Naval Information Warfare Center PACIFIC

TECHNICAL REPORT 3267
FEBRUARY 2022

Drift Improvement with Reinforcement Training – Inertial Sensors – Year 1

Eric Bozeman
Mohammad R. Alam
Minhdao Nguyen
Jeffrey C. Onners
NIWC Pacific

Naval Information Warfare Center Pacific (NIWC Pacific)
San Diego, CA 92152-5001

This page is intentionally blank.

TECHNICAL REPORT 3267
FEBRUARY 2022

Drift Improvement with Reinforcement Training – Inertial Sensors – Year 1

Eric Bozeman
Mohammad R. Alam
Minhdao Nguyen
Jeffrey C. Onners
NIWC Pacific

Administrative Notes:
This report was approved through the Release of Scientific and Technical Information (RSTI) process in October 2021 and formally published in the Defense Technical Information Center (DTIC) in February 2022.

NIWC Pacific
San Diego, CA 92152-5001

NIWC Pacific
San Diego, California 92152-5001

A. D. Gainer, CAPT, USN
Commanding Officer

W. R. Bonwit
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed by the Non-Linear Dynamics & Materials Branch of the Basic & Applied Research Division, Naval Information Warfare Center Pacific (NIWC Pacific), San Diego, CA. The NIWC Pacific Naval Innovative Science and Engineering (NISE) Program provided funding for this Basic Applied Research project.

Released by
John deGrassie, Division Head
Basic and Applied Research Division

Under authority of
Carly Jackson, Department Head
Cyber/S&T Department

ACKNOWLEDGMENTS

Editor: MRM

EXECUTIVE SUMMARY

OBJECTIVE

This report details the system, test environment, and results used to evaluate Reinforcement Learning (RL) algorithms for their ability to reduce the rate of drift in the positional error of an Inertial Navigation System (INS) without aiding from external sensors.

METHODS

A custom RL environment was created to train a RL algorithm to correct the raw inertial measurements from an INS in such a way that the position more closely matched the INS position after being corrected by a Global Navigation Satellite System (GNSS). When GNSS aiding was removed, the RL system would continue to correct the inertial measurements as it was trained to do before GNSS aiding was removed. Multiple RL algorithms were used in the RL system, and their performance was evaluated on their ability to correct inertial measurements to allow for more accurate position solutions (reduce positional error). The algorithms were also evaluated on their use of computer resources and ability to operate in real time.

CONCLUSIONS AND RECOMMENDATIONS

The data collections and evaluations described in this report show that a RL system can help reduce the positional error of an INS without aiding from external sensors (such as GNSS). It also shows that some RL algorithms lend themselves better to this type of system than others. In the end, this research identified two RL algorithms that will continue to be used in further testing related to this work.

This page is intentionally blank.

ACRONYMS

A3C	Asynchronous Advantage Actor Critic
AC	Actor Critic
ARHS	Attitude and Heading Reference System
DDPG	Deep Deterministic Policy Gradient
DIRT-I	Drift Improvement with Reinforcement Training – Inertial systems
DPPO	Distributed PPO
DQN	Deep Q-Network
DVL	Doppler Velocity Logger
ECU	Engine Control Unit
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
MEMS	Micro ElectroMechanical System
ML	Machine Learning
NavFil	Navigation Filter
NUC	Next Unit of Computing
OBD-II	On-Board Diagnostics version 2
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAC	Soft Actor Critic
TD3	Twin Delayed DDPG
TRPO	Trust Region Policy Optimization

This page is intentionally blank.

CONTENTS

APPENDICES

FIGURES

TABLES

This page is intentionally blank.

1. INTRODUCTION

The main objective of the Drift Improvement through Reinforcement Training – Inertial sensors (DIRT-I) project is to extend the holdover time of inertial sensors in the absence of a Global Navigation Satellite System (GNSS), through the use of Reinforcement Learning (RL) or training. For the purposes of this document, the acronyms GNSS and GPS (Global Positioning System) are used interchangeably. The basic concept is to train an RL system with an inertial sensor that is GNSS aided. This will allow the RL system to learn about the inertial data before and after being corrected by GNSS aiding. Once it is sufficiently trained, the GNSS aiding will be disabled to simulate a GNSS-denied environment, and the RL system will provide corrections to the inertial data. This inertial data will then be used to provide the user with a position solution. The idea is that the RL system will look at the sum of all the error sources that can cause inertial sensors to drift, and correct them together to provide an accurate position solution without GNSS than would typically be possible without the RL system. This concept is illustrated in Figure 1.

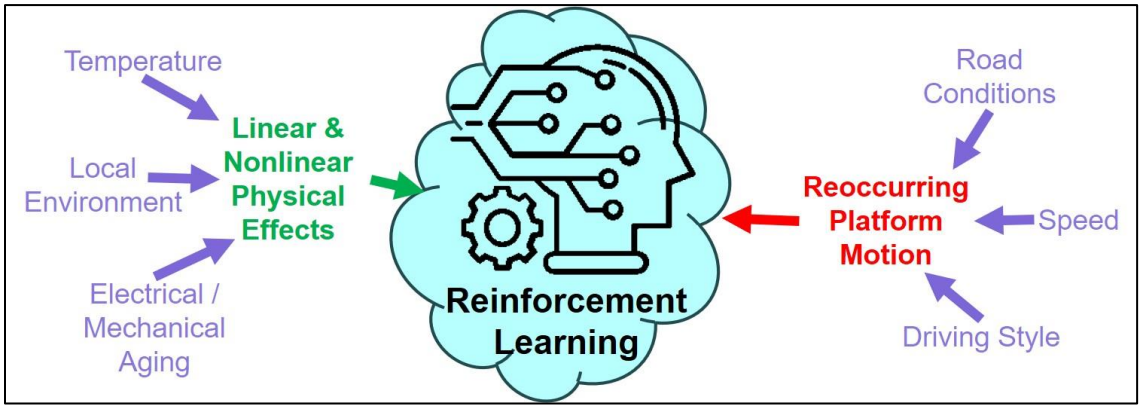

Figure 1. DIRT-I Reinforcement Learning approach.

Since the RL system is trained on the inertial sensor, it creates its own model of the sensor and its noise sources during each training session. This means that the RL system can be used with a wide variety of inertial sensors. Since the error sources are all lumped together, it doesn't matter if they are linear or nonlinear in nature, or if they are platform or user specific noise sources. In other words, an inertial sensor might have something built into it to correct for temperature changes that would adversely affect the performance, but the manufacturer doesn't know anything about the platform the sensor will be used on, so there is no way to correct for error sources that come from any particular platform. This is where the RL system will be able to make performance gains. For the first year of the DIRT-I project the focus was on a ground-based vehicle and an Inertial Navigation System (INS).

This page is intentionally blank.

2. BACKGROUND

2.1 INERTIAL SENSORS

An inertial sensor measures the acceleration and angular velocity of the object the sensor is mounted on. They come in a wide variety of configurations, performance grades, and technologies. They are used for anything from simple tilt or orientation sensors, to precision-guided munitions, and their price range is anywhere from a few dollars to hundreds of thousands of dollars. Popular types of inertial sensor technologies are based on Microelectromechanical Systems (MEMS), fiber optics, piezoelectric materials and magnetic induction. There are no universal standards or even clear definitions for the performance classes of inertial sensors. However, the performance classes are typically broken up in to three or four grades, with "Consumer Grade" being the lowest quality and "Navigation Grade" being the highest. Although they have the lowest performance, consumer grade inertial sensors still have their applications. Table 1 lists these performance grades and their ranges according to VectorNav's website [1] .

Table 1. Comparison of inertial sensor performance grades.

Grade	Cost	Gyro In-Run Bias Stability	GNSS-Denied Navigation Time	Applications
Consumer	< $10	--	--	Smartphones
Industrial	$100 - $1,000	< 10 °/hour	< 1 \minute	UAVs
Tactical	$5,000 - $50,000	< 1 °/hour	< 10 \minute	Smart Munitions
Navigation	< $100,000	< 0.1 °/hour	Several Hours	Military

Inertial sensors are either accelerometers or gyroscopes; and typically consist of a combination of the two. Accelerometers are sensors that measure the linear acceleration of the sensor. Gyroscopes measure the angular velocity (rate of rotation) of the sensor. It is also very common to find magnetometers being used with inertial sensors. Although they don't provide any inertial data, magnetometers can provide a vector that points to the Earth's magnetic North. Beyond the consumer or automotive grades, inertial sensors are typically combined together to form Inertial Measurement Units (IMU), Attitude and Heading Reference System (AHRS), or Inertial Navigation Sensors (INS).

In its simplest form, an IMU consists of a 3-axis accelerometer and a 3-axis gyroscope. Without some sort of onboard processing, the IMU simply provides the linear acceleration and angular velocity of each axis. An AHRS (sometimes referred to as Magnetic, Angular Rate and Gravity (MARG)).

An AHRS is a step up from a simple IMU. An AHRS will have some onboard processing to preform sensor fusion between the accelerometer, gyroscope and magnetometer. These three sensors complement each other very well to provide orientation or attitude of the system. Integrating over time, a 3-axis gyroscope can provide an angular position relative to its previous position. However, that is limited usefulness without some sort of reference. A 3-axis accelerometer is able is able to

determine the direction of the Earth's surface by detecting the acceleration of gravity, which is used as a reference vector. A 3-axis magnetometer's ability to determine the direction of Earth's magnetic North also provides a reference vector. These reference vectors are used to calculate the absolute angular position (a.k.a. attitude or orientation) of the system. Therefore, unlike an IMU that only provides raw inertial measurement data, an AHRS provides the system's attitude which is comprised of the roll, pitch and yaw. An AHRS also provides heading information based on current and previous attitude information. This means that an AHRS can be used to determine relative position, but it needs some sort of starting point as a reference to be useful for true navigation.

A significant drawback to any inertial sensor is the small imperfections in the inertial measurements. These imperfections exist regardless of the quality of the sensors, or the type of technology used. There are many internal and external factors that can cause these imperfections. Some internal factors include temperature changes (self-heating), component variation/mismatch and component aging. Some external factors include temperature, humidity, platform stability (vibrations) and platform dynamics. In navigation systems such as AHRS and INS, these small measurement imperfections become part of the position solution which is used to calculate future position solutions. This means that these measurement imperfections are integrated over time, which causes the position error to grow exponentially as time goes on. This is known as the drift of the inertial sensor. Table 2 compares several inertial navigation sensors.

Table 2. Drift rate comparison of various inertial navigation systems.

Inertial Navigation System	Approximate Price†	Drift Rate (Gyro Bias Instability)††	Type
SBG Systems ELLIPSE-E	$3,500	7°/hr.	MEMS INS
VectorNav VN-110	$7,500	< 1°/hr.	MEMS AHRS
VectorNav VN-210	$12,500	< 1°/hr.	MEMS INS
SBG Systems EKINOX-E	$19,000	≤ 0.5°/hr.	MEMS INS
KVH P-1725 IMU	$9,950	≤ 0.05°/hr.	FOG IMU
KVH GEO-FOG 3D Dual	$30,750	≤ 0.05°/hr.	FOG INS

† *Approximate prices obtained from quotes dated June 2021*

†† Drift rates based on Gyro in-run bias instability from individual sensor datasheets available through the manufacturers' public websites

An INS takes the AHRS a step further and embeds a GNSS receiver (or at least an input for one) to allow that relative position to be converted into an absolute position anywhere on Earth. It is common for an INS to also have other embedded sensors or inputs for sensors that are useful for aiding the navigation calculations such as odometers, Doppler Velocity Loggers, etc. An INS uses some variant of a Kalman Filter to fuse the data from all of these sensors (especially GNSS) to help correct the inherent drift of the inertial sensors. This allows an INS to provide perpetual drift-free attitude, heading, absolute position and velocity solutions. A Kalman Filter is a statistical algorithm that is used in control theory. It uses Linear Quadratic Estimation (LQE) to estimate unknown variables based on a series of measurements observed over a period of time.

The lines between these different inertial sensor systems is often blurred. It is not uncommon to find IMUs with AHRS capabilities, or an AHRS with a GNSS input. These definitions are more like general guidelines.

An INS using a GNSS to bound its drift, forms the backbone of modern navigation systems. This is especially true in high-dynamic platforms. The benefits of combining these systems is that the INS provides attitude and heading information and the GNSS provides absolute position. However, the GNSS is also used to correct the drift of the INS, and the INS is much faster than GNSS so it can fill in the gaps between GNSS updates. Filling in these gaps can actually be of critical importance under non-standard or non-ideal operating conditions. GNSS depends on the receiver's ability to have a line-of-sight to satellites orbiting the Earth. Many operating scenarios, such as tall buildings or trees, deep canyons, or operating indoors or underwater, can cause this line-of-sight to be blocked, rendering the GNSS useless. Additionally, GNSS signals are very susceptible to jamming and spoofing which makes their position solutions useless. It is for these reasons that there has been such a push for GNSS-denied navigation technology in recent years. The goal of the DIRT-I project is to extend the holdover time of inertial systems when GNSS is not available. In this context, the holdover time is the amount of time that an inertial navigation system's position solution is still valid without access to a GNSS. What constitutes a "valid" position solution is operation and/or platform specific. For instance, guided munitions will have much tighter constraints on what is considered a "valid" position than a cargo ship crossing the Pacific Ocean. The standard holdover time is system specific, and depends on the quality of the inertial sensors and other sensors or information that can the systems can use for corrections. One such method for ground-based platforms in the absence of GNSS is called Zero Velocity Updates (ZVU). ZVU relies on a velocity sensor to determine when the system is no longer moving (velocity = zero). This piece of information can be used to suppress the drift from the inertial sensors.

2.2 REINFORCEMENT LEARNING

Machine Learning (ML) has been an increasingly popular "buzz word" over the last few years. Although it shouldn't be applied to every problem, it has been shown to be a useful tool in other areas of navigation such as visual odometry and map reading for autonomous vehicles. Reinforcement Learning (RL) is a subset of ML and optimal control with the goal of applying the best action for an agent to make given the current state of the agent and its environment. Examples of RL usage can be seen in a wide variety of applications from demonstrations of its capability in simple physics problems (e.g.: cartpole and pendulum balancing) and games (e.g.: Go and Chess), to challenges in robotics and computer vision. RL uses continuous feedback to train the system on a dynamic set of data. The DIRT-I project uses an RL algorithm to train a dynamic model of the inertial sensor system in use while it is being aided by a GNSS. Once the GNSS is no longer available, the DIRT-I system will use the model that the RL system created to apply corrections to the inertial measurements. When a proper model has been created, these corrections will be very close to the corrections that were applied as a result of the GNSS-aiding when it was available. Although this solution will not work indefinitely, it will slow the rate of drift and increase the holdover time (the time the inertial system can still provide useful position information without GNSS).

RL is well-suited for this application because it utilized feedback from its environment rather than stored and/or static information. RL enables an agent to learn the best way to act on its environment based on its past experiences, which are known as rewards. Figure 2 illustrates this general concept with a high-level block diagram of a reinforcement learning system.

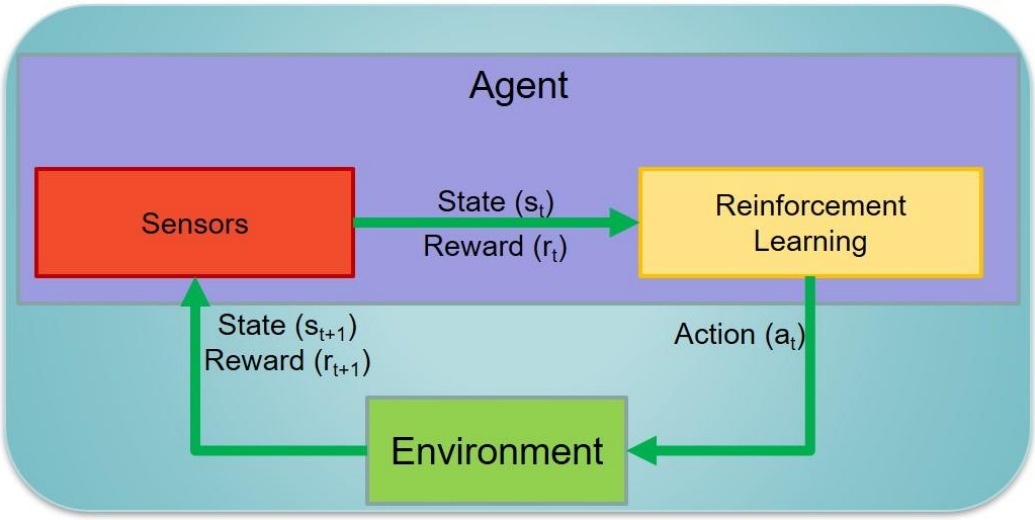

Figure 2. General Reinforcement Learning diagram.

The DIRT-I RL system is a bit different than the generic RL model because the DIRT-I RL system has no way to influence the environment that the inertial sensors are monitoring. Instead, the RL system calculates a position estimate that is compared with the true position from the GNSS-aided inertial system (or the GNSS itself if the inertial system does not compute position solutions – i.e., an IMU).

2.3 EXTENDED KALMAN FILTER (EKF) [2], [3], [4], [5]

IMU data is noisy and cannot be used directly to produce a navigation solution. Kalman Filters can be used to process the data and produce a navigation solution. Kalman Filters can optimally incorporate all the information provided regardless of precision and estimate the values of interest. Kalman Filters do require knowledge of the system and measurement devices dynamics, statistical description of noises and measurement errors, and initial conditions of the values of interest. For this case, the values of interest are position, velocity, and orientation. The knowledge of the system and measurement devices dynamics, statistical description of noises and measurement errors is expressed as a linear mathematical model and error estimations.

Extended Kalman Filters are a nonlinear version of the standard Kalman Filter. Most INS use Extended Kalman Filters because they allow for sensors with nonlinear models to be used. Extended Kalman Filters require more processing, but modern processors are more than fast enough to run in real time.

3. HARDWARE

3.1 INERTIAL SENSORS (KEARFOTT)

For this inaugural year of the DIRT-I project, the focus of the inertial sensors was limited to a single inertial system. The Kearfott KI-4901S is a navigation-grade INS. It is able to receive aiding inputs from a Doppler Velocity Log (DVL), speed sensor, depth sensor and position inputs from a GNSS receiver. The Kearfott utilizes a Monolithic Ring Laser Gyroscope, and has a bias stability of 0.003°/hr. [6]. In addition to providing a position solution, the Kearfott also output raw inertial sensor data from the gyroscope (angular velocity) and accelerometer (acceleration), as well as attitude, heading and velocity information. The Kearfott is configured to require aiding information from a speed sensor. This does improve the position solution from the Kearfott, but the speed sensor information was not directly used by the DIRT-I RL algorithm (speed sensor aiding in the RL system is planned for future efforts). Figure 3 shows the Kearfott using this velocity data to correct its heading. Near the middle of this dataset, an error in the inertial measurements caused a large heading error. The red line shows the error in the heading from NavFil (see Section 4.2), and the blue line is the heading error of the Kearfott. Even though the Kearfott is providing the inertial measurements used to calculate both the Kearfott and NavFil headings, the Kearfott's heading error stays relatively low while NavFil's error grows exponentially. This is because the Kearfott used the velocity data to correct itself, and NavFil had no aiding data to perform a similar correction. The DIRT-I system will help improve this situation, but when comparing the position errors of the DIRT-I system and the Kearfott it is important to keep in mind that the Kearfott is being aided by an external sensor (speed), while the DIRT-I system's NavFil does not have this benefit. This is true in both the training and testing modes.

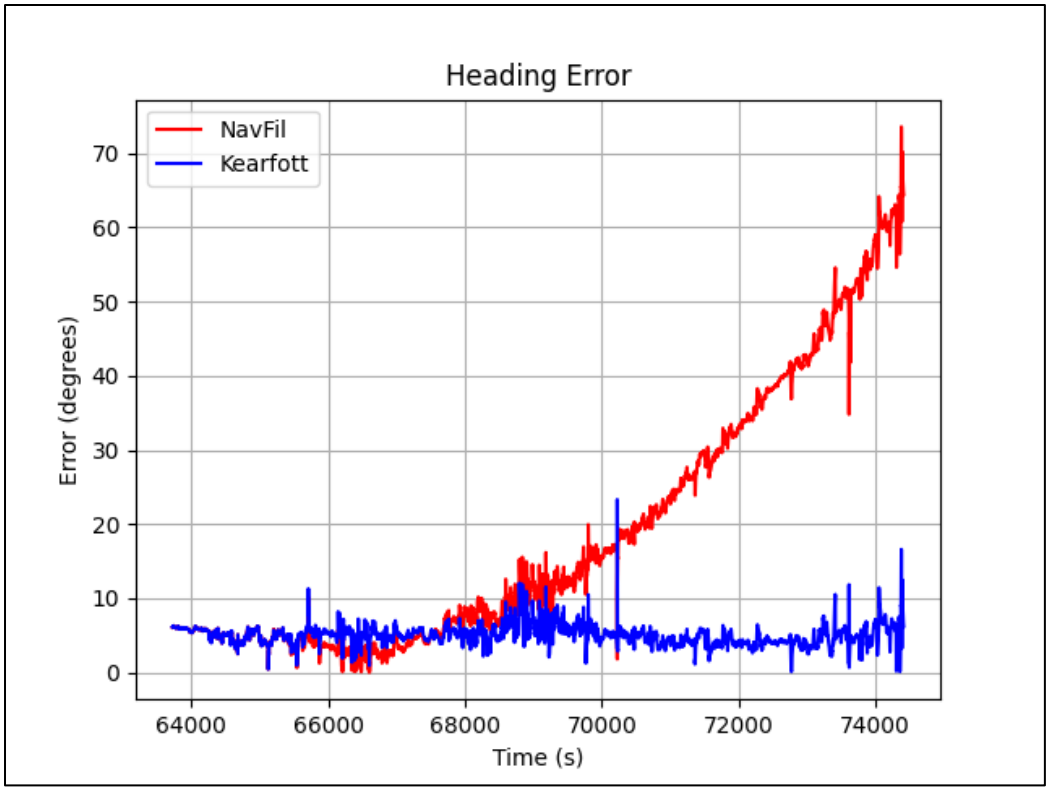

Figure 3. Instantaneous heading error for Kearfott and NavFil over time.

3.2 GPS ANTENNA [7]

The DIRT-I project used the Hemisphere V102 GNSS Vector Compass to provide position inputs to the Kearfott during the training phase, and position truth data during the testing phase. The V102 is a low-profile differential GNSS antenna and receiver that also includes a compass. The differential positioning accuracy is 1.0m, 95% of the time. Although the V102 does include a compass with a gyroscope and tilt sensors, they were not directly used by the DIRT-I project. However, these sensors allow the V102 to maintain accurate position information for up to three minutes in the event that GNSS satellites are temporarily unavailable. Therefore, these sensors were used indirectly by the DIRT-I project, but only as part of the true position solution. They were not part of the inertial sensor testing or analysis.

3.3 SPEED SENSOR (OBDII)

The speed sensor being used by the DIRT-I project is actually the vehicle's own speed sensor. The vehicle measures its speed by monitoring the rotational speed of the wheel(s). Since the diameter of the wheel and tire is known, the linear distance traveled per wheel rotation can be calculated; and the number of rotations in a given period of time provides the vehicle's speed. These calculations are performed in the vehicle's Engine Control Unit (ECU) and displayed to the driver on the vehicle's odometer and speedometer. Since this calculation depends on the ECU knowing the diameter of the wheel and tire, the calculation can be incorrect if tires or wheels are installed on the vehicle that are a different size than what was installed at the factory. Fortunately, this is not the case for the vehicle used in the DIRT-I project.

Most (if not all) modern vehicles have a digital communication port for providing real-time diagnostic information to a connected device. Vehicle mechanics can diagnose problems with the help of devices that can scan this port to read ECU information in real time, or look at the standardized Diagnostic Trouble Codes (DTCs) stored in the vehicle's memory. One of the most common interface standards for this port on vehicles currently and recently sold in the United States is called On-Board Diagnostics version 2 (OBD-II). The DIRT-I project uses a module that plugs into the OBD-II port and translates the real-time data over Bluetooth or USB to the processor. The processor uses Python's Python-OBD library to read the vehicle's speed data, which is logged and sent to the Kearfott via one of its serial ports.

3.4 PROCESSOR (NUC, DATALOGGER, REAL-TIME UPDATES ON LOCAL SERVER)

The processor being used for logging data and displaying real-time updates on the DIRT-I project is one of Intel's® Next Unit of Computing (NUC) systems. Specifically, it is the NUC8i7BEH model, which is based on the 8th generation Intel® Core™ i7 processor. This processor is the central hub of the DIRT-I real-time system. It receives data from the Kearfott INS, GNSS receiver and OBD-II port scanner. It logs all of this incoming data for post-processing analysis and also sends the speed and position information on to the Kearfott. In addition to logging all the DIRT-I data, this processor will also be responsible for running the DIRT-I software during real-time operations in the next phase of the project. The processor also hosts a local web server that provides up-to-date status information over a shared wireless network. This allows users to view a map display with an overlay of the vehicle's current positions as presented by the GNSS receiver and Kearfott. Once the real-time version of DIRT-I is implemented, this will include an overly of the DIRT-I position solution as well. In addition to the map display, other status information is also displayed such as calibration status,

the time of the last GNSS fix and whether or not the Kearfott is currently using the velocity information. The local web server also allows limited control over the data logging system. This is currently limited to enabling/disabling GNSS, which allows the GNSS position data to continue to be logged, but prevents it from being sent to the Kearfott to simulate a GNSS-denied scenario.

This page is intentionally blank.

4. SOFTWARE

4.1 OVERVIEW/TOPOLOGY

The DIRT-I system consists of an RL system and two custom Navigation Filters (NavFils). The RL system can be configured to run a variety of RL algorithms using the data received from the inertial system and/or other aiding sensors (only inertial inputs currently supported). The NavFils are Kalman Filters that provide a position solution and help the RL system model the Kalman Filter inside the inertial system (Kearfott). Both NavFils are copies of each other with the difference that one is aided by GNSS (during training mode only), and the other is always denied GNSS aiding. This allows the RL system to more closely model the inertial sensors because it doesn't need to model the inertial sensors plus the Kearfott's Kalman Filter. Ideally, the NavFil would be an exact copy of the Kalman Filter used in the Kearfott, but since that is proprietary information, this is a close approximation. The advantages of this particular setup are that the instances of NavFil can be replaced with more robust or better matching Kalman Filters if they become available, and inertial sensors without their own Kalman filter can still use the DIRT-I system by simply adding a third NavFil instance to the inertial sensor external to the DIRT-I system. The current test setup is only configured to run in a post-processing environment where all sensor data is logged during data collection runs, and then read into the DIRT-I system at a later date. However, the next phase of the project will implement a real-time DIRT-I system, leaving only the evaluation and analysis aspects for post-processing. Figure 4 shows a top-level block diagram for the DIRT-I test environment. The DIRT-I system receives (pre-collected) data from the GNSS receiver, Kearfott's inertial sensors, and Kearfott's Kalman Filter (position solution). It uses this data to train the RL system which creates a model of the Kearfott's inertial sensors while incorporating its own Kalman Filter models (NavFils). A GNSS-denied scenario can be simulated with a software switch that disables the GNSS output to both the DIRT-I system and the Kearfott. However, the GNSS position data is still logged for post-processing analysis. Position estimates from the unaided Kearfott and the DIRT-I system are also logged for post-processing analysis. Both of these position estimates are compared to the true GNSS position at each sampling timestep, and the position error is calculated as the difference between each position estimate and the GNSS position.

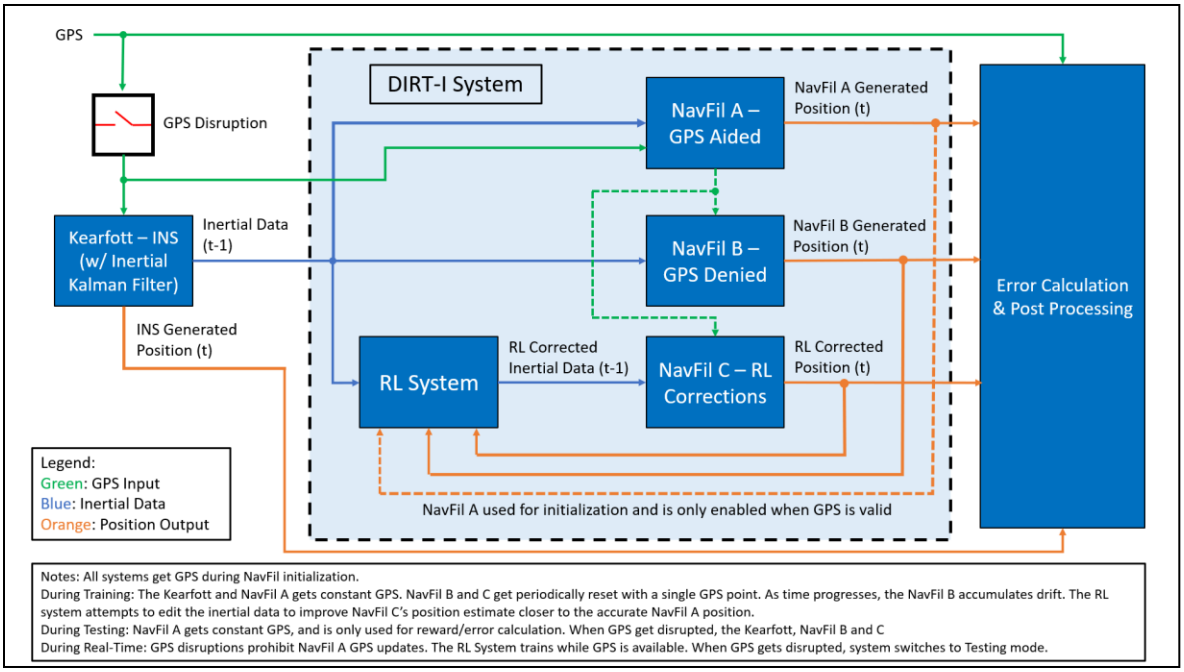

Figure 4. Block diagram of DIRT-I system and evaluation setup.

The DIRT-I system has three operational modes:

1) Training Mode:

The RL system makes corrections to the raw inertial data from the inertial system (Kearfott), and generates a position estimate from both the aided and unaided NavFils. These estimates are compared to the position solution from the inertial sensor which is being aided (corrected) by the GNSS receiver.

This mode can only be enabled when a valid GNSS position solution is available to both the inertial system and the DIRT-I system.

2) Testing Mode:

The DIRT-I system receives the raw inertial data from the inertial sensor and makes corrections to it based on the model that the RL system defined while the system was in Training Mode. The position data from the inertial system and the "aided" NavFil are ignored in Testing Mode. The unaided NavFil receives the altered inertial data from the RL system and generates a position estimate, which is logged as the position solution from the DIRT-I system.

This mode can only be enabled when there is no valid GNSS position solution available.

3) Skip Mode:

This is a catch-all, fail-safe mode where the system neither updates the RL model or provides a position solution. It preserves all system states as they are, and addresses issues that arise from essentially pausing the system in one location, and unpausing it in a different location.

This mode will be entered if there is an issue with any of the aiding sensors. This could be a problem with the speed sensor data not being available, or an unreliable GNSS position solution that is not the result of a real GNSS-denied scenario. For instance, when passing tall buildings, the number of visible GNSS satellites may drop to the point that the position is still being provided, but

at an increased error. In this case, training could be detrimental but a DIRT-I position solution may be unnecessary (if situation only persists a short time) or unreliable (if not enough training has occurred yet).

4.2 NAVFIL

NavFil is a software suite developed at NIWC Pacific that implements an Extended Kalman Filter to produce a navigation solution. NavFil requires acceleration and angular velocity from an initial measurement unit (IMU) to produce a navigation solution, but it can also utilize data from other sensors like GPS, magnetometers, air data sensors, and speed sensors to produce a better solution. NavFil has a flexible design to easily utilize different IMUs of various grades, and different GPS, magnetometers, air data sensors, and speed sensors.

4.3 REINFORCEMENT LEARNING SYSTEM

4.3.1 RLzoo/OpenAI-Gym

Our RL System can be broken into two parts: the algorithm framework (RLzoo/Gym) and the Gym Environment (*DirtEnv-v1*). "RLzoo is a collection of the most practical reinforcement learning algorithms, frameworks and applications." [8]. OpenAI-Gym (Gym) "is a toolkit for developing and comparing reinforcement learning algorithms." [9]. WGym is commonly used to emulate and learn how to solve video game environments, including the popular Atari games Pong and Space Invaders [10]. RLzoo provides the algorithms to run a Gym environment in order to learn which sequence of *actions* and *states* earn high *rewards*.

As an example, imagine a game of Pong: positive rewards occur when the RL's AI scores points, and negative rewards occur when its opponent's AI scores points. The system makes correlations between states (the paddle's position in relation to the ball) and actions (moving the paddle up or down). Each Pong game (referred to as an *episode*) generates rewards for every action to a state, and the cumulative reward adjusts the *policy network*. Simply put, when the reward of an action is high, the network saves this state/action pair as a 'good' solution; if this state occurs again, the RL can produce an action to receive another high reward.

A simple algorithm might be able to determine that less negative rewards occur when its paddle is closest to ball. A more advanced algorithm may discover that it can score higher rewards when its paddle hits the ball at an angle, making it harder for the opponent to hit the ball back. No specific algorithm is a 'one size fits all' solution; different algorithms may perform better or worse on a specific environment, mainly depending on the environment's complexity and number of dimensions. RLzoo allows us to easily test a variety of configurations and figure out which algorithms can perform best for a navigation-based environment.

4.3.2 DirtEnv-v1

The DirtEnv-*v1* Gym environment was developed to represent a latitude/longitude map. The goal is to input a GPS-Denied vehicle path into the map environment and output a predicted 'correct' path closer to the vehicle's actual GPS position. While in training mode, the system operates as follows: Three NavFil instances are initialized: one GPS-Aided (NF_{aided}), one GPS-Denied (NF_{denied}), and one RL-Aided (NF_{rl}) – See Figure 4. Each are given the same sequence of inertial states (S_I=[$Acceleration_{X,Y,Z}$, Angular $Velocity_{X,Y,Z}$, Velocity]) and same initial position state. Each NavFil will use the inertial states to generate a sequence of position states ($S_{P,(aided,denied,rl)}$= [latitude, longitude]). NF_{aided} will get periodic (1 Hz) GPS updates to correct its position estimates. NF_{denied} and NF_{rl} do not

get GPS updates, causing its position estimates to drift over time. The RL system applies adjustments (actions) to $S_{I,rl}$ in an attempt to produce a $S_{P,rl}$ closer to $S_{P,aided}$.

The environment reads in data for each episode with a user-set duration. It is important to confirm that positional drift can occur between NF_{aided} and NF_{denied} within the episode duration. There should also be episodes where minimal drift occurs, so that the RL will not overfit or assume drift that is not actually occurring. It was found that an episode duration of 20s was sufficient in fulfilling these criteria. During training, NF_{denied} and NF_{rl} load the 'saved state' of NF_{aided} at the end of beginning of each episode, ensuring that the difference between NF_{aided} and NF_{denied} for the upcoming episode are solely related to positional drift. During testing, these NavFils do not load the NF_{aided} 'saved state' because NF_{aided} is no longer receiving GPS positions. Instead, they continue from the previous episode, and the drift accumulates as it would occur in a real GPS-Denied scenario.

This NavFil implementation was chosen as the method for calculating the position solutions, instead of Kearfott INS and direct GPS for a few reasons. 1) The NavFil is independent of the inertial system being used, allowing the generation of multiple models based on each system's specific IMU output accuracy. For instance, inertial outputs from the nav-grade Kearfott INS will likely require less RL adjustments than a lower end IMU. The DIRT-I system's NavFil configuration allows us to create separate models for each IMU (one for Kearfott, another for the lower-end IMU), ultimately increasing research and testing capability. 2) GPS-Aided NavFil operates at a higher frequency (50Hz) than GPS (1Hz). The RL system makes actions at every state, so if the lower frequency GPS were used as ground truth, the DIRT-I system would need to interpolate the points to 50Hz to generate step rewards. Interpolated GPS would ultimately cause an inaccurate ground truth, whereas the NF_{aided} position estimates produce a more accurate ground truth. 3) Using multiple NavFils to generate all necessary RL training data allows the most accurate detection of positional drift. Using a mix of GPS-Aided Kearfott, GPS, and NavFil for position estimates would inherit additional position errors that would make learning positional drift more difficult.

The following equations represent the Dirt-Env environment:

Equation 1: Observation Space

$$\begin{aligned} &Box\ (low = [Acceleration_{X,Y,Z},\ Angular\ Velocity_{X,Y,Z}\ Min.\ Bounds], \\ &\quad high = [Acceleration_{X,Y,Z},\ Angular\ Velocity_{X,Y,Z}\ Max.\ Bounds]) \end{aligned}$$

Equation 2: Action Space - Box

$$\begin{aligned} &Box\ (low = [Angular\ Velocity_{Z}\ Min\ Action\ Bound], \\ &\quad high = [Angular\ Velocity_{Z}\ Max\ Action\ Bound]) \end{aligned}$$

Equation 3: Action Space - Discrete

$$\begin{aligned} &Discrete\ (linspace\ (Angular\ Velocity_{Z}\ Min\ Action\ Bound, \\ &\quad Angular\ Velocity_{Z}\ Max\ Action\ Bound,\ n=11)) \end{aligned}$$

Where "Box" and "Discrete" are data-structures within the Gym module. The Box space contains continuous float values from a minimum to maximum bound. The Discrete space represents an array (length n = 11) of allowable float values. Some algorithms (DDPG, TD3, SAC) only allow for a Box action space. DQN only allows for a Discrete action space. The remaining algorithms allow for either Box or Discrete action space (algorithms in the results section used the action space type that performed better during preliminary tests).

The observation space dictates what the RL system can see at each step. Because the Kalman Filter uses current inertial data to estimate its next position, we look at the inertial data at *t-1* and position at time *t*. The first attempt was to feed the RL the raw data, but it was found that the system generally had difficulty solving the environment. This challenge most likely arose from the algorithms attempting to relate the action space to a specific inertial measurement. As a result, each algorithm was calculating how to best adjust the inertial data without information on the change in inertia and was unable to see any error caused by the drift with this state definition. To accommodate this issue, the observation space was modified to contain the difference between inertial states $S_{I,t}$ and $S_{I,t-1}$. This change produces a greater variation between states, and noise/drift is more recognizable for the algorithms. This new state variable ($S_{I,t}$ - $S_{I,t-1}$) is denoted as S_{RL}, since these are the states that the actual RL algorithms see. Minimum and maximum bounds were chosen based on the average derivative of inertial data from the data-collections: X, Y and Z acceleration between –10 and 10, X and Y angular velocity between -0.45 to +0.45, and Z angular velocity between -0.05 and +0.025.

The action space does not include position, since the position is generated directly from the original inertial data and the action states. Originally, an action space was used that affected acceleration and angular velocity for each axis (6 dimensional). However, this caused an issue where algorithms produced the same action for almost every state. It is likely that not enough action sequences can be reasonably tested with this specific environment when using multiple continuous action dimensions. Limiting the action space to only affect one dimension produced a wider diversity of actions for almost every algorithm. Therefore, it was decided to use actions only to the Z-axis Angular Velocity since it plays the greatest role in a vehicle's heading (negative actions rotate the vehicle clockwise, and positive actions rotate the vehicle counterclockwise).

Rewards are generated by comparing the distances between $S_{P,rl}$, $S_{P,aided}$, and $S_{P,denied}$ at each step. Multiple reward functions (RF) were set up to test the DIRT-I system:

RF 1) Distance Function (Meters) From NF_{aided}: $-1 * Dist\,(S_{P,rl}\,,\, S_{P,aided})$
RF 2) Position Improvement Over NF_{denied}: $Dist\,(S_{P,denied}\,,\, S_{P,aided}) - Dist(S_{P,rl}\,,\, S_{P,aided})$
RF 3) Hybrid: $(Dist\,(S_{P,denied}\,,\, S_{P,rl}) - Scalar{*}Dist\,(S_{P,rl}\,,\, S_{P,aided})$
RF 4) Heading Improvement Over NF_{denied}: $Scalar{*}(Yaw\,(S_{P,denied}\,,\, S_{P,rl}) - Yaw(S_{P,denied}\,,\, S_{P,rl}))$

where '*Dist*' is a function that calculates the distance between NavFil positions and '*Yaw*' is a function that subtracts the heading angle between NavFil instances. RF1 only takes the distance between NF_{rl} and NF_{aided} into account. A potential issue here is that even if the RL takes actions that greatly improve from NF_{denied}, the resulting reward can still be very negative, and interpret the result as a bad action. For this reason, the remaining reward functions add positive rewards when NF_{rl} has a shorter distance from NF_{aided} when compared to NF_{denied}. During RF2 preliminary tests, it was found that the system would generate consistent positive rewards, indicating that the RL corrections make improved position estimates over the GPS-Denied NavFil. However, the NF_{rl} position estimates appeared to have only minimal divergence from NF_{denied}, indicating that this RF teaches the system in which general direction the position drift takes place, but not enough to have any substantial improvement. RF3 rewards large positional adjustments to NF_{denied}, while punishing distance between NF_{rl} and NF_{aided}. This should address the issue from RF2 where even miniscule corrections from NF_{denied} are highly rewarded. Nevertheless, because minimizing the distance between NF_{rl} and NF_{aided} should be prioritized, the distance from NF_{aided} is multiplied by a scalar (a value of 6 was found to be effective). RF4 rewards making actions that reduce the angle of the heading (yaw) between NF_{aided} and NF_{rl}. Angles higher than NF_{aided} and NF_{denied} produce a negative reward. Small differences in yaw can have a huge effect on the total position error, so the reward is multiplied by a scalar to

punish and reward the system accordingly. RF4 was found to be most effective for limiting cumulative positional drift and was used for all algorithms seen in the results section.

4.4 RL ALGORITHMS BEING TESTED

Algorithms were chosen from those included in the RLzoo python module. RLzoo groups the algorithms into the categories: value-based, policy-based, and actor-critic. Full descriptions of each algorithm are listed below. One value-based (DQN), three policy-based (TRPO, DDPO, PPO), and four actor-critic (AC, SAC, TD3, A3C) methods were tested. Of these algorithms, four algorithms (SAC, AC, TD3, DDPG, TRPO) use a single agent while the remaining (A3C, DPPO, PPO, DQN) take advantage of multiple agents for exploring the environment. Asynchronous testing is possible for post-processed testing, but this may prove challenging to employ in a real-time environment. This section provides a brief description of each RLzoo algorithm that was assessed. Default hyperparameters, or additional built-in parameters each model has for controlling its learning, are used for every model used [11].

4.4.1 Actor-Critic (AC) [12]:

The Actor-Critic (AC) approach is an on-policy algorithm that optimizes a dual problem using two function approximators: (1) an "actor" seeks to find the best action given the current systems state and (2) a "critic" that evaluates the actors actions the action chosen, state, and environmental information. By working in tandem, the actor and critic should improve at their respective roles with training and weight optimization [12]. AC has been shown to have great success in a variety of reinforcement learning tasks [13] .

4.4.2 Soft Actor Critic (SAC): Main paper [14] and follow-up paper [15]

Soft Actor-Critic (SAC) shares a similar core premise as that of AC in that it seeks to optimize actions chosen by the actor and evaluations made by the critic, but the key difference lies in what actions are made during the training phase. SAC switches from an on-policy to an off-policy approach for exploring the environment, which means that other actions can be employed instead of always using the actions from the actor's learned policy. This algorithm aims to maximize the reward while maintaining policy entropy for exploration by applying randomness to that policy. This method has been shown to be sample efficient by achieving optimality with less data, while remaining stable.

4.4.3 Asynchronous Advantage Actor Critic (A3C): [16]

Asynchronous Advantage Actor Critic (A3C) is another variation on the AC algorithm. This uses multiple actors exploring the same space and using the same policy. Each actor is initialized differently. The critic compares the known reward value of the current state with its predicted value based on each actor's action to give each action a boost based on if it was greater than the known value. Unlike SAC, A3C is still an on-policy approach, but it can be modified to utilize entropy for the policy updates to increase exploration. This approach has been shown to perform better than AC and can make use of parallel computing to speed up training time.

4.4.4 Deep Deterministic Policy Gradient (DDPG) [17]:

Deep Deterministic Policy Gradient (DDPG) is a variation on the AC algorithm, but it is easier to compare DDPG to SAC since both are off-policy methods and both employ an alternative strategy when choosing actions. While SAC estimates a probability distribution over the action space, DDPG uses a more deterministic approach by directly calculating the appropriate action at each step. What

prevents this from becoming AC is in how that action is specifically decided on, and in its usage of an off-policy approach as opposed to an on-policy one. This method is still able to perform well while maintaining a smaller and constant hyperparameters and structure.

4.4.5 Twin Delayed DDPG (TD3) [18]:

Twin Delayed DDPG (TD3) is an extension to DDPG with the goal of handling a few shortfalls that arise from DDPG (1) exploiting errors by overestimating the reward values associated with each action for a given state and (2) having error accumulate due to variance in policy estimation. To tackle the overestimation, TD3 introduces a clipped double Q-learning function so that it learns two functions for estimating the reward values and updates with the smaller of the two. For mitigating error accumulation due to variance, TD3 uses a delayed policy update and target policy smoothing by adding noise to the target action. Altogether, these updates have been shown to reduce per-update error discovered in the original DDPG and improves performance.

4.4.6 Trust Region Policy Optimization (TRPO) [19]:

Trust Region Policy Optimization (TRPO) is an alternative approach to the RL optimization problem that focus on the local optimization of the policy. TRPO is an on-policy approach which aims to guarantee a monotonic increase in performance by using a trusted region approach as opposed to a gradient approach. Where gradient descent or ascent adjusts the parameters based on a present learning rate toward the appropriate critical extrema, trust region methods use a defined search radius around the current parameters to find the optimal parameter in this space. TRPO makes use of this trust region approach to find the next best policy. This approach has been successful in a variety of RL tasks such as robot localization and video games.

4.4.7 Proximal Policy Optimization (PPO) [20]:

Proximal Policy Optimization (PPO) is a simpler, modified version of TRPO. The changes made in PPO are specific to the trust region constrained optimization problem. TRPO's objective function is complex and requires extra computation to be performed since it requires calculation of a second order derivative for calculating the Kullback-Leibler (KL) divergence between the new and the old policies. This is where PPO’s core modifications lie. PPO-Penalty is one implementation that uses a family of first order derivatives to approximate the KL divergence. It also utilizes a coefficient to control its effect in the objective function, which makes it a soft constraint to act as more of a penalty. PPO-Clip, however, removes the KL-divergence entirely and essentially uses a ratio between the old and new policies to restrict how much the policy changes. As a result, both methods of PPO prove to be easier to implement and less computationally intensive without decreasing performance.

4.4.8 Distributed PPO (DPPO) [21], [22]:

Like A3C, Distributed PPO (DPPO) takes advantage of parallel computing to improve performance. There are multiple variations on DPPO, where multiple agents each are trained with one policy or multiple policies. This allows for faster convergence to an optimal policy and, thereby, reducing training time.

4.4.9 Deep Q-Network (DQN) [23]:

Deep Q-Networks (DQN) use a cycle of recording state, action, reward, and next state (SARS) pairs in a replay buffer and then sampling from the replay buffer to update the network. As implied by the name, a DQN is a neural network that estimates the true reward values associated with each

state-action pair. The output of the network is a list of values where each corresponds to a specified action. Network weights are updated using a gradient descent approach on the loss between the estimated true reward and the predicted reward by the DQN. If it is the final state, the true reward is simply estimated as the recorded SARS pair's reward. When this is not the final state, or in other words the terminal state, it is estimated from the sum of the reward and the maximum reward of the next state across the entire action space. DQNs allow for calculation of the best action and their associated reward at each state using a single pass through the network. This approach has shown to perform similarly to that of a professional human game tester on multiple video games.

4.4.10 Algorithm Selection:

All algorithms were run through preliminary tests to rule out algorithms with obvious issues. For instance, while DPPO and A3C have the added benefit of running multiple datasets in parallel to speed up learning, this causes issues with our future real-time implementation. In this instance, only one data-stream would be available, making asynchronous leaning unnecessary. This could prove useful for pre-training on multiple datasets prior to real-time training in year-two, but at the moment these algorithms were disregarded. AC was ignored as it is an older and less sophisticated implementation of many of the other provided algorithms. While SAC showed decent accuracy in preliminary tests, it is ultimately very slow, and therefore could not reliably be used in a real-time setting.

Only a couple algorithms from each group were chosen (Actor-Critic/Policy-Based, Deterministic/Stochastic, and Off-Policy/On-Policy). Ultimately, the algorithms that were evaluated were DDPG and TD3 (Both Actor-Critic, Deterministic, and Off-Policy), DQN (only Value Based algorithm), as well as PPO and TRPO (Both Policy-Based, Stochastic, and On-Policy).This will help narrow down differences between each category in the environment; for example, if both DDPG and TD3 do better that PPO and TRPO, then it is likely that the actor-critic, deterministic, and off-policy categories work better than policy-based, stochastic, on-policy categories for our environment. Additionally, as mentioned in Section 4.4, PPO has two alternate implementations, Clip and Penalty. This was used as an opportunity to compare discrete and box spaces on incredibly similar algorithms (for these tests, PPO Clip used the 'discrete' action space and PPO Penalty used a continuous 'box' action space). Each algorithm tested used the default hyperparameters provided by RLzoo.

5. TEST PROCEDURE

5.1 DATA COLLECTION

The first year of the DIRT-I project focused on setting up the RL environment and evaluating algorithms, so the DIRT-I system was not run in real-time. In order to test the environment and evaluate the RL algorithms, inertial data needed to be collected for both training and testing modes. Unlike many machine learning projects, the data required for DIRT-I was very specific and could not be found in a publicly available library or collection. The DIRT-I system required raw inertial measurement data from a single unit, and collected under very similar circumstances (mounted to same vehicle, being driven by the same driver, under the same weather and traffic conditions). DIRT-I also required GNSS position data that correlated to the inertial data in both time and space. Therefore, it was necessary for project team members to collect the data themselves.

The vehicle platform used throughout the first year of the DIRT-I project was a 2014 Dodge Grand Caravan. This vehicle is a shared asset for the entire division, so the data collection equipment needed to be installed, then removed for each data collection event, and no permanent alterations could be made to the vehicle. In order to get quality data from the inertial system, it needed to be rigidly mounted in such a way that it could not move relative to the frame of the vehicle. In order to install and remove the equipment quickly and easily, a custom equipment mount was designed and fabricated. This mount allowed all the data collection equipment to be mounted to a single platform that could easily be taken in and out of the vehicle together. The equipment mount was secured to the vehicle with just four simple toggle clamps that latched to brackets on the floor. These brackets normally kept the rear seats locked into place, but when the seats were folded down flat, the brackets were available as “tie-down” points. After some initial adjustments, the equipment could be installed in just a couple minutes. By comparison, the initial data collection event took 30-45 minutes to get the equipment installed. Ratchet straps were used to secure the Kearfott, but it was difficult to secure without the straps interfering with the buttons and switches on the front panel. Figure 5 shows the equipment mount, installed in the vehicle, with only the Kearfott inertial system mounted to it. Additional plates can be attached to the equipment mount to make room for more equipment. However, these plates are not installed in Figure 5, so that the clamps can be seen more easily. Figure 6 shows a close-up of one of the toggle clamps latched down to a bracket in the vehicle.

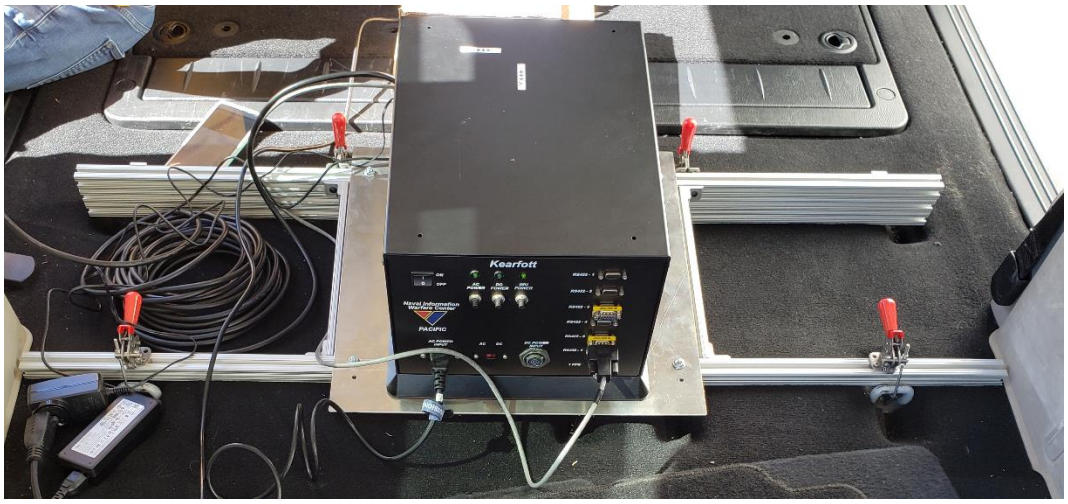

Figure 5. Kearfott mounted in vehicle using equipment mounting system.

Figure 6. Close-up of toggle clamp for equipment mounting system.

The only equipment that wasn't attached to this mount was the GNSS receiver/antenna, and the power source for the collection equipment. The GNSS receiver/antenna was mounted to the roof with a custom-made magnetic mount. The magnets had rubber feet so they would hold the antenna without slipping, and not scratch the vehicle's paint while mounting/removing the antenna. The antenna was mounted on the roof directly above the inertial system. The power source was a Duracell 1440-watt portable power station (a.k.a. a gasless generator). The decision was made to use a power source that was isolated from the vehicle in order to eliminate the possibility of crosstalk and noise from either the vehicle or the DC-AC inverter that would have otherwise have been used. This portable power station was not mounted with the rest of the equipment simply due to weight (it would have made the full equipment mount too heavy to be useful).

The data collection events were each comprised of two different runs. The first run was with the GNSS enabled, and all systems operating normally. The second run simulated a GNSS-denied event and GNSS positions were only recorded for post-production analysis. The inertial and DIRT-I systems did not receive GNSS positions during the second run. The standard procedure for the data collection events was to install the equipment in the vehicle, then move the vehicle to a relatively level surface to calibrate the Kearfott. The Kearfott Calibration takes about one hour to complete. After initialization, the vehicle was driven to the starting point. The starting point for each run depended on what was being testing, and whether or not the vehicle's fuel tank needed to be filled before the run. Once at the starting position, data logging was enabled and the vehicle was placed in park for about 15 minutes. This ensured everything was running correctly and allowed the data logger to collect initialization data for post-processing analysis. After the initialization period, the vehicle was driven around just like it would be under normal operation with GNSS positions provided to all systems. The exact path the vehicle was driven changed from run to run, but typically included some freeway driving and some surface street driving. Some data collection events were mostly freeway driving, to collect data on relatively straight and constant speed paths. Other data collection events were mostly preformed on surface streets to collect data with a lot of stops and turns, and various speeds. The run lasted between 30 and 90 minutes, and ended back at the starting position with the vehicle in the same orientation it was in at the beginning of the run. The vehicle was placed in park for another 15-minute initialization period, and then the second run began. At the start of the second run, GNSS was enabled and all systems were operated normally. After about five minutes, the GNSS was disabled so that it did not provide position information to the inertial or DIRT-I systems. This was done to ensure the systems were still working properly once the vehicle started moving, and to simulate the transition from a GNSS-enabled state to a GNSS-disabled state. The path for the second run was always similar, but not exactly the same, as the path for the first run.

The time and distance of both runs were roughly the same, and approximately the same amount of time was spent on surface roads vs. freeways in both runs.

5.2 POST PROCESSING ANALYSIS

For this first year of the DIRT-I project, all data was processed and analyzed after being collected. This allowed multiple algorithms to be compared, using the same data without running all of them at the same time (which would not have been possible on the computer being used in the field). Next year, a single algorithm will be chosen which will be run in real-time, to more closely represent an operational environment.

During the post processing analysis, the data collected from training data collections was used to train the RL algorithms being evaluated. Then the data collected from the testing data collections was used to test the trained RL algorithms, and analyze their performance. The position solutions from each RL algorithm were compared to the positions logged from the GPS. These comparisons occurred at each GPS time step (1 Hz). For the purposes of this document, the positions logged from the GPS receiver and the GPS-aided NavFil are considered to be the same. See Section 4.3 for a more detailed explanation.

The results were plotted as position comparisons on longitude vs. latitude plots where the GPS positions were overlaid on the same plots as position solutions from the RL algorithms, GPS-denied NavFil, and Kearfott INS (see Appendices). The difference, in meters, between GPS positions and the position solutions from the RL algorithms, GPS-denied NavFil, and Kearfott INS, were calculated. These differences are considered to be the position error of the various position solutions. The position errors for each system were overlaid on the same plot showing position error vs. time (see Appendices).

The instantaneous position error at each time step can be misleading because the error can get smaller by sheer coincidence. On a plot, this may give the impression that the error is being corrected, or otherwise improving. Figure 7 shows the instantaneous position error for the NavFil and Kearfott while being denied a GPS signal. Near the end, the NavFil (blue line) has a large drop in error that makes it look like its performance is improving. However, this is followed by a large spike in error because the error wasn't actually being corrected. To avoid these misleading plots, the position error plots (see Appendices) actually show the cumulative average position error vs. time, where the average error is calculated at each time step and then summed together with previous average errors. This scales the cumulative error to a reasonable value that represents the position error at each time step, without the ability to drop and give the impression that the error has been corrected.

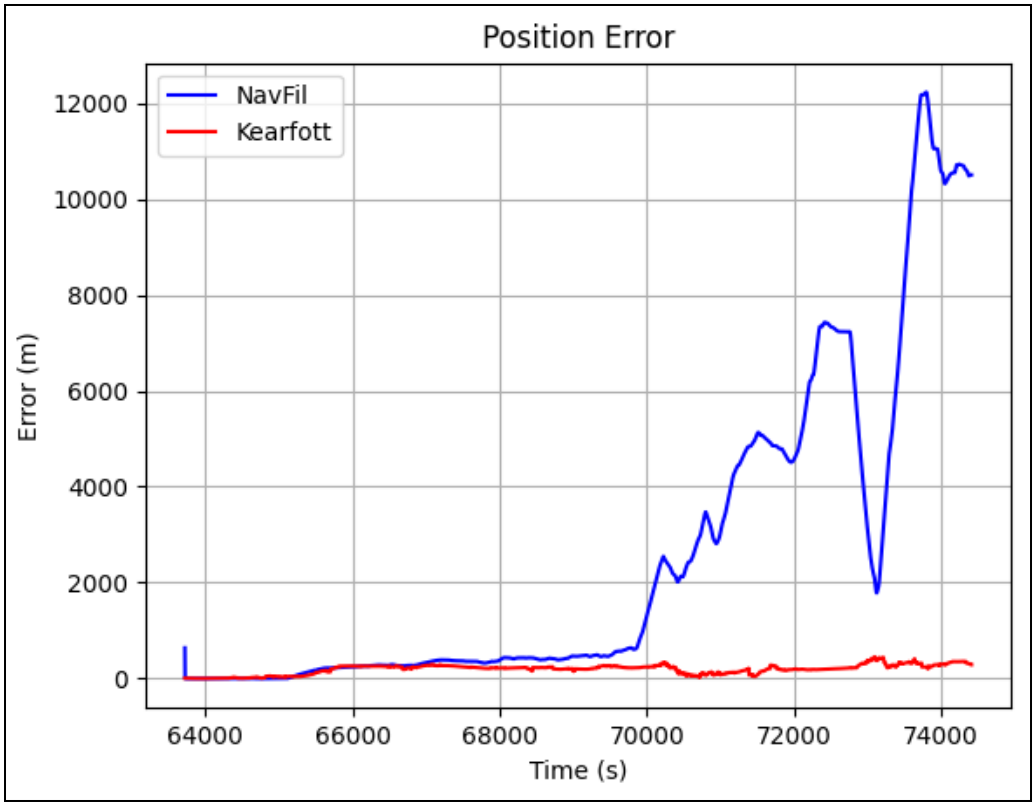

Figure 7. Instantaneous position error for Kearfott and NavFil over time.

The position comparison and position error plots were visually inspected to compare the performance of the various RL algorithms. These plots can be viewed in the Appendices, and the numerical results are covered in Section 6.

6. RESULTS

Algorithm performance is measured by both accuracy and time. Each algorithm ran on the same GPU and a fixed memory usage; the time it takes to process an episode determines how resource-intensive each algorithm is, and whether or not it can reliably be run in near-real-time.

Table 3 compares the times each algorithm took to process a single 20 second episode, and the time each took to process one full second of data. Values in **Black** can process in near-real time. Values in **Green** performed the best compared to the remaining algorithms. Values in **Red** took longer to process than the IMU frequency, and thus would not be expected to run in near real time without disregarding some training data.

Table 3. Training Time Comparison.

Algorithm	Time (s) to Process one 20s Episode	Time (s) to Process 1s of data
TD3	36.83	1.842
DDPG	15.43	0.772
PPO Clip	4.90	0.245
PPO Penalty	4.73	0.237
TRPO	3.70	0.185
DQN	2.95	0.148

For these tests, the system loads in prerecorded data practically instantaneously. When implemented in a real vehicle setting, each data point will be provided at the IMU/NavFil frequency (50 Hz). For Table 3, if an algorithm takes less than 20s to process a 20s episode, it is expected it to work in real time with no issues. If an algorithm takes longer than 20s to complete, that means that the processing and learning takes longer than the time between incoming data samples. For instance, an algorithm that takes 30s to process a 20s episode means that 10s (30-20) of data will unused for learning when running in real time. This unused data will still be able to be processed by NF_{aided} and keep the system up to date, but would cause the algorithm to have access to less data and would thus learn slower. In short, all algorithms that can process 1s of data in less than a second will be rated equally in terms of how much data can be processed in real-time. Algorithms with shorter durations have the added benefit of consuming less processing power.

Table 4 and Table 5 both compare the positional errors (in meters) for the GPS-denied NavFil, GPS-denied INS, and six different RL algorithms. Values in **Black** performed better than the GPS-Denied NavFil baseline. Values in **Green** performed the best compared to the remaining algorithms. Values in **Red** performed worse than the GPS-Denied NavFil baseline. Each Column aggregates training data from the columns prior (i.e. the first column was trained on Training Run 1 (see Appendices), The second column was trained on both Training Run 1 and 2, etc.). Table 4 shows the results of the algorithms being tested against Test Run 1 (see Appendices). Table 5shows the results of the algorithms being tested against Test Run 2 (see Appendices).

Table 4. Accuracy when training on multiple datasets - tested on Testing Run 1.

Algorithm	Distance (m) from GPS After Number of Trainings			
	Training Run 1	Training Run 2	Training Run 3	Training Run 4
GPS-Denied NavFil	158.50	158.50	158.50	158.50
GPS-Denied INS	76.27	76.27	76.27	76.27
TD3	205.08	219.83	150.49	118.70
DDPG	253.38	135.71	208.91	94.17
PPO Clip	428.28	448.75	370.77	482.31
PPO Penalty	271.84	194.80	90.53	257.27
TRPO	117.31	119.23	110.35	106.28
DQN	130.67	161.64	95.48	160.14

Table 5. Accuracy when training on multiple datasets - tested on Testing Run 2.

Algorithm	Distance (m) from GPS After Number of Trainings			
	Training Run 1	Training Run 2	Training Run 3	Training Run 4
GPS-Denied NavFil	1123.64	1123.64	1123.64	1123.64
GPS-Denied INS	170.62	170.62	170.62	170.62
TD3	1238.34	1247.70	1083.88	1100.57
DDPG	1467.47	907.45	1231.22	981.04
PPO Clip	1383.47	1520.78	1421.68	1151.81
PPO Penalty	649.21	1211.93	1008.27	1307.03
TRPO	1128.87	926.74	945.16	947.81
DQN	1077.71	839.91	1003.37	1112.97

Table 6 compares the positional errors (in meters) for the GPS-denied NavFil, GPS-denied INS, and six different RL algorithms. Values in **Black** performed better than the GPS-Denied NavFil baseline. Values in **Green** performed the best compared to the remaining algorithms. Values in **Red** performed worse than the GPS-Denied NavFil baseline. Values in **Blue** performed better than INS. Each algorithm was trained on Training Run 4 (Appendices) and tested against GPS-denied data that was collected while following a similar, but not identical path. Both datasets were collected on the same day. The difference is that the test run did not allow GPS aiding. Each data column in Table 6

shows the results of the algorithms being trained an additional time on the same data set, and then re-tested.

Table 6. Accuracy when training on the same dataset multiple times.

Algorithm	Distance (m) from GPS After Number of Trainings		
	One Training	Two Trainings	Three Trainings
GPS-Denied NavFil	253.26	253.26	253.26
GPS-Denied INS	122.22	122.22	122.22
TD3	215.19	244.75	186.88
DDPG	443.08	516.22	515.07
PPO Clip	225.65	200.94	464.80
PPO Penalty	192.07	749.45	235.72
TRPO	146.38	129.92	121.12
DQN	261.12	252.50	157.42

The overall accuracy performance of an algorithm can be measured by answering the following questions:

a) How quickly is the algorithm able to learn the environment?

b) When training on multiple datasets, do the results on the test set generally the same or improve?

c) When training on the same dataset multiple times, do the results on the test set consistently improve or does overfitting occur?

Regarding b: Adding more training data with high variation should increase generalizability. Consider Table 4: If one algorithm (tested on Test Set 1) performs slightly worse from 1 training to 2 trainings, it may be because the second set is less similar to Test Set 1. Now consider Table 5: If the algorithm (tested on Test Set 2) performed significantly better on two trainings than one training, it may be because the second set is more similar to Test Set 2. Even though the algorithm did slightly worse on Test Set 1, both of these conditions show that the algorithm showed the ability to generalize, which is essential for getting consistently good predictions.

Table 4 and Table 5 help answer a and b. For this test, the algorithms were trained each algorithm on one dataset at a time, and tested on two test datasets in between each training session. On both test sets, DDPG improved after two training sessions, worsened after three, and improved again after four. This indicates that the algorithm struggles to consistently improve (a, not b). TD3 was able to slowly improve (or at least not worsen by much) as it trained on more data (b, not a). PPO Clip gave consistently poor results (neither a or b), but PPO Penalty gave mixed results (c, not b). TRPO was the most consistent and was generally improved or remained the same as it received more data (a and b).

Table 6 can help answer a and c. For this test, the algorithms were trained and tested on data collected on a similar path. The algorithms were trained multiple times using the same

training dataset. Since the algorithms consistently see similar states, it was assumed that they would consistently improve with each additional training session. PPO Clip performed poorly on the first run, but consistently improved overtime. TD3 worsened slightly on the second train, but improved on the last run (which doesn't necessarily rule out consistency). DDPG was the only algorithm with consistently poor results. TRPO was the only algorithm with consistently good and improving results, so much so it performed on par with the nav-grade INS after the last training session. This final TRPO result was very promising, however, after additional fourth and fifth training sessions were completed the results were essentially the same or slightly worse than after the third training session, indicating the possibility that the algorithm has been trained too much on very similar data and now has difficulty with new/different data. This is referred to as overfitting.

7. CONCLUSION

The DIRT-I system was used to test the ability of multiple Reinforcement Learning algorithms to correct the drift in inertial measurements without aiding from GPS. This resulted in an improvement of the position estimate from the Kalman Filter (NavFil), without the benefit of external aiding. Overall the TRPO algorithm outperformed its competitors, and will likely be the main algorithm used in the DIRT-I system moving forward. In addition to more consistent improvements to the position error, TRPO was also the second fastest algorithm tested (with regards to training). This means it should have no problem running in a real-time environment, and will require less GPU power than other algorithms. TRPO was also shown to improve or maintain performance after multiple training sessions (either on different datasets or repetitions of the same dataset). This means that it is a good candidate for pre-training, which will allow the DIRT-I system to improve position error based on GPS-denied inertial measurements, with less training time in the field. After an hour without GPS aiding, TRPO allowed the DIRT-I system to achieve position errors on par with a navigation-grade INS. This is even more impressive since the INS was being aided by a speed sensor that DIRT-I was not taking advantage of.

In addition to configuring DIRT-I to run in real-time, other future efforts will be to include the ability of additional aiding sensors such as a speed sensor and magnetic compass. DIRT-I will also be tested on different inertial systems with varying degrees of performance. Since TRPO was the top performer, future efforts will focus on this algorithm. This will allow for a better understanding and more precise tuning of TRPO's parameters, which should result in a performance improvement.

DQN also performed fairly well, and showed promising results when trained on the same dataset multiple times. DQN's ability to improve after re-training on the same dataset may be a result of the way the algorithm works. If this is the case, the algorithm should continue to improve or perform at the same level when trained on the same dataset more than three times. TRPO also showed improvement after re-training on the same dataset three times, but (as mentioned earlier), its performance declined on the fourth re-training. At the time of this report, it is not known if DQN will react the same way with four or more re-training sessions. Due to the way DQN functions, there is a strong possibility that it will continue to improve, so this will be investigated as part of the follow-on effort.

This first-year effort of the DIRT-I project successfully proved that RL algorithms can be used to improve the position solution of a Kalman Filter based on inertial measurements in a GPS-denied environment. It also evaluated several types of RL algorithms for their ability to correct inertial measurements with the goal of an improved GPS-denied position solution.

This page is intentionally blank.

REFERENCES

1. VectorNav, "What is an inertial measurement unit IMU," 3 July 2021. [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu.

2. P. S. Maybeck, "Chapter 1, Introduction," in *Stochastic Models, Estimation, and Control Volume 1*, Wright-Patterson Air Force Base, Ohio, Academic Press, 1979, pp. 1-16.

3. S. Boyd, *Lecture 9 The Extended Kalman Filter,* Stanford: Stanford University, 2008.

4. Mathworks, "Extended Kalman Filter," Mathworks, [Online]. Available: https://www.mathworks.com/help/control/ref/ekf_block.html. [Accessed 1 10 2021].

5. G. A. Einicke and . L. B. White, "Robust extended Kalman filtering," *IEEE Transactions on Signal Processing,* vol. 47, no. 9, pp. 2596-2599, 1999.

6. Kearfott Corporation, *KI-4901S High Performance SEANAV INS/IMU Kit,* Kearfott Corporation, 2012.

7. Navtech GPS, "Hemisphere Vector V102 Datasheet," 6 2019. [Online]. Available: https://2w6vmg3m8cv83pn80b2dfi9f-wpengine.netdna-ssl.com/wp-content/uploads/V102_DS.pdf. [Accessed 1 10 2021].

8. Z. Ding, T. Yu, Y. Huang, H. Zhang, G. Li, Q. Guo, L. Ma and H. Dong, "RLzoo: A Comprehensive and Adaptive Reinforcement Learning Library," 19 8 2021. [Online]. Available: https://arxiv.org/pdf/2009.08644.pdf. [Accessed 1 10 2021].

9. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym," 5 6 2016. [Online]. Available: https://arxiv.org/abs/1606.01540. [Accessed 1 10 2021].

10. M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research,* vol. 47, pp. 253-279, 2013.

11. H. Dong, Z. Ding, S. Zhang, H. Yuan, H. Zhang, J. Zhang, Y. Huang, T. Yu, H. Zhang and R. Huang, "Deep Reinforcement Learning: Fundamentals, Research, and Applications," *arXiv preprint arXiv:2009.08644,* 2020.

12. V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Slgorithms," in *Advances in Neural Information Processing Systems (NIPS)*, Denver, CO, 1999.

13. I. Grondman, L. Busoniu, G. Lopes and R. Babuska, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 42, pp. 1291-1307, 2012.

14. T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Cornell University, Ithaca, 2018.

15. T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel and S. Levine, "Soft Actor-Critic Algorithms and Applications," Cornell University, Ithaca, 2019.

16. V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *33rd International Conference on Machine Learning*, New York, 2016.

17. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "CONTINUOUS CONTROL WITH DEEP REINFORCEMENT," in *International Conference on Learning Representations*, San Juan, 2016.

19. S. Fujimoto, H. van Hoof and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *35th International Conference on Machine Learning*, Stockholm, 2018.

20. J. Schulman, S. Levine, P. Moritz, M. Jordan and P. Abbeel, "Trust Region Policy Optimization," in *31st International Conference on Machine Learning*, Lille, 2015.

21. J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," 28 8 2017. [Online]. Available: https://arxiv.org/abs/1707.06347. [Accessed 1 10 2021].

22. N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. Ali Eslami, M. Riedmiller and D. Silver, "Emergence of Locomotion Behaviours in Rich Environments," Cornell University, 10 7 2017. [Online]. Available: https://arxiv.org/abs/1707.02286. [Accessed 1 10 2021].

23. Z. Zhang, X. Luo, T. Liu, S. Xie, J. Wang, W. Wang, Y. Li and Y. Peng, "Proximal Policy Optimization with Mixed Distributed Training," 30 9 2019. [Online]. Available: https://arxiv.org/pdf/1907.06479.pdf. [Accessed 1 10 2021].

24. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 26 2 2015.

APPENDIX A
TRAINING RUN MAPS

Figures A-1 through A-4 shows maps of the paths used for training the RL algorithms discussed in this document. The positions of these paths are the positions of the GPS-aided NavFil, which is identical to the recorded GPS positions. Data was collected while driving along these paths, with GPS enabled as it would be while the DIRT-I system is in its training mode. All of these paths start and end in roughly the same position.

Figure A-1. Training Run 1 (February 4, 2021).

Figure A-2. Training Run 2 (February 18, 2021).

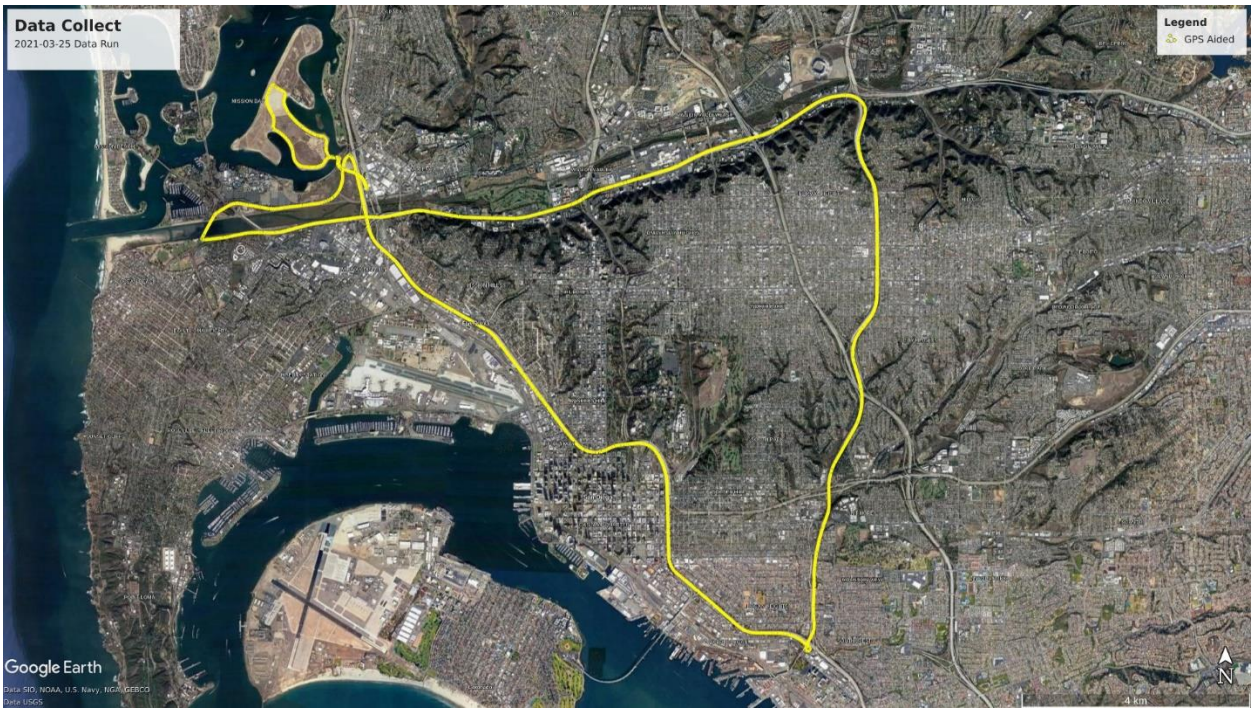

Figure A-3. Training Run 3 (March 25, 2021).

Figure A-4. Training Run 4 (April 28, 2021).

TESTING RUN 1 MAPS (JULY 17, 2021)

Figures A-5 through A-9 show maps of the path used for Testing Run 1, and the results of the associated RL algorithms. For these plots, the data was collected on July 17, 2021 with the GPS disabled (but still recorded for analysis) to simulate a GPS-denied situation. After training, the various algorithms were tested against this dataset. The GPS-aided (yellow), GPS-denied (red) and INS (green) are identical in all of these figures, and are only included for the sake of comparison. The RL algorithm specified for each figure is shown in blue. These figures show the results of the RL algorithms after being trained with all four training run datasets.

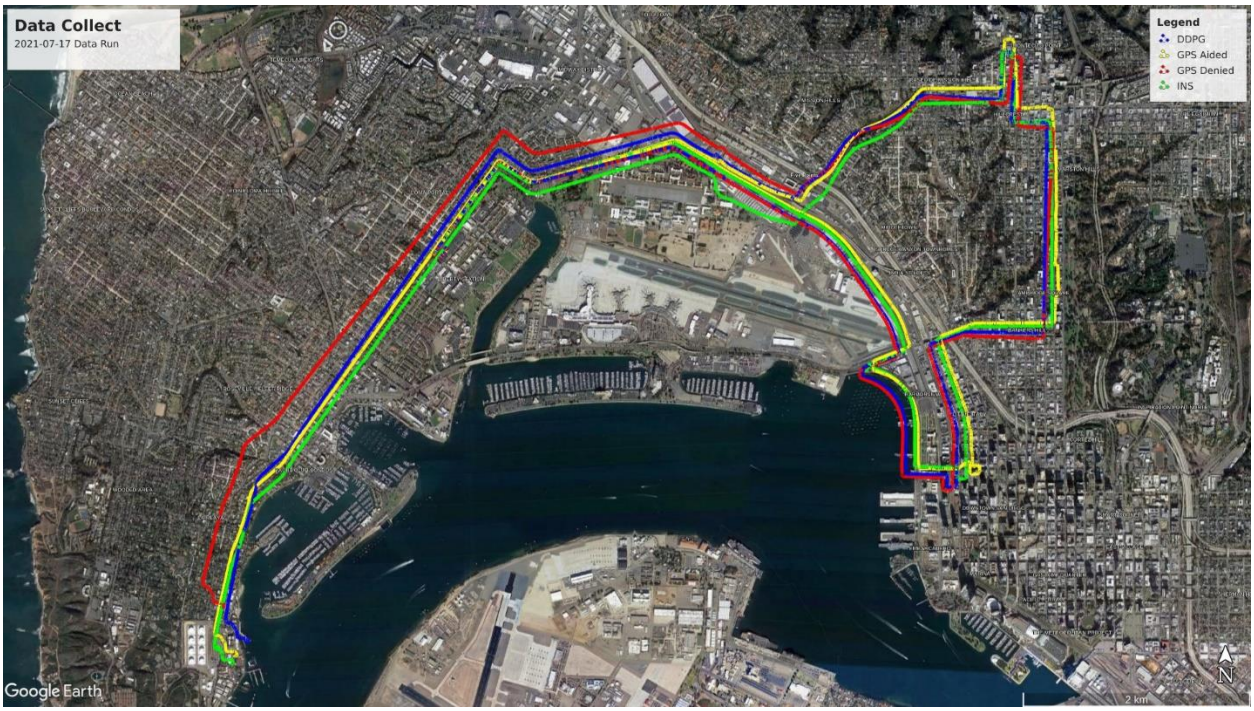

Figure A-5. Testing Run 1 (July 17, 2021) – DDPG results.

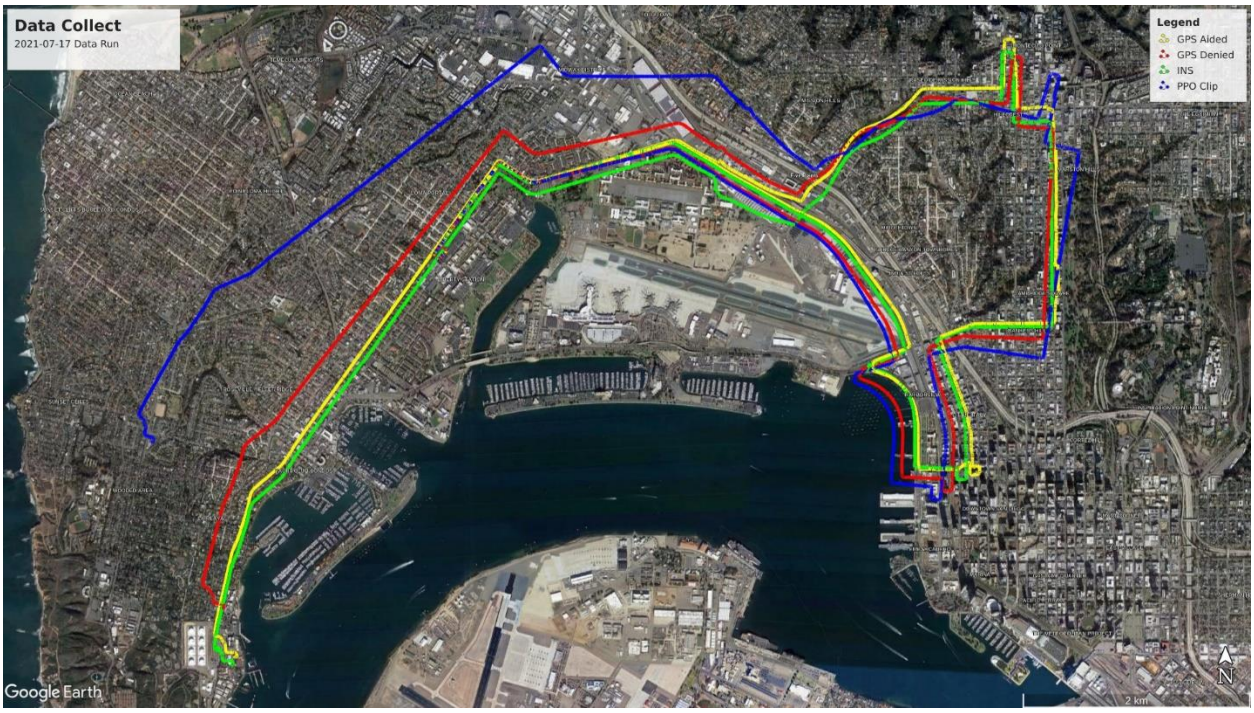

Figure A-6. Testing Run 1 (July 17, 2021) – Pro Clip results.

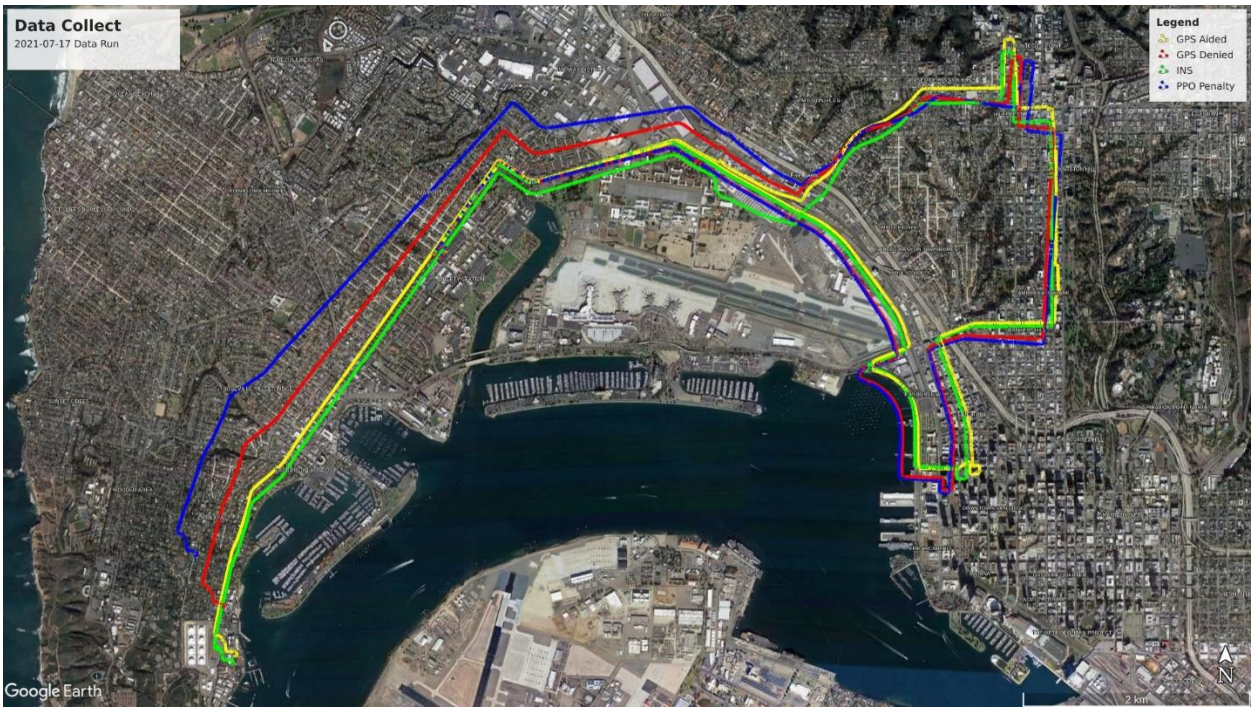

Figure A-7. Testing Run 1 (July 17, 2021) – Pro Penalty results.

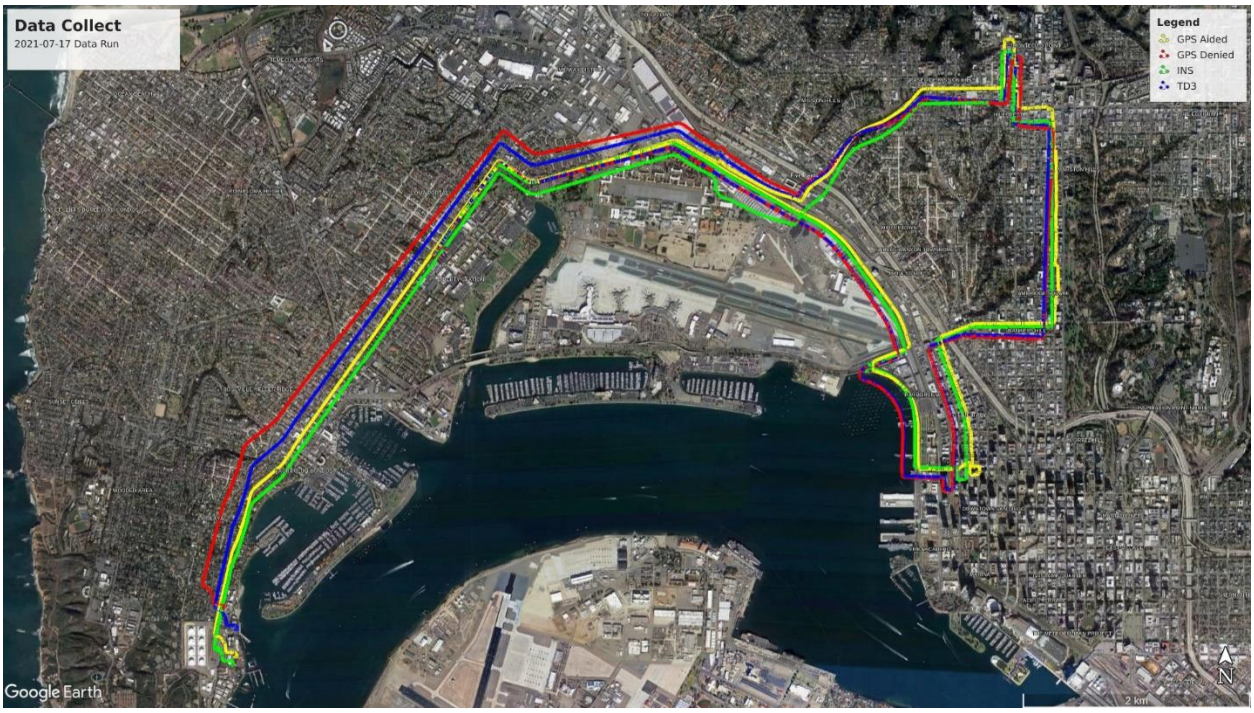

Figure A-8. Testing Run 1 (July 17, 2021) – TD3 results.

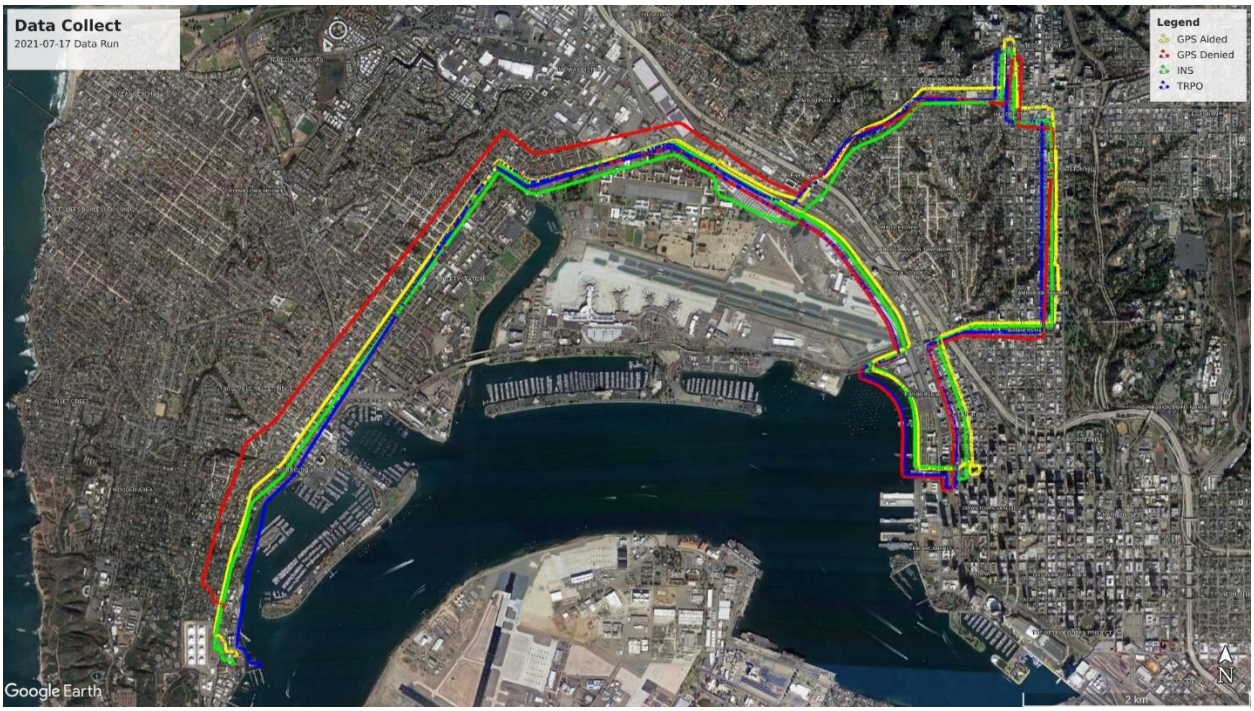

Figure A-9. Testing Run 1 (July 17, 2021) – TRPO results.

TESTING RUN 2 MAPS (AUGUST 3, 2021)

Figures A-10 through A-14 show maps of the path used for Testing Run 2, and the results of the associated RL algorithms. For these plots, the data was collected on August 3, 2021 with the GPS disabled (but still recorded for analysis) to simulate a GPS-denied situation. After training, the various algorithms were tested against this dataset. The GPS-aided (yellow), GPS-denied (red) and INS (green) are identical in all of these figures, and are only included for the sake of comparison. The RL algorithm specified for each figure is shown in blue. These figures show the results of the RL algorithms after being trained with all four training run datasets. Unlike the training runs and Testing Run 1, Testing Run 2 did not end where it started. Instead, it ended after the large loop on the right side of these figures. The paths on the left side of these figures (at the beginning of the run) were very well matched, so they overlap each other. This makes the lines look broken, but they are not. This run was also spread over a larger distance, so the map is zoomed out further than the maps for Training Run 1. This also makes the overlapping path lines harder to see near the beginning of the run.

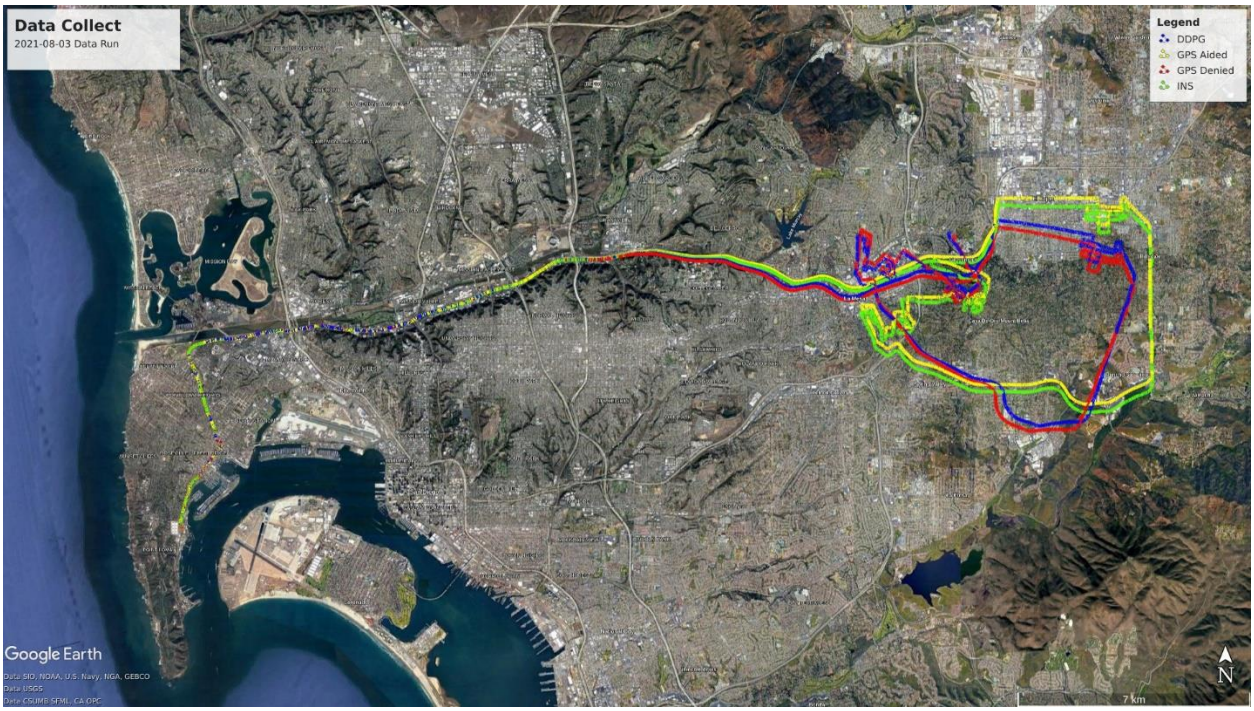

Figure A-10. Testing Run 2 (August 3, 2021) – DDPG results.

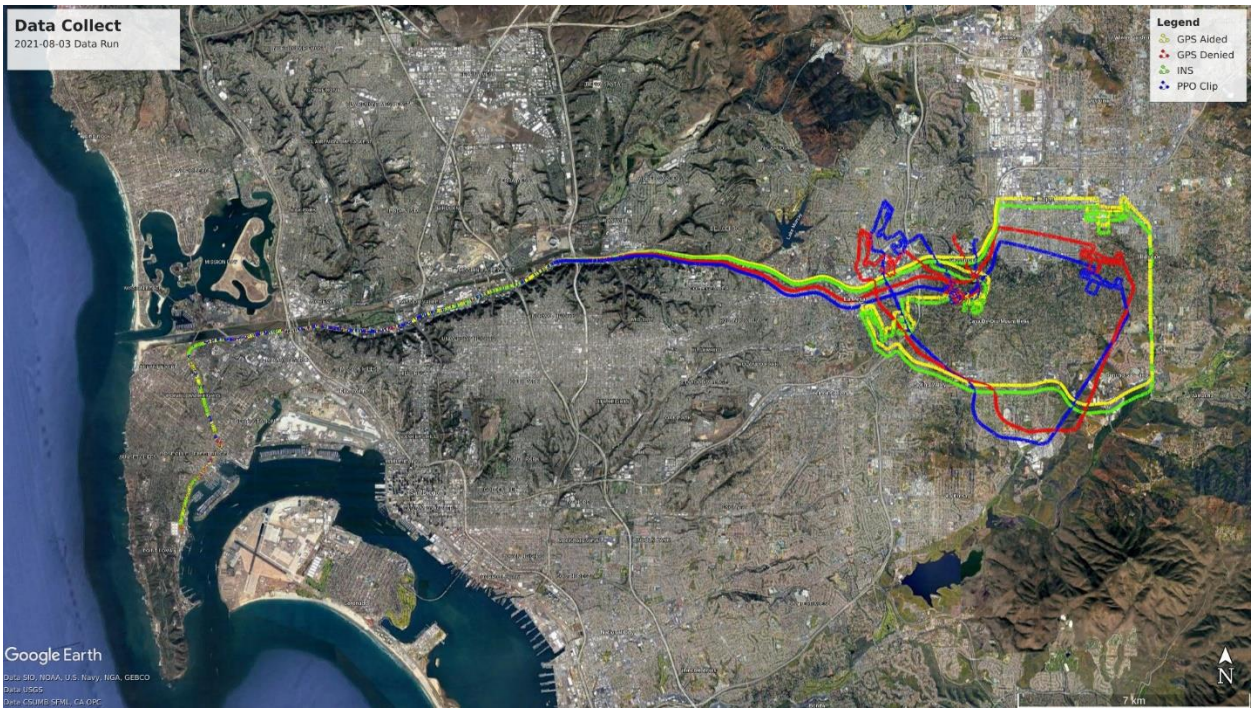

Figure A-11. Testing Run 2 (August 3, 2021) – Pro Clip results.

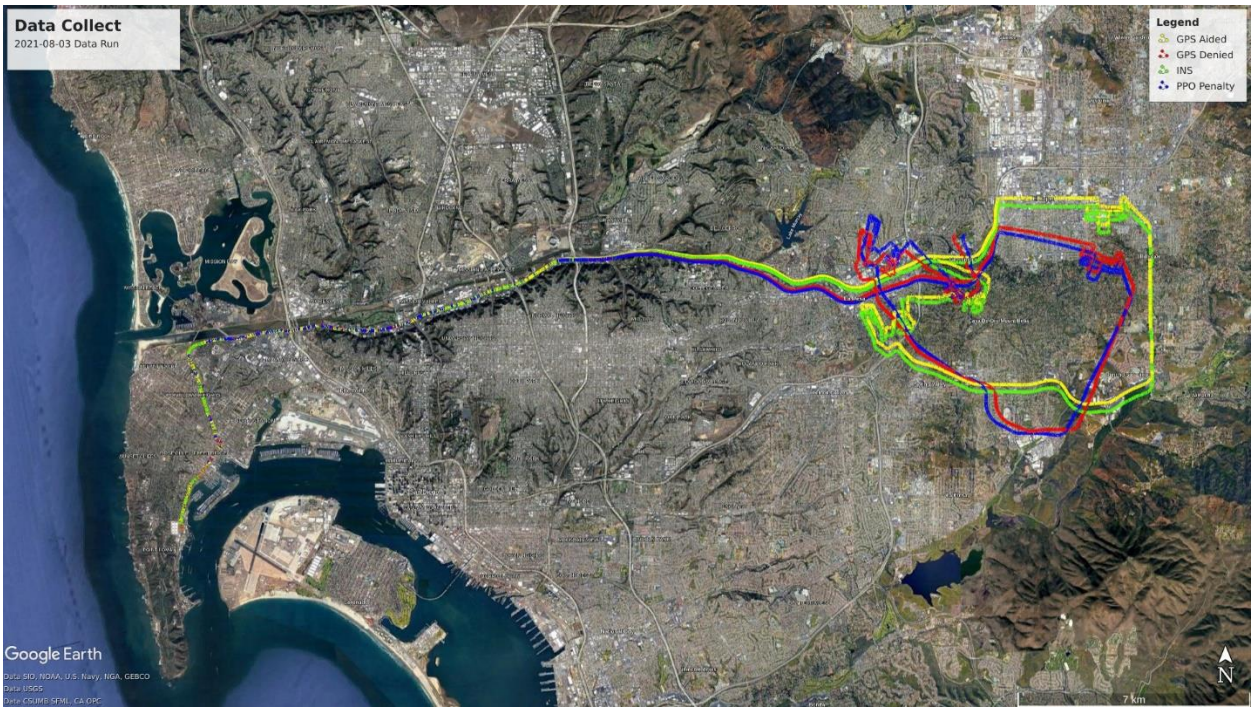

Figure A-12. Testing Run 2 (August 3, 2021) – Pro Penalty results.

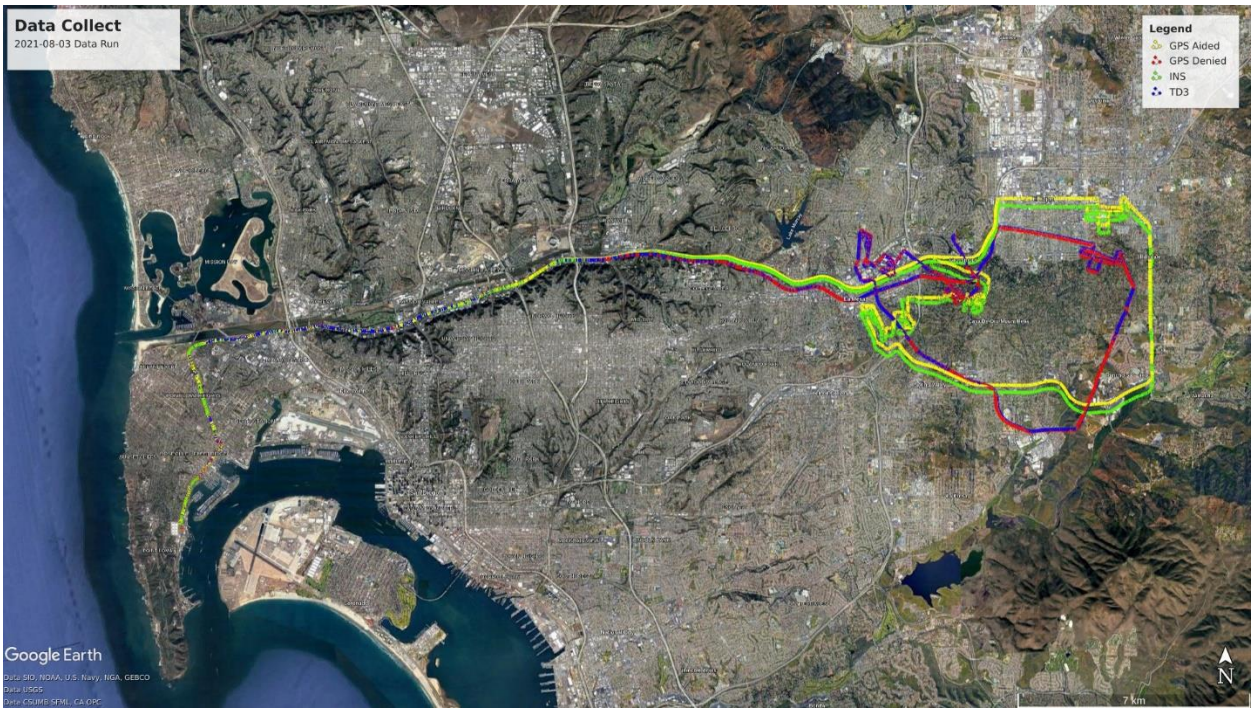

Figure A-13. Testing Run 2 (August 3, 2021) – TD3 results.

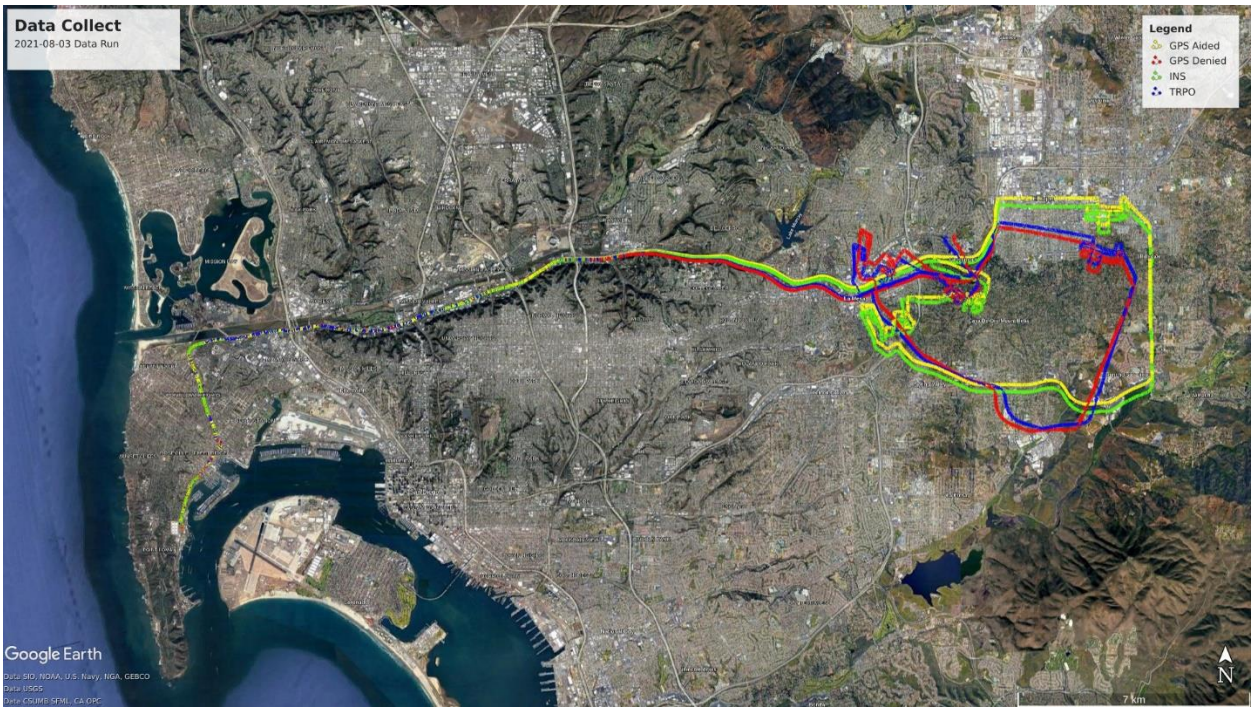

Figure A-14. Testing Run 2 (August 3, 2021) – TRPO results.

AVERAGE CUMULATIVE ERRORS FROM TESTING RUN 1

Figures A-15 through A-18 show the comparison of average cumulative errors for NavFil (red), INS (Green) and each of the RL algorithms being tested (colors listed in plot legends). These figures show the error comparison from Testing Run 1 after all algorithms had been trained on the training runs specified in the figure captions. The GPS-denied NavFil (red) should represent the worst error because it has no aiding, and is simply the position estimate based on un-corrected inertial measurements that are drifting over time. The GPS-denied INS (green) represents the ideal performance because it is a navigation-grade INS that was being aided by velocity measurements. The corresponding position comparison maps for the results shown in Figures A-15 through A-18, can be seen in Figures A-5 through A-9.

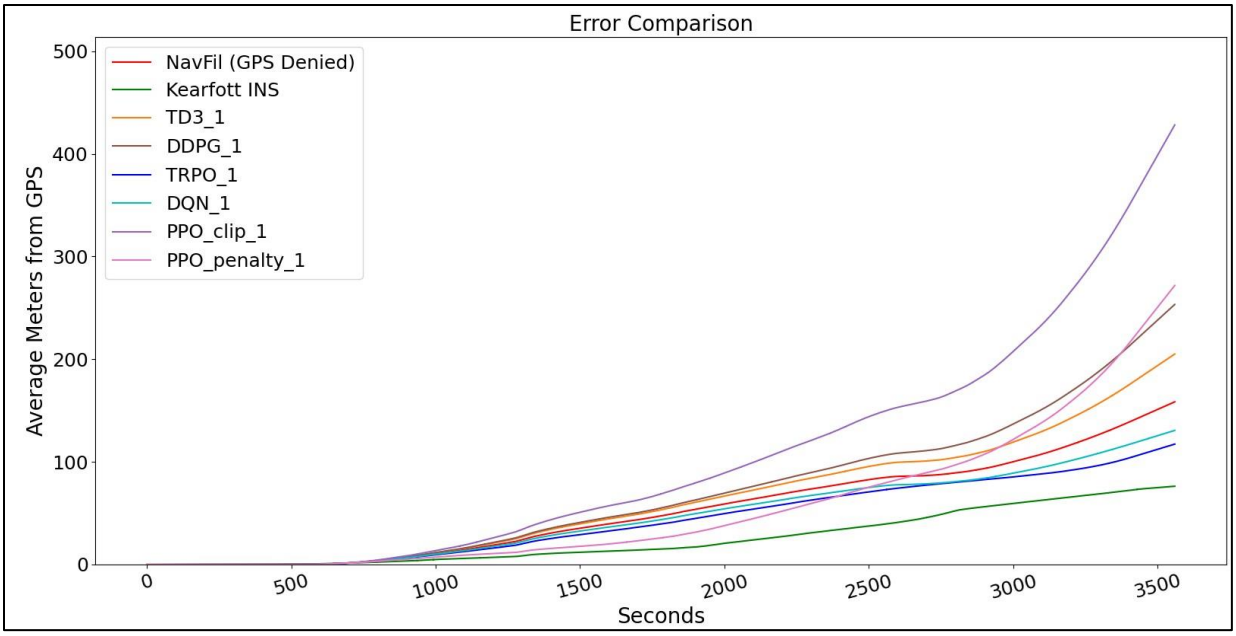

Figure A-15. Average cumulative error (m) – Training Run 1 - Testing Run 1.

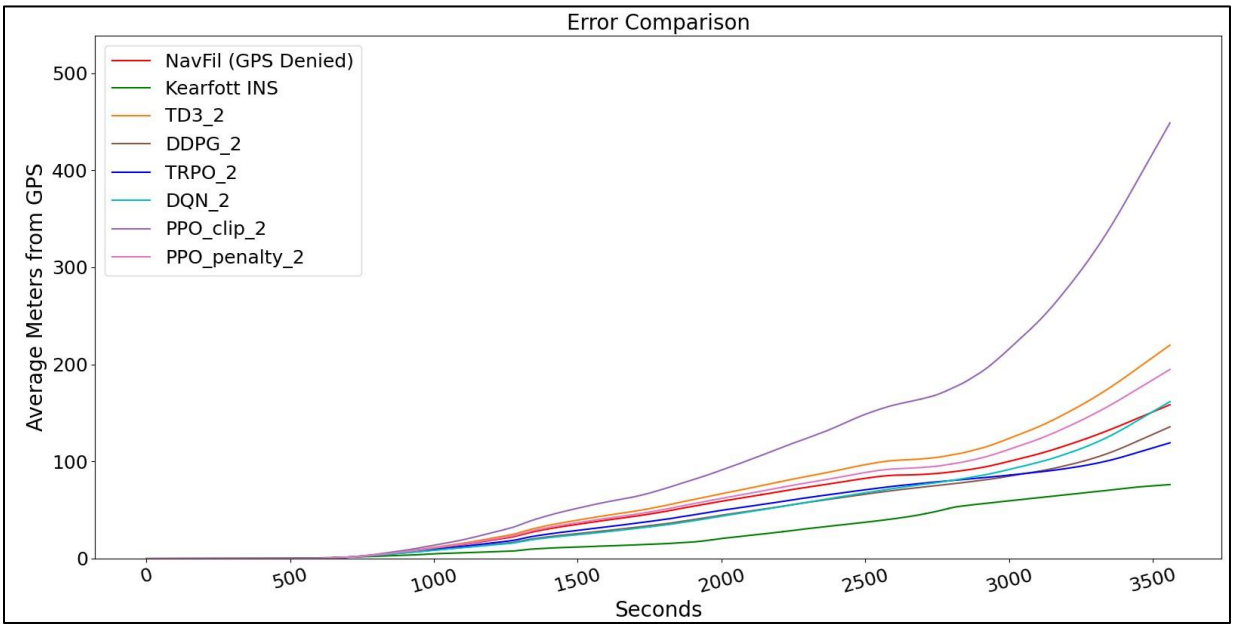

Figure A-16. Average cumulative error (m) – Training Runs 1 and 2 - Testing Run 1.

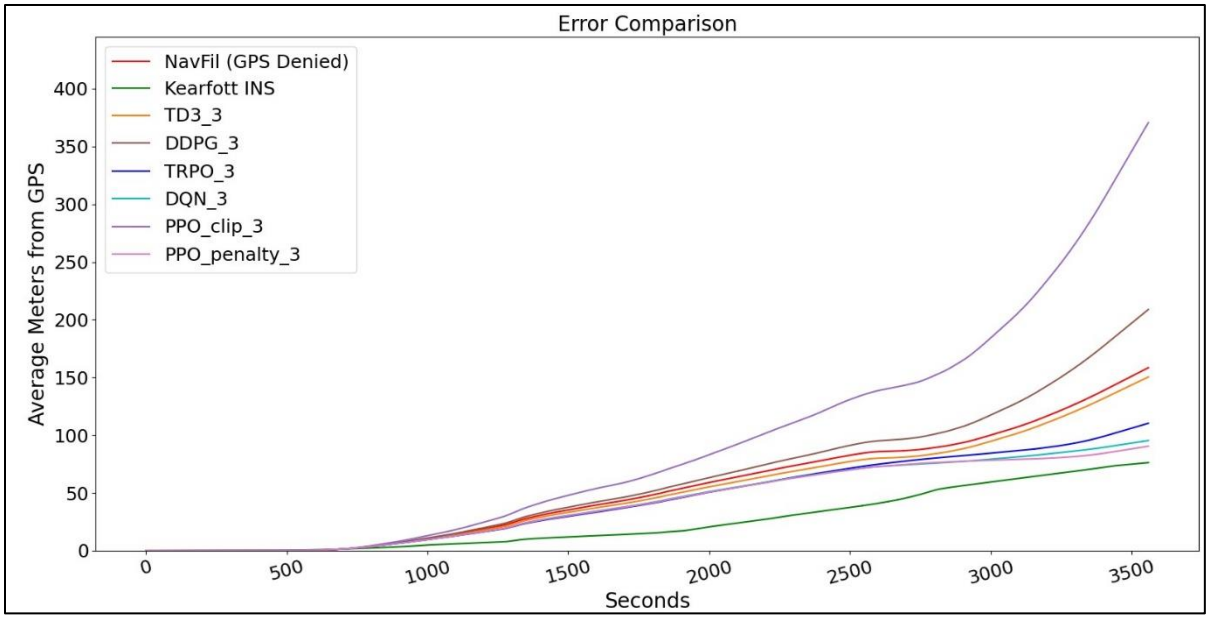

Figure A-17. Average cumulative error (m) – Training Runs 1, 2 and 3 - Testing Run 1.

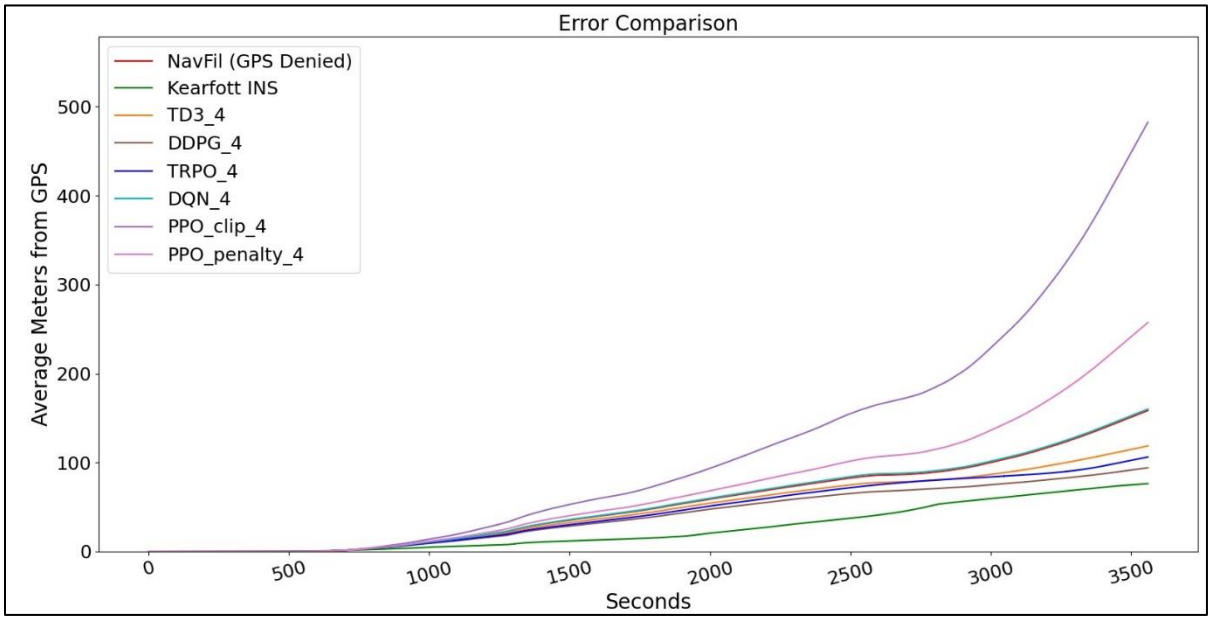

Figure A-18. Average cumulative error (m) – Training Runs 1, 2, 3 and 4 - Testing Run 1.

This page is intentionally blank.

APPENDIX B
AVERAGE CUMULATIVE ERRORS FROM TESTING RUN 2

Figures B-1 through B-4 show the comparison of average cumulative errors for NavFil (red), INS (Green) and each of the RL algorithms being tested (colors listed in plot legends). These figures show the error comparison from Testing Run 2 after all algorithms had been trained on the training runs specified in the figure captions. The GPS-denied NavFil (red) should represent the worst error because it has no aiding, and is simply the position estimate based on un-corrected inertial measurements that are drifting over time. The GPS-denied INS (green) represents the ideal performance because it is a navigation-grade INS that was being aided by velocity measurements. Testing Run 2 is different from other testing runs because it does not have a corresponding training run. Normally, data for a training run is collected first – with GPS enabled. Then data for a corresponding testing run is collected with GPS disabled. For Testing Run 2, there was no previous training data collected with GPS enabled. This was done because Testing Run 2 was meant to be a longer run, and there was not enough time to do both extended training and testing data collections on the same day. Unfortunately, there was an error during the data collection for Testing Run 2 that required the last half of the data to be removed. However, Testing Run 2 is still almost twice as long as Testing Run 1. The corresponding position comparison maps for the results shown in Figures B-1 through B-4, can be seen in Figures A-10 through A-14.

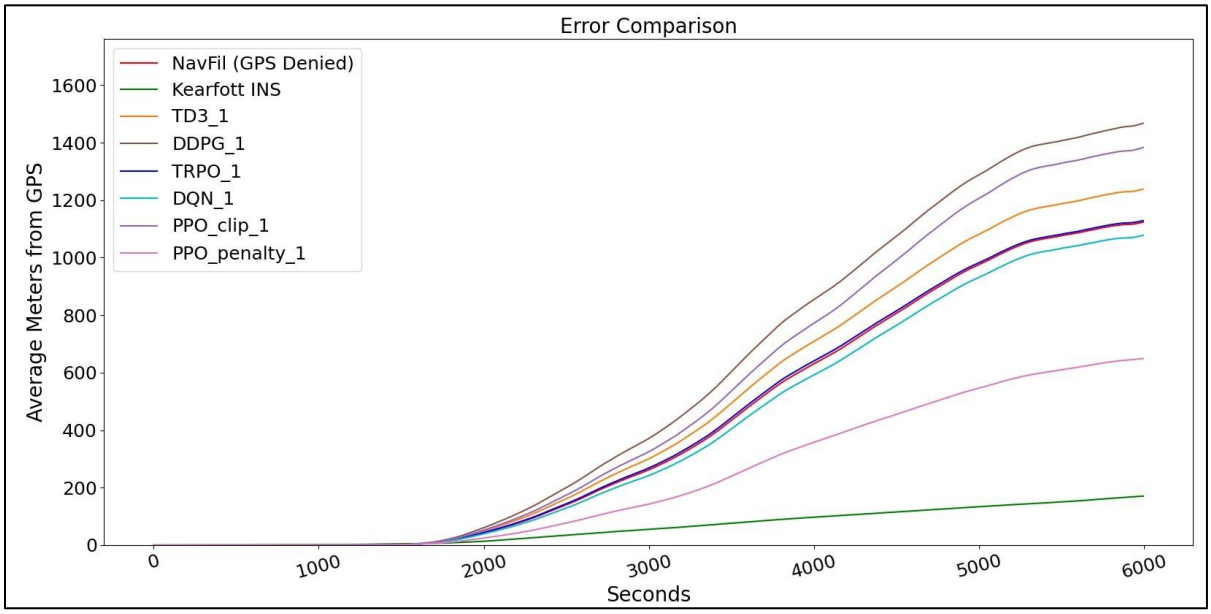

Figure B-1. Average cumulative error (m) – Training Runs 1 - Testing Run 2.

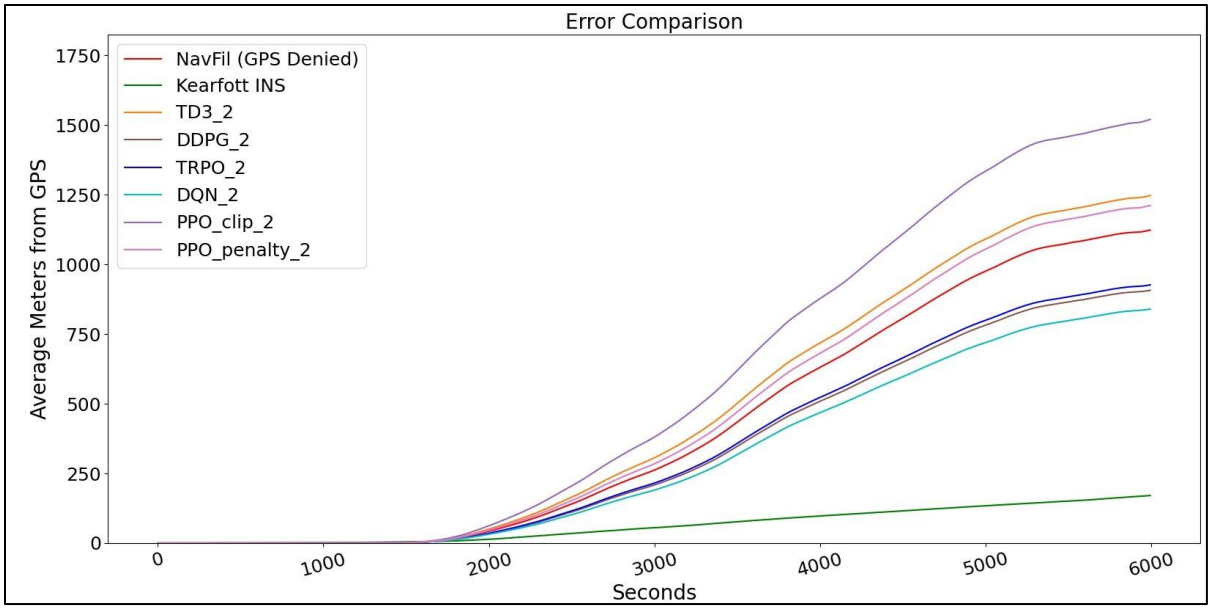

Figure B-2. Average cumulative error (m) – Training Runs 1 and 2 - Testing Run 2.

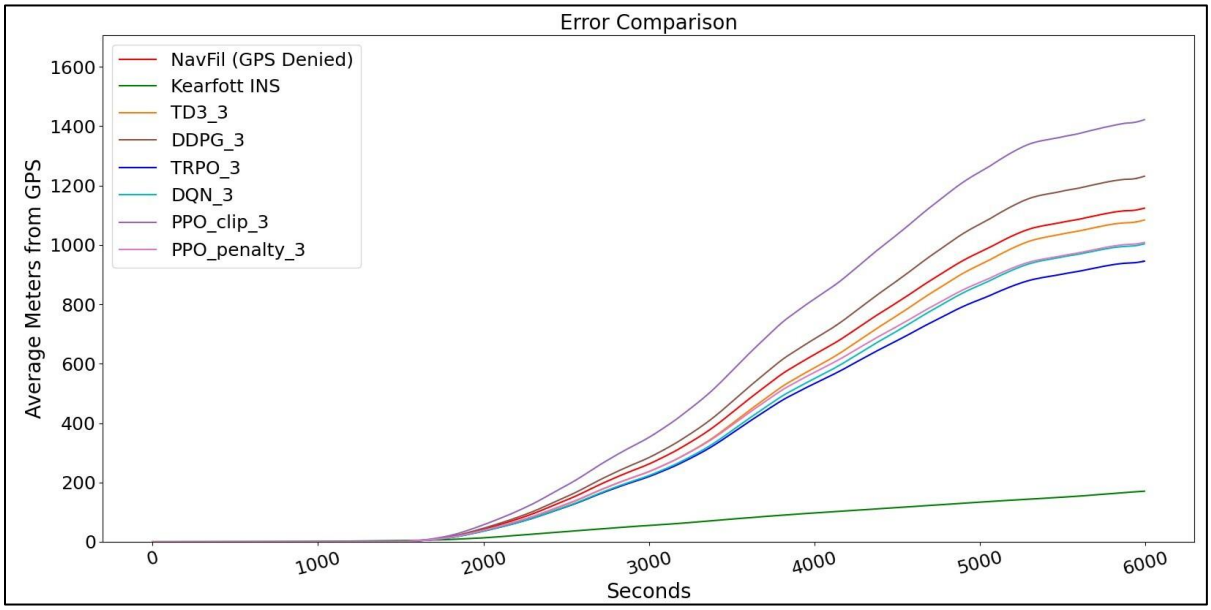

Figure B-3. Average cumulative error (m) – Training Runs 1 and 3 - Testing Run 2.

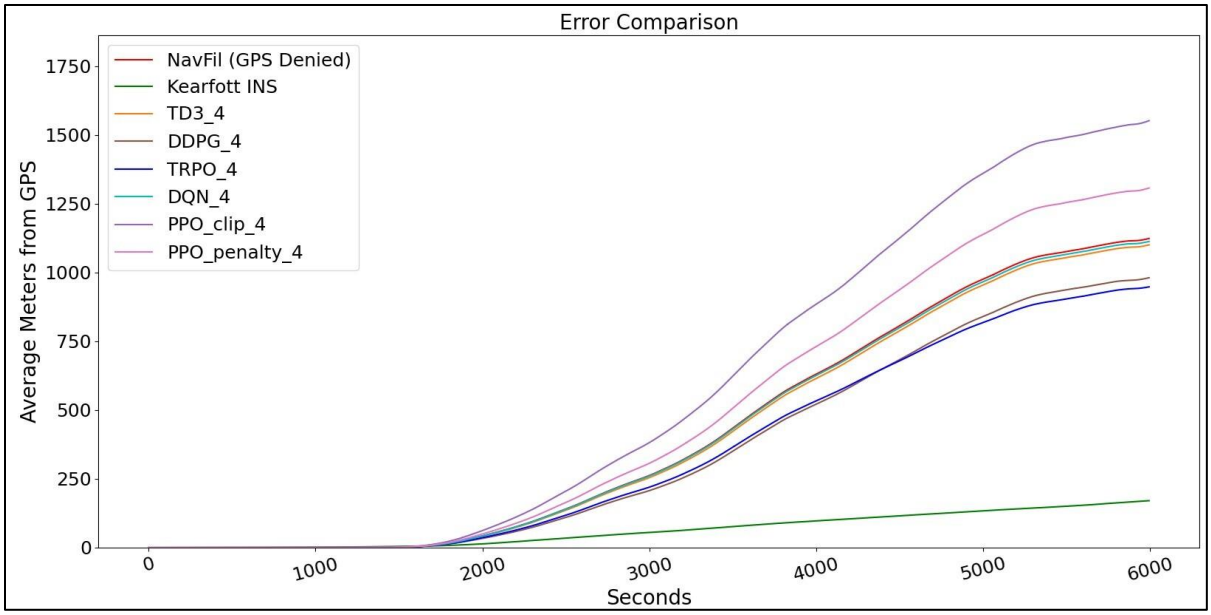

Figure B-4. Average cumulative error (m) – Training Runs 1, 2, 3 and 4 - Testing Run 2.

This page is intentionally blank.

INITIAL DISTRIBUTION

84310	Technical Library/Archives	(1)
71780	E. Bozeman	(1)
71740	M. Alam	(1)
71780	M. Nguyen	(1)
71740	J. Onners	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (1)

Naval Innovative Science and Engineering (1)

This page is intentionally blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
February 2022	Final	

4. TITLE AND SUBTITLE

Drift Improvement with Reinforcement Training – Inertial Sensors – Year 1

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

Eric Bozeman
Mohammad R. Alam
Minhdao Nguyen
Jeffrey C. Onners
NIWC Pacific

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NIWC Pacific
53560 Hull Street
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-3267

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Innovative Science and Engineering
53560 Hull Street
San Diego, CA 92152-5001

10. SPONSOR/MONITOR'S ACRONYM(S)

NISE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report details the system, test environment, and results used to evaluate Reinforcement Learning (RL) algorithms for their ability to reduce the rate of drift in the positional error of an Inertial Navigation System (INS) without aiding from external sensors. A custom RL environment was created to train a RL algorithm to correct the raw inertial measurements from an INS in such a way that the position more closely matched the INS position after being corrected by a Global Navigation Satellite System (GNSS). When GNSS aiding was removed, the RL system would continue to correct the inertial measurements as it was trained to do before GNSS aiding was removed. Multiple RL algorithms were used in the RL system, and their performance was evaluated on their ability to correct inertial measurements to allow for more accurate position solutions (reduce positional error). The algorithms were also evaluated on their use of computer resources and ability to operate in real time. The data collections and evaluations described in this report show that a RL system can help reduce the positional error of an INS without aiding from external sensors (such as GNSS). It also shows that some RL algorithms lend themselves better to this type of system than others. In the end, this research identified two RL algorithms that will continue to be used in further testing related to this work.

15. SUBJECT TERMS

DIRT-1; RL system; INS; GNSS; GPS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Eric Bozeman
U	U	U	SAR	64	19B. TELEPHONE NUMBER *(Include area code)* (619) 553-1044

This page is intentionally blank.

This page is intentionally blank.

Naval Information Warfare Center Pacific (NIWC Pacific)
San Diego, CA 92152-5001