G}\hat{\mathbf{q}})^{\top}\hat{\mathbf{a}} + {}^{G}g,\\
\dot{\hat{b}}_g &= \mathbf{0}_{3\times 1},\\
\dot{\hat{b}}_a &= \mathbf{0}_{3\times 1},\\
{}^{G}\dot{\hat{\mathbf{p}}}_I &= {}^{G}\hat{\mathbf{v}},\\
{}^{I}_{C}\dot{\hat{\mathbf{q}}} &= \mathbf{0}_{3\times 1},\\
{}^{I}\dot{\hat{\mathbf{p}}}_C &= \mathbf{0}_{3\times 1},
\end{aligned} \tag{1}$$

where ${}^{I}_{G}\dot{\hat{\mathbf{q}}}(t)$ is the unit quaternion which described the rotation from the global from (G) to the IMU frame (I), $\hat{\boldsymbol{\omega}} \in \mathbb{R}^3$ and $\hat{\mathbf{a}} \in \mathbb{R}^3$ are the IMU measurements of angular velocity and acceleration respectively with the biases removed.

$$\Omega(\hat{\boldsymbol{\omega}}) = \begin{bmatrix} -\lfloor \hat{\boldsymbol{\omega}}_{\times} \rfloor & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^{T} & 0 \end{bmatrix}$$

where $\lfloor \hat{\boldsymbol{\omega}}_{\times} \rfloor$ is the skew symmetric matrix of $\hat{\boldsymbol{\omega}}$.

Based on Equation 1, we get the following linearized continuous dynamics for the error IMU state:

$$\dot{\hat{x}}_I = \mathbf{F}\hat{x}_I + \mathbf{G}\mathbf{n}_I^{\top} \tag{2}$$

where $\mathbf{n}_I^{\top} = \begin{pmatrix} \mathbf{n}_g^{\top} & \mathbf{n}_{wg}^{\top} & \mathbf{n}_a^{\top} & \mathbf{n}_{wa}^{\top} \end{pmatrix}$ is the noise vector and $\mathbf{F}$ and $\mathbf{G}$ are the state transition matrix and the noise matrix respectively.

$\mathbf{F}$ is given by:

$$\mathbf{F} = \begin{bmatrix} -\lfloor \hat{\boldsymbol{\omega}}_{\times} \rfloor & -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ -C({}^{I}_{G}\hat{\mathbf{q}})^{\top}\lfloor \hat{\mathbf{a}}_{\times} & \mathbf{0}_3 & \mathbf{0}_3 & -C({}^{I}_{G}\hat{\mathbf{q}})^{\top} & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \tag{3}$$

and $\mathbf{G}$ is given by:

$$\mathbf{G} = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & -C({}^{I}_{G}\hat{\mathbf{q}})^{\top} & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \tag{4}$$

D. Predict New State

When new images and IMU readings are received, we predict the new state using an Extended Kalman Filter, which utilizes a 4th order Runge-Kutta integration to update the IMU's state based on new accelerometer and gyroscope data. This function calculates the norm of the gyroscope measurements and sets up the Omega matrix to update the IMU's orientation quaternion. Depending on the gyroscope norm, it adjusts the quaternion calculation for numerical stability. The method computes intermediate values for velocity and position using the Runge-Kutta method, applying transformations based on the IMU's current state and corrected acceleration. The final updated state, including orientation, velocity, and position, is then recalculated and stored back into the state server, readying the system for subsequent updates.

E. State Augmentation

State Augmentation adds a new camera pose and updates the covariance matrix when a new image is received

In visual-inertial odometry systems, the state augmentation function is crucial for integrating the most recent camera state updates based on the latest IMU data. The function specifically adjusts the camera's position (GpC) and orientation (C_Gq) using the previous IMU state information. The pose of the camera is computed using the following equations [2]:

$$GpC = GpI + C(C_Gq)^T IpC$$

$$C_Gq = C_Iq \otimes I_Gq$$

Where GpC represents the global position of the camera, GpI is the global position of the IMU, $C(C_Gq)^T$ is the rotation matrix derived from the quaternion describing the camera's orientation relative to the global frame, and IpC is the relative position vector from the IMU to the camera. The quaternion operation $\otimes$ signifies quaternion multiplication, which is used to combine the orientation of the IMU (I_Gq) and the relative orientation from the IMU to the camera (C_Iq).

Following the calculation of the new camera pose, the state covariance matrix P is augmented to reflect the updated state uncertainty. This is achieved through a Jacobian matrix J, which maps the influence of the new camera state onto the overall system uncertainty:

$$J = \begin{bmatrix} C(I_{Cq}) & 0_{3\times9} & 0_{3\times3} & I_3 & 0_{3\times3} \\ \lfloor C(I_{Gq})^T I_{pC\times} \rfloor & 0_{3\times9} & I_3 & 0_{3\times3} & I_3 \end{bmatrix}$$

The updated state covariance matrix $P_{k|k}$ is computed as:

$$P_{k|k} = JP_{K|K}J^T$$

This matrix J effectively accounts for the effects of camera motion relative to the IMU, ensuring that updates in camera position and orientation are accurately reflected in the state covariance.

F. Adding Feature Observation

After a feature is detected, it is added to the feature map server, which is done by getting the current IMU state ID and number of features in the map server, and iterating over all of the new features, creating new features for unseen features and updating existing features. After, the tracking rate is updated.

G. Measurement Update

For the measurement update, we first decompose the Jacobian to reduce its computational complexity using QR decomposition. We then calculate the residual between the predicted and observed feature positions, and compute the Kalman gain and update the state and covariance. The residual is calculated by subtracting the predicted feature position from the observed feature position. The state is then updated by adding the Kalman gain multiplied by the residual, and the covariance is updated by subtracting the Kalman gain multiplied by the observation model from the identity matrix. Also, the IMU state, extrinsics, and camera states are updated by applying small angle quaternion operations.

H. Results and Discussion

The graphs and images are the results of running our VIO on the MH_01_easy EuRoC dataset are shown in Appendix A. The trajectory and relative errors throughout the run are shown.

Fig. 1: Setup of the dataset generation, camera shown in orange.

II. Phase 2

In the second phase of this project, we aim to implement a visual inertial odometry pipeline using deep learning techniques. To accomplish this phase, we broke the goal down into four steps: data generation, inertial model, visual model, and then generation and analysis of final pipeline. We will discuss each of these steps in detail in the following sections. To ensure that time was used efficiently, multiple steps were performed in parallel, meaning that certain steps which depended on the completion of others were started before the completion of the previous step. This allowed us to make the most of the time available, but meant that some steps contained work which did not make it to the final version of the project.

A. Data Generation

The first challenge of this phase was to generate a good dataset for training and testing our VIO model. The dataset we decided to create was modeled after the EuRoC dataset [3]. As such, three artifacts were created for each sequence: a ground truth file containing time-stamped positions and orientations, an IMU file which contained timestamped 6-DoF IMU data, and finally a directory containing timestamped images in the sequence.

Setup: The basic setup of the data generation was as follows: A camera was positioned above an image which was placed on the ground plane of a 3d scene, as shown in Figure 1. The camera was then moved around and oriented in the scene based on the ground truth position and orientation information, and frames were rendered. We chose an IMU rate of 100Hz, and a camera rate of 10Hz, so each frame was associated with 10 IMU samples.

(a) Intersection 1

(b) Intersection 2

Fig. 2: Images used for the dataset

Choice of Images: Our team wanted to use images which would result in realistic and challenging visual odometry problems. As such, we decided to use aerial photographs of traffic intersections. These images were chosen because they contained a lot of texture and detail, and because they could be found in high resolution. The two images used in the dataset are shown in Figure 2.

Paths: Initially when attempting to create paths for our camera to follow, our team looked at the Blackbird dataset [4], but unfortunately the dataset was no longer available to download. As such, we generated our own paths inspired by the Blackbird dataset. To do this, we came up with parametric equations for the position of the camera in the scene, and then used the equation to generate a path. The equations we used were as follows:

1) A out and back line of length 15m along the x-axis at a fixed height h above the ground plane.

$$x(t) = t$$
$$y(t) = 0$$
$$z(t) = h$$

2) A circle of radius $5m$ at fixed height h above the ground

plane.

$$x(t) = 5\cos(t)$$
$$y(t) = 5\sin(t)$$
$$z(t) = h$$

3) A circle of radius $5m$ which moves up and down twice during the path.

$$x(t) = 5\cos(t)$$
$$y(t) = 5\sin(t)$$
$$z(t) = 10 + 2\sin(2t)$$

4) A stretched Lissajous curve with $a = 1$, $b = 3$ at fixed height of $10m$ above the ground plane.

$$x(t) = 2\cos(3t)$$
$$y(t) = 4\sin(t)$$
$$z(t) = 10$$

5) A complex 3 dimensional curve

$$x(t) = 3\cos(3t) + 2\cos(2.3t) + 0.9\cos(6t)$$
$$y(t) = 3\sin(3t) + 2\sin(2.3t) + 1.34\cos(6t)$$
$$z(t) = 11 - 3\sin(t/2)$$

The paths were sampled non linear with a slow start and end, as to not have the camera move too quickly at the start and end of the path. This was done my passing the time variable t through the following function:

$$f(t) = \frac{32\sin(t)^3}{279} - \frac{64\sin(t)}{93} - \frac{14\sin(2t)}{93} - \frac{\sin(4t)}{372} + t$$

Each of these paths can be seen visualized in Appendix B.

Rendering: The step of rendering the view was accomplished using blender. Notably, to improve render times, the image texture was placed onto a plane as a purely emissive texture, which allowed the rendering to occur at around 20 frames per second. The camera was then moved along the path, and frames were rendered at each time step. This process was automated using a python script which interfaced with blender through the bpy library.

IMU Data: The IMU data was generated by simulating a 6-DoF IMU in the scene. The IMU was located exactly coincident with the camera, with the camera facing in the $-z$ direction with $+y$ up. After generating raw IMU readings using basic discrete differentiation of the position and orientation data, the readings were passed through a model of the IMU noise. The noise model was a simplified version of the GNSS-INS-SIM codebase, as given in the code for the OysterSim project [5]. The noise model was a simple random walk model with a bias term, and was used to generate the final IMU data.

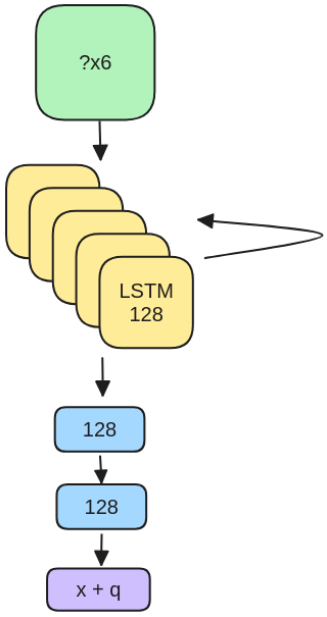

Fig. 3: Inertial Odometry Model

B. IO

The next step in the project was to implement a inertial odometry pipeline. This pipeline would take in IMU data and output a trajectory estimate. The pipeline was implemented using a LSTM network, with an input of 6-DoF IMU data, hidden state of size 128, and an output of a 7 vector representing the position and orientation of the camera. The network was trained on the generated dataset.

The key insight for this portion of the model is that the task of predicting a path from IMU data is inherently temporal, as at the most basic level one needs to integrate the acceleration and angular velocity data to get the position and orientation. As such, a LSTM network was chosen as the model for this task. The network was implemented using the PyTorch library, and was trained using the Adam optimizer with a learning rate of 0.001. Unfortunately, this network was unable to learn anything, and thus the results are not included in this report. The loss used for the IMU was the MSE loss between the ground truth change in position, and the predicted change in position. This was added to the cosine similarity loss between the ground truth and predicted orientation. The loss was then summed and used to train the model.

C. VO

The final step in the project was to implement a visual odometry pipeline. This pipeline would take in a pair of images and output a relative pose estimate. The pipeline was implemented using a CNN network, with an input of two images and an output of a relative pose. The network was trained on the generated dataset.

Figure 4 shows the architecture of the visual odometry model. The model was implemented using the PyTorch library, and was trained using the Adam optimizer with a learning rate

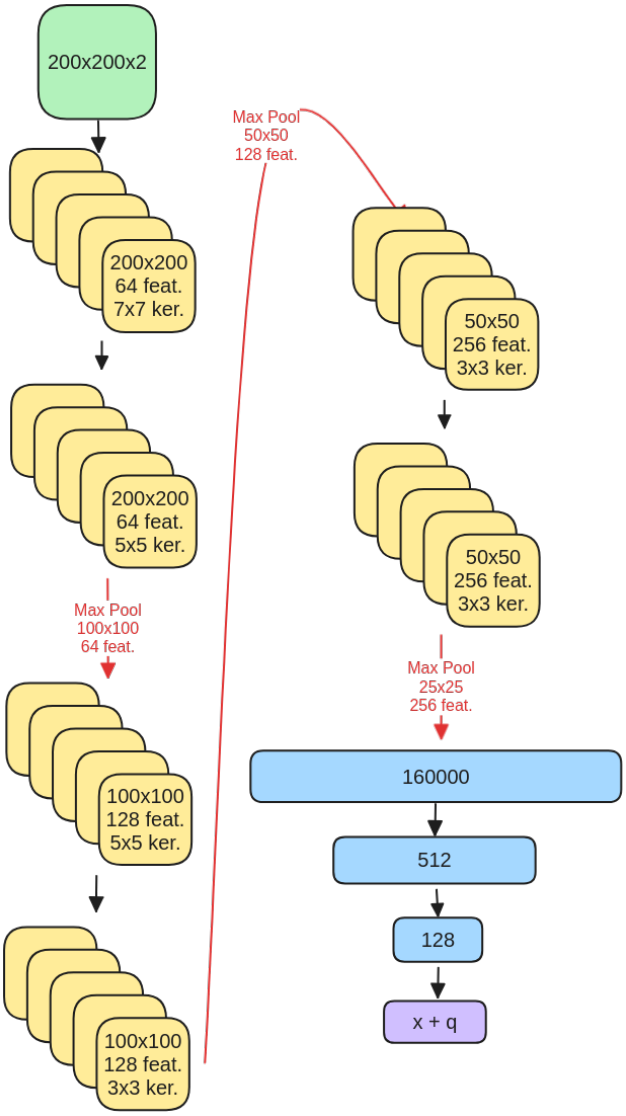

Fig. 4: Visual Odometry Model

of 0.001. The model was trained for 20 epochs, and the results are shown in the next section. The same loss function was used for the visual odometry model as was used for the inertial odometry model. The loss was the MSE loss between the ground truth change in position, and the predicted change in position. This was added to the cosine similarity loss between the ground truth and predicted orientation. The loss was then summed and used to train the model.

D. Results

The previously described techniques did not perform all that well, and overfitted to the data, or didn't learn any generalized knowledge. The loss over time for the VO model is shown in Figure 7. The loss for the IO model was similar, and is not shown here. The results of the VO model were not very good, and the model was unable to generalize to new data.

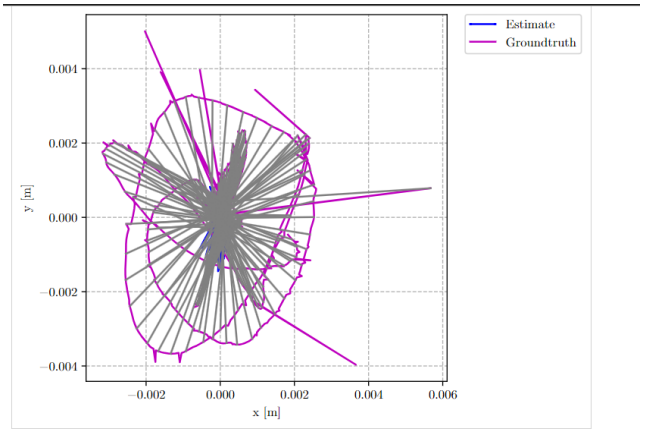

Fig. 5: Degenerate results of the VO model, just predicting zeros

The results of the IO model were even worse, and the model was unable to learn anything at all.

E. Future Research

The application of deep learning to the task of VIO is a well explored field, with techniques such as LSTM networks, temporal CNNs, and many others having been applied to the task. One architecture which we did not see well explored were transformers. Transformers are great at taking in sequences of data and outputting sequences of data, and as such could be a good fit for the task of VIO. Additionally, the use of a transformer would allow for the model to take in the entire sequence of data at once, rather than having to be fed the data in a sequential manner. This could allow for the model to learn more complex temporal relationships in the data, and could potentially lead to better results. Finally, transformers have been well explored for images, and so the use of a transformer for the visual portion of the model could be a good fit. For the inertial portion of the model, a transformer could be used to take in the entire sequence of IMU data and output the trajectory. This would allow for the model to learn more complex temporal relationships in the data, and could potentially lead to better results.

References

[1] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572. DOI: 10.1109/ROBOT.2007.364024.

[2] K. Sun, K. Mohta, B. Pfrommer, *et al.*, "Robust stereo visual inertial odometry for fast autonomous flight," *CoRR*, vol. abs/1712.00036, 2017. arXiv: 1712.00036. [Online]. Available: http://arxiv.org/abs/1712.00036.

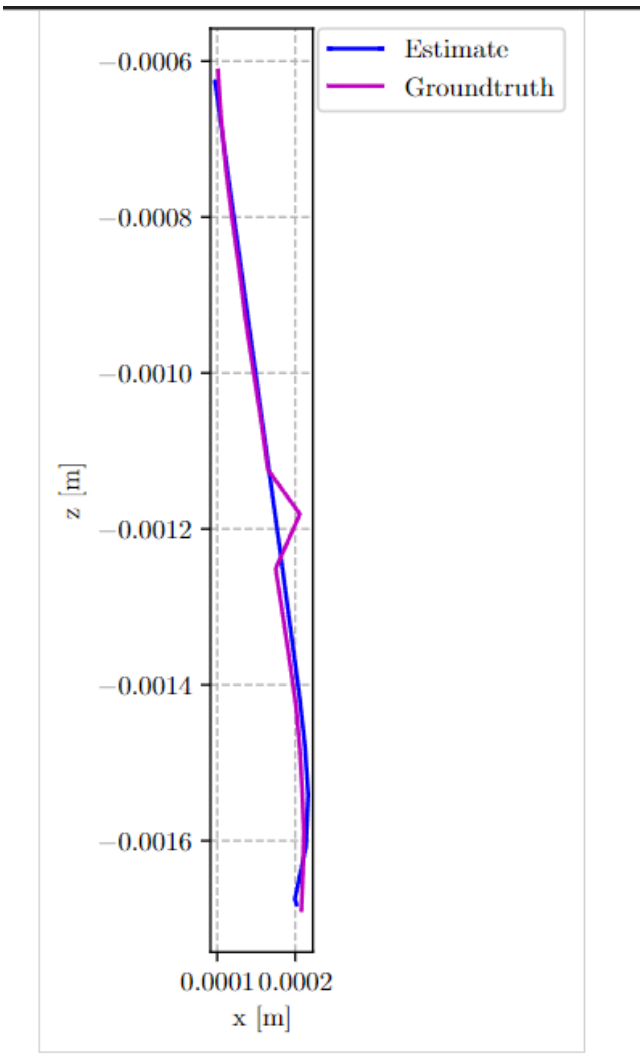

Fig. 6: Overfitted results of the VO model, just memorizing the data

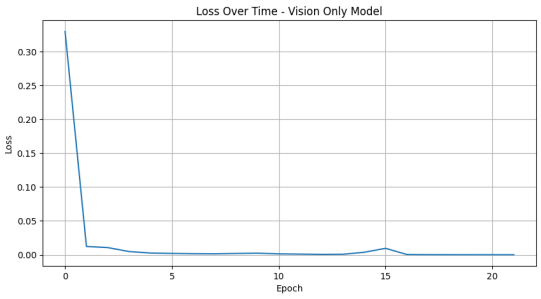

Fig. 7: Loss over epochs for the VO model

[3] M. Burri, J. Nikolic, P. Gohl, *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[4] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, *The blackbird dataset: A large-scale dataset for uav perception in aggressive flight*, 2018. arXiv: 1810.01987 [cs.CV].

[5] X. Lin, N. Jha, M. Joshi, N. Karapetyan, Y. Aloimonos, and M. Yu, "Oystersim: Underwater simulation for enhancing oyster reef monitoring," in *OCEANS 2022, Hampton Roads*, IEEE, Oct. 2022. DOI: 10.1109/oceans47191.2022.9977233. [Online]. Available: http://dx.doi.org/10.1109/OCEANS47191.2022.9977233.

APPENDIX A
CLASSICAL VIO RESULTS

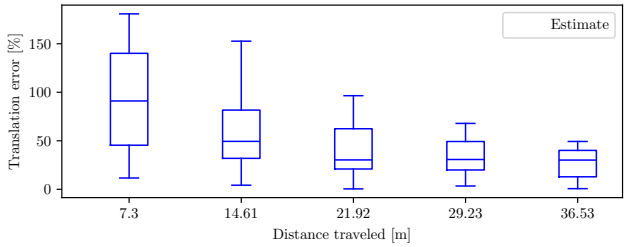

Fig. 8: Relative Translation Error Percentage

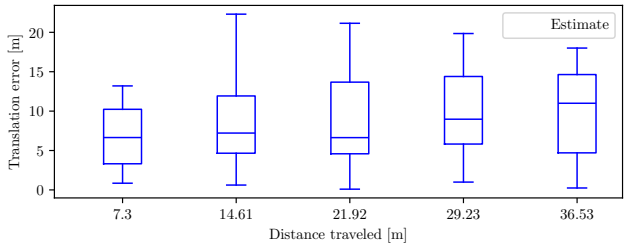

Fig. 9: Relative Translation Error

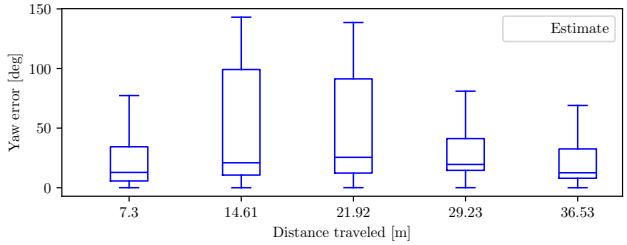

Fig. 10: Relative Yaw Error

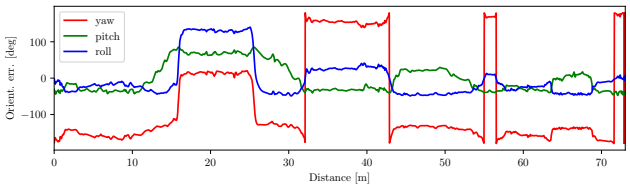

Fig. 11: Rotation Error

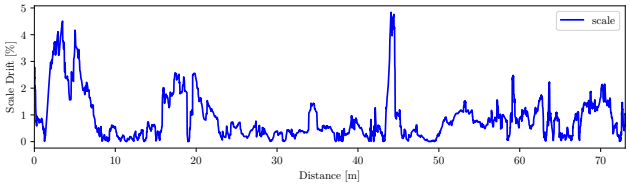

Fig. 12: Scale Error

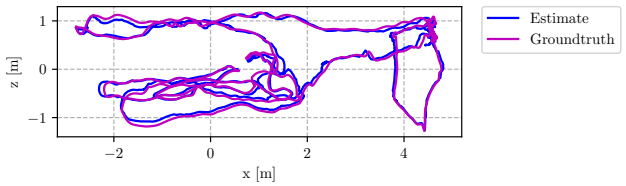

Fig. 13: Trajectory Side

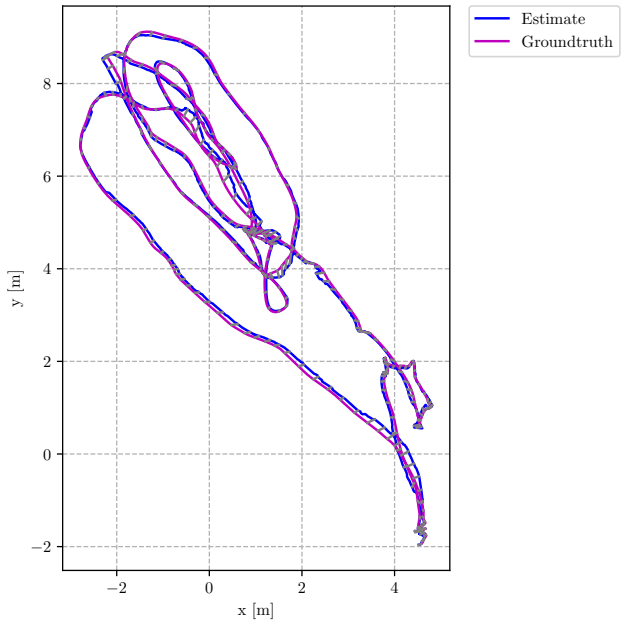

Fig. 14: Trajectory Top

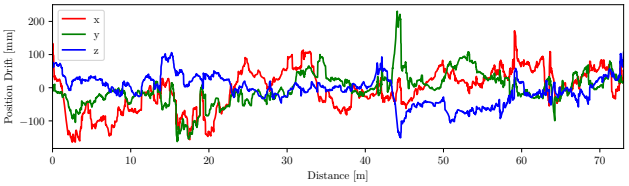

Fig. 15: Translation Trror

APPENDIX B
DATASET PATHS

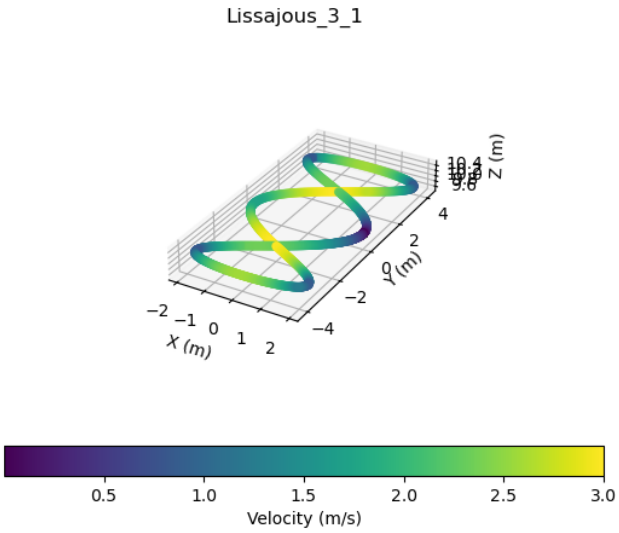

Fig. 16: Lissajous 3:1

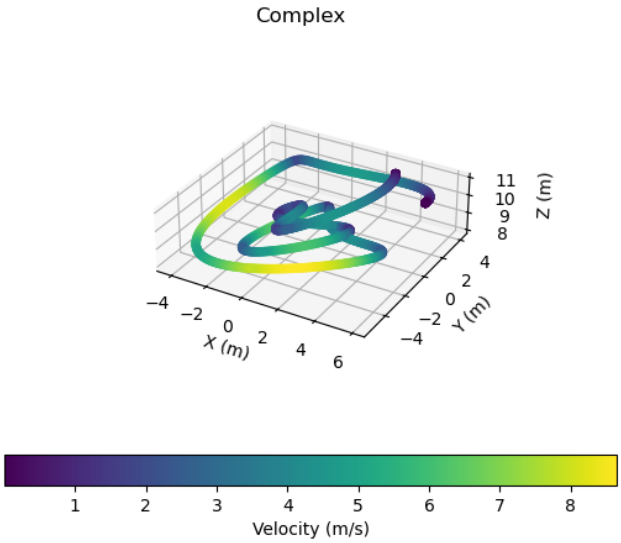

Fig. 17: Complex Path

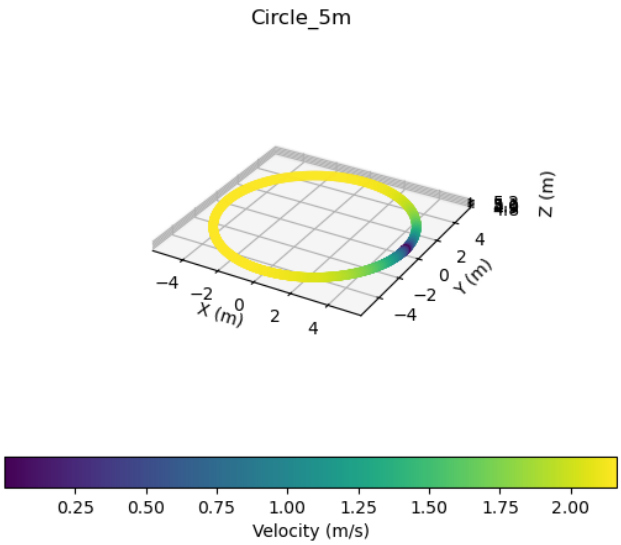

Fig. 18: Circle 5m

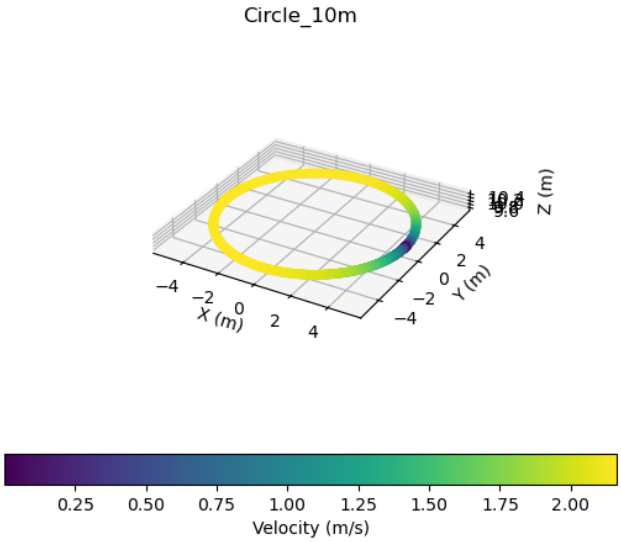

Fig. 19: Circle 10m

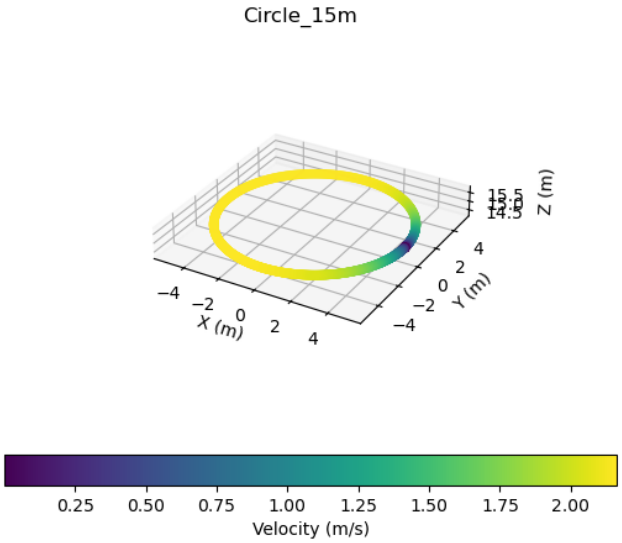

Fig. 20: Circle 15m

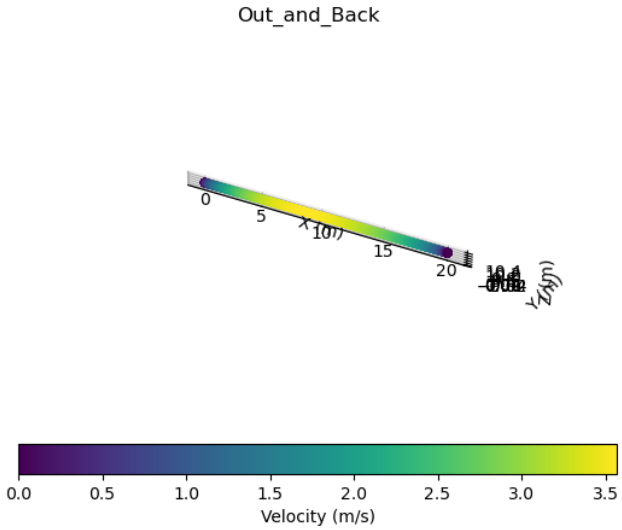

Fig. 22: Out and Back

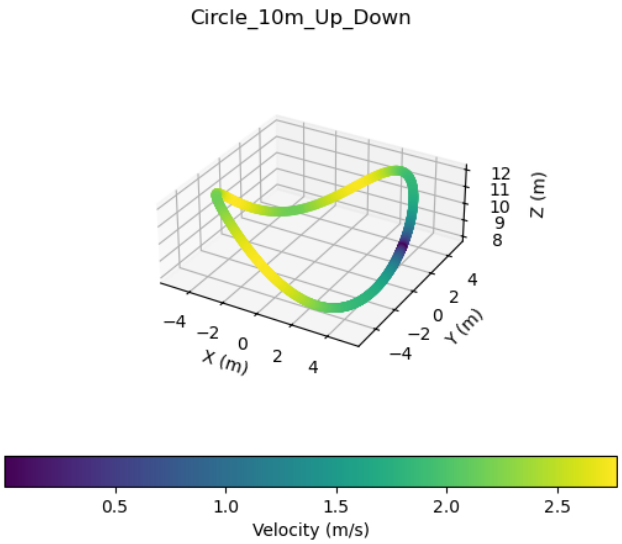

Fig. 21: Circle 10m Up Down