Flinders UNIVERSITY
ADELAIDE · SOUTH AUSTRALIA

Vision-Based Real-Time Velocity Estimation

By

Gayeon Lee

Thesis
Submitted to Flinders University
for the degree of Bachelor of Engineering (Robotics)(Honours) / Master of Engineering (Electronics)

Bachelor of Engineering (Robotics)(Honours) / Master of Engineering (Electronics)

College of Science and Engineering
<30/10/2023>

TABLE OF CONTENTS

ABSTRACT

For autonomous system, it is often useful to know the position and velocity of nearby objects for tasks such as obstacle avoidance or object tracking. The hidden states of an object, such as velocity, can be estimated by knowing the mathematical model that describes the relationship between the measurable state and the hidden state. Hence, both position and velocity of the object can be found by object position measurement. For autonomous Micro-Aeria-Vehicle (MAV), various research investigated in using camera for real-time object state estimation as it is a low-cost and low-complexity alternative of ranging sensors. However, little literature is available when the scope is narrowed down to a moving object and moving MAV, despite the position of ground object can be obtained with the camera on MAV. This project aims to develop real-time object velocity estimation system, only using strapdown camera for the exteroceptive sensor, to be integrated onto a MAV (or other aerial system). The project assumes the object is a ground vehicle, on a flat surface with known altitude, moving at a constant acceleration. The system must be robust to measurement noise for practical application and must be applicable for object at a long-range (i.e. 100 m+). The ground vehicle (for the object) and MAV was simulated at maximum of 30 and 50 m/s of speed. Considering the reviewed literatures, this project initially proposed to use the pin-hole camera model, with Extended Kalman Filter (EKF) as a baseline, and to fuse with optical flow model for inertial state-estimation by Unscented Kalman Filter (UKF). The baseline EKF was developed and tested on a sample dataset, then in depth analysis of EKF was done to assess its suitability for velocity estimation using a strapdown monocular camera. The dataset for system analysis was extracted which consisted of scenarios with stationary vehicle, vehicle in linear and angular motion in a combination of different MAV motion. The baseline system was able to achieve all project aim in the tested scenarios except achieving robust performance under measurement noise; it was found the system has significantly high sensitivity to noise in MAV orientation. It was assumed that the system sensitivity is caused from the measurement model. The rotation matrices, that contains MAV orientation and camera parameters, in the measurement model is highly non-linear. EKF uses linear approximation of model which is not as effective to highly non-linear model where UKF performs better in such cases. As the future work, including system application to UKF, a list of tasks was identified to improve the system and further analyse the system performance.

DECLARATION

I certify that this thesis:

1. does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university
2. and the research within will not be submitted for any other future degree or diploma without the permission of Flinders University; and
3. to the best of my knowledge and belief, does not contain any material previously published or written by another person except where due reference is made in the text.

Signature of student.....Gayeon Lee.....

Print name of student.....Gayeon Lee.....

Date.....30/10/2023.....

I certify that I have read this thesis. In my opinion it is/is not (please circle) fully adequate, in scope and in quality, as a thesis for the degree of Bachelor of Engineering (Robotics)(Honours) / Master of Engineering (Electronic). Furthermore, I confirm that I have provided feedback on this thesis and the student has implemented it minimally/partially/fully (please circle).

Signature of Principal Supervisor.....

Print name of Principal Supervisor.....

Date.....

ACKNOWLEDGEMENTS

I acknowledge that this project was conducted under a large support from multiple supervisors. I would like to thank my supervisor Professor Karl Sammut for providing huge support and resources during the project especially regarding the university management and assessment related agendas. I also would like to thank supervisor Dr Jia Kok for constantly providing project resources and technical assistance/guidance throughout the project. I thank supervisor Dr Bradley Fraser for providing support to meet the project agendas and timeline along with necessary management and arrangement for the project and related assessment. Last but not least, I would like to thank Dr Anna Dostovalova for allowing me to share her office space and providing opportunity to attend the workshop that covered the contents related to the project.

Besides the supervisors, I would like to thank Dr Yiding Hu for sharing his knowledge on the Bayesian filters, its application, and the modelling process. Furthermore, I would also like to thank Dr Jijoong Kim who provided related resources and implementation examples which also supported me in understanding the concepts and its implementation. The project was improved significantly from the information and materials provided through these supports.

LIST OF FIGURES

1. INTRODUCTION

1.1. Background

Autonomous systems often need a sensor to detect distance to any nearby objects or obstacles. The information is used for tasks such as obstacle avoidance, state estimation, etc (Siegwart, et al., 2011). The state estimation can be done for the estimation of inertial state of the autonomous system (Grau & Pansiot, 2012; He, et al., 2017) or the state of a nearby object (Strydom, et al., 2015) with appropriate measurement and system model. For object state estimation, the global position of the object can be derived from the range measurement using appropriate model (Strydom, et al., 2015). Furthermore, the velocity of an object can be also estimated from the position estimate; although, it cannot be taken from a single instance as it must be estimated from the position changes over a certain time due to the concept of velocity.

A good object velocity estimation using a monocular camera and on-board INS can often be estimated by differentiating target position. The object position estimation on an autonomous system is often derived from two types of measurements which are the states of the autonomous system (i.e. position and attitude) and the range/bearing measurement to the object. The range and bearing measurement can be taken from either using a ranging sensors such as LiDAR (Javanmardi, et al., 2019; Kubelka, et al., 2015) or a camera with aid of computer vision under a list of assumptions (Pagello, et al., 2004; Gunawan, et al., 2019). A camera is often integrated onto an autonomous system due to the advantage that the visual field contains more information than a range and bearing to the object, including object type, pose or the scene context (Szeliski, 2011). However, it may require a significant processing time for real-time application depending on the algorithm. In contrast, ranging sensor often provides more accurate measurement at a faster update rate where the quality may still vary by the cost. It is advantageous for an autonomous system to obtain any measurements at a fast update rate to react to the nearby objects in real-time; a ranging sensor may be more appropriate for such applications, although, it is often costly compared to a camera where the cost increases with the sensor quality.

1.2. Project Motivation

Multiple research projects have successfully investigated using camera to perform real-time state estimation on an autonomous platform (Gunawan, et al., 2019; Pan, et al., 2023). Among the goals of research are to achieve such capability at a lower cost and to reduce system complexity by involving no ranging sensor; the camera also can be used for other capabilities concurrently. When the context is limited to a camera attached on a Miniature Aerial Vehicle (MAV) platform, the current research mainly focuses on the inertial state estimation rather than an object state estimation.

It is usually in context of a limited camera and object motion when the context is narrowed down to the object state estimation. These research with the limit in motion, such as a stationary camera and ground vehicle moving at a known linear path, investigates problems specifically for traffic control (Gunawan, et al., 2019; Ning, et al., 2022). Some deploy a mathematical model of the camera projection to convert a certain image coordinate to the global coordinate system using the MAV states to measure the global coordinate of a ground object (Barber, et al., 2006; Beard & McLain, 2012; Cai, et al., 2022). Furthermore, any states that are not directly measurable, such as velocity of an object, can be estimated by developing a mathematical state model and filtering the measurement according to the model (Rigatos, 2010; Ullah, et al., 2020; Wang, et al., 2015). Despite the points, literatures on object velocity estimation with less limitation is not readily available.

The currently available methodologies are capable to perform real-time velocity estimation of a ground object from camera on a MAV, with fewer assumptions and limitations on camera and object motion. This further allows for a decrease in cost and system complexity to build a MAV (or capability extension for existing MAV with camera with no ranging sensors) for the real-time object velocity estimation and availability for further capability extension through development of computer vision algorithms.

1.3. Project Outline

1.3.1. Terms and Clarification

This report uses the term ‘object’ and ‘vehicle’ interchangeably due to the project using a ground vehicle to simulate a moving object. The term ‘filter’ and ‘system’ was also used interchangeably. The model that converts image coordinates to world coordinates is referred to as Geolocation Model or Position Measurement Model. Any ranging sensors mentioned in this report indicates those ranging sensors used or required to take ranging measurement to the object. Note that an MAV may still be integrated with a ranging sensor dedicated for altitude or other inertial state measurement.

1.3.2. Aim

This project aims to develop an object velocity estimation system applicable to a MAV camera to detect a single ground object. The system must be able to estimate the object velocity under the following conditions:

- Real-time detection
- Noises in all measurements
- Non-stationary MAV and object
- No prior knowledge of the object trajectory
- No limitation from the distance to the object

The conditions are set to ensure the system to be available for wider range of application and not limited to specific scenarios. The system will be capable to be paired to any object detection system, if the required input is provided, although, the performance of the estimation may be affected by the performance of the object detection. The real-time capability enables the MAV to perform other tasks concurrently, for example, it may use the object velocity information to improve the object tracking performance.

The project aim was set in anticipation to achieve:

- a reduction in development cost and system complexity of a MAV by eliminating the need to integrate a ranging sensor dedicated to the estimation task;
- a capability extension to existing MAVs with integrated camera without a ranging sensor.

1.3.4. Scope and Objectives

This project was planned to progress through below objectives to achieve the project aim:

1. Develop the real-time object velocity estimation system in a simulated environment.
2. Investigate the estimation performance under different types of MAV and vehicle motion.
3. Assess whether the system is applicable at a long-range (operatable at 100 m or a longer distance) and in real-time (100 Hz or higher update rate).
4. (Extended Scope) Test the system on a MAV hardware.

This project focuses on the development of a methodology to process the object bounding box for object velocity estimation. The project scope does not include the development and performance analysis of the object detection system.

1.3.5. Assumptions

This project was conducted based on a list of assumptions as follows:

- Flat-surface assumption
    - It was assumed that the problem would become extremely challenging to simulate an object in a map with dynamic altitude. To simplify the problem, for the project to be feasible within the given time limit, it was assumed that the ground vehicle placed in an environment with constant altitude.
    - This project represents coordinates in North-East-Down convention.
- MAV
    - The simulated MAV was assumed to move at maximum of 50 m/s (180 km/h)
    - Any inertial state of the MAV was assumed to be accessible for the estimation. Those states include global position, attitude or other MAV states that is required for the estimation.
- Camera
    - This project assumes that the camera is strapdown to a MAV and the relative position and attitude of the camera to the MAV is kept constant throughout the estimation (i.e. no gimbal involved).
    - The camera parameters are also known prior to the estimation.
- Object
    - The simulated object is a ground vehicle which is assumed to move at a constant linear or angular acceleration.
    - The vehicle is assumed to move at a maximum 30 m/s (108 km/h) of speed. This is to ensure the system to be functional for measuring an object moving at a high speed.

2. LITERATURE REVIEW

This section reviews the works relevant to the project to achieve the project aim based on the scope and assumptions. The papers were selected with consideration that the system must be applicable for real-time, long-range and sensors with noise in addition to satisfying the project assumptions. Initially, a several camera types and its models were investigated to identify the suitable methodology to visually measure the object position, which is to be differentiated into velocity information. To achieve a system robust to measurement noise, examples of state estimation filters and their general performance were reviewed to be applied for the most appropriate camera model identified from the review of camera types and models. Lastly, other applicable methodologies were investigated to extract the object velocity measurement to be fused with the differentiated position estimate to seek possible improvement in the velocity estimation.

2.1. Camera Types and Models

Focusing on object position measurement using camera vision, there are various approaches available where the available options differ by the camera types due to the different projection properties (Guo & Li, 2022; Kim, et al., 2020; Trigkakis, et al., 2020). The different types of camera vision are reviewed in this section to determine the type of vision that would be the most efficient to achieve the project aim.

A binocular camera is one of the visual sensors frequently used for visual-ranging tasks on autonomous systems (Lee & Yoon, 2018; McGuire, et al., 2017; Strydom, et al., 2015). The distance to an object from the camera is measured by the difference in object position on the two imaging planes with a known baseline (i.e. the distance between the parallel perspectives) and camera parameters, using trigonometry, similar to the model shown in Figure 1 (Krotkov, et al., 1990; Ma, et al., 2019; McGuire, et al., 2017).

Figure removed due to copyright restriction

Figure 1: Example model for binocular camera projection geometry (Ma, et al., 2019)

This method is applicable in real-time under the camera and object motion; however, it is not often applied for a range beyond 10 m maximum (Lee & Yoon, 2018) (McGuire, et al., 2017) (Strydom, et al., 2015). This is due to the feasible range being proportional to the baseline which causes the measurement to be noisy at a long-range (Strydom, et al., 2015) or requiring the baseline distance to

be increased which causes the sensor to grow to the size infeasible for a small vehicle (e.g. MAV). Thus, it is expected that this methodology would not be applicable for long ranges. Also binocular cameras are often more costly than monocular cameras in both processing time and cost (Engel, et al., 2014; Trigkakis, et al., 2020), although, the processing time can be varied depending on the approach.

Figure removed due to copyright restriction

Figure 2: Object (left) distorted in image (right) by translational motion due to rolling shutter effect (Kim, et al., 2020)

A monocular camera with a rolling shutter returns an image distorted by the rolling shutter (RS) effect as shown in Figure 2. In Chun, et al. (2008), Fan, et al. (2021), Lao, et al. (2018), Wu, et al (2021), researchers attempted to remove the unwanted distortion to ease image processing tasks (e.g. object detection). It was shown that the image distortion can be corrected and can be utilised to extract information such as the depth map (Fan, et al., 2021), the inertial motion of the camera (Grau & Pansiot, 2012) and the motion (e.g. position, velocity, etc) of the object (Ait-Aider, et al., 2006; Kim, et al., 2020). It is assumed that the restored image would still consist of distortion on an object moving at a fast speed across the imaging plane, although, in the project context, a ground vehicle seen from a MAV at a maximum of 300 m distance brings a concern that it would not always move considerably fast in the image to cause the distortion. Furthermore, the literature mostly focuses on extracting depth maps on stationary objects/backgrounds or stationary cameras that seem to require a significant knowledge to understand and utilise the RS effect. It was assumed that the given project time would be insufficient to investigate the RS effect in the appropriate detail to achieve extracting velocity information from the project scenario where both camera and the object move at varying speeds.

A global shutter on a monocular camera provides detailed visual information with minimal distortion in the image compared to the RS cameras as all part of images are taken at a single instance; note that different types of distortion (e.g. motion blur, camera lens distortion) may still exist (Fan, et al., 2021). This property ensures that the image of an object is taken at a constant time-interval regardless of the object position in the image. The time-interval can slightly vary in RS camera images due to the rolling-shutter effect. The global shutter camera provides more simple

solution than RS camera in estimating object velocity from the differentiated object position. This is because the accuracy in estimated velocity is sensitive to the time measurements.

Figure removed due to copyright restriction

Figure 3. Pinhole camera projection model (Beard & McLain, 2012)

The pinhole camera model, as shown in Figure 3, is one of a camera model that is often used to extract the object position relative to the camera under certain assumptions (Fan, et al., 2021; Grau & Pansiot, 2012; Kim, et al., 2020). Those assumptions include that the object is on a flat surface with the camera parameters known prior to the estimation and the MAV position and attitude to be accessible for the time that each image was taken (Barber, et al., 2006; Cai, et al., 2022; Pan, et al., 2023; Trigkakis, et al., 2020). Note that all the listed assumptions do align with the project assumption. The advantage of using the monocular global shutter camera by the pinhole camera model is that the maximum measurable range is not dependent on the sensor size as opposed to the binocular camera (from the baseline distance requirement). Thus, the maximum measurable range is dependent on the detection range of the object detector/classifier, which is often much longer than the maximum measurable range of stereo cameras (Strydom, et al., 2015).

Based on the camera model review, it was expected that using the monocular global shutter camera would be best suited for the project out of all camera types, that meets the requirement, within the given project duration.

2.2. State Estimation Filters

It is essential to have a system robust to noise. Any noise in the measurement, regardless of the system model, can easily cause errors in the estimation. These must be filtered. The performance of the filter and whether it is applicable for the system, varies by the system model. For this project four types of filters were reviewed: Kalman Filter (KF) (Kalman, 1960), Extended Kalman Filter (EKF) (McGee, et al., 1962), Unscented Kalman Filter (UKF) (Julier & Uhlmann, 2004) and Particle Filter (PF) (Del-Moral, 1996), with consideration that the system uses the pinhole camera model for position measurement, as identified in Section 2.1.

KF and its variants (Julier & Uhlmann, 2004; Kalman, 1960; McGee, et al., 1962) are one of the most frequently used state filters for those estimation problems that uses a mathematical model of the state and the measurement, with assumption that the noise in both state and measurement is Gaussian. KF improves the estimation by interchanging between state prediction and update phase. It is a filter that can efficiently reduce noises in the linear system with the predict-update concept which has a low calculation cost (Kim, 2023). However, KF is not often used for autonomous systems since most of them are nonlinear, and KF cannot be applied to nonlinear system, unless it can be linearised (Maley, 2015; Sun, et al., 2015; Zhang, et al., 2005; Zhang, et al., 2010). The pinhole camera model is linear, although, converting the relative position to a global position involves multiple rotation matrices which is a nonlinear process. To overcome this issue, a number of variations of KF was developed to apply the same concept to the nonlinear system (Julier & Uhlmann, 2004; McGee, et al., 1962).

Extended Kalman Filter (EKF) is one of the KF variants developed for a nonlinear system. It linearise the system using the Jacobian matrix of the system state transition and measurement model to update the covariance matrices (McGee, et al., 1962). EKF is one of the filters that has been dominantly used over multiple decades for autonomous systems due to its low-cost and high-efficiency performance to the nonlinear system regardless of the system platform (McElhoe, 1966; Suddath, et al., 1967; Wu & Karkoub, 2019). However, EKF is often not suitable for highly non-linear systems due to the possibility of divergence in the estimation due to the linear approximation of highly nonlinear systems (Garcia, et al., 2019; Julier & Uhlmann, 2004; Maley, 2015). EKF also uses an assumption that the system noise is Gaussian (McGee, et al., 1962) which may not be appropriate to filter the noise from a visual measurement (Benini, et al., 2012), although there are a few successful cases of using EKF on visual estimation (Jeon, et al., 2022; Wu & Karkoub, 2019). Thus, EKF may show the best for the project as the proposed system uses visual measurements (i.e. object detection and optical flow estimate) as the prime source for the object velocity estimation with highly non-linear system model (i.e. rotation matrices).

Unscented Kalman Filter (UKF) is another KF variant for nonlinear system. It uses unscented transformation (UT) to estimate the probability distribution of noise in individual state vectors (Julier & Uhlmann, 2004). The filter does not approximate the system but rather estimates the state distribution by mapping sigma points throughout the state spaces which preserves the nonlinear properties of the system. This supports the filter to achieve a better performance compared to EKF for highly nonlinear system (Fang & Huang, 2013; Garcia, et al., 2019). Despite the Gaussian noise assumption, the filter also successfully filtered visual measurements in some research (Cha & J. G. Chen, 2016; Sun, et al., 2015; Zhang, et al., 2005). Furthermore, the performance can almost be similar to EKF with higher processing cost in some cases, although, the UKF tends to converge much faster than EKF (Garcia, et al., 2019). Considering the points, it is expected that the UKF would perform better for the project than EKF for the appropriate system approximation and to achieve a fast recovery of the error from the processing delay.

Particle Filter (PF) is a measurement filter that can estimate the noise distribution of a nonlinear system without being limited by the gaussian model by plotting sampled estimation from the estimated probability (Del-Moral, 1996). The performance is proportional by the amount of sampling done for the estimation which can often achieve high accuracy on any nonlinear system as it does not take any assumption on the system or the noise unlike EKF (system approximation) and UKF (Gaussian assumption) (Del-Moral, 1996); however, due to the computational cost from the sampling approach, it is often not best suitable for real-time applications and still needs solution for processing optimisation which is a continuing issue of PF despite the advances in current technology (Moreno, et al., 2009; Saha, et al., 2010; Wahrudin, et al., 2019; Zhang, et al., 2010). Thus, it is assumed that PF is not suitable to satisfy the project aim in having a fast estimation system for real-time application.

Overall, the literature suggests that UKF would be the most suitable state filter for the project as KF and PF are expected not to be able to satisfy one or more of the project requirements (i.e., nonlinear system model for KF and real-time capability for PF) where EKF is still applicable to the project but expected to perform less effectively compared to UKF. Nevertheless, this project aims to investigate the performance of the system model under both EKF and UKF where the model is to be applied to EKF, prior to UKF, as a baseline.

2.3. Velocity Measurement

The filter performance depends on whether what system model it is applied to and what measurements are available, where the performance often improves with having more measurement available than less (Del-Moral, 1996; Julier & Uhlmann, 2004; Kalman, 1960; McGee, et al., 1962). The existing methodology to achieve the object velocity estimation has been identified in previous sections, and additional reviews were done to identify other methodologies to extract the object velocity information from the visual field to be fused with the differentiated position in anticipation to achieve a better system performance.

The methodology to estimate the object velocity from image sequence is actively being investigated with various approaches, including extracting the velocity information from the motion blur (Lin, 2005; Lin & Li, 2004; Mohammadi, et al., 2010) or the flow in visual field, which is also referred as optical flow (Ashraf, et al., 2018; He, et al., 2017; Ho, et al., 2017).

Figure removed due to copyright restriction

Figure 4. Clear (left) and blurred (right) object image due to object motion (Lin, 2005)

The motion blur in the image, as shown in Figure 4 for example, is a capture of a visual motion that appeared during the short exposure time of the camera shutter (Mohammadi, et al., 2010). It is often an undesired distortion for tasks such as object detection, although, it can be processed into a velocity information of an object by measuring the magnitude and direction of the blur (Lin, 2005; Lin & Li, 2004; Mohammadi, et al., 2010). However, all reviewed works assume the camera to be stationary and no literature was founded on its application for moving cameras. The additional blur component from the camera motion causes difficulty in decoupling the blur caused by two different motions (i.e. camera motion and object motion). It may be possible to use this methodology, with detailed investigation to decouple the blur effect from camera/object motion, although, it is expected to be difficult to develop such system within the due time with the limited literature available.

Figure removed due to copyright restriction

Figure 5. Visualisation of optical flow between current and previous image frame (He, et al., 2017)

Optical flow is the flow in the visual field that can be measured from consecutive images (Szeliski, 2011). The optical flow in the image stream, when the camera is facing the ground, can be used to derive the inertial velocity of the camera (e.g. MAV) as it is a measure of movement in the image due to the camera motion (Ashraf, et al., 2018; He, et al., 2017; Ho, et al., 2017). This alternatively means that relative velocity of the ground can be found with respect to the camera and ultimately, it is possible in some cases that the same methodology can be applied to measure the relative velocity of the object to the camera with measuring the flow on the object imagery. He, et al (2017) has developed and tested such algorithm to estimate the velocity of the moving vehicle with moving camera, as shown in Figure 5, which performed well for relative speed of up to 70 m/s; although, the test was conducted under a simulated environment only with maximum 5% noise in the optical flow measurement with no noise added to the camera state which does not guarantee the same performance for the identical scenario with real-world results (He, et al., 2017).

The optical flow is calculated based on the assumption that the intensity I of the point at world coordinate x is constant over the time which is represented by the optical flow equation as shown by Eq. (1) (Beauchemin & Barron, 1995):

$$\boldsymbol{I(x,t) \approx I(x+\delta x, t+\delta t)} \quad \textbf{(1)}$$

Eq. (1) is under-constrained which requires additional constraint and there are approaches that apply different constrains to solve the problem. For example, the classical methods (e.g. template matching, Lukas-Kanade) constrains Eq. (1) assuming that the brightness pattern of an object is constant and rich in texture or the flow between consecutive images is appropriately small for Taylor Series approximation (Lucas & Kanade, 1981; Szeliski, 2011). Both of these constraints may not be applicable to the project as the object (i.e. ground vehicle) in the image is expected to often move significantly due to the motion, and the brightness of a world point in the image can often be easily affected by the camera movement and other noise (Szeliski, 2011).

It is speculated that the model/equation for the estimation of inertial velocity of a camera (Andersh, et al., 2015; Zhong & Chirarattananon, 2020) would still be applicable for the object velocity estimation, if the flow is measured from a stationary camera. Another speculation is that, using the object position estimate on the previous x_{t-1} and current time x_t, the two world coordinates can be converted into image coordinates of the same image using the pinhole camera model and the difference between the two image coordinates may be possible to be represented as the estimated optical flow $\delta\hat{I}$ as shown in Figure 6.

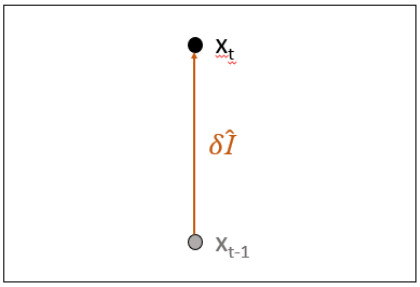


Figure 6. Diagram representation of the speculation on optical flow estimation

To speculate, the velocity estimate from the method (Figure 6) and differentiated position could be the same measurement that goes through two different models. This may result in better, similar, or even worse performance in comparison to using only the position differentiation. The two model can be complementary to one another, for example, one model can be sensitive to a variable while the other is not. Unless the two models are complementary as such, it is expected the system performance would significantly decrease due to the position error with its effect amplified through the two model. Despite the uncertainty, it was planned to review the mentioned optical-flow methodology once the baseline EKF (i.e., EKF with differentiated object position) is developed. Note that this method may be disregarded when any flaw in the above methodology is found in the further review.

2.4. Summary

In summary, it was found from the literature that there is range limitation to the position estimation using binocular camera, whereas using the RS camera would be difficult to achieve within the given project time. Thus, the pinhole camera model was selected to be used to derive the object position measurement that can be differentiated for the object velocity estimation. Multiple state estimation filters were reviewed for the resultant system to be robust to the noises. EKF and UKF were selected for the implementation due to being applicable for nonlinear system and applicable for real-time. The model is to be applied to EKF prior to UKF as a baseline to be compared with the proposed system. Further review and research are to be done on taking velocity measurement by optical flow once the baseline system (EKF with differentiated object position) has been implemented.

3. METHODOLOGY

Based on the discussion on the project methodology (i.e. Section 2), the baseline and proposed system were planned to be designed as shown in Figure 7 and Figure 8 respectively. This section describes the development process of the baseline system. Note that it was not possible to develop the proposed system (Figure 8) within the given project time. This will be included in the future work.

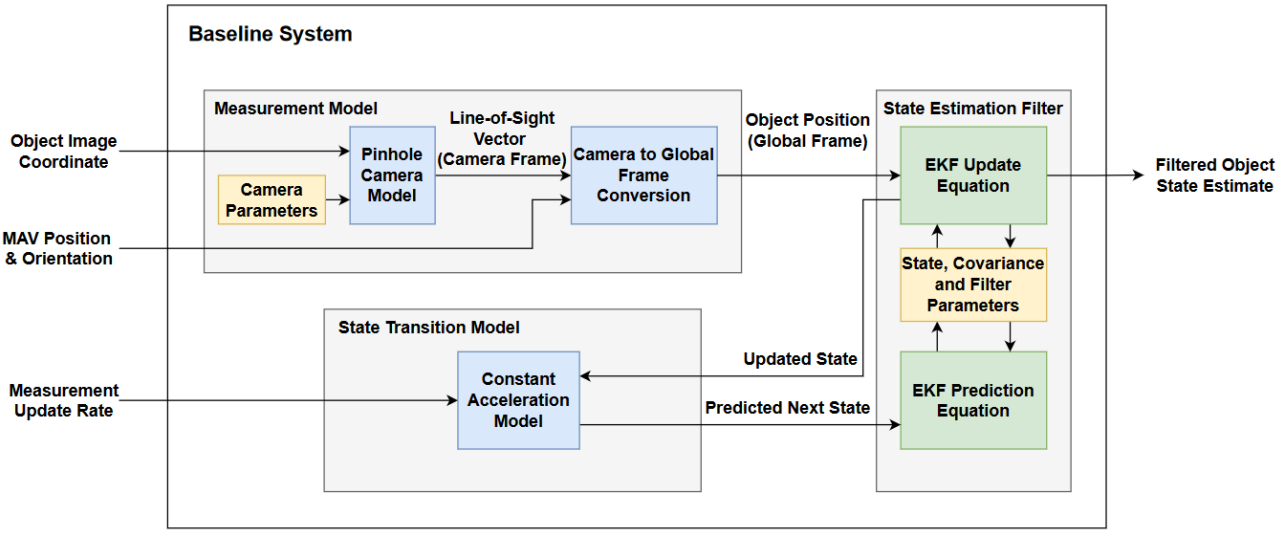


Figure 7. Baseline System Diagram

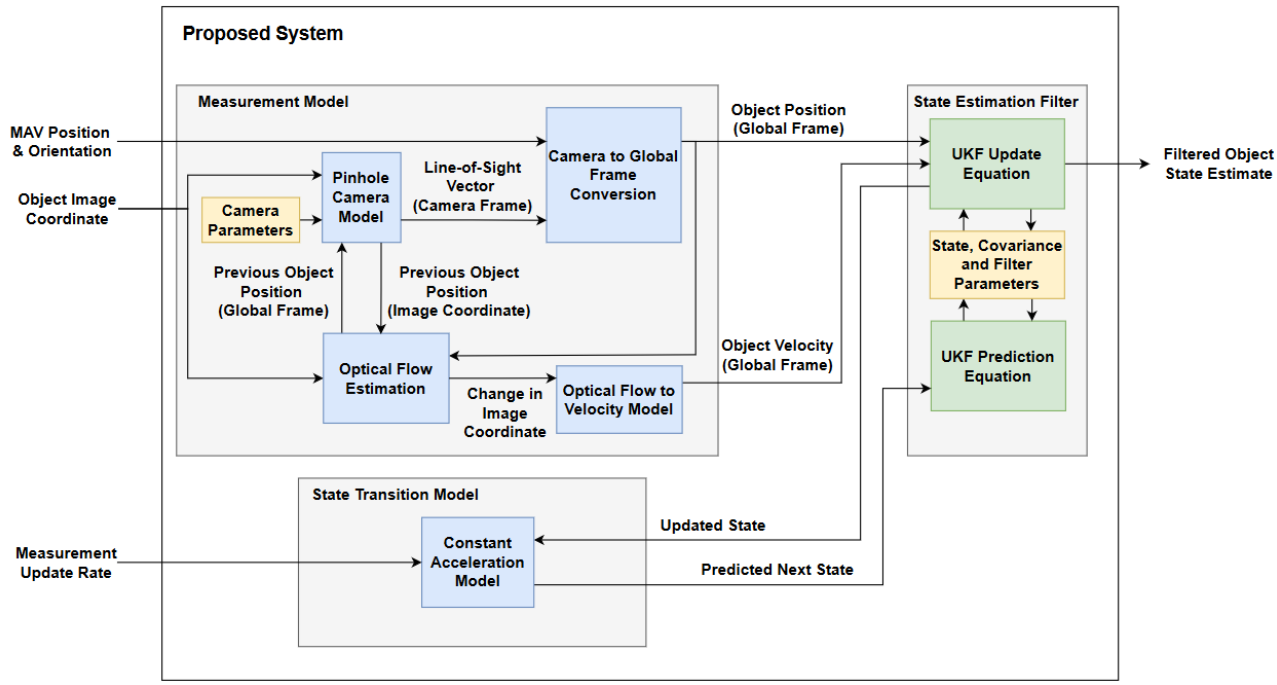


Figure 8. Proposed System Diagram

3.1. Equipment and Settings

3.1.1. Hardware

All the software and scripts were used in a remote desktop with the following specifications:

- OS – Ubuntu 20.04.6 LTS
- Processor – Intel Xeon (R) Platinum 8362 CPU @ 2.80GHz x 16
- RAM – 64 GiB
- GPU – 8GB slice of V100s

This hardware was capable to run the required software and scripts, although, there was a frequent latency and crashes while running a software for dataset extraction as the imaging tasks were highly demanding in graphics and processing in a certain environment.

3.2.2. Software

The software used for the project are listed below with a description of the purpose:

- Python (v3.8.10)
    - All scripts used and developed for this project is written in python programming language.
    - Refer Appendix A for list of packages and the versions used for this project.
- Airsim (v1.8.1 - Linux)
    - This software was used to generate imagery of the simulated environment where the environment settings for each development process are described in Section 3.3 and 3.4.
    - This software was used by downloading the binaries with no other installation process involved.
    - Refer Appendix A for list of python packages and the versions used for this software.
- Ardupilot Simulation-in-the-Loop (SITL)
    - This software was used to simulate flight dynamics of the MAV to generate a sample dataset to test the system model. More detail is to be discussed in Section 3.3.
    - The software was pulled from git with last update on 6th September 2023.
    - ArduPlane (v.4.5.0-dev) was used for the dynamics of simulated vehicle.
    - Refer Appendix A for list of python packages and the versions used for this software.

3.2. System Development Procedure

Refer Appendix B for the project Gantt chart to see the project timeline.

3.2.1. Geolocation Model Description

The project was commenced with implementation of the model that converts image coordinate to world coordinates, referred as Geolocation Model. This project used the geolocation model that Beard, et al (2012) describes in Section 13.3. The model is described as follows.

Based on the Pinhole Camera Model (Figure 3), let the unit vector, in any frame of reference, that represents the direction from camera to the object be denoted as $\check{\boldsymbol{\ell}}$ that is consisted of horizontal (x), vertical (y) and depth (z) component where $\boldsymbol{\ell}$ is the same directional vector with magnitude $\mathbb{L}$ with the relationship as shown in Eq. (2).

$$\check{\boldsymbol{\ell}} \triangleq \begin{pmatrix} \check{\boldsymbol{\ell}}_x \\ \check{\boldsymbol{\ell}}_y \\ \check{\boldsymbol{\ell}}_z \end{pmatrix} \triangleq \frac{\boldsymbol{\ell}}{\mathbb{L}} \tag{2}$$

Based on Eq. (2), the described unit vector in camera frame $\check{\boldsymbol{\ell}}^{\boldsymbol{c}}$ then can be represented as Eq. (3) where ϵ_x and $\epsilon_y$is the horizontal and vertical displacement of the image coordinate from the image centre coordinate and f is the focal length of the camera. The displacement of the object image coordinate from the image centre can be found by subtracting the object bounding box centre to the image centre; the focal length f is a constant parameter of a camera which can be found by Eq. (4) with field-of-view v and image width (in pixel) M.

$$\check{\ell}^c = \frac{\ell^c}{|\ell^c|} = \frac{1}{\sqrt{\epsilon_x^2+\epsilon_y^2+f^2}} \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ f \end{pmatrix} \tag{3}$$

$$f = \frac{M}{2\tan(\frac{v}{2})} \tag{4}$$

The unit vector in Eq. (3) needs to be converted from camera frame to vehicle frame to obtain the inertial coordinate of the object where such conversion can be done by multiplying rotation matrix to the coordinate as shown in Eq. (5). Note that a rotation matrix R for conversion from A frame to B frame is represented as R_A^B and the orientation of the vehicle frame is equivalent to inertial frame with its origin at the vehicle position.

$$\check{\ell}^v = R_c^v \check{\ell}^c \tag{5}$$

The generic property of a rotation matrix is that if there is a rotation matrix to convert C to B and B to A, the conversion from C to A is also possible as shown in Eq. (6). In addition, the inverse of a rotation matrix is the rotation matrix for conversion in inverse direction as shown by Eq. (7).

$$R_C^A = R_B^A R_C^B \quad \textbf{(6)}$$

$$R_A^B = (R_B^A)^{-1} \quad \textbf{(7)}$$

The rotation matrices to convert body to vehicle, gimbal to body and camera to gimbal frame are shown in Eq. (8), (9) and (10) respectively where ϕ, θ and ψ is the roll, pitch and yaw of the MAV body and α_{el}/α_{az} is the elevation and azimuth angle of the gimbal, which is equivalent to the elevation and azimuth of the strapdown camera to MAV; meaning that those matrices can be simplified into a single equation to represent the equation to convert a vector in camera frame to inertial frame as Eq. (11). Note that the cosine and sine notation was shorted to c and s respectively for Eq. (8). Refer Beard, et al (2012) for detailed derivation of the rotation matrices.

$$R_v^b(\phi,\theta,\psi) = \begin{pmatrix} c_\theta c_\psi & c_\phi s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{pmatrix} \quad \textbf{(8)}$$

$$R_b^g(\alpha_{el},\alpha_{az}) = \begin{pmatrix} \cos(\alpha_{el})\cos(\alpha_{az}) & \cos(\alpha_{el})\sin(\alpha_{az}) & -\sin(\alpha_{el}) \\ -\sin(\alpha_{az}) & \cos(\alpha_{az}) & 0 \\ \sin(\alpha_{el})\cos(\alpha_{az}) & \sin(\alpha_{el})\sin(\alpha_{az}) & \cos(\alpha_{el}) \end{pmatrix} \quad \textbf{(9)}$$

$$R_g^c = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad \textbf{(10)}$$

$$\widetilde{\ell^v} = R_c^v \widetilde{\ell^c} = R_b^v R_g^b R_c^g \widetilde{\ell^c} = (R_v^b)^{-1}(R_b^g)^{-1}(R_g^c)^{-1}\widetilde{\ell^c} \quad \textbf{(11)}$$

Using the unit vector of relative position of object from MAV, the object position $P_{obj,}$ in inertial frame, can be found by scaling the vector by appropriate length $\mathbb{L}$ and offsetting the unit vector by the MAV inertial position P_{MAV} as shown in Eq. (12).

$$P_{obj}^i = P_{MAV}^i + \mathbb{L}(R_c^v \widetilde{\ell^c}) \quad \textbf{(12)}$$

The length $\mathbb{L}$ can be calculated with the assumption that the vehicle is always at an altitude of 0 m (i.e. flat-surface) as it allows the relative position of the object in z axis is the inverse of the MAV altitude. Letting $\mathbf{k}^i$ to be the z axis component of a vector, Eq. (13) shows that $\mathbb{L}$ is the ratio of the MAV altitude to the z axis component of the relative position unit vector.

$$\mathbb{L} = \frac{-(\boldsymbol{k}^i \cdot P^i_{MAV})}{\boldsymbol{k}^i \cdot R^v_c \widetilde{\ell}^c} \quad \textbf{(13)}$$

Finally, combining Eq. (11), (12) and (13), the object position can be found using the centre of the object bounding box, camera parameters and MAV states by Eq. (14).

$$P^i_{obj} = P^i_{MAV} + \frac{-(\boldsymbol{k}^i \cdot P^i_{MAV})}{\boldsymbol{k}^i \cdot R^v_c \widetilde{\ell}^c} \left(R^v_c \widetilde{\ell}^c\right) \quad \textbf{(14)}$$

Overall, Eq. (14) is the geolocation model used in this project to calculate the raw position measurement and for the state estimation filter as the measurement model.

3.2.2. Geolocation Model Implementation

Once the model (i.e. Eq. (14)) was implemented to a script, to verify the implementation, a sample dataset was obtained from a single instance. The dataset was obtained from Airsim in AirsimNH environment, which has multiple ground vehicles, with simulated MAV that can be either controlled manually by keyboard or by the script. Four states were obtained from the simulation which are ground truth object position, MAV position, MAV orientation and object bounding box in the image taken at the corresponding MAV states. Note that the camera parameters were accessible and adjustable by Airsim commands. For true object position, the position of the world objects spawned in the environment was not accessible, thus, the MAV was manually controlled to be directly above a vehicle and the North and East coordinate of the MAV was recorded assuming it for the true object position with 0 m for the altitude. The MAV was then commanded in a script to move to a point near the same object, as shown in Figure 9, and an image request was sent to airsim while recording the current MAV state. The image, as in Figure 10, was saved and the bounding box of the object was found by manually adding rectangular shape to the image using OpenCV. All the recorded values were saved into a JSON file.

Figure 9. MAV and Vehicle Position for Geolocation Model Test

Figure 10. Image taken from Airsim camera with manually identified object bounding box

The calculated object positions from the obtained variables were printed on the terminal to check whether they match with the ground truth object position, noting the calculated position may slightly vary from the ground truth. The model script was modified until the calculated position matched with the ground truth position. The main code to bind the recorded data is shown in Appendix C, the model script is shown in Appendix D and the script with the Airsim commands are shown in Appendix E. Note that they are the latest versions of the script (as the original script used during the development could not be recovered) and may perform differently with the followed description.

3.2.3. Sample Dataset Extraction

To test the model on a moving MAV, a list of scripts was provided to obtain a sample dataset. The scripts use Ardupilot to simulate the MAV flight dynamics to command and obtain simulated MAV states which is used to set the camera state in Airsim to obtain the image at the MAV states (with azimuth and elevation offset to represent strapdown camera). The image is then used to track the object bounding box in the image where the object bounding box must be initially selected by manually clicking the bounding box that contains the object where the bounding box is returned from blob detection filtered by colour threshold. The script uses the object bounding box to identify the object position, using Eq. (14), and commands the MAV to approach the object position estimated by the given states without any filtering which goes back to the commanding phase and loops throughout the simulation. Figure 11 shows the connection of the software to the script and the flow of the process with the list of variables extracted from the script to the dataset file.

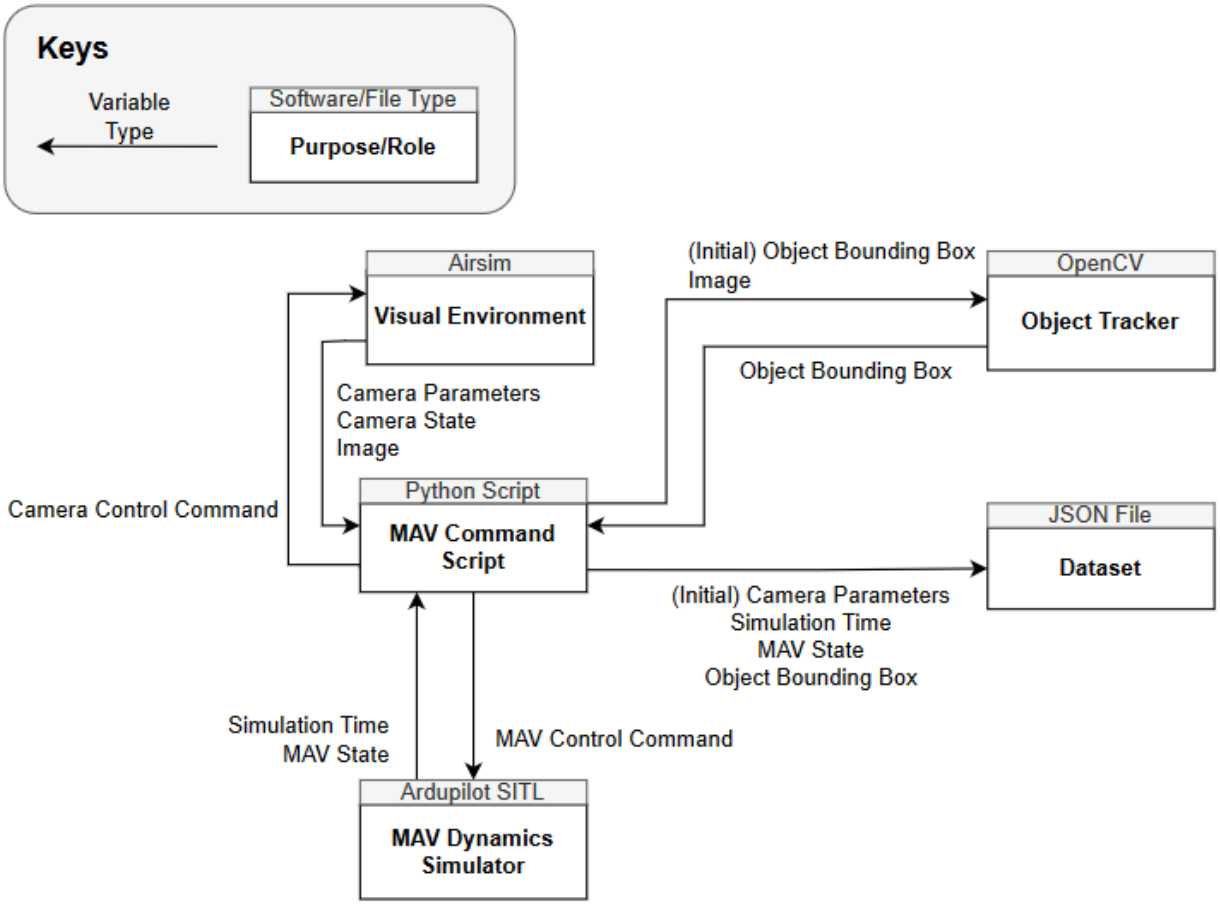


Figure 11. Flow Diagram for Sample Dataset Extraction

3.2.4. Framework Implementation

Initially, the dataset at single instance was stored in separate files (which were combined into one for convenience in the later part of the development). To be able to read multiple datafiles, multiple Python scripts were created with different purpose which were connected as shown in Figure 12. The code corresponding to the Dataset Reader, Data Processor and Filter Script Selector are shown in Appendix F, Appendix G and Appendix H respectively, where the class definition used for the scripts are shown in Appendix I and 'EKF script' was interchangeable between different model scripts if correct variables were provided from the Filter Script Selector. Note that the MAV state was later extracted from the visual environment (Airsim) due to an issue mentioned in Section 3.2.5.

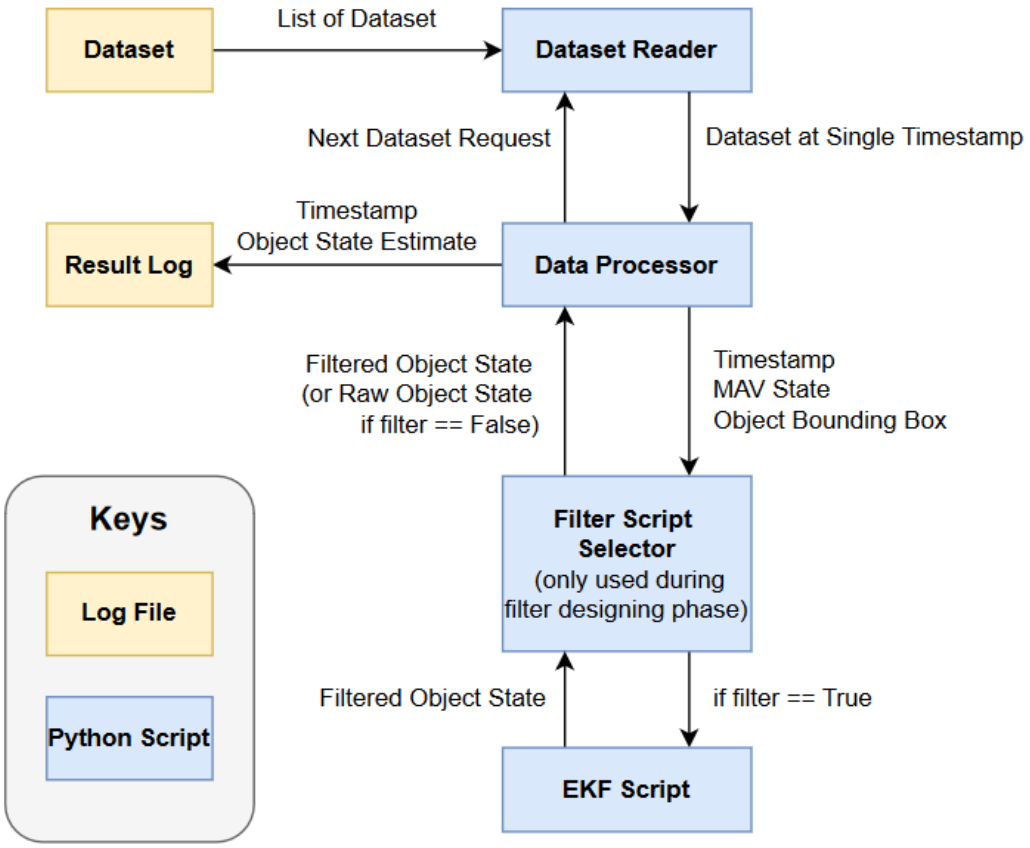


Figure 12. Flow Diagram for Sample Dataset Processing

3.2.5. Model Design and Implementation to Estimation Filters

The concept of KF and EKF was revisited prior to design the mathematical model for the estimation filter. KF is a filter that provides predicted state mean x and the covariance P based on Bayes rule with gaussian distribution (Kalman, 1960). One of the properties of gaussian properties is that, suppose a gaussian distribution is linearly transformed as shown in Eq. (15), the distribution of linearly transformed variables then becomes Eq. (16) (Kim & Ristic, 2023).

$$\boldsymbol{x} \sim \mathcal{N}(\bar{\boldsymbol{x}},\ \boldsymbol{P}_{\boldsymbol{x}}) \ \rightarrow \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \qquad \textbf{(15)}$$

$$\bar{\boldsymbol{y}} = \boldsymbol{A}\bar{\boldsymbol{x}} + \boldsymbol{b}, \quad \boldsymbol{P}_{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{P}_{\boldsymbol{x}}\boldsymbol{A}^{\boldsymbol{T}} \qquad \textbf{(16)}$$

KF has predict equation for state and covariance as shown in Eq. (17) and (18) respectively with update equation Eq. (19) and (20); $\hat{\boldsymbol{x}}$ is the estimated state, $\boldsymbol{F}$ is the state transition matrix, $\boldsymbol{P}$ is the state covariance, $\boldsymbol{Q}$ is the state transition noise, $\boldsymbol{z}$ is the measurement, $\boldsymbol{H}$ is the measurement model, $\boldsymbol{K}$ is the Kalman gain that is calculated as shown in Eq. (21) and $\boldsymbol{S}$ is referred as 'innovation term' calculated as Eq. (22) where $\boldsymbol{R}$ is the measurement noise (Kim & Ristic, 2023). The subscript k indicates the variables at current timestamp and k+1 is for the next timestamp.

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} = \boldsymbol{F_k}\hat{\boldsymbol{x}}_{\boldsymbol{k|k}} \quad \textbf{(17)}$$

$$\boldsymbol{P_{k+1|k}} = \boldsymbol{F_k P_{k|k} F_k^T} + \boldsymbol{Q_k} \quad \textbf{(18)}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}} = \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} + \boldsymbol{K_{k+1}}\left(\boldsymbol{z_{k+1}} - \boldsymbol{H_{k+1}}\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}}\right) \quad \textbf{(19)}$$

$$\boldsymbol{P_{k+1|k+1}} = \boldsymbol{P_{k+1|k}} - \boldsymbol{K_{k+1} S_{k+1} K_{k+1}^T} \quad \textbf{(20)}$$

$$\boldsymbol{K_{k+1}} = \boldsymbol{P_{k+1|k} H_{k+1}^T S_{k+1}^{-1}} \quad \textbf{(21)}$$

$$\boldsymbol{S_{k+1}} = \boldsymbol{H_{k+1} P_{k+1|k} H_{k+1}^T} + \boldsymbol{R_{k+1}} \quad \textbf{(22)}$$

Comparing the KF equations and the linear transformation, it can be found that KF filter uses the linear transformation of gaussian distribution to predict and update the state and covariance, thus, KF is only applicable to a system with linear state transition model and linear measurement model. EKF enables KF to be applicable to a nonlinear system by using Jacobian matrices of the state and measurement model only where the model is required to be in matrix form (i.e. calculating Kalman gain and covariance). With notation for Jacobian matrix for state and measurement model as shown in Eq. (23) and (24) where $\boldsymbol{f}$ and $\boldsymbol{h}$ is the nonlinear state and measurement model, the EKF predict equations for state and covariance are shown in Eq. (25) and (26), and the update equations are as shown in Eq. (27) and (28) respectively, note that it uses the same calculation for Kalman gain $\boldsymbol{K}$ and innovation term $\boldsymbol{S}$.

$$\boldsymbol{F_k} = \left[\nabla_{\boldsymbol{x_k}} \boldsymbol{f}^{\boldsymbol{T}}(\boldsymbol{x_k})\right]^{\boldsymbol{T}} \quad \textbf{(23)}$$

$$\boldsymbol{H_k} = \left[\nabla_{\boldsymbol{x_k}} \boldsymbol{h}^{\boldsymbol{T}}(\boldsymbol{x_k})\right]^{\boldsymbol{T}} \quad \textbf{(24)}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{\boldsymbol{k|k}}) \quad \textbf{(25)}$$

$$\boldsymbol{P_{k+1|k}} = \boldsymbol{F_k P_{k|k} F_k^T} + \boldsymbol{Q_k} \quad \textbf{(26)}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k+1}} = \hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}} + \boldsymbol{K_{k+1}}\left[\boldsymbol{z_{k+1}} - \boldsymbol{h}(\hat{\boldsymbol{x}}_{\boldsymbol{k+1|k}})\right] \quad \textbf{(27)}$$

$$\boldsymbol{P_{k+1|k+1}} = \boldsymbol{P_{k+1|k}} - \boldsymbol{K_{k+1} S_{k+1} K_{k+1}^T} \quad \textbf{(28)}$$

The Jacobian matrices can be mathematically calculated, although, it is often calculated numerically in practice for those system with complex system equations (Beard & McLain, 2012). Due to the difficulties in mathematically calculating the Jacobian matrix of the measurement model, Eq. (14), this project decided to use the numerical approach. The code implementation of the numerical approach for Jacobian calculation is shown in Appendix J.

With the understanding of these concepts, it was commenced to develop the system model. Several different models were tested with the dataset obtained from Section 3.2.3. First, the model from Chapter 13.3 of Beard, et al (2012) was referred which assumes that the object is stationary; although this project aims to have a moving vehicle, this was referred initially to gradually proceed to the larger scope and the sampled dataset also uses stationary vehicle. The model described in Chapter 13.4 of Beard, et al (2012) was also attempted to be implemented, although, it was unsuccessful, and the trend of filtered state estimation did not match with the unfiltered state estimation.

Multiple different combination of state and measurement models were tested where a few important decisions were made from the results. One of the decisions is to extend the object state variable from 2D vector with North and East coordinate to NED coordinate. This was to check if the flat-surface assumption is applied correctly to the model where it was expected that the Down coordinate must be zero at all times. This decision was made when testing the model with state and measurement as shown in Eq. (29) and (30) respectively which was a modified model from Beard, et al (2012) where state vector of the model contains the inertial position (NE) of the object $\widehat{\boldsymbol{p}}_{\boldsymbol{obj}}^{\boldsymbol{i}}$ and the distance to the object from the MAV $\widehat{\mathbb{L}}$ as shown in Eq. (29). When the model was implemented with the line-of-sight measurement $\check{\boldsymbol{\ell}}$ as Eq. (30), obtained from the image coordinate by Eq. (3), it is found that the estimation diverges rapidly when without having the distance $\widehat{\mathbb{L}}$ in the state vector as the line-of-sight vector does not provide any information of the depth component (i.e. distance to the object) of the image. This was noticed when the object position was extended from 2D to 3D coordinate, the Down coordinate was not fixed to 0 but rather increased from 0 to a large number (e.g. 10000 m).

$$\widehat{\boldsymbol{x}} = \left\{ \widehat{\boldsymbol{p}}_{\boldsymbol{obj}(\mathbf{2}\times\mathbf{1})}^{\boldsymbol{i}},\ \widehat{\mathbb{L}} \right\}^{\boldsymbol{T}} \quad \textbf{(29)}$$

$$\mathbf{z} = \left\{ \check{\boldsymbol{\ell}}_{(\boldsymbol{3x1})} \right\}^{\boldsymbol{T}} \quad \textbf{(30)}$$

Throughout the development, multiple combinations of variables were put to the state vector as a test. The object states were extended to velocity and acceleration based on Eq. (31) and (32) where p, v, a is the position, velocity and acceleration with NED components respectively and T is the sampling time interval. MAV position and orientation were also put to the state vector in anticipation to help in reducing measurement prediction error; the state vector of the tested model with combination of MAV and object states are shown in Eq. (33) where $\boldsymbol{\Theta}$ indicates the orientation in roll, pitch and yaw.

$$\mathbf{p_{t+1}} = \frac{1}{2}\mathbf{a_t}\boldsymbol{T^2} + \mathbf{v_t}\boldsymbol{T} + \mathbf{p_t} \quad \textbf{(31)}$$

$$\mathbf{v_{t+1}} = \mathbf{a_t}\boldsymbol{T} + \mathbf{v_t} \quad \textbf{(32)}$$

$$\hat{\boldsymbol{x}} = \left\{ \hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{MAV}(3\times1)},\ \hat{\boldsymbol{\Theta}}^{\boldsymbol{i}}_{\boldsymbol{MAV}(3\times1)},\ \hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{obj}(3\times1)},\ \hat{\boldsymbol{v}}^{\boldsymbol{i}}_{\boldsymbol{obj}(3\times1)},\ \hat{\boldsymbol{a}}^{\boldsymbol{i}}_{\boldsymbol{obj}(3\times1)} \right\}^{\boldsymbol{T}} \quad \textbf{(33)}$$

The model was also tested with putting the relative states of object to the MAV which was inspired by Maley (2015) and Nielson, et al (2017) as shown in Eq. (34) where the states with arrow (e.g. $\hat{\vec{\boldsymbol{p}}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}}$) are the relative state of object to the MAV.

$$\hat{\boldsymbol{x}} = \left\{ \hat{\vec{\boldsymbol{p}}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\vec{\boldsymbol{v}}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\vec{\boldsymbol{a}}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)} \right\}^{\boldsymbol{T}} \quad \textbf{(34)}$$

Despite the attempts on implementing different models, none of the attempts were successful. After those attempts, it was advised that it would be the best to keep the state vector only to be filled with the object states. The state vector was kept as shown in Eq. (35) with nearly constant acceleration model. The state transition matrix and noise model for the nearly constant acceleration model in 1-dimension (i.e. Eq. (36)) are shown in Eq. (37) and (38) respectively where the same model was applied to North and East component of the object states with all Down components set to 0. dt is the sampling time interval between t and t+1 and q is the noise in object acceleration (DSTL, 2021). The derivation of Q_t is shown in Appendix K.

$$\hat{\boldsymbol{x}} = \left\{ \hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\boldsymbol{v}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\boldsymbol{a}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)} \right\}^{\boldsymbol{T}} \quad \textbf{(35)}$$

$$\hat{\boldsymbol{x}} = \{ \hat{\boldsymbol{p}}^{\boldsymbol{i}}, \hat{\boldsymbol{v}}^{\boldsymbol{i}}, \boldsymbol{a}^{\boldsymbol{i}} \}^{\boldsymbol{T}} \quad \textbf{(36)}$$

$$\boldsymbol{F_t} = \begin{bmatrix} \mathbf{1} & \boldsymbol{dt} & \frac{\boldsymbol{dt^2}}{\mathbf{2}} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{dt} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad \textbf{(37)}$$

$$\boldsymbol{Q_t} = \begin{bmatrix} \frac{\boldsymbol{dt^5}}{\mathbf{20}} & \frac{\boldsymbol{dt^4}}{\mathbf{8}} & \frac{\boldsymbol{dt^3}}{\mathbf{6}} \\ \frac{\boldsymbol{dt^4}}{\mathbf{8}} & \frac{\boldsymbol{dt^3}}{\mathbf{3}} & \frac{\boldsymbol{dt^2}}{\mathbf{2}} \\ \frac{\boldsymbol{dt^3}}{\mathbf{6}} & \frac{\boldsymbol{dt^2}}{\mathbf{2}} & \boldsymbol{dt} \end{bmatrix} \cdot \boldsymbol{q^2} \quad \textbf{(38)}$$

The measurement vector used for the project is shown in Eq. (39) with measurement model as in Eq. (14). The measurement noise matrix **R** is a diagonal matrix with the diagonals filled with the covariance of respective measurement (δ^2).

$$\mathbf{z} = \left\{\epsilon_{(2x1)},\ \widehat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{MAV(3\times1)}},\ \widehat{\boldsymbol{\Theta}}^{\boldsymbol{i}}_{\boldsymbol{MAV(3\times1)}}\right\}^{\boldsymbol{T}} \quad \textbf{(39)}$$

The described model was implemented (code shown in Appendix L) and tested on the sample dataset which returned result as shown in Figure 13. Note that the result also shows the filtered result from KF; the KF implementation code (same state model, object position in NED for the measurement) was received as an example to refer while developing the state and measurement model for EKF, in addition, note that Figure 13 is missing the title for the horizontal axis which indicates time in seconds.

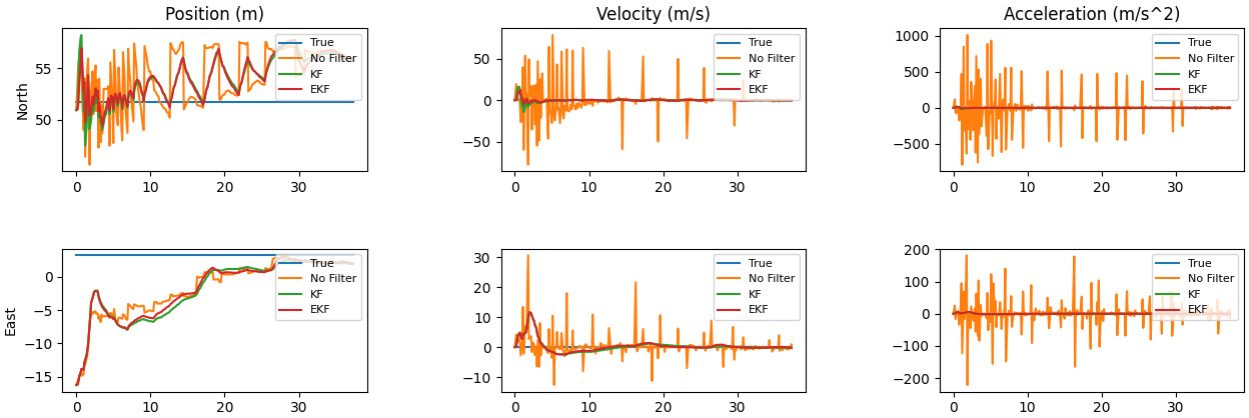


Figure 13. Result from Sample Dataset

Despite the effort in testing multiple different models, the filter was not able to give the expected accuracy. However, it was noted that the unfiltered state was having a strong noise and jumps between the measurement despite that the sampled dataset is not expected to have such a large noise; furthermore, the estimated North position of the object seemed biased to a larger value. It was initially assumed that it was from the object bounding box centre being not centred on the object (Figure 14), although, it was noticed that the bias should be present in East coordinate and be smaller than the actual bias. When the sampled dataset was revisited, multiple critical issues were found which caused the filtered state to be inaccurate.

Figure 14. True (red) and measured (green) object bounding box centre with vehicle facing North (bottom-to-the-top)

Issue 1. MAV Orientation Assignment

The orientation of the MAV was stored as a list with three items: roll, pitch and yaw. It was initially assumed that those orientation values were stored as sequence of [roll, pitch, yaw], although, due to the Airsim convention of storing orientation, the values were stored as [pitch, roll, yaw]. This was noted during the implementation of geolocation model (Section 3.2.2), however, this was completely disregarded during the implementation of filter model which caused a significant delay (i.e. 3 weeks) in the project progress. The effort, during those delays, was focused on re-designing the filter model as described previously as the result from unfiltered position was more accurate than the filtered position. This was due to the geolocation model used to display unfiltered position measurement was using correct values for roll, pitch and yaw which was not the case for the measurement model of the filter. Once the correct roll, pitch and yaw value was used for the filter model, the estimation in East position have improved, although, the unexpected jumps in the unfiltered measurement still existed (Appendix N).

Issue 2. Timestamp Measurement

Another issue with the sampled dataset was that the timestamp was recorded incorrectly. The timestamp was originally extracted from the PC clock time, although, it was later found that the simulation time (of Ardupilot) must have been used instead of the PC clock as the flight dynamic of the MAV was slowed down by 0.2 of speed by the provided script. The timestamp was corrected to be extracted from the simulation which reduced the jumping effect, although the effect still exsisted (Appendix O).

Issue 3. Airsim State Lag

The MAV command script (of Figure 11) commands the Airsim camera to be placed at the MAV position received from Ardupilot where the camera orientation is calculated to be at a certain elevation and azimuth to the MAV orientation. The last issue found from the dataset is that, when the Airsim receives the camera pose command, there is a delay in Airsim to update the camera orientation. This caused the estimation to diverge during the delay, as the measured MAV orientation (from Ardupilot) does not match with the actual MAV orientation when the image was taken (Airsim camera orientation update). The mismatch in the measurement could be treated as the measurement noise, although, the noise is not gaussian but is rather at a stair-like trend (Appendix M). Thus, the Airsim camera orientation was directly accessed, and the elevation and azimuth angle were removed to obtain the MAV orientation. The estimation result after the correction is shown in Figure 15. The unfiltered position estimate only contains noise within expectation with a little bias ($<\pm 1$ m) due to the bias in object bounding box centre.

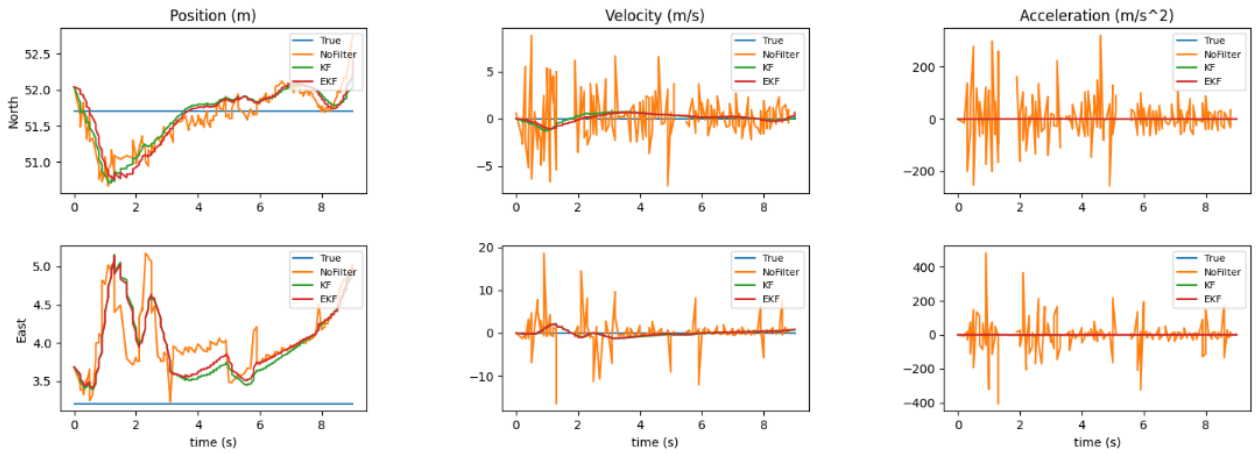


Figure 15. Estimated position after MAV state correction (from Ardupilot to Airsim)

Once the dataset issues were resolved, the filter parameters were adjusted accordingly. The filter initially assumes that the object is stationary (i.e. zero velocity and zero acceleration) at the initial position measurement, as Eq. (40), with initial covariance as shown in Eq. (41) where $\boldsymbol{p}_{\mathbf{max}_{error}\,OBJ}$ is the maximum object position error, $\boldsymbol{v}_{max_{OBJ}}$ is maximum object velocity and $\boldsymbol{a}_{max_{OBJ}}$ is maximum object acceleration. The covariance should be the square of standard deviation ($\boldsymbol{\sigma}$) where it was assumed that $\boldsymbol{\sigma}$ would be roughly similar to the third of the maximum values as, in general, approximately 99.7% of samples fall within 3σ range (i.e. ≈ maximum deviation) of gaussian distribution. The maximum object velocity is set to 30 m/s, given by the project assumption, and the maximum position error was set to 10 m for both North and East component where the maximum values for Down components were set to 0 to indicate the filter that the object is not going to be taking any motion in Down component. The maximum object acceleration is temporarily set to 1.5 G and it is to be adjusted when the dataset with moving object is obtained; similarly, the object acceleration noise **q** (of Eq. (38)) was temporarily set to 1 ms^{-2}.

$$\widehat{\boldsymbol{x_0}} = \left\{\widehat{\boldsymbol{p_0}}_{OBJ_{(3\times1)}}, \mathbf{0}_{(3\times1)}, \mathbf{0}_{(3\times1)}\right\}^{T} \qquad \textbf{(40)}$$

$$\widehat{\boldsymbol{P_0}} = \boldsymbol{diag}\left\{(\boldsymbol{p}_{\mathbf{max}_{error}\,OBJ}/3)^2{}_{(3\times1)}, (\boldsymbol{v}_{max_{OBJ}}/3)^2{}_{(3\times1)}, (\boldsymbol{a}_{max_{OBJ}}/3)^2{}_{(3\times1)}\right\} \qquad \textbf{(41)}$$

The measurement noise matrix R was set as shown in Eq. (42) where $\boldsymbol{\epsilon}_{max\_error}$ is the maximum error of object bounding box centre (image coordinate), $\boldsymbol{p}_{max\_error_{MAV}}$ is for the MAV position error and $\boldsymbol{\Theta}_{max\_error_{MAV}}$ is for the MAV orientation error at maximum. When the filter was tested with different combinations of measurement noise parameters, it was noticed that there was minimal impact to the filter behaviour when the MAV state noise was adjusted, and the behaviour is

only affected by changes in the image coordinate noise (Appendix P). It was assumed that, due to the MAV states being independent to the object states, the effect of the MAV state noise is considered irrelevant to the object state estimation, especially with the Jacobian matrix of the measurement model with respect to the object state (i.e. $\frac{\partial h}{\partial x}$); if so, with the noisy MAV state, the expected image coordinate of a known coordinate P by the filter, would differ with the true image coordinate of point P as shown in Figure 16. This would cause error in the position estimate even with image coordinate measurement with a good accuracy. To compensate such error, it was noted to adjust ϵ_{max_error} parameter according to the MAV state error.

$$\boldsymbol{R = diag}\left\{(\epsilon_{max\_error}/3)^2{}_{(2\times1)}, (p_{max\_error_{MAV}}/3)^2{}_{(3\times1)}, (\Theta_{max\_error_{MAV}}/3)^2{}_{(3\times1)}\right\} \quad (42)$$

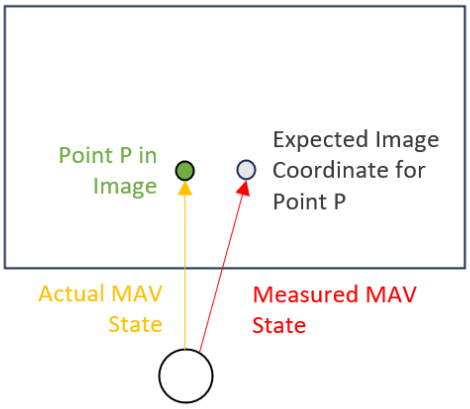


Figure 16. Effect of MAV State Error to the Expected Image Coordinate of Point P

The filter parameter was adjusted to expect gaussian noise in measurement with maximum of 2 m and 0.2^c (~ 10^o) of error in all dimensions of MAV position and orientation respectively. There is no noise in the MAV states of the dataset, although, the noises were to be added as a in the later part of the project to test the final estimation system. The centre of the object bounding box were measured with the error of maximum of 32 pixel (calculated by the measured coordinate subtracted by the expected coordinate) without added noise to the MAV states. The noise parameter for the image coordinate were decided to be set as 64 pixels (i.e. safety factor of 2) to take account of the noise in the image coordinate measurement and the noise in the MAV state.

Note that the proposed method for the optical flow measurement, flow estimation from current versus previous position in image, may not be accurate due to the effect shown in Figure 16. Different method to measure optical flow must be investigated where it must be light in computation and involves little image processing, if possible, to prevent a significant decrease in the system update rate.

3.3. Dataset Extraction

Once all the dataset issue were resolved, there were little project time remaining to achieve the further milestones (i.e. model application to UKF, further literature review on optical flow) as a significant time were spent on the re-modelling (before recognising the dataset issue). It was decided for the remaining milestones to be put to the future work. The project moved directly to extracting dataset for the result analysis. The project aimed to extract the dataset from 8 scenarios (Figure 17). Scenario 0 and 1 has a stationary vehicle with MAV taking linear/circular motion respectively, Scenario 2, 3 and 4 has the vehicle in linear motion with MAV moving parallel in the same (2) and opposite (3) directions, and perpendicular (4) to the vehicle, and Scenario 5, 6 and 7 has vehicle circling around with MAV approaching vehicle with linear motion (5) and circling directly above the vehicle in the same (6) and opposite (7) direction.

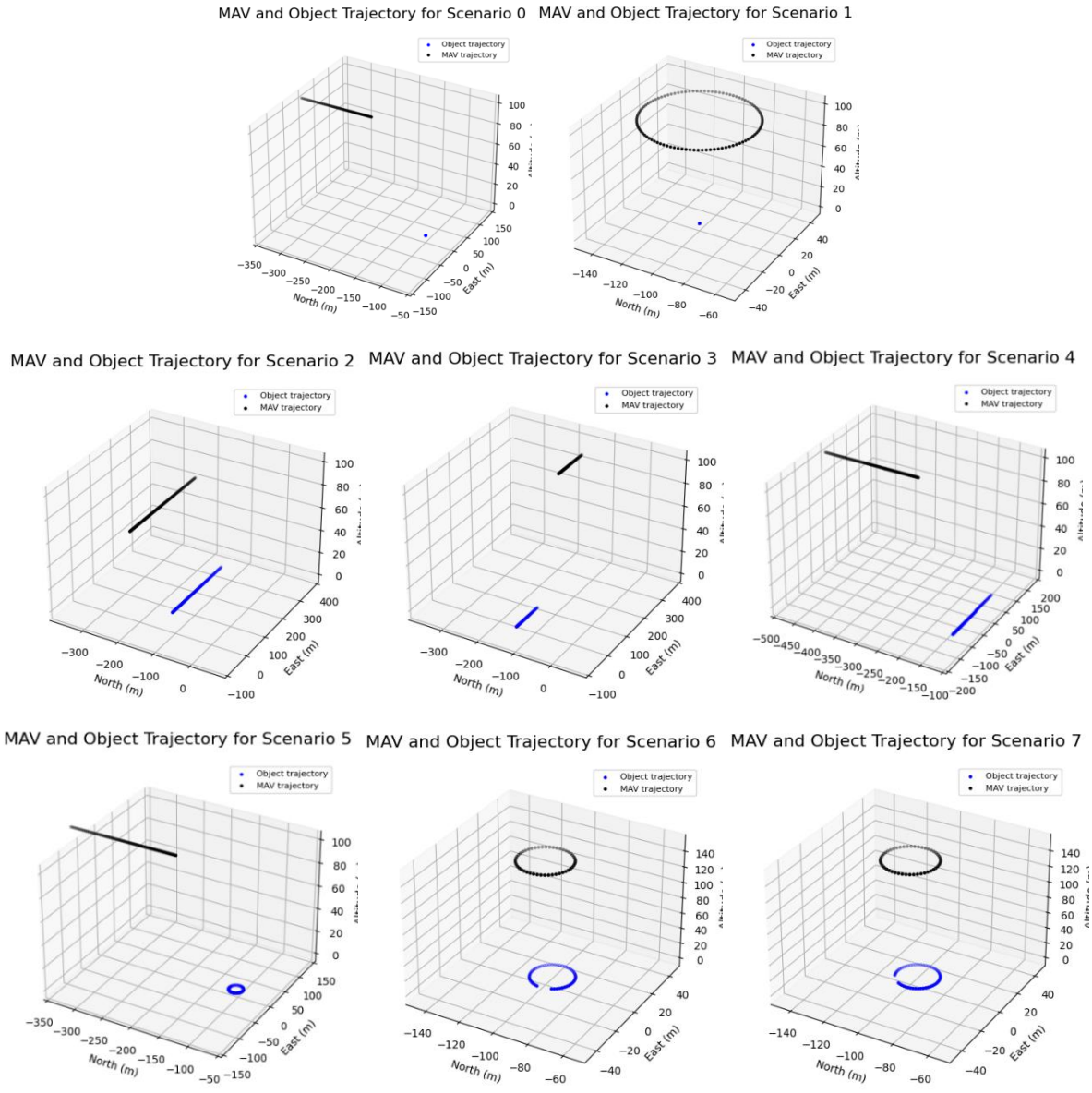


Figure 17. Trajectory Plot of MAV (black) and Vehicle (blue) in Scenario 0 to 7

It is preferrable to keep Ardupilot to simulate MAV dynamics for the 8 scenarios, although, due to time limitation, the dynamics simulator was decoupled and the Airsim camera was directly moved by calculating MAV trajectory, as described in each scenario, with added elevation and azimuth angle to the desired MAV orientation. No roll or pitch were added to the MAV orientation for the scenario with linear MAV trajectories (i.e. all scenario except 1, 6 and 7). Note that those scenarios with circling MAV may not be appropriate to be tested in practice with strapdown camera at a certain elevation and azimuth to the MAV as the ground vehicle would not appear in the image with a certain range of angle (as a MAV must tilt to take circular trajecotry); those scenarios are purely for testing purpose in simulation. All scenarios were simulated with the same camera parameters with 1224 x 1024 image size and 45.4 degrees of field-of-view. The azimuth angle to the MAV was kept 0^o, although, the elevation angle of -25^o was used for the scenarios with linear MAV trajectory and -85^0 was used for those scenarios with the circular MAV trajectory; the vehicle was not able to be spotted in the image without the change in elevation.

With the decoupled Ardupilot, a different method must be used to obtain the timestamp. Airsim does not provide any simulation time, although it has a function to run the simulation only for a certain amount of time; using this function, the dataset was extracted at 0.1 seconds of interval (i.e. 10 Hz). In addition, it was noted that the OpenCV tracker failed to track a moving vehicle due to the high noise in the image background when the vehicle was simulated in AirsimNH environment; when the same script was used with Blocks environment of Airsim, the tracker was able to track the vehicle. For this reason, the dataset from the 8 scenarios was extracted from Blocks environment. Refer Appendix Q for example images from AirsimNH and Blocks environment.

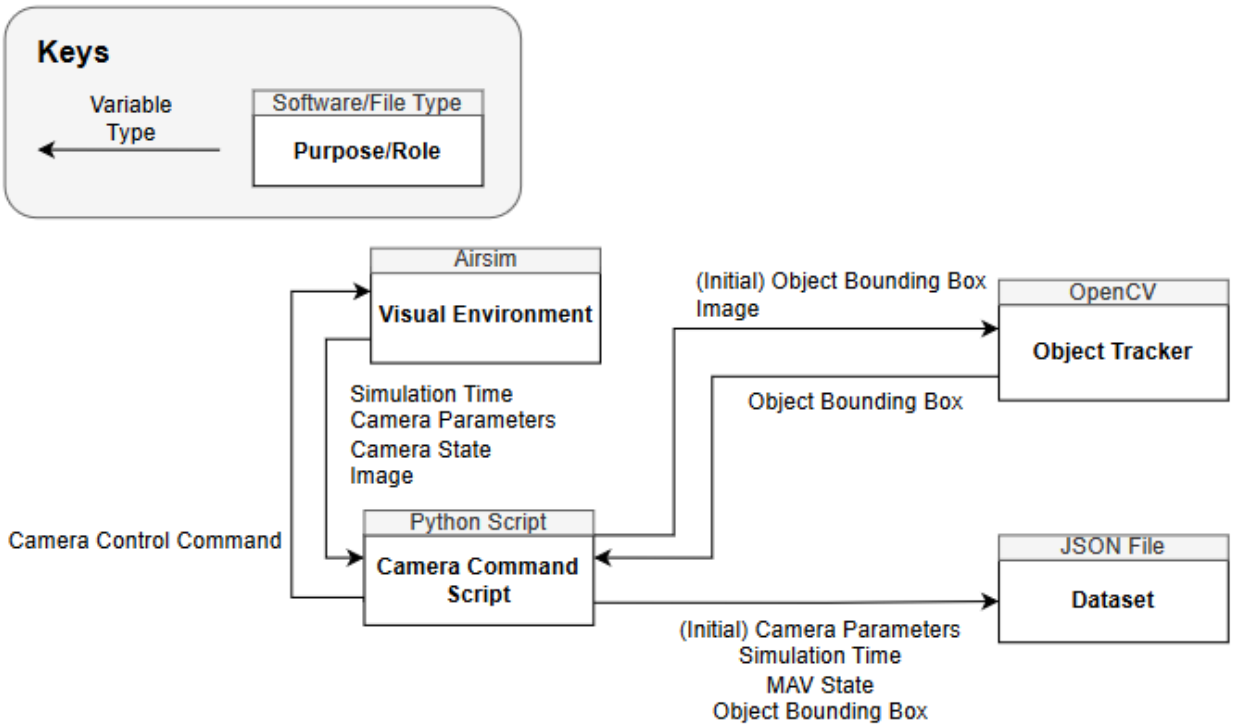


Figure 18. Flow Diagram for Dataset Extraction

The MAV Command script of Figure 11 were modified to extract dataset as shown in Figure 18 where 8 different scripts were created to simulate each scenario. The script was run for 100 times to extract 100 datasets for each scenario. It was noted that the memory space taken by Airsim increased significantly and eventually crashed after running 3-4 runs by resetting the camera and vehicle pose, which did not occur when Airsim was restarted every run. A shell script was created to automatically open Airsim, run the camera command script to extract single dataset, and close Airsim for 100 times (Appendix R).

The simulated vehicle in Airsim is possible to take up to 20 ms^{-2} of acceleration during the circular motion. The maximum acceleration for initial state covariance was updated to 20 ms^{-2} and the acceleration noise (**q** of Eq. (38)) was updated to 0.2 ms^{-2} for slow filter convergence. Note that the state noise parameter can be increased to achieve a faster convergence. The 800 datasets were processed through the filter.

4. RESULTS AND DISCUSSION

The result from the final system is shown in this section. Note that all the plots from unfiltered estimation are included in appendices. The trajectory plot of each scenario is shown in Figure 17 Refer Appendix S for the sample images from those scenarios.

The final system is the baseline EKF with the state vector and the measurement vector as shown in Eq. (43) and (44) respectively, where the state transition model and measurement model equation are shown in Eq. (45) and (46). The initial filter state and covariance are shown in Eq. (47) and (48). The state and measurement noise matrix are constant as shown in Eq. (49) and (50) respectively.

$$\hat{\boldsymbol{x}} = \left\{\hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\boldsymbol{v}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)},\ \hat{\boldsymbol{a}}^{\boldsymbol{i}}_{\boldsymbol{OBJ}(3\times1)}\right\}^{\boldsymbol{T}} \quad \textbf{(43)}$$

$$\mathbf{z} = \left\{\epsilon_{(2x1)},\ \hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{MAV}(3\times1)},\ \hat{\boldsymbol{\Theta}}^{\boldsymbol{i}}_{\boldsymbol{MAV}(3\times1)}\right\}^{\boldsymbol{T}} \quad \textbf{(44)}$$

$$\boldsymbol{P}^{\boldsymbol{i}}_{\boldsymbol{obj}} = \boldsymbol{P}^{\boldsymbol{i}}_{\boldsymbol{MAV}} + \frac{-(\boldsymbol{k}^{\boldsymbol{i}}\cdot\boldsymbol{P}^{\boldsymbol{i}}_{\boldsymbol{MAV}})}{\boldsymbol{k}^{\boldsymbol{i}}\cdot\boldsymbol{R}^{\boldsymbol{v}}_{\boldsymbol{c}}\widetilde{\boldsymbol{\ell^c}}}\left(\boldsymbol{R}^{\boldsymbol{v}}_{\boldsymbol{c}}\widetilde{\boldsymbol{\ell^c}}\right) \quad \textbf{(45)}$$

$$\mathbf{p_{t+1}} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{a_t}\boldsymbol{T}^{\mathbf{2}} + \mathbf{v_t}\boldsymbol{T} + \mathbf{p_t} \quad \textbf{(46)}$$

$$\widehat{\boldsymbol{x_0}} = \left\{\widehat{\boldsymbol{p_0}}_{\boldsymbol{OBJ}_{(3\times1)}}, \mathbf{0}_{(3\times1)}, \mathbf{0}_{(3\times1)}\right\}^{\boldsymbol{T}} \quad \textbf{(47)}$$

$$\widehat{\boldsymbol{P_0}} = \boldsymbol{diag}\left\{\left(\frac{\boldsymbol{p}_{\max_{\boldsymbol{error}}\boldsymbol{OBJ}}}{\mathbf{3}}\right)^{\mathbf{2}}_{(3\times1)}, \left(\frac{\boldsymbol{v}_{\boldsymbol{max}_{\boldsymbol{OBJ}}}}{\mathbf{3}}\right)^{\mathbf{2}}_{(3\times1)}, \left(\frac{\boldsymbol{a}_{\boldsymbol{max}_{\boldsymbol{OBJ}}}}{\mathbf{3}}\right)^{\mathbf{2}}_{(3\times1)}\right\},$$

$$\boldsymbol{p}_{\max_{\boldsymbol{error}}\boldsymbol{OBJ}} = \mathbf{10}\ \boldsymbol{m},\ \boldsymbol{v}_{\boldsymbol{max}_{\boldsymbol{OBJ}}} = \mathbf{30}\ \boldsymbol{ms^{-1}},\ \boldsymbol{a}_{\boldsymbol{max}_{\boldsymbol{OBJ}}} = \mathbf{20}\ \boldsymbol{ms^{-2}} \quad \textbf{(48)}$$

$$\boldsymbol{Q_t} = \begin{bmatrix} \frac{\boldsymbol{dt^5}}{\mathbf{20}} & \frac{\boldsymbol{dt^4}}{\mathbf{8}} & \frac{\boldsymbol{dt^3}}{\mathbf{6}} \\ \frac{\boldsymbol{dt^4}}{\mathbf{8}} & \frac{\boldsymbol{dt^3}}{\mathbf{3}} & \frac{\boldsymbol{dt^2}}{\mathbf{2}} \\ \frac{\boldsymbol{dt^3}}{\mathbf{6}} & \frac{\boldsymbol{dt^2}}{\mathbf{2}} & \boldsymbol{dt} \end{bmatrix} \cdot \boldsymbol{q^2},\ \boldsymbol{q} = \mathbf{0.2},\ \boldsymbol{dt} = \mathbf{0.1} \quad \textbf{(49)}$$

$$\boldsymbol{R} = \boldsymbol{diag}\left\{(\boldsymbol{\epsilon}_{\boldsymbol{max\_error}}/\mathbf{3})^{\mathbf{2}}_{(2\times1)}, (\boldsymbol{p}_{\boldsymbol{max\_error}_{\boldsymbol{MAV}}}/\mathbf{3})^{\mathbf{2}}_{(3\times1)}, (\boldsymbol{\Theta}_{\boldsymbol{max\_error}_{\boldsymbol{MAV}}}/\mathbf{3})^{\mathbf{2}}_{(3\times1)}\right\},$$

$$\boldsymbol{\epsilon}_{\boldsymbol{max\_error}} = \mathbf{64}\ \boldsymbol{px},\ \boldsymbol{p}_{\boldsymbol{max\_error}_{\boldsymbol{MAV}}} = \mathbf{2}\ \boldsymbol{m},\ \boldsymbol{\Theta}_{\boldsymbol{max\_error}_{\boldsymbol{MAV}}} = \mathbf{0.2}^{\boldsymbol{c}} \quad \textbf{(50)}$$

It was found that the dataset from 800 runs (100 runs each for 8 scenarios) contains 51189 timesteps of data which was processed by the system in 93.73 seconds including opening and closing the iterated log files. This gives an average of 546 Hz for the update rate purely from the system filter which suggests that the system is applicable for real-time application. Note that the current system does not involve any image processing and the filter update rate may significantly decrease when images are read/written during the process. Note that the measurement update rate for all the dataset is 10 Hz unless stated.

The entire dataset was taken with the MAV on altitude of 100 m or higher, with vehicle at 0 m altitude, to test the estimation on the object at a range further than 100 m. This is to examine whether the system is applicable at a long distance.

The dataset was obtained with no noise in MAV and camera states; the only noise in the dataset is in the object bounding box centre where additional noise can be added as desired before feeding the dataset into the filter. The exact distribution of the object bounding box noise could not be obtained, although, it was noted that the noise presented at most 10 m of error in calculating the vehicle position (Refer Appendix T). Furthermore, the noise distribution in the object bounding box is not gaussian and the average of the unfiltered position error is biased to non-zero value where the effect is less presented in the velocity and acceleration estimation (Appendix T). The focus of the analysis will be on the accuracy in object velocity estimation.

Note that the time duration varies by the scenario due the object escaping the camera field-of-view at different timing. For error plots (e.g. Figure 19), ‘true’ plots were inserted to visualise the zero-error line.

4.1. No MAV state noise

4.1.1. Results

Refer Appendix T and Appendix U for the state estimate plot of the entire dataset by system with no filter and with the baseline EKF respectively. Appendix V shows the average error from the baseline EKF in all scenarios using two different maximum object bounding box error ϵ_{max_error} parameter and Appendix W shows the result from the filter with low and high q (i.e. 0.2 and 1000) values.

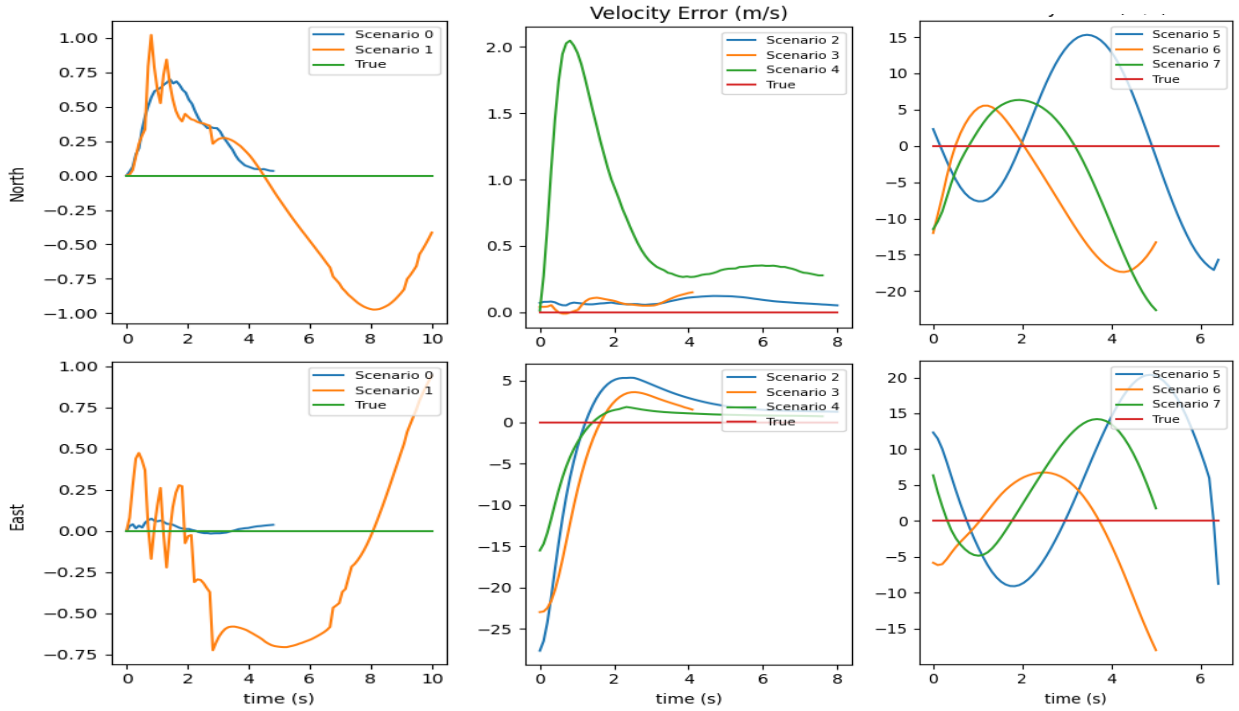


Figure 19. Baseline EKF velocity estimation error under no MAV state noise

4.1.2. Discussion

Despite having no measurement directly available for the object velocity and acceleration, the system was able to filter the noises in the velocity and acceleration estimate from differentiated position. It was assumed that the position error bias presents less effect to the velocity estimation as long as the object bounding box centre is at a constant offset from the vehicle because the velocity and acceleration information only depends on the change in the object position.

The system was able to estimate within 1 m/s of velocity estimation error for scenarios with stationary vehicle which was significantly less error than those from the other scenarios (Figure 19). This is assumed that it is because that the initial assumption in the object motion is that the object will be stationary. The filter does not receive a measurement on the object velocity and acceleration and requires multiple measurements to converge the estimation close to the ground truth. The convergence of those state would be much faster with initial estimate close to the ground truth.

The error in the East position estimate is much less than the North position for Scenario 0. A wider field-of view (FOV) is available in the East direction than the North direction, as shown in Figure 20, where the East direction is left to right and the North direction is bottom to top direction. In addition, the space covered by a single pixel in the image differs by the angle of camera view and the pixel location. Figure 21 shows that a ground represented by the grid on the left is transformed similar to the grid on the right with the given camera configuration which is a similar configuration used for the dataset; the grid with the area closer to the camera is projected larger than the area further from the camera where the effect is larger in the longitudinal direction. This indicates that, when an error in the object bounding box centre occurs equally in horizontal and vertical direction of image, the resulting position error in longitudinal direction could be larger than the error in lateral direction. It was assumed that the filter produced less error in East direction (lateral direction in image) in Scenario 0 and 4 by this reason. Furthermore, it was also noted that the geolocation model with pinhole camera is sensitive to the height of the measurement plane in general due to the homogenous coordinates (Mundy & Zisserman, 1992). Figure 22 shows the position error from the height difference in the geolocation model with flat-surface assumption and the actual height of the measurement plane for the project (i.e. visual object centre). The position error caused by the effect varies by the line-of-sight angle which is possible to add noise in the velocity and acceleration estimate. In some cases, it is crucial to provide a predicted object height information to the algorithm to remove such error (Cai, et al., 2022). Thus, it is recommended to provide the object height to the system, if known, for a better result.

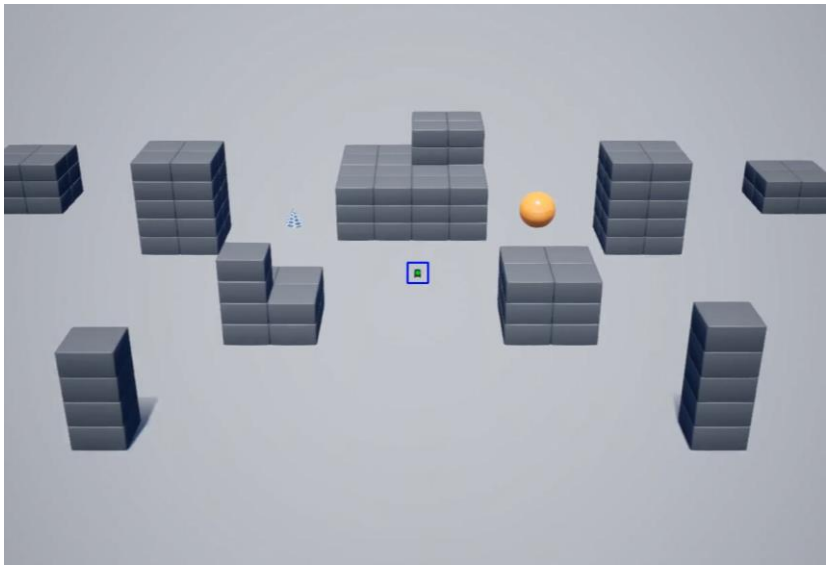


Figure 20. Sample Camera Image from Scenario 0

Figure removed due to copyright restriction

Figure 21. Deformation of Grid Projection in Camera Image (Gunawan, et al., 2019)

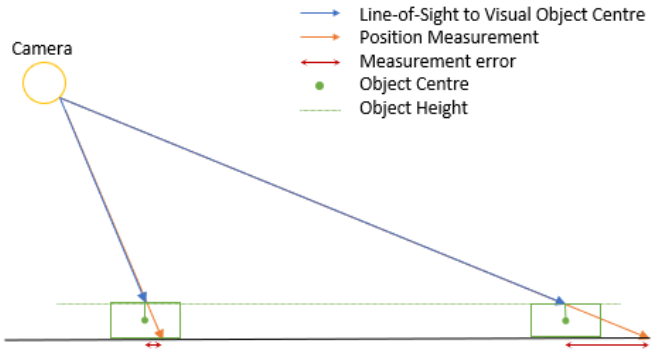


Figure 22. Position Error with Correct Measurement

For the scenarios with vehicle in linear motion, the vehicle was simulated in 30 m/s of speed (i.e. assumed maximum speed) in West to East direction. The velocity estimation successfully converged after minimum of 30 to maximum of 60 measurement updates (i.e. 3 and 6 seconds), as shown in Figure 19. The fastest convergence on East velocity estimate was achieved in Scenario 4, MAV trajectory perpendicular to the vehicle, and the slowest convergence with Scenario 2, MAV following the vehicle. The system is intendedly set with low q for strong filtering, although, note that a faster convergence can be achieved by increasing q. Appendix W shows that the filter converges with 15 measurements by increasing q to 1000. The expectation was that the system would converge better in Scenario 2 than in Scenario 3 (MAV moving parallel in opposite direction) as it observes the vehicle for longer duration. It also could use the information that the vehicle is moving at a similar speed and direction to MAV. However, it was noticed that the vehicle was simulated at a higher speed in Scenario 4 compared to Scenario 2 and 3. Due to the low flexibility in vehicle control in Airsim, the vehicle was simulated in different speeds. The convergence is expected to be similar for vehicle in the same speed in all scenarios. For velocity estimation in North component, the estimate temporarily diverges in the early few measurements in Scenario 4 where the velocity converged immediately to near 0 for Scenario 2 and 3. This was assumed that it is because the North axis was the longitudinal direction to the camera in Scenario 4 unlike Scenario 2 and 3 (i.e. sensitivity in longitudinal position measurement). In general, the system was able to converge to near 0 error for all scenarios with vehicle in linear motion (Figure 19).

Scenarios with vehicle in circular motion returned the most unexpected result from the system; the estimation error does not converge and oscillates (Appendix U) where the unfiltered state is at

better accuracy (Appendix T). The oscillation is expected to be due to filter lag as the true North and East component of vehicle acceleration also oscillates. This was assumed that the state model used for the system only considers the translational acceleration and does not consider any angular acceleration. The dataset also contains a little noise. Although, note that the state model was able to estimate the state of manoeuvring object with giving higher state noise q (Appendix W). Thus, with the current system, it is suggested to increase the q values of EKF when the system is used for object with angular or rapidly changing acceleration. In addition, it was decided to modify and test the state transition model to represent the velocity and acceleration by magnitude and direction with respect to the vehicle heading. This is to remove the oscillating error in scenarios 5, 6 and 7 for the system to have a better performance on a similar scenario.

The system was also tested with the noise parameter of maximum object bounding box changed from 64 to 32 px (for measurement noise as in Eq. (50)) as there is yet no noise in MAV states to see if the filter lag in Scenario 2, 3 and 4 can be decreased (Appendix V). The system performed similar to how it performed with the initial parameters, although, the East velocity and acceleration converged faster with the changed parameter as it provides less filtering with lower noise parameters and causes less filter lag.

4.2. With MAV state noise

4.2.1. Results

Refer Appendix X for the average comparison plots of all scenarios. Note that 'No Noise' indicates there is no noise added to the MAV states (position and orientation), although, the dataset still has noise in the object bounding box.

For summary of the result, the estimation in all object states were significantly affected by noise in MAV orientation whereas much less effect is applied to the estimation with the noise in MAV position.

4.2.2. Discussion

The estimation error increased with noise added to both MAV position and orientation, as per Eq. (50), where the estimation error significantly increased with the orientation noise compared to the position noise. This is because that the geolocation with the pin-hole camera model is highly sensitive to the MAV orientation noise, especially at a further distance to the object, due to the line-of-sight error as shown in Figure 22 (Barber, et al., 2006). In addition, the current system does not account the MAV state noise significantly as they are independent from the object states and the only measurement dependent on the object states is the object bounding box, although, the true

object bounding box may not return true object position due to noisy MAV states (Figure 16). This also indicates the optical flow estimate by method, as shown in Figure 6, would not be valid with noisy MAV orientation as it is highly likely that the expected previous object position in the image would be incorrect. It was also expected that the approach would significantly decrease the filter update rate, without using the optical flow estimation method, due to requiring image processing. It was decided to reserve the optical flow approach as it is expected the approach would provide minimal benefit to the current system with adding a list of limitations.

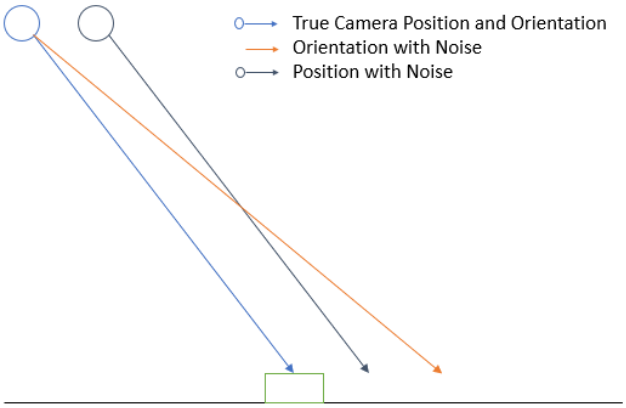


Figure 23. Resultant Position Error from MAV Position vs Orientation Noise

Lastly, it is also possible that the linear approximation of EKF is inappropriate for the given measurement model. Kim and Ristic (2023) showed that the distribution estimate of polar-to-Cartesian coordinates by linear approximation and unscented transformation. These are the methods used in EKF and UKF respectively for estimation of gaussian distribution through non-linear transform. Despite the polar-to-Cartesian transform involving less trigonometry terms compared to the system measurement model for the project, the distribution estimate from linear approximation is far less accurate than the estimate from unscented transform (Figure 24). Thus, as proposed, it is suggested to implement the same model to UKF for the future work to confirm whether the system performance can be improved.

Figure removed due to copyright restriction

Figure 24. Distribution estimation by linear approximation (left) vs unscented transform (right) (Kim & Ristic, 2023)

4.3. With lag in object bounding box measurement

4.3.1. Results

Refer Appendix Y for the average comparison plots of all scenarios. The result plots shows that the system performance significantly decreased in Scenario 3, 4 and 5 followed by Scenario 0, 2, 6 and 7 and almost no impact is given to Scenario 1 with the 0.1 seconds of lag in the object bounding box. The system showed a similar performance with and without the lag in the object bounding box for scenario 1. A large lag was applied to the data to clearly view its impact to the estimation performance of the system, although, note that the sensor lag in practice can be much less than 0.1 seconds.

4.3.2. Discussion

The main difference between Scenario 1 and the other scenarios is that the centre of the object bounding box does not differ significantly over the time in Scenario 1 as much as in other scenarios (Appendix S). The object bounding box measurement varies less when there is less change in the line-of-sight to the object from the camera, meaning line-of-sight rate close to 0. This reason explains the reason that the system performance is worst in Scenario 3 (i.e. MAV approaching in linear motion to the vehicle in linear motion from the opposite side) which is the scenario with the most rapidly changing line-of-sight. It was assumed that the lag in the other measurement would provide similar impact. The system is expected to be affected in scenarios with variation in line-of-sight for MAV orientation lag, similar to the object bounding box lag. It is expected the scenarios with variation in relative position of MAV to the object will affect the system with MAV position lag; the effect is expected to be larger at the scenarios with rapid change in MAV altitude. It is used for calculating the range to the object and the system is sensitive any noise in the line-of-sight and the range measurement.

4.4. Decrease in update rate

4.4.1. Results

Refer Appendix Z for the average comparison plots of all scenarios. In general, the estimation from 2 Hz measurement was lagging, at maximum of 0.5 seconds compared to the estimation made by 10 Hz measurement in all scenarios. In addition, due to the lag in the estimation, the system required a slightly longer time to converge for system with 2 Hz of measurement update rate compared to 10 Hz of measurement update rate.

4.4.2. Discussion

The system required a longer time to converge due to the lag in the estimation, although, it also means that the system was able to converge with less measurement steps. This was assumed that it is due to the system model to estimate the velocity and acceleration, which related to the rate of change in position and depends on the time scale. Furthermore, the impact of decrease in update rate is not as significant compared to the impact of the measurement lag. This was assumed that, because the velocity and acceleration measurement is sensitive to the time, the measurement lag causes error in estimation due to the inconsistency in time of the measurements where the decrease in update rate, although it provides less information, it returns the measurement with little error in time. The low update rate would not cause a significant error as it is expected the estimation would still converge, as opposed to the error caused by the measurement lag.

4.5. Additional Analysis Plan

The following list shows the variables that were planned to be measured for further analysis of the system performance as a stress test. However, it was not able to be achieved within the project time and are included for future work.

- Maximum q of the Q matrix with fixed R without estimation divergence
- Maximum prediction-step to update-step ratio without estimation divergence
- Maximum no. of timesteps with no measurement (i.e. object out of camera field-of-view) without estimation divergence

In addition, it was noticed that EKF may unexpectedly diverge when the P matrix becomes asymmetric which can occur due to the numeric error. Although, the Joseph formular can prevent the asymmetry in covariance matrix (Bucy & Joseph, 2005). Despite having no divergence observed from the dataset from the analysis yet, the Joseph formular is to be reviewed and applied to the current system to examine the effect, to ensure the system stability in practical applications.

CONCLUSIONS

In summary, the baseline EKF system was able to estimate the object velocity with little noise in the object bounding box when the object at more than 100 m range. The system was able to filter the object position information to velocity and acceleration estimates with approximately 546 Hz of filter update rate. Note that the system behaviour differs depending on different filter parameters (i.e. Q and R matrices); a fine-tuning of the parameters is required for the system to perform at optimal condition for each application. The system was able to converge to a 0 m/s of velocity error with no noise in MAV states in all tested scenarios where the speed of convergence varies by the Q matrix. It is recommended to provide the object height to the system for better accuracy if known. The system is found to be sensitive to the line-of-sight and range error from the camera to the object due to the measurement model. Thus, the system performance significantly reduces with noise in MAV orientation followed by the MAV position which does not satisfy the project aim to have a system robust to measurement noises. It was assumed that UKF may perform better under such condition as EKF may have deformed the non-linear measurement model significantly due to the linearisation. The limitation of the current system is that it is only applicable to an object on flat surface with known altitude, known camera parameters where there is no noise or lag in MAV orientation. The system also cannot handle a significant lag in the object bounding box. The noise and lag in the MAV position measurement can be tolerated, although it still lowers the system performance. There is a delay in estimation with a decrease in the measurement update rate which is expected to be tolerable by adjusting Q matrices. From the observations, it is concluded that the system satisfies all project aims, except to be robust under measurement noise. The system is recommended to be improved as described in the following section.

FUTURE WORK

Assuming the continuation of the project, the current model is to be modified to have a velocity and acceleration represented with respect to the vehicle heading. The modified system is to be implemented to UKF and tested with a new dataset with the camera script coupled with the MAV dynamics simulator for more realistic movement of strapdown camera on MAV. The application of the Joseph form is to be investigated if it is later decided to use EKF instead of UKF once the result is obtained. The same analysis, as done in this report, is to be done with the modified system. The system is also to be tested to find the maximum q value for Q matrix, maximum prediction-step to update-step rate and maximum timesteps with no observation before the estimation diverges. Lastly, the system is to be tested on a drone hardware to be observed on a practical basis.

REFERENCES

Ait-Aider, O., Andreff, N., Lavest, J. M. & Martinet, P., 2006. Simultaneous Object Pose and Velocity Computation Using a Single View from a Rolling Shutter Camera. *European Conference on Computer Vision,* Volume 3952, pp. 56-68.

Andersh, J., Cherian, A., Metter, B. & Papanikolopoulos, N., 2015. A vision based ensemble approach to velocity estimation for miniature rotorcraft. *Autonomous Robots,* Volume 123-138, p. 39.

Ashraf, S. et al., 2018. A low-cost solution for unmanned aerial vehicle navigation in a global positioning system-denied environment. *International Journal of Distributed Sensor Networks,* 14(6), pp. 1-17.

Barber, D. B., Redding, J., McLain, T. & Beard, R., 2006. Vision-based Target Geo-location using a Fixed-wing Miniature Air Vehicle. *Journal of Intelligent & Robotic Systems,* 47(4), pp. 361-382.

Beard, R. W. & McLain, T. W., 2012. *Small Unmanned Aircraft: Theory and Practice.* Princeton, Princeton University Press.

Beauchemin, S. & Barron, J., 1995. The computation of optical flow. *ACM Computing Surveys,* 27(3), pp. 433-466.

Benini, A., Mancini, A. & Longhi, S., 2012. An IMU/UWB/Vision-based Extended Kalman Filter for Mini-UAV Localization in Indoor Environment using 802.15.4a Wireless Sensor Network. *Journal of Intelligent & Robotic Systems,* 70(1-4).

Bucy, R. S. & Joseph, P. D., 2005. *Filtering for Stochastic Processing with Applications to Guidance.* 2nd ed. s.l.:AMS Chelsea Publishing.

Cai, Y. et al., 2022. Review of Target Geo-Location Algorithms for Aerial Remote Sensing Cameras without Control Points. *Applied Sciences,* 12(24).

Cha, Y.-J. & J. G. Chen, O. B., 2016. Output-only computer vision based damage detection using phase-based optical flow and unscented Kalman filters. *Engineering Structures,* Volume 132, pp. 300-313.

Chun, J.-B., Jung, H. & Kyung, C.-M., 2008. Suppressing rolling-shutter distortion of CMOS image sensors by motion vector detection. *IEEE Transactions on Consumer Electronics,* 54(4), pp. 1479-1487.

Del-Moral, P., 1996. Nonlinear Filtering: Interacting Particle Solution. *Markov Processes and Related Fields,* 2(4), pp. 555-580.

DSTL, S., 2021. *Transition Models - Stone Soup 1.1 documentation.* [Online] Available at: https://stonesoup.readthedocs.io/en/v1.1/stonesoup.models.transition.html#stonesoup.models.transition.linear.ConstantAcceleration
[Accessed 18 October 2023].

Engel, J., Sturm, J. & Cremers, D., 2014. Scale-aware navigation of a low-cost quadcopter with a monocular camera. *Robotics and Autonomous Systems,* 62(11), pp. 1646-1656.

Fan, B., Wang, K., Dai, Y. & He, M., 2021. RS-DPSNet: Deep Plan Sweek Network for Rolling Shutter Stereo Images. *IEEE Signal Processing Letters,* Volume 28, pp. 1550-1554.

Fang, Q. & Huang, S. X., 2013. UKF for Integrated Vision and Inertial Sensors Based on Three-View Geometry. *IEEE Sensors Journal,* 13(7).

Fu, Q., Quan, Q. & Cai, K.-Y., 2017. Robust Pose Estimation for Multirotor UAVs Using Off-Board Monocular Vision. *IEEE Transactions on Industrial Electronics,* 64(10), pp. 7942-9851.

Garcia, R. V., Pardal, P. C. P. M., Kuga, H. K. & Zanardi, M. C., 2019. Nonlinar filtering for sequential spacecraft attitude estimation with real data: Cubature Kalman Filter, Unscented Kalman Filter and Extended Kalman Filter. *Advances in Space Research,* 63(2), pp. 1038-1050.

Grau, O. & Pansiot, J., 2012. Motion and velocity estimation of rolling shutter cameras. *Proceedings of the 9th European Conference on Visual Media Production,* pp. 94-98.

Gunawan, A. S. A., Tanjung, D. A. & Gunawan, F. E., 2019. Detection of Vehicle Position and Speed using Camera Calibration and Image Projection Methods. *4th International Conference on Computer Science and Computational Inteliigence,* Volume 157, pp. 255-265.

Guo, H. & Li, Z., 2022. *Object Detection and Ranging System Based on Binocular VIsion.* s.l., IOP Publishing Ltd.

He, H., Li, Y. & Tan, J., 2017. Relative motion estimation using visual-inertial optical flow. *Autonomous Robots,* 42(3), pp. 615-629.

Ho, H. W., de-Croon, G. C. & Chu, Q., 2017. Distance and velocity estimation using optical flow from a monocular camera. *International Journal of Micro Air Vehicles,* 9(3), pp. 198-209.

Javanmardi, E., Gu, Y., Javanmardi, M. & Kamijo, S., 2019. Autonomous vehicle self-localization based on abstract map and multi-channel LiDAR in urban area. *IATSS Research,* 43(1), pp. 1-13.

Jeon, H., Min, J., Bang, H. & Youn, W., 2022. Quaternion-Based Iterative Extended Kalman Filter for Sensor Fusion of Vision Sensor and IMU in 6-DOF Displacement Monitoring. *IEEE Sensors Journal,* 22(23).

Julier, S. J. & Uhlmann, J. K., 2004. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE,* 92(3), pp. 401-422.

Kalman, R. E., 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering,* Volume 82, pp. 35-45.

Kim, D. Y. & Ristic, B., 2023. *Introduction to Tracking & Sensor Fusion.* Adelaide: IEEE AESS Short Course.

Kim, N. et al., 2020. Object distance estimation using a single image taken from a moving rolling shutter camera. *Sensors,* 20(14).

Kim, Y. V., 2023. *Kalman Filter: Engineering Applications.* London, UK, IntechOpen.

Krotkov, E., Henriksen, K. & Kories, R., 1990. Stereo ranging with verging cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 12(12), pp. 1200-1205.

Kubelka, V. et al., 2015. Robust Data Fusion of Multimodal Sensory Information for Mobile Robots. *Journal of Field Robotics,* 32(4), pp. 447-473.

Lao, Y., Ait-Aider, O. & Araujo, H., 2018. Robustified Structure from Motion with rolling-shutter camera using straightness constraint. *Pattern Recognition Letters,* Volume 111, pp. 1-8.

Lee, J.-K. & Yoon, K.-J., 2018. Temporally Consistent Road Surface Profile Estimation Using Stereo Vision. *IEEE Transactions on Intelligent Transportation Systems,* 19(5), pp. 1618-1628.

Lin, H.-Y., 2005. Vehicle speed detection and identification from a single motion blurred image. *Seventh IEEE Workshops on Applications of Computer Vision,* Volume 1, pp. 461-467.

Lin, H.-Y. & Li, K.-J., 2004. Motion blur removal and its application to vehicle speed detection. *International Conference on Image Processing,* Volume 5, pp. 3407-3410.

Liu, L., Xi, Z. & Sun, Q., 2019. Multi-vision tracking and collaobration based on spatial particle. *Journal of Visual Communication and Image Representation,* Volume 59, pp. 316-326.

Lucas & Kanade, 1981. An Iterative Image Registration Technique with an Application to Stereo Vision. *Proceedings of Imaging Understanding Workshop,* pp. 121-130.

Maley, J. M., 2015. *Line of Sight Rate Estimation for Guided Projectiles with Strapdown Seekers.* Kissimmee, Florida, AIAA Guidance, Navigation, and Control Conference.

Ma, W., Li, W. & Cao, P., 2019. Ranging Method of Binocular Stereo Vision Based on Random Ferns and NCC Template Matching. *IOP Conference Series: Earth and Environmental Science,* 252(2).

McElhoe, B. A., 1966. An assessment of the navigation and course corrections for a manned flyby of mars or venus. *IEEE Transactions on Aerospace and Electronics Systems,* AES-2(4), pp. 613-623.

McGee, L. A., Schmidt, S. F. & Smith, G. L., 1962. *APPLICATION OF STATISTICAL FILTER THEORY TO THE OPTIMAL ESTIMATION OF POSITION AND VELOCITY ON BOARD A CIRCUMLUNAR VEHICLE,* Washington: NASA.

McGuire, K. et al., 2017. Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomus Pocket Drone. *IEEE Robotics and Automation Letters,* 2(2), pp. 1070-1076.

Mohammadi, J., Akbari, R. & Ba-haghighat, M. K., 2010. Vehicle speed estimation based on the image motion blur using RADON transform. *2nd International Conference on Signal Processing Systems,* Volume 1, pp. 243-247.

Moreno, F. A., Blanco, J. L. & Gonzalez, J., 2009. Stereo vision specific models for particle filter-based SLAM. *Robotics and Autonomous Systems,* 57(9), pp. 955-970.

Mundy, J. L. & Zisserman, A., 1992. *Geometric Invariance in Computer Vision.* London: The MIT Press.

Nielsen, J. & Beard, R., 2017. Relative target estimation using a cascade of extended Kalman filters. *BYU Scholars Archive.*

Ning, M., Calderara, S. & Cucchiara, R., 2022. SeeFar: Vehicle Speed Estimation and Flow Analysis from a Moving UAV. *Image Analysis and Processing,* Volume 13233, pp. 278-289.

Pagello, E. et al., 2004. Overview of RoboCub 2003 Competition and Conferences. In: D. Polani, B. Browning, A. Bonarini & K. Yoshida, eds. *RoboCup 2003: Robot Soccer World Cup VII.* Berlin Heidelberg: Springer-Verlag, pp. 1-14.

Pan, T. et al., 2023. Monocular-Vision-Based Moving Target Geolocation Using Unmanned Aerial Vehicle. *Drones,* 7(2).

Rigatos, G. G., 2010. Extended Kalman and Particle Filtering for sensor fusion in motion control of mobile robots. *Mathematics and Computers in Simulation,* Volume 81, pp. 590-607.

Saha, S., Bambha, N. K. & Bhattacharyya, S. S., 2010. Design and implementation of embedded computer vision systems based on particle filters. *Computer Vision and Image Understanding,* 113(11), pp. 1203-1214.

Siegwart, R., Nourbakhsh, I. R. & Scaramuzza, D., 2011. *Introduction to Autonomous Mobile Robots.* 2nd ed. Cambridge: MIT Press.

Strydom, R., Thurrowgood, S., Denuelle, A. & Srinivasan, M. V., 2015. UAV Guidance: A Stereo-Based Technique for Interception of Stationary of Moving Targets. *Conference Towards Autonomous Robotics Systems,* Volume 9287, pp. 258-269.

Suddath, J. H., Kidd-III, R. H. & Reinhold, A. G., 1967. *A LINEARIZED ERROR ANALYSIS OF ONBOARD PRIMARY NAVIGATION SYSTEMS FOR THE APOLLO LUNAR MODULE,* Washington D. C.: National Aeronautics and Space Administration.

Sun, T. et al., 2015. Line-of-Sight Rate Estimation Based on UKF for Strapdown Seeker. *Mathematical Problems in Engineering,* Volume 2015.

Szeliski, R., 2011. Recognition. In: D. Gries & F. B. Schneider, eds. *Computer Vision Algorithms and Applications.* London: Springer, pp. 575-630.

Trigkakis, D., Petrakis, G., Tripolitsiotis, A. & Partsinevelos, P., 2020. Automated Geolocation in Urban Environments Using a Simple Camera-Equipped Unmanned Aerial Vehicle: A Rapid Mapping Surveying Alternative?. *International Journal of Geo-Information,* 9(7).

Ullah, I. et al., 2020. A Localization Based on Unscented Kalman Filter and Particle Filter Localization Algorithms. *IEEE Access,* Volume 8, pp. 2233-2246.

Wahrudin, A., Widyotriatmo, A. & Joelianto, E., 2019. Design of Localization System Based on Particel Filter Algorithm for Mobile Soccer Robot Using Encoders Compass, and Omnidirectional Vision Sensor. *Internetworking Indonesia Journal,* 11(2), pp. 57-64.

Wang, W., Ma, H., Wang, Y. & Fu, M., 2015. Performance analysis based on least squares and extended Kalman filter for localization of static target in wireless sensor networks. *Ad Hoc Networks,* 25(Part A), pp. 1-15.

Wu, H.-M. & Karkoub, M., 2019. Hierarchical inversion-based output tracking control for uncertain autonomous underwater vehicles using extended Kalman filter. *Asian Journal of Control,* 23(1), pp. 228-240.

Wu, H., Xiao, L. & Wei, Z., 2021. Simultaneous Video Stabilization and Rolling Shutter Removal. *IEEE Transactions on Image Processing,* Volume 30, pp. 4637-4652.

Zanetti, R. & DeMars, K. J., 2013. Joseph Formulation of Unscented and Quadrature Filters with Application to Consider States. *Journal of Guidance, Control, and Dynamics,* 36(6).

Zhang, G. J., Zhang, Y., Yao, Y. & Ma, K. M., 2005. LINE OF SIGHT RATE ESTIMATION OF STRAPDOWN IMAGING GUIDANCE SYSTEM BASED ON UNSCENTED KALMAN FILTER. *Proceedings of the Fourth international Conference on Machine Learning and Cybernetics,* Volume 3, pp. 1574-1578.

Zhang, Y. C., Li, J. J. & Li, H. Y., 2010. Line of Sight Rate Estimation of Strapdown Imaging Seeker Based on Particle Filter. *3rd International Symposium on Systems and Control in Aeronautics and Astronautics,* pp. 191-195.

Zhong, S. & Chirarattananon, P., 2020. Direct Visual-Inertial Ego-Motion Estimation via Iterated Extended Kalman Filter. *IEEE Robotics and Automation Letters,* Volume 5, pp. 1476-1483.

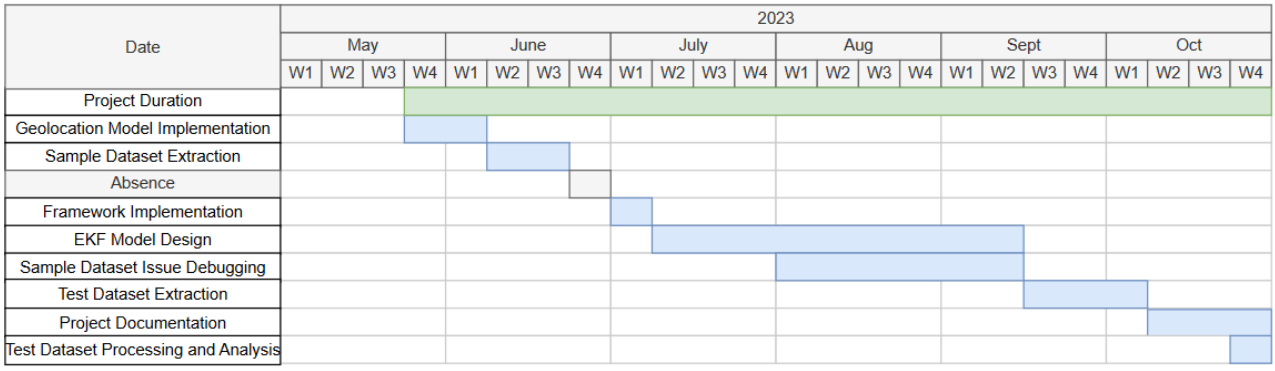
APPENDICES

Appendix A. Python Package Information

Name	Version	Related Software
airsim	1.8.1	Airsim
dronekit	2.9.2	Ardupilot SITL
MAVproxy	1.8.67	
pymavlink	2.4.40	
opencv-python	4.8.0.74	OpenCV
json	2.0.9	-
math	N/A	
matplotlib	3.7.2	
numpy	1.21.0	
scipy	1.3.3	
sys	N/A	
time	N/A	

N/A – Package version not available

Appendix B. Project Gantt Chart



Appendix C. Main Code for Geolocation Model Implementation

```
# Example Script for obtaining MAV camera image with position, orientation, timestamp
import airsim
import os
import cv2 as cv
import numpy as np
import math
import time
import json

# To separate between airsim tools and algorithm-relevant logics
from AirsimHandler import AirsimHandler
from Camera import Camera

def initialise(sim):
# Set sim camera parameters
name = "MAV" # as per airsim settings.json
elevation, azimuth = -np.pi/2, 0 # facing downwards
relative_orientation = airsim.to_quaternion(elevation, azimuth, 0)
distortion_params = {"K1": 0.0, "K2": 0.0, "K3": 0.0, "P1": 0.0, "P2": 0.0} # no distortion
sim.initialiseCamera(name, relative_orientation, distortion_params)

# Initialise algorithm's camera parameters
mav_cam = Camera(sim.field_of_view, sim.image_center, elevation, azimuth)
return mav_cam

if __name__ == '__main__':
# Takeoff and get to initial position, wait to stabilise
sim = AirsimHandler()
sim.client.moveToPositionAsync(55, 0, -30, 5).join() # call join() to wait for task to complete
time.sleep(3)

# Initialise camera parameters and create mav camera
mav_cam = initialise(sim)

# Update camera
sim.updateCameraPoseWithMAV()
mav_cam.update_states(sim.image, sim.MAV_position, sim.MAV_orientation, sim.timestamp)

# Object Coordinate Ground truth (replace with detector here)
object_image_coordinate = [140,89]
object_world_coordinate = [51,3,0]
im = cv.circle(mav_cam.image, np.array(mav_cam.params.image_center, dtype=np.int32), 2, (0,255,0), 2)
mav_cam.image = cv.circle(im, object_image_coordinate, 2, (0,0,255), 2)

# Perform Geolocation
inertial = mav_cam.geolocation(object_image_coordinate, inertial=True)

print("Calculated Relative: {}".format(inertial-mav_cam.MAV.position))
print("Actual relative: {}".format(object_world_coordinate-mav_cam.MAV.position))

# Save sample image and corresponding states
cv.imwrite(r'.\drone_image.png', mav_cam.image
```

Appendix D.1. Camera Script for Geolocation Model Implementation (1)

```
import numpy as np
import cv2
from scipy.spatial.transform import Rotation
from numpy import cos, sin



"""

Parameter Classes

"""
class CameraParameters:
def __init__(self, field_of_view, image_center):
# fov: float
# image_center: 1D int vector [height/2, width/2]
self.image_center = image_center
self.focal_length = 2*(self.image_center[1]) / (2*np.tan(np.deg2rad(field_of_view)/2))

class MAV_states:
def __init__(self):
self.position = []
self.orientation = []
self.velocity = []
self.prev_position = []
self.prev_orientation = []

def update_states(self, position, orientation, velocity):
"""
position: 1D float vector [N,E,D]
orientation: 1D float vector in quaternion[q1, q2, q3, q4]
"""
# Save current states to previous states
self.prev_position = self.position
self.prev_orientation = self.orientation

# Update current states
self.position = position
self.orientation = orientation


class Object_states:
def __init__(self):
self.position = []
self.velocity = 0
self.prev_position = []
self.prev_velocity = 0

def update_states(self):
self.prev_position = self.position
self.prev_velocity = self.velocity

class GeolocationCoefficients:
def __init__(self, elevation, azimuth):
# elevation, azimuth - floats in radian
self.gimbal_to_camera = np.array([[0,1,0],[0,0,1],[1,0,0]])
```

Appendix D.2. Camera Script for Geolocation Model Implementation (2)

```
self.body_to_gimbal = self.btg_matrix(elevation, azimuth)
self.rotation_matrix = []

def btg_matrix(self, el, az):
# el, az - floats in radian
r1 = [cos(el)*cos(az), cos(el)*sin(az), -sin(el)]
r2 = [-sin(az), cos(az), 0]
r3 = [sin(el)*cos(az), sin(el)*sin(az), cos(el)]
return np.array([r1,r2,r3])

def update_rotation_matrix(self, vehicle_orientation):
# vehicle_orientation - 1D float vector in pitch roll yaw (radian)

phi = vehicle_orientation[1] # roll
theta = vehicle_orientation[0] # pitch
psi = vehicle_orientation[2] # yaw

vehicle_to_body = np.array([
[cos(theta)*cos(psi) , cos(theta)*sin(psi) , -sin(theta)],
[sin(theta)*sin(phi)*cos(psi) - cos(phi)*sin(psi) ,
sin(theta)*sin(phi)*sin(psi) + cos(phi)*cos(psi) , cos(theta)*sin(phi)],
[cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi) ,
sin(theta)*cos(phi)*sin(psi) - sin(phi)*cos(psi) , cos(theta)*cos(phi)]
])

rotation = np.matmul(self.gimbal_to_camera, np.matmul(self.body_to_gimbal,vehicle_to_body))
self.rotation_matrix = np.linalg.inv(rotation)

"""

Camera Class

"""
class Camera:
def __init__(self, field_of_view, image_center, elevation, azimuth):
"""
field_of_view, elevation, azimut: float
image_center: 1D int vector [height/2, width/2]
"""

# Create state variables
self.image = []
self.prev_timestamp = 0
self.timestamp = 0
self.MAV = MAV_states()
self.obj = Object_states()
# Setup Parameters
self.params = CameraParameters(field_of_view, image_center)
self.geo_coeffs = GeolocationCoefficients(elevation, azimuth)


def update_states(self, image, MAV_position, MAV_orientation, MAV_velocity, timestamp):
"""
image: 2D array 3 channel BGR
MAV_position: 1D float vector [N,E,D]
MAV_orientation: 1D float vector [pitch, roll, yaw]
```

Appendix D.3. Camera Script for Geolocation Model Implementation (3)

```
MAV_velocity: 1D float vector [VN, VE, VD]
timestamp: float
"""
self.prev_timestamp = self.timestamp
self.MAV.prev_position = self.MAV.position
self.MAV.prev_orientation = self.MAV.orientation

self.image = image
self.timestamp = timestamp
self.obj.update_states()
self.MAV.update_states(MAV_position, MAV_orientation, MAV_velocity)
self.geo_coeffs.update_rotation_matrix(MAV_orientation)

def geolocation(self, obj_x, obj_y):
"""
image_center: 1D vector [py, px]
"""
# Calculate distance from center
px = -(obj_x - self.params.image_center[1]) # Image x axis inverted
py = obj_y - self.params.image_center[0]
image_coordinate = [px,py,self.params.focal_length] # Homogenous coordinate

# Calculate LOS vector (order corresponds to NED)
LOS_vector = image_coordinate/np.linalg.norm(image_coordinate)
LOS_vector = np.reshape(LOS_vector, (3,1))

# Range from Flat Earth Model
coefficients = np.matmul(self.geo_coeffs.rotation_matrix,LOS_vector)
altitude = np.abs(self.MAV.position[2])
distance = altitude / coefficients[2]

relative_vector = np.reshape(distance*coefficients, (1,3))
self.obj.position = self.MAV.position + relative_vector

```

Appendix E. Airsim Handler Script for Geolocation Model Implementation

```
import numpy as np
import airsim
import time

class AirsimHandler:
def __init__(self):
# connect to the AirSim simulator
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
client.armDisarm(True)
client.takeoffAsync().join()
time.sleep(1)

self.client = client

def initialiseCamera(self, camera_name, relative_orientation, distortion_parameters, ext=True):
"""
camera_name: string, as per simulation setting
relative_orientation: airsim Quaternionr class (refer airsim/types.py)
distortion_parameters: dictionary in form of {"K1": 0.0, "K2": 0.0, "K3": 0.0, "P1": 0.0, "P2": 0.0}
ext: boolean
"""

self.name = camera_name
self.relative_orientation = relative_orientation
self.external = ext

self.client.simSetDistortionParams(self.name, distortion_parameters, external=self.external)
self.field_of_view = self.client.simGetCameraInfo(self.name, external=self.external).fov
self.image_requester = [airsim.ImageRequest(self.name, airsim.ImageType.Scene, False, False)]

response = self.client.simGetImages(self.image_requester, external=self.external)[0]
self.image_center = np.array([response.width/2, response.height/2])

self.image = []
self.position = []
self.orientation = []
self.timestamp = []

def updateCameraPoseWithMAV(self):
# get current MAV position and move camera over
MAV_states = self.client.getMultirotorState().kinematics_estimated
position = MAV_states.position
orientation = MAV_states.orientation + self.relative_orientation

cam_pose = airsim.Pose(position, orientation)
self.client.simSetCameraPose(self.name, cam_pose, external=self.external)

# get image at the current position
response = self.client.simGetImages(self.image_requester, external=self.external)[0]
image_1d = np.fromstring(response.image_data_uint8, dtype=np.uint8)
self.image = image_1d.reshape(response.height, response.width, 3)

# update stored image and MAV states
self.MAV_position = MAV_states.position.to_numpy_array()
self.MAV_orientation = MAV_states.orientation.to_numpy_array()
self.timestamp = response.time_stamp
```

Appendix F. Code for Dataset Reader during System Development

```
import json
import cv2
import numpy as np
from numpy import deg2rad as rad

"""
Note every angle information is stored in radians
"""

class Params:
def __init__(self, param):
self.target_pos = param["target_pos"]
self.fov = param["fov"]
center = param["image_center"]
self.image_center = [center[0], center[1]]
self.elevation = rad(param["elevation"])
self.azimuth = rad(param["azimuth"])

class DataEntry:
def __init__(self):
self.image = []
self.states = None

class DatasetReader:
def __init__(self):
# Read dataset
with open("./dataset/data_parameters.json") as json_file:
self.parameters = Params(json.load(json_file))
with open("./dataset/dataset.json") as json_file:
dataset = json.load(json_file)
self.initialise_dataset(dataset)
self.current_number = -1
self.update()

def initialise_dataset(self, dataset):
o = rad(np.array(dataset["MAV_ori"])) # deg to radian
p,r,y = o[:,0], o[:,1], o[:,2] # pry to rpy
dataset["MAV_ori"] = np.array([r,p,y]).T.tolist()
dataset["Time"] = np.asarray(dataset["Time"])/1000 # ms to s
self.dataset = dataset

def reset(self):
# Read first dataset
self.current_number = -1
self.update()

def update(self):
self.current_number+=1
self.image = cv2.imread(r'./dataset/data_{}.png'.format(self.current_number))
self.timestamp = self.dataset["Time"][self.current_number]
self.obj_bbox = self.dataset["OBJ_bbox"][self.current_number]
self.cmd = self.dataset["Command"][self.current_number]
self.pos = self.dataset["MAV_pos"][self.current_number]
self.ori = self.dataset["MAV_ori"][self.current_number]
self.vel = self.dataset["MAV_vel"][self.current_number]
self.pqr = [0,0,0]
self.uvwdot = [0,0,0]
# self.uvwdot = self.dataset["MAV_trans_acc"][self.current_number]

def states(self):
return self.image, self.cmd, self.pos, self.ori, self.vel, self.pqr, self.uvwdot, self.timestamp, self.obj_bbox
```

Appendix G.1. Code for Data Processor during System Development (1)

```
import json
import cv2
import numpy as np
from numpy import sin, cos
import matplotlib.pyplot as plt
import time
import sys

from tools.DatasetReader import DatasetReader
from tools.LPF import ImageCoordinate_LPF, Gyro_LPF
from StateEstimator import StateEstimator

def bbox_center(bbox):
# [down,right,h,w]
right, down, w, h = bbox
center = [int(down+0.5*h), int(right+0.5*w)]
return center

def visualise(image, center, data, coordinate=None, current_number=None):
cx,cy = data
cx, cy = int(cx), int(cy)
image_obj = cv2.circle(image, (cx+center[1], cy+center[0]), 2, (0, 255, 0), 2)
image_obj = cv2.circle(image_obj, (center[1], center[0]), 1, (0, 0, 255), 1)

# Visualise estimated obj position
if type(coordinate) != type(None):
ex, ey = coordinate
fy = center[1] + int(ex) # Horizontal right
fx = center[0] + int(ey) # Vertical down
image_obj = cv2.circle(image_obj, (fy, fx), 5, (0, 0, 255), 2)

# Display image unless current_number is parsed
if type(current_number) == type(None):
cv2.imshow("Image", image_obj)
cv2.waitKey(0)
else:
# Save images
if current_number < 10:
cv2.imwrite("./video_and_images/frames/data_00{}.png".format(current_number), image_obj)
elif current_number < 100:
cv2.imwrite("./video_and_images/frames/data_0{}.png".format(current_number), image_obj)
else:
cv2.imwrite("./video_and_images/frames/data_{}.png".format(current_number), image_obj)


if __name__ == '__main__':
# Note that all image coordinates are indexed as [y, x]
reader = DatasetReader()
t_pos = reader.parameters.target_pos
fov = reader.parameters.fov/2
center = reader.parameters.image_center
el = reader.parameters.elevation
az = reader.parameters.azimuth

```

Appendix G.1. Code for Data Processor during System Development (1)

```
last_data_no = 200
initial_time = reader.timestamp
time_log = []
obj_PN_log = []
obj_PE_log = []
obj_PD_log = []

obj_VN_log = []
obj_VE_log = []
obj_VD_log = []

obj_AN_log = []
obj_AE_log = []
obj_AD_log = []
cov_log = []
use_ground_truth = False
show_sigma = False
# Uncomment one of the below three

# No filter
filter = None

# Initial object state covariance
max_position_error = 10 # m
max_velocity = 20 # m/s
max_acceleration = 1.5*9.81 # m/s^2 (around 1.5 G)
initial_covariance = np.array([max_position_error/2, max_velocity/2, max_acceleration/2]) ** 2
motion_noise = 1 ** 2 # near constant acceleration noise (m/s)

# # KF
filter = 0
measurement_noise = (max_position_error/2)**2 # pos

# # EKF
filter = 1
max_px_error = 64 # px
max_uav_pos_error = 5 # m
max_uav_att_error = 0.1 # radian
measurement_noise = np.array([max_px_error/2, max_uav_pos_error/2, max_uav_att_error/2])**2 # pxpy, position, orientation

# Initialise State Estimator
estimator = None
if filter != None:
estimator = StateEstimator(fov, center, el, az, initial_covariance=initial_covariance, otion_noise=motion_noise, measurement_noise=measurement_noise, filter=filter)
else:
estimator = StateEstimator(fov, center, el, az, filter=filter)


# # Initialise LPF
enable_Image_LPF = False
LPF = None
if enable_Image_LPF == True:
LPF = ImageCoordinate_LPF()

```

Appendix G.1. Code for Data Processor during System Development (1)

```
# Set to 0 if filtering effect is not needed
GLPF = Gyro_LPF(time_constant=0)
# GLPF = Gyro_LPF(time_constant=15)


init_t = time.time()
while True:
print("Processing Dataset {}/{}".format(reader.current_number, last_data_no))

# Get the dataset, update estimatorera states and calculate center of the bounding box
image, cmd, pos, ori, vel, pqr, acc, timestamp, obj_bbox = reader.states()

# if reader.current_number != 0:
# GLPF.filter(estimator.timestamp - estimator.prev_timestamp, pqr)
# pqr = GLPF.filtered

estimator.update_states(image, cmd, pos, ori, vel, pqr, acc, timestamp)
coordinate = bbox_center(obj_bbox)

# Obtain filtered value of x,y image coordinate
if enable_Image_LPF == True:
LPF.filter_coordinate(timestamp - estimator.prev_timestamp, coordinate)
filtered = np.array(LPF.filtered, dtype=int)

# Override x,y with filtered values if LPF is enabled
if enable_Image_LPF == True:
coordinate = filtered

# Calculate the State Estimates from Image Coordinate
ex = (coordinate[1] - center[1]) # Horizontal left
ey = (coordinate[0] - center[0]) # Vertical down

if use_ground_truth == True:
relative_position = np.array([51.7,3.1,0]) - pos
unit_LOS_vector = relative_position / np.linalg.norm(relative_position)
unit_cam_vector = np.matmul(estimator.rot_matrices.calculate_rotation_matrix(ori,c2i=False),
unit_LOS_vector)
tx,ty,_ = (estimator.params.focal_length / unit_cam_vector[2]) * unit_cam_vector
tx = int(tx)
ty = int(ty)
ex, ey = tx, ty
# ex = int(np.random.normal(tx, 15))
# ey = int(np.random.normal(ty, 15))


estimator.state_estimation(ex,ey)
# Print and log the state estimation
time_log.append(round(reader.timestamp - initial_time,1))
obj_PN_log.append(estimator.OBJ.position[0])
obj_PE_log.append(estimator.OBJ.position[1])
obj_PD_log.append(estimator.OBJ.position[2])
obj_VN_log.append(estimator.OBJ.velocity[0])
obj_VE_log.append(estimator.OBJ.velocity[1])
obj_VD_log.append(estimator.OBJ.velocity[2])
obj_AN_log.append(estimator.OBJ.acceleration[0])
```

Appendix G.1. Code for Data Processor during System Development (1)

```
obj_AE_log.append(estimator.OBJ.acceleration[1])
obj_AD_log.append(estimator.OBJ.acceleration[2])
if type(filter) != type(None):
cov_log.append(list(estimator.state_filter.P))


"""
Visualise
"""
if reader.current_number > 1 and filter != None and filter > 0:
coord = estimator.state_filter.estimated_pxpy
visualise(image, center, [ex,ey], coordinate=coord)
## Uncomment this and comment above to save video
# visualise(image, center, [ex,ey], coordinate=coord, current_number=reader.current_number)
else:
visualise(image, center, [ex, ey], coordinate=None)
# visualise(image, center, [ex, ey], coordinate=[tx,ty], current_number=reader.current_number)


# End the program if all data has been processed
if reader.current_number >= last_data_no:
break

# Load dataset of the next time frame
reader.update()
print("Process took {} seconds for {} dataset - average {} Hz processing rate". format(round(time.time()-
init_t,2), last_data_no, round(1/((time.time()-init_t) / last_data_no),2)))

# Output Logs
title_string = ''
if filter == None:
title_string = 'NoFilter'
elif filter == 0:
title_string = 'KF'
elif filter == 1:
title_string = 'EKF'


# Log covariance if filter
log=[]
if filter == None:
log = {"t": time_log, "PN": obj_PN_log, "PE": obj_PE_log, "PD": obj_PD_log,
"VN": obj_VN_log, "VE": obj_VE_log,
"VD": obj_VD_log, "AN": obj_AN_log, "AE": obj_AE_log, "AD": obj_AD_log}
else:
c_log = np.array(cov_log)
PNcov = list(c_log[:,0,0])
PEcov = list(c_log[:,1,1])
VNcov = list(c_log[:,3,3])
VEcov = list(c_log[:,4,4])
ANcov = list(c_log[:,6,6])
AEcov = list(c_log[:,7,7])
log = {"t": time_log, "PN": obj_PN_log, "PE": obj_PE_log, "PD": obj_PD_log, "VN": obj_VN_log, "VE":
obj_VE_log,
```

Appendix G.1. Code for Data Processor during System Development (1)

```
"VD": obj_VD_log, "AN": obj_AN_log, "AE": obj_AE_log, "AD": obj_AD_log, "PNcov":PNcov, "PEcov":PEcov, "VNcov":VNcov, "VEcov":VEcov, "ANcov":ANcov, "AEcov":AEcov}
with open("./output/{}_Result.json".format(title_string), "w") as json_file:
json.dump(log, json_file)


# Plot results
fig, ax = plt.subplots(3, 3)

# Plot true states
true_PN = np.full((len(time_log),), 51.7)
true_PE = np.full((len(time_log),), 3.1)
true_PD = np.full((len(time_log),), 0)
true_V = np.full((len(time_log),), 0)
true_A = np.full((len(time_log),), 0)

ax[0, 0].plot(time_log, true_PN, label='True')
ax[1, 0].plot(time_log, true_PE, label='True')
ax[2, 0].plot(time_log, true_PD, label='True')
ax[0, 1].plot(time_log, true_V, label='True')
ax[1, 1].plot(time_log, true_V, label='True')
ax[2, 1].plot(time_log, true_V, label='True')
ax[0, 2].plot(time_log, true_A, label='True')
ax[1, 2].plot(time_log, true_A, label='True')
ax[2, 2].plot(time_log, true_A, label='True')

# Plot resulting estimated states
ax[0, 0].plot(time_log, obj_PN_log, label='Estimated')
ax[1, 0].plot(time_log, obj_PE_log, label='Estimated')
ax[2, 0].plot(time_log, obj_PD_log, label='Estimated')
ax[0, 1].plot(time_log, obj_VN_log, label='Estimated')
ax[1, 1].plot(time_log, obj_VE_log, label='Estimated')
ax[2, 1].plot(time_log, obj_VD_log, label='Estimated')
ax[0, 2].plot(time_log, obj_AN_log, label='Estimated')
ax[1, 2].plot(time_log, obj_AE_log, label='Estimated')
ax[2, 2].plot(time_log, obj_AD_log, label='Estimated')


# Plot uncertainty region
if type(filter) != type(None) and show_sigma == True:
# Plot resulting estimated states
c_log = 3*(np.array(cov_log)[:,0,0]**0.5)
ax[0, 0].fill_between(time_log, obj_PN_log-c_log, c_log+obj_PN_log, alpha=0.2, label='3σ')
c_log = 3*(np.array(cov_log)[:,1,1]**0.5)
ax[1, 0].fill_between(time_log, obj_PE_log-c_log, c_log+obj_PE_log, alpha=0.2, label='3σ')
c_log = 3*(np.array(cov_log)[:,3,3]**0.5)
ax[0, 1].fill_between(time_log, obj_VN_log-c_log, c_log+obj_VN_log, alpha=0.2, label='3σ')
c_log = 3*(np.array(cov_log)[:,4,4]**0.5)
ax[1, 1].fill_between(time_log, obj_VE_log-c_log, c_log+obj_VE_log, alpha=0.2, label='3σ')
c_log = 3*(np.array(cov_log)[:,6,6]**0.5)
ax[0, 2].fill_between(time_log, obj_AN_log-c_log, c_log+obj_AN_log, alpha=0.2, label='3σ')
c_log = 3*(np.array(cov_log)[:,7,7]**0.5)
ax[1, 2].fill_between(time_log, obj_AE_log-c_log, c_log+obj_AE_log, alpha=0.2, label='3σ')


```

Appendix G.1. Code for Data Processor during System Development (1)

```
# Limit axis range
# ax[0,0].set_ylim(45, 57)
# ax[1,0].set_ylim(-20, 5)
# ax[2, 0].set_ylim(-1, 1)
# ax[0,1].set_ylim(-25, 35)
# ax[1,1].set_ylim(-10, 25)
# ax[2, 1].set_ylim(-1, 1)
# ax[0,2].set_ylim(-20, 20)
# ax[1,2].set_ylim(-10, 15)
# ax[2, 2].set_ylim(-1, 1)

# Legend
ax[0, 0].legend(loc=1, fontsize=8)
ax[0, 1].legend(loc=1, fontsize=8)
ax[0, 2].legend(loc=1, fontsize=8)
ax[1, 0].legend(loc=1, fontsize=8)
ax[1, 1].legend(loc=1, fontsize=8)
ax[1, 2].legend(loc=1, fontsize=8)
ax[2, 0].legend(loc=1, fontsize=8)
ax[2, 1].legend(loc=1, fontsize=8)
ax[2, 2].legend(loc=1, fontsize=8)

# Label Titles
fig.suptitle('Estimated vs True Object States with {}'.format(title_string))

# Label NED
plt.setp(ax[0, 0], title='Position (m)', ylabel='North')
plt.setp(ax[1, 0], ylabel='East')
plt.setp(ax[2, 0], ylabel='Down')
plt.setp(ax[0, 1], title='Velocity (m/s)')
plt.setp(ax[0, 2], title='Acceleration (m/s^2)')

# Show plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()


```

Appendix H.1. Code for Filter Script Selector during System Development (1)

```
import numpy as np
from numpy import sin, cos
import time
from tools.StateWrapper import CameraParameters, MAV_states, Object_states, RotationMatrices

from models.EKF import ObjectState_EKF as EKF
from models.KF import ObjectState_KF as KF

class StateEstimator:
def __init__(self, field_of_view, image_center, elevation, azimuth, initial_covariance=None,
motion_noise=None, measurement_noise=None, filter=None):
"""
field_of_view, elevation, azimut: float
image_center: 1D int vector [height/2, width/2]
"""

# Create state variables
self.image = []
self.prev_timestamp = 0
self.timestamp = 0
self.MAV = MAV_states()
self.OBJ = Object_states()
# Setup Parameters
self.params = CameraParameters(field_of_view, image_center)
self.rot_matrices = RotationMatrices(elevation, azimuth)
self.filter = filter
if self.filter != None:
# Filter == 0 for KF, 1 for EKF
if self.filter == 0:
self.state_filter = KF(initial_covariance=initial_covariance, motion_noise=motion_noise,
measurement_noise=measurement_noise)
elif self.filter == 1:
self.state_filter = EKF(self.params.focal_length, self.rot_matrices, initial_covariance=initial_covariance,
motion_noise=motion_noise, measurement_noise=measurement_noise)

def update_states(self, image, MAV_command, MAV_position, MAV_orientation, MAV_velocity, MAV_pqr,
MAV_acc, timestamp):
"""
image: 2D array 3 channel BGR
MAV_position: 1D float vector [N,E,D]
MAV_orientation: 1D float vector [pitch, roll, yaw]
MAV_velocity: 1D float vector [VN, VE, VD]
MAV_pqr: 1D float vector [p, q, r] (gyro reading)
timestamp: float
"""
self.prev_timestamp = self.timestamp
self.MAV.prev_position = self.MAV.position
self.MAV.prev_orientation = self.MAV.orientation

self.image = image
self.timestamp = timestamp
self.OBJ.update_states()
self.MAV.update_states(MAV_command, MAV_position, MAV_orientation, MAV_velocity, MAV_pqr,
MAV_acc)

```

Appendix H.2. Code for Filter Script Selector during System Development (2)

```
def image_to_relative(self, image_coordinate):
# Calculate LOS vector (order corresponds to NED)
LOS_vector = (np.array(image_coordinate) / np.linalg.norm(image_coordinate)).T

# Range from Flat Earth Model
rotation_matrix = self.rot_matrices.calculate_rotation_matrix(self.MAV.orientation, c2i=True)
coefficients = np.matmul(rotation_matrix, LOS_vector)
altitude = np.abs(self.MAV.position[2])
scale = altitude / coefficients[2]
relative_vector = np.reshape(scale * coefficients, (1, 3))[0]
return relative_vector

def state_estimation(self, ex, ey):
"""
ex, ey: center to right, center to down
image_center: 1D vector [px, py], although OpenCV inverts axis (i.e. will be [py, px] for OpenCV)
"""

image_coordinate = [ex,ey,self.params.focal_length]
relative_position = self.image_to_relative(image_coordinate)
object_position = (relative_position + self.MAV.position)
Tp = self.timestamp - self.prev_timestamp

# Filter object_position with KF (0) or EKF Relative Model (1) EKF Decoupled Model (2)
if type(self.filter) != type(None):
# Filter position with KF
if self.state_filter.initialised == False:
# If performed geolocation for first time
initial_Tp_estimate = 0.1
initial_position_estimate_offset = [0, 0, 0]
initial_velocity_estimate = [0, 0, 0]
initial_acceleration_estimate = [0, 0, 0]
self.state_filter.initial_state((object_position+initial_position_estimate_offset), initial_velocity_estimate,
initial_acceleration_estimate)
self.state_filter.predict(initial_Tp_estimate)
else:
self.state_filter.predict(Tp)
if self.filter == 0:
self.state_filter.update(object_position)
else:
self.state_filter.update(image_coordinate, self.MAV, Tp)
object_position = self.state_filter.estimated_object_position()
self.OBJ.position = object_position


"""
Velocity Estimation
"""
if self.filter == None:
position_change = np.array(self.OBJ.position) - np.array(self.OBJ.prev_position)
time_change = self.timestamp - self.prev_timestamp
estimated_velocity = (position_change / time_change)
self.OBJ.velocity = estimated_velocity
else:
if self.state_filter.initialised == True:
self.OBJ.velocity = self.state_filter.estimated_object_velocity()
```

Appendix H.3. Code for Filter Script Selector during System Development (3)

```
106
107
108 """
109 Acceleration Estimation
110 """
111 if self.filter == None:
112 velocity_change = np.array(self.OBJ.velocity) -
np.array(self.OBJ.prev_velocity)
113 time_change = self.timestamp - self.prev_timestamp
114 estimated_acceleration = (velocity_change / time_change)
115 self.OBJ.acceleration = estimated_acceleration
116 else:
117 if self.state_filter.initialised == True:
118 self.OBJ.acceleration = self.state_filter.estimated_object_acceleration
```

Appendix I.1. Code for Class Definitions during System Development (1)

```
1 import numpy as np
2 from numpy import cos, sin
3
4
5 class CameraParameters:
6 def __init__(self, field_of_view, image_center):
7 # fov: float
8 # image_center: 1D int vector [height/2, width/2]
9 self.image_center = image_center
10 self.focal_length = int((2*self.image_center[1]) / (2*np.tan(np.deg2rad(field_of_view)/2)))
11
12
13 class MAV_states:
14 def __init__(self):
15 self.position = [0,0,0]
16 self.orientation = [0,0,0]
17 self.prev_position = [0,0,0]
18 self.prev_orientation = [0,0,0]
19
20 # Required
21 self.pqr = [0,0,0] # Angular Velocity in Body frame
22 # self.uvw = [0,0,0] # Translational Velocity in Body frame
23 self.uvw_dot = [0,0,0] # Translational Acceleration in Body frame
24
25
26 def update_states(self, command, position, orientation, velocity, pqr, acc):
27 """
28 position: 1D float vector [N,E,D]
29 orientation: 1D float vector in quaternion[q1, q2, q3, q4]
30 """
31 # Save current states to previous states
32 self.prev_position = self.position
33 self.prev_orientation = self.orientation
34
35 # Update current states
36 self.command = command
37 self.position = position
```

Appendix I.2. Code for Class Definitions during System Development (2)

```
self.orientation = orientation
self.velocity = velocity
self.pqr = pqr
self.uvw_dot = acc


class Object_states:
def __init__(self):
self.iteration = 0
self.position = [0,0,0]
self.prev_position = [0,0,0]
self.velocity = [0,0,0]
self.prev_velocity = [0,0,0]
self.acceleration = None


def update_states(self, Tp=0.1, position=[0,0,0]):
self.prev_position = self.position
self.prev_velocity = self.velocity
self.position = position

if self.iteration != 0:
self.velocity = ((np.array(self.position) - np.array(self.prev_position))/Tp)
if self.iteration == 3:
self.acceleration = ((self.velocity-self.prev_velocity)/Tp)
else:
self.iteration+=1


class Relative_states:
def __init__(self):
self.iteration = 0
self.position = None
self.prev_position = None
self.velocity = None
self.prev_velocity = None
self.acceleration = None


def update_states(self, Tp, position, MAV_down_vel):
self.prev_position = self.position
self.prev_velocity = self.velocity
self.position = position

if self.iteration != 0:
self.velocity = ((self.position - self.prev_position)/Tp)
self.velocity[2] = -MAV_down_vel
if self.iteration == 3:
self.acceleration = ((self.velocity-self.prev_velocity)/Tp)
else:
self.iteration+=1


class RotationMatrices:
def __init__(self, elevation, azimuth):
# elevation, azimuth - floats in radian
```

Appendix I.3. Code for Class Definitions during System Development (3)

```
94 self.gimbal_to_camera = np.array([[0,1,0],[0,0,1],[1,0,0]])
95 self.body_to_gimbal = self.btg_matrix(elevation, azimuth)
96 self.camera_orientation = [elevation, azimuth]
97 self.rotation_matrix = []
98
99 def btg_matrix(self, el,az):
100 # el, az - floats in radian
101 r1 = [cos(el)*cos(az), cos(el)*sin(az), -sin(el)]
102 r2 = [-sin(az), cos(az), 0]
103 r3 = [sin(el)*cos(az), sin(el)*sin(az), cos(el)]
104 return np.array([r1,r2,r3])
105
106 def calculate_rotation_matrix(self, vehicle_orientation, c2i=True):
107 # vehicle_orientation - 1D float vector in theta phi psi (radian)
108
109 phi = vehicle_orientation[0] # phi
110 theta = vehicle_orientation[1] # theta
111 psi = vehicle_orientation[2] # psi
112
113 vehicle_to_body = np.array([
114 [cos(theta)*cos(psi) , cos(theta)*sin(psi) , -sin(theta)],
115 [sin(phi)*sin(theta)*cos(psi) - cos(phi)*sin(psi) ,
sin(phi)*sin(theta)*sin(psi) + cos(phi)*cos(psi) , cos(theta)*sin(phi)],
116 [cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi) ,
sin(theta)*cos(phi)*sin(psi) - sin(phi)*cos(psi) , cos(theta)*cos(phi)]
117 ])
118
119 # if c2i == True:
120 # # Camera to Inertial
121 # return np.matmul(np.linalg.inv(vehicle_to_body),
np.linalg.inv(self.gimbal_to_camera))
122 # else:
123 # # Inertial to Camera
124 # return np.matmul(self.gimbal_to_camera,vehicle_to_body)
125
126 if c2i == True:
127 # Camera to Inertial
128 return np.matmul(np.linalg.inv(vehicle_to_body),
np.matmul(np.linalg.inv(self.body_to_gimbal),
np.linalg.inv(self.gimbal_to_camera)))
129 else:
130 # Inertial to Camera
131 return np.matmul(self.gimbal_to_camera,np.matmul(self.body_to_gimbal,
vehicle_to_body))
```

Appendix J. Numerical Calculation of Jacobian Matrix

```
def jacobian(self, model, x):
f = model(x) # f returns predicted rate in change of states
m = f.shape[1]
n = x.shape[1]

eps = 0.01 # deviation
J = np.zeros((m,n))
for i in range(0,n):
x_eps = np.copy(x)
x_eps[0][i]+=eps
f_eps = model(x_eps)
df = (f_eps - f) / eps
J[:,i] = df[0,:]

return J
```

Appendix K. Derivation of Motion Noise Matrix Q_t

Constant Acceleration with Noise q

$$a = a_{cmd} + q$$

Acceleration Noise in Velocity

$$v_t = (a_{cmd} + q)dt + v_{t-1}$$

Acceleration Noise in Position

$$p_t = \frac{1}{2}(a_{cmd} + q)dt^2 + v_t dt + p_{t-1}$$

$$\sigma_p = \frac{1}{2}dt^2 \cdot q, \qquad \sigma_v = dt \cdot q, \qquad \sigma_a = 1 \cdot q$$

$$Q = \int \begin{bmatrix} \sigma_p^2 & \sigma_p\sigma_v & \sigma_p\sigma_a \\ \sigma_v\sigma_p & \sigma_v^2 & \sigma_v\sigma_a \\ \sigma_a\sigma_p & \sigma_a\sigma_v & \sigma_a^2 \end{bmatrix} dt$$

$$= \int \begin{bmatrix} \frac{dt^4}{4} & \frac{dt^3}{2} & \frac{dt^2}{2} \\ \frac{dt^3}{2} & dt^2 & dt \\ \frac{dt^2}{2} & dt & 1 \end{bmatrix} \cdot q^2 \, dt$$

$$= \begin{bmatrix} \frac{dt^5}{20} & \frac{dt^4}{8} & \frac{dt^3}{6} \\ \frac{dt^4}{8} & \frac{dt^3}{3} & \frac{dt^2}{2} \\ \frac{dt^3}{6} & \frac{dt^2}{2} & dt \end{bmatrix} \cdot q^2$$

Appendix L.1. EKF Implementation Code (1)

```
import numpy as np
from numpy import tan
from numpy import cos
from numpy import sin

class ObjectState_EKF:
def __init__(self, f, rotation_matrices, initial_covariance=[5, 5, 5], motion_noise = 2,
measurement_noise=[1,1,0.1,10]):
self.initialised = False
self.rot_mat = rotation_matrices # For measurement prediction (body to camera frame)
self.focal_length = f

# Object Initial State Covariance (mav_ori,mav_pos, pos, vel, acc)
obj_pos_cov, obj_vel_cov, obj_acc_cov = initial_covariance
self.P = np.diag([obj_pos_cov, obj_pos_cov, 0, obj_vel_cov, obj_vel_cov, 0, obj_acc_cov, obj_acc_cov, 0])

self.q = motion_noise

# Object Image Pixel Noise (No noise on focal length)
pxpy_noise, pos_noise, ori_noise = measurement_noise
self.R =
np.diag([pxpy_noise,pxpy_noise,pos_noise,pos_noise,pos_noise,ori_noise,ori_noise, ori_noise])

def initial_state(self, OBJ_pos, OBJ_vel, OBJ_acc):
N,E,D = OBJ_pos
VN,VE,VD = OBJ_vel
AN,AE,AD = OBJ_acc
self.last_position = OBJ_pos
self.x_hat = np.array([[N,E,D,VN,VE,VD,AN,AE,AD]])
self.initialised = True

"""
MAV State Conversion Equations
"""
```

Appendix L.2. EKF Implementation Code (2)

```
33 def angular_rotation_matrix(self, phi, theta):
34 # Body to Inertial
35 return np.array([[1, sin(phi) * tan(theta), cos(phi) * tan(theta)],
36 [0, cos(phi), -sin(phi)],
37 [0, sin(phi) / cos(theta), cos(phi) / cos(theta)]])
38
39 def translational_rotation_matrix(self, phi, theta, psi):
40 # Body to Inertial
41 return np.array([[cos(theta) * cos(psi), sin(theta) * sin(phi) * cos(psi) - cos(phi) * sin(psi), cos(phi) * sin(theta) *
cos(psi) + sin(phi) * sin(psi)],
42 [cos(theta) * sin(psi), sin(theta) * sin(phi) * sin(psi) + cos(phi) * cos(psi), sin(theta) * cos(phi) * sin(psi) -
sin(phi) * cos(psi)],
43 [-sin(theta), cos(theta) * sin(phi), cos(theta) * cos(phi)]])
44
45 def Q(self, Tp):
46 Q = np.array([[(1/20)*Tp**5, 0,0, (1/8)*Tp**4, 0,0, (1/6)*Tp**3, 0,0],
47 [0, (1/20)*Tp**5, 0,0, (1/8)*Tp**4, 0,0, (1/6)*Tp**3, 0],
48 [0, 0, 0, 0, 0, 0, 0, 0, 0],
49 [(1/8)*Tp**4, 0,0, (1/3)*Tp**3, 0,0, (1/2)*Tp**2, 0,0],
50 [0, (1/8)*Tp**4, 0,0, (1/3)*Tp**3, 0,0, (1/2)*Tp**2, 0],
51 [0, 0, 0, 0, 0, 0, 0, 0, 0],
52 [(1/6)*Tp**3, 0,0, (1/2)*Tp**2, 0,0, Tp, 0,0],
53 [0, (1/6)*Tp**3, 0,0, (1/2)*Tp**2, 0,0, Tp, 0],
54 [0, 0, 0, 0, 0, 0, 0, 0, 0]])
55 Q = self.q * Q
56 return Q
57
58 def f_obj(self, Tp):
59 f_obj = np.array([[1, 0, 0, Tp, 0, 0, 0.5*(Tp**2), 0, 0],
60 [0, 1, 0, 0, Tp, 0, 0, 0.5*(Tp**2), 0],
61 [0, 0, 0, 0, 0, 0, 0, 0, 0],
62 [0, 0, 0, 1, 0, 0, Tp, 0, 0],
63 [0, 0, 0, 0, 1, 0, 0, Tp, 0],
64 [0, 0, 0, 0, 0, 0, 0, 0, 0],
65 [0, 0, 0, 0, 0, 0, 1, 0, 0],
66 [0, 0, 0, 0, 0, 0, 0, 1, 0],
67 [0, 0, 0, 0, 0, 0, 0, 0, 0]])
68 return f_obj
69 def state_prediction(self, x_hat):
70 # OBJ States Prediction
71 state_trainsition_matrix = self.f_obj(self.Tp)
72 state = np.matmul(state_trainsition_matrix, x_hat.T).T
73 return state
74
75 def predict(self, Tp):
76 self.Tp = Tp
77 self.x_hat = self.state_prediction(self.x_hat)
78 F = self.f_obj(Tp)
79 self.P = np.matmul(np.matmul(F,self.P),F.T) + self.Q(Tp)
80
81 def measurement_covariance(self,z):
82 relative_position = self.x_hat[0][:3] - z[0,2:5]
83 unit_LOS_vector = relative_position/np.linalg.norm(relative_position)
84
85 inertial_to_camera_matrix = self.rot_mat.calculate_rotation_matrix(z[0,5:8], c2i=False)
86 unit_cam_vector = np.matmul(inertial_to_camera_matrix, unit_LOS_vector)
```

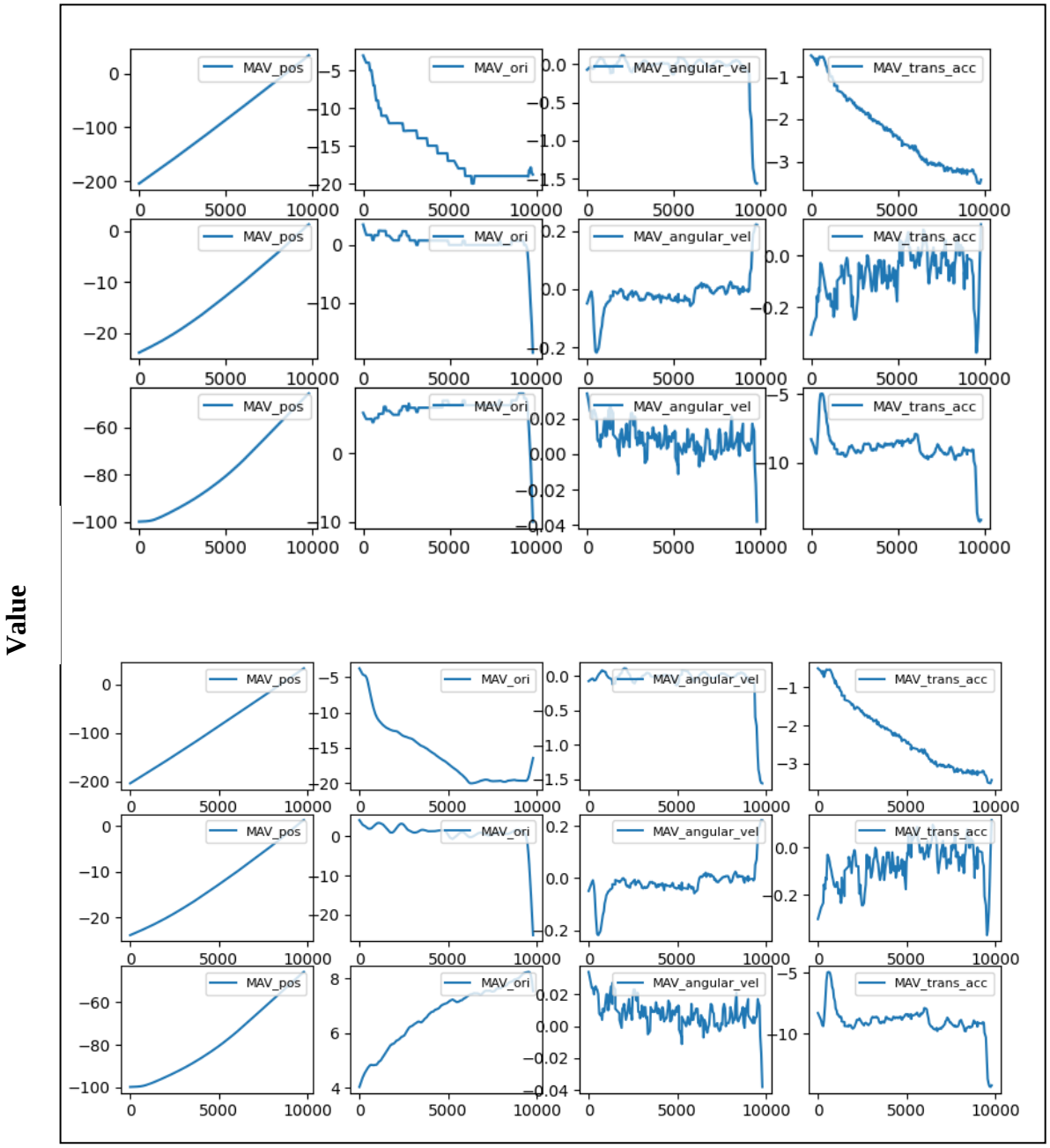
Appendix L.3. EKF Implementation Code (3)

```
87 px, py, f = (self.focal_length/unit_cam_vector[2]) * unit_cam_vector
88
89 n,e,d = z[0,2:5]
90 phi, theta, psi = z[0,5:8]
91 predicted_measurement = np.array([[px,py,n,e,d,phi,theta,psi]])
92 return predicted_measurement
93
94 def calculate_covariance_matrix(self, z):
95 cov_matrix = self.jacobian(self.measurement_covariance, z)
96 cov_matrix[0,0], cov_matrix[1,1] = 1, 1
97 return cov_matrix
98
99 def measurement_model(self,x_hat):
100 # phi,theta, _ = self.MAV.prev_orientation
101 # phi, theta, psi = self.MAV.prev_orientation +
np.matmul(self.angular_rotation_matrix(phi,theta),np.array(self.MAV.pqr)) *
self.Tp
102 # n,e,d = self.MAV.prev_position + np.array(self.MAV.velocity)*self.Tp
103
104 n,e,d = self.MAV.position
105 phi,theta,psi = self.MAV.orientation
106 N,E,D = self.x_hat[0][3:6]
107 relative_position = x_hat[0][:3] - [n,e,d]
108 unit_LOS_vector = relative_position/np.linalg.norm(relative_position)
109
110 inertial_to_camera_matrix =
self.rot_mat.calculate_rotation_matrix(self.MAV.orientation, c2i=False)
111 unit_cam_vector = np.matmul(inertial_to_camera_matrix, unit_LOS_vector)
112 px, py, f = (self.focal_length/unit_cam_vector[2]) * unit_cam_vector
113
114 predicted_measurement = np.array([[px,py,n,e,d,phi,theta,psi]])
115 return predicted_measurement
116
117 def update(self, image_coordinate, MAV, Tp):
118 self.Tp = Tp
119 self.MAV = MAV
120 n,e,d = MAV.position
121 r,p,y = MAV.orientation
122 px, py, _ = image_coordinate
123
124 actual_measurement = np.array([[px,py,n,e,d,r,p,y]])
125
126 C = self.jacobian(self.measurement_model, self.x_hat)
127 cov = self.calculate_covariance_matrix(actual_measurement)
128 noise_cov = np.matmul(np.matmul(cov, self.R), cov.T)
129 term = noise_cov + np.matmul(np.matmul(C, self.P), C.T)
130
131 inverse_term = np.linalg.inv(term)
132 self.kalman_gain = np.matmul(np.matmul(self.P, C.T),inverse_term)
133 self.P = np.matmul((np.eye(9)-np.matmul(self.kalman_gain,C)), self.P)
134
135
136 predicted_measurement = self.measurement_model(self.x_hat)
137 prediction_error = (actual_measurement - predicted_measurement).T
138 prev_pos = self.x_hat[0][:3]
139 self.x_hat = self.x_hat + np.matmul(self.kalman_gain, prediction_error).T
```

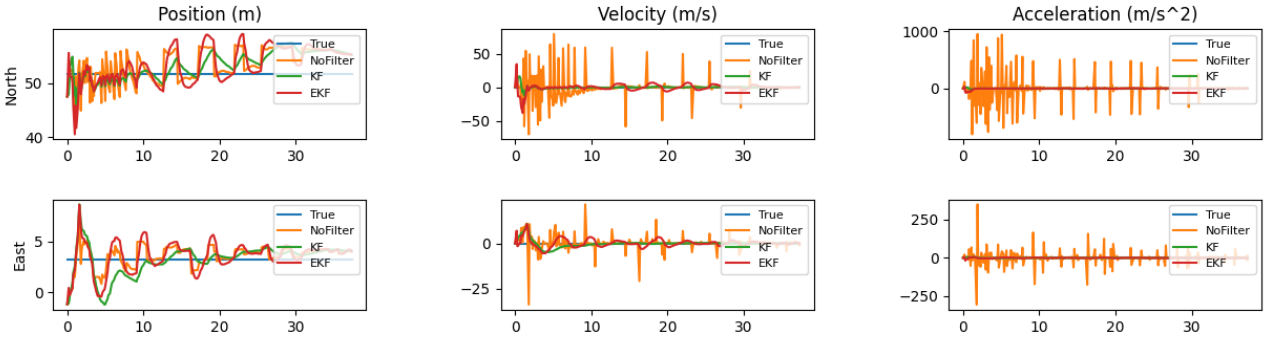
Appendix L.4. EKF Implementation Code (4)

```
140 self.last_px = px,py
141 position_error = self.x_hat[0][:3] - prev_pos
142 # print("Prediction Error in position: {}".format(position_error))
143 # print("Prediction Error in pxpy: {}".format(prediction_error[:2].T))
144
145 self.estimated_pxpy = self.measurement_model(self.x_hat)[0][:2]
146
147 def estimated_object_position(self):
148 estimated_obj_position = self.x_hat[0][:3]
149 return estimated_obj_position
150
151 def estimated_object_velocity(self):
152 estimated_obj_velocity = self.x_hat[0][3:6]
153 return estimated_obj_velocity
154
155 def estimated_object_acceleration(self):
156 estimated_obj_acceleration = self.x_hat[0][6:9]
157 return estimated_obj_acceleration
158
159
160 def jacobian(self, model, x):
161 f = model(x) # f returns predicted rate in change of states
162 m = f.shape[1]
163 n = x.shape[1]
164
165 eps = 0.01 # deviation
166 J = np.zeros((m,n))
167 for i in range(0,n):
168 x_eps = np.copy(x)
169 x_eps[0][i]+=eps
170 f_eps = model(x_eps)
171 df = (f_eps - f) / eps
172 J[:,i] = df[0,:]
173
174 return J
```

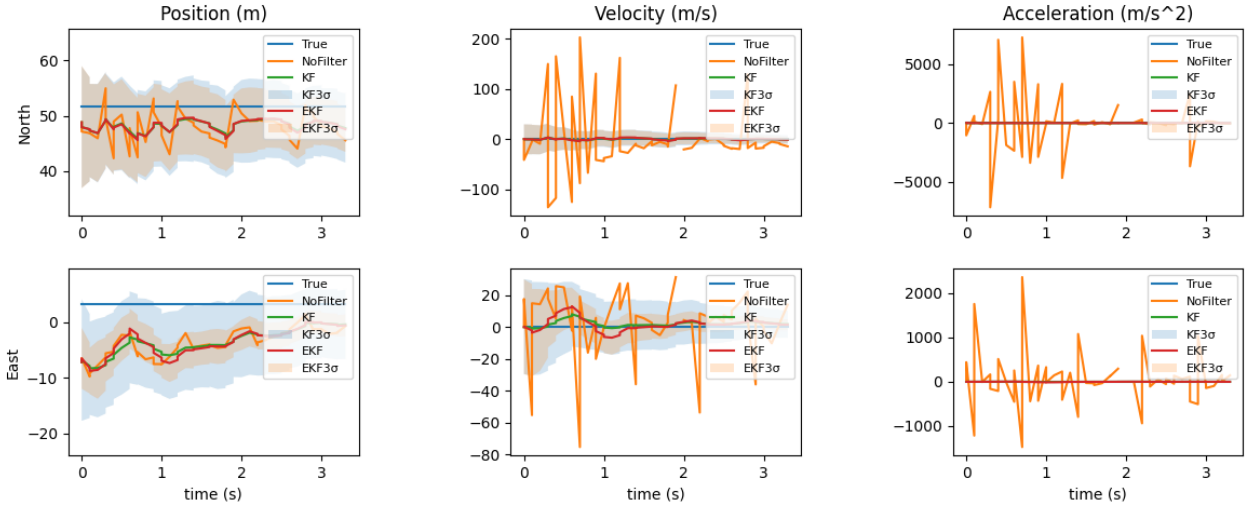
Appendix M. MAV position, orientation, angular velocity and translational acceleration obtained from Airsim (top) and Ardupilot (bottom)



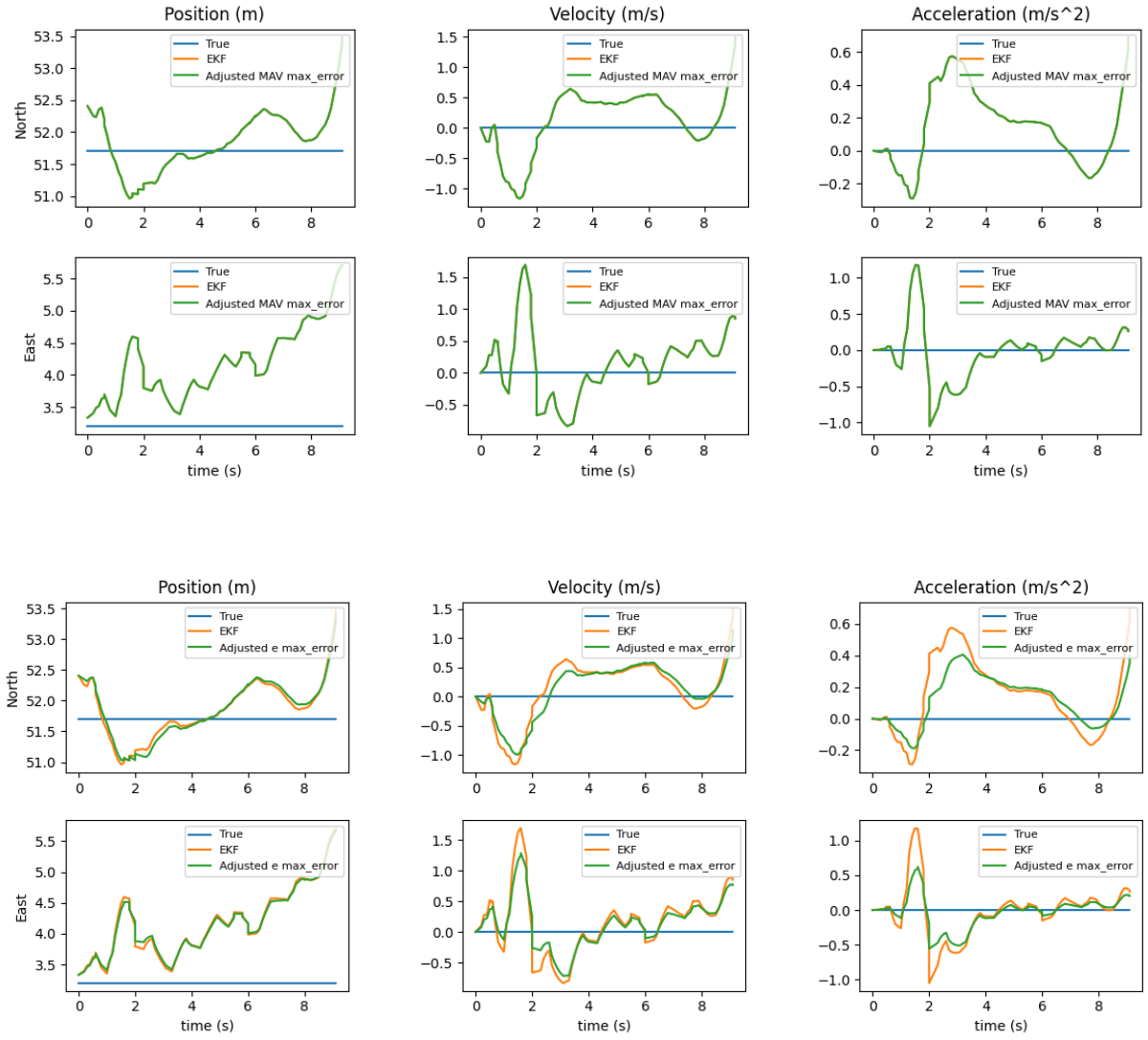
Appendix N. Result from Sample Dataset after orientation correction



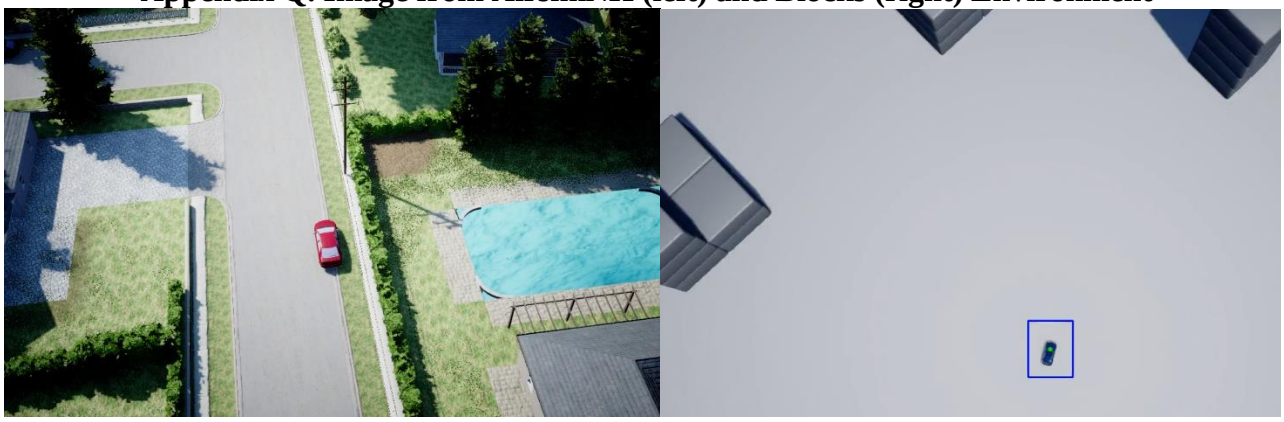
Appendix O. Result from Sample Dataset after timestamp correction



Appendix P. Results from different MAV state (top) and image coordinate (bottom) noise parameters



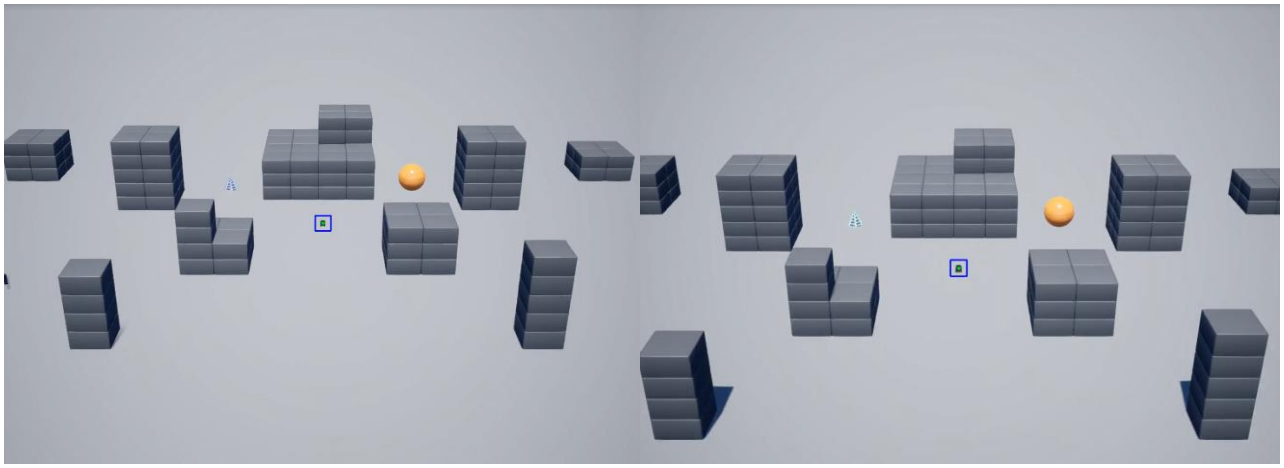
Appendix Q. Image from AirsimNH (left) and Blocks (right) Environment

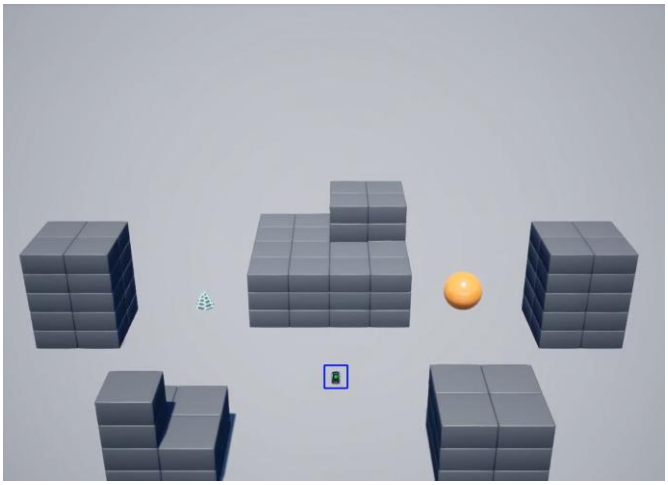


Appendix R. Shell script for automated 100 Airsim simulation

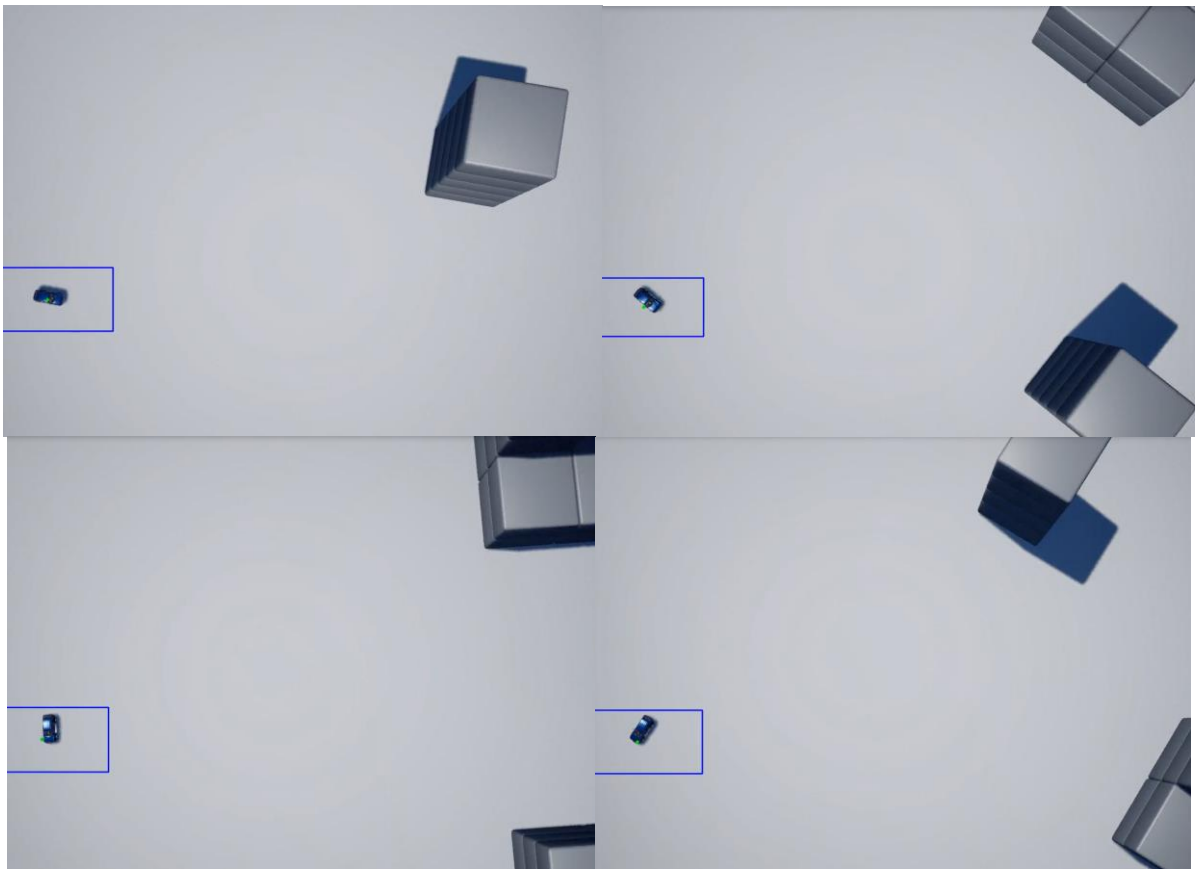
```
for i in {0..99}
do
echo Run $i
gnome-terminal -e 'sh -c ~/Airsim/Blocks/LinuxNoEditor/Blocks/Binaries/Linux/Blocks
-windowed'
sleep 10
python ~/Desktop/SRCE/dataset_generator/DATASET_circle_circle.py $i &
pid=$!
wait $pid
kill $(pidof Blocks)
done
```

Appendix S.1. Sample Images from Dataset (Scenario 0)

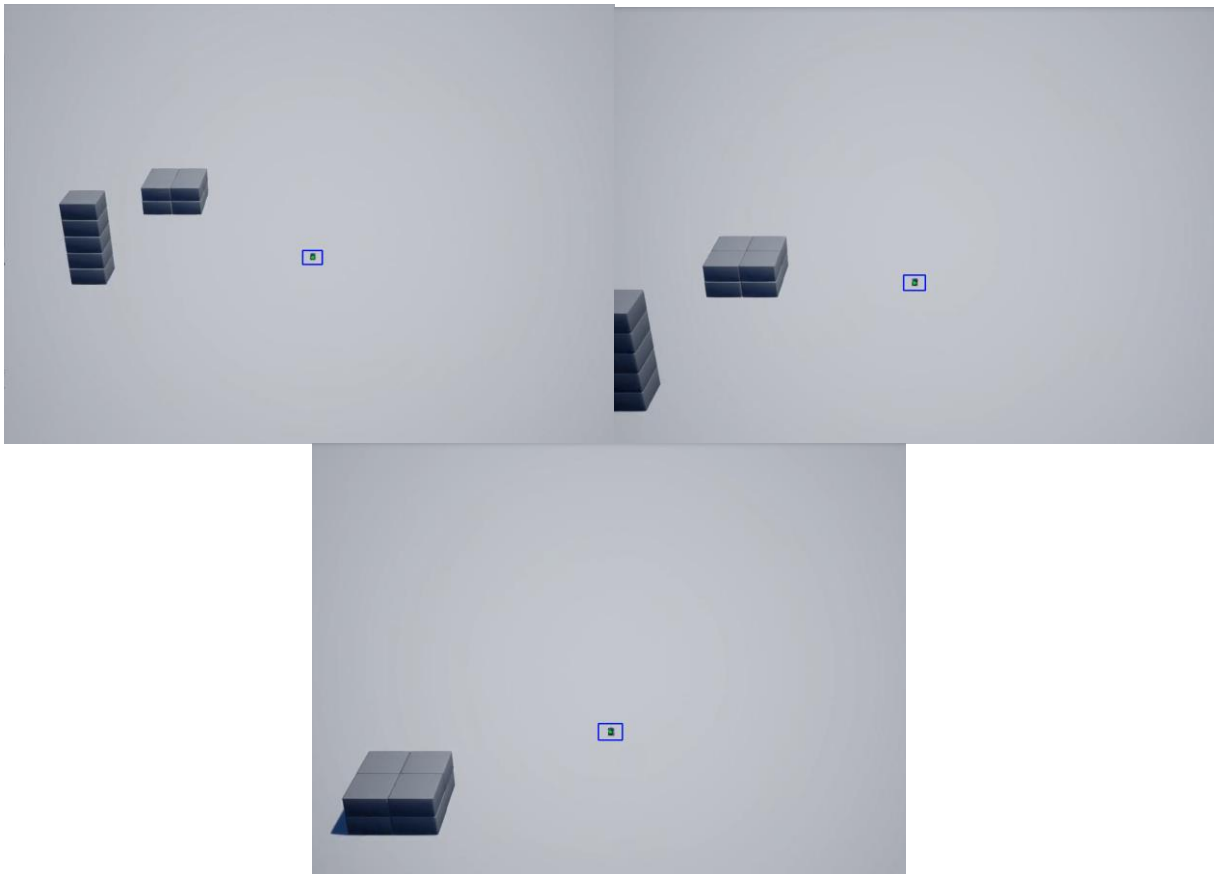




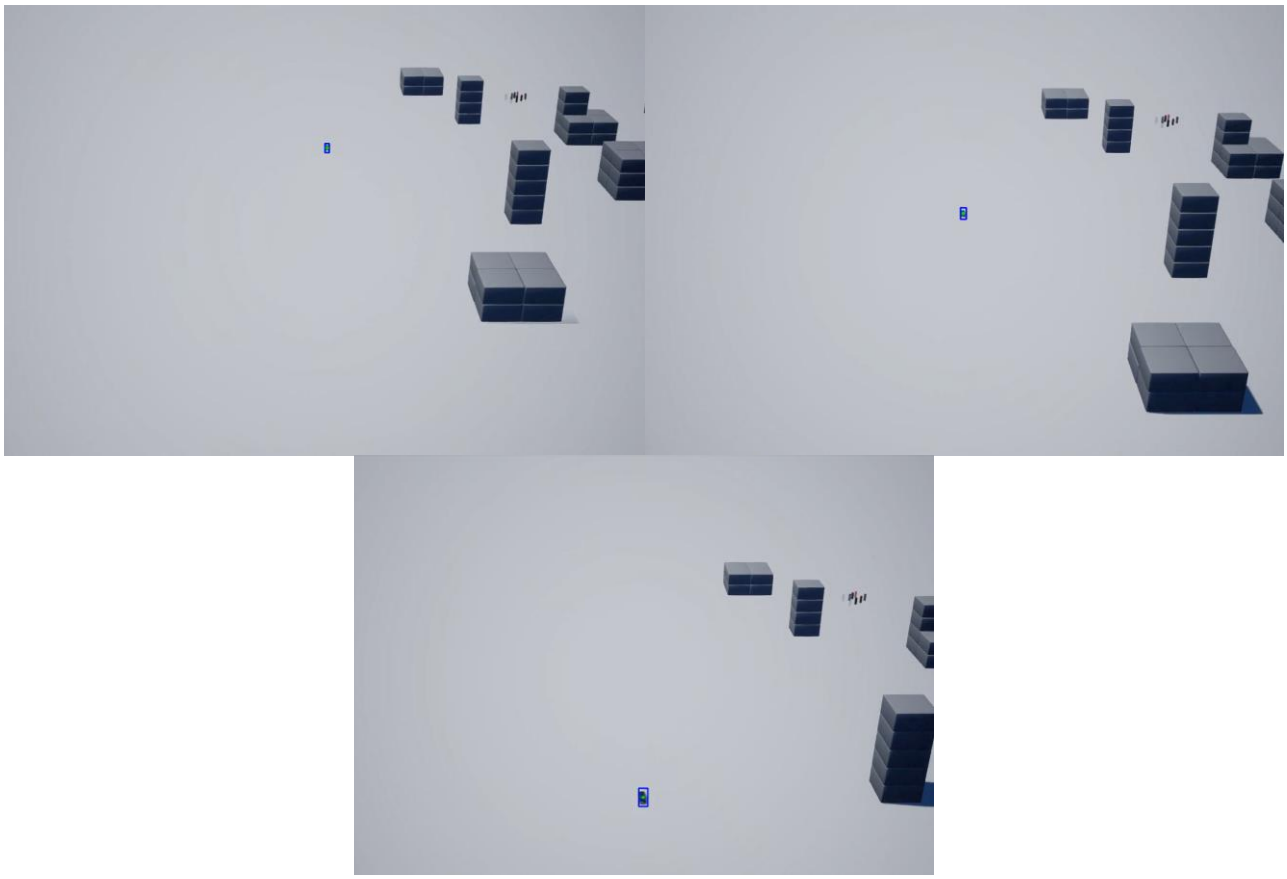
Appendix S.2. Sample Images from Dataset (Scenario 1)



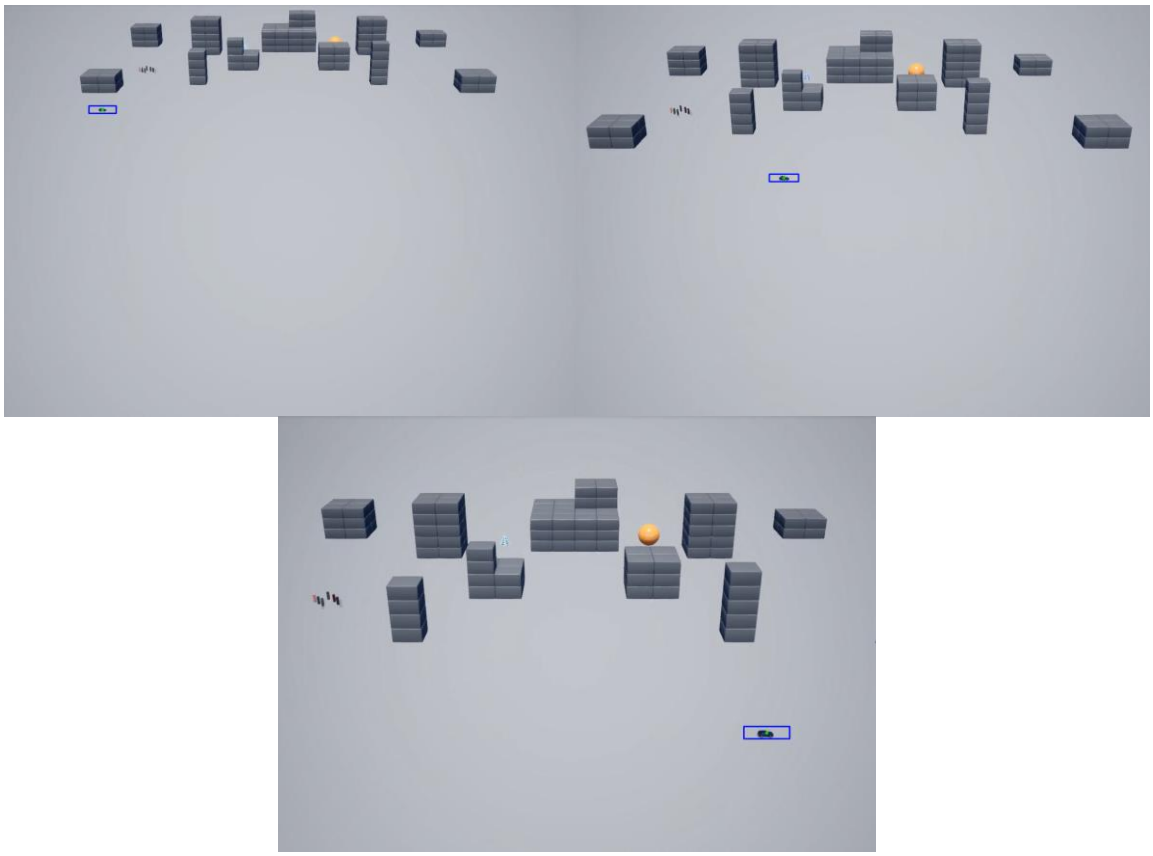
Appendix S.3. Sample Images from Dataset (Scenario 2)



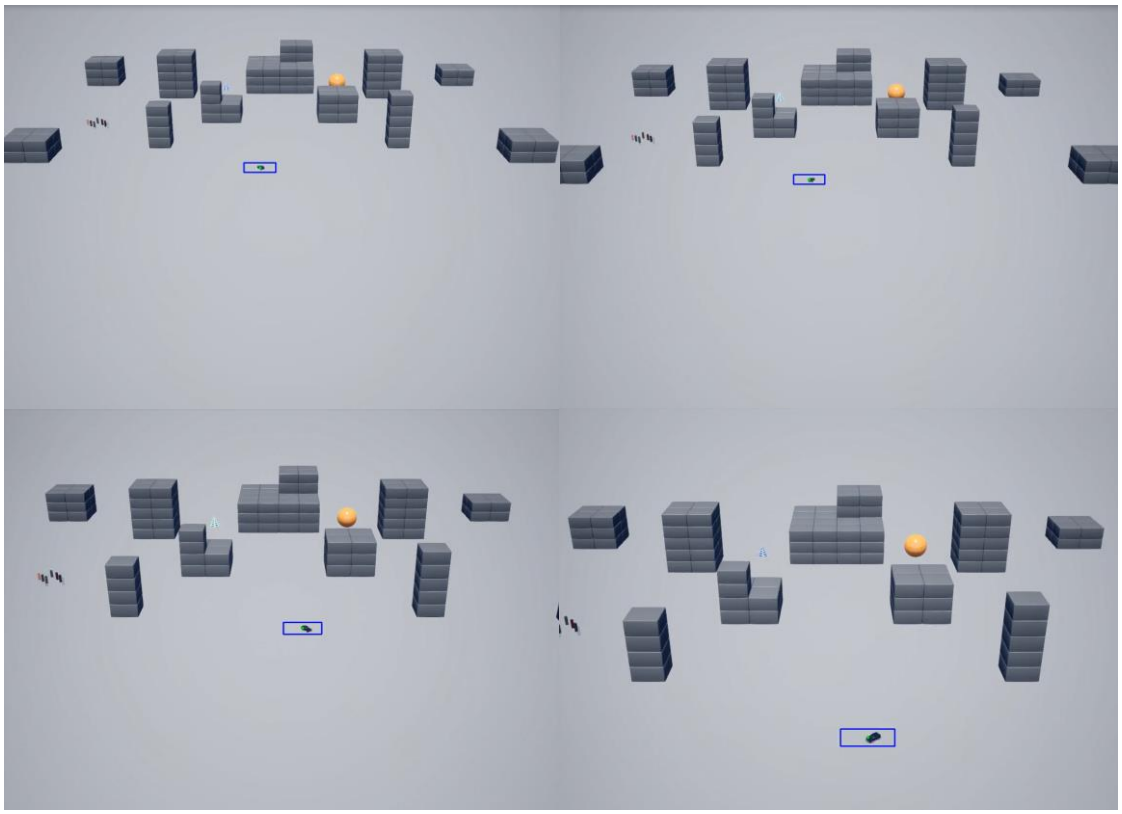
Appendix S.4. Sample Images from Dataset (Scenario 3)



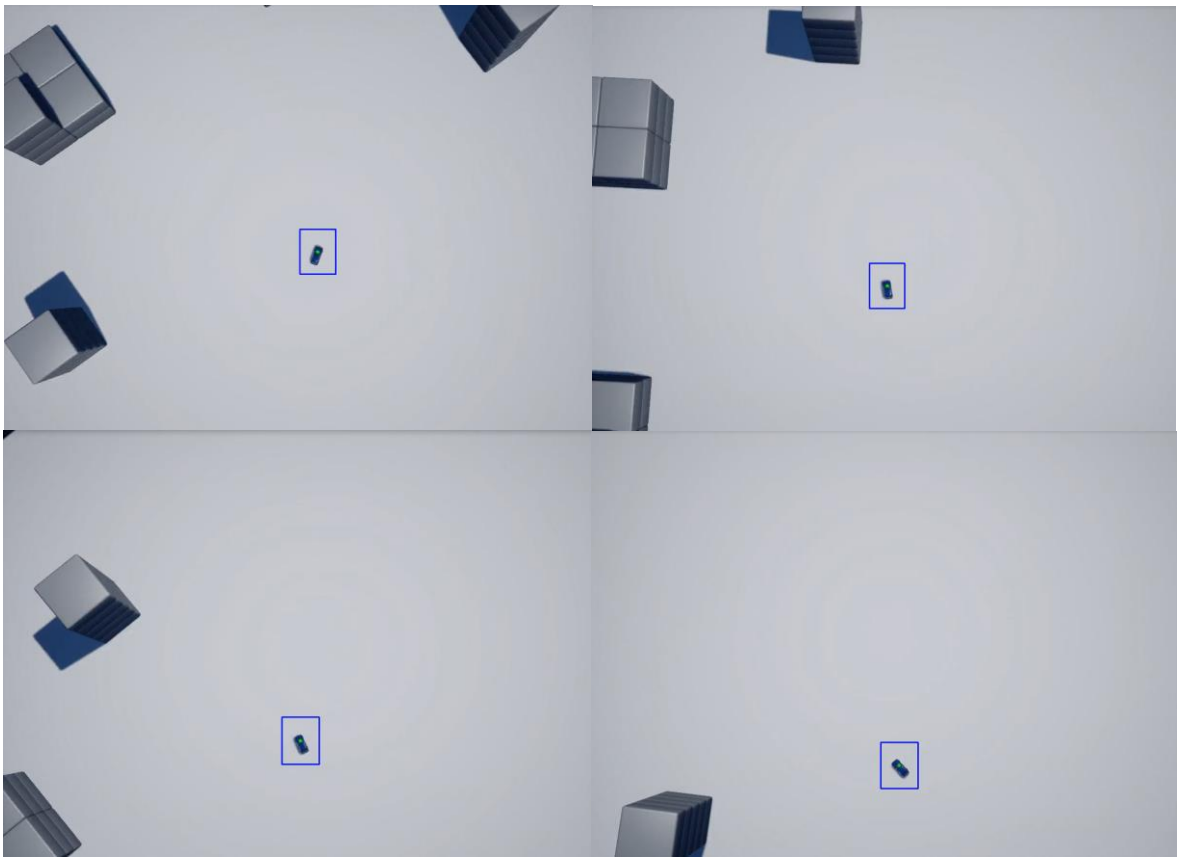
Appendix S.5. Sample Images from Dataset (Scenario 4)



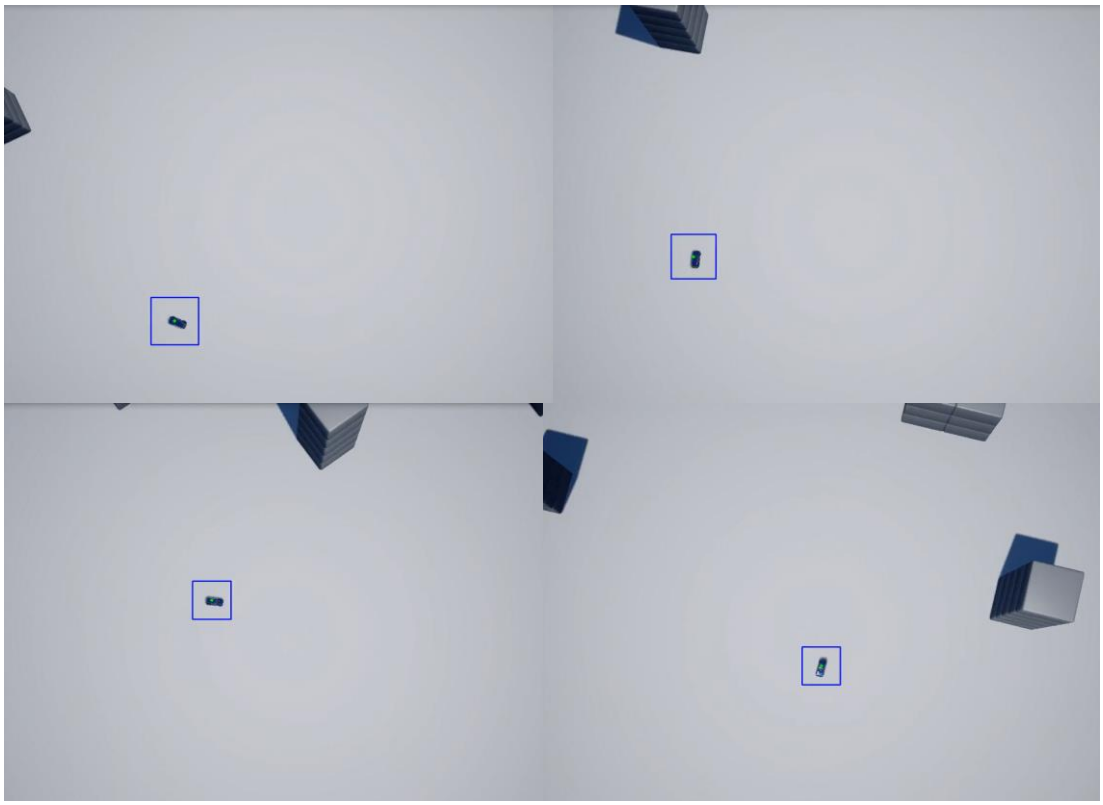
Appendix S.6. Sample Images from Dataset (Scenario 5)



Appendix S.7. Sample Images from Dataset (Scenario 6)

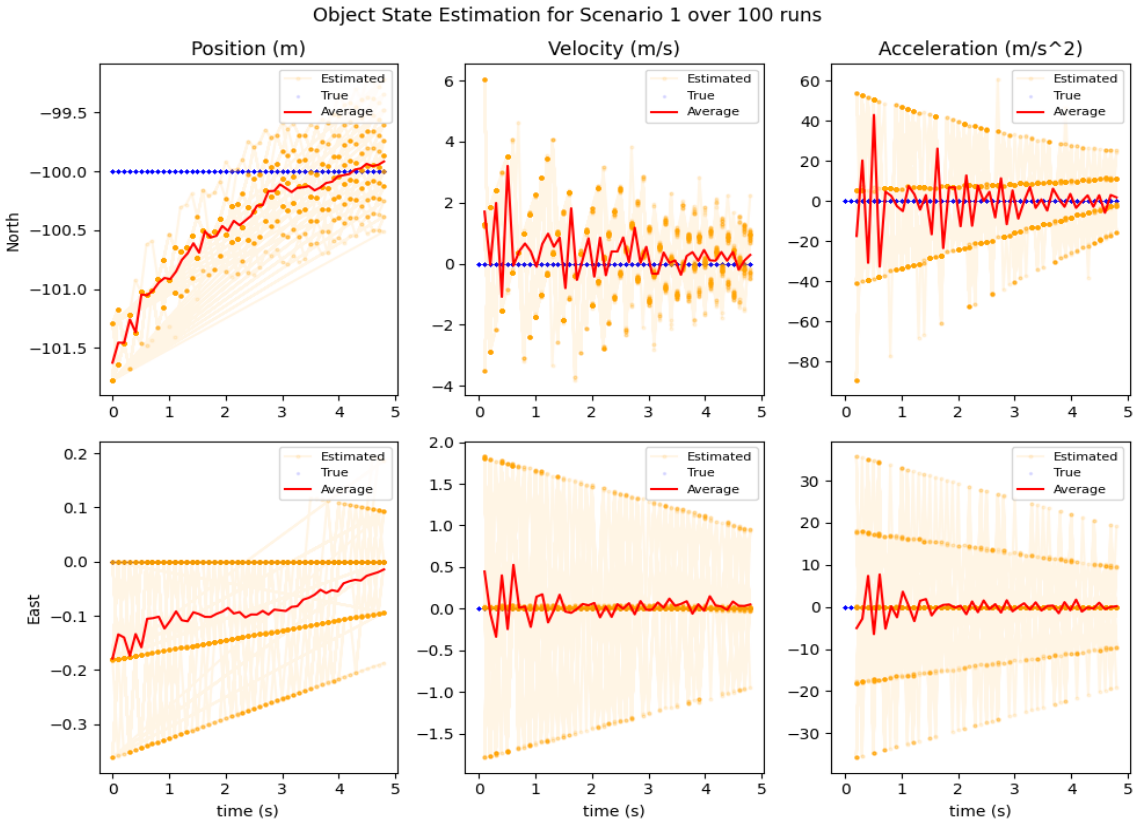


Appendix S.8. Sample Images from Dataset (Scenario 7)



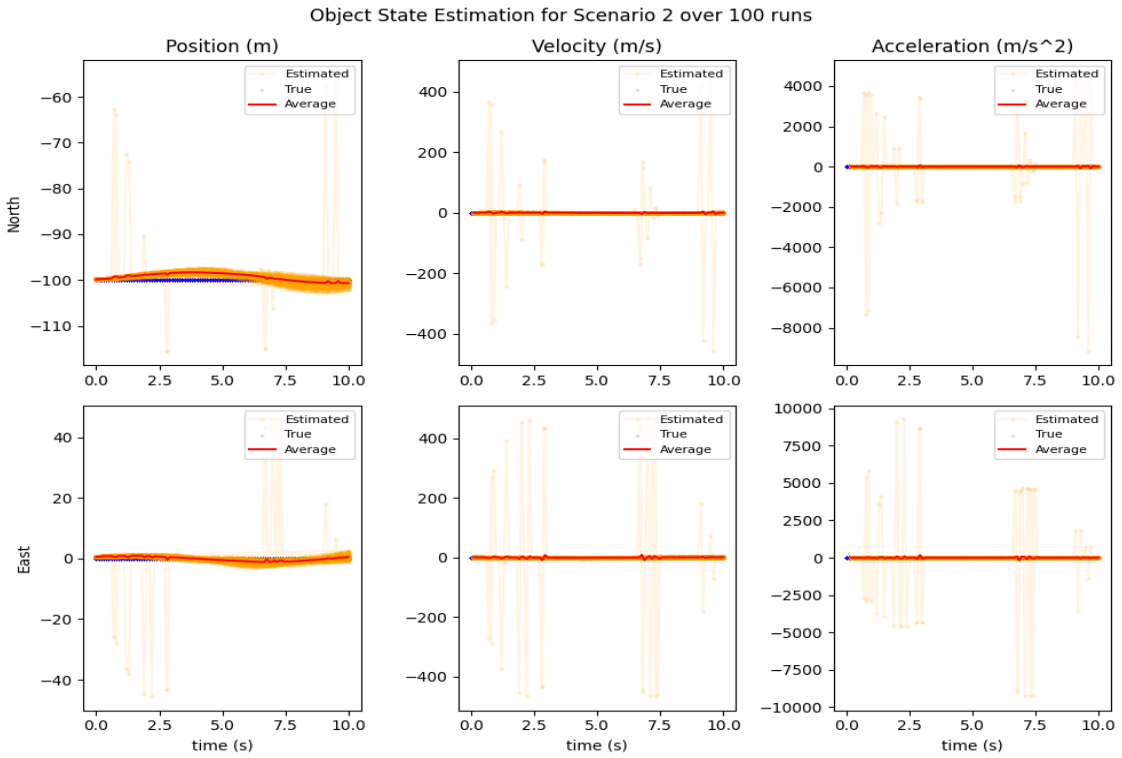
Appendix T.1. Unfiltered State Estimate Plots (Scenario 0)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



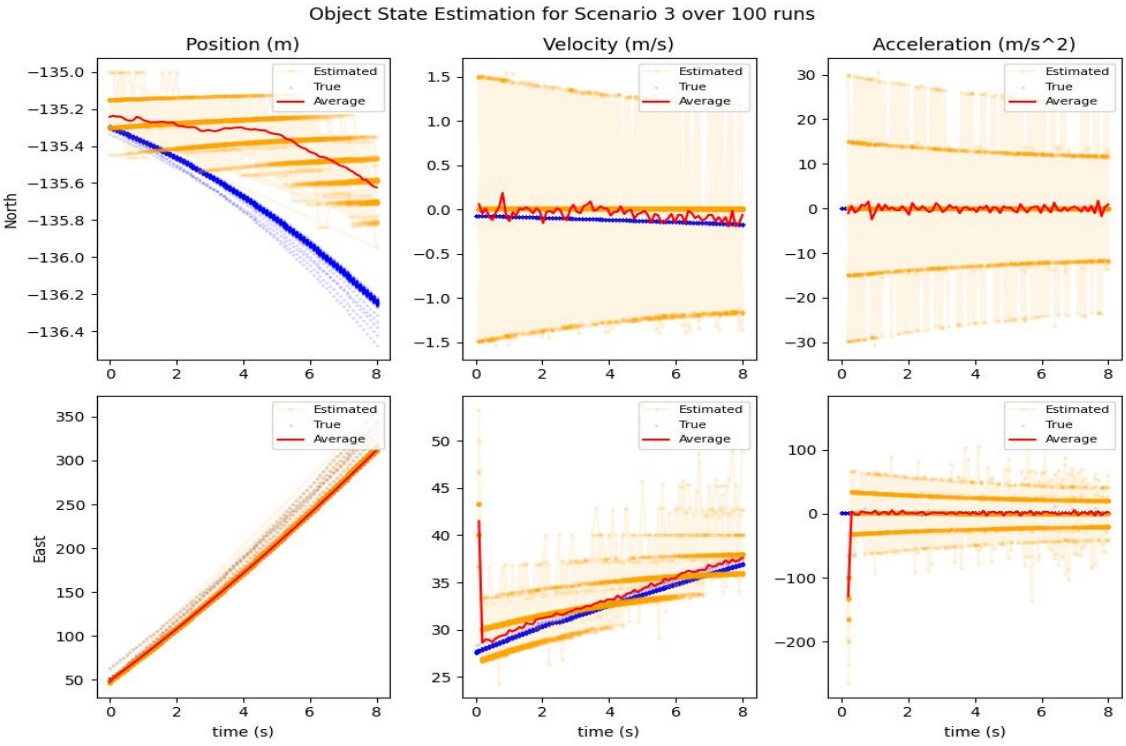
Appendix T.2. Unfiltered State Estimate Plots (Scenario 1)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



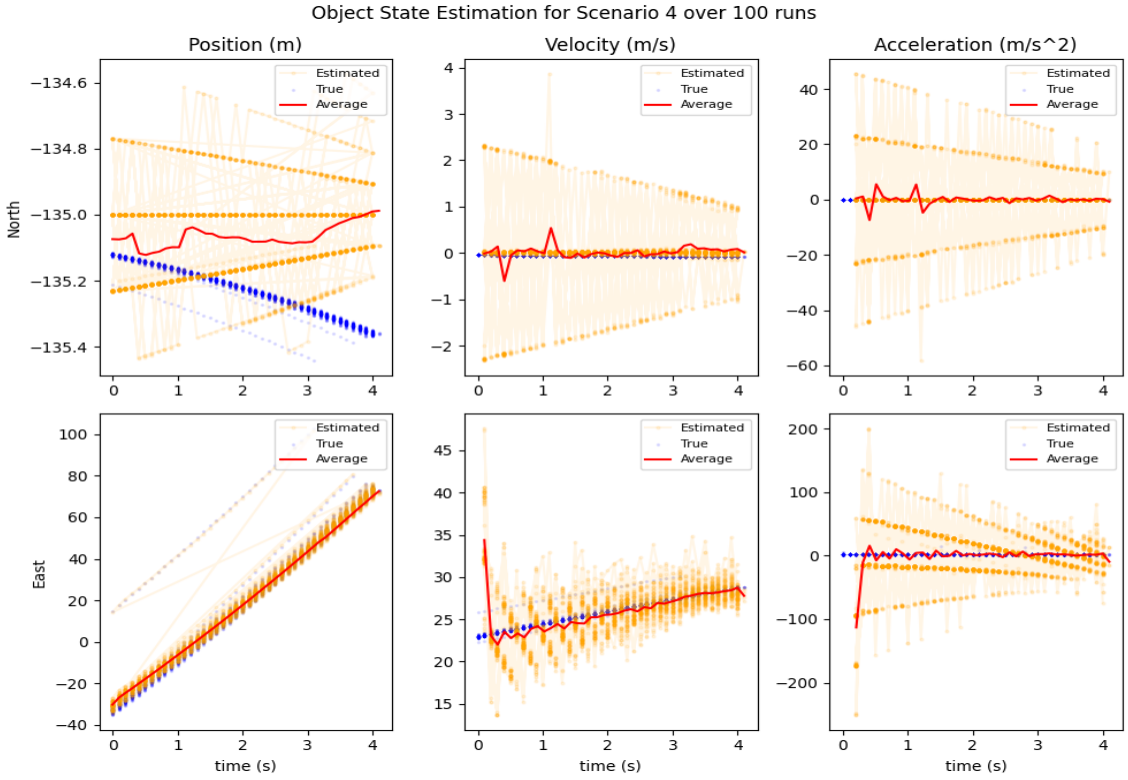
Appendix T.3. Unfiltered State Estimate Plots (Scenario 2)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



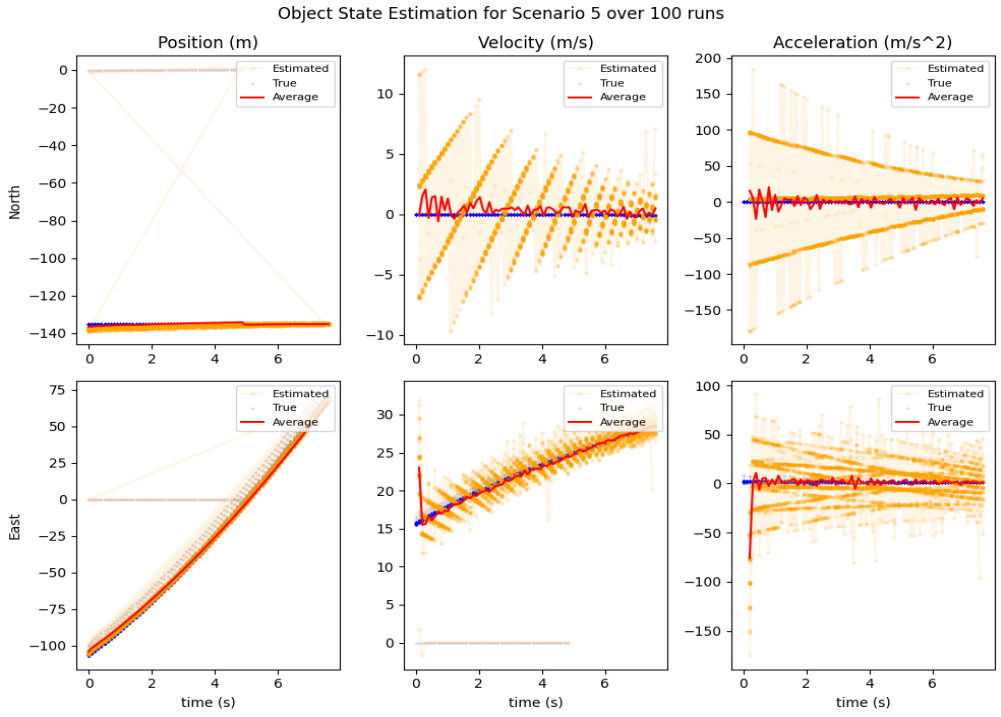
Appendix T.4. Unfiltered State Estimate Plots (Scenario 3)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



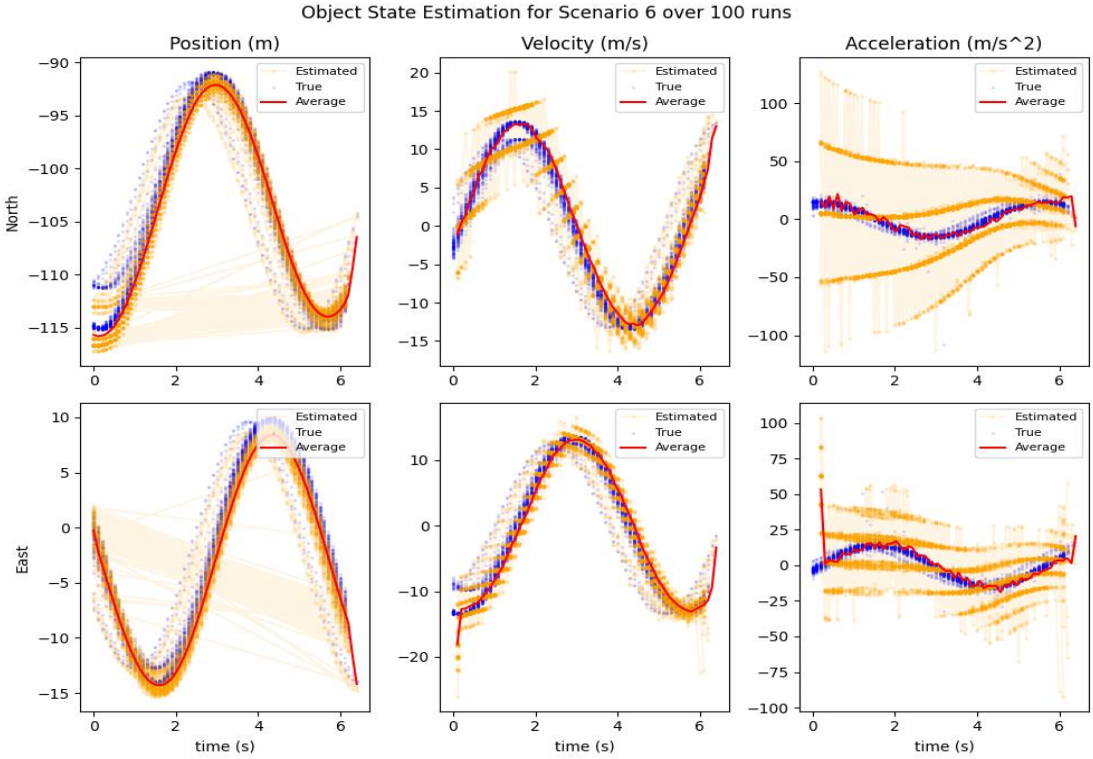
Appendix T.5. Unfiltered State Estimate Plots (Scenario 4)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



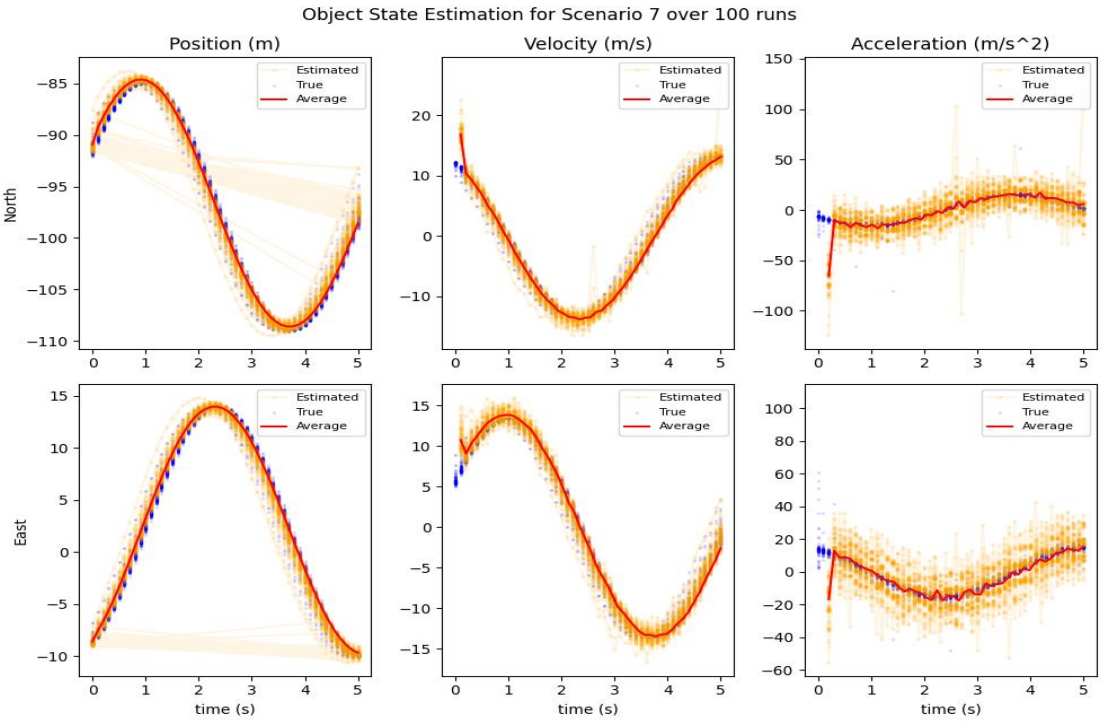
Appendix T.6. Unfiltered State Estimate Plots (Scenario 5)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



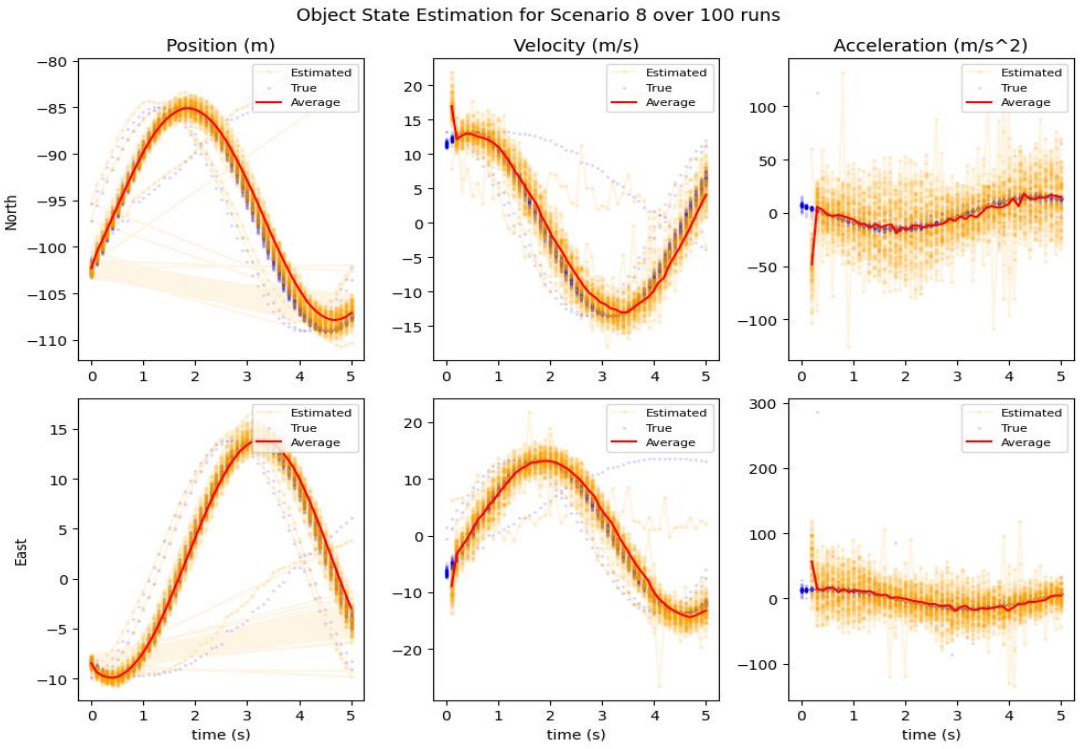
Appendix T.7. Unfiltered State Estimate Plots (Scenario 6)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



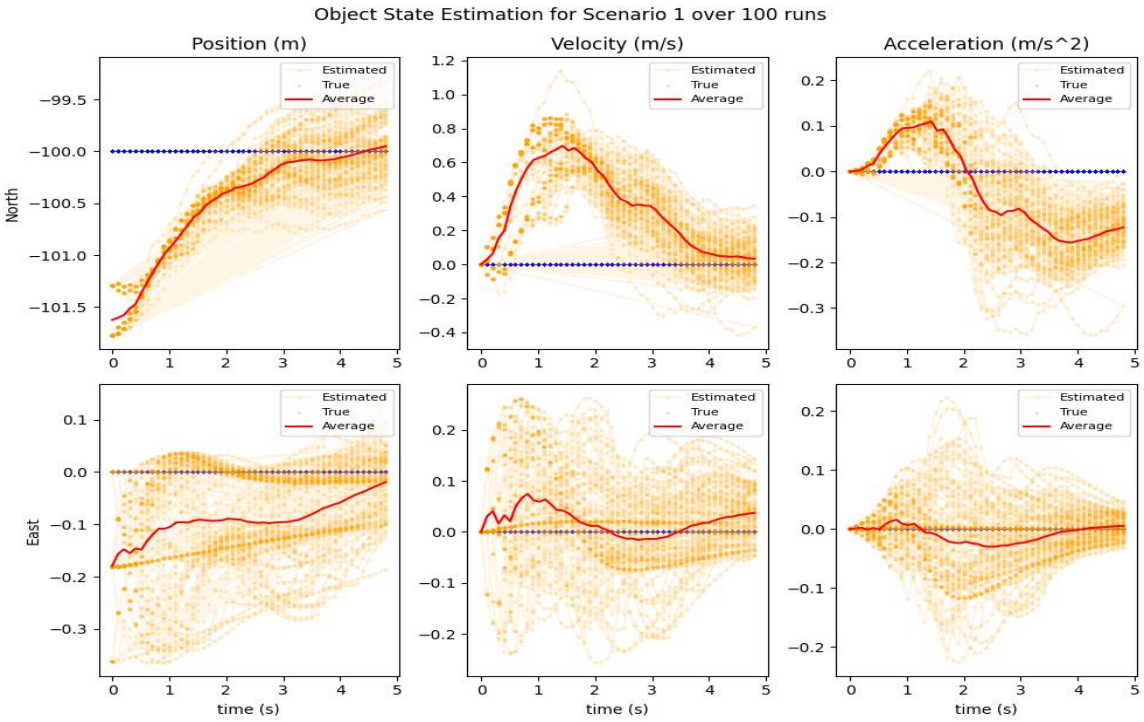
Appendix T.8. Unfiltered State Estimate Plots (Scenario 7)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



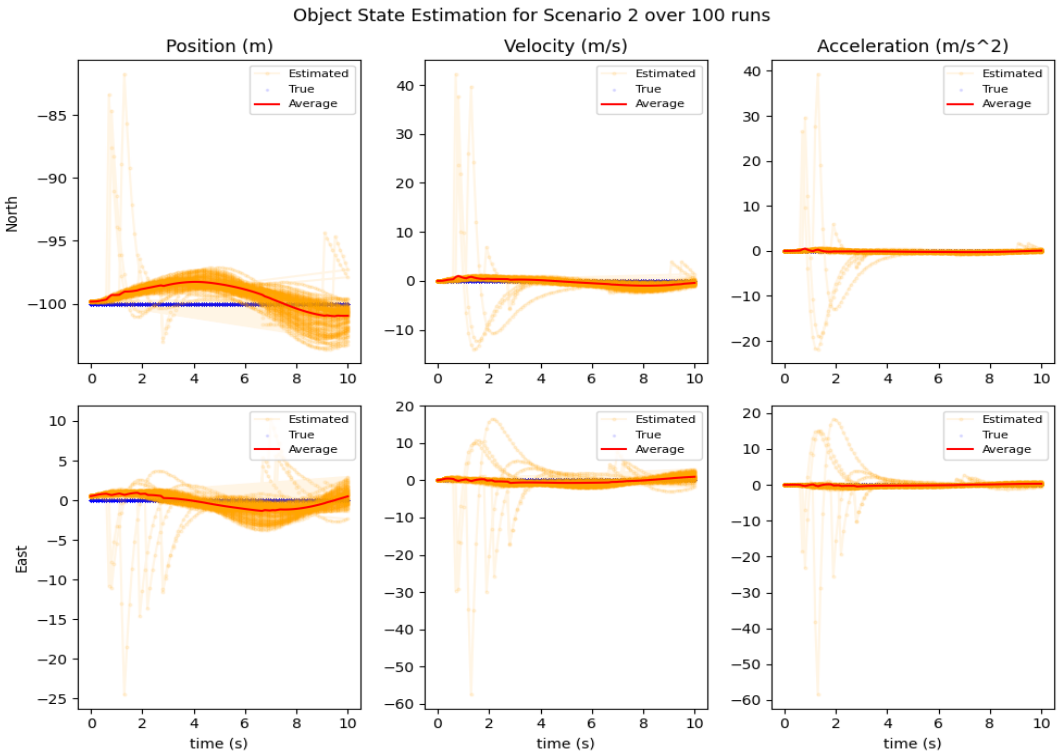
Appendix U.1. Baseline EKF State Estimate Plots (Scenario 0)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



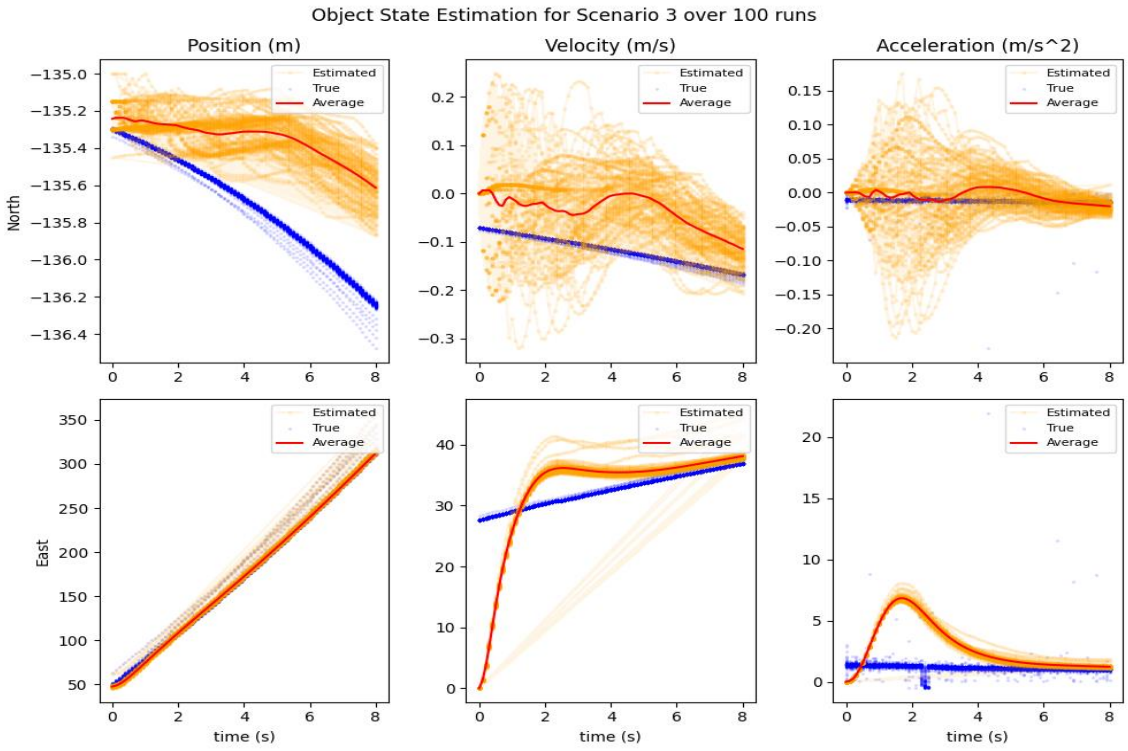
Appendix U.2. Baseline EKF State Estimate Plots (Scenario 1)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



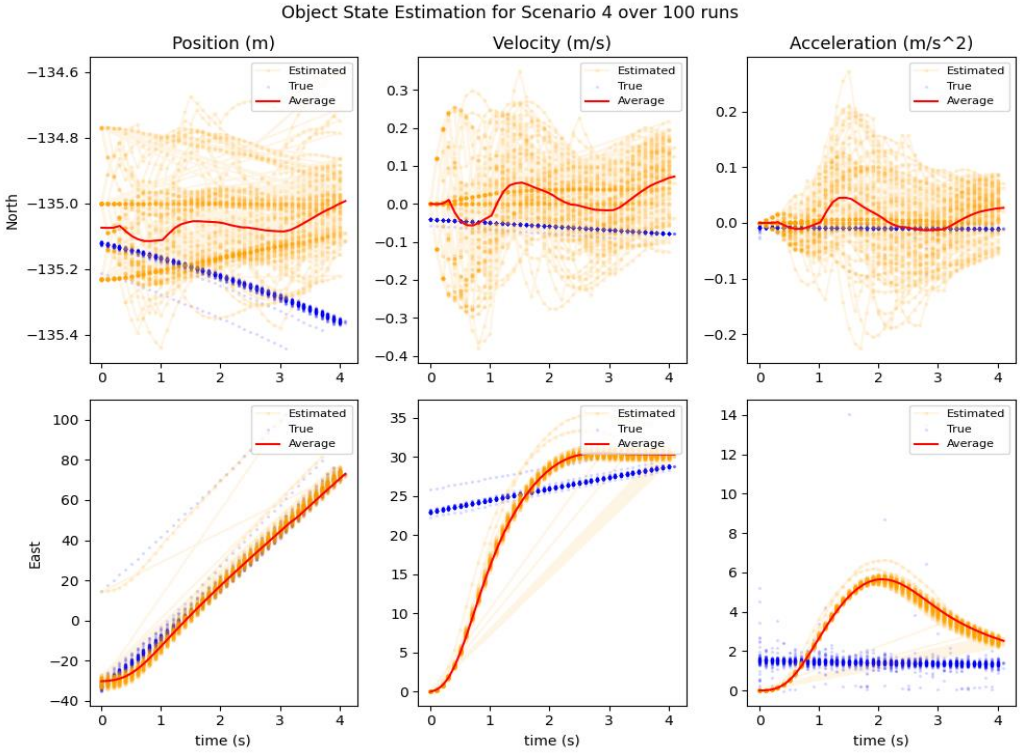
Appendix U.3. Baseline EKF State Estimate Plots (Scenario 2)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



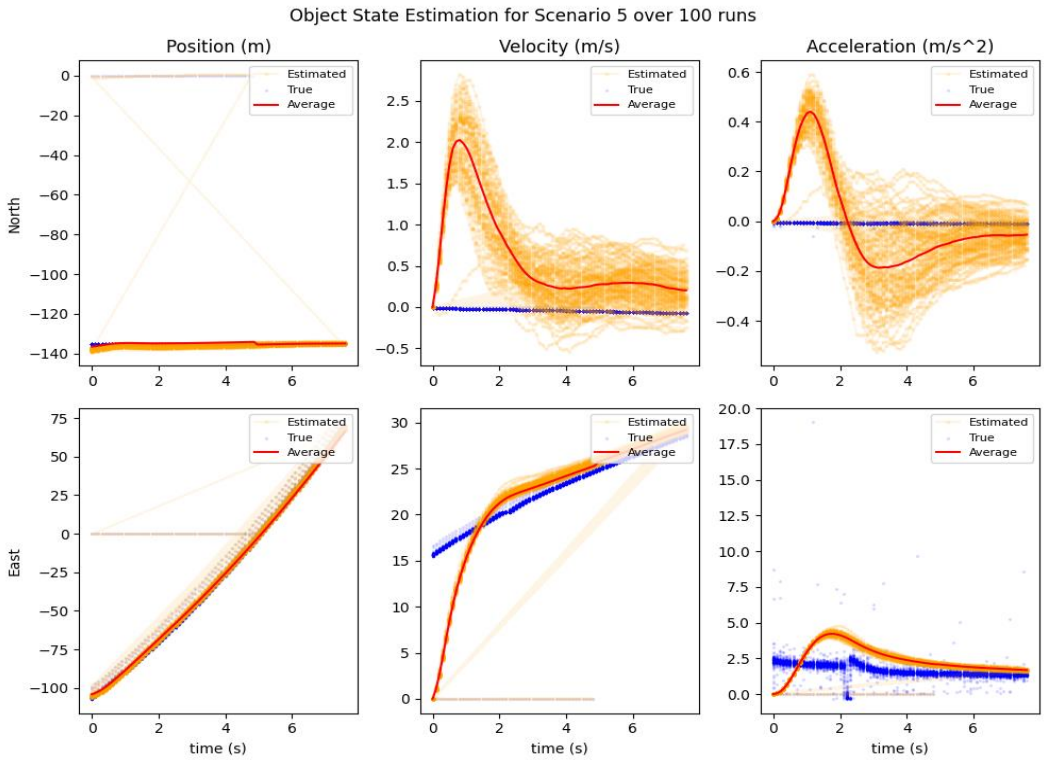
Appendix U.4. Baseline EKF State Estimate Plots (Scenario 3)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



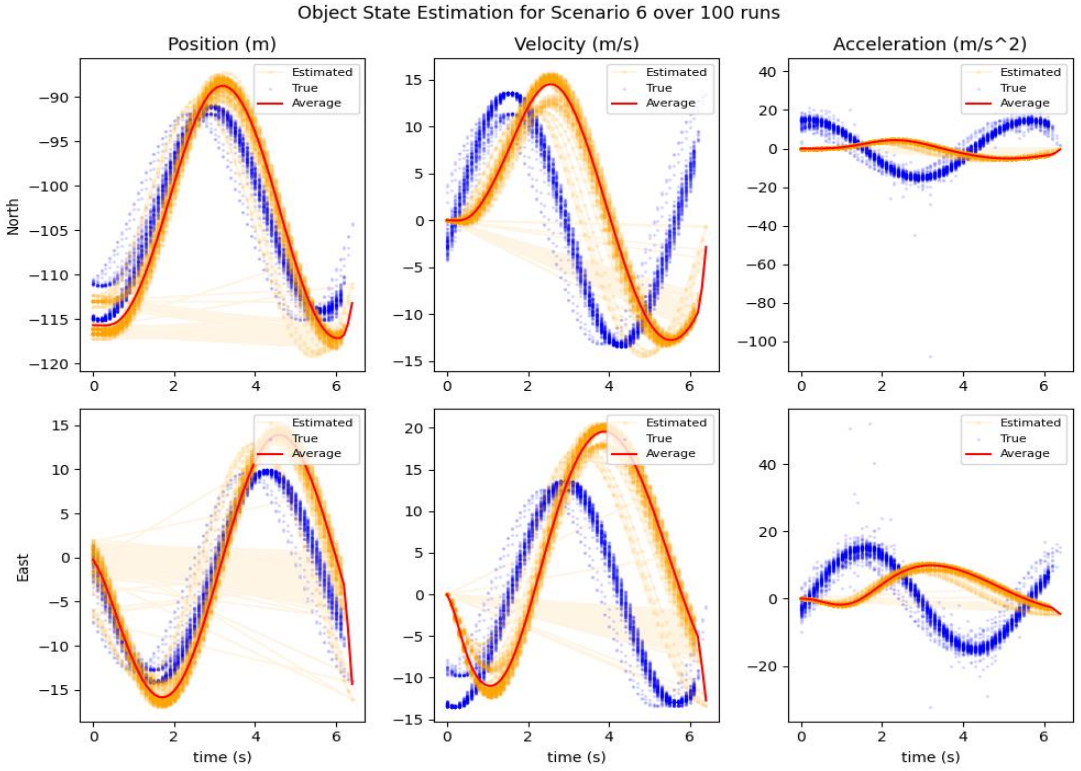
Appendix U.5. Baseline EKF State Estimate Plots (Scenario 4)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



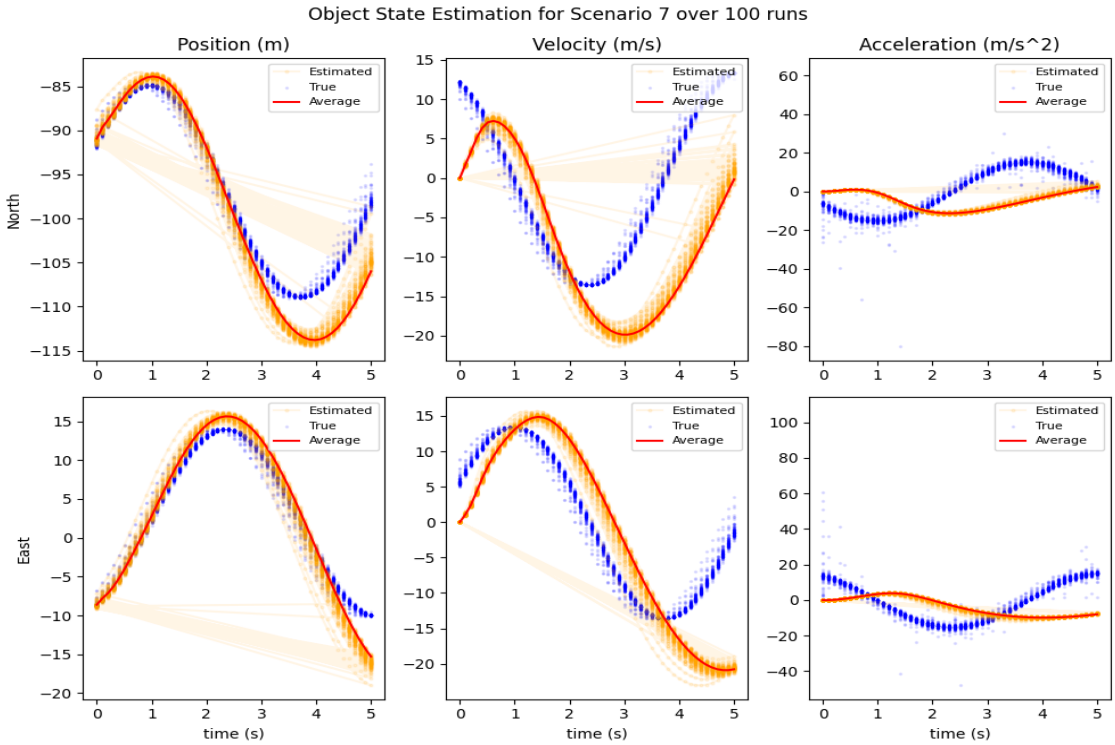
Appendix U.6. Baseline EKF State Estimate Plots (Scenario 5)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



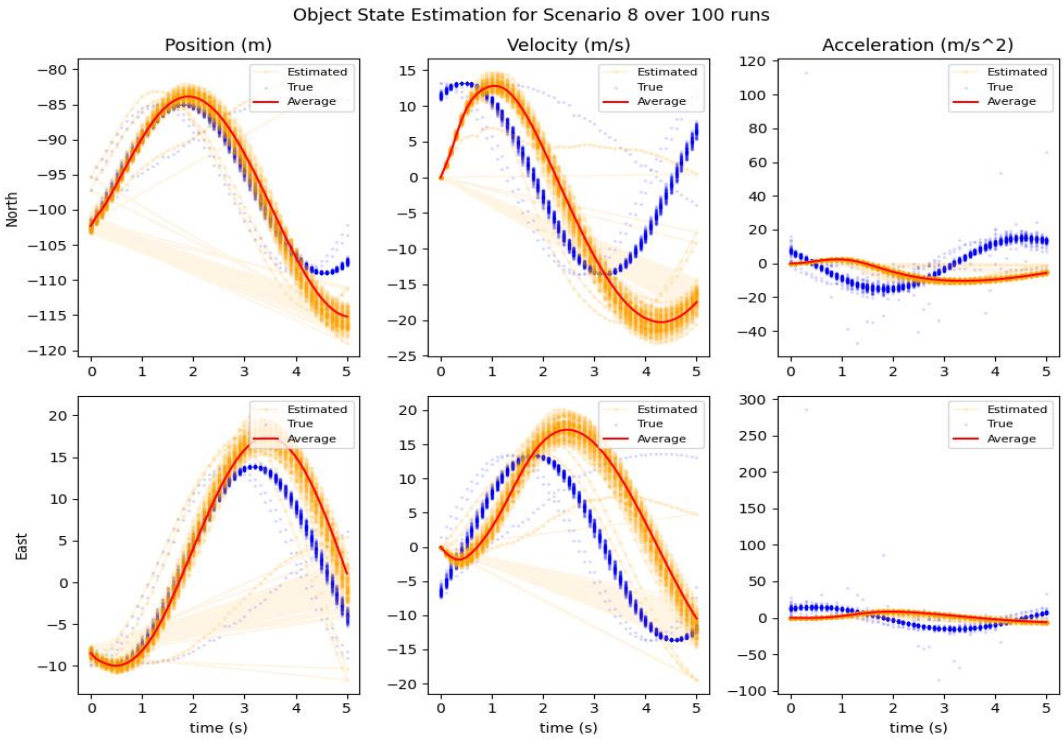
Appendix U.7. Baseline EKF State Estimate Plots (Scenario 6)

*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario

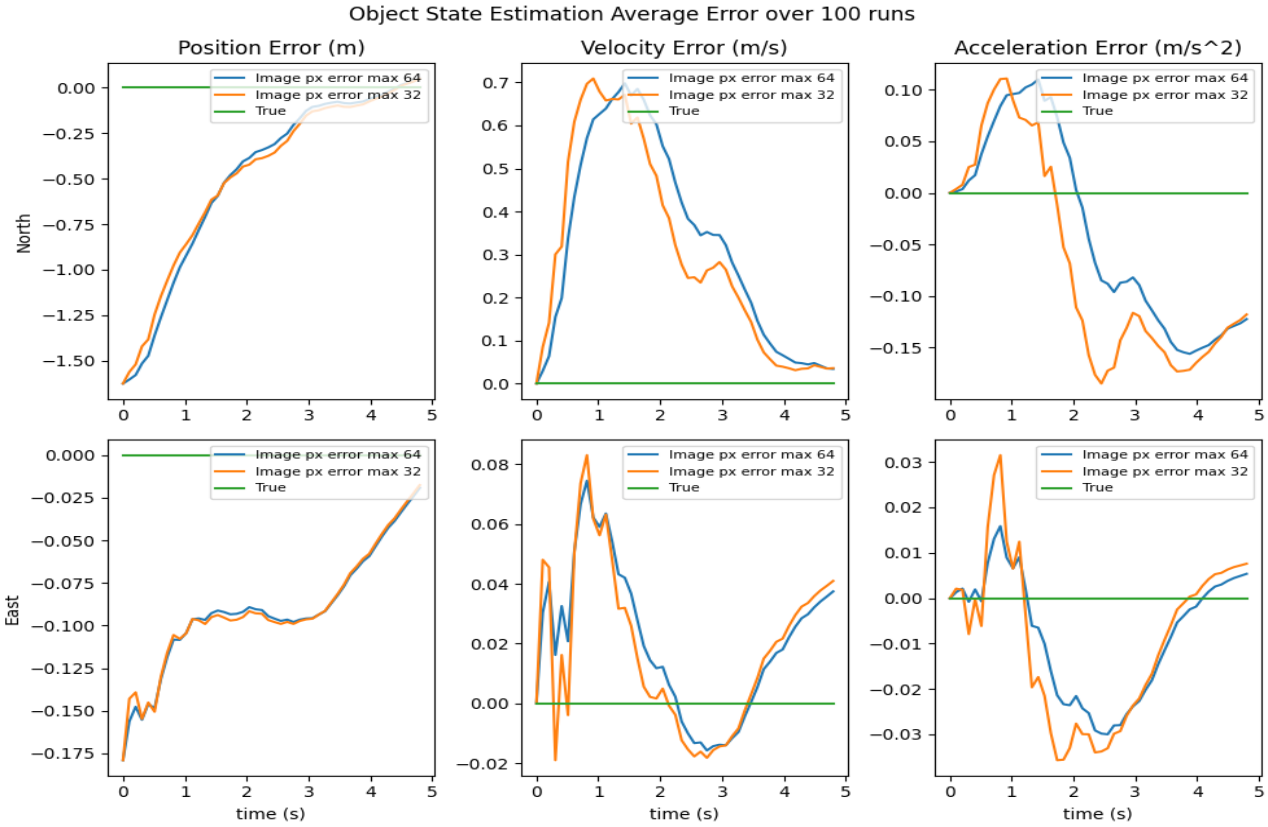


Appendix U.8. Baseline EKF State Estimate Plots (Scenario 7)

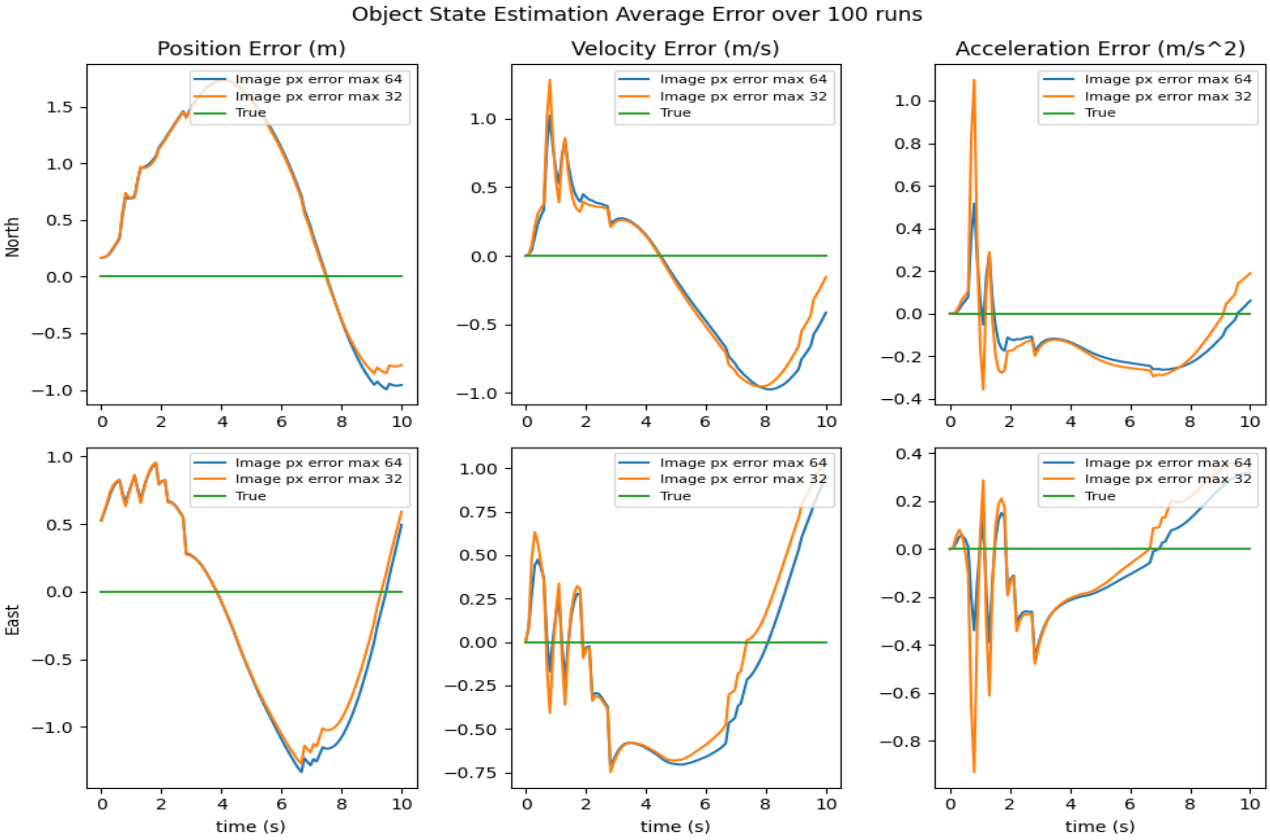
*Note that plot title was incorrectly labelled with the scenario number 1 larger than the actual scenario



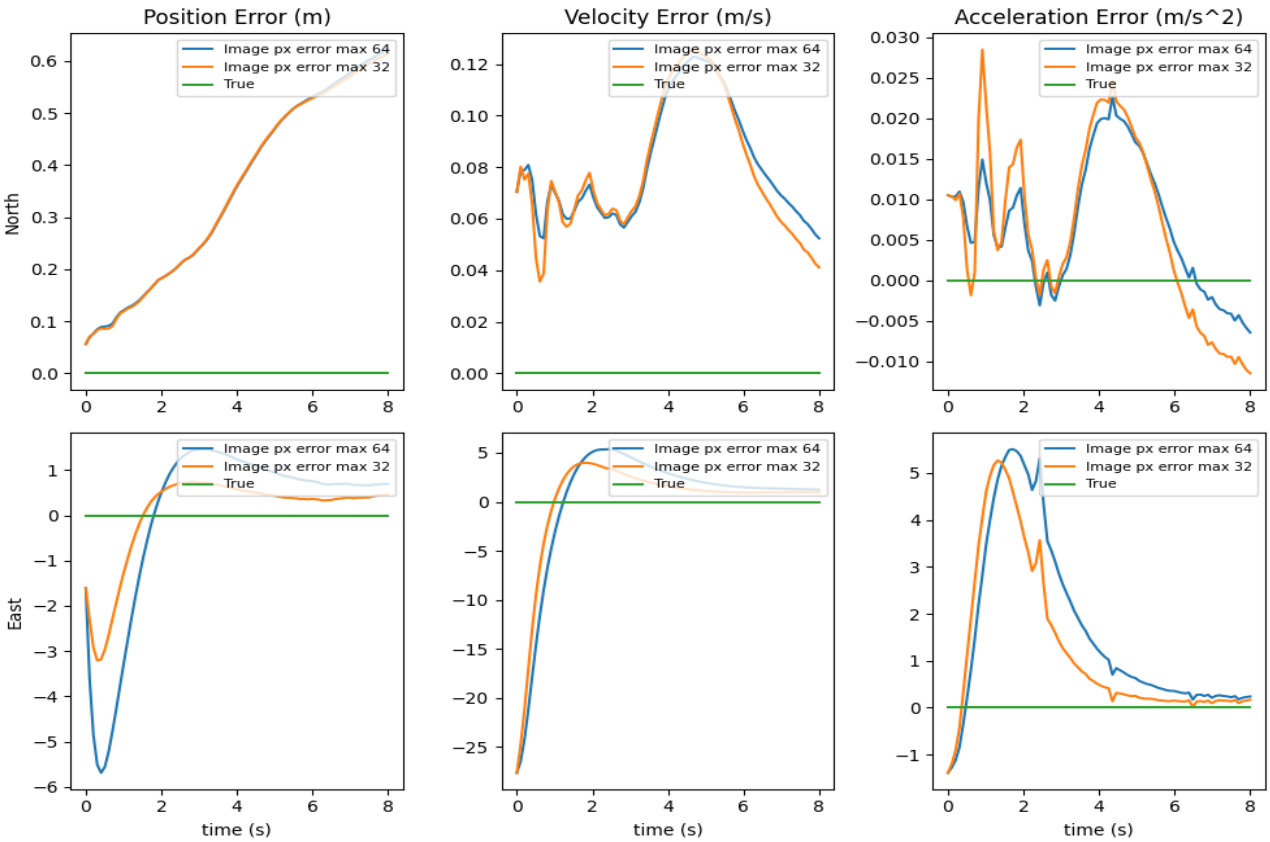
Appendix V.1. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 0)



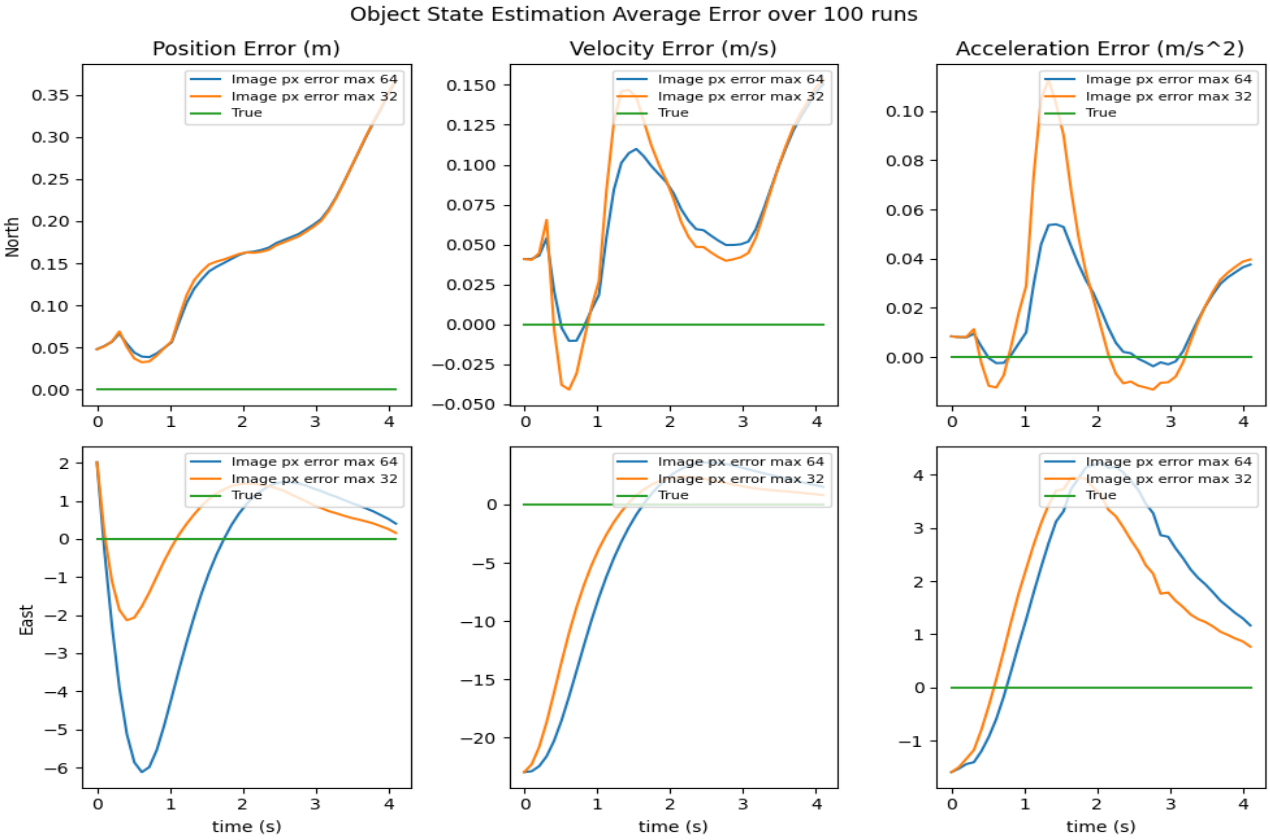
Appendix V.2. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 1)



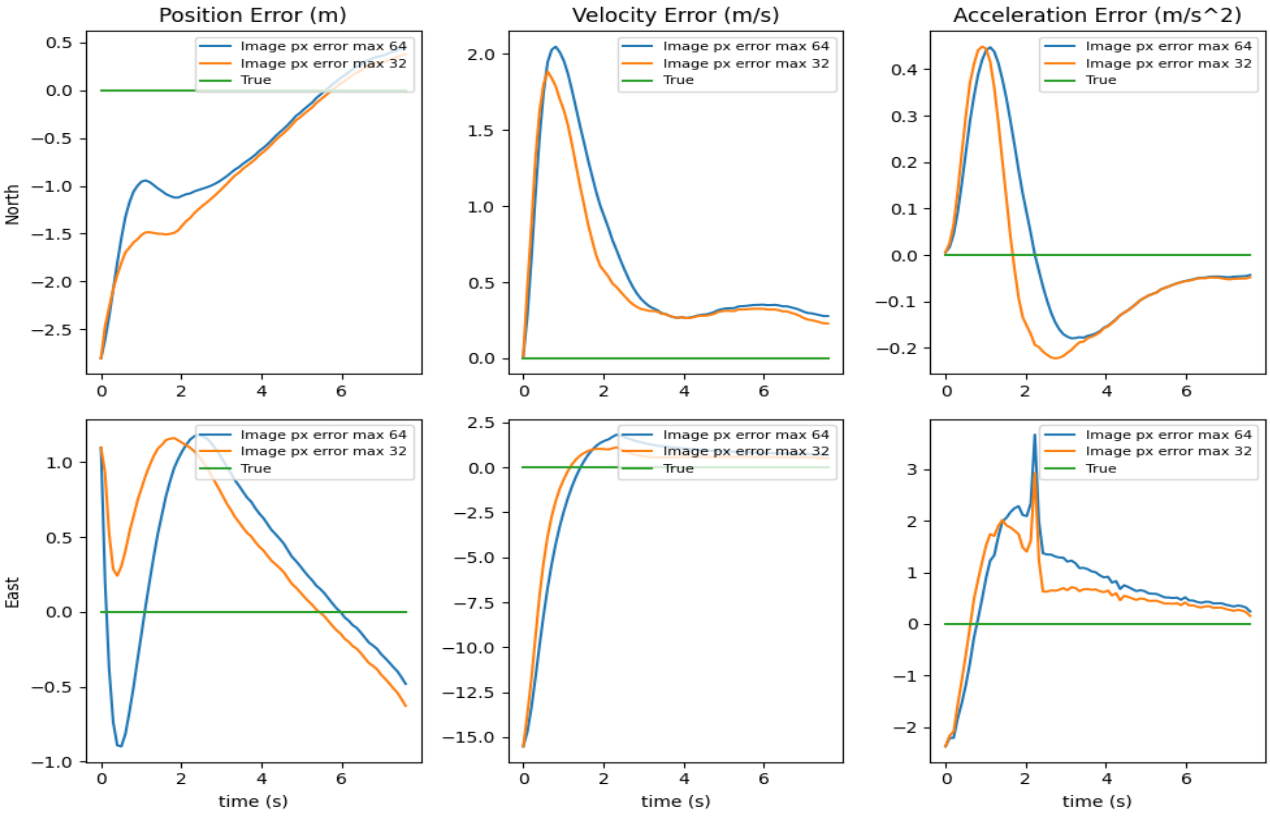
Appendix V.3. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 2)



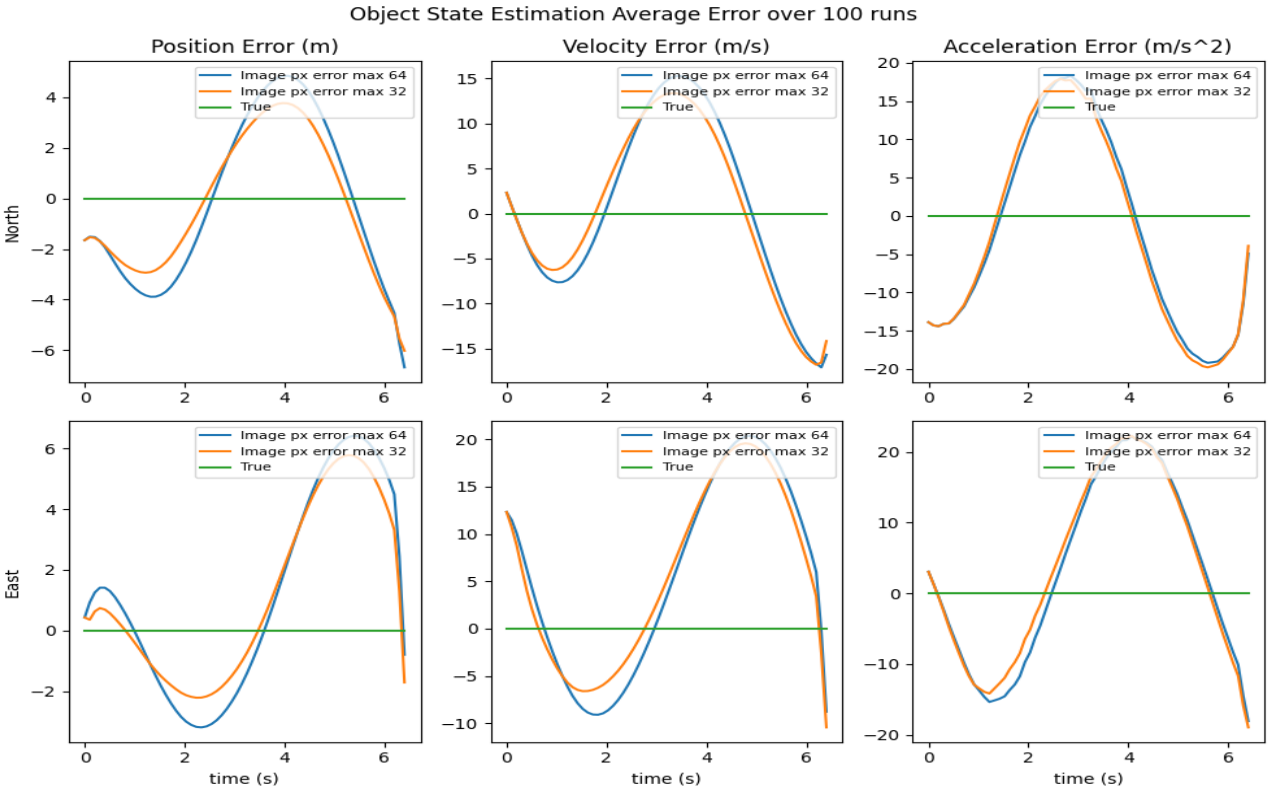
Appendix V.4. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 3)



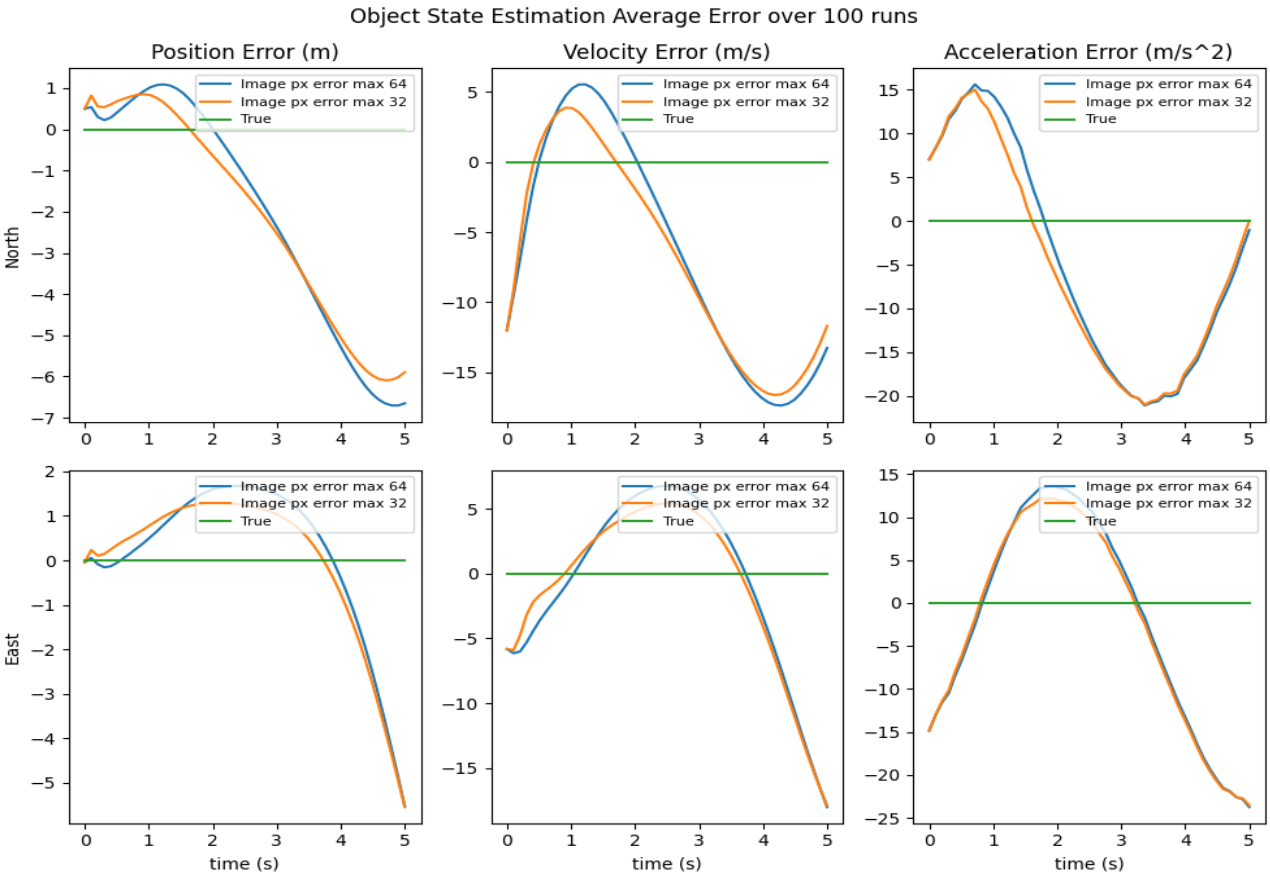
Appendix V.5. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 4)



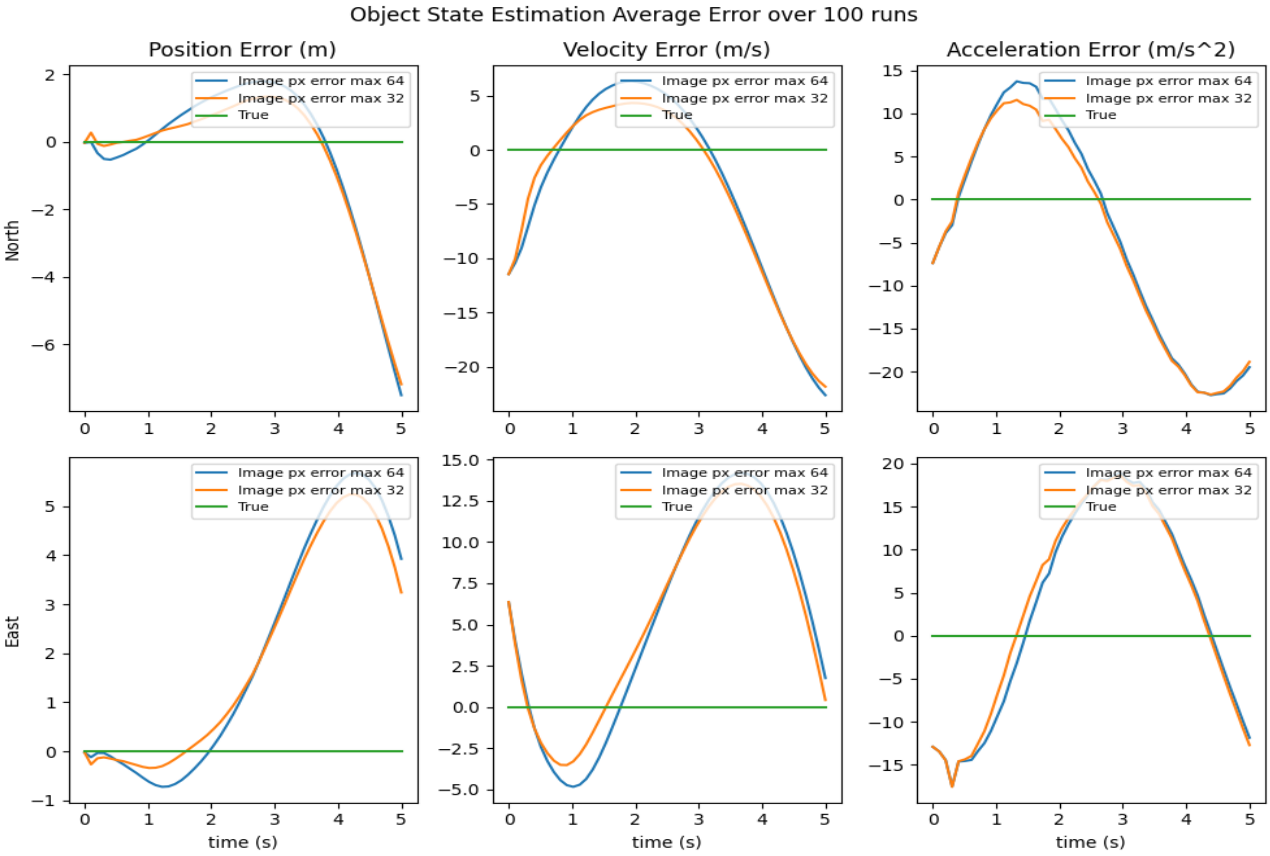
Appendix V.6. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 5)



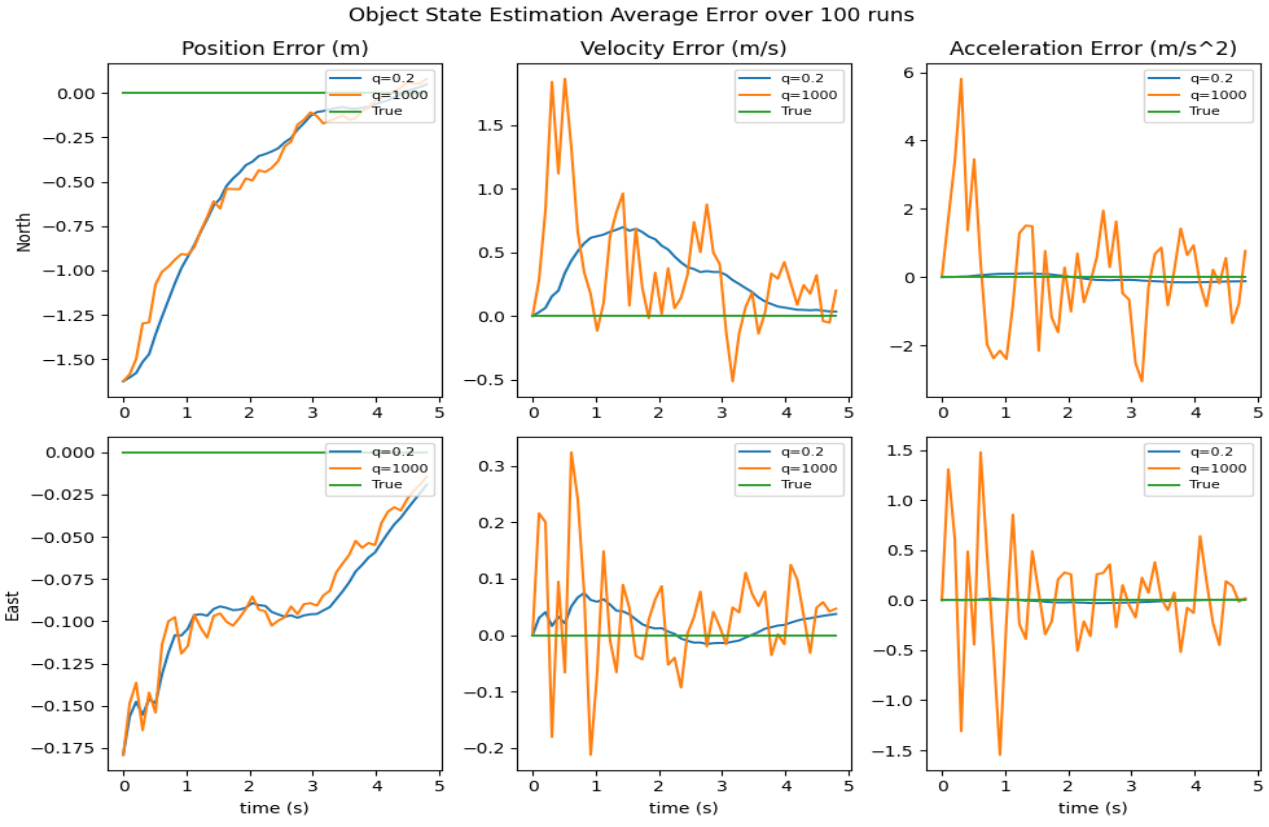
Appendix V.7. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 6)



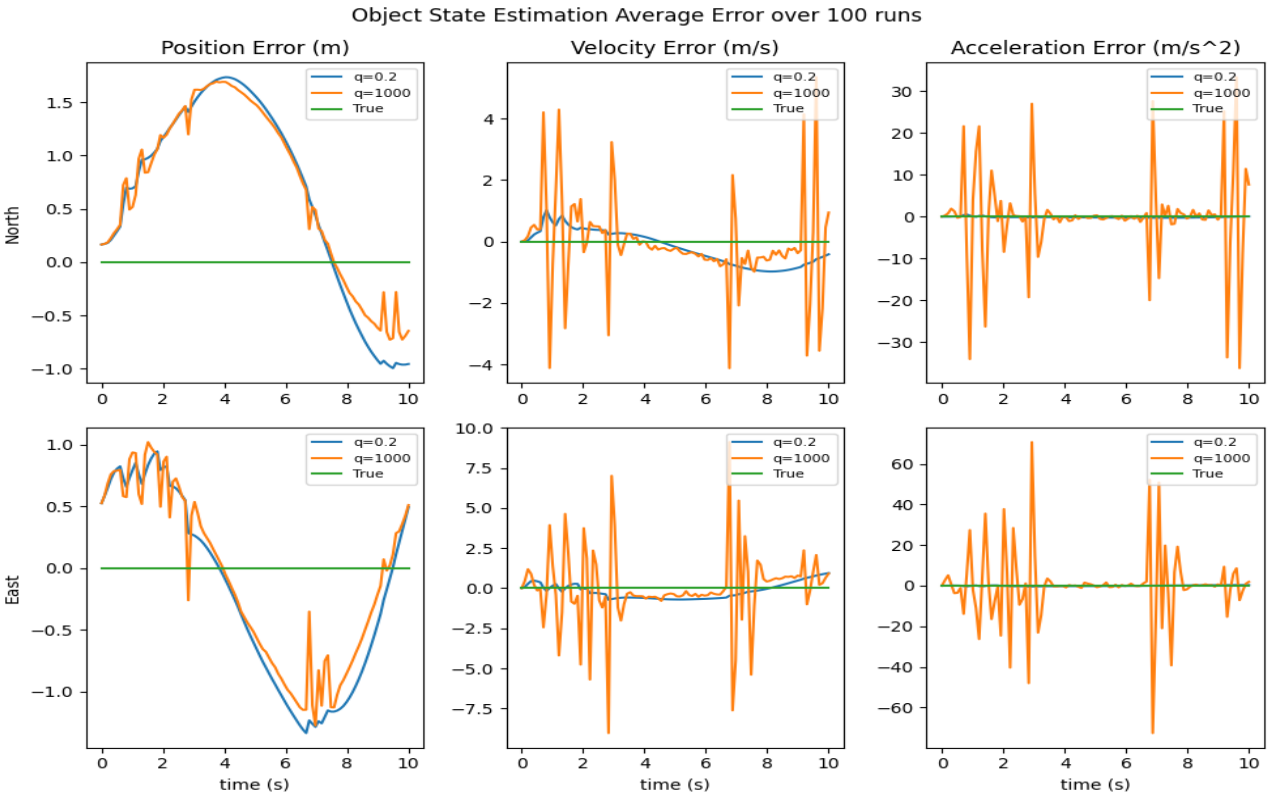
Appendix V.8. Baseline EKF State Estimation Average Error with different pixel noise parameter (Scenario 7)



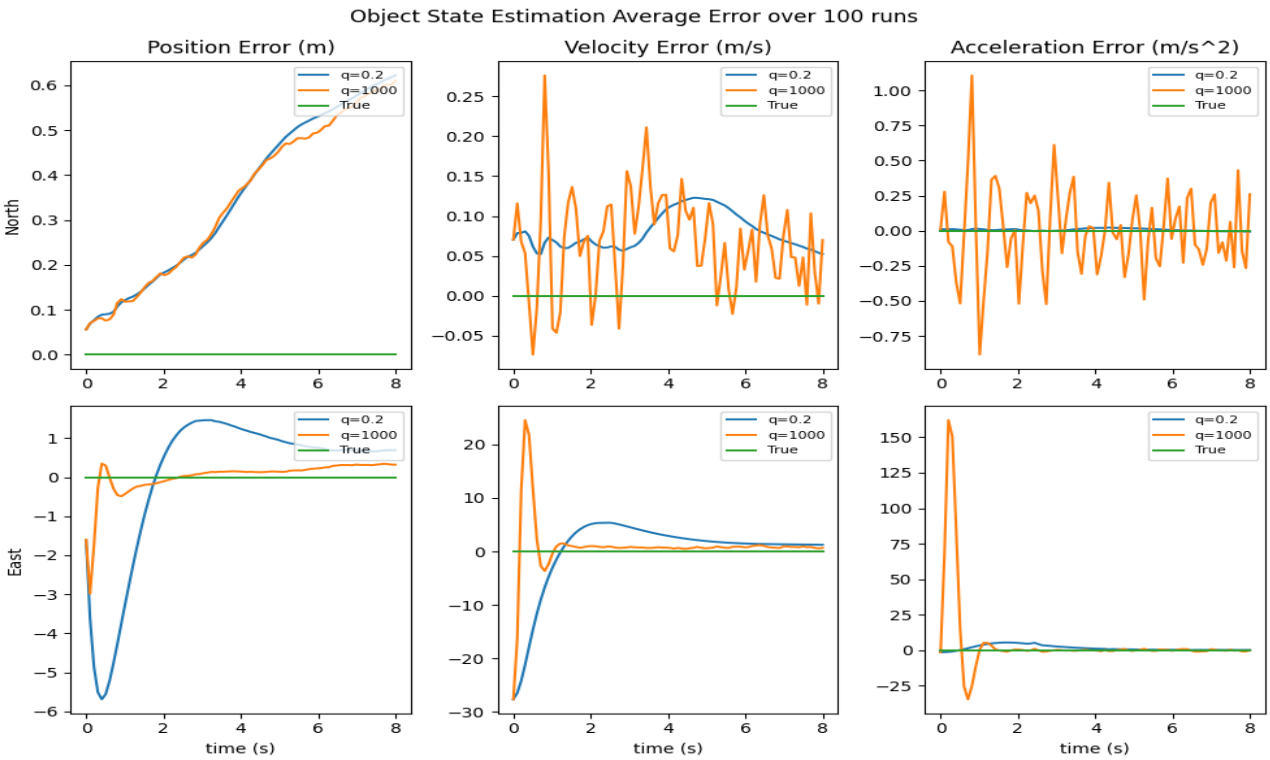
Appendix W.1. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 0)



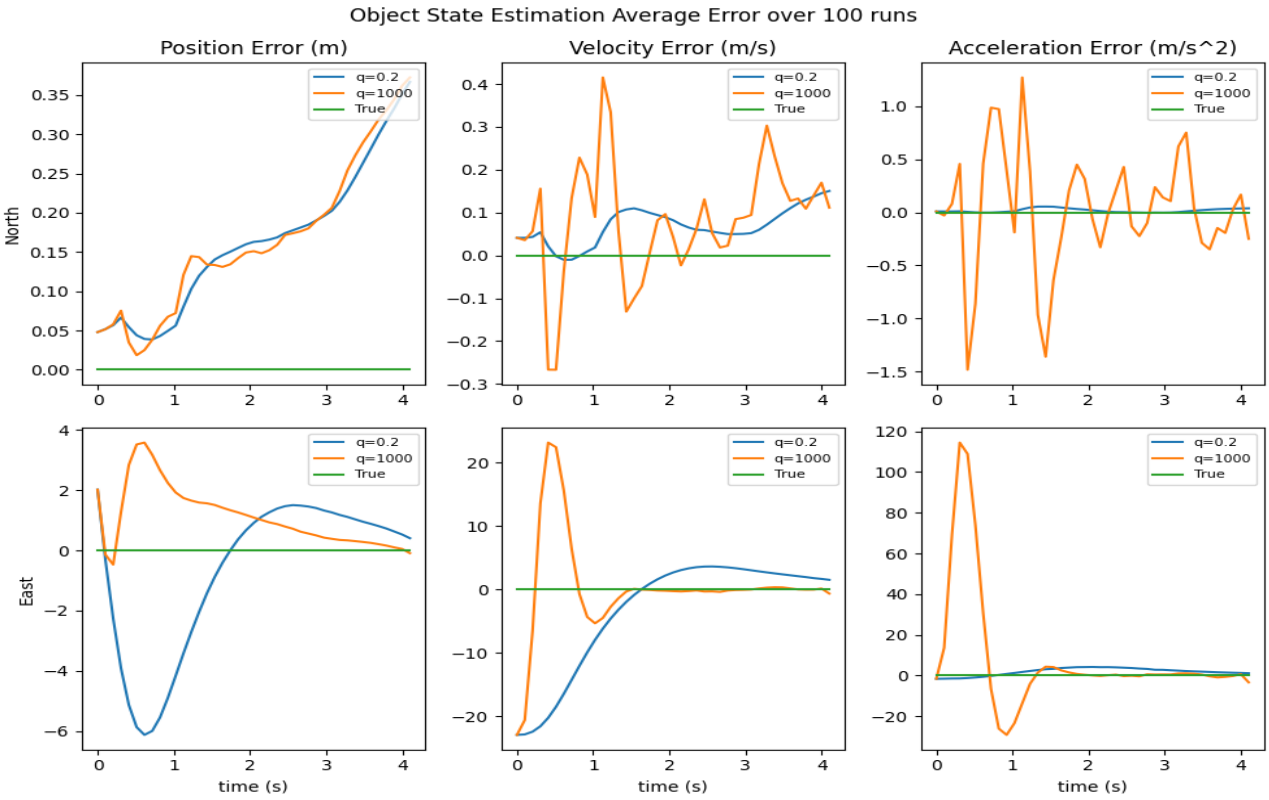
Appendix W.2. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 1)



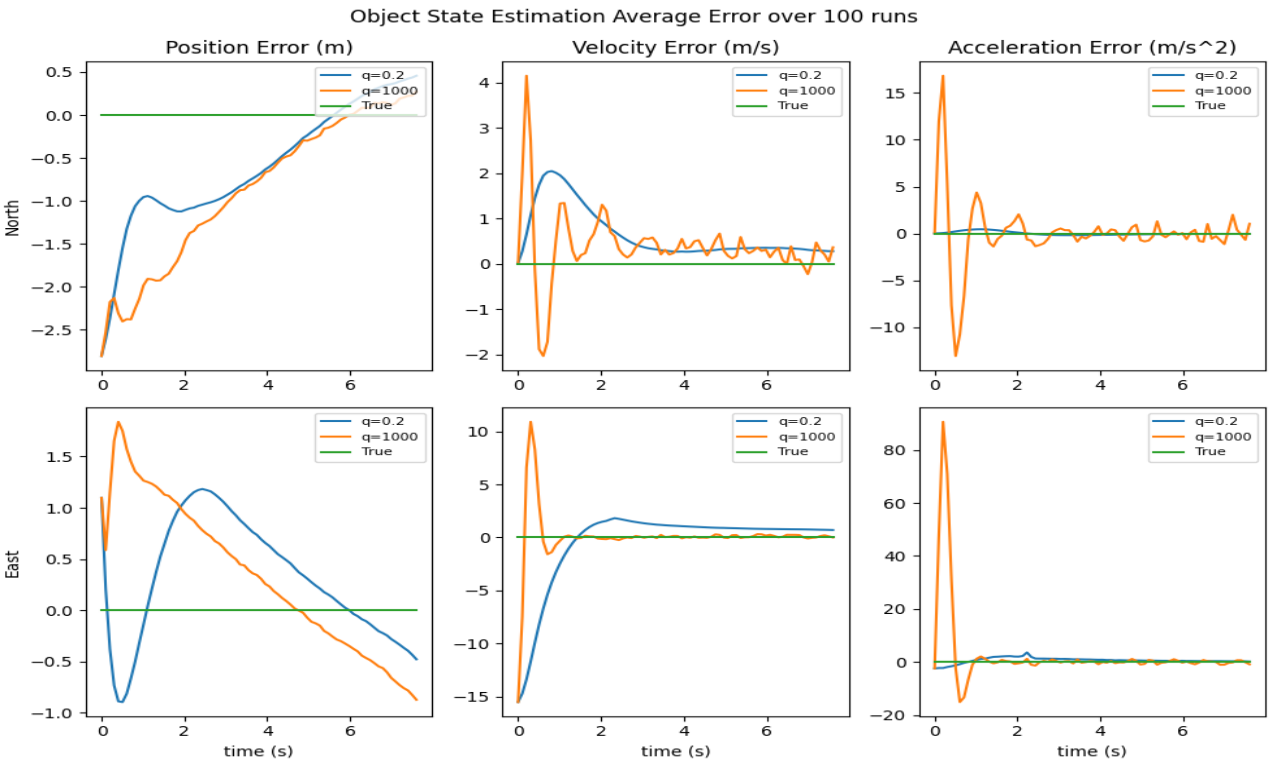
Appendix W.3. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 2)



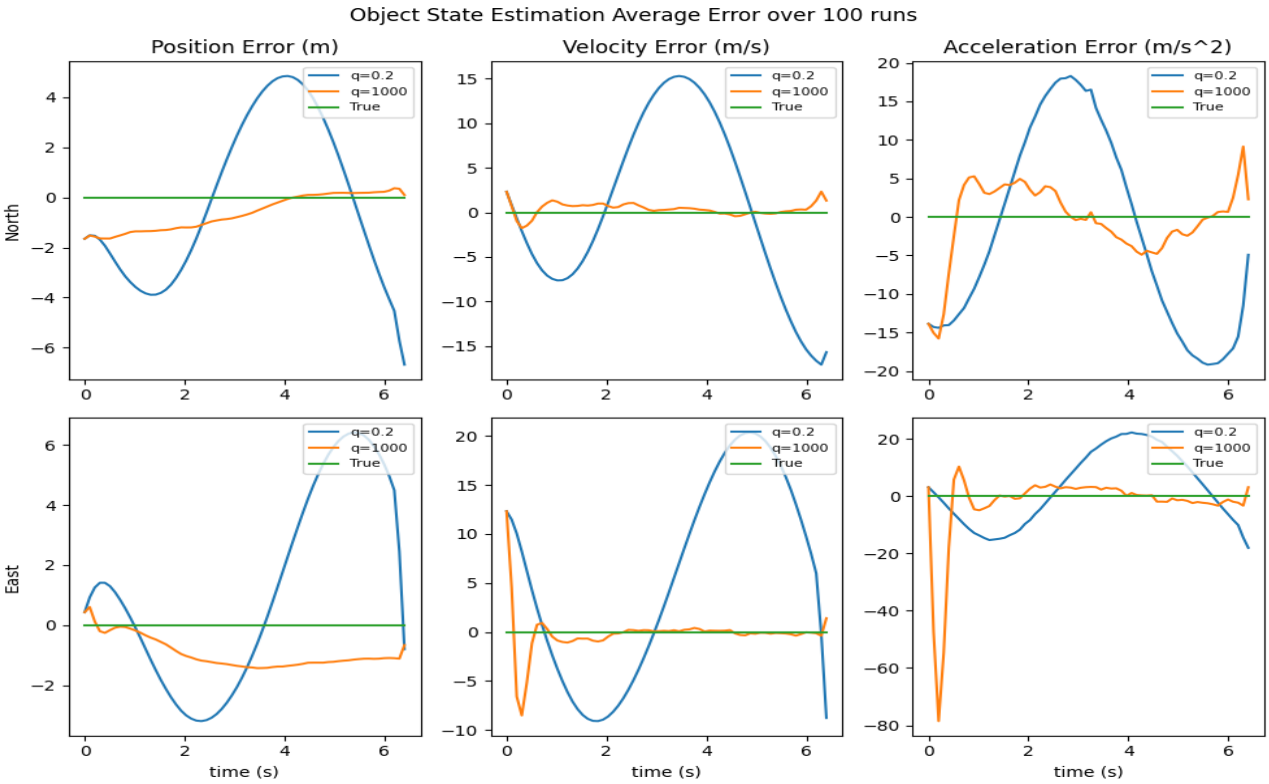
Appendix W.4. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 3)



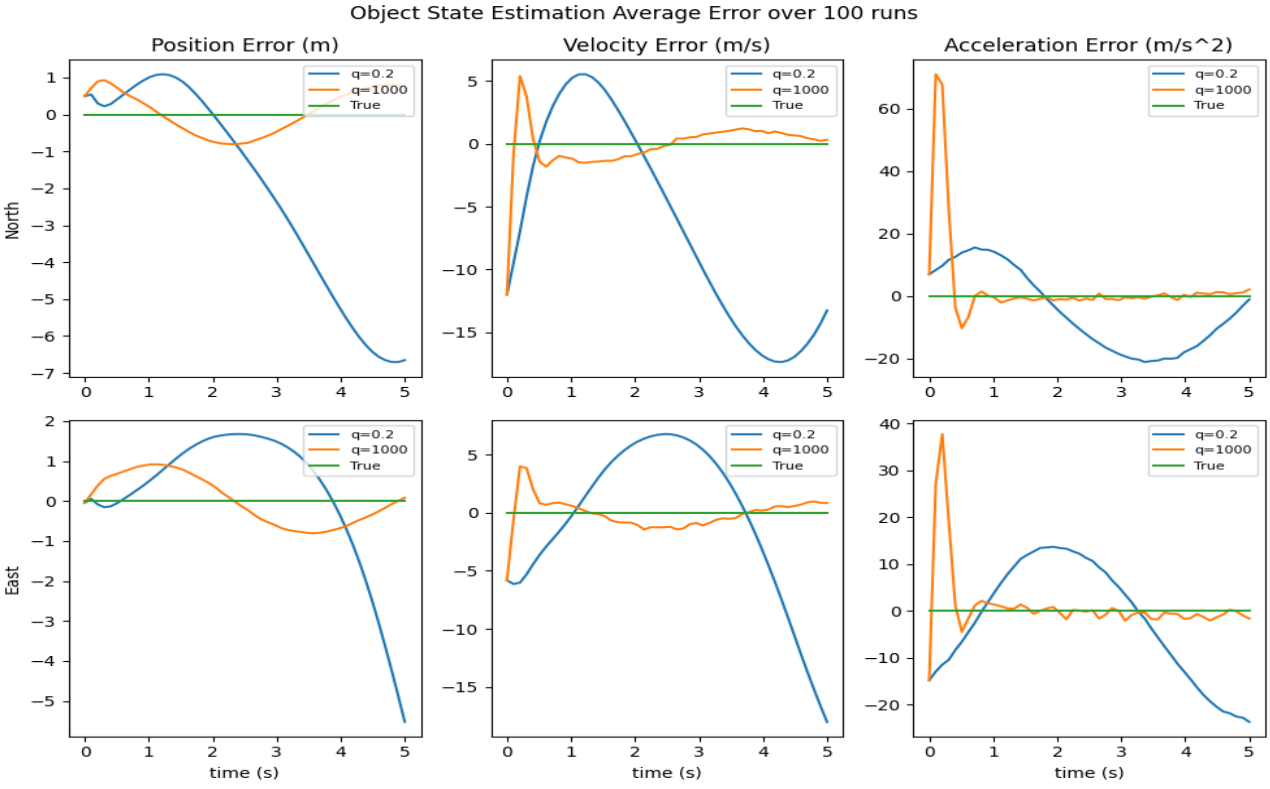
Appendix W.5. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 4)



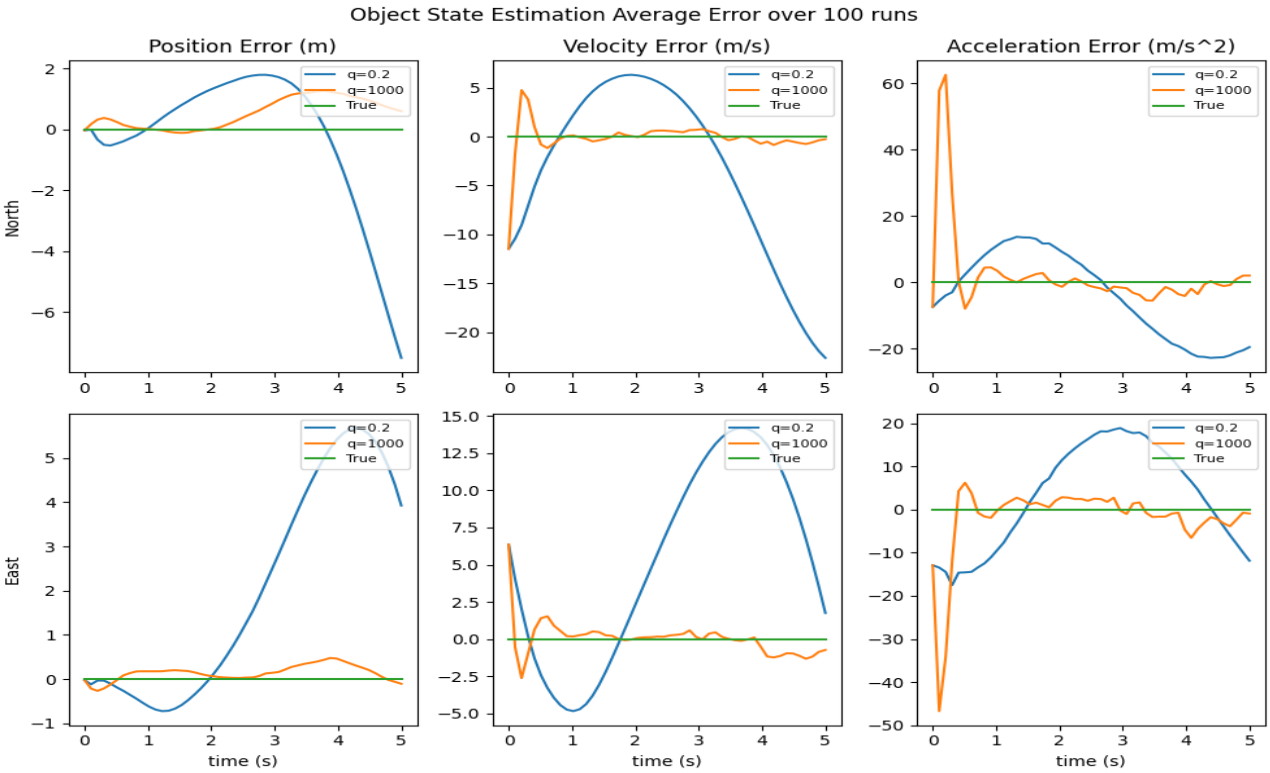
Appendix W.6. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 5)



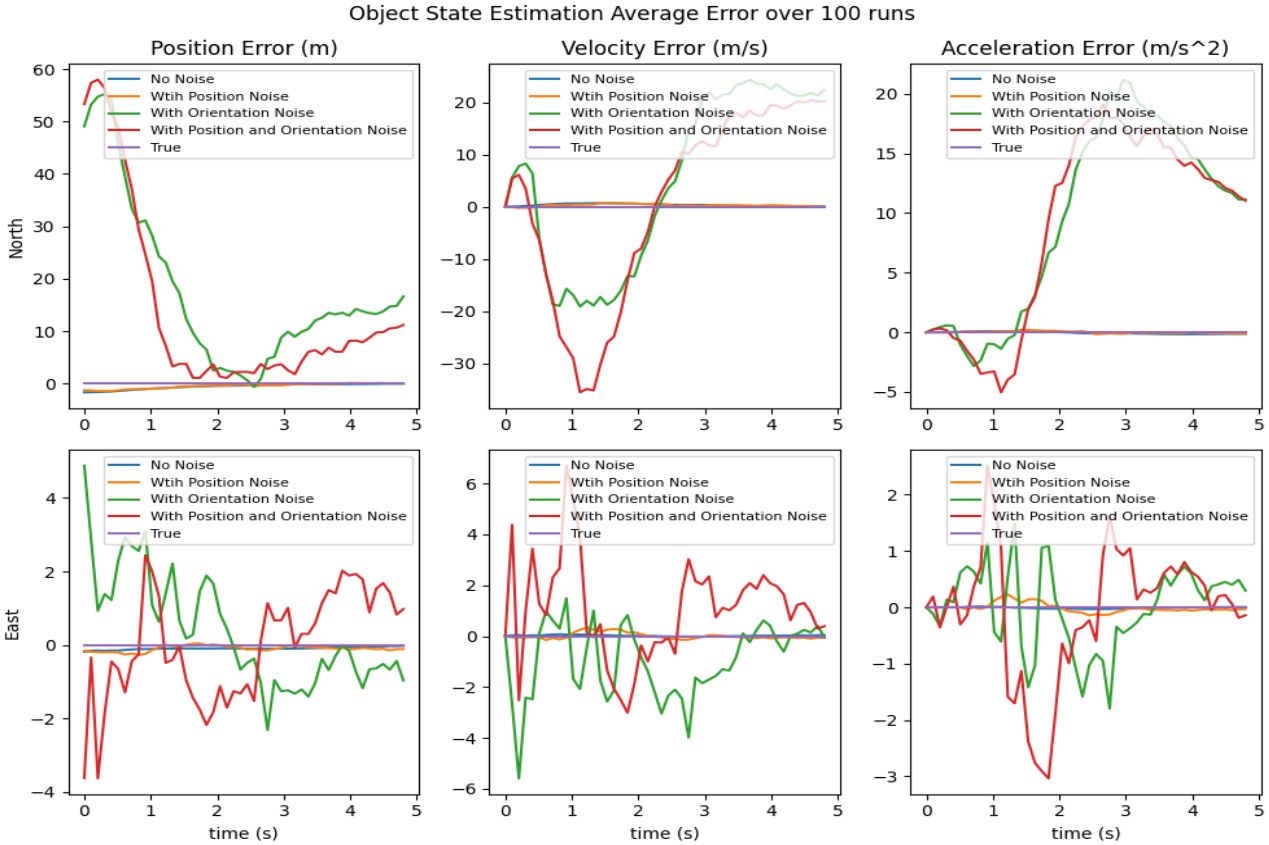
Appendix W.7. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 6)



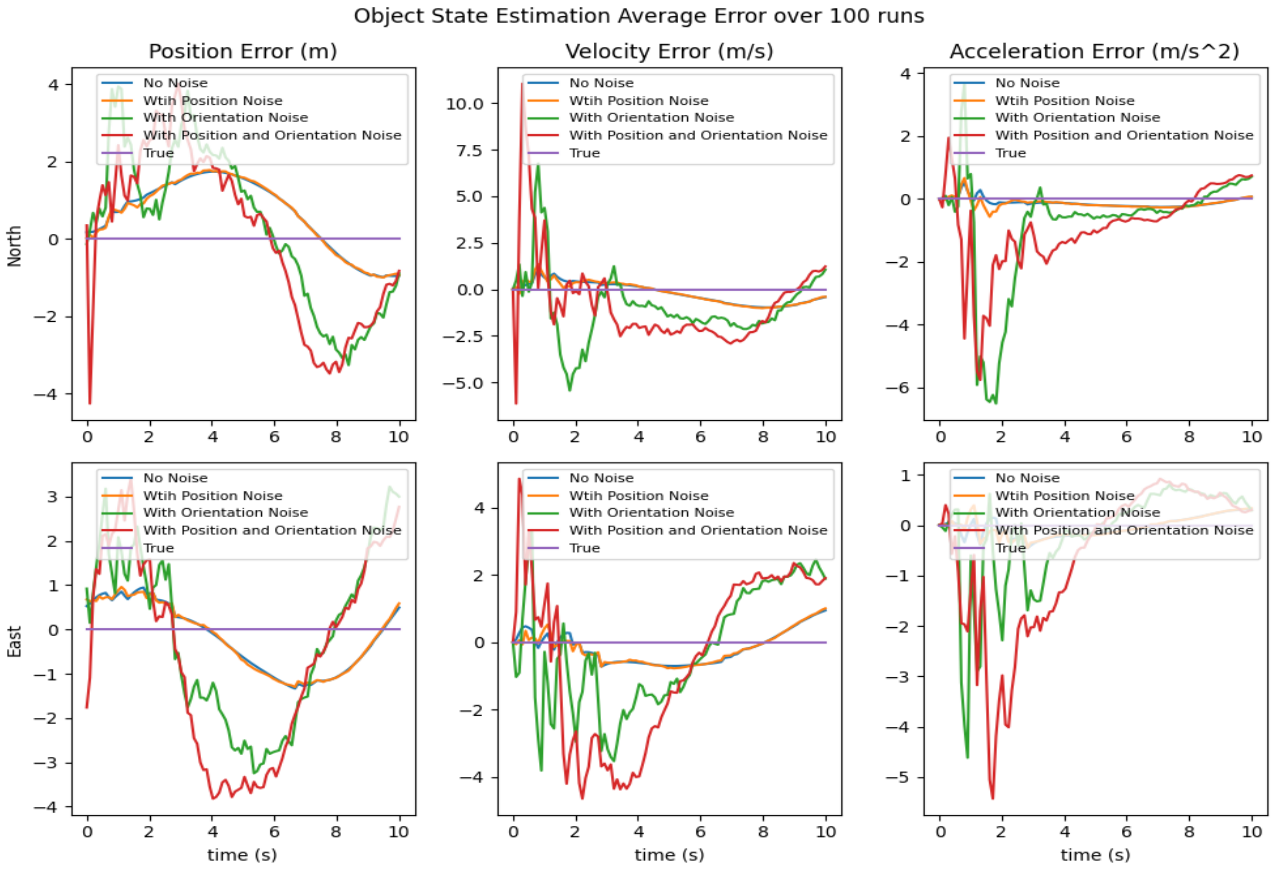
Appendix W.8. Baseline EKF State Estimation Average Error with different acceleration noise q (Scenario 7)



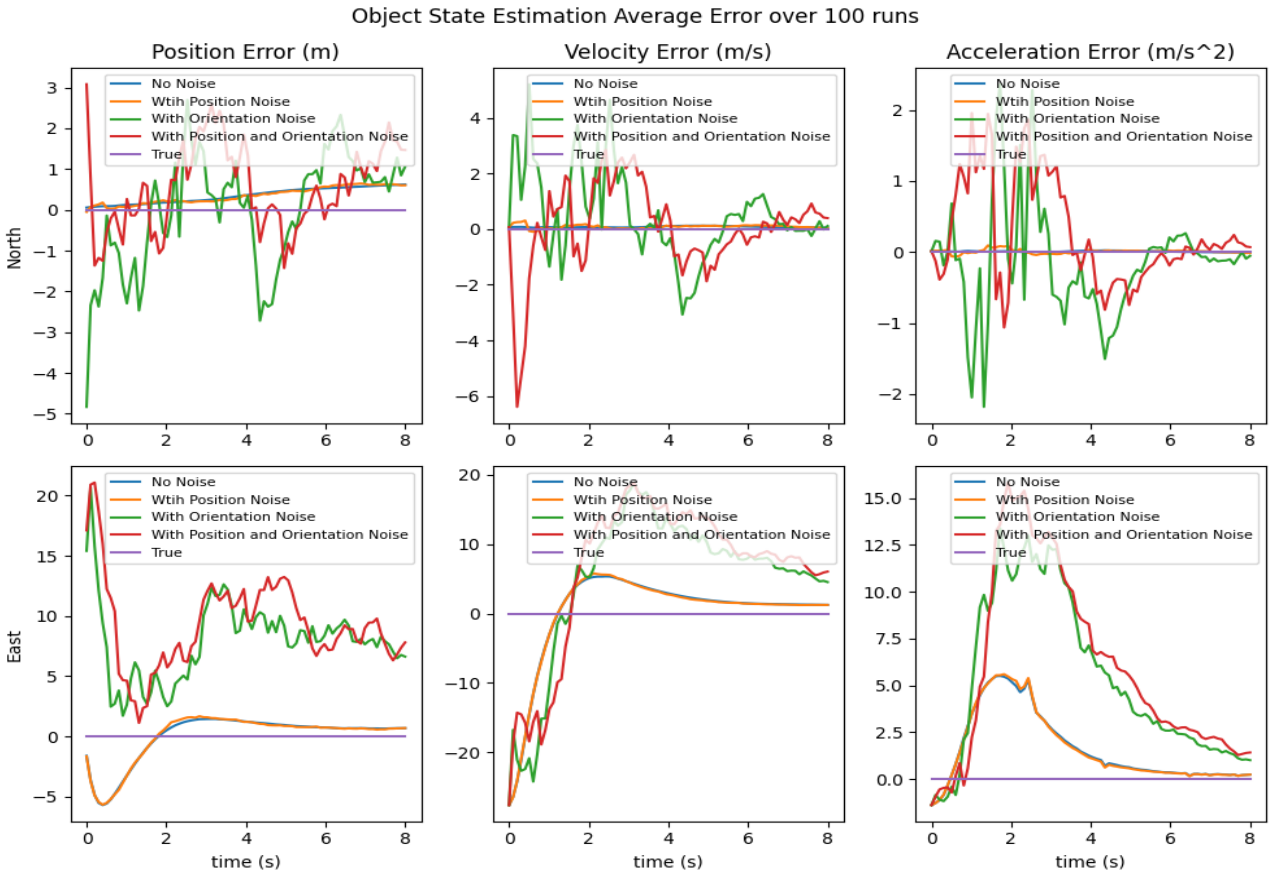
Appendix X.1. Baseline EKF average estimation error with MAV state noise (Scenario 0)



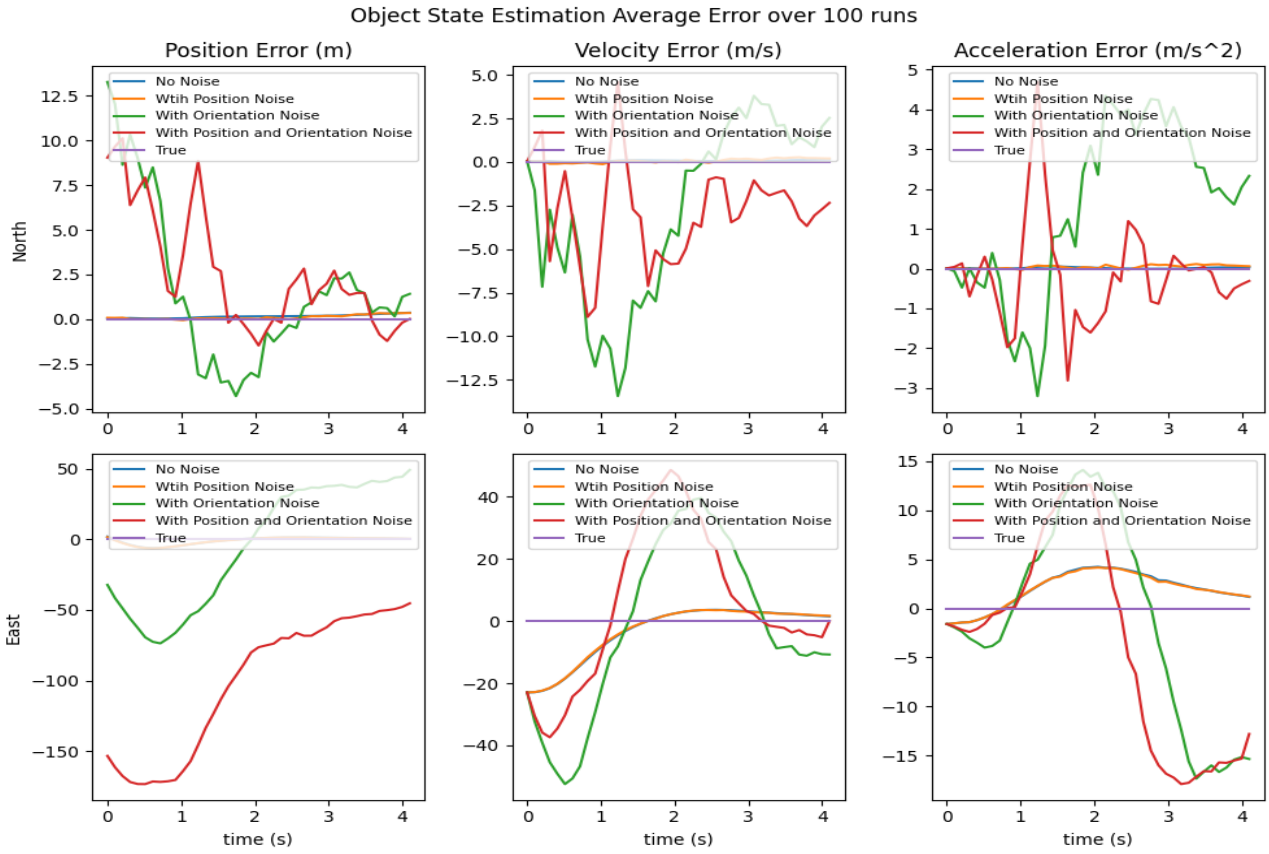
Appendix X.2. Baseline EKF average estimation error with MAV state noise (Scenario 1)



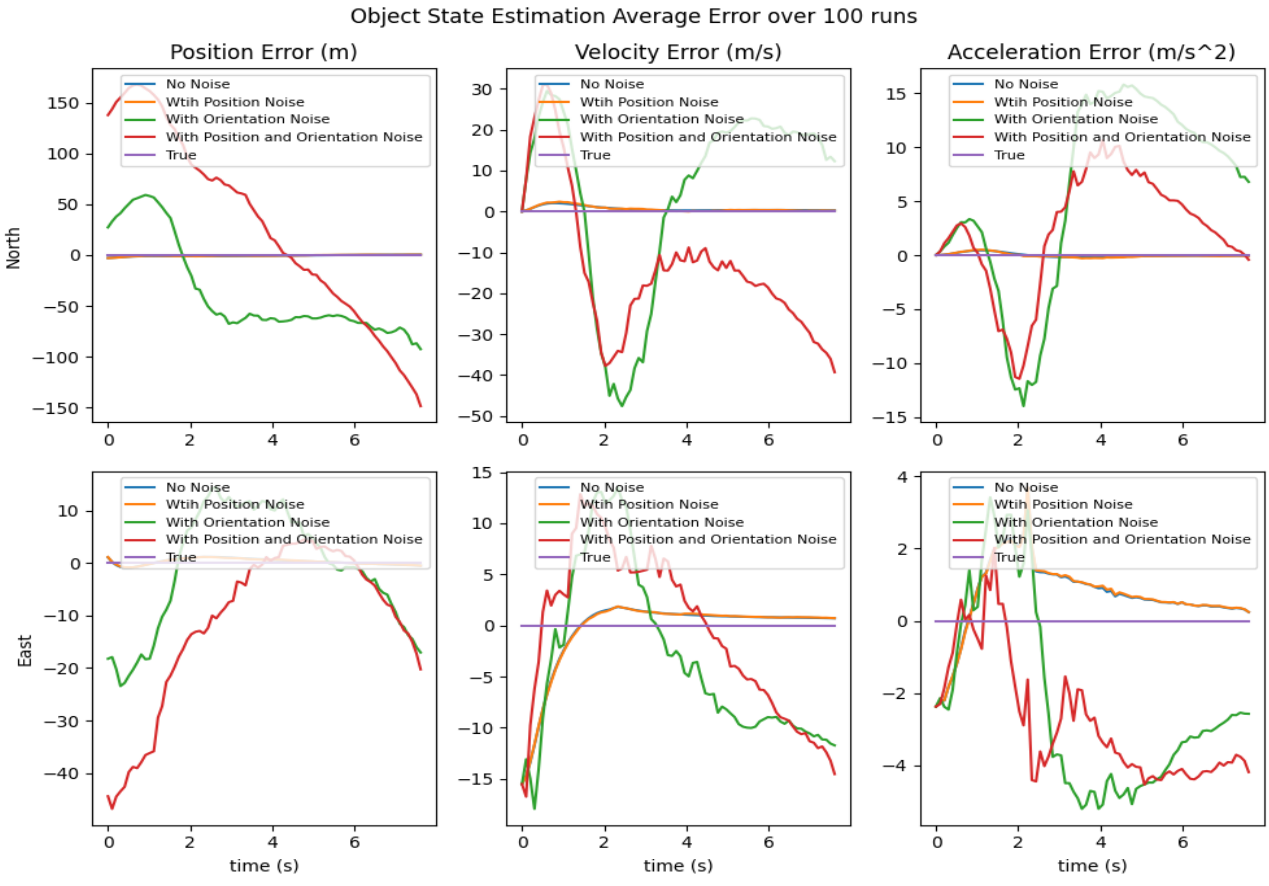
Appendix X.3. Baseline EKF average estimation error with MAV state noise (Scenario 2)



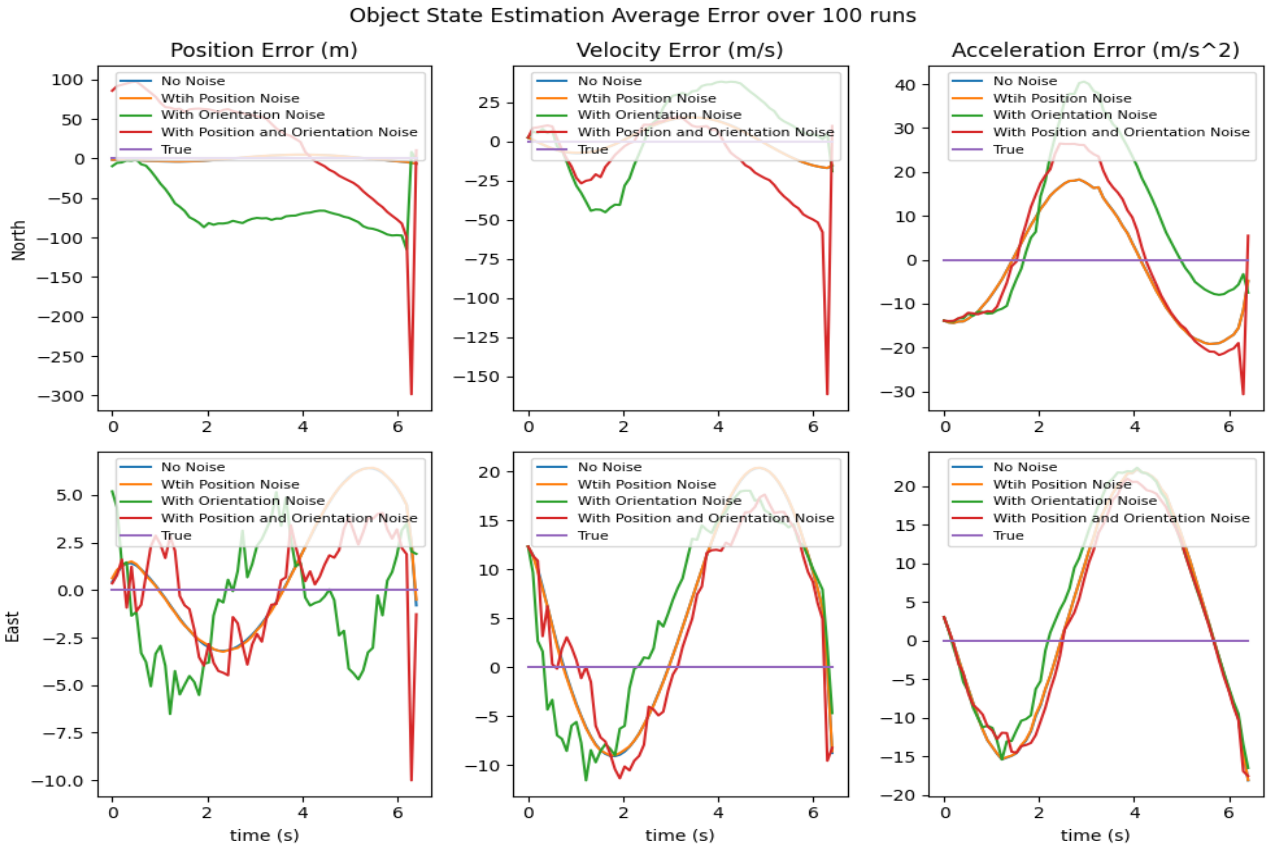
Appendix X.4. Baseline EKF average estimation error with MAV state noise (Scenario 3)



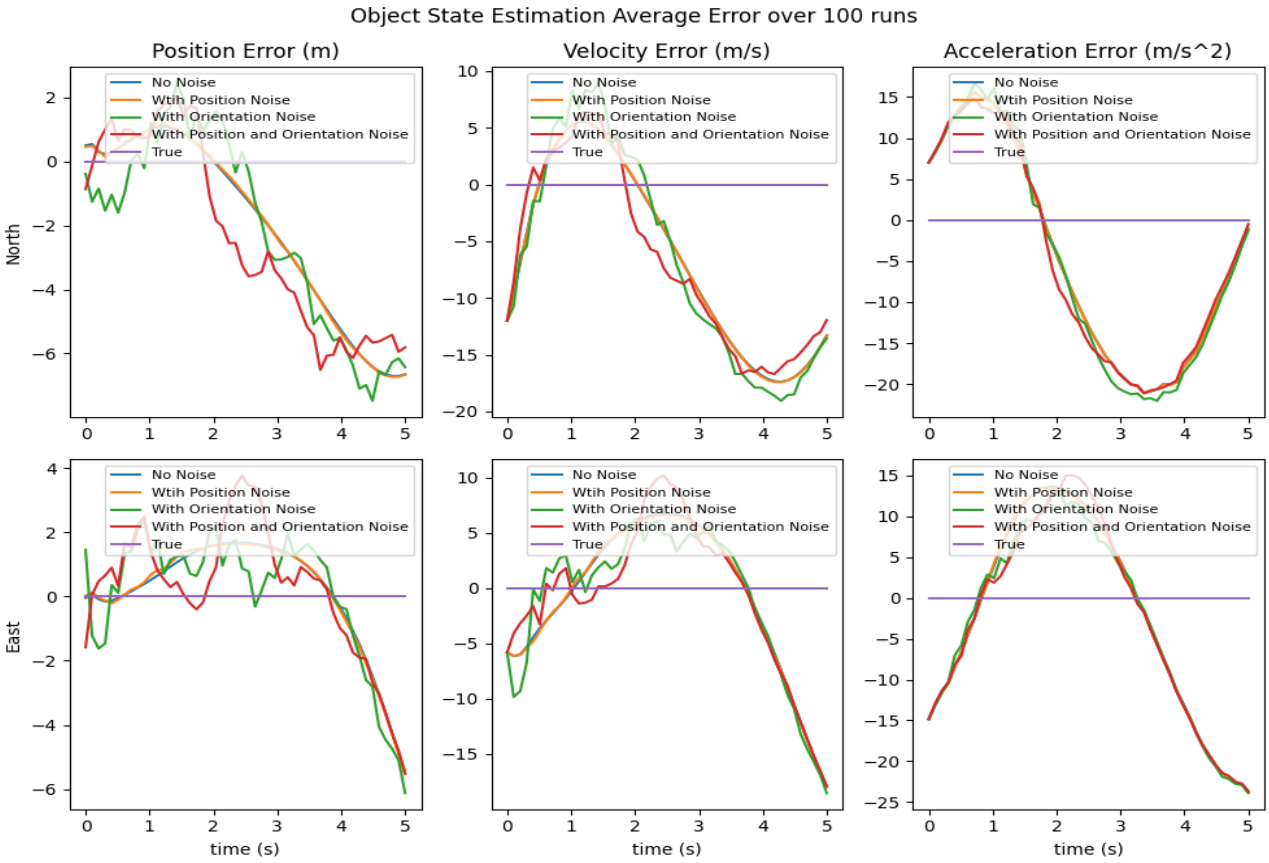
Appendix X.5. Baseline EKF average estimation error with MAV state noise (Scenario 4)



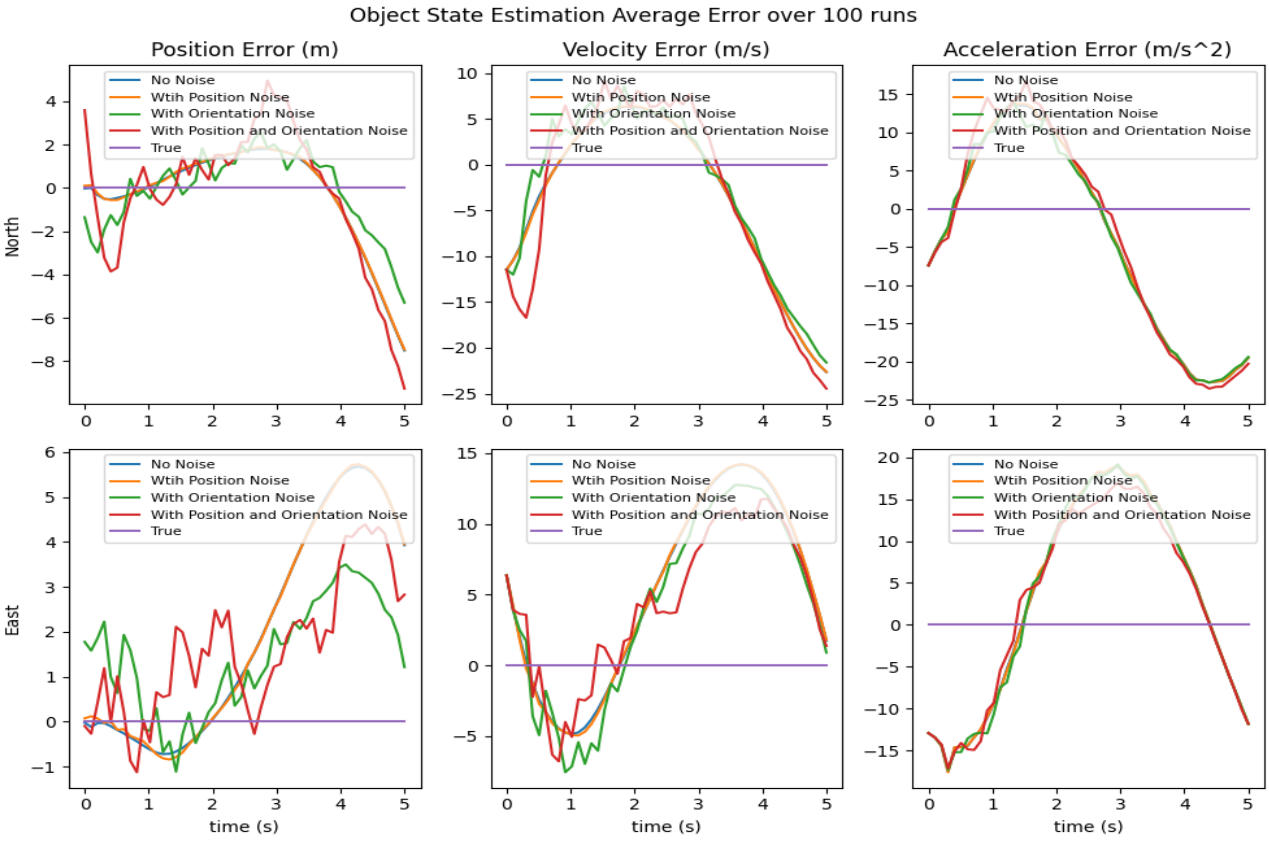
Appendix X.6. Baseline EKF average estimation error with MAV state noise (Scenario 5)



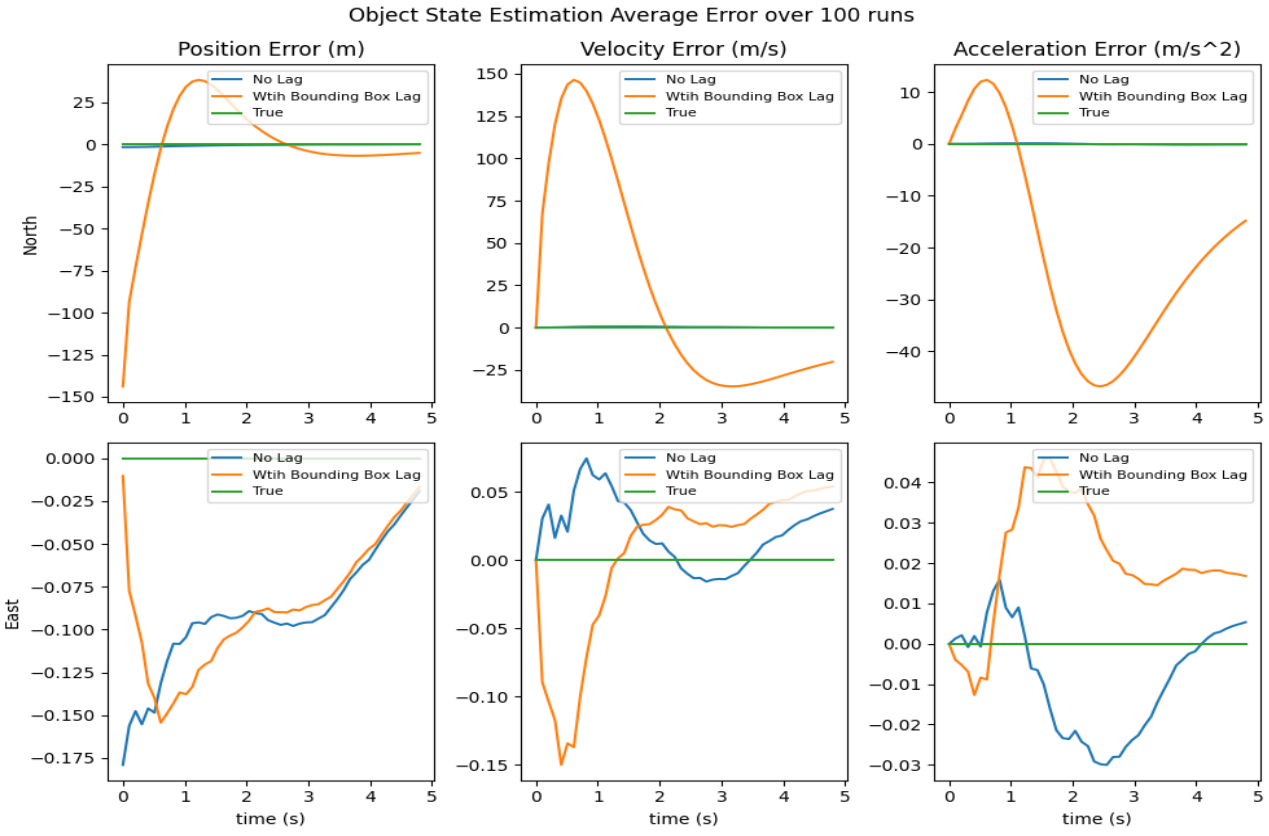
Appendix X.7. Baseline EKF average estimation error with MAV state noise (Scenario 6)



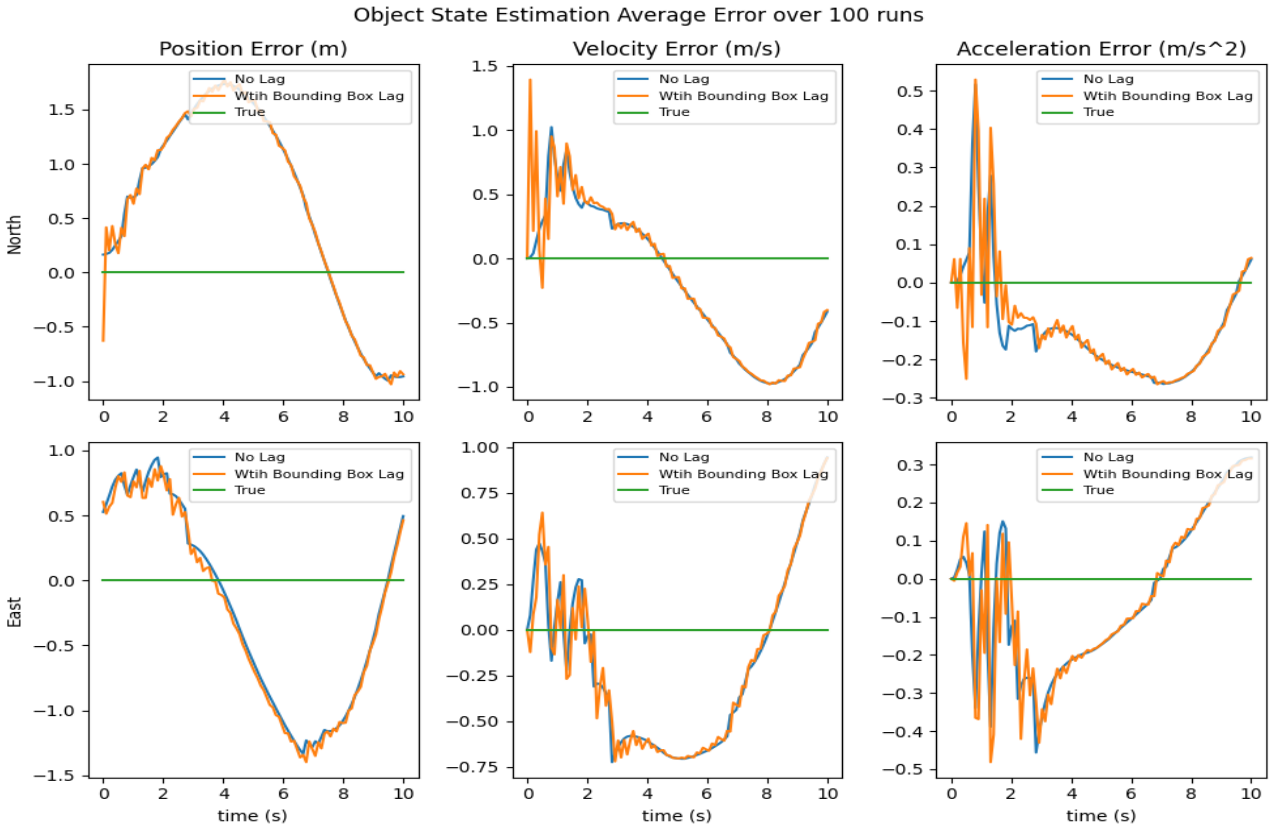
Appendix X.8. Baseline EKF average estimation error with MAV state noise (Scenario 7)



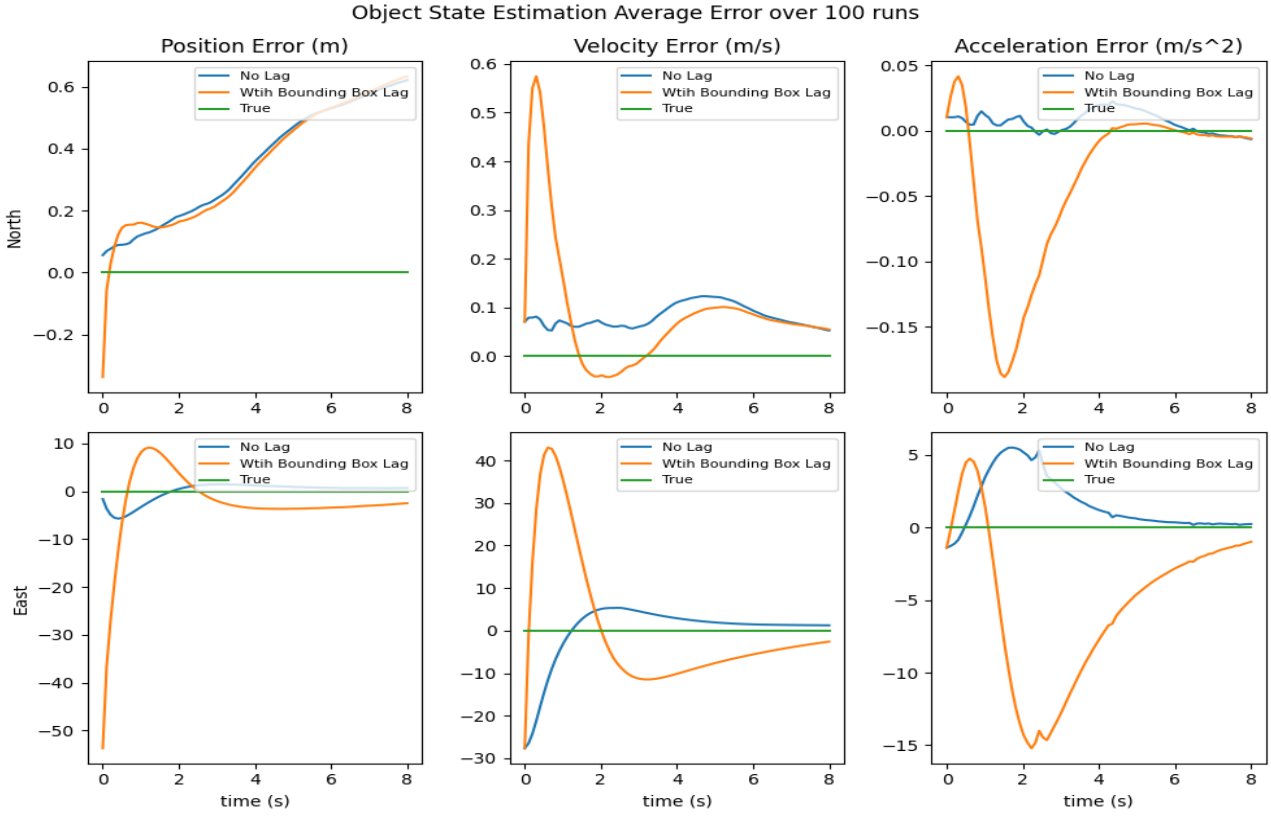
Appendix Y.1. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 0)



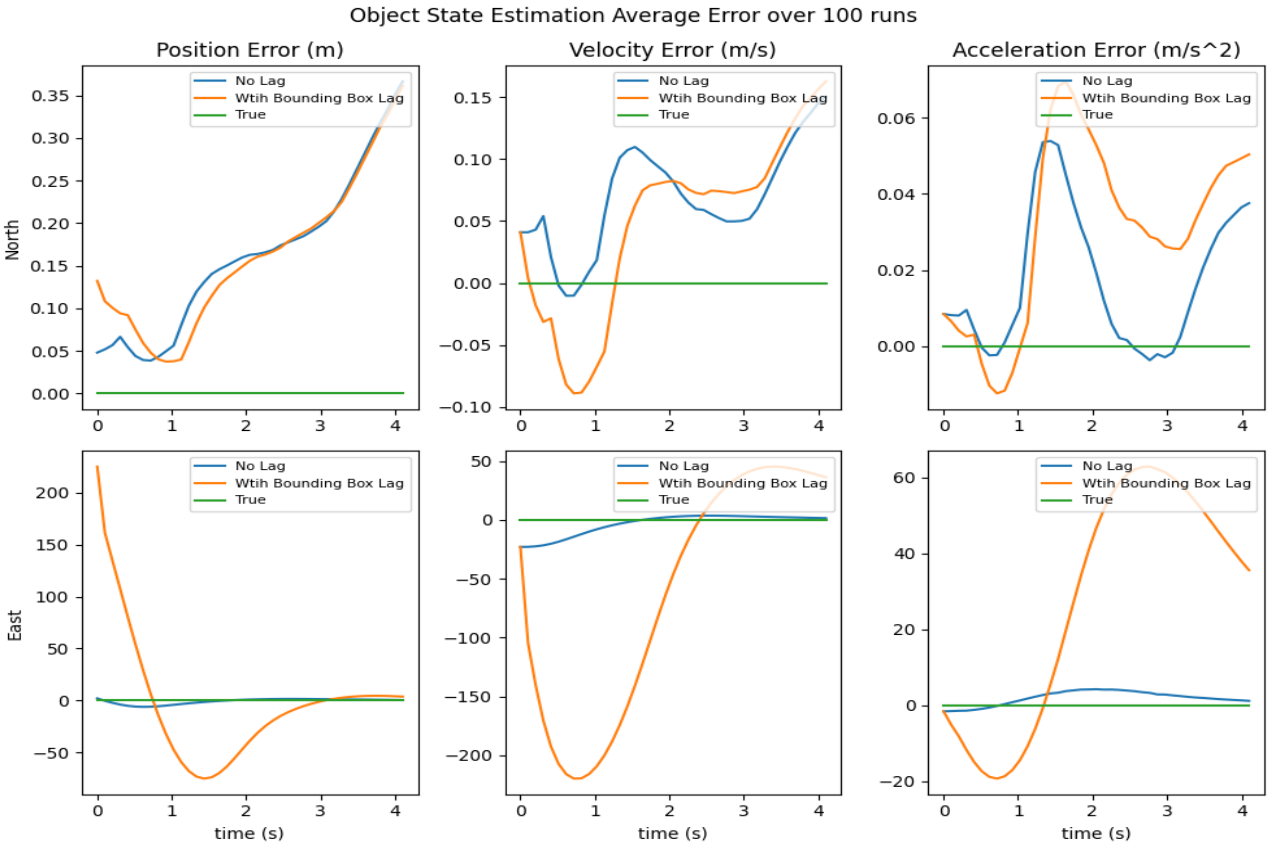
Appendix Y.2. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 1)



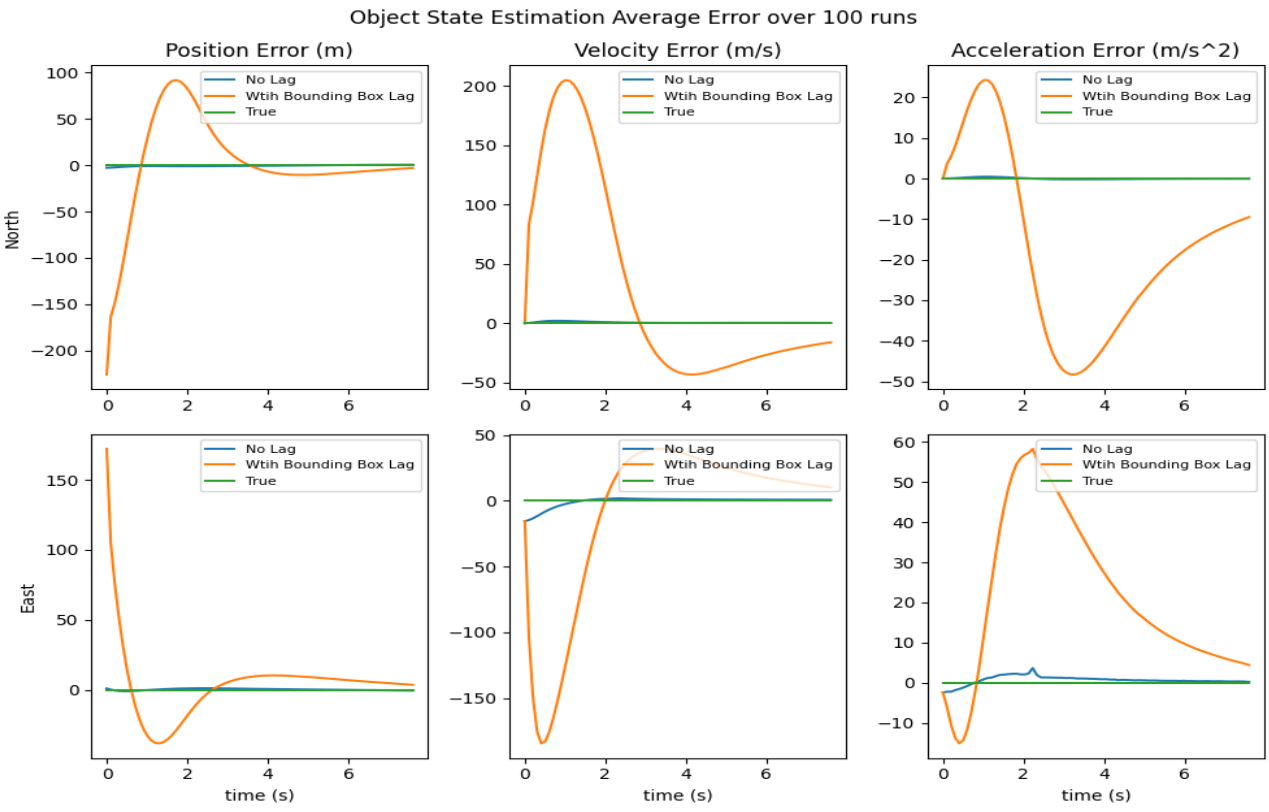
Appendix Y.3. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 2)



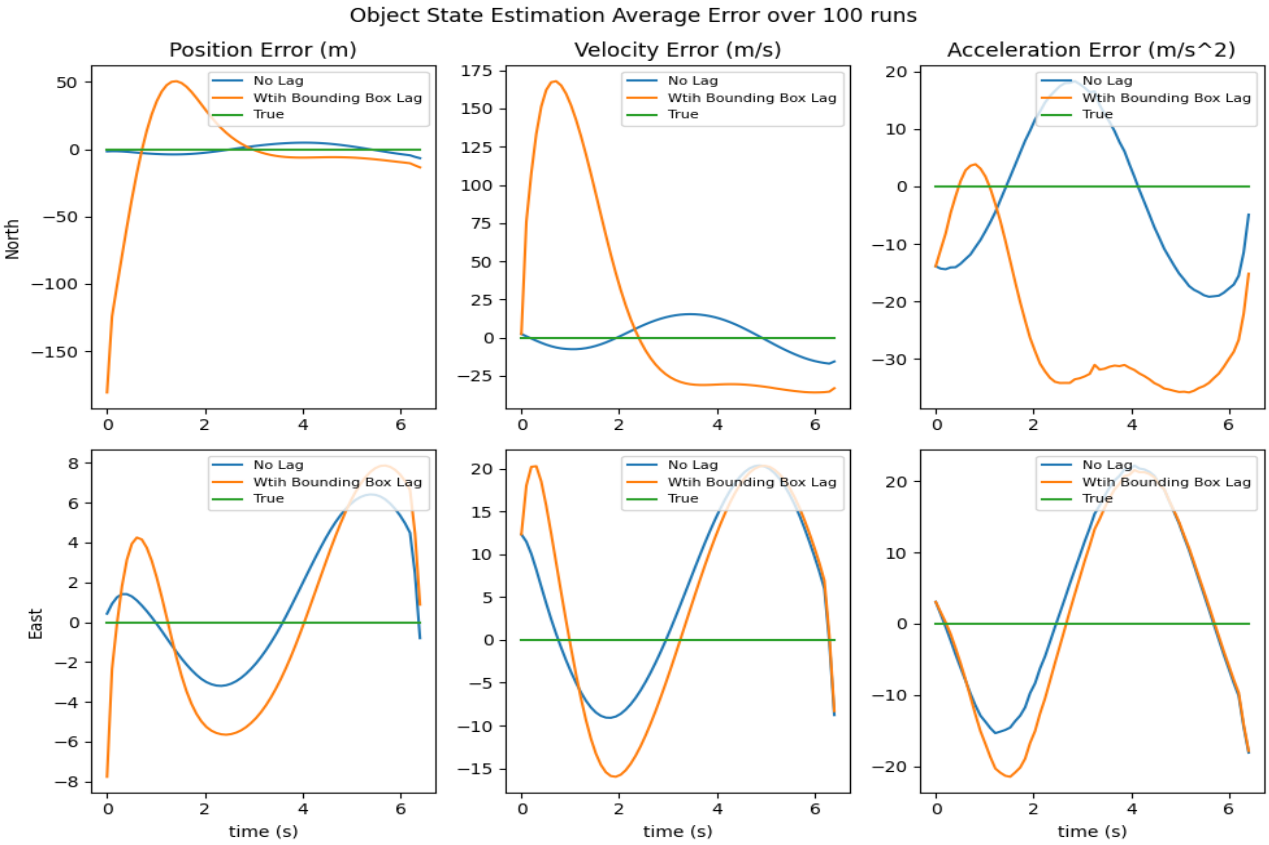
Appendix Y.4. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 3)



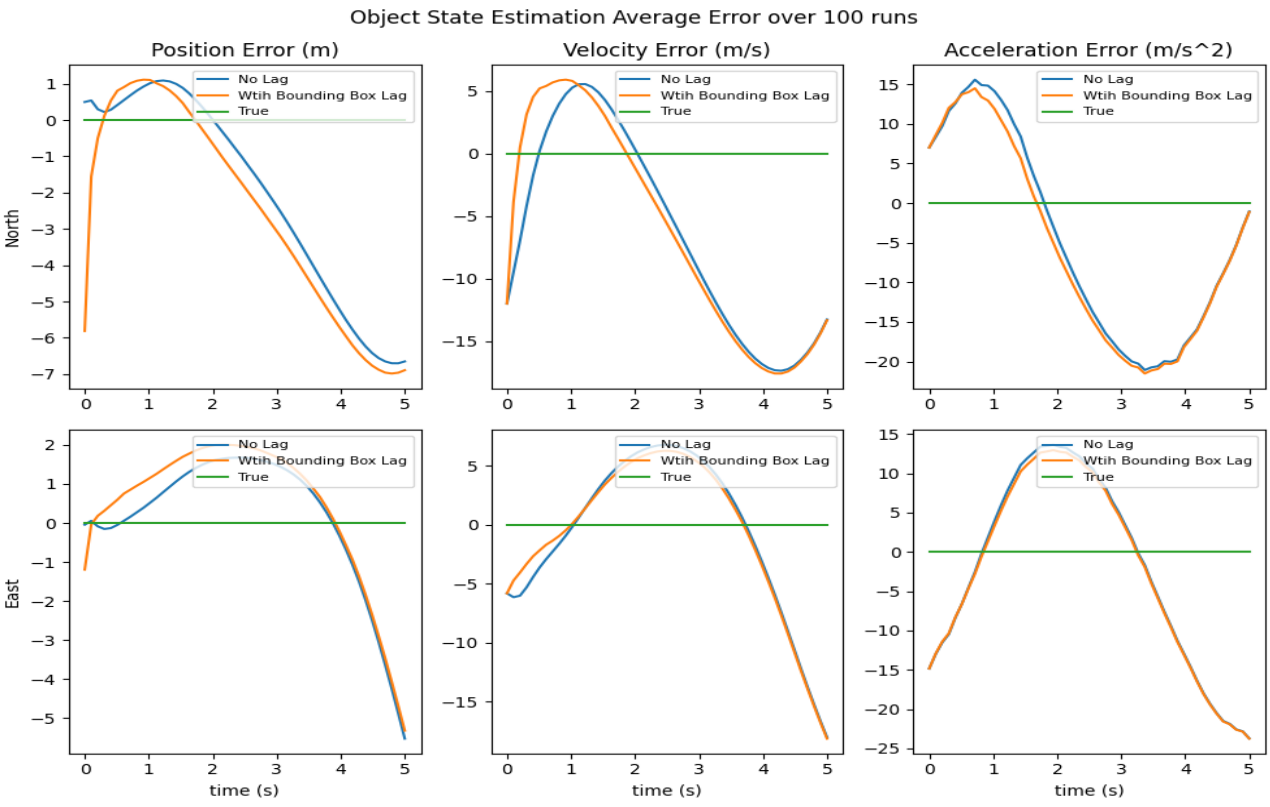
Appendix Y.5. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 4)



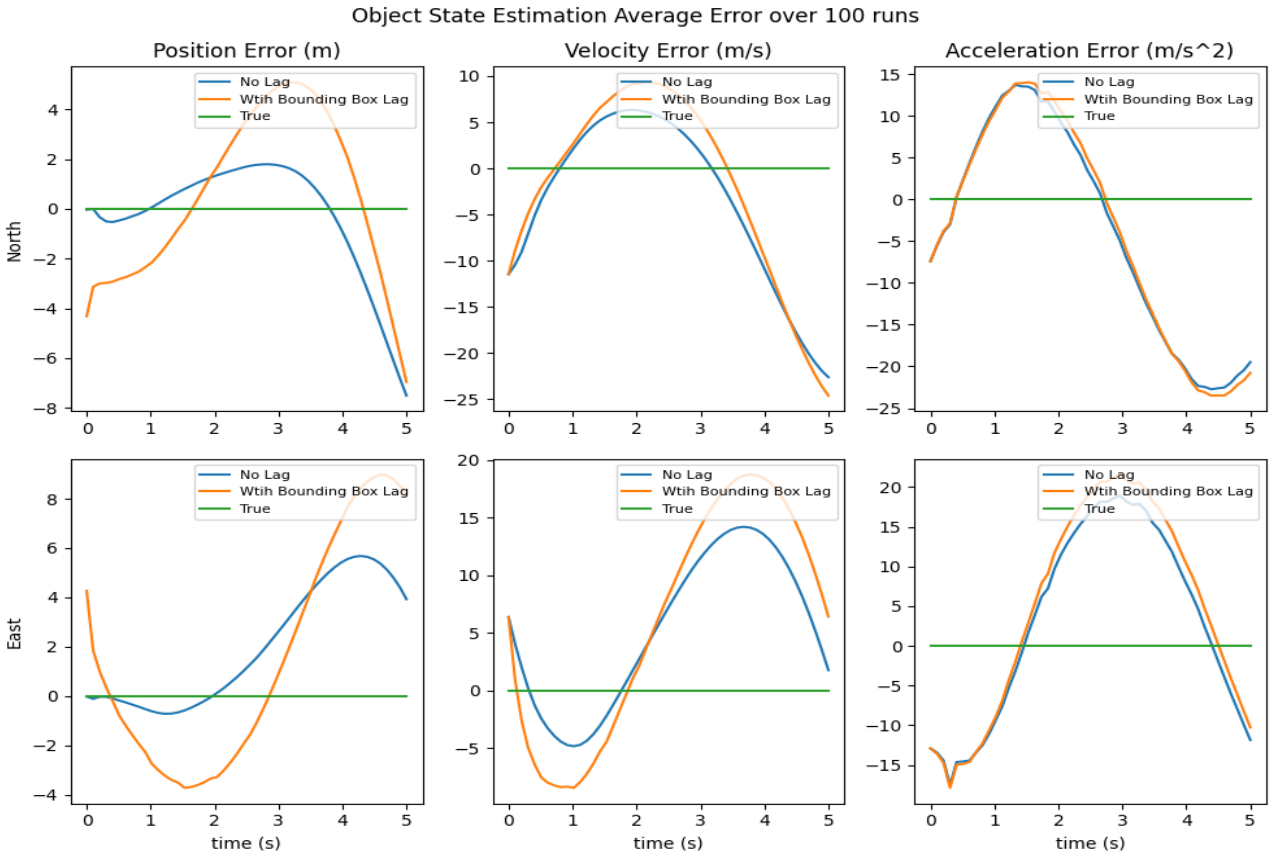
Appendix Y.6. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 5)



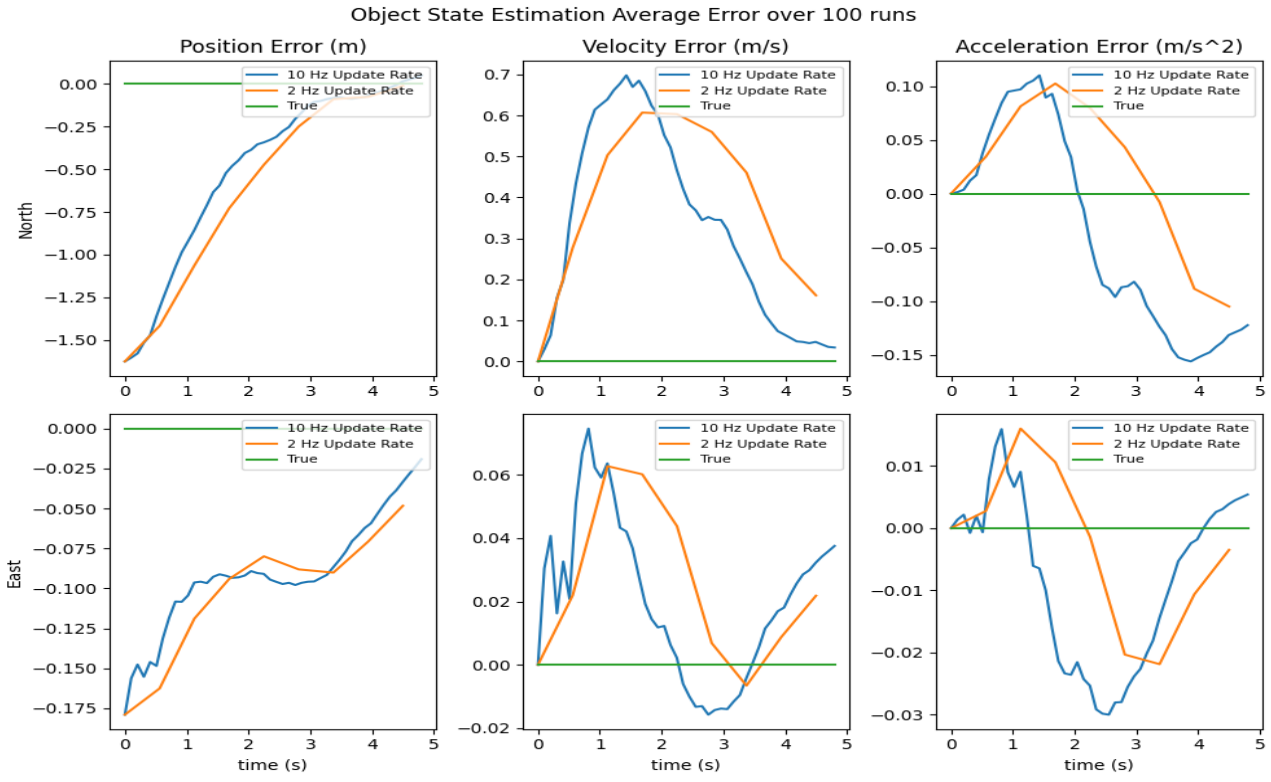
Appendix Y.7. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 6)



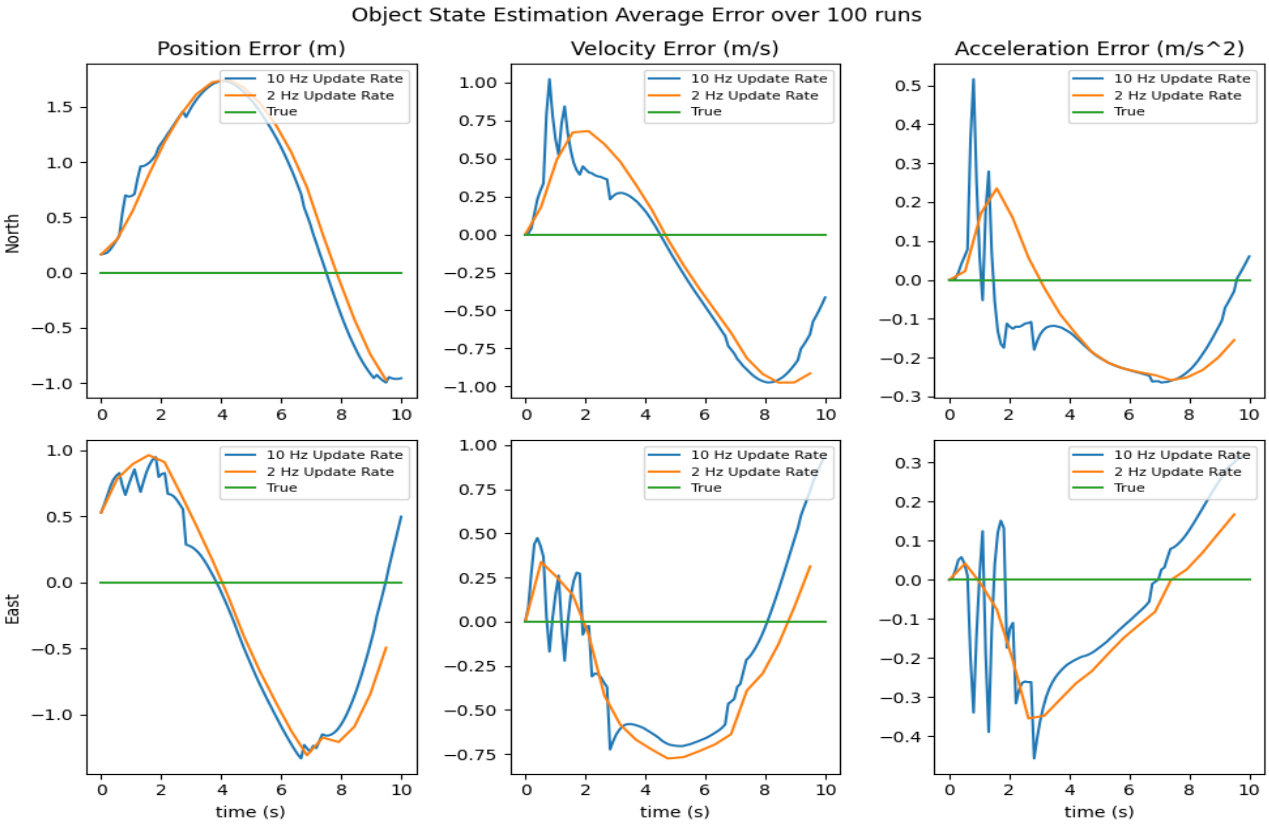
Appendix Y.8. Baseline EKF average estimation error with 0.1 seconds lag in object bounding box (Scenario 7)



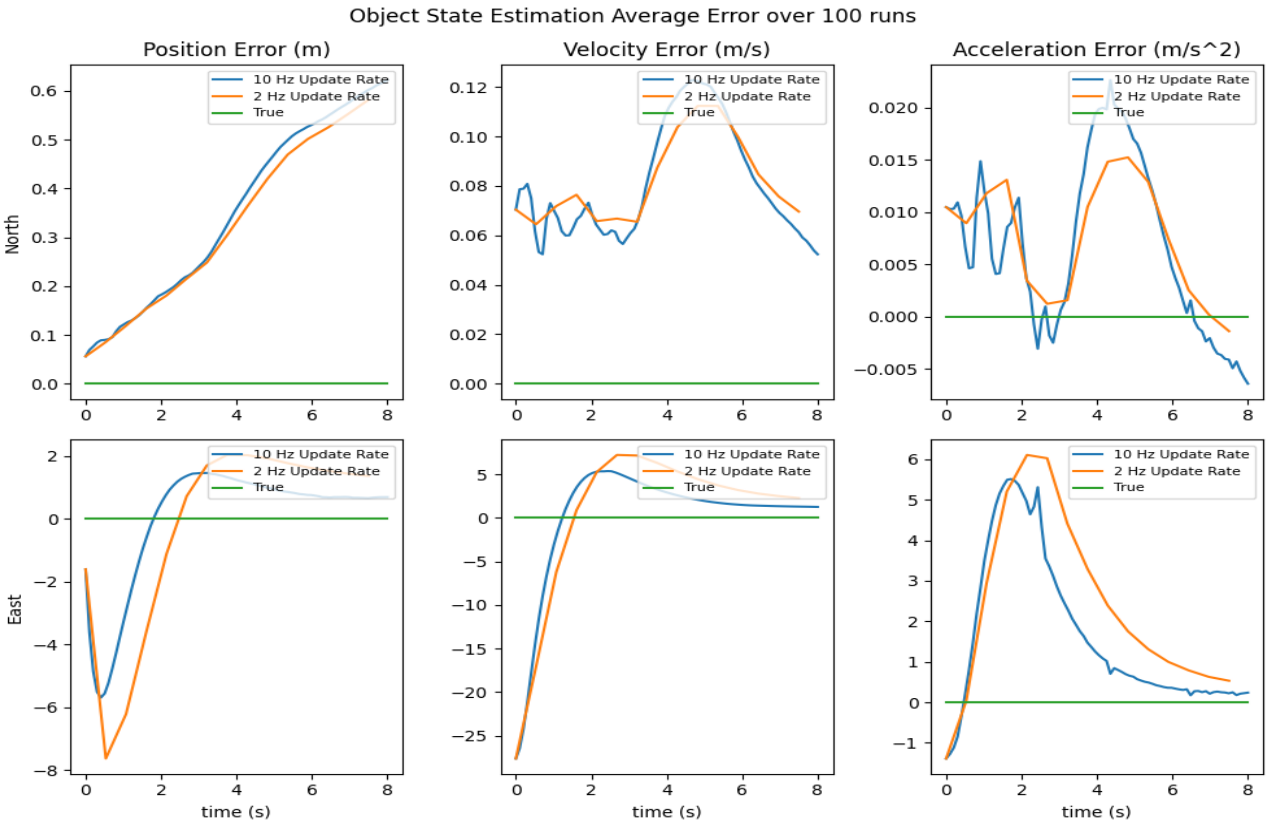
Appendix Z.1. Baseline EKF average estimation error with different measurement update rate (Scenario 0)



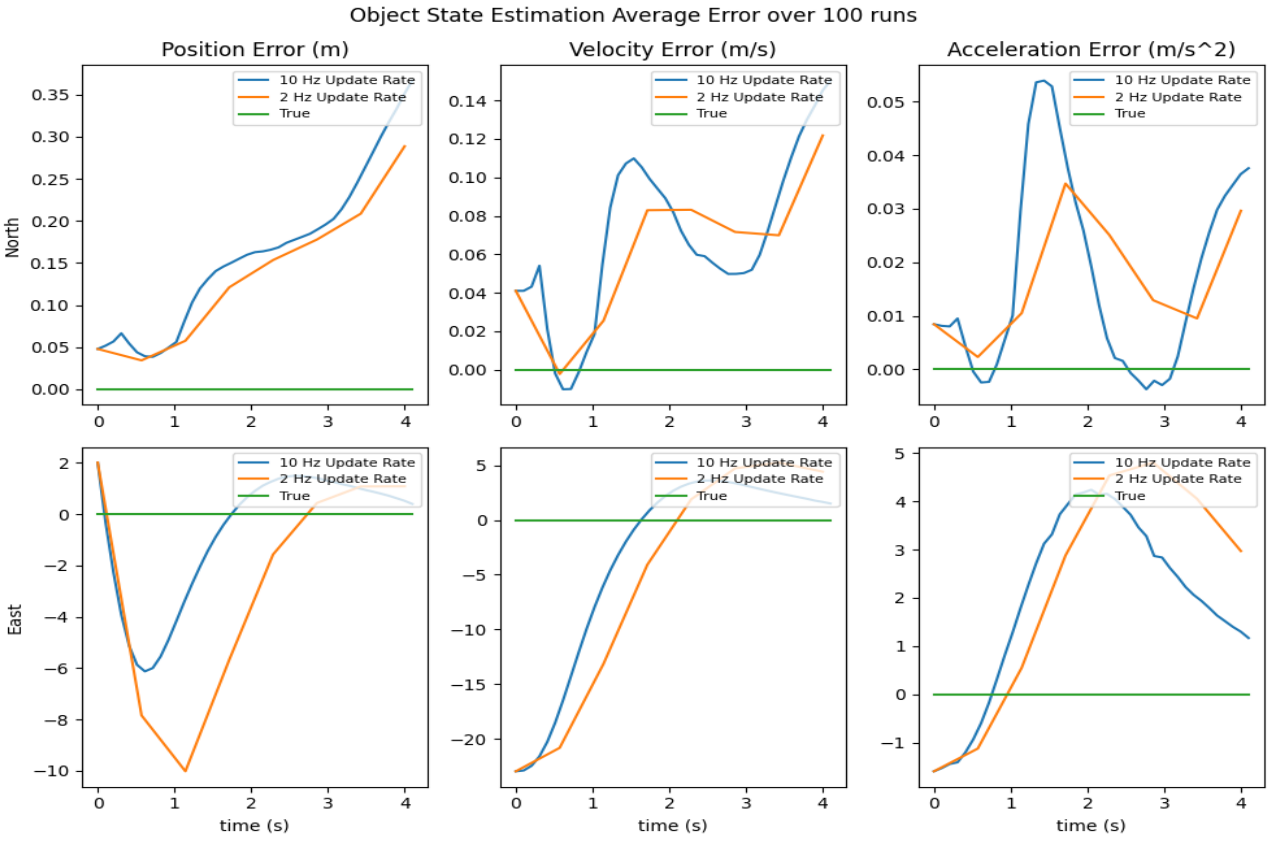
Appendix Z.2. Baseline EKF average estimation error with different measurement update rate (Scenario 1)



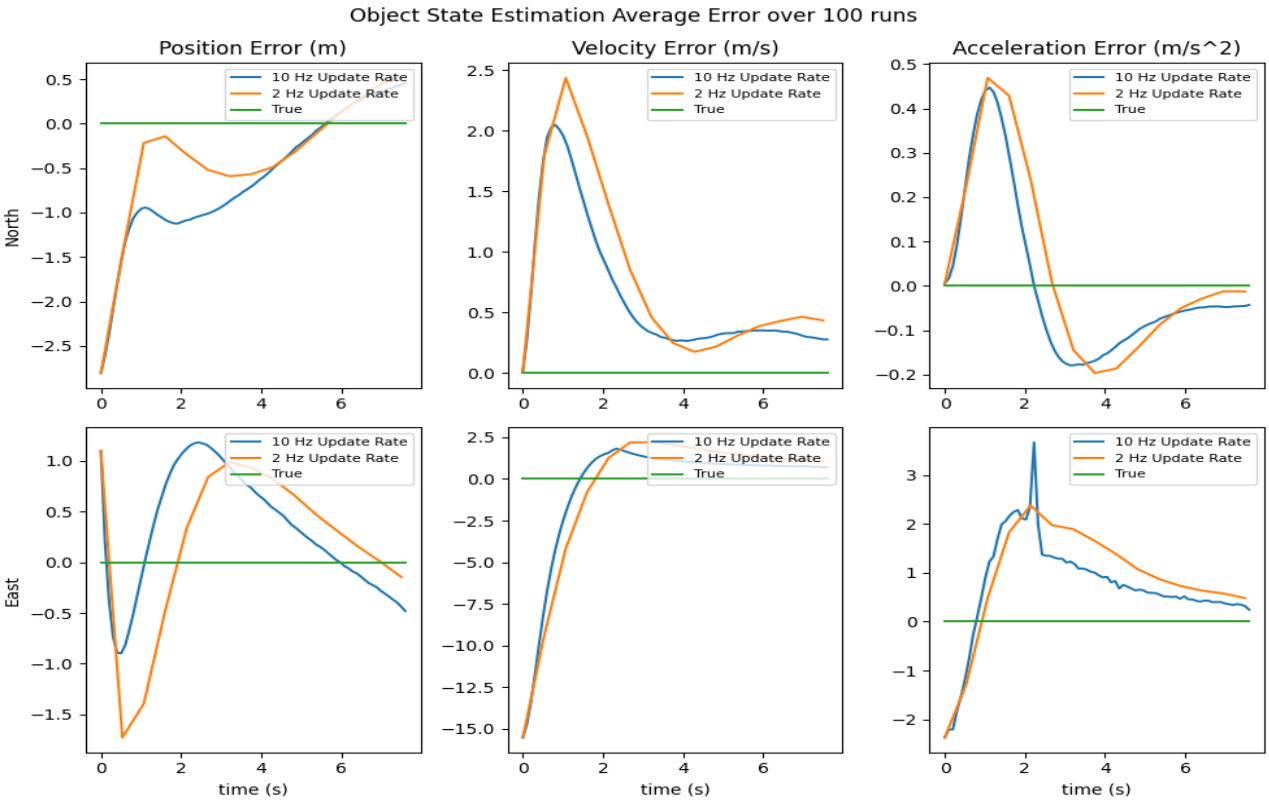
Appendix Z.3. Baseline EKF average estimation error with different measurement update rate (Scenario 2)



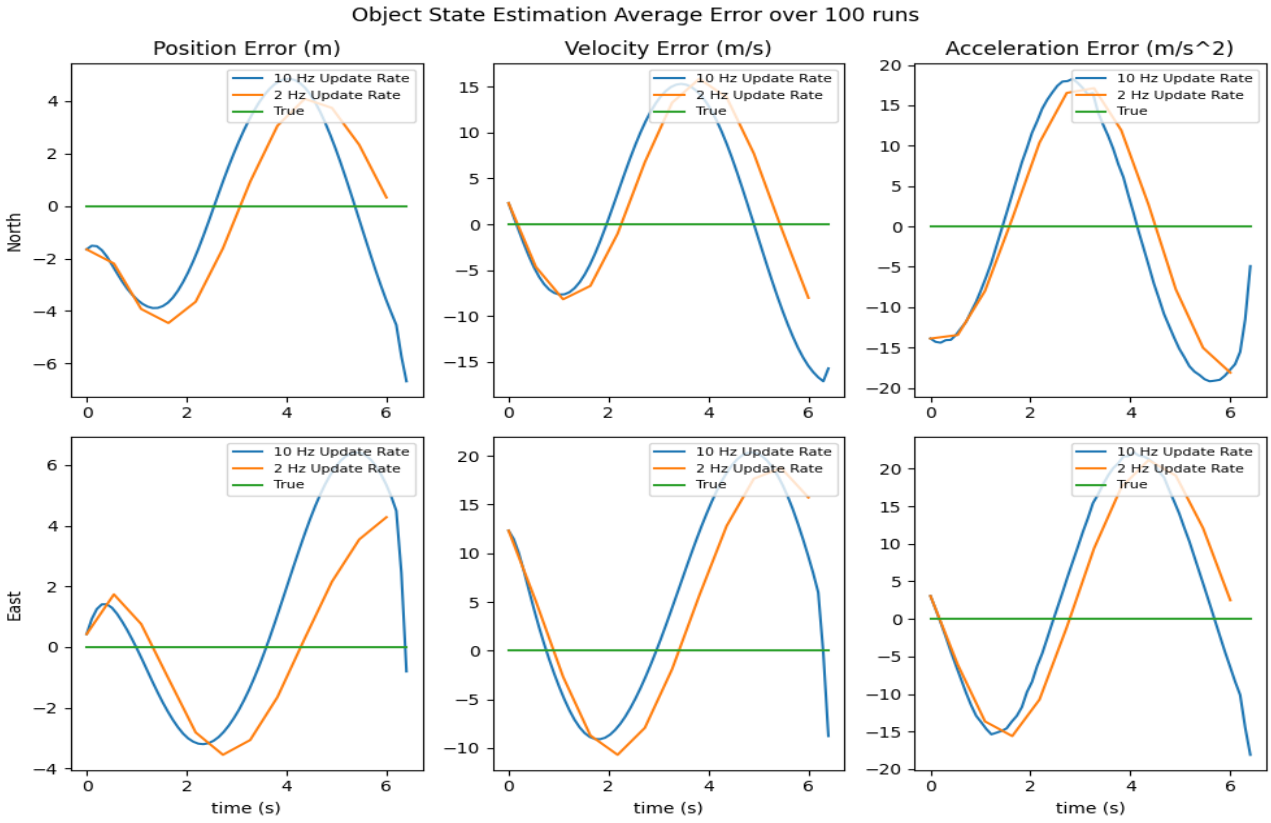
Appendix Z.4. Baseline EKF average estimation error with different measurement update rate (Scenario 3)



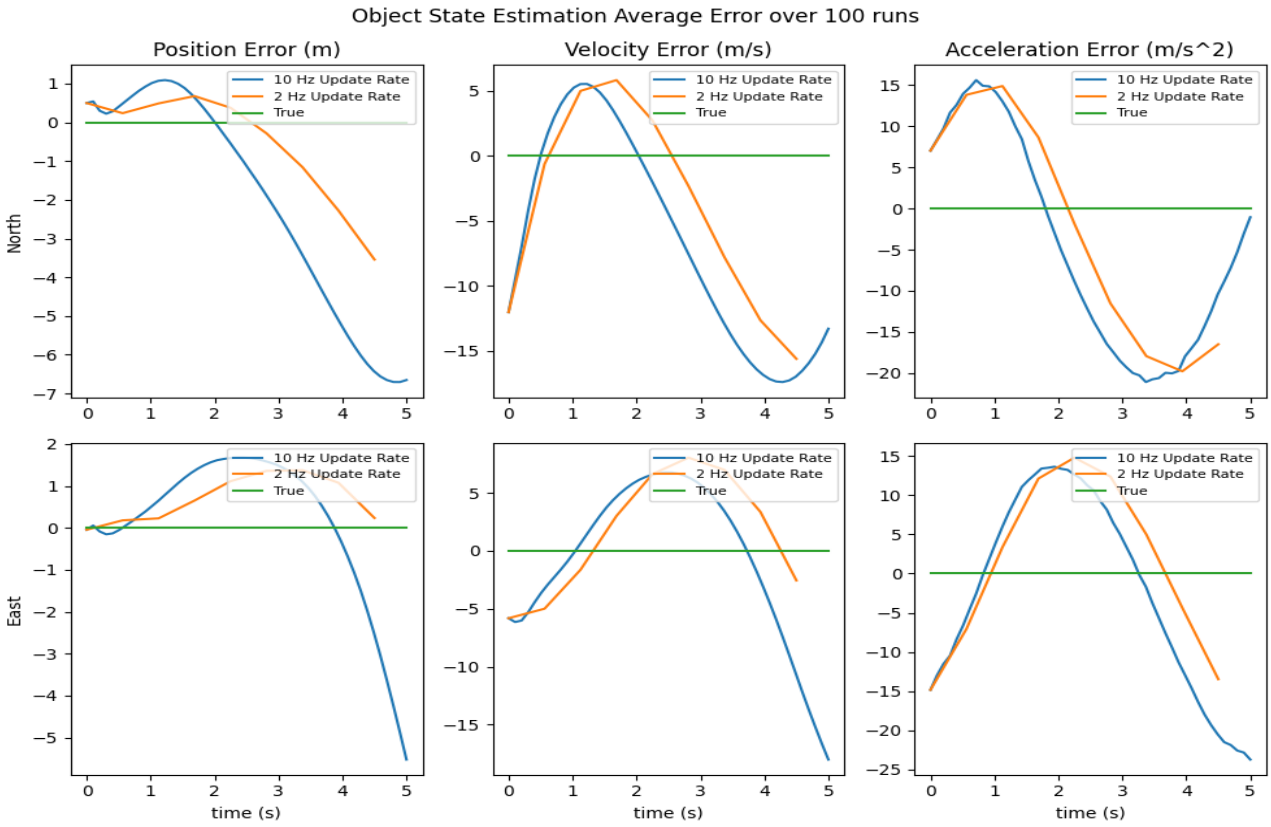
Appendix Z.5. Baseline EKF average estimation error with different measurement update rate (Scenario 4)



Appendix Z.6. Baseline EKF average estimation error with different measurement update rate (Scenario 5)



Appendix Z.7. Baseline EKF average estimation error with different measurement update rate (Scenario 6)



Appendix Z.8. Baseline EKF average estimation error with different measurement update rate (Scenario 7)