RBE 549 Project 4 Phase 2
Deep Visual-Inertial Odometry

Niranjan Kumar Ilampooranan
MS Robotics Graduate Student
Worcester Polytechnic Institute

Thanikai Adhithiyan Shanmugam
MS Robotics Graduate Student
Worcester Polytechnic Institute

***Abstract*—In this report, network architectures and loss functions are proposed that is used to estimate odometry/ relative pose between scenes using IMU (Inertial measurement unit) readings and camera frames. The process used to generate synthetic data for training and testing of these models is discussed along with recorded results.**

I. Introduction

As seen in the classical approach used in phase 1 to estimate odometry using fusion of inertial and visual data, the error between the ground truth and the estimates were satisfactory. Yet, there is heavy dependency on the feature tracking and detection, which raises the question of replacing it with deep feature matching methods. Although straightforward for visual data, the same cannot be said for the data collected by IMU.

Given the scarcity in previous work on using deep learning methods to estimate odometry using visual-inertial data fusion, this work aims to propose three network architectures that could be trained on such data to estimate the relative states of the camera. The first network (VO) predicts the relative poses using only frames, while the second network (IO)is fed IMU data to predict the relative pose. Comparison between the performance of two different architectures for VO is also shown in this work. Finally, the third network uses both the visual and inertial data for the same task. The dataset used for this case along with the network architectures, loss functions, and results are discussed in further sections.

II. Dataset

Given that the problem involves estimating the odometry from combination of images and 6-DoF IMU data, using preexisting datasets would be another challenge by itself. To alleviate this issue, we have opted for synthetic data generation by scene rendering in Blender. To train, validate, and test the networks, sample trajectories were generated using simple primitives. These include simple circular trajectory, simple helical ascent, and the former combined with sinusoidal variation in ascent for additional complexity. The plots for visualization are shown in the figure below.

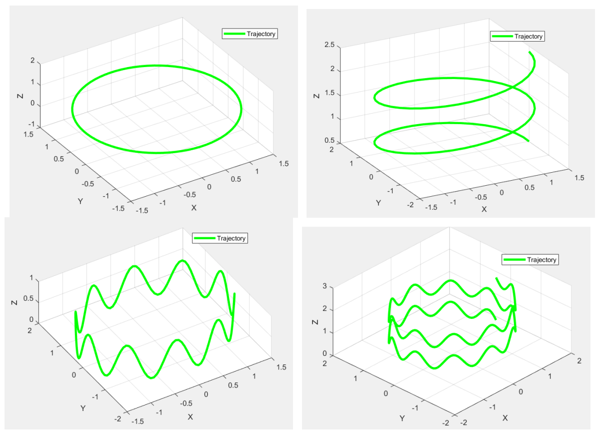

Fig. 1. Camera movement - Simple Trajectories

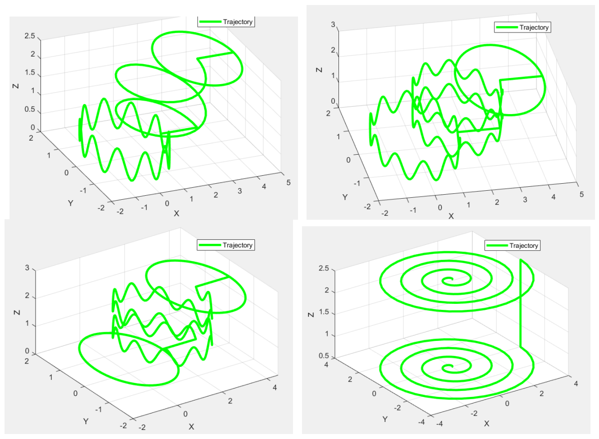

Fig. 2. Camera movement - Combined Trajectories

Using the Blender models and quadrotor dynamics given by Prof. Sanket's Autonomous Aerial Robotics course, the image dataset is generated [1]. Quite simply, the trajectory data is fed to the quadrotor (or a flying object with camera facing downwards) to generate an adequately realistic motion profile and state values at each timestamp along with scene capture. As far as the visual features are concerned, a large plane with image texture rich in features is placed on the ground, essential for good odometry estimation using visual data. The image texture used in our case is shown below.

Fig. 3. Image Texture used in Blender

In the case of IMU data, multiple ways exist to generate, given the trajectory data. One such method is finding the first and second order derivatives of the trajectory using the data, after which noise is added (as done in [2]). Alternative method include using MATLAB's IMU model, which provides accelerometer and gyroscope readings, given the acceleration, angular velocities and orientation data. Out of these two, the former was used to generate the necessary IMU data. Furthermore, only every 10^{th} frame is considered for IMU datastream. Hence for the model training, 20000 IMU readings and 2000 images were used. A sample scene of camera moving along the proposed trajectory in Blender while gathering the image data is shown below.

Finally, dataset for 8 trajectories were generated, of which 7 are used for training and 1 is used for testing (helical ascent). The observations and results for these shall be discussed in further sections.

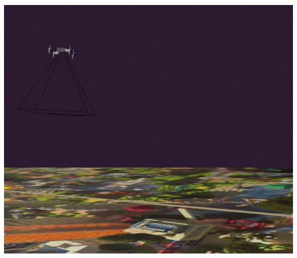

Fig. 4. Blender - Scene capture of Image Texture

III. Network Architecture

A. VO Architecture

We conducted a literature review on previous year papers to learn a optimal network for Visual Odometry and found LSTM+CNN and Deep VO (fully convolutional) interesting. We trained initally a circle trajectory on the both the networks and found out both provided almost similar losses.

The architecture we finalized for Visual Odometry is a network of Convolutional Layers with Fully Connected Layer each for classifying the pose and the orientation quaternion. The input given in this network is a stack of 2 frames of timestamp t and t+1. The model is a supervised network where the ground truth is the pose and orientation data of the drone obtained from the blender environment. The model has 9 layers of 2D convolutional layers and 3 Fully Connected Layers. Max Pooling has been used in alternate layers to reduce computational time and the activation function ReLu has been used in all the layers except the final regression layer. [?] has suggested using Leaky ReLu provided better results but the dataset generated provided better results on the relu activation function. The LSTM+CNN model had 128 hidden layer with 2 LSTM layers and 3 Convolutional Layers which was passed after the LSTM followed by 2 Fully Connected Layers.

The training on both LSTM+CNN and DeepVO overfitted the data and peformed exceptionally on the training dataset but failed to provide good results on the test data. However, we decide to move on with DeepVO as it was computationally faster than LSTM+CNN and when training over infinte epochs to get the optimal number of epoch, DeepVO got a stable loss after 25 epochs while LSTM+CNN went over 100 epochs and was still fluctuating. The architecture of both networks used can be found below

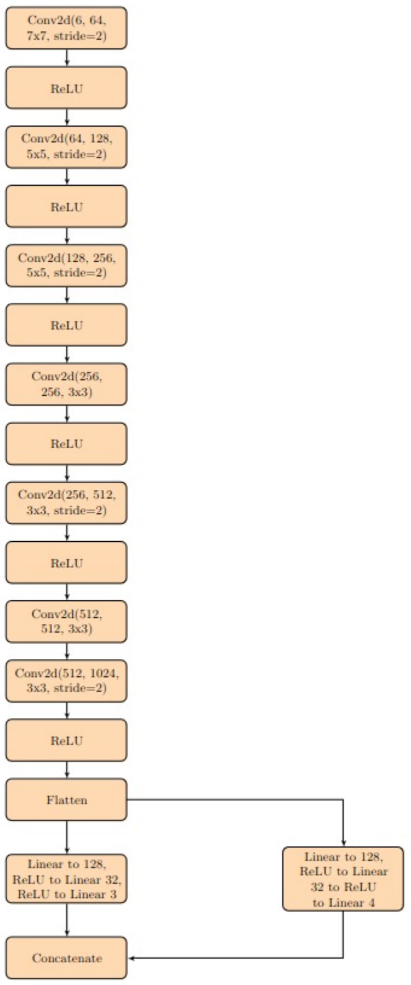
Conv2d(6, 64, 7x7, stride=2)
ReLU
Conv2d(64, 128, 5x5, stride=2)
ReLU
Conv2d(128, 256, 5x5, stride=2)
ReLU
Conv2d(256, 256, 3x3)
ReLU
Conv2d(256, 512, 3x3, stride=2)
ReLU
Conv2d(512, 512, 3x3)
Conv2d(512, 1024, 3x3, stride=2)
ReLU
Flatten
Linear to 128, ReLU to Linear 32, ReLU to Linear 3
Linear to 128, ReLU to Linear 32 to ReLU to Linear 4
Concatenate

Fig. 5. VO CNN Architecture

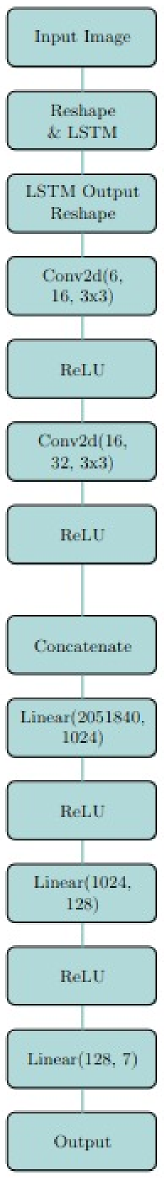

Fig. 6. VO LSTM Architecture

B. IO Architecture

The IO Architecture predicts the relative pose between frames in sequence. We thought to use a simple RNN architecture which takes in IMU data of 50 frames in sequence. However, this architecure was not learning anything at all. We tried 2 loss functions discussed in the loss subsection and still the model was not training at all. We got a stable error of 2.5 relative pose. Therefore, we implemented the LSTM architecture with hidden layers 512 and 2 LSTM layers. We then regressed 3 Fully connected Layers seperately for pose and orientation.

The LSTM also overfitted the intital circle trajectory we trained. The training merged with other trajectory was in vain as the model updates its parameters with the new trajectory and forgets the previous trajectory weights. We tried to rectify this error but could not implement a effective solution owing to time constraints.

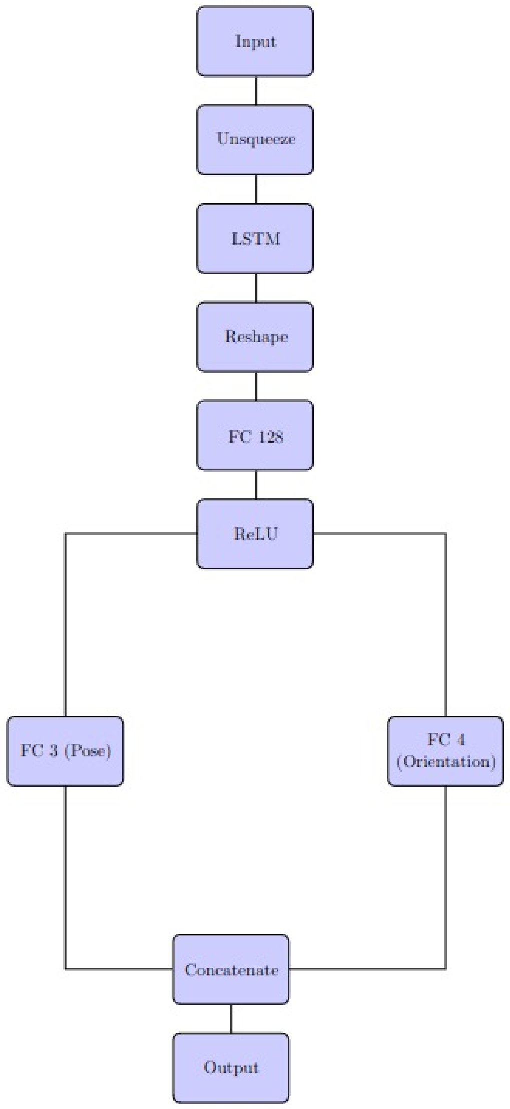

Fig. 7. IO Architecture

C. VIO Architecture

The Visual Inertial Odermetry architecture was combined from the VO and IO architecture. The VO convolutional layers were merged with IO LSTM layers and passed to seperate fully connected layers. The architecture implemented can be found below.

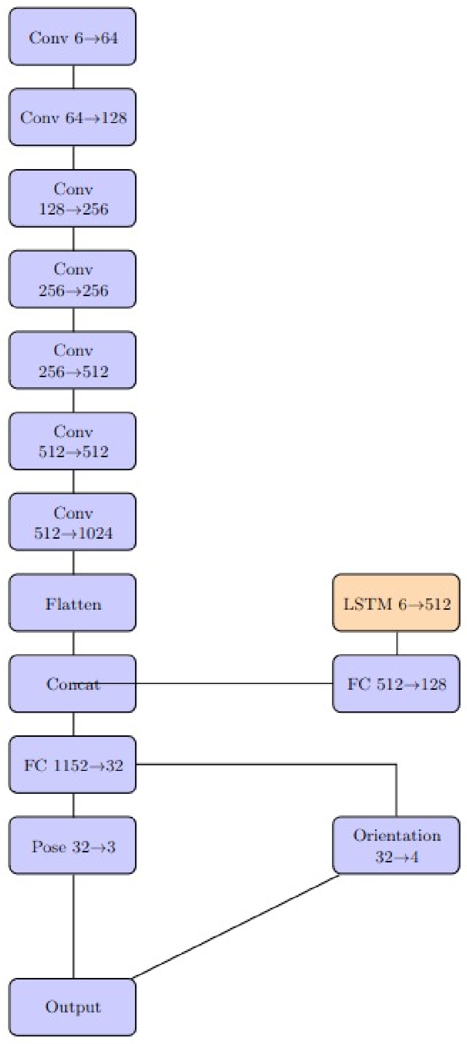

Fig. 8. VIO Architecture

D. Network Parameters

The 3 networks uses the Adam Optimizer with learning rate of 0.001 and $\beta 1 = 0.9$ and $\beta 2 = 0.999$. The batch size used is 16 and standardization is done for the images and also the IMU DOF for more effective training. We tried running on 100 epochs for all 3 models but stopped at 25, 38, 30 epochs for VO, IO, VIO models as there was not much learning after these epochs.

E. Loss Functions

We did an intensive study on different loss functions to select the optimal one. For all 3 models, the loss was calculated between the relative pose i.e difference between the current frame and the next frame. At first, we used MSE Loss on the whole output and to our surprise we over-fitted the training data. We tried to implement MSE Loss individual on pose and orientation. The results were better for VO architecture but was not helping in learning the IO architecture. We then referred [3] which implemented geodesic loss for the quaternion output. However, our data was not learning at all in both in LSTM and CNN network. Therefore, instead of the geodesic loss, we implemented the loss function defined in the official DeepVO implemenatation paper [4] given below for all 3 models but the data was overfitting but less compared to MSELoss.

$$pose\_loss = MSE(output, label) \tag{1}$$

$$quat\_loss = MSE(1 - MSE(output * label)) \tag{2}$$

$$total\_loss = pose\_loss + \beta * quat\_loss \tag{3}$$

Here, β is the loss factor which we defined to be 150 as mentioned in the above cited paper.

IV. Results

In this segment, we create a graphical representation of the discrepancy between the Actual Path and the Predicted Trajectory using the RPG trajectory evaluation toolbox.

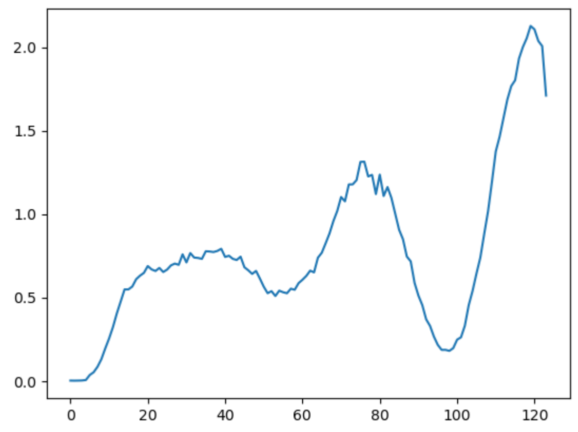

Fig. 9. VO Test Loss

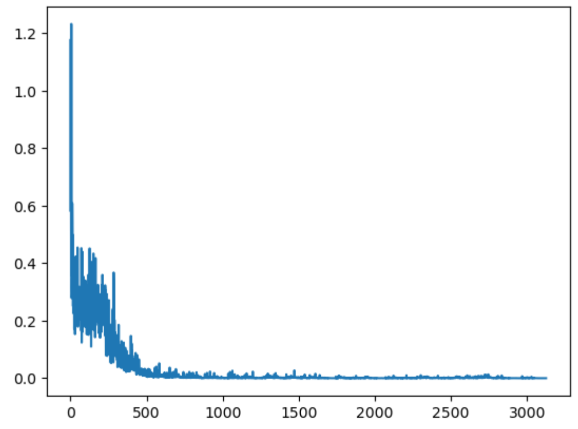

Fig. 10. VO Train Loss

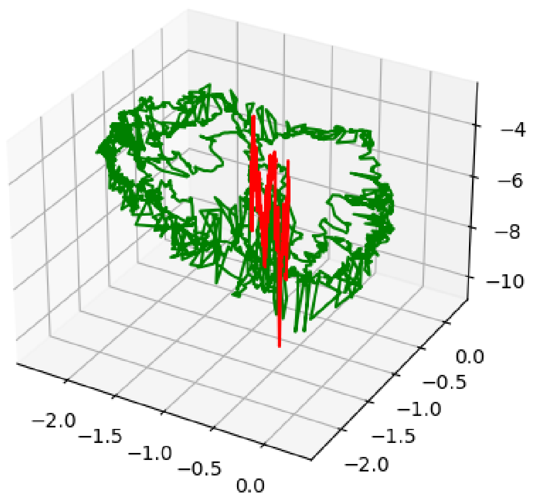

Fig. 11. VO Predicted Test - Spiral Trajectory

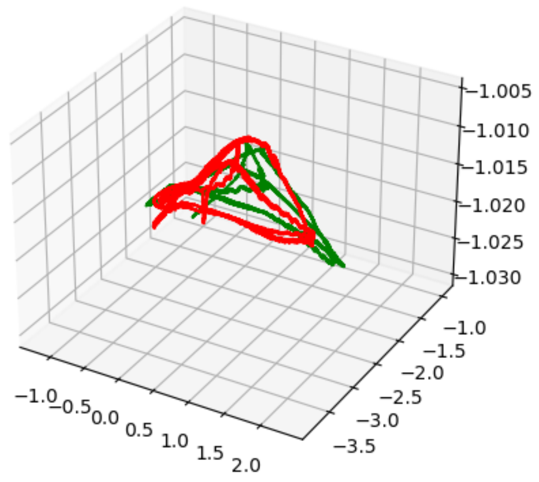

Fig. 14. IO Predicted Train - Spiral Trajectory

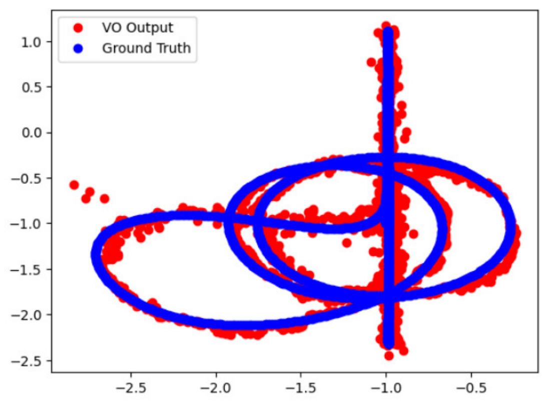

Fig. 12. VO Predicted Train - Spiral Trajectory

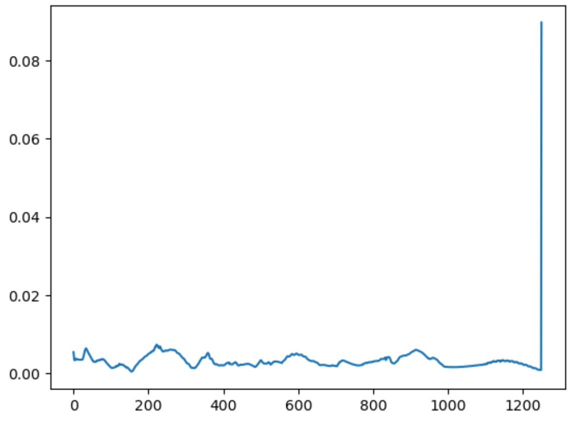

Fig. 15. IO Test Loss

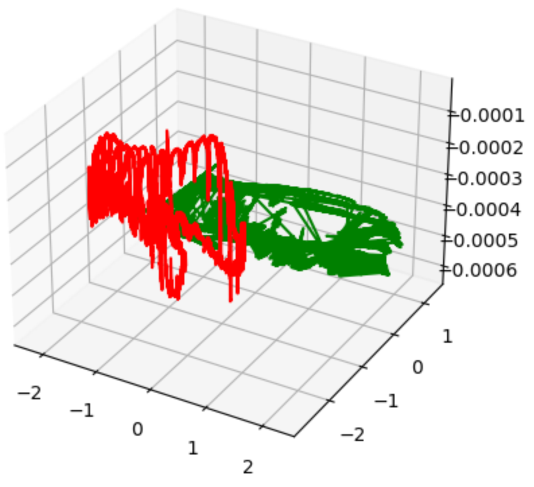

Fig. 13. IO Predicted Test - Spiral Trajectory

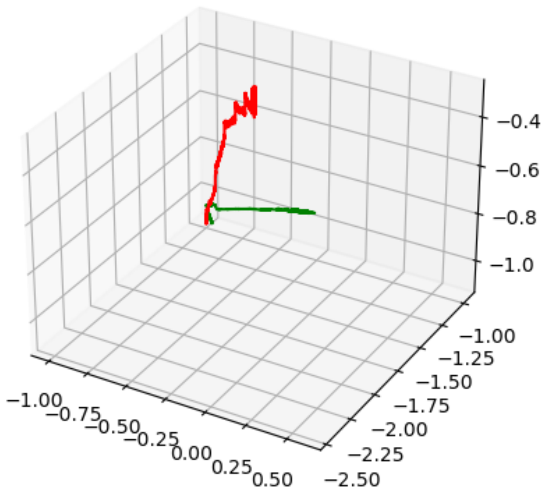

Fig. 16. VIO Predicted Test - Spiral Trajectory

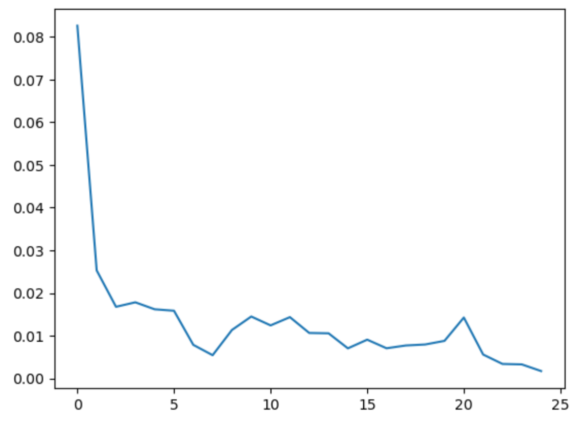

Fig. 17. VIO Train Loss

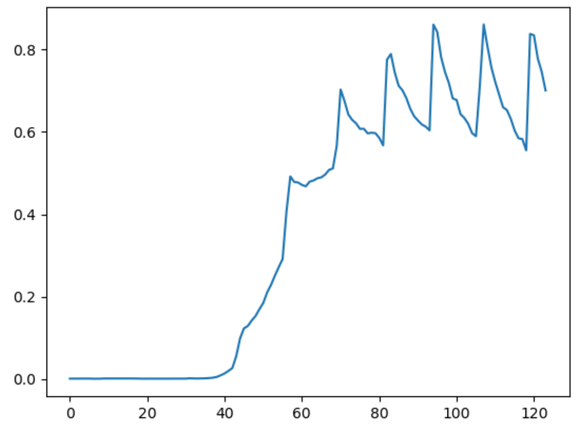

Fig. 18. VIO Test Loss

V. Future Directions

Some new research directions would include the integration of complementary sensors, such as event cameras, and deep learning.

Egomotion is one domain where VIO can be explored further. Egomotion is the three-dimensional movement of a camera within its surroundings. In computer vision, egomotion specifically involves determining a camera's motion relative to a fixed scene. For instance, in autonomous navigation scenarios, egomotion estimation entails discerning a vehicle's position as it moves in relation to features like road markings or observed street signs. This estimation is crucial for enabling autonomous robots to navigate effectively.

One aspect where current work could be improved would be in the following. Since the output is composed of sequence of asynchronous events, traditional frame-based computer-vision algorithms are not directly applicable. Hence, novel algorithms must be developed to deal with these cameras.

One research aspect in VIO can be used for human motion captures. Inertial Data is recording the human pose in the form of MOCAP. There are research which uses stereo camera along with MoCAP to actually improve pose estimation for healthcare as healthcare requires precision.

References

[1] "Rbe595-f02-st – hands-on autonomous aerial robotics."

[2] "prgumd/oystersim," 10 2023.

[3] S. S. M. Salehi, S. Khan, D. Erdogmus, and A. Gholipour, "Real-time deep registration with geodesic loss," *CoRR*, vol. abs/1803.05982, 2018.

[4] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2043–2050, 2017.