2022

Modified Structured Domain Randomization in a Synthetic Environment for Learning Algorithms

Bryan L. Croft

Dissertation Report
Modified Structured Domain Randomization in a Synthetic Environment for Learning Algorithms

by

Bryan L. Croft

CISD 901 Doctoral Dissertation
Dissertation report submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Computing and Engineering
Nova Southeastern University

April 2022

We hereby certify that this dissertation, submitted by Bryan L. Croft conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Michael J. Laszlo, Ph.D.
Chairperson of Dissertation Committee

5/25/22
Date

Francisco J. Mitropoulos, Ph.D.
Dissertation Committee Member

5/25/22
Date

Sumitra Mukherjee

Sumitra Mukherjee, Ph.D.
Dissertation Committee Member

5/25/22
Date

Approved:

Meline Kevorkian

Meline Kevorkian, Ed.D.
Dean, College of Computing and Engineering

5/25/22
Date

College of Computing and Engineering
Nova Southeastern University

2022

An Abstract of a Dissertation Report Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Dissertation Report
Modified Structured Domain Randomization in a Synthetic Environment for Learning Algorithms

by
Bryan L. Croft
2022

Deep Reinforcement Learning (DRL) has the capability to solve many complex tasks in robotics, self-driving cars, smart grids, finance, healthcare, and intelligent autonomous systems. During training, DRL agents interact freely with the environment to arrive at an inference model. Under real-world conditions this training creates difficulties of safety, cost, and time considerations. Training in synthetic environments helps overcome these difficulties, however, this only approximates real-world conditions resulting in a ‘reality gap’. The synthetic training of agents has proven advantageous but requires methods to bridge this reality gap. This work addressed this through a methodology which supports agent learning. A framework which incorporates a modifiable synthetic environment integrated with an unmodified DRL algorithm was used to train, test, and evaluate agents while using a modified Structured Domain Randomization (SDR+) technique. It was hypothesized that the application of environment domain randomizations (DR) during the learning process would allow the agent to learn variability and adapt accordingly. Experiments using the SDR+ technique included naturalistic and physical-based DR while applying the concept of context-aware elements (CAE) to guide and speed up agent training. Drone racing served as the use case. The experimental framework workflow generated the following results. First, a baseline was established by training and validating an agent in a generic synthetic environment void of DR and CAE. The agent was then tested in environments with DR which showed degradation of performance. This validated the reality gap phenomenon under synthetic conditions and established a metric for comparison. Second, an SDR+ agent was successfully trained and validated under various applications of DR and CAE. Ablation studies determine most DR and CAE effects applied had equivalent effects on agent performance. Under comparison, the SDR+ agent’s performance exceeded that of the baseline agent in every test where single or combined DR effects were applied. These tests indicated that the SDR+ agent’s performance did improve in environments with applied DR of the same order as received during training. The last result came from testing the SDR+ agent’s inference model in a completely new synthetic environment with more extreme and additional DR effects applied. The SDR+ agent’s performance was degraded to a point where it was inconclusive if generalization occurred in the form of learning to adapt to variations. If the agent’s navigational capabilities, control/feedback from the DRL algorithm, and the use of visual sensing were improved, it is assumed that future work could exhibit indications of generalization using the SDR+ technique.

Acknowledgments

My appreciation is given to my dissertation chair, Dr. Michael J. Laszlo, for his time, insight and guidance received. His input was highly valued as it helped to not only define the research but kept me on track and within scope. His kind nature, patience, experience, knowledge, and skills were very much appreciated during this process. In similar fashion this applies to the other members of the dissertation committee, Dr. Sumitra Mukherjee and Dr. Francisco Mitropoulos. Each member of the dissertation committee provided insightful instruction and guidance in the core classes taken at Nova Southeastern University (NSU) as well as feedback to this dissertation.

I am very thankful to NSU for their approach to distance learning especially at the Ph.D. level. Obtaining a Ph.D. degree has been a goal for many years. This goal was impeded for reasons of non-flexible programs with or without a distance learning component. Discovering NSU's program in Computer Science was not only a big win for me but aligned with the timing in my life and career. I appreciate the information and knowledge gained in the core curriculum as well as the insightful instruction and guidance provided by all the instructors for each of the classes. It made for a delightful but challenging educational experience.

I am also appreciative of the faculty and staff of the College of Computing and Engineering for their support accomplishing this goal. Special thanks to Laura Macias as my Graduate Academic Advisor. She was always helpful and friendly through various processes of registration, degree requirements, and so forth while, solving many related issues over the years. My appreciation also goes to the Dean, Dr. Meline Kevorkian and Department Chair Dr. Gregory E. Simco for their time and feedback regarding my queries regarding NSU and the college's policies and supporting this distance learning Computer Science Ph.D. program.

Finally, my sincere and deep appreciation goes to my family. They were cheerleading my way along the path while being deprived of my time and attention while working on this degree. They demonstrated their love, support, and understanding of my desire to achieve this goal.

Table of Contents

Chapters

Appendices

List of Tables

List of Figures

Chapter 1

Introduction

Background

The concept of generalization in learning is based on the premise that past learning can be applied to future situations. The premise is that previous experience, events, and actions provide a guide into future experiences, and this is achieved by finding similarities between past and current events and actions. The assumption is that there exists a transfer of knowledge that can apply across many domains and environmental conditions. Humans learn to abstract from these previous events from which they formulate principles, characteristics, and patterns. These are applied to future situations where the stimulus triggers a sense of familiarity with the current observed state and then recalls related situations from those of past states. This in turn allows the person to take actions with respect to current conditions based on experiences observed and the results of their reaction as seen in the consequences of those reactions. Such conditions allow people to better navigate their world by avoiding the negative and seeking positive rewards. As humans learn to generalize certain conditions, they build in mechanisms which allow them to react or understand more appropriately for that current event or even anticipated events even though the circumstances may differ.

Generalized learning, if not extensive in coverage, can lead to wrong generalizations where experiences possessing one or more similarities may not necessarily lead to a proper correlation with the calculated reaction used in evaluation and reaction to the current event. A more advanced form of learning called discrimination learning extends this by combining classical and operant conditioning. Classical conditioning involves learning associations where two conditions are repeatedly paired together. Extension beyond a single pair for comparisons in general can become complex very quickly. Operant conditioning utilizes reinforcement for reactions applied to positive stimulus and punishment for reaction applied to negative stimulus. Combining these two provide a synergistic learning paradigm.

Within the world of Artificial Intelligence (AI) these learning techniques have been examined, mimicked, and formulated into versions of these learning principles. One of the primary goals for AI applications is to generalize inferred solutions to real world use cases while striving to achieve an improved proximation of human learning. Reinforcement Learning (RL) (Sutton & Barto, 2018; Kaelbling, Littman & Moore, 1996) is one field in AI which closely resembles both generalized and discriminating learning principles. RL differs from machine learning in that it does not require the use of annotated data as is the case for supervised learning processes. RL allows an agent to freely act in an environment whereby the agent is rewarded for positive actions and punished for negative actions. This consequential exploration can then be maximized based on a cumulative reward and serve to provide a balanced exploitive capability based upon previous findings. As stated, annotated data is generally not required in this learning process. As an agent acts in an environment and then the state and reward for that action are fed back into process. This is then repeated while the agent traverses the environment. Deep Learning (LeCun, et al., 2018; Donahue, et al., 2016; Krizhevsky, Sutskever & Hinton, 2017), as a machine learning method, based on Neural Networks (NN) and representation learning, when combined with RL formulates Deep Reinforcement Learning (DRL) (Francois-Lavet et al., 2018;

Arldumaran, Deisenroth, Brundage & Bharath, 2017). This combination integrates the use of raw sensor or image signals to be used for the input into the RL. This sensor capability instruments the agent with better sensing capabilities to explore the environment. An additional benefit is that it is relatively easy to formulate reward functions in RL based algorithms. DRL has demonstrated achieving next level capability in AI for learning algorithms which also avoids the costs associated with data annotation.

DRL based algorithms provide a unique capability over supervised machine learning algorithms yet suffers itself from prohibitive costs, safety issues, and difficult to build and modify training environments. A current popular solution is to overcome these limitations and constraints by training in synthetic environments. Training in synthetic environments is cost effective and safe, and it provides capability to adapt the training environment for different scenarios. When considering the generalization of the trained inferred solutions, a new problem arises: the reality gap (Jakobi et al., 1995). The reality gap appears when failure or degradation in performance occurs when transferring the trained synthetic solution to real-world conditions. Real world conditions will always lie outside of the scope of the training received by agents in a synthetic environment. The Domain Randomization (DR) (Tobin, et al., 2017) technique along with improvements found in Structured Domain Randomization (SDR) (Prakash, et al., 2018) are recent synthetic training works which have shown reduction in the reality gap and have provided new research approaches for improvement in generalization of DRL inferred solutions.

Problem Statement

There are known problems of safety, cost, and training time for Deep Reinforcement Learning (DRL) algorithms. One solution, which supports in some degree this set of problems, has been the generation of synthetic data or the use of synthetic environments during training (Patki, Wedge & Veeramachanei, 2016; Peng, Sun, Ali & Saenko, 2015). Synthetic approaches, however, only represent a limited approximation of the real world. These solutions tend to work well under the same conditions in training but-often fail when real-world conditions, such as unanticipated observations, differ from those of synthetic training (Zhu et al., 2017; Gu et al., 2017).

When a synthetic trained model is applied to real-world use, it often causes a performance problem known as the reality gap (Jakobi et al., 1995). The inference model solution tends to work well in the synthetic environment in which it was trained but lacks the ability to generalize for real-world use. DR is a technique which provides variation to the synthetic environment so that it can better generalize to the real-world. While DR supports a partial solution to the reality gap, it can still lead to suboptimal high-variance policies and thus requires additional adaptation to achieve an improvement in generalization. This problem causes a uniform sampling in the DR process which in turn causes the agent to remain within a repeated cycle of solutions paths and thus limit the exploration. The remedy becomes one of balancing exploration with exploitation. Although DR has proven to be very effective, additional measures are required to achieve improved generalization. One DR-based adaptation to overcome this problem is Structured Domain Randomization (SDR) (Prakash, et al., 2018). SDR provides indications of improvement over DR by inclusion of embedded context aware elements in the synthetic training environment and providing more structure to randomization during training. SDR is a very recent research area with limited experimentation and application and thus requires more research. A modified SDR (SDR+) was examined and validated in this work through the use of DRL-based learning algorithms (Francois-Lavet, Henderson, Islam, Bellemare & Pineau, 2018;

Mnih, V., et al., 2015; Arulkumaran, Deisenroth, Brundage, Bharath, 2017), through experimentation of training the agent within a synthetic environment framework (Lester, et al., 2013; Frutos-Pascual & Zapirain, 2017; Unity, 2019) and based on the use case of drone racing (Loquercio, et al., 2019).

Dissertation Goal

A framework was developed a framework which incorporates a modifiable synthetic environment with an integrated DRL algorithm. The framework served to examine and validate a SDR+ technique for potential generalization improvement of the given DRL learning algorithm. An inferred DRL model was obtained from a SDR+ training technique which includes naturalistic domain randomization and training supported context-aware elements. These modifications to the training environment were assumed to provide sufficient variation to the environment and guidance through context aware elements to leverage and improve upon the SDR technique. This concept, incorporating guided variations within the training synthetic environments, while not changing the DRL algorithm, was the means to reach the goal in this work. The DRL-based framework applied use case scenarios in the domain of drone racing to quantitatively measure the effectiveness of the methodology. The framework provided the structure to generate, test, and evaluate gathered metrics of the DRL algorithm's performance all within synthetic environments. Generalization was tested with a drone use case. In summary, the key elements in achieving this goal are the use of naturalistic randomizations and context aware elements applied in synthetic environments for training the agent in a drone racing setting.

Research Questions and Hypothesis

Research Question 1: Can expanding upon and modifying SDR research through use of naturalistic domain randomization along with training guidance from context-aware synthetic environment elements improve upon the generalization capability of a DRL algorithm-based agent and help bridge the reality gap?

Research into DR (Tobin, et al., 2017) and then SDR (Prakash, et al., 2018) have shown the ability to close the reality gap caused through synthetic environment-based training of intelligent agents or learning algorithms. DR, followed by improvements in SDR, provide indications that additional variability in the synthetic environment can be learned by the agent. This added variability to the training environment provides the evidence in both DR and SDR that a given agent could better handle or generalize to unseen conditions that arise under real world conditions. The variation and guidance during training allows the agent to better adapt to variance in other environments. The level and type of generalization achievable can only be truly understood via experimentations and then examination of the results. This analysis determines the trained variability through SDR+ can increase generalization in the application of the agent in circumstances which differ from training.

Research Question 2: Can the expanded or SDR+ be effective within a synthetic environment without modifying the structure or nature of the DRL algorithm deployed in the technique?

This research was restricted to training, testing, and evaluation of the SDR+ technique all within synthetic environments. Extension to real-world conditions, such a porting the inferred solution into an actual drone, was not undertaken. Given this condition an assumption is made that experimental findings from the synthetic world would apply in some degree to real-world application. More importantly, it was assumed that this improvement to generalization to bridge the reality gap can take place through adaptation of the synthetic training process without changing the structure or nature of the DRL algorithm used in the agents. This constrains the research to be independent from modifications to the DRL algorithm itself.

Hypothesis 1: Naturalistic domain randomization provides an easier and a better structured approach to the existing domain randomization approach.

The methodology used in DR research tends to apply a wide range of randomization forms to synthetic features such as lighting, texture, size or scale, color and so forth. These randomizations are often performed in a non-realistic or natural way. This was done intentionally by design to easily modify the synthetic environment. This research hypothesized that moving away from a broad base of extreme randomizations, as seen in DR and SDR, to a more natural approach could provide the same or even better results than the more extreme DR applications. It was hypothesized that employing more realistic randomizations, closer to those more likely to be encountered, and providing more instinctive environments to train the agent which align better with variations than might be seen under real-world conditions. Such conditions would include time of day, shading, shadowing, climate changes such as rain, snow, fog, smoke and haze, time of year or seasonal based changes, foliage changes, region of environment (coastal, inland, desert, snow bound etc.). Naturalistic randomizations would include changes to manmade elements such as roads, vehicles, and buildings without morphing outside the norm of these elements. Experimentation through ablation studies examined the effect of the timing, frequency, and the degree of modifications to the synthetic environment to determine their effect during the training.

Hypothesis 2: SDR+-based context aware elements can provide training guidance.

The second hypothesis builds upon the SDR concept (Prakash et al., 2018) of using context aware elements within the synthetic environment. Context aware elements are understood as a means that let DRL-based agent understand the context of specified elements within the synthetic environment. The basic premise is that sensors provided feedback on objects in the synthetic environment. The return from the sensor provides an indication of the context of that element. Is it a road or building? If so, it might have context of something which the agent should avoid. It might also signify that the road may serve as a path for the agent to follow. It is hypothesized that such context aware elements could help guide the agent in the proper path as well to overcome uniform randomizations in DR which leads to stagnation of agent actions around local minima. Evidence to point to the usefulness of embedded context was also demonstrated in the determination of segmentation using a deep neural network for object detection (Zhu, Urtasun, Salakhutdinov & Fidler, 2015). By determining global parameters of a scene within a synthetic environment, context regarding a region in this environment can be placed into a graphical entity in the environment such as spline or mesh with control points that can then be triggered as the agents interacts with the environment. A relationship exists between the global parameters and context splines. That context is further classified by a relationship between the context spline and the objects which reside along that spline. Control points placed along the spline provide indications of context and clarify the object to control spline relationships. Control points without objects reflect the global parameters to context spline relationships. The hypothesis proposed that these context aware elements play a critical role in the learning algorithm's ability to balance the fundamental concepts of DRL algorithms, those of exploration and exploitation during training. Furthermore, they provided a guide to better identify difficult to determine objects due to limited pixel size and the ability to better distinguish objects of limited size or lack of detection due to occlusion. In summary, the context aware splines act as advisors to DRL agents and supply hints that direct toward reward and away from punitive objects within the scene. This is useful because the naturalistic domain randomization in the process resets the agent's learned responses and would thus cause the training time to lengthen.

Relevance and Significance

AI applications have been developed for knowledge reasoning, planning, recommender systems, natural language processing, computer vision, robotics and much more. This demonstrates a wide scope of applicability. A good portion of recent applications and innovation in AI are based on the development of intelligent unmanned autonomous systems or UASs. Conveyors of such systems can be used on the ground, in the air, on or below the water, underground or in space. Within the domain of intelligent UASs, RL and DL and have demonstrated the means to advance UAV capability through integration of autonomous systems. Combining RL and DL to form DRL based algorithms was a natural progression in achieving better UAS autonomy when used in conjunction with their sensing capabilities.

Historically, DRL first gained success in games such as backgammon (Tesauro, 1995), Atari games (Mnih et all, 2015), Go, Texas hold'em: Libratus (Silver et al., 2016), Deepstack (Moravcik et al., 2017), and more recent games such as Montezuma's Revenge (Burda, Edwards, Storkey & Klimov, 2018) and StarCraft II (Vinyals et al., 2017). Many of these techniques have demonstrated success in achieving better than human play in environments where the game itself becomes the environment in which the DRL algorithm was applied. The environment plays a critical part in both the RL and the DL components. More importantly, DRL has shown promise in real-world applications such as self-driving cars (You, Pan, Wang & Lu, 2017), speech systems (Frazel-Zarandi et al., 2017), smart grids (Francois-Savant et al., 2016), robotics (Kalashnikov et al., 2018) and finance (Deng, Boa, Kong, Ren & Dai, 2017). This variety implies a potential for DRL on a wide range of problems especially in consideration of using a Serious Games-based approach (Lester, et al., 2013; Frutos-Pascual & Zapirain, 2017) which uses a gaming technology to train in synthetic environments.

DRL supports conditions where human abilities to formulate decisions and act appropriately are exceeded in speed and action as well as when human interaction is a safety concern. All of these are valid points for the development of autonomous system applications. Recent advancements in DRL indicate potential for augmenting UASs for task processing or human assistance with intelligent capability (Humr, 2019). Examples of recent works within the DRL domain for autonomous vehicles illustrate its potential and provide use cases for scenario to drive virtual experimentation (Anderlini, Parker & Thomas, 2019; Chaoxing & Li, 2017; Zhang, Wang, Liu & Chen, 2019; Khan, 2018; Wang, Wu, Liu, Li & Negenborn, 2019; Munoz, Barrado, Cetin & Salami, 2019; Huang, et al., 2019; Liaq & Byun, 2019; Yang & Liu, 2018; Zeng, Ju, Qin, Hu, Yin & Hu, 2019; Kersandt, Munos & Barrado, 2018; Polvara et al., 2017; You, Diao & Gao, 2019; Kumar, Bharathi & Indhumathi, 2019; Leclerc et al., 2018). The significance of the performance that DRL demonstrates, provides value and incentive to overcome any limitations or issues which currently exist while providing a significant use case for autonomous systems. By design, a DRL algorithm is positioned as an ideal learning algorithm to learn variance from exploration and exploitation in a synthetic training environment.

Indicators for Success

Indicators which provide insight into the successfulness of this research are based primarily on the success of the related research works in literature. Examination of the various research works illustrate the probability of success of this work.

First, DRL techniques demonstrated their capability in winning challenging games starting in the mid-1990s up to the current time (Tesauro, 1995; Mnih et all, 2015; Silver et al., 2016; Moravcik et al., 2017; Burda, Edwards, Storkey & Klimov, 2018; Vinyals et al., 2017). These works demonstrated DRL's capability for complex, diverse and higher dimensional inputs. Even more impressive was the use of DRLs for real-world applications of self-driving cars, natural language processing, robotic, smart grids, finance and defense related applications (You, Pan, Wang, & Lu, 2017; Fazel-Zarandi, et al., 2017; Kalashnikov, et al., 2018); Francois-Lavet, 2016 ; Deng, Boa, Kong, Ren & Dai, 2017; Anderlini, Parker & Thomas, 2019; Chaoxing & Li, 2017; Zhang, Wang, Liu & Chen, 2019; Khan, 2018; Wang, Wu, Liu, Li & Negenborn, 2019; Munoz & Barrado, 2019; Huang, et al., 2019; Liaq & Byun, 2019; Yang & Liu, 2018; Zeng, Ju, Qin, Hu, Yin & Hu, 2019). The success of DRL learning algorithms imply their adaptability and robustness as an algorithm would be suited to learn variability through applied SDR+ techniques in synthetic training environment.

Second, for any complex intelligent system, real-world training is a costly endeavor and impractical in many situations. The generation of synthetic data (Patki, Wedge & Veeramachanei, 2016; Peng, Sun, Ali & Saenko, 2015) has presented itself as a training solution especially for machine learning. A DRL-based approach using a synthetic environment training is a natural fit. Serious Games, which leverage gaming environments, have proven to be successful in simulation and training based on aspects of real-world applications. Examples of this has been used in applications for military operations (You, Diao & Gao, 2019; Kumar, Bharathi & Indhumathi, 2019; Leclerc et al., 2018). Leveraging serious games, virtual environments, or simulations as a framework for testing and evaluation of DRL-based AI techniques provides a cost-effective means for training, testing, and validation in a moldable synthetic environment. Serious gaming community has grown significantly over the past two decades.

A third indicator for success is related research which has shown promise in areas of robotics (Tobin et al., 2017; Ramos, Possas & Fox, 2019; Tobin et al., 2018; Muratore, Treede, Gienger & Peters, 2018; Vuong, Vikram, Su, Gao & Christensen, 2019), drones (Polvara et al., 2018; Sadeghi & Levine, 2017; Loquerico et al., 2019), self-driving cars (Johnson-Roberson et al., 2017; Pouyanfar, Saleem, George & Chen, 2019) and computer vision (Ren et al., 2019).

Finally, several works have modified DR to overcome the limitations of uniform sampling of environment parameters when suboptimal high-variance policies are manifested. Examples include learning the parameter sample strategy in Active Domain Randomization (Mehta, Diaz, Golemo, Pal & Paull, 2019), leveraging Actor Critic methodologies (Pinto, Andrychowicz, Welinder, Zaremba & Abbeel, 2017), applying Pyramid Consistency with DR (Yue, et al., 2019), minimizing the Lipschitz constant (Slaoui, Clements, Foerster & Toth, 2019), applying non-realistic style in DR (Tremblay et al., 2018), and using context-aware synthetic data in SDR (Prakash et al., 2018).

Barriers and Issues

The barriers related to this work have been addressed in the research literature. RL (Sutton & Barto, 2018), which plays a large role in DRL algorithms, demonstrates the barriers and issues in the inherited techniques. There are many approaches to RL (Hafner et al., 2018; Mnih et al., 2015; Silver et al., 2016) as well as evidence that RL based applications perform well in synthetic environments (Lillicrap et al., 2015; Mnih et al., 2015; Silver et al., 2016) however, they still present difficulties and issues in overcoming the reality gap. Although slightly different, this also applies to uses of DR and SDR which themselves were meant to bridge this reality gap. They illustrate improvement but also demonstrate issues and limitations to the degree of generalization expected. The real world provides an extreme case of high-dimensional continuous states and action spaces, making it impractical to form even a close approximation. Being able to sort out and reduce the dimensionality to a small degree is possible, however, it introduces issues on determining the proper dimensional reduction from both an algorithmic point of view and the implementation of synthetic environments regarding its defined structure.

Agent tasks may be only partially observable and dynamic in nature which becomes challenging for even the best performing intelligent agents. This also makes the training process even more difficult in terms of how the synthetic environment can be modified to instill behaviors learning to accept and understand variability or adjustments for unseen events. The concept of integrated context-aware elements assumes improved support in these areas when coupled with advances in sensing via the DL elements in DRL. These add their own issues and barriers as well. One barrier associated with RL is the formulation of reward functions. Integrating DL with RL elements, makes it easier to formulate RL reward functions. In the SDR research it was observed that DL becomes less effective for object recognition of obscured or very small objects. In similar fashion it was a difficult task to determine and define how to properly integrate context aware elements to provide the correct amount of guidance. The potential problem was that use of inappropriate context-aware elements may invalidate the usefulness of the DRL algorithm. This was caused through overly deterministic context and information which causes the agent to just follow the context-aware elements versus understanding the context of the scene from the elements. The DRL algorithms capability to explore was overshadowed. It could also lead the DRL algorithm to explore incorrect space or re-explore space unnecessarily.

Another issue, which was only partially technical in scope, deals with the human need to have explainable policies and actions of the agents. Humans require reassurance that the intelligent autonomous systems act and perform as intended. Public policies are put into place to protect the public from developing autonomy which would do harm either intentionally or unintentionally. The industrial concern was that the agent initiates actions which can have detrimental consequences thus voiding the use of any such agent. This set of issues is beyond the scope of this work, however, the concept to assure an agent's response to given stimuli was as expected or appropriate.

The final barrier, also considered mostly out of scope of this work, was the requirement for agents to react at the speed and frequency of the intended use. It was often a goal of agents to provide solutions and actions faster than current systems or human capabilities. In synthetic environments most items behave in an ideal manner. Real world the physics, dynamics and interactions are difficult to model especially for cases of failure, decreased system performance, faulty components, degraded environments in terms of weather, communications, and many other anomalous considerations. This work recognizes the difficulty these conditions imply.

The intent was to scope and experiment within a reasonable set of constraints which can at least demonstrate the method's ability to improve generalization through modified synthetic training.

Assumptions, Limitations and Delimitations

Assumptions

A DRL algorithm serves as the base component in this work and was assumed it to be the best choice of artificial intelligence-based algorithm for this work. A set of research works (You, Pan, Wang, & Lu, 2017; Fazel-Zarandi, et al., 2017; Kalashnikov, et al., 2018); Francois-Lavet, 2016 ; Deng, Boa, Kong, Ren & Dai, 2017; Anderlini, Parker & Thomas, 2019; Chaoxing & Li, 2017; Zhang, Wang, Liu & Chen, 2019; Khan, 2018; Wang, Wu, Liu, Li & Negenborn, 2019; Munoz & Barrado, 2019; Huang, et al., 2019; Liaq & Byun, 2019; Yang & Liu, 2018; Zeng, Ju, Qin, Hu, Yin & Hu, 2019) gave credit to DRL through demonstrations of its promise as a learning algorithm. The DL elements that make up part of DRL acts as a sensor for the agent, however, it was founded upon NN whose solutions tend to be opaque and difficult to explain. Furthermore, the RL part of DRL provides a good learning algorithm for exploration and exploitation of synthetic environments. One of the major assumptions of this work was that variations in synthetic environments during training transfers to an agent's capability to handle variations seen in real-world conditions. Furthermore, it was assumed that the embedded context aware elements provide DRL algorithms with sufficient training guidance without any modification to the DRL algorithm.

Limitations

This work emphasized a technique that provided improvement in the generalization of synthetically trained DRL algorithms. A conscious decision was made to use a specific DRL algorithm to limit the focus to methods of generalization achieved via DR related techniques. The approach was to accept the algorithm as is, rather than apply modifications to its structure. Although such modifications are possible it was constrained to its initial form to scope the effort and provide focus on the generalization techniques applied via the variations applied to the synthetic training environment. Modification to the DRL algorithm configuration and parameter settings still need to be adjusted as is the case with any agent development.

The ability to represent the real world in a synthetic environment is impossible and the closer the approximation is to the real world the greater the complexity and computation time. Real world conditions will always be out scope for any simulation and modeling effort. Any synthetically trained system is only a limited approximation but may serve the scoped need. This work set to limit the experiments to simple cases of drone racing scenarios in simple scenes.

Delimitations

The use case of drone racing was specifically selected to serve as a confined and well scoped set of measures used in the evaluation of this work. The parameters and goals of this use case are well understood. A drone, as an unmanned autonomous air vehicle, adjusts its flight parameters in a manner to be most efficient in time between targets which are the gates in this case. This includes the ability to be on target by passing through the gate and not missing it, the ability to navigate the course as prescribed, the ability to detect and avoid static and dynamic obstacles, and to recognize and disregard distractors. Limiting measures of training to these parameters provided better management during development, testing, and evaluation. These parameters and goals also set the metrics by which the effectiveness of this research was measured.

Limiting the scope of realism in the synthetic environment was determined as a requirement, otherwise attempting to provide the best realism possible would consume most of the time and effort for this research. A good representation of adequate realism exists in many serious gaming platforms which serve as the standard for this research. Research (Shah, Dey, Lovett & Kapoor, 2017; Sadeghi & Levine, 2017; Amini et al., 2020) performed in this area points out the effects of using high resolutions along with realistic simulator environments. Generally, they have costs which outweigh the benefits associated with the use of synthetic environments.

Additional limitations were imposed upon the research to achieve manageability during development, testing, and evaluation. One specific DRL algorithm was used as defined with the assumption that similar DRL algorithms have similar results. A limited number of synthetic environments were used for training, testing, and evaluation of inferred solutions. This was practical in terms of the labor required to instrument a large variety of synthetic environments. In this work, initial but simple synthetic environment with naturalist variations were used for purposes of development. A slightly more complex synthetic environment was used for evaluation purposes. Exact physics and dynamics of drone model actuations were not imposed rather the use of a less complex model was suitable for evaluation as outlined. Drone racing metrics are simple, measurable, and easy to understand. Limitations were applied to the quantity and complexity of context aware elements. Placement and use of context-aware elements used were adjusted for the intended use.

List of Acronyms

AI	Artificial Intelligence
DL	Deep Learning
DR	Domain Randomization
DRL	Deep Reinforcement Learning
NN	Neural Network
RL	Reinforcement Learning
SDR	Structured Domain Randomization

Summary

An overview of DRL based algorithms was performed with the intent to understand the makeup of DRL as a combination of RL and DL. More importantly however, was the explanation of utility of DRL learning algorithms that was used for the technique of this research to have an agent learn to accept and handle variability when trained in a synthetic environment. A research goal was presented which defined a framework for training a DRL algorithm and then evaluating it in a synthetic environment to measure improvement in bridging the reality-gap. The value of DRL-based algorithms for use with intelligent autonomous agents was discussed, along with its shortcomings and methods for overcoming these issues. SDR and concepts from DR were presented as works that have shown improvement in this area. A hypothesis was presented which assumes that a modified version of SDR can improve generalization of the solution. This SDR+ utilizes context-aware element integration along with naturalistic randomization of the domain as the proposed means for generalization improvements. DRL-based algorithms have proven to be quite effective and support the costly and labor-intensive data annotation processes currently required in supervised learning techniques. Synthetic data and environments have been proposed and used to illustrate its effectiveness for reduction in costs, flexibility in training, and elimination of safety issues. These added benefits make the case for the value of this research and its potential to add to the body of work in generalization of AI algorithms. The research presented to date for DRL, DR and SDR warrant further research in these areas.

Chapter 2

Review of Literature

Overview

A review of research literature has illustrated how the field of AI has increased in capability due to improvements in computing power, better algorithms, and the availability of data; however, there is a long way to go to make its intended end use practical and reliable. Often gaps and shortcomings in algorithms, processing capability and availability of data often require additional research to improve upon existing concepts. Additionally, there exist other drivers that require measures of improvement which include issues of safety, computational training time, requirement of and degree of labor associated with the annotation of data, financial costs, and the ability to explain what the AI is doing. A common underlying goal in solving several of these AI based problems is to arrive at an inferred solution which can generalize when used in practice. A review of current research literature reveals these facts, however, also provide solutions in specific areas and in part some degree of advancement in this field.

Research literature and the AI community have classified three branches of machine learning algorithms: supervised learning (Russell & Norvig, 2010), unsupervised learning (Hinton & Sejnowski, 1999) and reinforcement learning (Kaelbling & Littman, 1996). Much effort is spent to optimize AI applications to achieve the best computational performance and provide a solution for the intended domain. Each area in machine learning was examined in the research literature to determine the most applicable and beneficial learning technique for the use case which applies for this research work.

For the case of supervised learning, the data is annotated which explicitly provides the answers or indicators thus allowing a direct learning relationship to be formed for either classification or regression type problems. Classification separates data as a member of a class or group based on the training data. Regression applies to continuous data for prediction based on previous instances.

In the case of unsupervised learning, no annotated data was used to compare, group, and surmise an output based on patterns discovered in the input data. The difference between this learning technique and supervised learning is how the data is analyzed. This is done in real-time for unsupervised learning as opposed to offline analysis for supervised learning. Unsupervised learning can be improved using back propagation which reconsiders erroneous classifications by feeding back results which help modify it for the next pass.

The final category of machine learning is reinforcement learning (RL) which differs from both unsupervised and supervisor learning. The algorithm self-learns by interacting within the environment it explores and responding to the rewards or punishment received for a given action. This is essentially a trial-and-error methodology. RL best suits the general use case of autonomous unmanned vehicle navigation and control considered useful in many circumstances including the drone racing domain. When considering RL over supervised and unsupervised learning methods, it was shown to be the better choice for drone racing and autonomous vehicles. The use of RL aligns not only with the selected domain of drone racing but also includes the ability to freely explore the environment during training. There was no requirement to obtain large amounts of annotated data to better achieve generalization. There were limitations in RL as well as training considerations being performed in a synthetic or a real environment. Conclusions examined from the research literature provided evidence to both successful and unsuccessful efforts to achieve generalization of AI based applications as well as the gaps that exist.

Criteria Justification

Advances for RL algorithms have assumed many forms. One of these forms, the addition of DL, enhances the capability which applies to this research work. A literature review and consideration of the utility of DL (Bengio, LeCun, Hinton, 2015) played a significant part in the formulation of DRL, which combines both RL and DR. DL can be considered as an autonomous agent used to find patterns in the data presented using NNs. DL is trained from a dataset and applies that learning in some deployable fashion through training of the NN. DL is added to RL with the intent that they are good function approximators. In the practical application of RL, it is often difficult to determine a good reward function, which is essential. Often a reward function is in the form of a value function or policy functions upon which the NN map states to values or state-action pairs. In the case of autonomous vehicles, the NN or more specifically a Convolutional Neural Network (CNN), can process sensor-based state information such as visual input and then use that input as feedback into the RL algorithm to identify and better predict the reward-based action. For the use case of drone racing, a particular type of DRL called Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017) was selected based on its wide application and ability to perform well. An additional benefit was that a PPO-based DRL algorithms was already integrated into Unity's ML-Agents, which constitutes a fundamental part of the experimental framework used in this study. This provided a solid implementation of the DRL for use in experimentation while constraining the type of learning algorithm used. This permitted the concentration on synthetic training techniques versus details of a given algorithm or trying to determine the impact among a multitude of learning algorithms.

The choice to restrict the use to a PPO-based DRL algorithm was because the intent of this research work was not to explore the nature of a specific DRL algorithm, but rather to explore how external factors can support and improve an existing algorithm. In this case the PPO-based DRL was well known in the literature as a good performing algorithm. The intent was to see how new approaches in the research literature have used techniques in agent training environments to improve generalization. Consideration for the costs and safety issues associated with training a DRL based agent in the real world was a driver for this research. Synthetic based training, while alleviating a large portion of the cost and safety issues, also provides flexibility in modifications to the training scenarios. The results from synthetically trained agents (Patki, Wedge & Veeramachaneni, 2016; Nikolenko, 2019; Schraml, 2019) have demonstrated their own issues related to the reality-gap. Research has shown that the reality gap lessens the ability of the algorithm to generalize beyond the training environment itself. These results led to a relatively new approach called DR. This approach maintained the advantages of training in a synthetic environment while striving to overcome the reality-gap. The basic premise behind DR is to train in a simulator which randomizes the visual renderings. This concept was based on providing sufficient variability in the simulated environment that the inferred model can handle and interpret unseen real-world situations as just another variation. This initial development of DR (Tobin et al., 2017) uses a uniform probability distribution for the placement and modification of objects in the synthetic environment. This uniform probability distribution gives way to suboptimal high-variance policies and thus requires additional adaptation to achieve the desired generalization.

The ability of DRL to solve problems above and beyond human level capability has been demonstrated first in Atari 2600 video games (Mnih, V., et al., 2015) in AlphaGo's (Silver, et al., 2016) ability to win against a top human expert Lee Sedol in the game of Go. Later AlphaGo Zero (Holcomb, Porter, Ault, Mao & Wang, 2018), which did not utilize data from human play, defeated AlphaGo with a score of 100 to 0. Before and since that time there have been various implementations of DRL successfully playing against games such as backgammon, Atari, and Texas Hold'em card game and Deepstack (Tesauro, 1995; Mnih et al., 2015; Brown & Sandholm, 2017; Morvcik et al., 2017). Production quality environments for DRL exist for Facebook (Gauci et al., 2019) and in various fields of application such as sequence prediction (Ranzato, Chopra, Auli & Zaremba, 2015; Bahdanau et al., 2016), solving classic computer science problems such as the travelling salesman (Bello, Pham, Le, Norouzi & Bengio, 2016) and others. In principle DRL can be used to solve many types of real-world problems given the proper training environment.

Often the difficulty in solving real-world problems comes in the limitation of permitting freedom of action of a DRL agent in the real world. This brings concerns for safety, cost, and training time constraints. Additional reasons include the fact that real world conditions change such as weather, location, non-recurring or unseen events which modifies the environment. Overcoming these issues with a virtual environment from a serious games' perspective provides value in lifting many of these restrictions because of the ability to easily make modifications and then re-training. On the other hand, there is a known drawback that is often observed when transferring from simulation to the real-world domains. This drawback is called a reality gap

(Jakobi, Husbands & Harvey, 1995) where limitations of training in a synthetic environment can result in improper ability to handle the injection of noise into the applied real-world environment. This is because the underlying dynamics of the system do not provide a means to handle variation seen under these real-world conditions. SDR modifications to the virtual environment during training allows the learning to see multiple variations of the domain resulting in the assumption that its policy is more robust towards uncertainties and errors. A context-aware capability built into SDR, which was added to DR, serves to give intended context to the DRL training and avoid the problem of reduced exploration as seen in the DR research. This work did not explore the actual transfer from virtual to real-world usage. This was left for future research. This exclusion does not hinder the ability to determine the validity for this method. This was because the validation of the inferred solution was applied to different and more complex synthetic environments not seen during training. These synthetic environments were used for validation. Validating the inferred solutions in synthetic environments was assumed sufficient to demonstrate improved generalization without the extra complexity of applying it to real world environments. The experimentation and metrics obtained in this work were thus restricted to synthetic environments for training, testing, and validation. The experimentation examined and compared of effectiveness of using a SDR+ technique to achieve generalizing the DRL based solutions within the baseline serious game environment of drone racing.

Existing Studies and their Analysis

Previous Works on Generalization

DRL generalization as a concept, strives to achieve improved performance through DRL training methods. This improvement in generalization applies to any operational environment regardless of the differences in the training received versus the operational deployment. The application of a well performing DRL agent trained in a synthetic environment typically demonstrates good results when it is applied to environments which are the same or closely match the one in which it was trained. The size of the state-action space becomes impossible to fully cover in a real-world setting. This introduces the idea of sample efficiency where large amounts of interaction in the environment by the agent becomes necessary to approach better generalization. The question remains, can similar results be achieved through some specialized subset of the training or by other means (Yu, 2018)? Several methods have been examined for sample efficiency in the areas of exploration (Plapper et al., 2018; Fortunato et al., 2018; Singh, Barto, & Chentanez, 2004; Pathak, Agrawal, Efros & Darrell, 2017), experience transfer (Finn, Abbeel & Levine, 2017; Peng, Andrychowicz, Zaremba & Abbeel, 2018; Yu, Chen, Da & Zhou, 2018; Zhang, Yu & Zhou, 2018; Lazaric, Restelli & Bonarini, 2008; Ferrente, E., Lazaric, A. & Restelli, M., 2008; Sutton, R. S., Precup, D. & Singh, S. P., 1999), abstraction by hierarchy (Dai, Xu, Yu & Zhou, 2018; Evans & Grefensette, 2018; Hu, Ma, Liu, Hovy & Xing, 2016; Graves et al., 2016; Sahni, Kumar, Tejani & Isbell, 2017; Vezhnevets et al., 2017; Bacon, Harb & Precup, 2017; Florensa, Duan & Abbeel, 2017; Dayan & Hinton, 1992; Parr & Russell, 1997; Sutton et al., 1999), sample optimization methods (Jaderberg et al., 2017; Raginsky, Rakhlin & Telgarsky, 2017; Beyer & Schwefel, 2002; Snoek, Larochelle & Adams, 2012; Hu, Qian & Yu, 2017; Qian, Hu & Yu, 2016; Wang, Qian & Yu, 2018) and environment modeling (Pong, Gu, Dalal & Levine, 2018; Shi, Yu, Da, Chen & Zeng, 2018).

The use of exploration versus exploitation gives an agent the ability to discover solutions not available under a rigid policy. Noise was typically injected into the output so that the agent continues to explore new space. This injection of noise, however, can lead the agent away from the intend policy path. One work (Plappert et al., 2018) showed that rather than adding noise to the action space it should be added to the actual parameter space. The results of noise added to the parameter space indicated improvement for the given algorithms, domain and experiments performed when compared to action space noise. Additionally, another work (Fortunato et al., 2018) explored randomizing the neural network for both action and parameter space through application near the output layers. Both works suffered from searching without intent or guidance often going back to the same search space. A different type of exploration called curiosity-driven exploration (Singh et al., 2004), has a built in a reward to promote going to less visited spaces but falters in high-dimensional space due to difficulty in determining if the state has been visited before.

Experience transfer is a process to learn from a set of previous experiences. Works in this area include: 1) learning from a model based on averages (Finn, Abbeel & Levine, 2017) which assumes focus on a small set of tasks, 2) using crudely trained policies to look at the environment (Yu, Chen, Da & Zhou, 2018), 3) using a LSTM network to infer the environment during interactions (Peng, Andrychowicz, Zaremba & Abbeel, 2018), and 4) using calibration actions in probing the environment parameters (Zhang, Yu & Zhou, 2018). All these methods lack general transfer capabilities because they only work for a small subset of uses.

The concept of abstraction attempts to take a higher-level state space to a lower dimension to achieve sample efficiency. Several older works (Sutton, Precup & Singh, 1999; Parr & Russell, 1997; Dayan & Hinton, 1992) strive to use a hierarchy with abstraction but have reliance on prior knowledge of the space which requires manual creation of the hierarchies. More current studies (Florensa, Duan & Abbeel, 2017; Bacon, Harb & Precup, 2017; Vezhnevets et al., 2017) have tried to use sub-polices in learning higher-level policies to alleviate the manual creation of hierarchies but, these still required prior knowledge to design rewards. Other works on abstraction (Dai, Xu, Yu & Zhou, 2018; Hu, Ma, Liu, Hovy & Xing, 2016; Graves et al., 2016; Evans & Grenfenstette, 2018) noted that neural networks do perform computations which are abstract in nature but, this works attempted to overcome these limitations by applying a variety of techniques. These attempts included adding memory and embedding logic in the neural networks. These approaches however, created models too large for use or would only allow primitive logic within neural networks.

The optimization approach (Schulman, Levine, Abbeel, Jordan & Mortiz, 2015; Duan, Chen, Houthooft, Schulman & Abbeel, 2016) strives to use the gradient of an objective to optimize a solution but, it provided no means to explore. A work around to the lack of exploration was the use of derivative-free optimization (Beyer & Schwefel, 2002; Snoek, Hugo & Adams, 2012; Such et al., 2018; Whiteson, 2012) via use of random samples in the search space. This approach suffered from slow convergence, sensitivity to noise and scaling difficulties.

Finally, the environment model approach utilized a transition function to identify how state changes based on an action and follow-on reward. This approach qualified it as a supervised learning approach. This work performed well in limited states but fails in higher dimensional space. Attempts have been made to combine both model-based and model-free approaches

(Tamar, Wu, Thomas, Levine & Abbeel, 2016; Weber et al., 2018; Pong, Gu, Dalal & Levine, 2018) where adding the model-free approaches to exiting model-based approaches does tend to improve results. This, however, has difficulty in modeling stochastic environments and to date have only worked in limit cases. Another approach (Shi, Yu, Da, Chen & Zeng, 2018) has incorporated a simulator in which environments were manually constructed and then a generative adversarial network was added to improve the learning technique. In this case learning is separated between the environment and the policy but concluded that the concept of combining them may prove to provide a better generalization.

The set of works examined here took several approaches to reduce or control the amount of state-action space required by DRL agents in each environment. Justification for these approaches was due to the enormity of the state-action space coverage required by the DRL algorithm and the need to reduce that space through various methods of sample efficiency. Many of the pros and cons of these methods were mentioned which also illustrates a wide variety of techniques used to solve the interaction of DRL agents in environments. It was noted that many of these works provide some benefit while falling short in other areas which still leaves gaps in generalization improvements.

Deep Reinforcement Learning

DRL is an Artificial Intelligence (AI) technique that comprises the two widely used techniques of Reinforcement Learning (RL) (Sutton & Barto, 2018; Kaelbling, Littman & Moore, 1996) and Deep Learning (DL) (Bengio, LeCun & Hinton, 2015) both of which have shown remarkable results in recent years. RL utilized the Markov Decision Process (MDP) (Bellman, 1957) and represents a process as an agent which observes and acts within some defined environment. A reward was derived from the action based upon the state and the feedback of that action as well as a diminished summation of previous actions. From this continuous time step loop of action-state feedback, the agent learns.

This work utilized the model of Serious Games (Lester, et al., 2013; Frutos-Pascual & Zapirain, 2017), which creates a game-like or synthetic environment that models a real-world application and is well suited for as an environment for DRL. This environment was abstracted and simplified with respect to the real world. It served as a playground for observations and actions to be taken by an agent which in turn provided the learning experience for the agent. Included in that environment were representations of items to observe and act upon with consideration of object detection and navigation required in a drone racing scenario. The general concept was that items of reward are sought, and punitive items are avoided. In RL, the reward is often based on some function which is generally not easy to define. The definition of a reward function is a common problem in RL approaches. This becomes even more difficult in conditions with sparse rewards within in an expansive and diverse environment.

DL comes into play as the second component of DRL because of its ability to leverage neural networks to represent the often difficult to define reward function in RL. DL, which is a supervised learning technique in machine learning, can utilize a convolutional neural network to determine outputs from specified inputs. A DL network is known to perform well as a reward function approximation which applies to the V-Value, Q-Value, policy, and models used in RL (Francois-Lavet, Islam, Pineau, Henderson & Bellemare, 2018). This simplifies and automates the determination of the reward function in support of the RL based computations.   These combined elements of DRL can provide the means to learn by action and observation within a virtual serious game environment. This implies a self-learning structure with the additional support of defining the elements which drive the self-learning. This work concentrated on synthetic environments which served to provide the location-based scenarios which applied to the use case of drone racing. This approach, using a DRL agent for self-learning in a synthetic training environment was an ideal candidate to examine an AI based technique for improving generalization. This has been applied to a drone racing scenario which has shown promise in research (Loquerico et al., 2019) which has also extended the capability to real world drone racing experiments.

The use of a DRL based algorithms for this work was limited to one DRL, that based on Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017). Details with respect to PPO are explained later. A variety of DRL algorithms exist and have generally been found to provide a higher degree of generalization (Li, 2017). Deep Deterministic Policy Gradients (DDPG) algorithm (Rodriguez-Ramos, Sampedro, Bavle, de la Puente & Campoy, 2019) has been used as an effective learning strategy for autonomous landing of UAVs for continuous state and action domains. A research work (Kang, Belkhale, Kahn, Abbeel & Levine, 2019) utilized a modified DRL to achieve generalization through simulation for a vision-based autonomous flight of a quadrotor. This modified DRL utilized a deep neural network Q-function approach along with training gather from real world data. A survey on DRL (Arulkumaran, Deisenroth, Brundage & Bharath, 2017) indicates that the primary types of DRL algorithms are either value-based or policy-based methods. The survey also included how DL has accelerated the progress of RL in addressing dimensional scalability. DL adds this advantage via deep neural networks which tend to mathematically create low-dimensional features from high dimensional data based upon the inherent nature of how neural networks are formulated. Because of the DL addition to RL, it allows decision-based autonomy to be less intractable as previously seen in RL by itself.

There are many approaches for both the use of value functions and the use of policy search in DRL algorithms. Value functions achieved early success with applications such as Backgammon (Tesauro, 1995) but more recent research works (Mnih et al., 2014; Schulman, Levine, Abbeel, Jordan & Moritz, 2015; Mnih et al., 2016; Oh, Chockalingam, Singh & Lee, 2016; Zhu et al., 2017) have seen success in more complicated environments. An example of a value function based DRL is DQN (Mnih et al., 2014) and follow-on adaptations which have achieved success in playing arcade type games. Policy search methods obtain policies by means of back propagation or gradient-based methods however recent research indicates better scalability through back propagation (Salimans, Ho, Chen & Sutskever, 2017; Werbos, P. J., 1974; Rumelhart, Hinton & Williams, 1988). When a policy was being searched, local minima tend to be problematic in this approach. Guided policy searches (Levine & Abbeel, 2014; Levine & Koltun, 2013; Levine, Finn, Darrell & Abbeel, 2016) helped to overcome local minima stagnation through integration of a few actions obtain from another control. Trust Region Policy Optimization (TRPO) was another improvement for policy search where optimization was limited to regions in which the true cost function still holds. TRPO, by constraining the optimization, requires the calculation of second-order derivatives which can be costly and difficult to formulate. PPO (Abbeel & Schulman, 2016; Heess et al., 2017; Schulman, Wolski, Dhariwal, Radford & Klimov, 2017) works around this optimizing constraint by only requiring the use of first-order gradients and as such is less expensive in computation while maintaining the performance of TRPO. It has also been more widely adopted for RL uses. PPO is essentially a stochastic type of policy gradient method and has the advantage of being used in either discrete or continuous action spaces in environments. Actor-Critic methods tend to combine the advantages of policy search methods with the integration of learned value functions and have shown promise over PPO methods, however, tend to be more complex.

Determination of the PPO method as the selected type of DRL for use in this research work was based upon the advantages stated for PPO. One of those advantages is that it is less complex than other DRL methods such as the Actor-Critic method. Furthermore, two other major decision factors support the selection of PPO. First, PPO is a well-known and often used method. Second, it is implemented and integrated into the ML-Agents module of Unity. Unity with ML-Agents are both key elements of the experimental framework. This makes the use of the PPO DRL very practical for experimentation purposes. Since the emphasis of this work is on the use of SDR+ techniques in synthetic environments and not on a specific type of DRL algorithms, the PPO selection suffices for this work. Future research could evaluate different DRL algorithms with regards to the same process.

Synthetic Based Training

Training AI agents in a real-world environment has obvious benefits of learning from the same environment in which it was deployed. There are however significant hurdles to overcome. The scale of the real-world not only impacts the agent's ability to handle the unknown or unforeseen but the enormity of variations that occur in a real-world environment are impractical to predict or to train for. Furthermore, the deployment of most any algorithm is more than likely to occur in an environment which differs from the training environment. The exponential rise in costs is directly proportional as the real-world representation increases. An incremental improvement requires significant additional costs for both the training environment and impact of needing varied and multiple real-world training environments. Training with agents in a real-world environment brings on concerns of safety, damage during training to the environment, robots, and autonomous agents. There are expenses of labor and materials as well as the length of training and computation time due to the complexity and number of training iterations required.

Overcoming these impacts of real-world training motivates research into other suitable training methods such as synthetic based training and simulation systems. Recent use of synthetic data or environments for AI based training and model development has been in the areas of visual representation understanding. This has a large emphasis in robotics and general machine vision applications. Specific research has been done in the areas of image classification (Borrego et al., 2018), object detection (Gaidon, Wang, Cabon & Vig, 2016; Johnson-Roberson et al., 2017; Mueller, Casser, Lahoud, Smith & Ghanem, 2017; Hinterstoisser, Lepetit, Wohlhart & Konolige, 2017), 3D reconstruction (Handa, Patraucean, Badrinarayanan, Stent & Cipolla, 2016; McCormac, Handa & Leutenegger, 2017), optical and scene flow (Butler, Wulff, Stanley & Black, 2012; Dosovitskiy et al., 2015; Mayer et al., 2016), stereo and depth (Qiu & Yuille, 2016; Zhang, Qiu, Chen, Hu & Yuille, 2016), pose estimation with 3D key point extraction (Mueller, Casser, Lahoud, Smith & Ghanem, 2017; Suwajanakorn, Snavely, Tompson, Norouzi, 2018), and semantic and instance segmentation (Ros, Sellart, Materzynska, Vazquez & Lopez, 2016; Richter, S. R., Vineet, V., Roth, S. & Koltun, V., 2016). These examples, within this specific domain of machine vision have demonstrated the utility of synthetic data and synthetic environments for training vision systems.

Reality Gap

The driving issue for this work was the reality gap. The reality gap occurs when synthetic training fails to translate into adaptation to a more expansive variations encountered in real world operations. A different view on this same problem was manifested by the need to provide an overwhelming large set of real-world data to support generalization (Deng et al., 2012; Everingham et al., 2015; Hinterstoisser et al., 2013). The inability to provide even a close representation of real-world environments became evident in the field of robotics where simulations proved to be useful but lacked the ability to simulate the actual dynamics of robotic system (Brooks, 1992). Synthetic environments are also hampered by the true representation of discrepancies and failures of elements in the environment or from non-ideal sensors and mechanics used to explore the environment (Miglino, Lund & Nolfi, 1995). This creates unseen variations which arise and differ from the expected norm. Such discrepancies and failures are even harder to represent in a synthetic environment. The causes illustrated here have been addressed in many research studies, for example, such as using synthetic environments to train swarm robots (Francesca, Brambilla, Brutschy, Trianni & Birattari, 2014).

Current research efforts to bridge this reality gap are Domain Adaptation (DA), Domain Randomization (DR) and Structured Domain Randomization (SDR). DA is a method which shows promise for learning from unrelated sources and adapting them to a particular problem. The concept was to apply a known solution from another source and then adapt it into the domain of interest (Redko, Morvant, Habrard, Amaury & Bennani, 2019; Ben-David, John, Kolby, Kulesza, Pereira & Wortman Vaghan, 2012; Bridle & Cox, 1990). A common approach to achieve this adaptation was using adversarial machine learning. This creates a feature space which is representative of both domains to make the difference in the domains become more indistinguishable thus reaching a solution space which works for both domains (Hajiramezanali, Siamak, Karbalayghareh, Zhou & Qian, 2017; Ganin et al., 2016). This method has proven to be successful in finding solutions to problems with little to no annotated data. It does become heavily reliant upon the other source which is used to adapt to a new common domain. The model trained on the source domain distribution must still be able to relate or have some similarity to the target distribution. The closer the relationship between the two domains, the better the resultant target. The existence of similarity between domains is generally not be the case. It is assumed that the resulting adaptation becomes more difficult as the domains become less and less similar. Research in DR and SDR illustrate how these approaches differ from the approach taken by DA. The DA research further implies the requirement to have a well-defined domain with ample data. Under these conditions, adaptation is not only difficult but not guaranteed. This provides motivation to use DR and SDR over that of DA. The concentration of this work was on DR and SDR and the following two sub-sections cover research in these areas.

Domain Randomization

Domain randomization (Tobin et al., 2017) was formulated to support robotics research which leverages the ability to train deep neural networks and more specifically DRL in a simulation environment for the task of robotic manipulation skill development. The ability to perform the training in simulation provides the ability to accelerate and fully test capabilities of the robot. In the initial DR research, the emphasis was placed upon object localization with the goal to determine the object of interest. Then the robot could utilize a robotic grasper to pick up the specified object and eventually place it in the desired location as part of a manufacturing process. The application of this research work was hindered by effects of the reality gap. The translation from simulation to real world applications, through a process called system identification, was determined to be time-consuming and error prone. The complex physical characteristics of such things as part flexibility, gear backlash and unforeseen motion of robotic elements are very difficult to model. In related works, real world imagery would be used in the simulation however, the noise associated with the corresponding real-world robot in its vision system was not being captured. These factors for this domain add to the reality gap.

DR (Tobin et al., 2017) was specifically proposed to investigate a means to bridge the reality gap. The approach was to train a model in multiple environments while randomizing elements in the environment during training. The DR work hypothesized that the exposure to a greater amount of variability in the simulated environment during training would generalize better to real world conditions without requiring the tedious and error prone system identification processes. In addition, the need for large amounts of labeled data required in DNN-based applications would instead be replaced by the known data in the simulation which was then randomized. Furthermore, the work moved away from doing any training on real world images and applied the process strictly to learning in a simulated environment. Object randomization occurred by randomly changing the textures of objects in the experiment which included an ablation study to determine the effects of different approaches in randomization.

Other research works tried to match the simulator to real world conditions including the use of high-quality rendering (James & Johns, 2016). These works did not achieve the success expected. One work (Planche et al., 2017) utilized depth information while others (Richter, Vineet, Roth & Koltun, 2016; Cutler & How, 2015; Kolter & Ng, 2007; Ghadirzadeh, Maki, Kragic & Bjorkman, 2017; Zhang, Leitner, Upcroft & Corke, 2016; Rusu et al., 2016; Tzeng et al., 2016; Mitash, Bekris & Boularias, 2017; Abbeel, Quigley & Ng, 2006; Cutler, M., Walsh, T.

J. & How, J. P. (2014); Ven Den Berg, et al., 2010; Antonova, Cruciani, Smitth & Kragic, 2017; Koos, Mouret & Doncieux, 2013) used more exact representation in the simulator combined with fine-tuning, domain adaptation, integration of real-world data, 3D models and iterative learning control approaches. The claims established by the DR works state that it does not rely upon Domain Adaptation for the transfer of simulation to real, it was easier in its implementation and requires less training while being easier to scale.

The DR approach randomized synthetic elements such as: 1) number of lights and characteristics of those lights used in the scene, 2) the position and texture used on the objects of interest, 3) the textures of surroundings, 4) random noise, and 5) the position, orientation, and field of view of the camera and 6) the shape and number of distractor objects used in the scene. These types of the randomizations appeared to be usable and appropriate for the domain of this problem. The DR work was striving to experimentally evaluate the accuracy of localization of objects of interest while determining which elements of the approach provide the best transfer of methods sufficiently accurate for robotic manipulation tasks. In this example for an average sized robot arm, an average tolerance of 1.5 cm in localization was achieved which included distractors and occlusions based on camera angles. The performance of the DR method was weakened because of domain mismatches. The method produced other errors by not being able to localize the edge cases. Misinterpretation of objects near the edges of the table was higher than those further in on the interior of the tabletop. The DR work experimented with more complex textures during the randomization and determined that the model had learned object representation in such a manner that they were invariant to these variations in textures. DR ablations studies determined that pretrained models performed well with fewer samples however, both pretrained and training from scratch models perform equally when the number of samples are sufficiently high. The ablation studies also determined that the localization error continued to reduce as the number of unique textures were utilized in the training. Additional findings indicated that using distractors in the training helped to improve the detection of distractors in the real world. Camera randomization only provided slight improvements. The addition of noise provided no effect other than helping convergence of the algorithm out of local minima.

Since the inception of the DR technique many other research works have examined its potential based on new domain applications or modifications to the technique. Such examples provide qualitative evidence of DR utility. Examples include thermostatically controlled loading with RL using DR for knowledge transfer (Peirelinck, Hermans, Spiessens & Deconinck, 2020); object detection using parametric shapes and synthetic textures (Dehban et al., 2019); automobile detection with pose estimation (Khirodkar, Yoo & Kitani, 2019); combining with pyramid consistency for generalization without domain data (Yue et al., 2019); 3D localization using CNNs and DR (Zuo, Zhang, Zheng, Gu & Gong, 2019); transfer learning research based on DR (Grun, Honinger, Scheikl, Hein & Kroger, 2019); using DR for active pose estimation (Ren et al., 2019); training drones with DR for drone racing (Loquercio et al., 2020); generative models using DR for robot grasping, a follow-on work (Tobin et al., 2018); deep networks with synthetic data bridging the reality gap with DR (Tremblay et al., 2018); DR used to help transfer of control policies to mobile robots (Sheckells, Garimella, Mishra & Kobilarov, 2019); and other similar works.

Structured Domain Randomization

Structured Domain Randomization (SDR) (Prakash et al., 2018) was built on the foundation of DR but implemented with additional features to overcome the limitations of DR. DR utilizes a uniform probability distribution when applying randomization to objects in the environment. This uniform probability distribution was proven to work well for large and obvious objects within a scene. For objects in the scene which are small or occluded, in terms of visual representation for the number of pixels that represent the object, the performance of DR is significantly degraded. SDR took the approach to keep the benefits of DR and address DR limitations by embedding structure and context in the synthetic environment. The SDR work claimed that by adding context-aware elements in the environment, the DR algorithms performed better for the detection of small or occluded objects. This implied an improvement to the NN as it learns and incorporates the context provided in the NN structure. The work of SDR claimed to achieve state-of-the-art for 2D object detection in the domain of car detection for the easy, moderate, and hard KITTI datasets (Geiger, Lenz & Urtasun, 2012).

SDR research work also included the examination of related works which used synthetic data for object detection. Findings from an optical flow research work (Mayer et al., 2018) discovered simplistic synthetic versus realistic environment to be sufficient for training. This led to being able to perform randomizations of less complex synthetic scenes over those that use more realistic renderings. These limitations noted in the DR technique was researched in the SDR work. Improvement in detailed object recognition was achieved when a certain level of realism was added to the synthetic environment. A prior research work (Tremblay et al., 2018) by the same authors of SDR, utilized DR to train for object detection of cars. The work provided results that indicated that DR requires a large set of data for the randomization process. Furthermore, the prior research noted the difficulty in detecting small image representations of cars without providing some context. The importance of context was also noted in other research on robotics and in object detection work through semantic segmentation (Georgakis, Mousavian, Berg & Kosecka, 2017; Zhu, Urtasun, Salakhutdinov & Fidler, 2015).

The approach taken by SDR was to procedurally generate synthetic images of a scene which can then be randomized at the same time. For purposes of this work, context was an indicator of what the object means for the surrounding area. An example would be a road. In the context of a city scene, it would be a pathway. It would also be considered as a hard object which the drone should avoid impacting with such as a building. The context was striving to convey that the road may lead to the desired path for the drone but, it should not run into such an object. It provides additional guidance. In the SDR work, context was structured through a graphical tree representation of the scenario, global parameters, context splines and objects in the scene. The scenario is placed at the root of the tree from which global parameters are determined as a node of the scenario. This hierarchical structure provides a high-level context at the root and works its way to details of a given object at leaf nodes. From the global parameters, which are represented as a single node, a one-to-many number of context splines are defined as children of the global parameters. The context splines lead to specific common objects in the scene which lie along the splines. Each child as a context spline represents each of the specific global parameters defined. These are represented by a spline shape with control points. The control points specify locations in the synthetic environment where there exists context of interest. Objects are then placed at or near the context splines control points based on the specified relationship of the object to the global parameter representation. Other control points along the context spline can include control points with objects defined as distractors or as objects of no interest. This approach creates a conditional dependence between objects in the scene and the context provided by in a tree like data structure. This provides the underlying understanding of the scene structure which was lacking in DR. The context spline concept for SDR was also used to support the procedural generation of the scene. Two-dimensional snapshots or images of the scene were then used for object detection training.

A comparison between SDR and DR for both Average Precision (AP) and the dataset size, indicate a much higher AP for SDR while saturation for training data required by SDR is one fifth that of DR. Improvement in AP while reduction in size of the dataset. are both strong indications of the value of the context-aware nature of SDR. SDR outperformed real-world dataset training when results were validated against differing domains. The SDR researchers (Prakash, et al., 2018) claim improvement over the DR (Tobin et. al., 2017) but further claim that SDR when it was used just for the initialization of the neural network with real data to be used in training, it significantly increased performance when switched back to using real data for training. From the SDR ablation study it was determined that under the normal DR approach that randomization of lighting provided the most impact amongst the various randomized parameters. When SDR was applied the effects of eliminating saturation, scene randomization and context provided the lowest AP results which indicate their relative importance in the SDR process.

In general, the results for SDR are positive. There are, however, additional considerations to understand with SDR. The foremost consideration was that SDR is a singular body of research for the niche area of domain randomization. As a singular body of work in this area there are no known research methods which extend or validate SDR. This fact has the implication that the SDR technique has not been validated through additional research. This implies either a gap, which would be useful to examine, or that the method itself provided no benefit. An assumption was made that the later argument was not valid based on the findings of the SDR work. There are also no other work to validate or contradict their findings. The literature review for this research effort did not find any research related to SDR, although that does not mean it does not exist. It is noted that several thorough searches were made in the most common and recognized research libraries to discover related works or works that referenced SDR. The work of SDR is just over two years old. Most DR related works were only published a year earlier. This research took on the assumption that SDR has useful principles upon which new techniques can be formulated especially using the discovery environment structure by an agent through context-aware elements while simultaneously applying the advantages of DR.

Result Formats from Related Literature

There exists a pattern of presentation and formatting of results found in the research works examined which were leveraged for this study (Tobin et al., 2017; Loquerico et al., 2019; Prakash et al., 2018; Tremblay et al., 2018).

In the seminal work (Tobin et al., 2017) on DR, the study of robotic grasping devices realized value to be found using a synthetic training environment. Their analysis and results were focused on the amount of distance error determined in the positioning of a robotic grasper to the object to be detected, grabbed, and moved. Formats were based on an actual distance in centimeters plus or minus typical error due to the robotic accuracy itself. Seven shapes including distractors and occlusions were applied. For each of the object shapes examined, an error distribution was plotted. For all attempts the number of similar distances were plotted to show the distribution for the given object of distances error. The mean values of this were calculated to show the relative error for a given shape. Additionally, the error distribution was examined based on where the object was positioned on the platform versus the amount of error obtained. A heat map distribution across the table was utilized and divided the area into to a 4 by 4 grid to illustrate if certain locations on the table had impact upon the degree of error obtained. A scale of 0 to 6 was utilized for values across the distribution. Objects on the table edges turned out to be the most prone to errors. Distance error was also measured for use of complex and baseline textures used on the object during domain randomization which included a sensitivity plot of the average distance error determined as the number of unique textures that were utilized in the domain randomization. Finally, ablation studies were performed to understand the effects that certain components have on the over performance. SDR looked at components such as noise, camera randomization and the use of distractors. Both DR and SDR included measurement to determine average precision indicators for the methodology. This research work followed these patterns for average precision indicators and apply them to drone racing in terms of the agent's performance in the drone racing courses.

In the Deep Drone Racing research work (Loquerico et al., 2019) a CNN was used to make predictions on determining the next target direction based on images generated from a camera or from a synthetic image. Local control systems then calculate a minimum-jerk trajectory to stay on target. The agent can pass through the designated gates in the racecourse by using inflight predictions which use the current pose of the drone along with a global target trajectory to calculate the needed corrections to stay on the targeted path. This trajectory formulates a task-based effort for the drone and thus provides metrics of percentage of task completion. Three such tasks presented and formatted in deep drone racing work are completion of the track, number of successful consecutive laps completed and the success of passing through dynamically moving gates based upon the relative distance of moving gates. The first task was based upon the speed of the drone going through the course. It was a measure of how fast the drone can fly before the performance degrades because the velocity begins to be too fast making it harder to pass through gates. The second task measured success on consecutive laps through the course without a crash or fault. The last task measured how far the gates moved as part of a dynamic track or course. The first task with respect to speed was also examined as an ablation study to illustrate the effect of a particular item such as background, shape, or illumination. Another format of the results was the percentage of gate passes to that of percentage of gate occlusion. The work on deep drone racing provides good insight into the data, format, and presentation because the domain is very similar. Given that these stated tasks are formats and approaches which match the intent of this work, it served as a guide for the metrics to be studied.

The SDR work of (Prakash et al., 2018) examined the size of a given dataset and compared average performance for detecting vehicles and providing bounding boxes around them. They were classified as easy, moderate, or hard based upon the bounding box size and occlusion. The SDR work examined the more difficult part of object detection where the object's apparent size was small, and the object was also partly occluded. The size of the datasets used was also examined for average precision. The ablation studies provide the insight into the utility of specific randomization items as well as the context-aware splines. A simple chart illustrated each of ablation studies as an evaluation of average precision at 0.7 Intersection of Union. This presentation of ablation studies made it clear the importance of scene randomization, context, and saturation. It served as a good presentation to illustrate this point when full SDR performance is used in comparison with each ablation study.

Literature Gap

There is plenty of research to indicate the benefits that simulations using synthetic environments to train AI models provides. Research continues to overcome shortcomings related to the reality gap because of the mismatch between synthetic and real-world training. Most research efforts in this area have occurred within the past five years. Given that this is a new and developing area within the AI domain, the amount of literature in support of bridging the reality gap remains somewhat limited. Research such as DR however, provided success in several areas and initiated a series of research efforts to improve or apply DR techniques as outlined in the Domain Randomization sub-section of Existing Studies. The Structured Domain Randomization sub-section as well pointed out the faults of DR and illustrated a method to gain improvement over DR. No known research exists which has incorporated concepts of the SDR method like this work. This works filled this gap. In this area, a literature gap exists because there was only one known research method that has applied context-aware structure to AI related synthetic environment training which was also somewhat limited in scope and application.

DR related research was also lacking where the randomization was directly applied to the synthetic environment in naturalistic and non-uniform manner versus the known methods used in DR. This included works that used high fidelity renderings as the means to bridge the reality gap. The research for SDR also used similar patterns of randomization because of its foundation upon the DR approach. Specific research on the use of naturalistic non-uniform randomization in such processes were unknown and assumed by the efforts of this research to be non-existent. The use of high-fidelity or high-quality rendering of synthetic environments has been considered a means to achieve improved generalization but refuted by another research (Mayer et al., 2018). The general concept of varying these high-quality rendered environments has been applied in these research efforts. This was done in a limited sense and not specifically related to the approach of providing variation to the environment during synthetic training so the model can handle real-world variations.

Summary

The primary goal for this work was to generalize learning algorithms via methods used in the training phase. Examples from literature were examined regarding generalization techniques. Research works in sample efficiency were examined to reduce or control the state-action space which is integral to RL methods. This applies to the PPO-based DRL method. The sample efficiency methods in literature covered a wide range of implementations including exploration, experience transfer, abstraction from hierarchy, sample optimization and environment modeling, which provided a range of techniques used to generalize AI solutions.

The fundamental instrument to which the generalization applies was the PPO-based DRL algorithm. The selection of this learning algorithm was based on a broad view of AI techniques, research literature recommendations that a DRL based approach would provide the optimum approach given the use case under investigation. RL, as a sequence which observes, acts, rewards and updates state-action processes provided the fundamental algorithm. Research also indicated that faults in the RL approach alone could be overcome by adding DL to RL, thus creating a DRL method. Due to difficulties in formulating reward functions and processing sensor-based observations, DL research results proved useful and served as the means to overcome these difficulties. Both value function and policy search research were examined as types of DRL methods. The use of policy search method was considered the best approach for this work due to its common use in literature and it ease in implementation versus a value function approach.

For RL based applications the agent required an environment in which it can observe and act. Doing so in a real-world environment comes with complications of costs for training setup, safety issues, impact and cost associated with embodied agents in forms of unmanned autonomous systems, as well as the lack of flexibility and scalability in training. Research which used and introduced simulation, synthetic data, and synthetic environments for training was examined. The exact negative impacts of training in the real-world were shown in literature to be solved in part using simulation and synthetic approaches. The computational time savings and flexibility in setting up training scenarios were key factors pointed out in the research literature.

Research using synthetic and simulated methods determined that a reality gap was created when deploying a trained model from the simulated or synthetic based-training environment to that of the real-world. Synthetic trained models have a limited representation of the real-world and as illustrated in the research literature an agent would act poorly in the deployed real-world environment. This is essentially the reality gap problem. Research was examined regarding Domain Adaptation as one of the first approaches to help bridge the reality gap. This led to additional research in DR which had indications of being more suitable to solving the reality gap.

DR was formulated based on a problem in robotics where training and testing in the real world held complications. Simulations and synthetic environments provided an approach to overcome the complications but also introduced the realty gap into the solution. The research for DR was based on solving the reality gap for object localization of robotic grasping and manipulation. The uniform probability distribution of randomization of objects in the scene was successfully tested in several research efforts. The concept was to have the agent learn variability as it was trained in a synthetic environment and integrated with DR. The success promoted DR as a research mean to bridge the reality gap.

SDR was founded upon DR with the addition of context-aware splines. SDR research determined problems with DR caused by the uniform probability distribution it used during randomization. This caused an agent to be stuck in local minima. Additionally, for object detection, objects which appear to be small, occluded, or indistinguishable their identification could not be handled properly by DR. Context-aware splines created embedded elements in the synthetic environment with the goal which aided the agent in continuous exploration and provided structured context to the environment as a guide to the agent. The context-aware splines also provided the means to procedurally generate the synthetic environments. The splines contained control point which either contained context for that point or an object placed at that point. Given the research search performed, the SDR methodology is a singular and limited work in this research area specifically for use of context-aware elements.

This discovery path found in the research literature led to the substance of this research work. The requirement to improve generalization of AI techniques for better deployment usage was the overarching goal and provided the guidance in leading towards a proposed solution. Selection of a drone racing domain and the desire to avoid the massive amounts of annotated data required in machine learning, led to the decision to use a RL-based algorithm which allows an agent to learn by exploring the environment. In the use of RL to explore the environment, a sensor-based observation capability was required along with a better way to formulate reward functions. These capabilities are provided in DRL which is a method that incorporates both RL and DL in a PPO-based implementation. The DR methods was one of several methods that demonstrated ability to bridge the reality gap. Further research literature investigation discovered the SDR methodology as a modification to DR. SDR reported even better results for generalization through methods to augment DR. Finally, conclusions from the literature led to improvement concepts which are more unique to this research work. First, the randomization was based on naturalistic randomization and, second, additional modifications to context-aware elements while providing more control over the environment helped the agent learn how to understand and perform with unseen or unplanned variability. The experimentation and validation of concepts and approaches of this work were built upon the body of work covered in the literature review. This foundation provided good indications that this work could contribute and provide an additional step forward in training AI systems in more convenient and cost-effective ways.

Chapter 3

Methodology

Overview

The following list provides a high-level view of the methodology for this research work. Each item in the list illustrates key factors in the methodology. They comprise the techniques and methods used to arrive at a solution to the proposed problem.

1. Determine through simple agent testing those DRL PPO-based algorithm configuration settings that should be used in the drone agent. Several options exist and providing a series of simplified tests yielded a better selection for configuration parameters.
2. Run experiments to determine if a proposed training methodology can provide improvement in generalization to the inferred solution. Experiments include the training, testing, and validation of inferred solutions.
3. Utilize a DRL algorithm that is based on a PPO implementation. No modifications to the algorithm are made, however, there are typical allowances made for configurations, parameters, and settings.
4. Perform all training, tests, and validations within the structure of synthetic environments. Apply a use case of drone racing to gather and score metrics.
5. Leverage DR and specifically SDR techniques in the methodology.
6. Develop an experimental framework which integrates all necessary components and allows for variations in the synthetic simulation environments, agents based on a DRL algorithm to effectively explore and exploit the environment, incorporate variance through naturalistic DR with improved training via context-aware elements.
7. Build into the framework methods to capture and measure specific metrics. Include ablation studies to examine the sensitivity and effects on the results for both naturalistic DR components, use of context-aware elements and the application of settings and parameters applied to the DRL PPO agent.
8. Compare and contrast a baseline trained agent to that of agents which incorporate various elements of naturalistic domain randomization and embedding of context aware elements.

This work was based on the two research questions and two hypotheses that were previously mentioned. Each question and hypothesis provided the foundational structure to the type of research to be conducted and was formulated through examination of research literature. These are discussed in the following sections which describe the specific sections of the methodology.

Research Methods

The methods utilized in this research are described in distinct subsections to help illustrate the purpose of the methodology by breaking it down into logical groupings. When these are combined, they provide a composite approach in answering the research questions of this work along with testing the proposed hypotheses. Relationships, connections, or reliance for each category to other categories was declared in each section. The last subsection describes the integrated picture of the entire methodology.

Testing to Select DRL PPO-Based Algorithm Configuration Parameters

This work did not concern itself so much with the modifications or adjustments to the algorithm used. There are three factors, however, which were required of the algorithm to be used for learning agents. First, the algorithm must be suitable for the primary purpose of the research, the use of the SDR+ approach with the purpose to yield improved generalization. Second, the algorithm would be suitable for use in synthetic environments and without annotated data, rather learn from exploring and exploiting the synthetic environment. The third factor is the main discussion for this section. Given the selection of the DRL PPO-based algorithm from previous discussions, what should be the configuration parameters for this learning algorithm that were best suit the methodology? This was answered via a methodology to perform simple agent tests to examine each of the configuration parameters and determine which set of parameters would be most effective.

A description of “simple agent tests” was outlined as follows. Determine which configuration parameters were available to the DRL PPO based learning algorithm found in the Unity ML-Agents software. These parameters defined the series of tests required. The framework for these tests consisted of the following:

- In Unity define the agent as a sphere which is restricted to motion along the plane of the x and z axes. This agent was configured with DRL PPO-based algorithm supplied in ML-Agents.
- In Unity define a stationary target as a block. The goal of the agent is to run into the target. A reward of 1.0 would be obtained. This process would then repeat until the ability to consistently achieve this goal was obtained.
- In Unity define connected sections of flat planes upon which the sphere agent can traverse and where the stationary target can reside. The agent sphere is gravity based and when it left the define platform gravity would take over the agent sphere would fall. At this point it was penalized and a new training episode would begin.
- A degree of difficulty was provided to the agent via the shape of the platform upon which the agent moved. The concept was to bridge two square regions by a narrow plane which would bridge the two areas. The agent and target were placed on opposite squares or side of the platform so that the agent needed to traverse the narrow part of the platform.
- A series of test were developed based on the various combinations of configuration parameters that existed for the ML-Agents learning agent. Each test was evaluated to determine the best set of configurations to use.

The series of simple agent tests that took place were based the various combination of the learning algorithm parameters. Three main configurations existed which were: 1) environment observations, 2) ray cast-based sensors, and 3) visual sensors based on a special Unity camera object which is attached to the agent. For these three configurations a total of seven combinations existed: just observations, just ray cast sensing, just visual observations, combination of observations and ray casts, combination of observations and visual, combination of visual and ray cast sensing, and finally a combination of all three. It should be noted that visual observations needed to be included given the visual nature of naturalistic domain randomizations. Before the experiments began it was assumed that all three configurations combined would be the logical choice based upon the added capability of each configuration.

The visual observation sensor also provided a total of four more additional configurations. These options are based on the type, size and configuration of neural networks used for the visual observation sensor. The options consisted of simple, natural CNN, resnet and match3. For testing purpose this set of configurations was only tested with all three main configurations combined, that being, environment observations, ray cast sensor and visual observations sensor.

The visual observation sensor configurations are part of the ML-Agents neural network settings. The simple configuration is the default simple encoder which has two convolutional layers with minimum observation size of 20x20. The nature_cnn configuration is a Convolutional Neural Network (CNN) implementation (Mnih et. al, 2015) made up of three convolutional layers with minimum observation size of 36x36. The resnet configuration uses the IMPALA Resnet (Espeholt et. al., 2018) implementation which has three stacked layers with two residual blocks. The minimum observation size is 15x15 however, this is a much larger neural network and thus more computational expensive. The last configuration option is match3 which is a smaller CNN (Gudmundsoon, Eisen, Poromaa & Nodet, 2018) but can capture the granular spatial relationships as one might find in the examination of board games. The minimum observation size limitation for match3 is 5x5.

With these configurations for visual sensor observations along with the other combinations, there was therefore a total of ten simple agent test that took place in the experimental framework previously discussed. These tests are listed below:

1. Simple Agent Test 01 – environment observation only
2. Simple Agent Test 07 – ray cast sensor only
3. Simple Agent Test 08 – visual sensor only
4. Simple Agent Test 03 – environment observation with ray cast sensor
5. Simple Agent Test 04 – environment observation with visual sensor
6. Simple Agent Test 06 – ray cast and visual sensors combined
7. Simple Agent Test 05 – ray cast sensor, environment observations, and visual sensor with simple configuration combination

8. Simple Agent Test 09 - ray cast sensor, environment observations, and visual sensor with natural CNN configuration combination
9. Simple Agent Test 10 - ray cast sensor, environment observations, and visual sensor with resnet configuration combination
10. Simple Agent Test 11 – ray cast sensor, environment observations, and visual sensor with match3 configuration combination

Experimental Framework

An experimental framework was be developed to serve as the means to test and validate the methodology. Furthermore, the framework allowed data to be gathered and then analyzed to determine the level of generalization achievable compared to a baseline model from the training environment. This same process assesses the viability of the proposed hypotheses. The framework is primarily software but includes the supporting hardware specifications such as a good quality workstation computer with sufficient memory, processing power and a high-end graphics card. These are components commonly found in most modern workstations.

The framework was based on the Unity game engine (Unity, 2019). Unity, as a Serious Game framework (Frutos-Pascual & Zapirain, 2017), provides the structure and API support required to develop and utilize synthetic environments. The API supports a scripting tool utilizing C# to integrate development code into Unity objects and interfaces. For this work Unity constituted the main component of the framework and it permitted integration of required support components. Unity provides modules as add-on features or as synthetic objects, both which enhance the framework's capability and extensibility.

A more detailed understanding of Unity helps illustrate how it was be used both conceptually and operationally. Unity is primarily considered a game engine used to develop video games and simulations. It is a cross platform development tool but, for this work the Windows 10 operating system platform was utilized. Unity as a serious game platform provides the ability to construct the synthetic environments as training environment for AI agents. Built-in features of Unity allow the creation, manipulation, and destruction of two- and three-dimensional objects as well as obtaining access to the objects via C# scripting and built-in tools. Events for objects were created which include time and spatial manipulations and both physics and graphics-based effects. Objects were created or provided as pre-built objects or obtained through third-party asset sources. Unity has a concept of a 'prefab' A prefab is a single or composite set of objects which was re-used multiple times and modified in their re-use. An environment can contain either a portion or a composition of both separate objects and combined objects as prefabs. All these Unity based capabilities are stored, positioned, accessed, modified within the Unity application with various tools for scene management. The content of the created synthetic environment, generally viewed as a scene, was run directly in the Unity application, or complied as an executable application. This work leverages these features of Unity for the creation of synthetic environments which were then run as the training, testing, and evaluation environments. These same tools used in conjunction with the integrated C# scripting allowed this work to modify the synthetic environment both for the naturalistic domain randomization and embedding context-aware elements within a scene. A common feature of DR is to procedurally generate synthetic content to save labor and costs in building synthetic training environments. This work in contrast, utilized Unity supported hand-crafted environments, versus procedurally generated environments. This allowed for a more controlled and understood structure for the experiments and since there are a limited number of environments this became less of a factor. Although this was more manually intensive, it was better suited for the experimental nature of this work.

A Unity module, ML-Agents, is an enhancement which this work utilized to integrate existing capabilities for a PPO-based DRL algorithm. PPO or Proximal Policy Optimization (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017) has proven itself to be comparable to other similar algorithms but is easier to implement using a first-order optimizer like the gradient decent method. This single PPO based DRL algorithm was used as is. This allowed emphasis to be placed on the SDR+ techniques put into the training environment. The integration of this algorithm was essentially automatic when ML-Agents was added as a Unity package. ML-Agents comes pre-installed, however, requires some integration and adjustments. ML-Agents comes with several working code examples of AI algorithms linked to synthetic training environments in Unity. The PPO-based DRL algorithm is one of several that have been used with ML-Agents. There exist several advantages to this ML-Agents integration. First, a well implemented PPO-Based DRL algorithm is readily available. Since the emphasis of this work was not on the inner workings of a DRL algorithm, time and effort are saved from an implementation and in turn was focused on how generalization of that algorithm could be improved. Second, with the addition of ML-Agents to Unity, the algorithm comes integrated into the framework including the procedure to connect to Python-based code used to run the algorithm. This connection was already established in ML-Agents. Access to Python code was available and no code modifications were required other than applying the proper configuration and setup of parameters. These were used to adjust the settings of the algorithm.

Unity's C# scripting API becomes the means within the framework for the additions of the naturalistic randomization of the environment and context-aware elements. Unity provides many instruments within its application that were programmatically achieved through the C# scripting API. This scripting capability was used to modify objects, prefabs, scenes, lighting, camera properties and any element of the environment. Such manipulation of the environment supports the ability to naturally randomize elements of the synthetic environment. C# code was tied to objects, prefabs, and other element in a scene as well as associated events. The C# scripts were written to trigger randomizations on Unity based objects, graphics, or simulation properties. The control to randomize the objects, graphics and simulation properties subscribed to natural affects. C# scripts can control environmental elements and conditions such as time of day, weather, shadows as well as the physics of objects. Some of the C# scripts allowed management of environmental randomization while others provide information to the sensors of the agent through being able to build content into the objects examined by the sensors. With the scripts in place an agent can explore the synthetic environment which was automatically preset to randomize during the training. For purposes of controlled experimentation, a set of seeds would allow a range of parameterizations or strategies for the agents to undergo within a controlled set of randomized environments. In a similar fashion the C# scripts was also tied to the activation and use of context-aware elements. This becomes a bit more involved since the context gets tied to objects and locations in the scene. For the framework, it was sufficient that context-aware elements are formulated with a combination of objects and their integration in Unity scripts.

The final element of the experimental framework was the ability to gather data based on metrics for the domain of drone racing. This includes data related to the agent's interaction within the synthetic environment. Instrumentation of the synthetic environment was achieved through existing elements in the Unity application as well as C# script code. This was written specifically to record an agent's actions and the elements within the synthetic environment with which it interacts. Part of this is driven by intersection or collisions of volumes, surfaces, and lines from the agent's actions in the environment. These intersection-based collisions trigger events and responses which were include the recording of specific information related to the metrics of interest and potentially to understand how well the training is being absorbed by the agent.

Deep Reinforcement Learning PPO-based Algorithm

The experimental framework provides a synthetic environment for an agent to explore as part of its training process. It was mentioned that in the framework there wa an integrated ML-Agent capability in Unity which provides a process and the code to apply this to synthetic agent training. ML-Agents has a Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017) based DRL algorithm which was used for this work to see how external factors might affect the algorithm's training capability. Direct modifications to the algorithm were not necessary. This research maintains scope by restricting it to one DRL agent algorithm with the assumption that adequate results were obtained with a PPO-based DRL algorithm.

ML-Agents comes with examples of two primary DRL algorithms. They are both effective DRL algorithms, but each have separate advantages. SAC is utilized when the environment is expensive in terms of sampling because of its sample efficiency. If the sampling is more abundant and easier to obtain, then PPO is better because of its straightforward implementation, ease of set up and robustness in terms of the hyperparameter settings.

The methodology of this work required a proper setup of ML-Agents to be able to have the drone agent work with the PPO-based DRL algorithm inside the synthetic environment. ML-Agents communicates via Python through a tool such as Anaconda. The ML-Agents library and code was downloaded and installed in Unity as a package and the code which leveraged the ML-

Agents package was placed and utilized within the specific project. The code was tied to specific items which gets acted upon by the agent or to the agent itself. The Unity interface allowed the developer to connect the scripts to the representative objects in the scene to provide the behaviors to the objects or agent. An Anaconda environment was created and run on the same machine containing the Unity application. ML-Agents provides the connections between Python, TensorFlow and the Unity engine. The Python environment was set using an existing ML-Agents code which installs TensorFlow, and other utilities required. A training configuration file was set up and then used to configure how the Python code was executed including the necessary instructions regarding location and inclusion of parameters when training the agent with this code. The Unity application was started with the specific project and waits for the connection with the Python code. The Python code indicated that "Play" in the Unity interface should be executed. In Unity the Play feature not only starts the synthetic simulation but it also initiates ML-Agents. For training purposes this was a requirement based on the way ML-Agents is integrated into Unity. A product of the training was a model which was stored in a file with an onnx extension. These files can then be used for the testing and evaluation phases of the experimentation. Since TensorFlow is utilized, ML-Agents can also leverage a dashboard called Tensorboard, where output during training were examined to help determine convergence and performance of the agent during training.

Synthetic Environment

Synthetic training for agents permits flexibility in the creation and use of synthetic environments. Many obstacles found in training agents in a real-world environment have been overcome using synthetic environments. This provides emphasis for the continued use of synthetic environments. An adaptable environment was a strong requirement for this work for both domain randomization and the use of context-aware elements.

The Unity framework was a natural fit for training agents in a synthetic environment. The core functionality of Unity is the creation of objects, construction of scenes and interaction in that scene with objects of interest. The building of the synthetic environment was based on the domain of drone racing. The environment contained terrain, bodies of water, man-made structures and other objects which were added to the scene as a series of simple or complex objects. In Unity, three-dimensional models are constructed in the application or imported as third-party assets. Unity allows the placement and orientation of such objects. In general, rudimentary objects can be created in Unity through its interface but, more often some types of existing assets are incorporated. Such assets were created in three-dimensional modeling tools or obtained as pre-constructed models. It is also possible to leverage entire environments for existing projects such as AirSIM (Shah, Dey, Lovett & Kapoor, 2017). This work leveraged existing synthetic environment, when possible, if domain randomization and context-aware elements were applied within the synthetic environment. Such environments were beneficial not only in terms of time and labor savings but, more importantly, because many such pre-constructions are well formed and provide rich environment which generally provide a broad use in simulation, AI agent training, and games.

Existing environments were modified to fit a domain for drone racing where the drone can explore and train in a naturalistic environment. The racecourse was a predetermined ordered path through the synthetic environment which was populated with a series of gates. These were easy additions to existing environments that requires only the placement of gates in the environment. Racecourses were constructed from a simple to a sufficiently challenging drone racing course.   A simple environment was used during development and at least two more complex environments for the final training, testing, and evaluation. A dynamic changing drone racing course was instrumented to examine the effect on training and as a means for naturalistic domain randomization of the position of the gates. Additional objects were placed into the scene to serve as distractors. Distractors are placed into the scene randomly in terms of type, number, placement, and orientation. Their purpose was to distract the agent and provide negative reward when the agent responds to them or collides with them. This supported the opposing nature to the RL algorithm. Instead of seeking reward for a correct action by the agent, a penalty is exacted for a wrong action. Distractors may also vary in their ability to lead the agent away from the intended task. Objects such as the terrain, walls or structures indirectly serve as distractors which became a negative reward when a drone collides with them.

Synthetic environments allowed an unlimited number of training, testing, and validation scenarios to take place because they were modified more easily than real-world environments. The development of such synthetic environments can however still be time consuming and costly. Scoping out the number of environment and their complexities must be considered in context of the desired result. Synthetic environments have been created which provide a large range in terms of their complexity, detail, and number of objects within that environment. Unity has recently expanded its capabilities to scale up in these areas through Data Oriented Technology (DOTS) and Entity Component System (ECS). DOTS improves level of performance by taking advantage of multicore processing on a data-oriented stack. ECS incorporates DOTS where it consists of entities and data but organizing this in a structure that is data oriented versus entity oriented. These improvements and extensions to Unity permit additional scale for synthetic environments and brings them one step closer to real-world environments. Such features were not required for this work and not used to maintain the scope of the project. DOTS and ECS require an additional level of effort as well as the creation of a more complex environment. Such environments do exist such as Unity's Adam and MegaCity datasets. however, consideration must also be given to the effect on the training in terms of computation time and number of episodes required by the DRL algorithm. DOTS and ECS are mentioned to indicate potential future research and the impact that a more complex environment would have on this work's methodology and other related works which have utilized synthetic environments.

Naturalistic Domain Randomization

The original work of DR (Tobin, et al., 2017) introduced the concept of domain randomization to overcome the reality gap. It performs this by randomizing objects in a scene to be manipulated by a robotic agent. The randomization in this technique were performed through modifications to position, orientation, shape, size, color, textures, and lighting. These randomizations were often varied in such ways that they did not appear to be realistic. Often the textures where complete mismatches to the object. Sometimes objects like cars were floating in the air or turned sideways. Some object became multiple times their size or diminished by the same factor. The point for such extreme randomization was to make the object different enough to add variability, but still be recognizable. The assumption made by the DR work provided more than enough variance to the agent while training. The agent learned to accept variance to better adjust for items not seen during training. DR used neural networks which made it difficult to determine how the learning occurred other than looking to the result of the inference model's performance. The methodology of this research assumed that such extreme randomizations may not be necessary.

Randomization in this work's method kept all objects in a scene in a natural looking state and behaving as normal. The framework for this relied upon Unity-based simulations and games which have various techniques to modify an environment in a natural manner. Unity has built in capabilities to dynamically modify such things as camera properties, lighting and shading, weather conditions and so forth. Using weather as an example, a drone agent can run through the course while at various locations it may be sunny and bright and then runs into inclement weather or some form of visual impairment. Lighting, fog, smoke, haze, and water or rain are all integral to existing techniques in simulations and games built with Unity. Changing from one weather environment to the next while the drone is traversing along the drone racecourse was applied in this methodology. This introduces a variance into the environment, which is more naturalistic like changes in weather patterns. In DR, such an effect is more blunt and often very synthetic is its approach to demonstrate variance. These concepts explained here, apply beyond the theme of weather.

Differing approaches to DR based on photo-realistic simulation as seen in the GTA-based (Richter, Vineet & Roth, 2016; Johnson-Roberson et al., 2017; Richter, Hayder & Koltun, 2017) and Virtual KITTI-based (Gaidon, Wang, Cabon & Vig, 2016) simulators assumed the reality gap could be bridged by providing high fidelity environments which are modeled to look as much as possible as the real world. As previously stated, (Mayer et al., 2018), this technique did not achieve the desired effect rather it cost in terms of the labor and time to create such environments. Furthermore, the expense and availability of hardware in terms of graphics and computing speed to achieve this for real-time behavior provided further evidence against using such a methodology. This research was based upon synthetic training environments which are in a middle ground between simplistic and high-fidelity. Most development for simulations and games tend to meet at the middle ground based upon constraints mentioned. Environments found in these simulations and games were adapted for use in this work.

Another technique for domain randomization applied to deep drone racing (Loquercio et al., 2019) included the dynamic motion of drone racing course gates during training. This work examined the ability to transfer the training to real-world drones where the gates were moved during the real-world validation phase. The methodology for this work took on a similar dynamic modification of the drone racing course during training. This approach included the movement of the gates and course path. This was done in conjunction with other randomizing techniques such as changing the size, shape, and color of drone course objects such as the gates. This became an added part of domain randomization and was achieved in a realistic fashion to maintain the prescribe type of naturalistic domain randomization.

In this work there were several possible types of domain randomizations which occurred in a natural fashion in a synthetic environment. The value for each type of randomization was determined through experimentation which includes ablation studies to discern the effects of specific features. The list of features that were randomized follows:

- Synthetic environment lighting variations were performed by changing the degree of lighting in various scenes used during training, testing, and validation. No dynamic lighting variations were performed other that switching between scenes that held various levels of lighting from nighttime to mid-day.
- Shadows were not directly used as measures of variability but are inherent to the lighting changes. The shadows performed a key function in the visual representation of a scene through changing the lighting hitting was specific areas covered by the shadows. Visibility was varied through smoke, fog, steam, or other elements. In the work visibility was adjusted using fog, rain, and snow.
- Weather and seasons were varied. In this work, rain and snow were used as weather variations. Seasons were used which included autumn, spring/summer, and winter.

- Backgrounds which include sky color, clouds were also varied. These domain changes were generally associated with other environment changes such as lighting, seasons, and visibility
- Variation to man-made structures via scale, position, orientation and texture or coloring occurred in this work, however, represent an additional domain randomization for future work.
- Drone racecourse gates provide another key element in domain randomization. This work deployed the use of varying the drone course layout which included specific locations and number of gates used. The gates themselves were modified in size, shape and descriptive coloring or texturing.

Context-Aware Elements

The SDR technique (Prakash, A. et al., 2018) supports overcoming the suboptimal and high-variant policies produced by uniform sampling of environment parameters in the DR methodology. SDR utilized context-aware splines with control points and known objects which lie along the splines or close to them. This proved beneficial in overcoming the weaknesses of uniform sample randomization in DR. This work leveraged this methodology to keep the drone agent on the exploration track during training. Uniform randomization caused agents to get caught within a minimum solution. The context-aware spline with specific objects planted at control points provided information back to the agent relative to what that object means in the current scene. The RL-based nature of the agent's algorithm allows the context-aware elements to drive the agent onto continued exploration. This process helped solve the problem associated with the DR method.

The SDR+ technique used for context-aware elements was used in a different manner. This work used it to guide the agent during training to improve the speed and performance of the training. In the original SDR method, one of the main reasons for the spline-based approach was to augment procedural generation of the synthetic environment. The splines would provide paths along which given objects would be placed thus allowing procedural generation of these objects in the scene. This work's methodology utilized context-aware element as more than just splines for procedural generation of scene objects. They applied to surfaces, volumes, splines, and points. This permits a broader base in the type of elements used which could provide context to the drone agent. In synthetic environments such entities were represented as points, splines, surfaces, or volumes. This work did not use procedurally generated environments. The context-aware elements served just as information in the scene that are gleamed by the agent rather than to be used to determine where and how the objects were generated and placed. For this work it became a manual process which was suitable for the scale of the project. Future work can examine the inclusion of procedural generation of the environment.

Parameters set within Unity served as context-aware elements. The information provided by the elements became an embedded part of objects, terrain, and other items which make up the scene. Often intersection testing based on ray casting was the means to activate the context-aware element within an object which then were programmed to send information back to the agent. The agent receives this information to gain improved context of elements in the surround region. The algorithms utilized this information to assess if each element scanned would contain a positive or negative reward based on exploitation from previous experience or to explore new potential rewards for the objects scanned that contain context-aware elements. This gets incorporated into the RL-based exploration of the agent during training with the intent to provide guidance which helped to keep the agent on target. This is similar in nature to a plan, or a mission being used as a guide during execution. It is common that the actual outcome differs from the plan, nevertheless, it still serves its purpose in providing direction. In this work the guides or context-aware elements were less structured than would be seen in a plan. The concept still applies and becomes more relevant if complexity associated with objects in the scene is required.

Domain of Interest

A series of drone racing scenarios, which occur within the synthetic environment, provided use cases for experimentation. In the field of study for autonomous unmanned systems two common use cases are drones and self-driving cars, both of which are rich in terms of related research and development. Drones provide an ideal use case for this work because of similar works (Loquericio et al., 2019; Polvara et al., 2018) and simulation environments (Shah, Dey, Lovett & Kapoor, 2017; DRLSIM, 2020; DroneSim Pro, 2020; Zephyr-Sim, 2020; Liftoff, 2020; Velocidrone, 2020; FPV FreeRider, 2020; Real Drone Simulator, 2020). AirSim (Shah, Dey, Lovett & Kapoor, 2017), a Microsoft simulator used for drones and ground vehicles, was such an example. AirSIM provides a capability very similar to the synthetic environment and for the domain for drones being flown within the environment. It did lack the connection to AI based algorithms to drive drones within the environment and the fact that its purpose or usage for drones was open ended. It served as an example for this work to follow and was recently modified to work with Unity.

Several drone racing simulators also served as examples for the methodology for this work. Key among these was the deep drone racing work (Loquercio et al., 2019). The reason for this was because that work examined the use of drones in a synthetic environment with the intent generalize to actual drones in the real world. That work claimed success in achieving this result which was supported by the DR methodology. This work extended that work by using SDR+; however, it was not applied it to real-world drones.

The work leveraged these existing research works which focus on the domain of drone racing. There are key elements to drone racing that make it well suited as the domain of interest for this work. First, it is not a complicated nor a complex use case. Drone racing is for the most part a drone flying through a prescribe path and along that path are a series of gates it must pass through. Performance is based and accuracy of staying on the drone racing course path, passing successfully through the gates, not running into obstacles, the number of consecutive laps achieved successfully and the average speed for each lap and overall course time to completion. There are ample research works, games and simulations related to drone racing which serve as examples of the domain.

Metrics

This work evaluated the effectiveness the SDR+ technique using the domain of drone racing. From this, domain metrics were used to determine its effectiveness. The essential elements of drone racing were based on having a drone traverse through a course which includes a series of gates to pass through. This was to be performed as quickly and accurately as possible noting that there exist tradeoffs in achieving both. With consideration for the objectives of drone racing, the goals of this work, and the means to perform the following experimentation list of fundamental metrics:

- The average precision in terms of distance error of a drone from the center of the gate.
- The average precision in terms of distance error of a drone from its current path to that of the ground truth path of the drone racing course. Specific points of interest were taken at the midpoints between gates.
- The average speed of the drone traversing the course. This pertains to successful completion of each lap and then averaged over all the laps taken in the course. The number of crashes or failures to complete the course or lap were considered.

- The consecutive number of gates passed without missing a gate, hitting an obstacle. Limit of 4 laps and number of completed laps in that set including consecutive laps was the set measure of performance.

This list is associated with fundamental tasks the drone agent must complete. This work examines variations in the drone racing course and its environment for training, testing, and validation of the learning process by the agent. These fundamental tasks and associated metrics get repeated but under various circumstances and are measured for all these variations. Such circumstances which provide variations within the synthetic environment while gathering metrics are listed below:

- Modification of gate location provides a metric to determine the amount of gate displacement which was measured based percentage completion of the tasks of gate passage, to lap completions, to course completions. This gate displacement provides another DR based metric.
- SDR+ techniques are only utilized in the synthetic training environments. The testing environments were equivalent to the training environments minus the application of SDR techniques. This allows a baseline and a set of gathered metrics to be used for comparison to runs on the validation environment which is different from the training or testing environment. This provides the actual metric of interest regarding the measure of generalization.

Ablation studies provided additional metrics with regards to both naturalistic domain randomizations and context-aware elements. Comparison of final generalization metrics between non-ablation and ablation studies were collected as a measure of the influence of the given item that was extracted in the ablation study. This also provides the ability to examine the influence of a given item on the performance of training for acceptance of variability.

Real-world training, tests and validations are beyond the scope of this research work. The question does arise concerning the determination of the level of generalization when no comparisons were performed using real-world drones. Even though such a comparison to real-world implementation and execution is the ideal, there are many issues related with a real-world application that are beyond the scope of this work. There was a degree of difficulty in matching the simulated or synthetic environment to one in the real-world. This comes down to both the creation of a drone racing course as well as the transfer from the synthetic based drone to the real drone in terms of behavior, physics, unknown differences and hard to model attributes. This also applied to the transfer of the inference model from the synthetic to real drone agent. The methodology used for this work assumes that a first order approximation to the degree of generalization can still be obtained totally within a synthetic environment experiment. This was achieved by utilizing a completely different synthetic environment and drone racecourse in the evaluation phase from those used for training and testing. This also assumes that the level of variation provided in just the difference of the validation synthetic environment still provided indications of how well generalization can be being achieved.

Integrated Methodology

The previous subsections provided pieces of the integrated methodology used in this research work. Each subsection plays a critical part within the methodology. It is noted that the next section contains information relative to instrumentation, ablation studies and baseline comparisons which were an integral part of the overall methodology. These deal with the instrumentation of the framework to capture specific data. In summary, the methodology is based on an experimental framework most of which was founded on Unity. A module, ML-Agents, provides the means to integrate the PPO-Based DRL algorithm to provide the intelligence to the agent which in this case is the simulated drone. Unity itself provided the mechanisms to construct, import and utilize elements which make up the synthetic environment.

The Unity application and its use of a C# scripting tool allowed elements in the synthetic environment to be controlled, examined, and extended which allowed for the integration of both naturalistic domain randomization and context-aware objects. A domain of drone racing provided the scenario and allowed for the specification of metrics which were utilized to validate the methodology. This integrated methodology is illustrated in Figure 1.

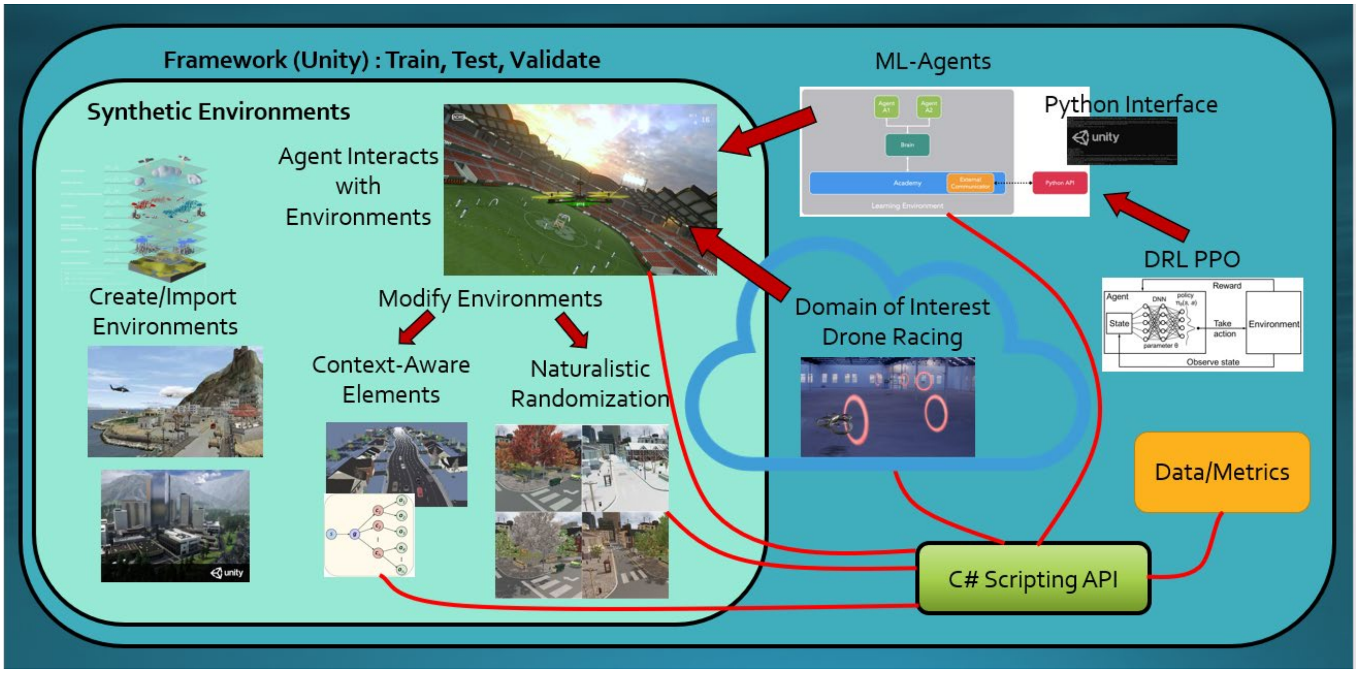


Figure 1: Methodology overview

Validation and Instrument Development

Instrumentation

This work required analysis and means to qualitatively and quantitatively measure the degree of generalization that occurs due to training with SDR+. Benchmarks, metrics, and experimental protocols provided an important part in determining what to measure and how the framework was to be instrumented to obtain data relevant for analysis. A previous work in using DR for simulated robotics learning (Dai et al., 2020) provided great insights into analytics and metrics to understanding and measure the effects of DR. These analytics and metrics, however, are more specific to a robotics use case and less useful for this study. Another work (Packer et al., 2018) demonstrated a benchmark and experimental protocol with an empirical study for use in examining DRL algorithms. This work included evaluation on DRL algorithms of the PPO type. This work divided generalization into two regimes: interpolation and extrapolation. Interpolation was defined as how generalization is achieved for use in environments which are in line with aspects of the training environment. Extrapolation was where generalization was achieved when parameters of the testing or validation environments differ from those seen in training. Although this was relative to this research, the emphasis was not on the study of the algorithm rather how external methods can potentially provide better generalization without modification to the algorithm. A work on deep drone racing (Loquercio et al., 2019) provided additional information on metrics and instrumentation required for a domain directly applied to this work. The three works mentioned here, even though there was not a direct correlation, they provide a structure for the metrics and instrumentation that were used for this experimental framework. With this inherited set of metrics and instrumentation concepts, the external validity of instrumentation extends in part to the internal validity of instrumenting the experimental framework.

The instrument implementation was made possible by the Unity API for C# scripting. Developing the code to control, sample and record data during training, testing and evaluation was achieved in similar fashion to how it is done for naturalistic randomization and embedding context-aware elements within the Unity application. C# scripts for instrumentation were integrated with objects in the scene. For naturalistic randomization and context-aware elements, the object integration allowed for control or manipulation of the object. In the case of instrumentation, it was used to collect data and performance metrics.   An example illustrates the point. For a drone racing gate, a planar object embedded in the gate was instrumented for object collision. When a drone successfully passes through the gate and this planar object, an event is fired which activates a C# script which can then record the event. The code is activated by this collision event as the drone passes successfully through the gate. There can exist additional criteria such as a minor collision against the gate during the passage which may negate the reward but, the reward is recorded by the script which was then revisited for results and analysis. For cases when the drone completely missed the gate, this missed gate was detected through an algorithm in the C# code which monitors previous and future sensed gates and can determine a missed gate based on the sequence of gates. Most instrumentation followed this pattern.

Instrumentation through Unity provided a well-connected set of instruments for elements within the synthetic environment. Since the experiments were all virtual in nature no real-world influences have impact upon the instrumentation. Such conditions might be weather, power outages, faulty drone sensor and so on. The chances of not providing valid data were greatly reduced.

One portion of the instrumentation not covered by Unity based methods is the inclusion of the DRL algorithm via the Python interface provided by ML-Agents. The performance of the algorithm is output by ML-Agents much of which is contained within the console running ML-Agents. Loss or interruption of this connection can lose data and potentially invalidate the means of instrumentation and the trained model. Since experiments are run virtually, they were easily repeated with minimal impact and allowed other attempts to provide valid data through this instrumentation. Tensorboard, an opensource based dashboard, provides a presentation of algorithmic statistics on the various measures of performance of the DRL algorithm. This is an instrument well suited and robust for this study. Tensorboard can plot cumulative reward, episodic length, policy loss and value loss, or other configured output during training. It was used as the primary source of data coming out of ML-Agents.

Ablation Studies

Ablations studies are important to the instrumentation and validation of these experiments. These studies were used to remove a component used within the methodology with the intent to understand its effect on the overall performance through determining how much degradation occurs because was not included as part of the solution. This primarily provides insight to the contribution of a given element of the methodology. For this work the two main considerations were the components that make up domain randomization and the context-aware elements. Experimentation occurred where one of either of these components was removed and then the AP of the new results compared to when all the components are present. An example of potential ablated components are as follows: lighting, camera viewpoint, shading, positioning, size, shape, color, and texture. In the DR work their ablation study noted that lighting was among the most prevalent sub-components with others having minimal effect. The framework instrumentation being used with ablation studies did not require any modifications other than noting the removal the ablated component and providing the implementation that allowed this without generating errors.

Validation and Reliability

Obtaining valid and reliable results was important for any experimental based research. Several topics regarding validation were previously discussed such as leveraging and comparing instrumentation of other research works, utilizing a total synthetic approach to avoid any real-world impacts and, reliance and validation through tools such as Unity, ML-Agents and TensorFlow with TensorBoard. Additional issues with validity and reliability comes with the development of software code. Specifically, the use of the Unity C# scripting API. Software bugs and improper coding can play havoc on getting the software to function correctly. If such an issue is introduced and goes undetected it can provide results good or bad which does not represent the intended application. This research work leveraged commercial and opensource software which has a greater chance of providing valid working code and which is generally more reliable than starting from scratch especially if the commercial or opensource software provided documentation for known errors or issues. The exception to this was with the C# scripting code which glues together much of the experimental framework. This C# scripting code was reviewed and tested for correct and repeatable results for each script implemented both at the functional level of the script and the integration and interaction with other scripts or components. There remain potential issues due to human errors being introduced but not discovered. Commercial and opensource tools can also perform code checking and functional tests~~.~~

Reliability is an issue with AI based applications. For certain types of applications, the AI based results are generally not repeatable and may not return the exact same results each time. Generally, the algorithm was considered reliable when repeated performance of the inferred model was within a small or designated amount of variance. The AP values may not match each time but if they are close in achieving similar values and acceptable tolerance, they were considered reliable. For this work, an acceptable tolerance was defined given output from several runs of the same experimental setup. Results became acceptable if within the tolerance, otherwise further investigation was required into the code for reasons for deviation outside the tolerance measures. This tolerance was difficult to determine ahead of time and generally comes once the framework has been constructed and run enough times to analyze the results for validity and reliability. In most cases validity and reliability were specified in commercial and opensource tools. Knowing the advantages and shortcomings of these tools helps to determine where errors and non-repeatable processes may occur. For specific code developed for this work, this was not the case. Sufficient testing and code reviews must be performed to assure some measure of validity and reliability.

Proposed Sample

This research work did not require any human subjects nor any participants. This work was strictly an experiment to be run to collect data and analyze the results of the applied methodology. For purposes of this research work the idea of a proposed sample was redefined to the quantity of certain components of the methodology used in the experimentation. The proposed samples requirement was required to be is sufficient for the experiment, sufficient to obtain the needed data, and sufficient to provide measurable and meaningful results. The proposed samples as components of the experimentation have been broken down into several categories which follow.

Synthetic Training Environments

The training environment was critical to the success of the experimentation to determine the degree of generalization to be achieved by the methodology. This was the environment which contains the components to achieve the goal which allowed the agent to learn to generalize from exploration of the synthetic training environment. Success for this work was only achieved within the use of synthetic environments. This work considered two synthetic training environments. Training took place but, at different levels of complexity. A simple synthetic training environment served the purpose to iterate on and adjust the experimental setup to ensure the training worked as designed. It was also determined if the instrumentation was adequate for recording the required data. The simple training environment was the easiest to implement and to understand therefore, served as the iterative playground for refining the methodology. A more complex environment had more detail to the environment and objects with more variations. The level complexity was determined through results obtain with the less complex environment. The more complex experimental environment was used to determine the number of iterations required by the algorithm to arrive at the best performance as the degree of complexity of the environment increases when compared to the simpler environment. The goal in developing the more complex environment was to achieve a level of richness in the synthetic training environment sufficient to match the goal of improved generalization. The relative nature of the complexity of the environment was subjective. A synthetic environment which was not too simple nor too complicated was selected and design via results from the development for the experiments for the simple case. Although exact quantitative measure cannot be determined for the level of complexity given to such an environment, a standard was set and a relative measure of increased or decreased complexity was provided based on this standard. For synthetic environments there were a few quantitative measurements that are possible such as total number of polygons, total number of objects in the scene and resolution used.

Synthetic Test Environments

Some of the synthetic test environments were identical to the synthetic training environments except for the controls used in the methodology to improve generalization. This establishes a baseline to see how well the trained model performs in the same environment it was trained in. The other synthetic environment used served as a test of the effect of domain randomization. The assumption for these test environments was that they served as baseline environments to determine results.

Synthetic Validation Environments

The synthetic validation environments were be used to determine the level of generalization obtained from the training environments. These environments did match up with the two levels of environmental complexity applied to the synthetic training environments. This allowed the same iterative development of a simple validation environment towards a more complex validation environment. The complexity of the synthetic validation environment was more complex and different from the training and testing synthetic environments as a requirement to validate the degree of generalization achievable. An interesting question to examine in consideration of the simple and more complex environments was if a model trained in a low complex environment can perform as well in as one trained in a more complex environment? Does the reverse examination have similar performance measures? This helped to determine the level of training required and if generalization learned applied between dissimilar training and validation environments.

There were differences between validation environment and those of the training and testing environments. The objects, terrain and associated environmental effects are similar; however, the methodology of randomization and context-aware elements are not applied to the synthetic testing and validation environment. Validation environments must differ from the training and test environments. The reasoning behind this is to avoid the condition that trained models for DRL based algorithms often perform well within the same or vary similar environment such as the testing environment. Another reason was that a naturalistic environment was required. The training environment randomized and varied the environment during training. For the validation environment only, naturalistic changes applied. This application may vary from run to run. An example might be the time of day, weather or other environmental factors which make the validation environment different from the training and testing environments. This required that several validation experiments need to be run to allow for differences in support of validating generalization. Ablation can also be performed within the validation environment to determine the effect a given change in the environment impact the generalization.

Drone Racing Courses and Scenarios

Various drone racing courses were provided for cases of training, testing, and validation. Each course provided sufficient environmental randomization to meet the goals of the agent to learn variability in training courses, check the inferred models in test courses, and validating the agent's ability to adjust to variability in validation courses. Scenarios were be created to define the settings and arrangements for each given run of the drone with multiple laps. The number of laps for validation runs were specified where the number for training was based upon the maximization of performance achievable during training within a reasonable timeframe. Course complexity correspond to the complexity of the environment in which it is placed. Complexity of the course was also be defined in scenarios by:

- Total number of gates
- Distances between gates
- Orientation and placement of gates
- Size and shape of the gates
- Color and/or texture of the gates
- Number, size, placement and type of obstacles and distraction objects
- Background and surrounds
- Environmental effects

Scenarios were constructed to specify the parameters for each training run or continuation of a training run as a configuration guide to modify the synthetic environment. Scenario files were used to specify randomizations, context-aware elements, and instrumentation applied to the environment and course. Typically, context-aware elements and instrumentation remained paired with the same objects in the environment and did not vary from scenario to scenario. Randomizations varied and were outlined within the scenario file. Scenarios defined the goals which equate to reward, or punishment values assigned for DRL based training. Scenarios also defined the metrics to be obtained from the embedded instrumentation.

Randomizations

Natural randomizations were changes in the environment which are naturally occurring such as time of day. These were implemented in the scenario as time and event triggers. These triggers were applied to objects or parameters within the synthetic environment. The scenario supported the way they got modified. Randomizations were defined to occur at configured number of steps in the training process. At each of these specified points in the training, ML-Agents paused to write out data. It was at this point in the training process where variations into the synthetic scene were made. The follow changes were made during certain of the training:

- Environment lighting with implied shading effects
- Environment obscuration via fog, rain, and snow
- Sky color and cloud density through skyboxes which are associated with weather, seasons, and obscurations.
- Foliage and other object in the scene color changes. This included vegetation coloration and coverage by snow
- Drone racecourse gate modifications of location, number, size, shape, and coloring

Experimentation and analysis provided indicators on how randomizations occurred to achieve an optimal training effect. Parameters to vary for the randomizations were the frequencies, durations, and locations for each of the randomized items. Based upon initial analysis, the frequency, duration, and randomized locations were adjusted to optimize its effect upon generalization.

Context-Aware Elements

The advantage of synthetic environments is the ability to have any part of the environment pre-defined with annotation. This predefinition must be built into the environment and context-aware elements to provide the guidance of agents during training. This wa considered a form of annotation to the objects in the scene which in turn were discovered by the agent to help improve its learning. A value of reward or punishment was assigned to any given object through context-aware elements and the interaction an agent had with those elements. For synthetic environments practically all objects became context aware. In the SDR method, splines were used with control points to act as locations to insert objects with defined context. The method of this work extended this concept to volumes and surfaces where larger objects were easier encapsulated and annotated. Analysis of results determined if the methodology regarding the number and type of context-aware elements provided effective results or impeded the agent. As a minimum, each gate along the drone racing course was a context-aware element along with intermediary points which define the course path and direction. Distractors were also included as context-aware elements. The exact number and type of context-aware elements was determined through experimentation and analysis of the results during the development phase.

Data Analysis

Analysis for Research Questions

The first research question - what generalization improvements are realized when using a SDR+ - was the primary focus of this work. This work's goal was not to improve upon the algorithm itself, rather it was to leverage the use of synthetic training environments and perform research experiments to determine how modification to the training environment may allow the DRL algorithm to better generalize. The data provided and associated with the performance of generalization of learning by the algorithm was based on average precision measurement of the drone agent performance to determine a baseline and, a test environment which was created from the training environment. The test environment was void of the methodology to provided randomization and use of context-aware elements. Runs of the test environment provided various naturalized environment settings without having them be randomized. This data was analyzed and holds to the assumption that a similar average precision for the drone agent was achieved in both the training and test environments. This allowed the test environment to be utilized as a baseline for comparison of the validation environments using the inferred model from the training environment. The metrics from the validation environment experimentation provided measures of average precision in an unseen environment from the viewpoint of the training environment. Typically, such results are quite degraded when applied to environments which have such dissimilarities without any means to bridge the reality gap. Analysis of average precision, which if found within acceptable range from the test environment results, provides indications of improved generalization. The indications for generalization improvement from this baseline was better understood using ablation and sensitivity studies. What becomes more apparent here, were the ablation studies and sensitivity analysis as it applied to the specific and unique features of this work's methodology. Ablation and sensitivity studies pointed out whether these specific elements of the techniques had any impact on the average precision and how relative that impact was to other common factors observed in other research works (Tobin et al., 2017; Loquerico et al., 2019; Prakash et al., 2018; Tremblay et al., 2018).

Analysis for Hypotheses

In similar form to the research questions, both hypotheses of this work were tested in the experimental framework and answered through ablation and sensitivity studies. Naturalistic DR provided an easier implementation. The qualitative measure for naturalistic randomization being an easier implementation was in part because it leverages from serious game frameworks. This method took advantage of such frameworks, specifically built-in naturalistic features of the Unity application. Implementation for this was built into Unity and only requires configuration according to the defined scenarios.

SDR+ with context-aware elements did support improved training through guidance for further exploration. The ablation and sensitivity studies provided a quantitative measure with regards to average precision effect and influence on the results.

Analysis for Ablation and Sensitivity Studies

This work's methodology was built into the series of scenarios a means to support ablation studies. Ablation studies provided the means where a given parameter was excluded to see its effects on the results. Sensitivity to the amount or type of a parameter utilized in the data analysis also provided metrics to indicate thresholds of performance. Related research works in the areas of related studies (Tobin et al., 2017; Loquerico et al., 2019; Prakash et al., 2018; Tremblay et al., 2018) have performed ablation studies for similar cases.

The work on DR (Tobin et al., 2017) used ablation studies to examine the sensitivity of both the number of training samples used and for the number of unique textures used. In both examples the sensitivity to the average error of distance to placement of the robotic grasper was measured. The average error converged after about 5000 images were used in the training. Sensitivity towards the same distance error continued to improve even beyond 100,000 textures used. Random noise was also examined but was least sensitive to improvement of the average distance error. The DR ablations studies further indicated that texture randomization was seen as more important than in randomizing the position of objects in the scene.

The work on Deep Drone Racing (Loquerico et al., 2019) provided sensitivity on the percentage of tasks completed to maximum speed, number of successful laps and to amount of relative drone gate movement. Low drone speeds up to about 6 meters per second performed 100% of their tasks. Achieving successful number of laps was near 100% up to about 3 laps and only reduced below 90% of lap completions for up to 5 laps. This performance quickly dropped to a rate of 30% task performance when the speed was doubled. This work also examined the effect of moving the gates during the evaluation process. Numbers fell to just under 90% of task completion when the relative movement was doubled. It fell further to 50% task completion when the relative movement was tripled. Regarding ablation studies on factors of background, object shape and the amount of illumination, the work determined that all three factors contributed to the improved task completion. Task completion was also compared to drone speed. The inclusion of all three factors provided the best results with speeds up to 10 meters per second still achieving a 90% task completion but quickly falling thereafter. Task completion when other elements were removed provided values between 50% and 40% at 4 meters per second continuing to drop between 30% and 20% at the 10 meters per second mark. One interesting result was the fact when no background was used, the drone would fail to complete even a single pass through a gate. The deep drone racing work also measured the sensitivity of re-running planning algorithms at different distances from the target. The planning distance was a measure of how far out from a gate a plan needs be formulated to correct course. Their results provided a large range which performed well but indicated that for both short and long planning lengths the performance dropped significantly. Short planning length causes aggressive drone behaviors while long planning tended to degrade the agility of the drone. Another analysis performed was the successful passes of gates based on the occlusion of the gate. When the gate was occluded up to 45%, a 100% of gate passes were completed. The successful gate passes dropped drastically to about 10% when 80% of the gate was occluded.

The SDR research work (Prakash et al., 2018) preceded its work with examination of DR. They performed an ablation study on their use of DR and determined that lighting proved to be the most important factor. Upon extending the DR work to SDR, their ablation studies determined that factors of context, saturation and contrast exceeded the importance of randomizing lighting. Their ablation studies examined full SDR, SDR without context, SDR without scene randomization, SDR with high contrast, SDR without random saturation, SDR without random lighting and SDR without multiple postures. As noted, contrast and random saturation had the highest effects in their ablation study but, the other factors were not too far off and provided a much higher resulting value using the full SDR technique. This work assumes that context is an important feature of their method and saturation and contrast were determined as contributing factor over other lighting effects.

The final work (Tremblay et al., 2018) looked at synthetic training of deep neural networks for object detection using DR to bridge the reality gap. Their ablation study looked at a fixed light on distractors, no textures, use of 4K textures, no augmentation (contrast, brightness, cropping, flipping, resizing and noise), no distractors and no light augmentation. Average precision did not change drastically in any one category. All factors seemed to contribute to the generalization as seen through just a small amount of degradation in average precision when the factor was not present in the ablation study. Of interest from the study was the increased performance due to fixed light on distractor objects versus providing some randomized lighting. This provides indications in the need to determine appropriate randomization added in the scene.

Results Format

This research work followed the applicable formatting of results found in the literature research of existing works. The results format commonly used was average precision. Average precision, or sometimes called mean average precision, is used to average out all possible thresholds versus a particular threshold. Average precision is also quantitatively related to the area under a precision-recall curve and its value is used to summarize the precision-recall curve. Essentially, precision-recall attempts to define what portion of the actual positives were identified correctly. The average precision therefore defines a summarized view and is often used in the comparison of how well models are making predictions without the reference to a specific threshold. This fits the nature of DRL based agents with the exploration of their environment.

In the literature, especially with regards to domain randomization techniques, the use of ablation studies provided a useful insight into the various elements that get randomized. Ablation studies allow separating out one specific randomized element to examine the effect when that element is not present. The results of a given ablation study can then be compared to results when all the elements are present to determine the degree of influence of that element. In the extreme cases the affects can range from negligible where the inclusion of that factor may not matter, to substantial enough to be considered a key contributing factor.

The work on DR provided an interesting but basic metric in the study of robotics. This was the measure of error in striving to obtain an exact position of the robotic arm. For the case of this study the measurement of distance off the proposed target is of interest. In this case the average precision was used in determination how far off the agent drone is from flying through each gate and its motion distances from the ground truth path along the drone racing course. The DR work utilized heat maps to visualize these distance errors and to examine the sensitivity in the error. This was similarly applied in a drone racing course given types, shapes and positioning of gates and the path the drone should follow. The use of the error distance is useful in this work and where possible heat maps may provide insight especially when combined with drone speed values.

Tasks performed by the drones are also key factors in formatting the results and corresponds well with the selected domain. These tasks relate to repeatable actions such as flying through the gates successfully, completing laps without incident, and the speeds which allow successful transit through the course. The other factor of interest was the distance offset from the ground truth location of gates and the drone course path. Each of these has been illustrated in the literature reviewed and provides a model for formatting and presentation of the results for this work.

The work on Deep Drone Racing evaluated target trajectory to improve path correction performance which provided another useful metric for results. A useful metric in evaluating the method was the outcome of specific tasks relative to the agent's ability to perform to specified drone racing factors. A task-based metric was formulated based on percentage of task completion. These same tasks and presentation of their results apply directly to this work. The first task was the measurement of the drone speed through the course and the point where performance in completion of the task degrades as the drone speed increases. This is measure of the drone successfully navigating through the course. For a given race there may be specified number of laps to complete. If all laps for the specified race are completed, then the task is complete. Crashes, misses, or failure to complete the track reduces task completion. A percentage of completion is the metric to determine. The second task-completion measure is with regards to the number of successful and consecutive individual laps being completed. A maximum limit of laps was specified, and task completion utilizes similar factors such to the previous task which includes a series of runs which vary in speed through the course. The last task-completion deals with the injection of variability into a scene. The motion of gates in the drone racing course are dynamic. Here the same metric for distance from center on hitting the targeted gate as well as the distance and orientation which the gates move during each lap are formulated as results. This is measured and the distance error from target, including the total number of gates successfully passed through consecutively. These results are also based on average precision during task completion.

Resource Requirements

Hardware and Software Requirements

This work utilized a powerful desktop computer with a newer multi-core processor, a minimum of 16GB of RAM, SSD hard drive with at least 500GB of disk space and a high-end graphics card such as an NVIDIA GTX 1080 or GTX 2080 Ti. The scope of the project allowed this work to be developed the intended framework, all running on the same machine. The operating system was Windows 10. A key element for the development was Unity, a 2D and 3D gaming engine. ML-Agents, an add-on bundle to Unity, provides an integrated AI component which includes the use of DRLs. Unity uses a scripting component to build code that connects to the 2D and 3D environment building tools. This scripting component uses C# programming language and was connected to Microsoft Visual Studio for code development and compilation. The Unity Asset Store provides several assets and third-party tools which were utilized to build the environment but also to concentrate more on the experiment versus the development of various components. Code development included the means to save data to disk from which other external tools for data analysis and presentation was used.

Simulation Environment and Networks

This work incorporated simulation of entities and events from within the toolset of Unity and ML-Agents with the potential to use an integrated simulation tool such as AirSim, drone racing simulators or other suitable simulation environments. The simulator provided an existing synthetic environment and built-in drone dynamics which was immediately leveraged. Unity provides the foundation for the synthetic environment which also allows application of DR and SDR and connection to DRL implementations through ML-Agents.

No network communications should be required but depends upon the simulation tool. Running both the simulation, DRL processes and the synthetic environment can run on the same machine. Should such circumstances require use of a network connection there are several known third-party networking tools for Unity such as Photon Unity Networking (PUN, 2020).

Data Collection, Metrics and Analysis

The framework for this research incorporated the collection of data for each of the scenarios that were run during experimentation as described in the approach section. Data and metrics were captured through the experimental framework. Data collected was run through external data analysis tools. Data collected was correlated to the three types of conditions run and the metrics as outlined in the approach of this work for the determination of the validation of generalization techniques. The existing hardware and commonly obtained software were adequate for this work other than any items required to be built into the framework.

Summary

The methodology of this research work was founded on methodologies from other research works with the end goal of producing a methodology that added to the body of research. This research leveraged related research and examined how DRL, or other AI algorithms were generalized. More specifically the methodology leveraged two specific research works, DR (Tobin et al., 2017) and SDR (Prakash et al., 2018) to gain this generalization improvement through modification of the training environment versus modification of the DRL algorithm itself.

An outline to the research methodology was formulated based on a variety of components which when put together provided the full capability to experiment, analyze and determined what this method achieved. The first component consisted of the experimental framework built primarily upon Unity and its ability to incorporate synthetic environments and modules such as ML-Agents. ML-Agents allowed a built-in agent to be created which can then explore the designed and developed synthetic environments. The framework included development capability to construct the agent using a PPO-based DRL algorithm and run in several synthetic environment for learning via exploration and exploitation. DR was embedded within the development API and built in Unity capabilities. DR was then implemented in a naturalistic manner during training session to achieve the variability of the environment to be learned by the agent. This included context-aware elements to support the agent's exploration as a guide to environmental context. A domain of drone racing was utilized to set the scenario and constraints of the agent to learn and then tested and validated with that experimental framework.

The Unity API allowed for the development of embedded environmental sensors to collect data during training, testing, and validation. This instrumentation was essential to gather data based upon a set of metrics particular to the drone racing domain and the particulars of the methodology for context-aware elements and naturalistic domain randomization. This supports the concepts of validation and reliability of the data being collected. Ablation and sensitivity studies were integrated into the framework to determine the influence of certain factors of the methodology.

Experiment samples were defined. The results from running the experimental framework required a synthetic training environment where the agent learns. Once it has performed the training, it needed to be tested in a similar environment to the training environment except for the applied methods for the agent to learn variability. This excluded the naturalistic domain randomization and the embedded context-aware elements; however, elements of the environment were modified in similar fashion to the results of naturalistic domain randomization for a given type of item. The instrumentation and data gathering capabilities were implemented in all synthetic environments so that the full set of data could be collected and analyzed. The validation environments were required were sufficiently different from the training and testing environments such that it imposed variations in contrast to the training and testing environment that was then observed by the agent. This environment specifically introduced variability to the agent. The experimental samples included a variety of scenarios which accounted for the variety of experiments which cover ablation and sensitivity studies. The randomizations incorporated as experimental samples were defined along with the context-aware elements.

The methodology outlined the way the data analysis occurred and how the results were formatted. This was based on the proposed research questions as well as the hypotheses to be tested. Average precision was determined as a primary result. Ablation and sensitivity studies provided value in determining the relative importance of certain factors of the methodology. This applied to naturalistic domain randomizations and context-aware elements. These were set against tasks associated with the drone racing domain. Offset distances to targeted gates and ground truth drone racing course provided indicators of the utility of the methodology. Additionally, important were the precision in performing drone tasks of successfully completing gates, successfully completing laps without incident The speed at which it can traverse the course while maintaining high values of average precision during the tasks was also measured and increased in value to determine a limit on speed. The results of the synthetic validation environment were essential in the comparison to the results from the synthetic test environment. The assumption of this work was that the comparison between these experiments would provide an indication of potential increase in generalization achieved via the methodology presented.

The resources required to construct the experimental framework and run the various experiments in differing synthetic environments under different conditions were outlined. Both the hardware and software requirements were reviewed. No specialized equipment other than a high-quality computer platform with the required software was necessary. The resources to construct synthetic environments and provide the means to collect and analyze data for specified metrics was covered. Existing resources have been deemed adequate to perform the research of this work.

Chapter 4

Results

Introduction

The preliminary research focused on leveraging and modifying existing research methods to overcome the reality gap when synthetic environments are used to train agents. The result of this research presents a select research focus from among approaches to improve upon generalization while overcoming the reality gap. The results obtained were leveraged by applying modified techniques of structured domain randomization (Prakash, et al., 2018). The results were used to determine the effects of naturalistic domain randomization while using context-aware elements to support improved training. This was performed in an experimental framework which consisted of an adaptable synthetic environment. This chapter presented key points related to the results.

The initial data collection and results analysis were performed on a simplified agent with a simple goal of finding, approaching and then colliding with stationary target object. This simplified approach was discussed in the methodology of Chapter 3. The concept was to determine the configuration setting to be used in the DRL PPO-based agent. An approach was taken to study the effects of the various configuration parameters and then select the best and most appropriate set of parameters. The configuration deemed best was then used for the more complex drone racecourse experiments. These results along with the analysis in the selection of the configuration parameters was illustrated in the first subsection of this chapter entitled Data Collection and Analysis.

Attention was then turned to the primary focus of the data collected from the drone agent experiments along with analysis of that data. The data collection and analysis for the drone agent experiments expanded due to issues and findings, resulting in additional subsections in the Data Collection and Analysis section. These subsections describe the data and data analysis for a baseline drone agent and several other agents trained at various levels of naturalistic domain randomization. Ablations studies required the training of several drone agents in various combinations of naturalistic domain randomization. One agent contains the complete set of domain randomizations while a set of agents provided a subset or single domain randomization. This permitted the ablation study to take place and analyze the significance of each component in the training of the learning agents. Analysis and findings are included in each of these subsections. The final section presents a summary of the data results, analysis, and findings.

The results from metrics collected in the experimentation framework supported the means to evaluate the level of effectiveness using a modified structured domain randomization in the agent learning process. The baseline agent data established metrics of comparison to the other learning agents with some type and degree of SDR+ applied. This comparison of results between the baseline agent and various SDR+ agents provided the indicators in determining the value of the approach of this work.

Data Collection and Analysis

The following sections provide the details of the data collection process, the metrics collected, and the analysis of that data. Additional experimentation occurred along with the data collection and analysis based on the output obtained and analysis of that data. Each subsections provides details why this occurred. A description of the methods used to collect the data in the framework are described and integrated into tables and graphs to presents the data in a form ready for analysis. Initially, simplified training tests were performed to determine the best selection of DRL PPO-based parameters. With these configurations applied, the primary experimentation began with data collected on the training of the agents. This was followed by tests to evaluate the quality of the inference models running in the same environment in which they were trained. This was followed by evaluations of the inference models each from the various experiments. These experiments included how the agent performed within the environment in which it was trained plus its performance in the environment with variations added to that synthetic training environments. This analysis provided an evaluation of how well the specific learning agent adapted to variations in the environment. Inference models obtained for agents learning under the SDR+ training environments were then compared to the baseline agent finding.

The following subsections, with the data collected and analyzed, provide a glimpse into the workflow performed using the experimental framework. The workflow demonstrated the determination of the DRL PPO-based learning algorithm configuration parameters via a simplified version of the experimental framework. The data from the training, testing and evaluation of the baseline agent is then followed in similar fashion by the various agents trained, tested, and evaluated that had specific elements of the SDR+ technique applied. Each subsection that follows provided the logical sequence to the experimental workflow. It is noted that some additional experimentation was required due to some findings in the data collection and analysis. This followed a logical progression of the workflow while making adjusting where required either due to issues with the techniques used or computing logic errors.

Selection of DRL PPO-Based Configuration Parameters

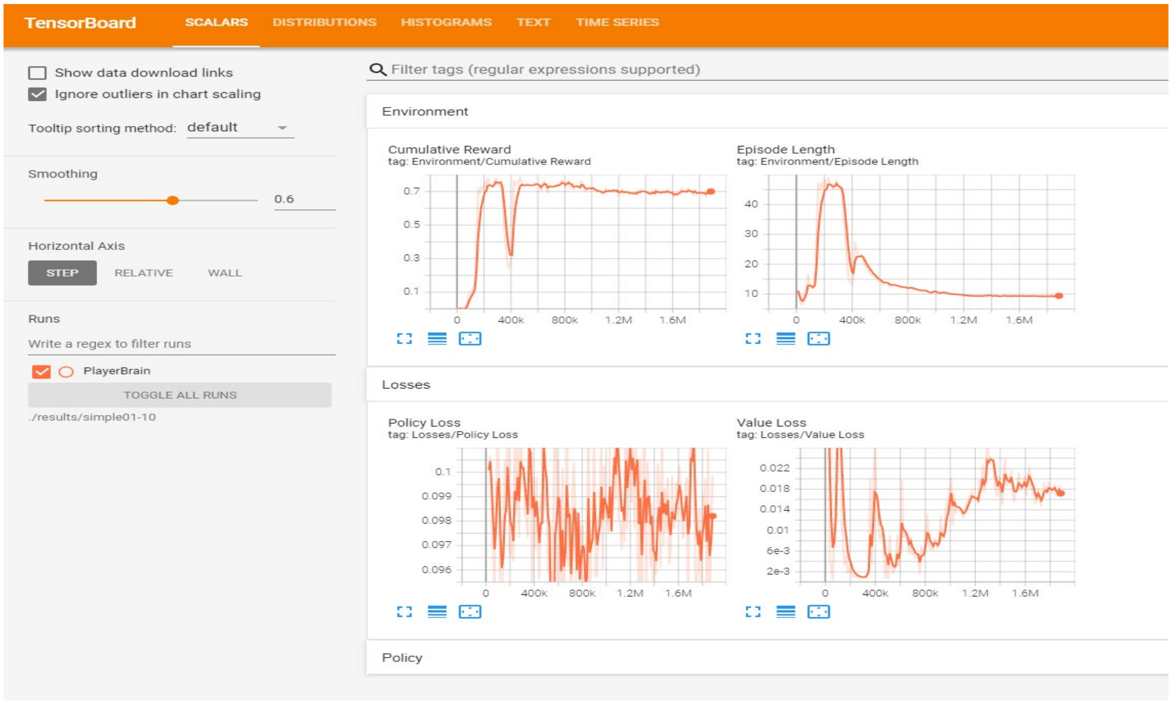
This section covers the data gathered and analyzed for the selection of DRL PPO-based learning algorithm configurations. A description of these tests was covered in Chapter 3. The following table summarizes each of these tests along with key results.

TEST	CONFIGURATION	REWARD	STEPS	COMMENTS
01	Env. Observations (O)	0.7	400 K	Reward ended up below of 1.0
07	Ray Cast Sensor (R)	0.14	2,000 K	Poor performance over training
08	Visual Sensor (V_S)	0.0	2,000 K	Very poor training performance
03	O + R	1.0	260 K	Quick rise to consistent reward
04	O + V_S	1.0	216 K	Quickest rise to consistent reward
06	R + V-S	0.53	2,000 K	Poor to mediocre performance
05	R + O + V-S	1.0	480 K	Good smooth/constant reward value
09	R + O + V-CNN	1.0	540 K	Quick rise then dropped to 0 reward
10	R + O + V-Resnet	1.0	530 K	Quick rise then dropped to 0.8
11	R + O + V-Make3	1.0	540 K	Longer rise but reached reward

Table 1: DRL PPO-Base Configuration Parameter Testing

The analysis of the results from these tests supported the selection of the best DRL PPO-base algorithm configuration. Configuration settings for Test 05 provided the best results from the set of simple agent tests. The graphs that follow along with comments on the analysis of each test, provided evidence to support the use of all three configurations of environmental observations, ray cast sensor and a visual sensor set to the simple neural net configuration. The data from each test was analyzed and commentary provided. Test identification numbers may appear to be out of order; however, it was left that way to insure the correct correlation of the data. Thus, no attempt was made to re-order them numerically.

For Test 01, environmental observations were the only means used for the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 2. The most important of the four graphs in Figure 2 is the cumulative reward achieved over the number of steps taken by the agent in exploring the environment. In this graph it is apparent that the agent reached a steady state result of a 0.7 reward value. For this set of tests, the goal was to reach a steady state value of 1.0. A drop in the reward value occurred around 400K steps but was regained but only to the 0.7 reward level. Unfortunately, the use of



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

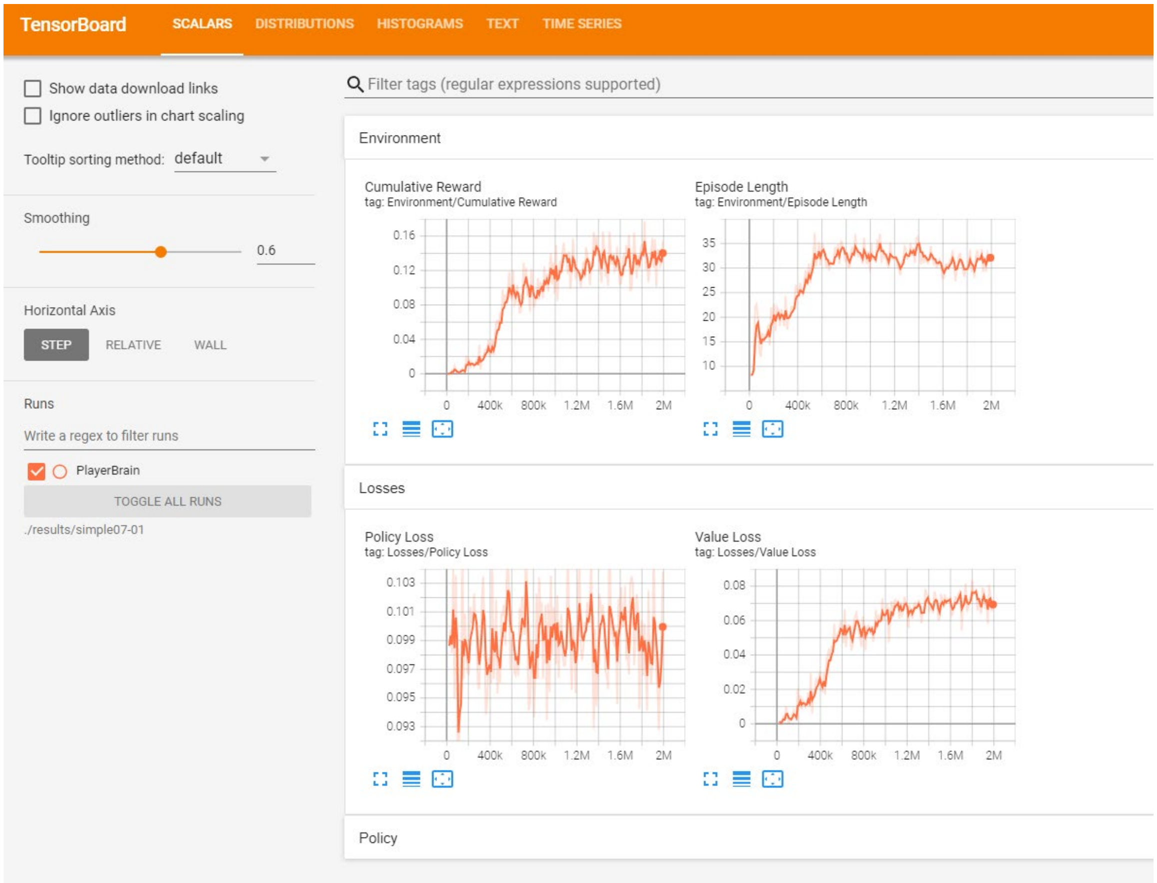
Horizontal Axis: STEP

Runs
☑ ○ PlayerBrain
./results/simple01-10

Figure 2: Environmental observations test results

environmental observations alone were insufficient to reach the targeted level of reward. It was also observed that the episode length corresponded to the reward level achieve. This means that as the agent learns, the episode length should be shortening. The policy loss graph illustrates how much the policy or agent decisions but indicates that exploration was continuing as actions are changed. In general, this was erratic over the course of the learning experience. Value loss was the mean loss of the value function. This relates to how well the model can predict the value at each state. Value loss generally increases when the agent is learning and then decreases once the reward becomes stable. For most of the experiments these graphs correspond to how well the agent was learning. For cases where this differs, it was noted.

For Test 07 the ray cast sensor was the only means used by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 3. In this graph it is noted that the agent learning process was allowed to run to the maximum number of steps (set to two million). The maximum cumulative reward achieved in this test was 0.14 which is indicative of a very poor learning for the agent. This indicates that the use of the ray cast sensor alone is inadequate to achieve any reasonable results in the agent training process.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

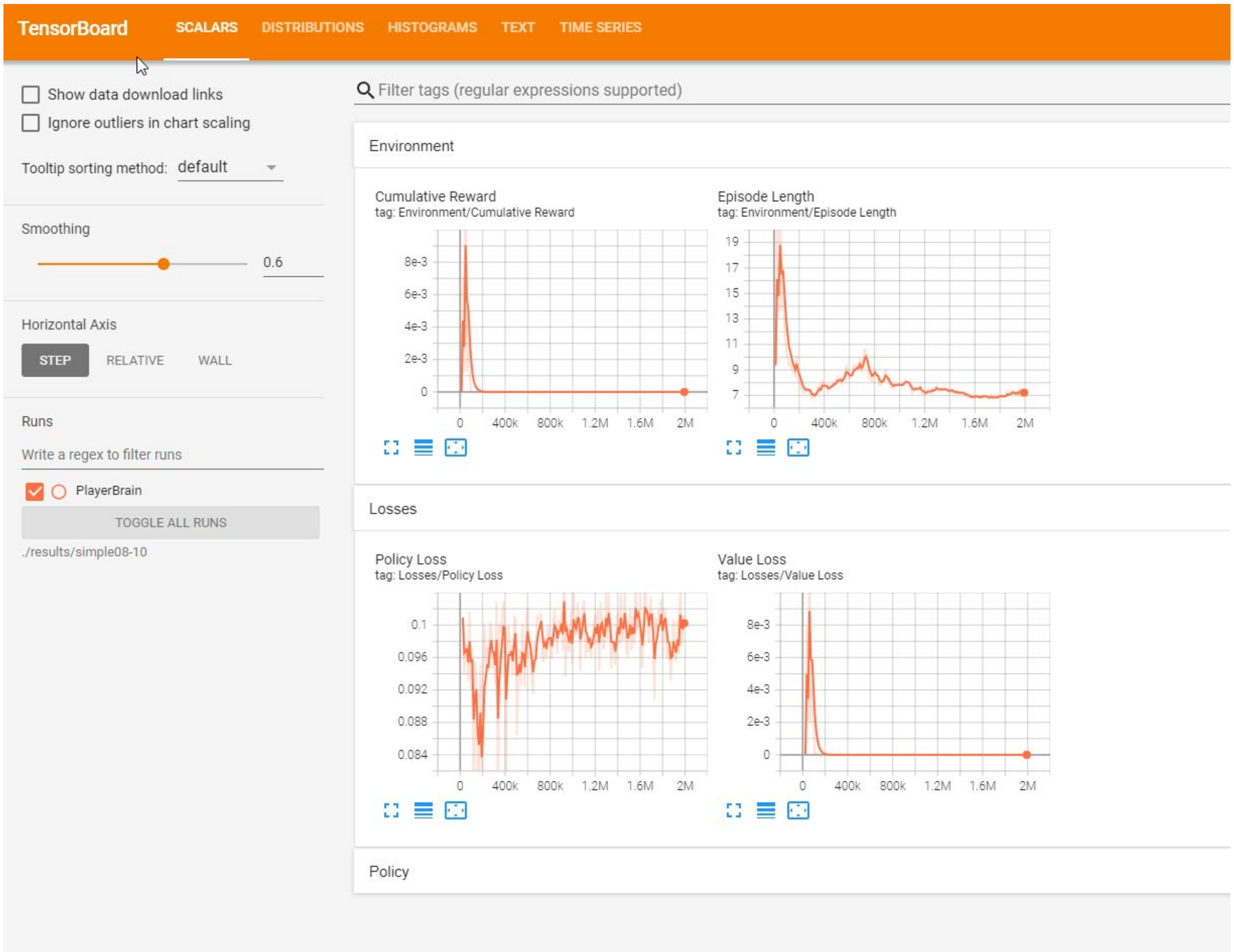
☑ ○ PlayerBrain

TOGGLE ALL RUNS

./results/simple07-01

Figure 3: Ray cast sensor test results

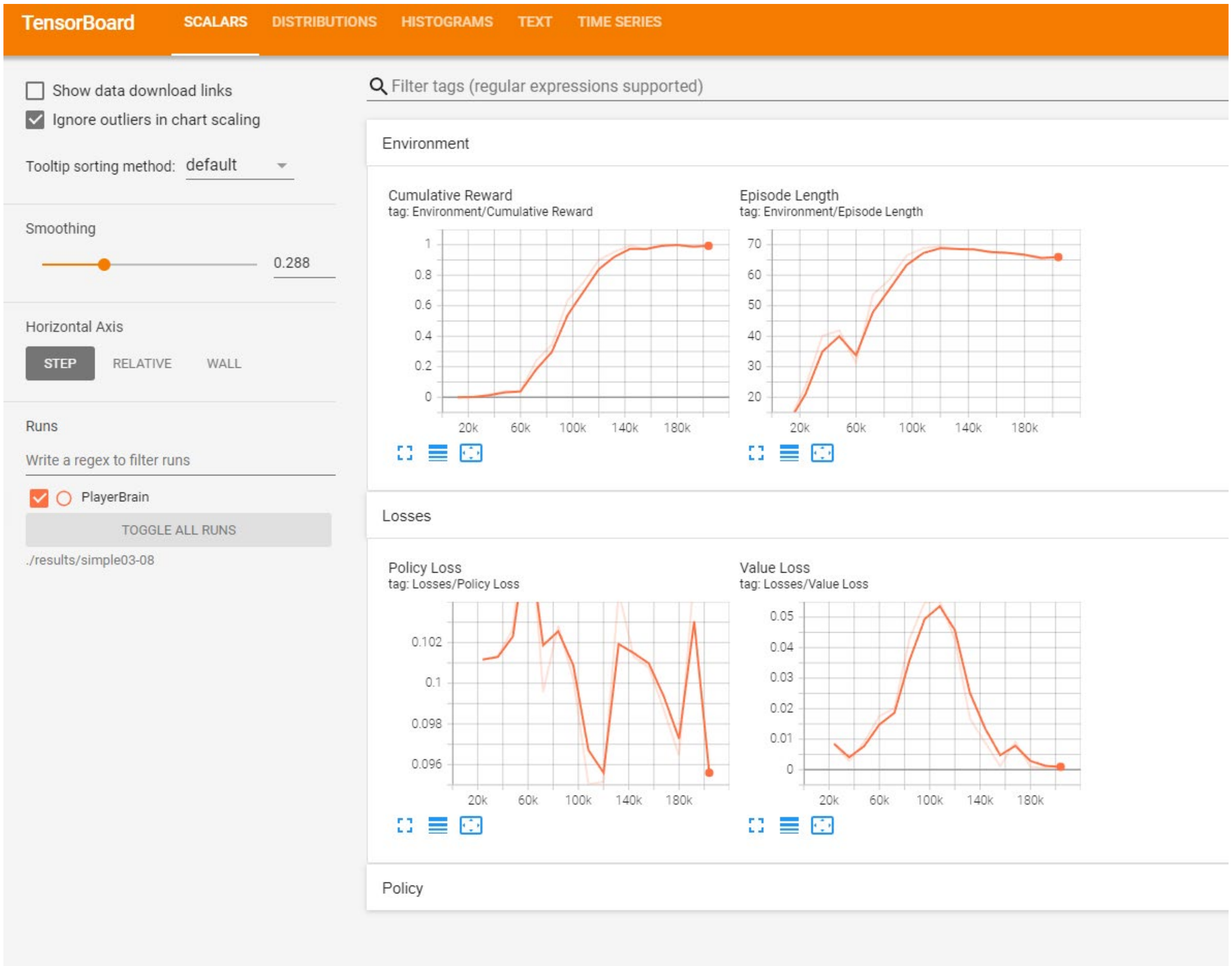
For Test 08 the visual sensor with neural network setting of simple was the only means used by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 4. In this graph it is noted that the agent learning process was allowed to run to the maximum number of steps (two million). The maximum cumulative reward achieved in this test was approximately zero aside from a small spike near the beginning. This was indicative of a totally ineffective configuration for the learning agent by itself.



☐ Show data download links
☐ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
☑ ○ PlayerBrain
./results/simple08-10

Figure 4: Visual sensor test results

For Test 03, both the environment observations and ray cast sensors were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 5. In this graph it is noted that the agent learning process achieved its stated goal of cumulative reward of 1.0 in approximately 260,000 steps. This wass indicative of a quick learning agent which achieves a consistent full reward and stays at that level for the remainder of the steps in the learning process.



□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.288

Horizontal Axis: STEP RELATIVE WALL

Runs

☑ ○ PlayerBrain

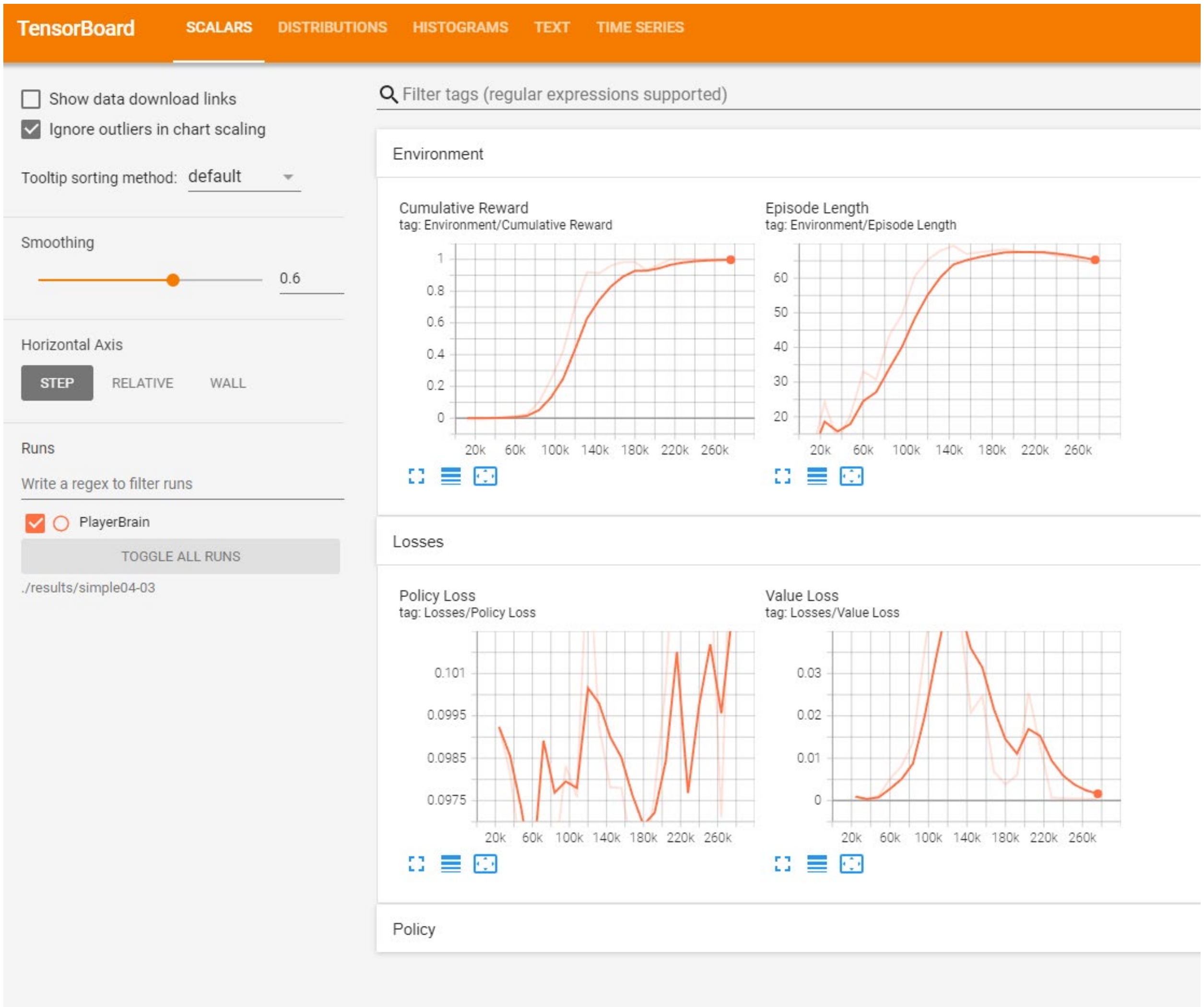
./results/simple03-08

Figure 5: Combined environmental observations and ray cast sensor test

From this test several important items are noted about the learning algorithm configuration. First, all the graphs, cumulative reward, episode length, policy loss and value loss correspond with the observed behavior. Second, the cumulative reward obtained a steady state at the desired reward value of 1.0 and does so quickly and consistently for the number of steps observed. Third, the agent's learning capability was much better than the previous results. It was assumed that the reason for this is that the environmental observations must be included in the configuration and must include at least one other configuration which explores the environment.

These results, because they obtained the expected results, were assumed to be a good choice for the configuration parameters. Follow on experiments dealing with the more complex case of the drone agent, determined that these configuration settings should be used.

For Test 04, both the environment observations and visual sensor with a simple neural network configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 6. In this graph it is noted that the agent learning process achieved its stated goal of cumulative reward of 1.0 in approximately 216,000 steps. This was indicative of a quick learning agent which achieves a consistent full reward and stays at that level for the remainder of the steps in the learning process. This test provided the quickest result in achieving the learning process. It was a slight improvement over the results of Test 03. These results also implied that similar assumptions made for Test 03 apply to Test 04. Essentially, environmental observations are required with an additional algorithm configuration. In this case the visual sensor was replaced by the ray cast sensor and similar results were achieved. This validates the assumptions made by having the same situation repeated and seeing similar results. This implies that the configurations used in this test would also serve well for training agents.



☐ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

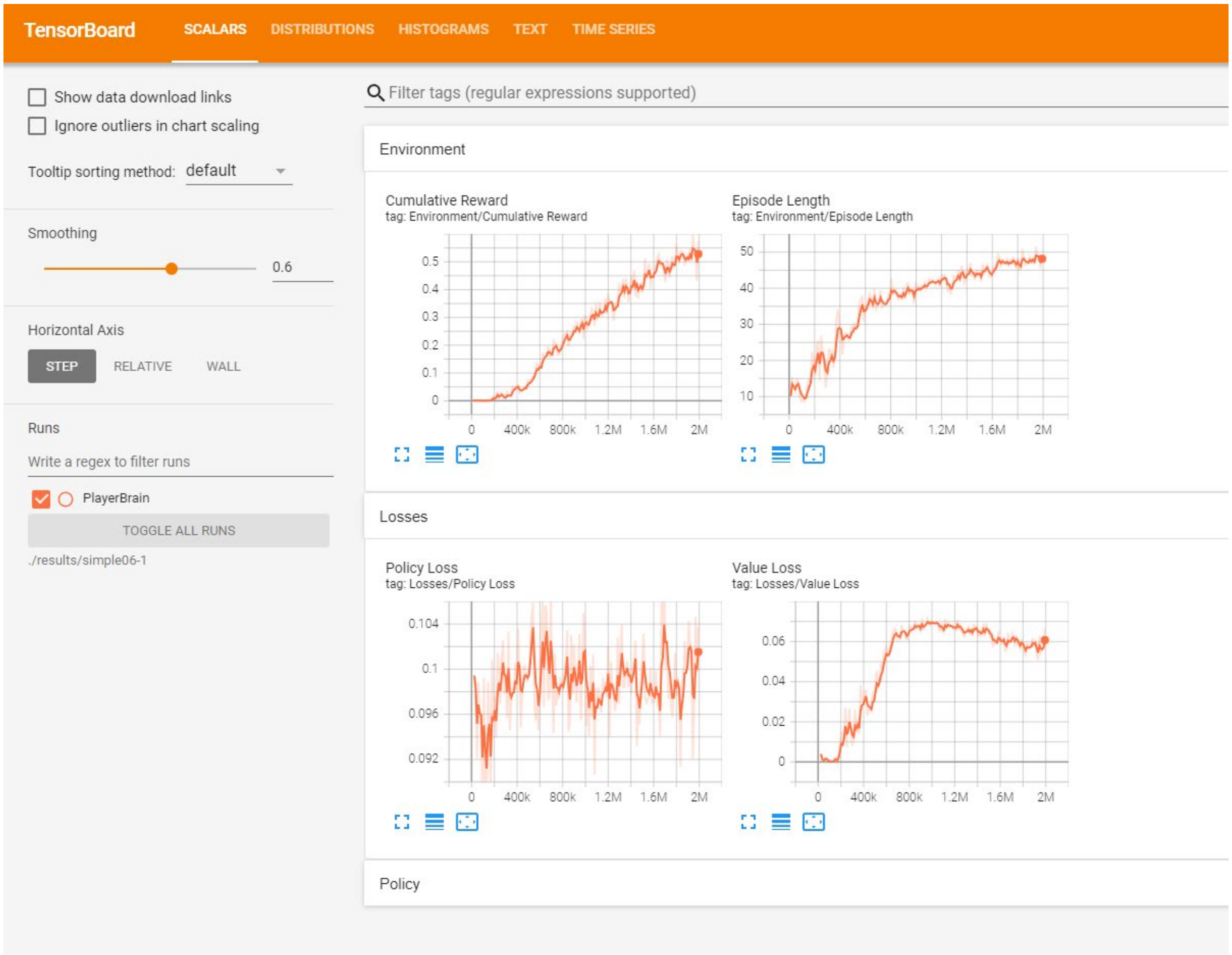
Write a regex to filter runs

☑ ○ PlayerBrain

./results/simple04-03

Figure 6: Combined environmental observations and visual sensor test

For Test 06, both the ray cast sensors and visual sensor with simple neural net configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 7. In this graph it is noted that the agent learning process only achieves a maximum cumulative reward of 0.53 for a maximum of two million steps taken. This is indicative of the implied assumption that environmental observations are a required part of learning algorithm configuration.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

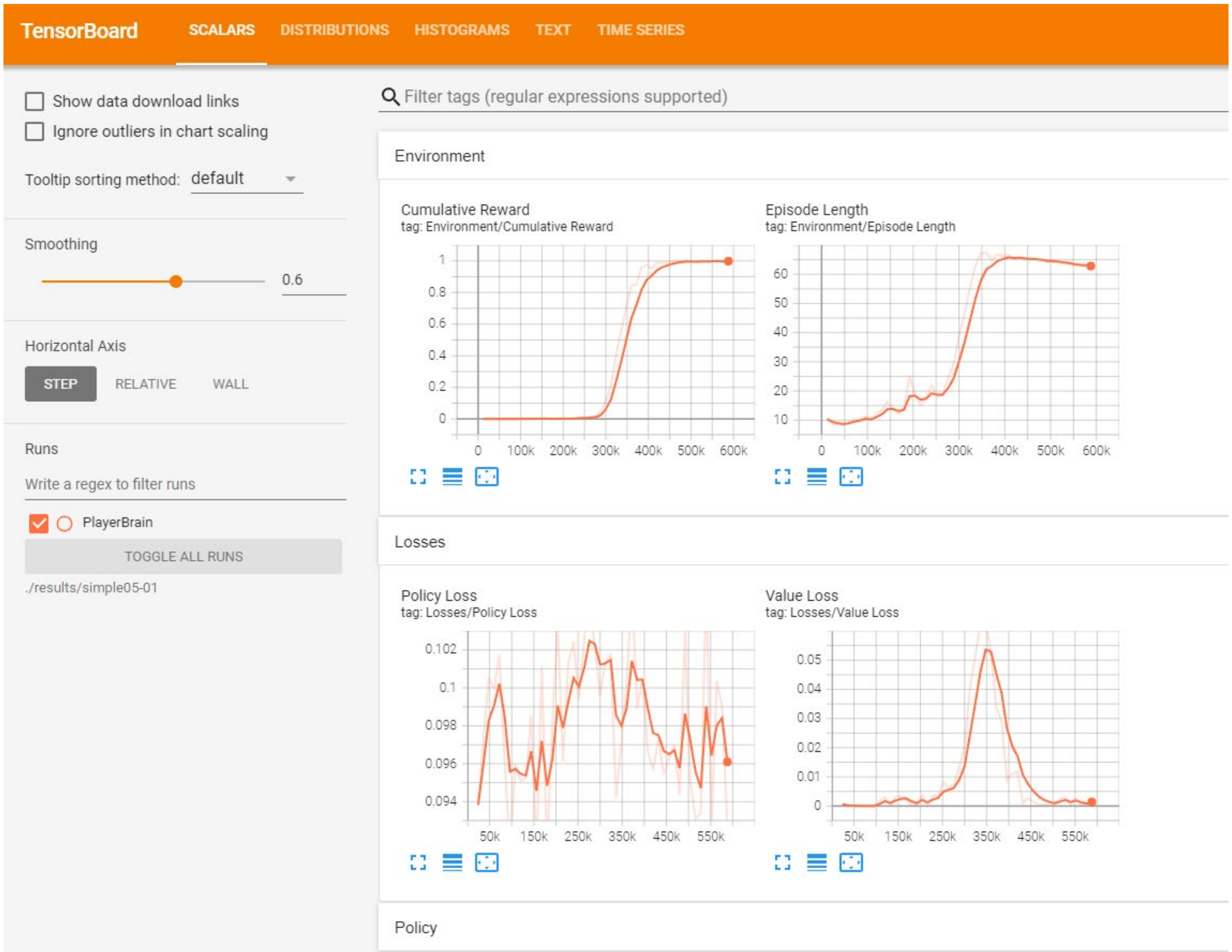
Runs

☑ ○ PlayerBrain

./results/simple06-1

Figure 7: Combined ray cast sensors and visual sensor test

For Test 05, all three configurations, environmental observations, ray cast sensors and visual sensor with simple neural net configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 8. In this graph it is noted that the agent learning process achieves the desired cumulative reward in 480,000 steps. This is indicative of a good learning process with a good steady state at maximum reward. This process may have taken a bit longer than Test 03 and Test 04.



☐ Show data download links

☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

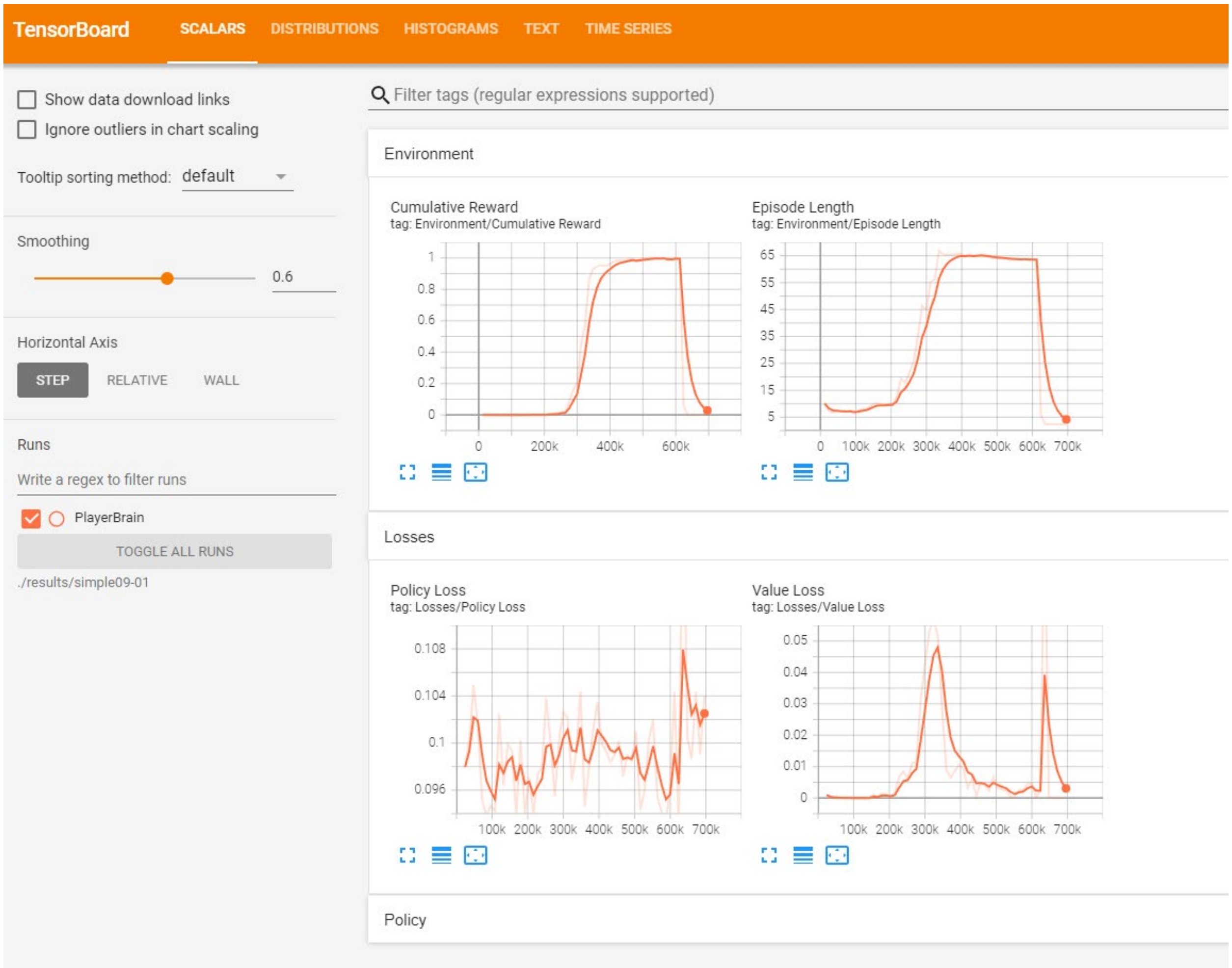
☑ ○ PlayerBrain

./results/simple05-01

Figure 8: Combined environmental observations, ray cast sensors, and visual sensor test

For Test 05, it was observed that the learning process required more processing steps, but the response was still considered quick in reaching the desired reward level of 1.0. It was assumed that in the early stages of the learning process that using all three primary configurations may require additional processing and thus longer to reach the desired state. The other implied assumption is that using all three primary configurations provided a more robust learning process which would be more suitable for more complicated training environments.

For Test 09, all three configurations, environmental observations, ray cast sensors and visual sensor with nature_cnn neural net configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 9. In this graph it is noted that the agent learning process achieves the desired cumulative reward in 540,00 steps. It was observed that the cumulative reward of 1.0 was short lived. At approximately 600,000 steps the cumulative reward drastically fell and achieved a cumulative reward of approximately 0 by 700,00 steps. The learning process was terminated at this point. This configuration only differs from test 05 in the use of the neural network configuration. The nature CNN neural network configuration was used and appeared to achieve the same results as using the simple neural network configuration up until the drop in reward occurred a little while later. An inference model obtained prior to the drop would provide a good learning model to use, however, the inference model must be saved prior to the drop in reward value. This makes this configuration with natural CNN a bit more of a risk than with the simple neural network configuration seen in Test 05. The conclusion here was to use Test 05 configuration over that of this test configuration.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

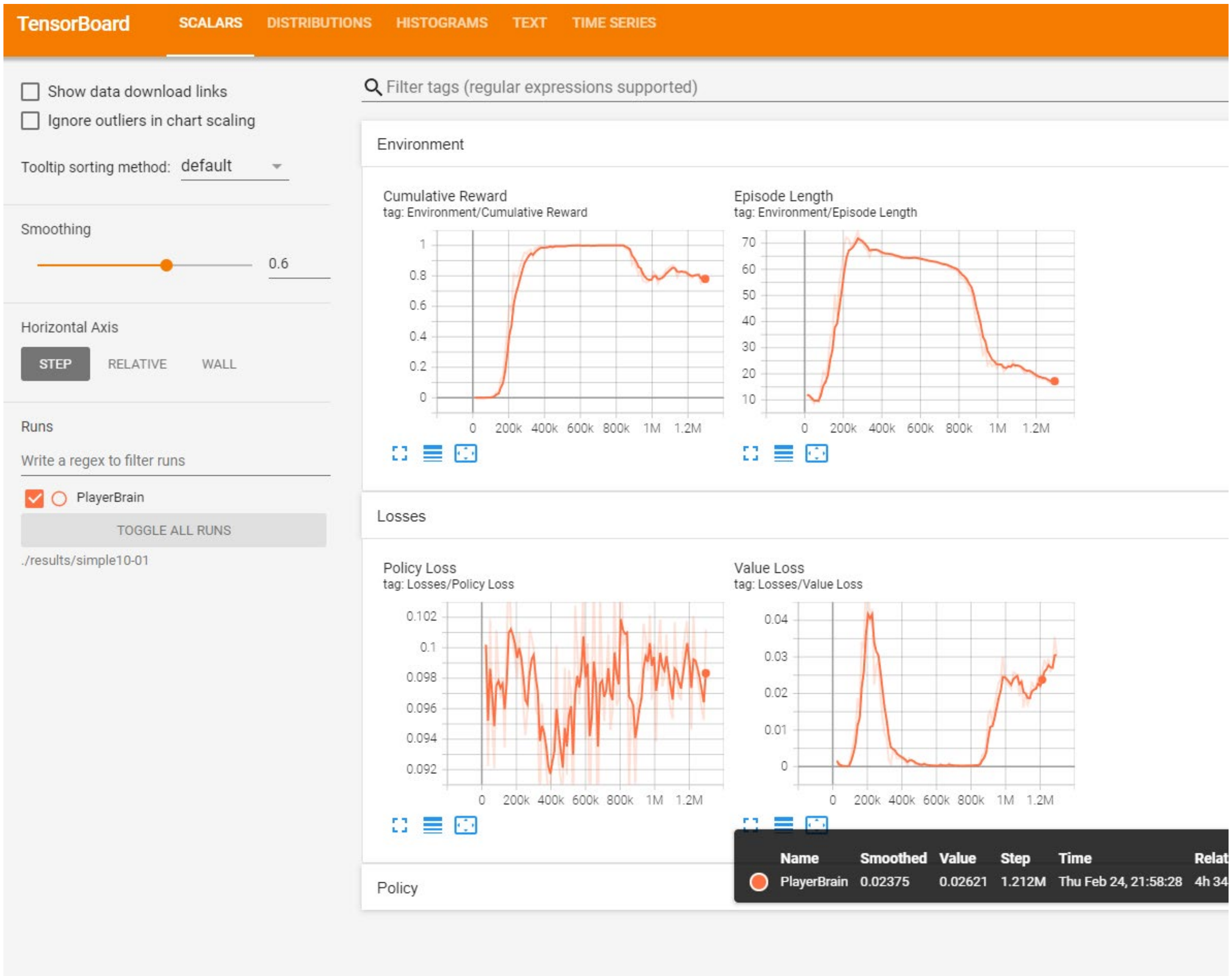
Write a regex to filter runs

☑ ○ PlayerBrain

./results/simple09-01

Figure 9: Combined sensors using nature CNN for visual sensors

For Test 10, all three configurations, environmental observations, ray cast sensors and visual sensor with Resnet neural net configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 10. In this graph it is noted that the agent learning process achieves the desired cumulative reward in 530,00 steps. It was observed that the cumulative reward of 1.0 was short lived. At approximately 850,000 steps the cumulative reward drastically fell and achieved a cumulative reward of approximately 0.8 by approximately 1 million steps. The learning process was allowed to run to 2 million steps. This configuration is very close to the that of test 09 however, the drop in cumulative reward leveled back out to a value of 0.8. This implies this was a better choice that used in Test 09 but not as good as Test 05.



□ Show data download links
□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

☑ ○ PlayerBrain

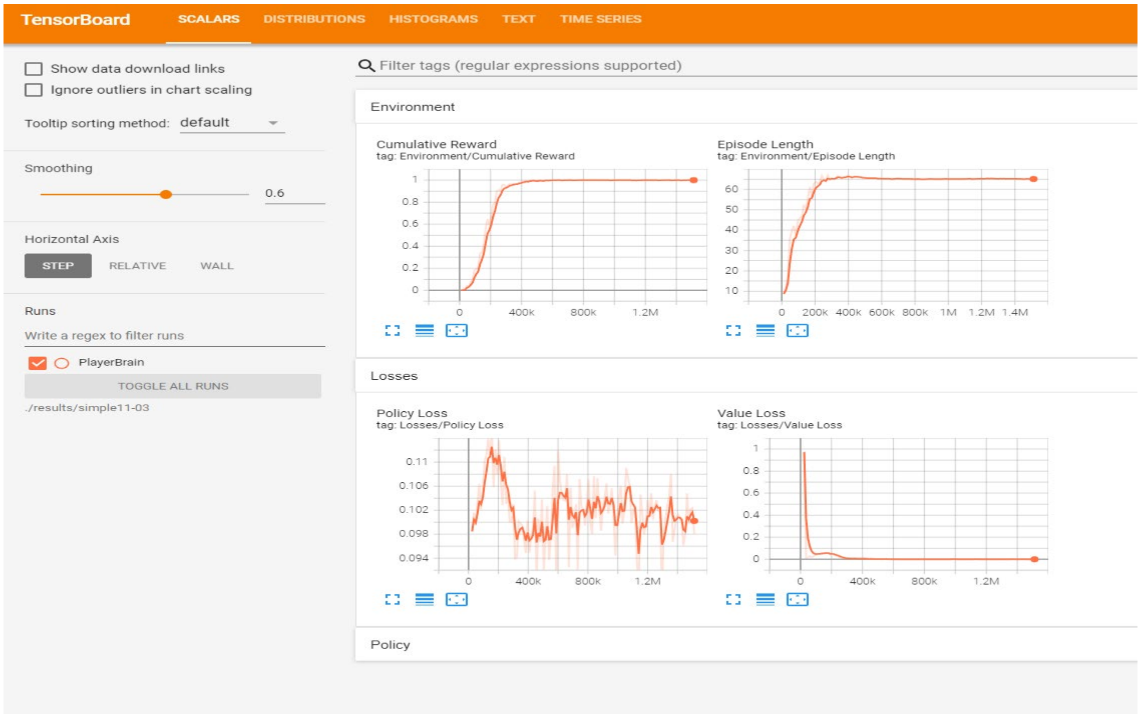
./results/simple10-01

	Name	Smoothed	Value	Step	Time	Relat
●	PlayerBrain	0.02375	0.02621	1.212M	Thu Feb 24, 21:58:28	4h 34

Figure 10: Combined sensors using RESNET for visual sensors

For Test 11 all three configurations - environmental observations, ray cast sensors and visual sensor with make3 neural net configuration were used as the means by the agent to explore and exploit the synthetic environment to determine how to reach its goal. The data results are illustrated in Figure 11. In this graph it is noted that the agent learning process achieves the desired cumulative reward in 540,000 steps. It was observed that the cumulative reward of 1.0 was consistent and maintained for up to 1.5 million steps. The learning process was excellent and on par with Test 05. The only difference was the amount computation time the make3 option uses over that of the simple neural network option. Considering this computation time increase it was still assumed that the configuration of Test 05 would still be a better configuration than the configuration of Test 11. The reasoning for this selection was based on ML-Agents' recommendation as the de facto configuration.

Considering the analysis of the set of simple agent tests the conclusion initially made at the start of this sections still holds. Configuration setting for Test 05 provide the best learning environment for the learning algorithm applied in this work.



□ Show data download links
□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

☑ ○ PlayerBrain

./results/simple11-03

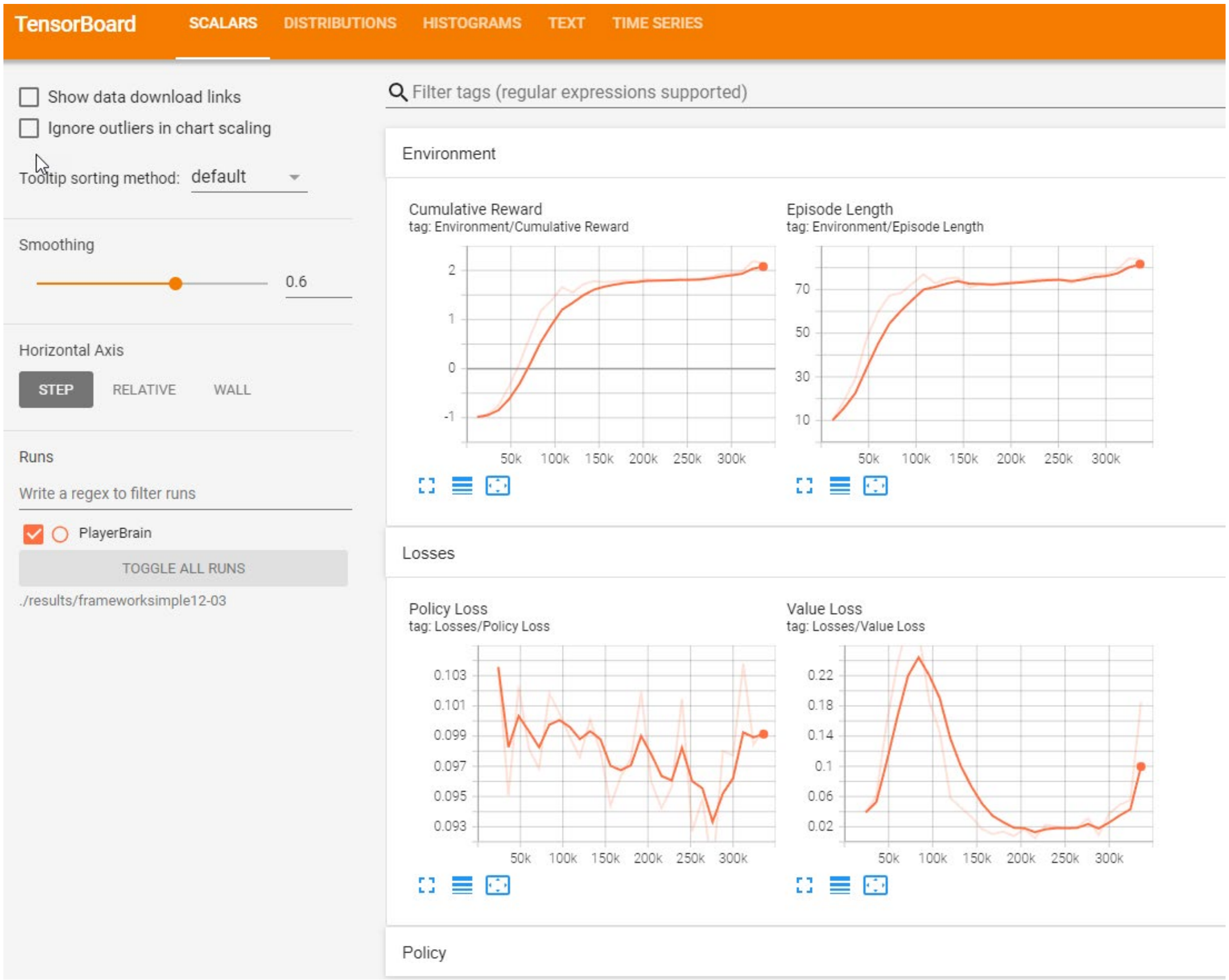
Figure 11: Combined sensors using Make3 for visual sensors

Learning Improvements with Context Aware Elements

Additional testing was performed on the simple agent testing framework. The goal was to determine if the addition of certain types of context aware elements could aid in the learning process. The concept was to improve learning performance as well as the rate at which the agent learned. A description of each type of test, modifications made, and their results are presented in this section.

One form of a context aware element used was the distances from the learning agent to the target. As the agent approached the target a distance measurement was made to determine if the proximity to the target was improving. In this test four distance measurements were used. These distance measurements were based on a straight line from the agent to the target. Each distance to the target was scored with a small reward which varied based upon the magnitude of the distance. The four distances measurement were spread evenly between the initial position of the learning agent to that of the target position. The trigger at the largest distance provided a reward value of 0.1. Each consecutive distance trigger location was increased by a reward of 0.1 giving the four reward values based on distance as 0.1, 0.2, 0.3 and 0.4. In ML-Agents, the documentation for the type of agents used suggests using total reward values between positive 1.0 and negative 1.0. Negative reward values were considered a value of punishment to the learning agent. In this test a negative reward of -1.0 was given when the agent fell off the platform. A positive reward of 1.0 was given when the agent reached the target. For this test with the added distance reward values, the suggested range became larger than the suggested 1.0 max reward value. The concept was that the agent would learn through incremental small rewards which increased as the agent continued to approach towards the target versus some other direction.

Figure 12 provides the results of Test 12-03. A quick evaluation of this test was performed and was terminated once a reward value of 2.0 was reached with just over 444000 steps taken in the learning process. Figure 12 illustrates this fact along with slight indications at the point of termination that the final reward value may even be higher. It is noted that episode length and value loss plots provide these same indications.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

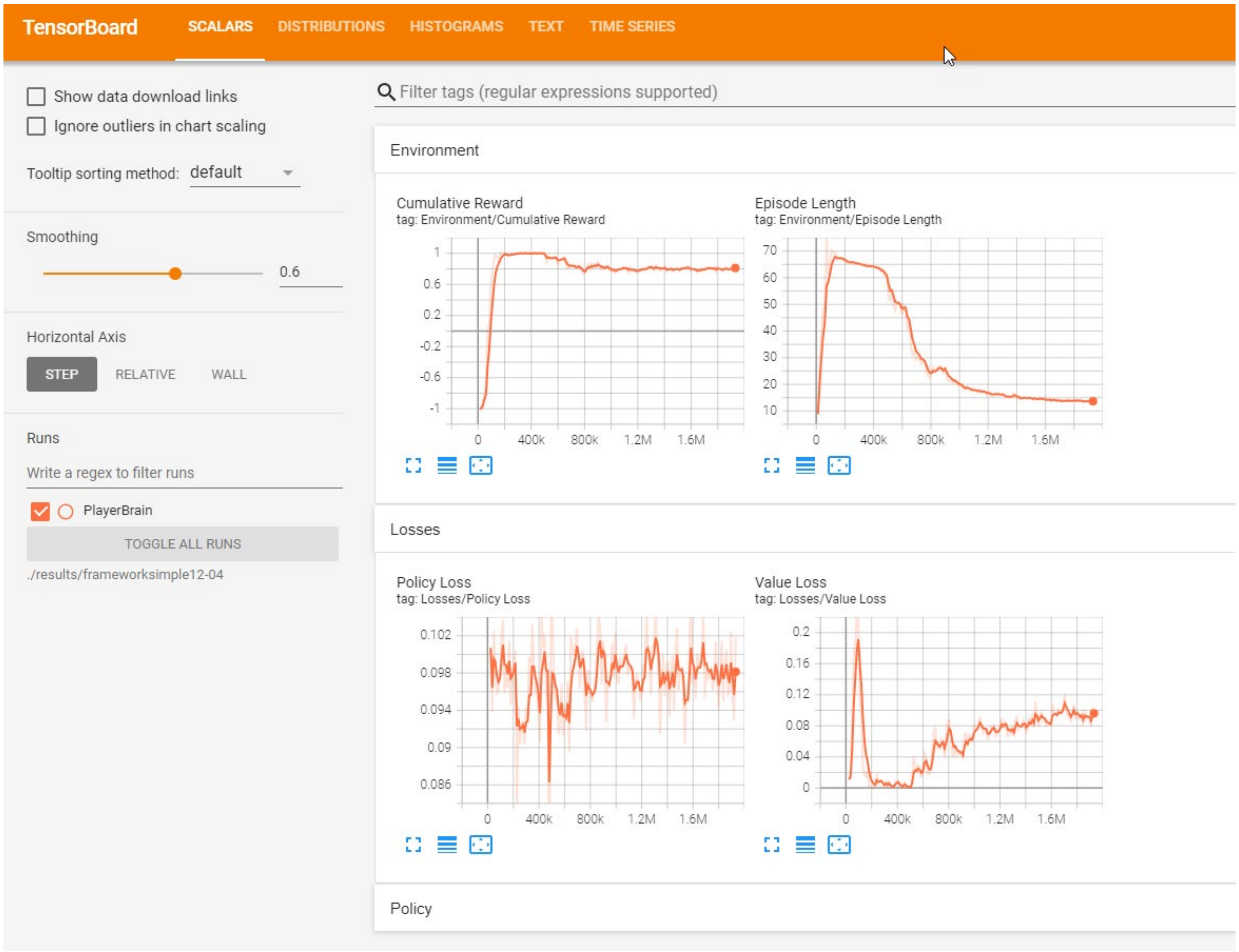
☑ ○ PlayerBrain

./results/frameworksimple12-03

Figure 12: Results of distance-based reward learning

The rate at which the agent learned indicated that it was much quicker than Test05 by incorporating the intermediate distance rewards. The reward also flattened out for over 200,000 steps. The total reward, however, reached a value just over 2.0 before termination. This is indicative of adding more reward-based context-aware elements which build up the total reward value. An important indicator was seen in the data collected from Test 12-03-3 which was not captured in Figure 12-03-3. The reward continued to increase to a value of over 8.0 after 432000 steps. A testing practice was to capture videos of the agents as they explored the synthetic environment. A video capture for Test 12-03-3 showed behaviors beyond the 300,000 plus steps seen in Figure 12-03-3. The data in 400,000 step range, and as seen in the video of the test, indicate that the agent learned to rock back and forth across the distance reward boundaries. This action caused the reward to continue to increase without continuing to get closer to the reward. This provides an explanation for the much higher reward value of over 8.0 at about 432000 steps in the learning process.

The next few tests were attempts to overcome this learning behavior of the agent. Methods were used to avoid the situation where the agent vacillates over the reward boundary because it is learning it gets rewarded to perform this action. This behavior does not align with the desired goal of simply moving towards the target. The first correction used was to modify this behavior was demonstrated in Test 12-04. The premise considered was to drop the reward value given in the distance reward method. The distance reward was dropped from 0.1 to 0.001. The effect of making the distance reward much smaller proved somewhat useful as seen in Figure 13.



☐ Show data download links

☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

Write a regex to filter runs

☑ ○ PlayerBrain

./results/frameworksimple12-04

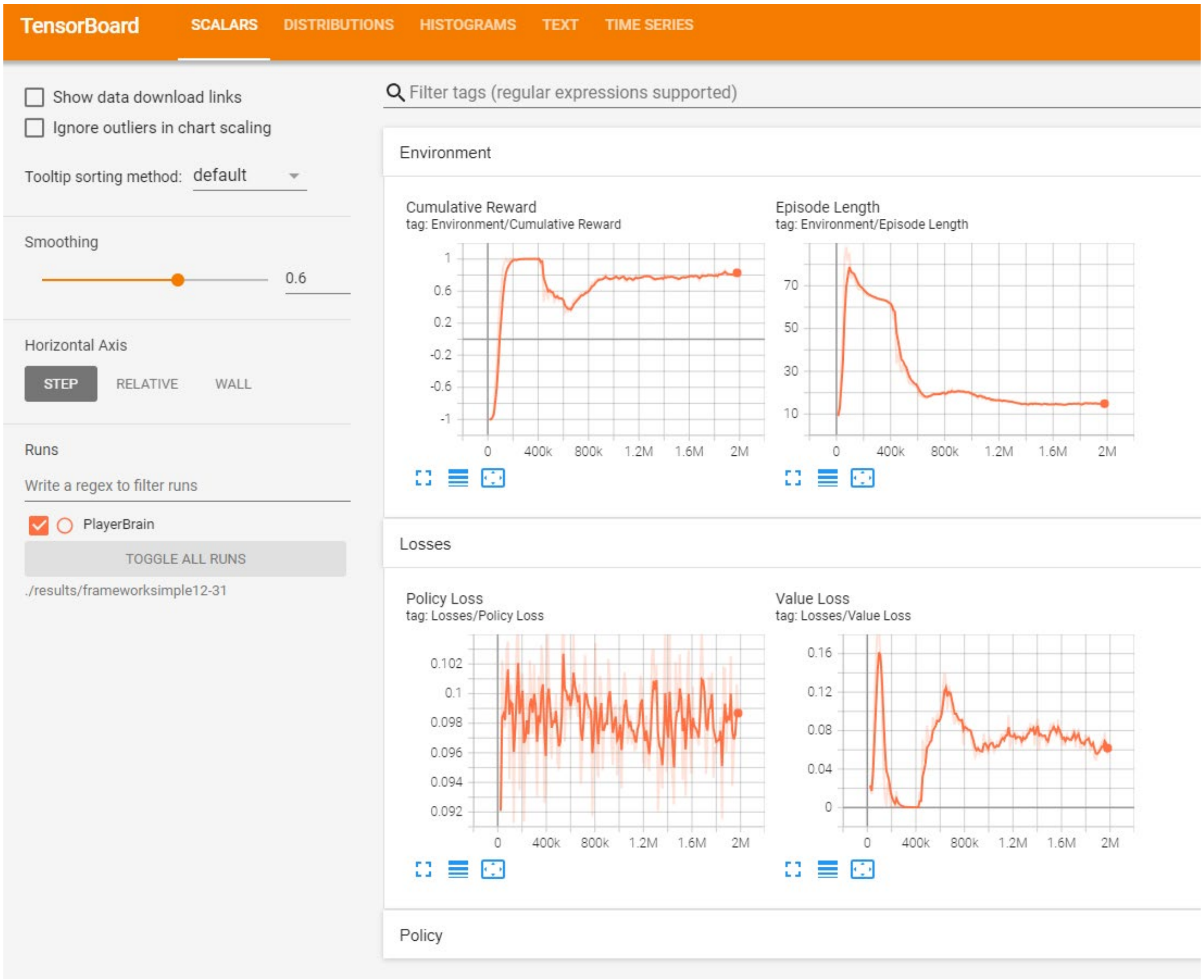
Figure 13: Reduction in reward using distance-based reward learning

In Test 12-04 the reduction in size of the reward made for a more stable reaction by the learning agent. In Figure 13 the reward of 1.0 was reach quickly within approximately 200,000 steps which lasted until about the 420,000-step range. At this point, the reward began to decrease to a value of around 0.8 and stayed there until completion of test at two million steps. The reduction in distance reward size proved valuable, however, the overall reward value dropped from the idea 1.0 to a 0.8 reward level.

Different approaches were taken for Test 12-05 through Test 12-30. Rather than apply a distance reward, a reward portal was positioned at givens distance and sufficient size for the agent to pass through. The learning agent was rewarded as it passed through these portals. A Unity box collier object with a trigger provided the structure of the reward portal. A reward was provided to the agent once it entered the box collider. Unity objects and scripts were utilized to provide this context-aware element. Initially, a single box collier object was used with the idea to expand it to several of portals along the path to the target. The concept was to provide additional guides to the agent to keep moving toward the target. Several modifications were made to this type of context-aware element along with changing the size and positions of the box collider. Over twenty-five attempts yielded no worthwhile results. It became apparent that the box collider was also interfering with the Unity 3D Ray Perception Sensor. It was blocking the rays from reaching the target by colliding with the box collider and never reaching the target until the agent had passed through the portal. Attempts were made to rectify this condition by examining various parameters and settings for box collider objects, but none yielded a solution.

The next context-aware element modification made was to switch from using box colliders to rectangular plane colliders. Plane colliers have a normal directional face which is the side that the agent must pass through to be detected. Unfortunately, the plane collider did not behave as declared and entrance on either side of the plane collider would activate the trigger. For the simple agent test it became easier to just determine which side of the plane the agent was positioned when the intersection triggered. This allowed a positive reward to be given when the agent passed through the plane in the correct direction. If the agent came back through the other direction, then a negative reward equal to the positive reward in the other direction would be applied. This allowed a positive reward to be taken away if the agent was going to vacillate over a reward boundary. This was used to prevent the agent's overfitting behavior by controlling the reward at the portal plane boundaries. A onetime reward for the context-aware element would not work within a DRL explore and exploit approach.

Test 12-31 was used as the test to explore the plane object as a reward portal. Like Test 12-04, Test 12-31 provided a more stable result than the distance reward concept however it exhibited the same if not worse behavior as Test 12-04 as shown in Figure 14.



□ Show data download links
□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

☑ ○ PlayerBrain

./results/frameworksimple12-31

Cumulative Reward
tag: Environment/Cumulative Reward

Episode Length
tag: Environment/Episode Length

Policy Loss
tag: Losses/Policy Loss

Value Loss
tag: Losses/Value Loss

Figure 14: Controlled plane portal using distance-based reward learning

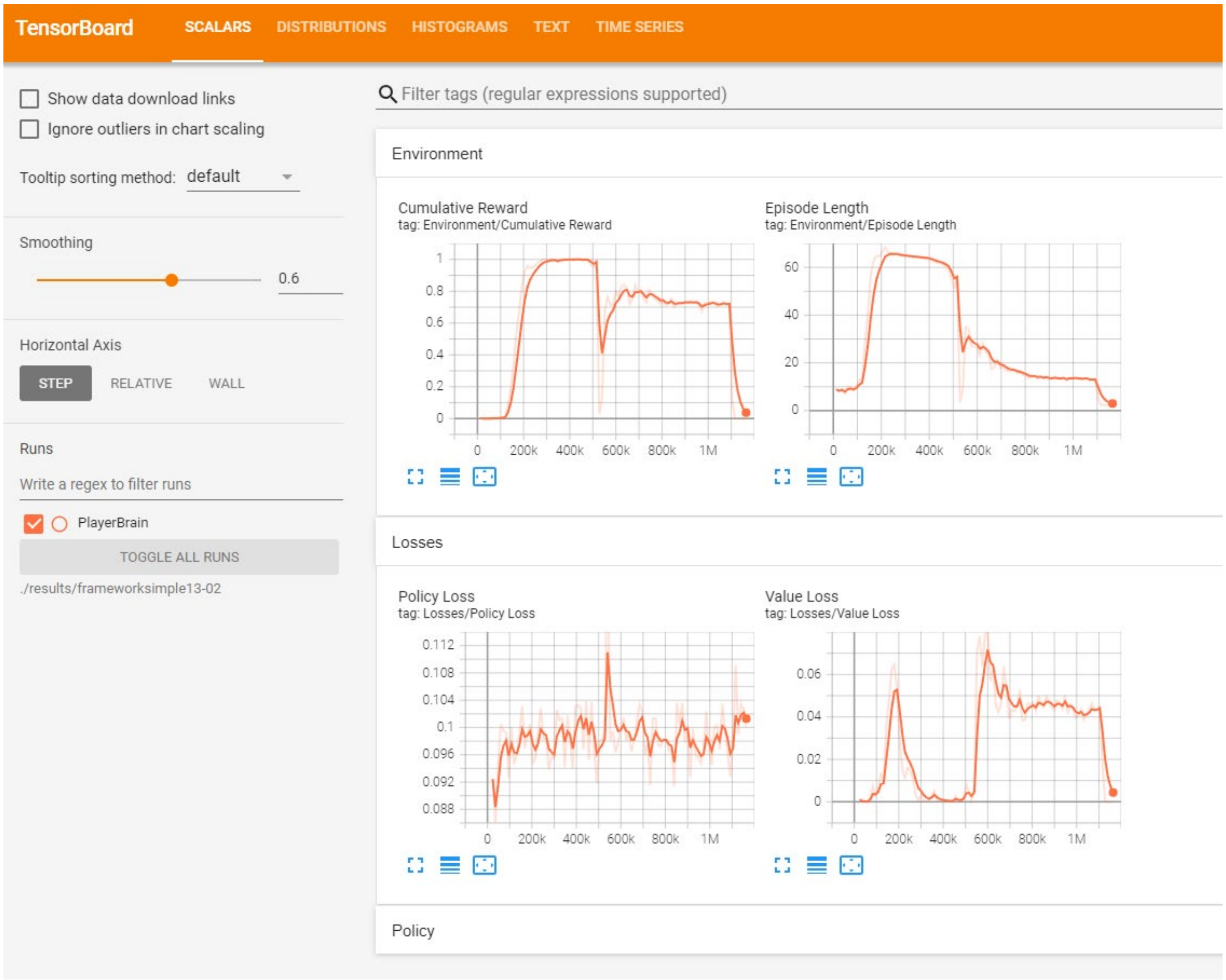
In Figure 14 a quick rise to a reward value of 1.0 was achieved at about 200,000 steps and ran to just over 400,000 steps. At this point a much larger drop in the reward occurred in comparison to Test 12-04. It did, however, regain back to a reward of 0.8 at about 900,000 steps and remained steady through the two million max steps in the configuration.

Overfitting of DRL PPO-Based Learning Agents

In the tests results stated in the previous section a potential problem of overfitting occurred in the DRL PPO-based learning algorithm. Overfitting is a known behavior for machine learning algorithms and not that well known for DRL algorithms. Research literature (Zhang, Vinyals, Munos & Bengio, 2018; Song, Jiang, Tu, Du & Neyshabur, 2019)) has provided research which indicates that such a phenomenon does occur in DRL algorithms. The most common reason for DRL overfitting occurs because the agent tends to correlate the reward it collects with spurious features it observes in the synthetic environment. This tends to cause problems in the Markov Decision Process (MDP) which is an integral part of DRL algorithms. The tests results of the previous section seem to indicate this behavior. One conclusion is that the attention given by the agent to the additional distance or portal reward mechanisms thus throwing off the true reward of reaching the target.

This was exhibited first by the continued increase in reward seen in the data of Test 12-03-3. In that test, an adjustment to the reward value size provided improvement, however, results for the follow-on test, Test 12-04, and Test 12-31, demonstrated a similar drop in the reward value after additional learning was processed. This is illustrated in Figures 13 and 14 respectively.

When determining the DRL PPO-based configuration parameters, Test05 results indicated the best set of configurations to be used. This test, however, was terminated early or just before the signs of DRL overfitting might have occurred. Test05 was re-run as Test13-02 to see if overfitting did occur in this test. The results are shown in Figure 15.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

☑ ○ PlayerBrain

./results/frameworksimple13-02

Cumulative Reward
tag: Environment/Cumulative Reward

Episode Length
tag: Environment/Episode Length

Policy Loss
tag: Losses/Policy Loss

Value Loss
tag: Losses/Value Loss

Figure 15: DRL overfitting

This check for overfitting indicated two more problematic effects on agent learning. As seen in Figure 13-02, overfitting began just over the 500,000 steps in the learning process. It stabilized between reward values of 0.8 and 0.7, however, this dropped the reward to a value of zero very quickly just after 1,200,000 learning steps in the process. This could be indicative of two things.

First, overfitting can occur even without the use of added reward-based context aware elements. Second, the addition of reward-based context-aware elements provided an increase in the rate at which the desired reward was achieved during learning and provided a more stabilizing effect when overfitting did occur. From this discovery of DRL overfitting, some type of overwatch or alerting mechanism must take place to terminate the learning to obtain the best reward value for the inference model. For the follow-on SDR+ tests, this was achieved through monitoring the reward response during training and cutting off the learning process before or just as overfitting occurs. The addition of reward-based context-aware elements was also used to reduce the training time for the inferred model. In the configuration settings, an additional method was used by saving inferred models at shorter intervals during the learning process. This provided better assurance of obtaining an inferred model with the best reward values prior to overfitting taking place.

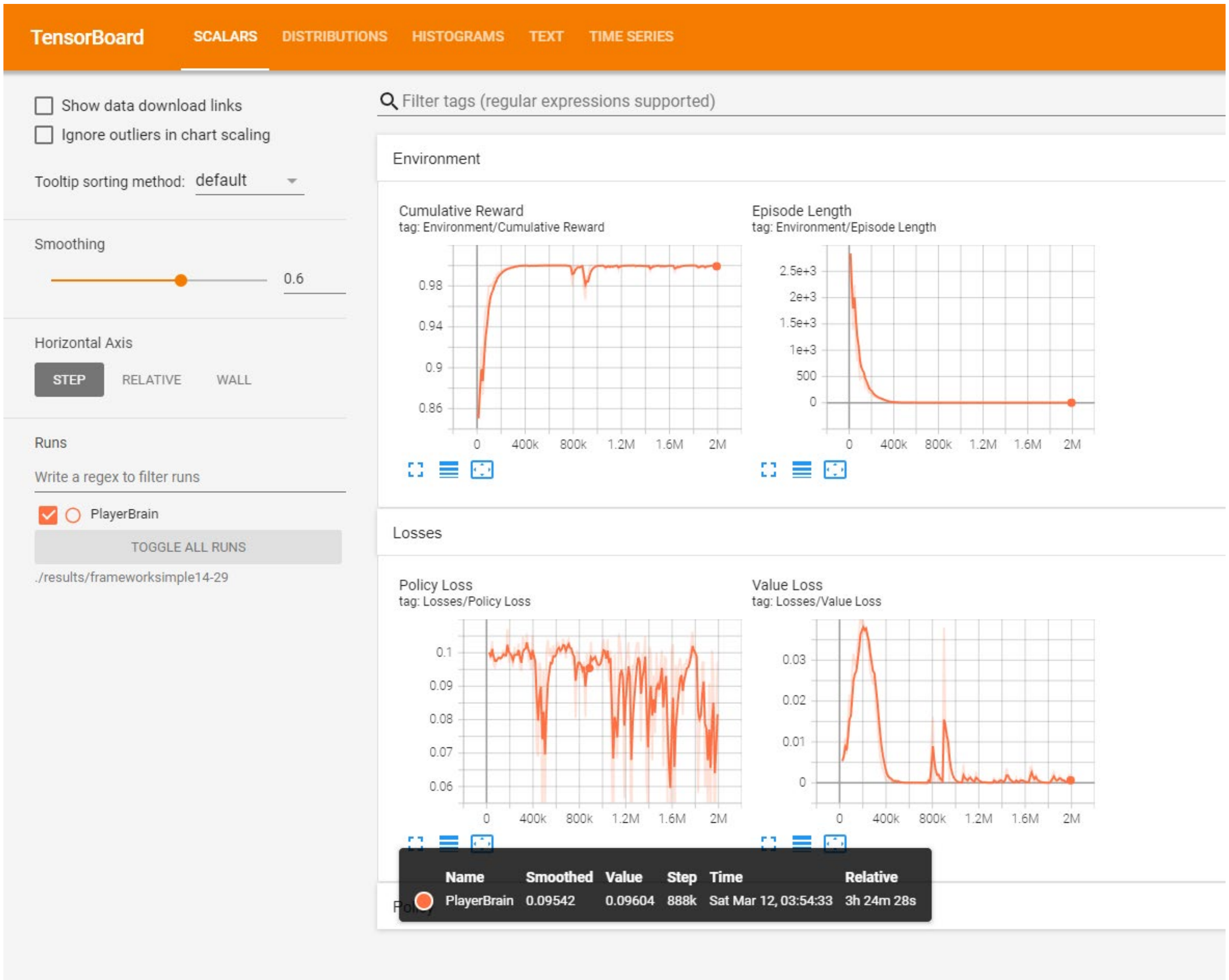
Learning Improvement with Semi-Reward-Based Context Aware Elements

In the previous sections the results for learning improvement through context aware elements was illustrated. These results provided indications of overfitting of the DRL algorithms during the learning process. This section discusses the results obtained by moving the context aware elements away from being solely reward-based and used more to provide guidance. This would also be accomplished without the injection of reward along with a means to control cumulative penalties through the context-aware elements. The objective was to avoid or limit the overfitting of DRL PPO-based algorithm used in this study.

Test 14-29 was formulated to explore the semi-reward-based context-aware elements. Previously reward-based the context-aware elements were set up to provide a small addition of a reward or punishment based on how the agent interacted with the elements. Test 14-29 was a test of the simple agent with a single target as its goal. In the case of Test 14-29, the agent movement was three dimensional versus a two-dimensional movement in previous tests. This modification provided a closer approximation to the drone use case and was performed to help solve training problems observed in the drone use case. For the semi-reward-based approach the objective was twofold. First, to provide guidance to the agent without reward. An example of this was to continually provide direction to the agent in terms of the direction of the next reward. Second, this approach needed to enclose the volume of exploration with the addition of walls and ceilings due to the extension of three-dimensional movement. These served as collision surfaces in the testing environment and provided negative reward for collisions that occurred. This approach used a controlled cumulative reward on the DRL algorithm. A small negative reward was applied for each collision; however, the agent was not reset for each collision. The agent continued to explore the space instead of ending the episode. The monitoring of the cumulative reward provided a threshold to the cumulative reward which when exceeded would end that episode and start the process again. This avoided the repeated reset of the agent for each collision but limited the amount of negative reward that could be accumulated, thus modifying, and reducing the effect of negative rewards provide in the context-aware elements.

Figure 16 illustrates the results of Test 14-29. The graphs for cumulative reward, episode length and value loss all demonstrated the expected behavior of the agent during the training process. The data collected indicated a quick rise to the expected 1.0 maximum reward value at about 350,000 steps into the learning process. The learning process was executed to the maximum steps limit set of two million steps. As illustrated in Figure 16 for the cumulative reward, the maximum reward of 1.0 was obtained for the remainder of the learning process.

Noise in the values was approximately no more than a difference of 0.02 in reward value. This noise was not considered to be associated with overfitting.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ PlayerBrain

./results/frameworksimple14-29

	Name	Smoothed	Value	Step	Time	Relative
●	PlayerBrain	0.09542	0.09604	888k	Sat Mar 12, 03:54:33	3h 24m 28s

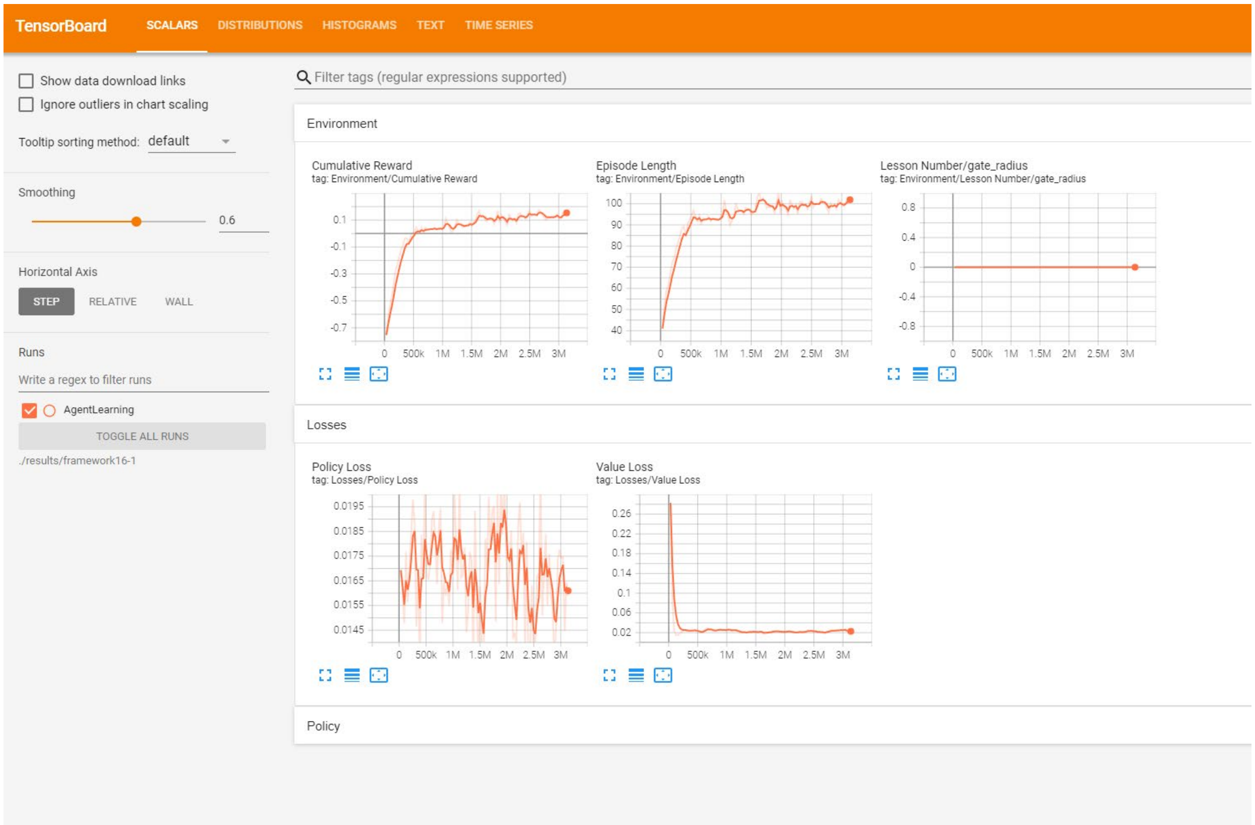
Figure 16: DRL overfitting correction

Analysis indicated that Test 14-29 provided improvement over the previously applied reward-based context-aware elements. This result was for a single result, however, for purpose of this work it was be used for the drone racecourse scenario. Although a single result is in general insufficient., there was a trend in improvement indicating value in their use for the drone use case.

City Scene Baseline Agent Metrics

This section describes the training of an agent that was used as the baseline for comparison against agents trained under the SDR+ technique. The baseline agent was founded on the test results discussed in the previous subsections. The configuration parameters were set to use a combination of environment observations, ray cast sensing, and visual sensing. Lessons learned from the use of context-aware elements were used to guide and improve the learning process. The shortcoming of DRL overfitting was addressed and implemented into the baseline agent through semi-reward-based context-aware elements. These provided the foundations for the configuration and structure of the baseline agent. The city scene environment was then utilized as the staging for the drone use case to be applied to the baseline agent.

The initial test for the baseline agent in the city scene, Test 16-1, indicated that further understanding of the DRL algorithm and use of context-aware elements was required for further training of the baseline agent. This occurred prior to the previous few subsections, which in turn led to those investigations. These indications are shown in Figure 17. In this case, the cumulative reward , episode length and value loss graphs were as expected. The cumulative reward rose quickly at the start, however, that rise leveled off and fell short of reaching an expected reward value of 1.0. As seen in Figure 17 the cumulative reward value only reached to a value of just over 0.1. This was a clear indication that the steps provided in the previous subsections were necessary to determine the necessary changes to the training methods to achieve a better trained baseline agent.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP | RELATIVE | WALL

Runs

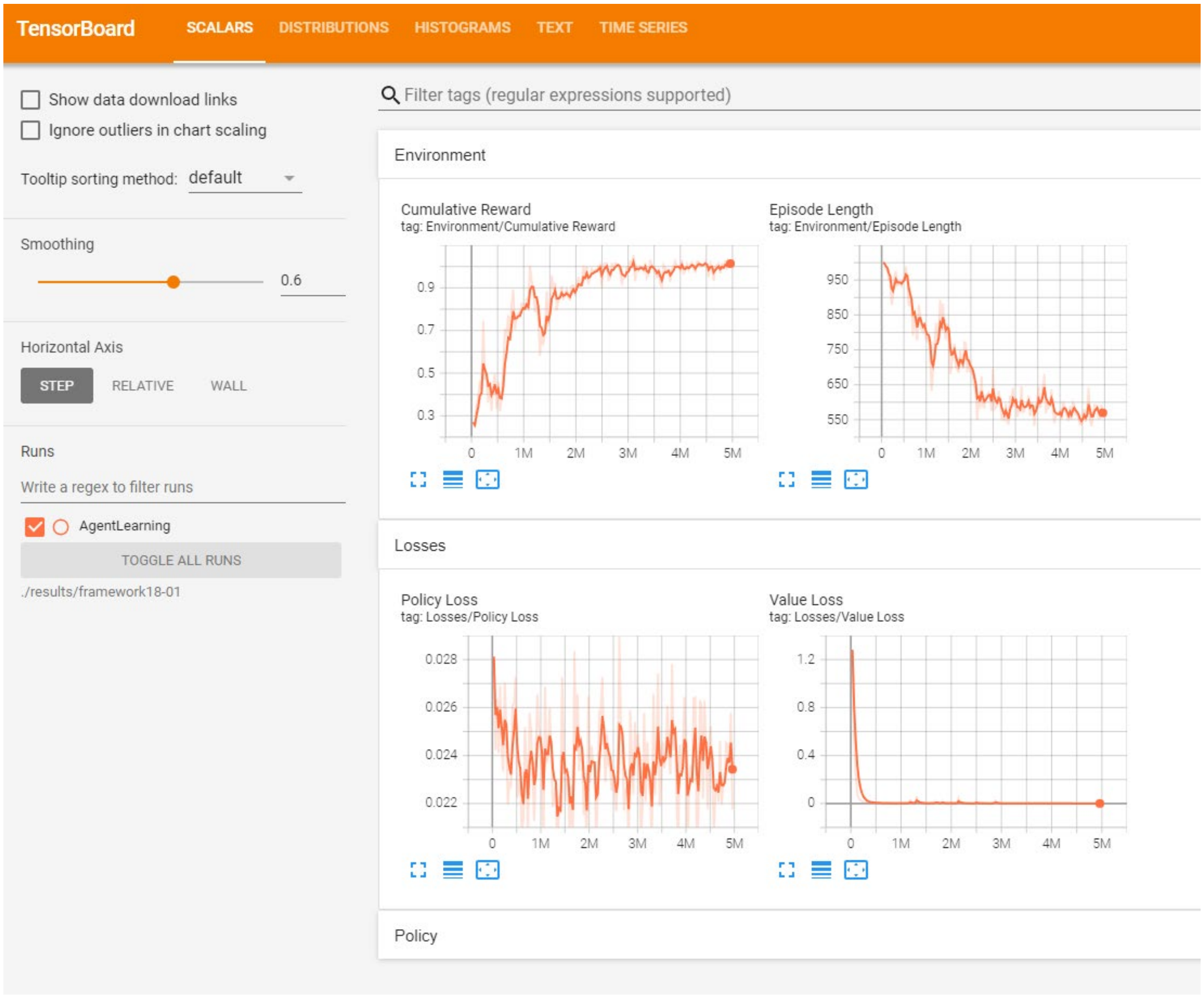
Write a regex to filter runs

☑ ○ AgentLearning

./results/framework16-1

Figure 17: Repeat test to verify DRL overfitting problem

Several additional training tests were performed under Test 17, none of which led to a better training solution. Discovery during Test 17 led to the conclusion that proper settings are required for environmental observations, ray cast sensing, and visual sensing to make training work correctly. Test 17 results led to revisiting the simplified learning sessions previously examined. These simplified training tests were expanded to simulate a structure much closer to the drone use case. These discoveries were then applied and reconfigured in training of agent baseline in Test 18. The results from Test 18 are shown in Figure 18.



☐ Show data download links

☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ AgentLearning

TOGGLE ALL RUNS

./results/framework18-01

Figure 18: Baseline agent training results

Results for cumulative reward, episode length, policy loss and value loss were well correlated. The cumulative reward followed an overall pattern of effective training. The duration of training took a much longer time on the order of about 2.5 million training steps before reaching the desired goal. The overall curve of the cumulative reward was also very unsteady at times, dropping at two or more distinctive points in the process. This was assumed to be noise throughout the entire training process. The result from Test 18-01 however, did provide what appeared to be a good inference model to work with for testing the baseline agent.

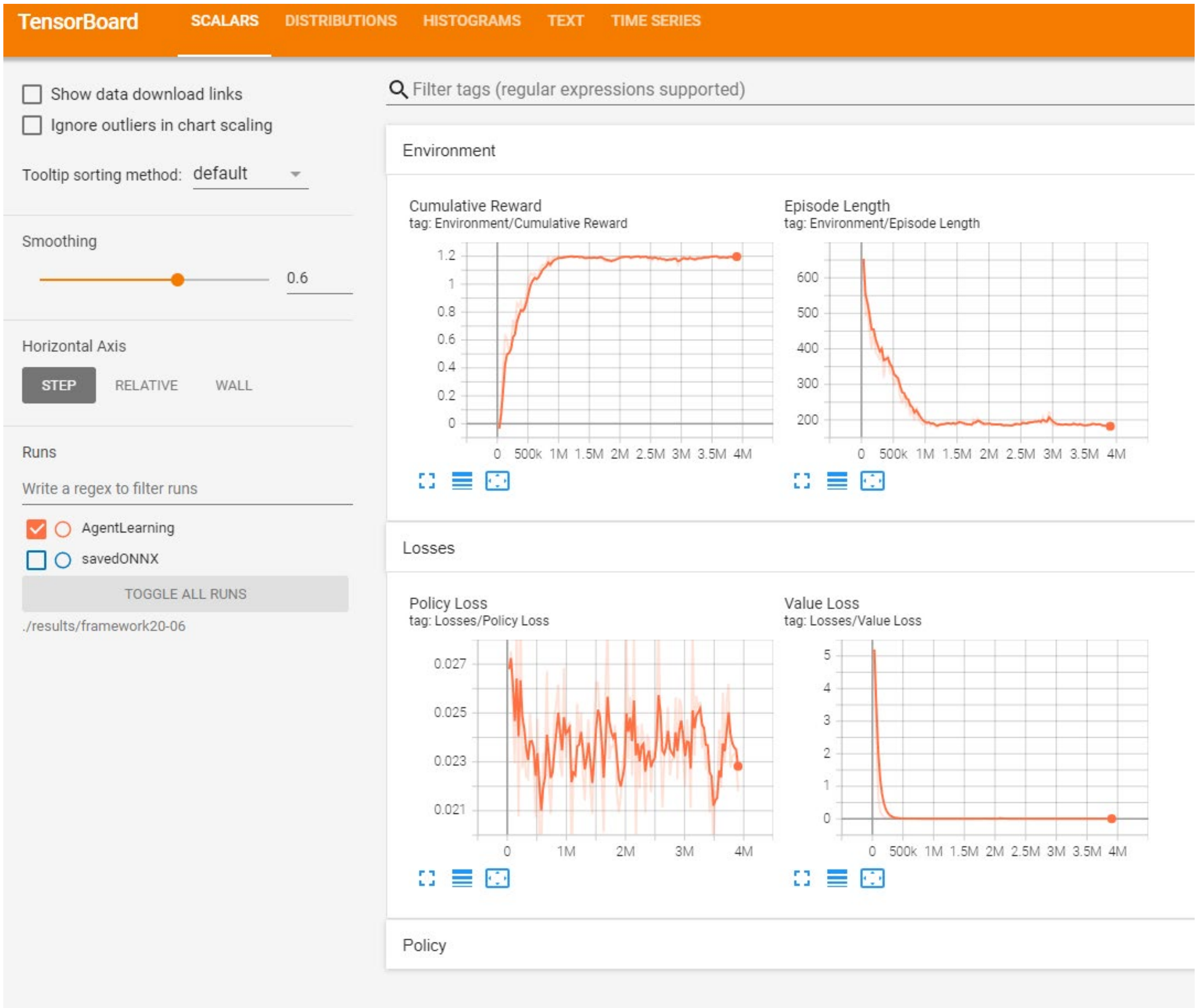
The inference model for Test 18-01 was integrated into Test 19-01. Test 19-01 was used to test the inference model within the same synthetic environment in which training occurred. In ML-Agents, a ONNX file is created at various points during training. These ONNX files represent the saved inference models during training of the agent. They are then used within the agent framework for further testing and validation. Modifications to the Unity application were made to transition from training to testing and evaluation of the inference model. During Test 19-01, the instrumentation for successful gate passages was used as a quick metric to evaluate the inference model. This test allowed the drone to try to run through the racecourse at least four times. Of interest was the number of consecutive gates passed through, the number of consecutive laps achieved, and how many restarts were required during the journey to pass through 109 gates in total. The total number of successful gate passes corresponds to an equivalent of four laps of the racecourse. Note, there were a total of 27 gates for each lap in the racecourse. The list below illustrated these results.

- Number of complete laps: 0
- Number of consecutive laps: 0
- Maximum number of consecutive gates: 19
- Minimum number of consecutive gates: 12 (at termination – could be higher)
- Total number of restarts during inferred model evaluation: 6

These results indicated that no single completion of the racecourse, totaling 27 gates, was ever accomplished by this inference model. At best, the agent could successfully go through three quarters of the course before crashing. It was also noted that for the specified duration of 109 successful gate pass throughs there was 6 restarts that occurred. The expectation for the baseline agent was to achieve at least two consecutive laps. These findings fall short of this expectation.

Another test, Test 20-06 was run to train the agent to be used as the baseline. The concept was to improve upon the inference model if possible. The approach taken was to provide a smaller ratio of penalty to reward during the learning process before resetting to the next episode. For Test 18-01, a very small penalty of -0.002 was given for each crash of the drone and the size for the reward value was +0.1. For Test 20-01 the values were adjusted to -0.1 for the penalty and +0.3 for the reward. The general idea was for the agent to learn to keep from crashing into other objects including the structure of the gates. The agent was also to learn to go through at least three or more gates successfully before an episode reset.

The results for Test 20-06 are shown in Figure 19. Here the data correlates for the behaviors seen in the graphs for cumulative reward, episode length, policy loss, and value loss. For cumulative reward the rise to a desired reward value of 1.2 occurred just over one million steps in the learning process. This was three times faster than that of Test 18-01. There were no major fluctuations in the reward values over the training. Noise levels seemed to be minimal and insignificant overall. This result provides good indications for the learning process without any indications of overfitting of the DRL algorithm. The next step taken was to validate the inference model created in Test 20-06.



☐ Show data download links

☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

☑ ○ AgentLearning

☐ ○ savedONNX

./results/framework20-06

Figure 19: Improved baseline agent training results

The inference model for Test 20-06 was integrated into Test 21-01, which was used to test the inference model within the same synthetic environment for which training occurred. The same modifications were applied to transition from training to testing and evaluation of the inferred model. During Test 21-01, the instrumentation for successful gate passes was used as a quick metric to evaluate the inferred model. This test allowed the drone to try to run through the racecourse at least four times. Of interest was the number of consecutive gates passed through, the number of consecutive laps achieved, and how many restarts were required during a selection of 109 gate passages. The total number of successful gate passes corresponds to an equivalent of four laps of the racecourse Note, there were a total of 27 gates for each lap in the course. The list below illustrated these metrics.

- Number of complete laps: 0
- Number of consecutive laps: 0
- Maximum number of consecutive gates: 21
- Minimum number of consecutive gates: 13 (at termination – could be higher)
- Total number of restarts during inferred model evaluation: 5

These results indicated that no single completion of the course, being 27 gates, was ever completed by this inferred model. At best 78% of the course was successfully completed by the agent passing through the gates. The lowest percentage of course completion was 59%. It was also noted that for the specified duration of 109 successful gate passages, there were 5 restarts that occurred. The expectation for the baseline agent was to achieve at least two consecutive laps. These findings fall short of this expectation; however, they are slight improvements over those obtained in the similar test of Test 19-01.

Upon consideration of the outcome for both Test 19-01 and 21-01, several more tests within the test Framework 21 were made. These additional tests were performed to determine if there were other reasons that better results were not observed. This series of tests examined potential areas that might affect the length of an episode which might have terminated a given drone navigation through the racecourse earlier than expected. The result led to a conclusion that there was a logic error, which was difficult to discover. In ML-Agents, the parameter called MaxSteps gets set to allow the maximum steps that any training or execution of an inferred model would take. In the previous processes, the MaxSteps was set to 0. This, per ML-Agent documentation, was considered to allow the inference model to keep running forever. The problem that was discovered was that the ML-Agents user interface program was overriding this value and setting it back to 5000 steps. This would be enough training steps to run through 16 to 20 gates before calling an end to the episode. This was seen in the results of both Test 19-01 and 21-01. This change in logic by the application interface was corrected and Test 21-08 was performed.

Test 21-08 provided the expected results based on the cumulative reward results of inference model of Test 20-06. This was the same conditions for the model formulated in Test 18-01 and evaluated in Test 19-01 respectively. Since Test 20-06 provided a better cumulative reward performance than Test 18-01, its configuration was used in Test 21-08 to validate the inferred model of Test 20-06. The list below illustrated these metrics.

- Number of complete laps: 4
- Number of consecutive laps: 4
- Maximum number of consecutive gates: 109
- Minimum number of consecutive gates: 109
- Total number of restarts during inferred model evaluation: 0

The results of this test, based on corrections made to faulty logic in the framework, achieved a 100% performance in each of the categories. These results indicate that the inference model of Test 20-06 was the best inferred model to apply to the baseline agent in this study.

City Scene with SDR+ Environment Tests on Baseline Agent

The next series of steps in the experimentation workflow evaluated the trained baseline agent in the city scene environment to evaluate if variance to the environment would affect the performance of the baseline agent. These tests were run using the inference model obtained from Test 20-06. Each test used a different environmental setting but having the same structure of city scene environment. The lighting test, Test 22-01, provided a significant change in the lighting of the scene. The inference model scored equivalent to Test 21-08 results. The agent passed through all gates over four laps without crashing. The modified environmental setting changes for the autumn test (Test 22-02), rain test (Test 22-03) and snow test (Test 22-04) all obtained equivalent performance to the lighting test. No degradation in the agent's performance was observed for all cases of environmental change. These changes were representative of the changes associated with naturalistic domain randomizations. Analysis of the setup for the baseline agent appeared to indicate an assumption that certain embedded training elements in the agent provided greater influence than the learned visual sensor training element. Two factors support this assumption. First, environmental observations and ray casting sensors are learned observations early in the training process and reach their full potential early in the training process. It was assumed that they provided a much stronger influence on the agents learned inference model. The visual sensor requires more compute power and speed and requires a much longer training cycle. There was insufficient training to allow the model to evolve to a functional state for visual sensing. This assumption corresponded to the environmental change results for the Test 22 series where visual based domain randomization had no impact upon the agent's performance. Because this is directly related to the use of naturalistic domain randomization, additional examination of the use of the visual sensor during training was deemed necessary. Additional experiments were performed to validate these assumptions of the visual sensor as well as the overpowering influence of the ray casting sensor and environmental observations.

The next experiments were designed to strengthen the influence of the visual sensor. The set of tests that followed were attempts to train an agent solely on the use of the visual camera sensor provided in Unity's ML-Agents. The separation of the visual sensor from environmental observations and ray casting sensing provided the best means to understand the influence of the visual sensor. It is often difficult to formulate what actions an agent should take based on the results of the visual sensor feedback. Typically, a CNN based visual model trains from the set of spatial regularities that occur within the learning process. The difficult part was to determine how the agent was to be rewarded or punished based upon the actions taken. Several tests were performed to arrive at an acceptable training process for just the visual sensor. Tests were performed and modifications made based on lessons learned during training along with applying the specification provided by ML-Agents for visual sensors. Once a good balance in visual sensor parameters and associated actions were determined and training in Test 27-14 was run.

The results of visual sensor Test 27-14, at 700,000 steps during the training process, provided insight to the compute time required for training via the visual sensor. Early in the training process it was clear that the visual sensor-based agent was having difficulty and was learning very slow. The agent's behavior was more erratic and less effective in locating and passing through the gates. Analysis indicated that the training process for the visual sensor was more intensive than initially considered.

This intense increase in training was based partially on the added number of observations required for the visual sensor. For a typical environment observation technique, this is usually based on the requirement to have one to several vectors of data of size 3 or 4 with a common number of observations ranging from 2 to 12 observations. Similarly, ray cast sensing can provide one to many ray casts. Each ray cast provides a result of intersection testing with some object in the environment. This typically involves a vector of size of 3 values for each ray that is cast with typical range of rays being used from 1 to 32. This constituted a range between 3 and 96 values for ray casting data. The ML-Agents recommendation for the visual sensor was the use of images of size 84x84 pixels. If the gray scale option is selected this results in an 84x84x1 resolution which is equivalent to 7,056 observations. If RGB values are used, then the number of observations becomes an 84x84x3 resolution equivalent 21,168 observations. Simple visual sensor cases can use a resolution as low as 32x32 or 8x8 and still perform adequately if they are used for simple visual use cases. Complexity grows as the scene becomes more complex which is the case for this study. Considering that these observations are made frequently during the learning process, a significant increase in computation time for the learning process became evident.

The computation cost of using a visual sensor in Unity was lacking or inadequate in expressing these costs. Initial research suggested that the use of the visual sensor appeared to be a promising approach. In most cases the tutorials and online documentation led to the assumption that visual sensors could be extended to more complicated use cases. Only very simple examples were used in these examples and tutorials or in the reference materials. Several more recent tutorial references (Kelly, 2021; Schreurs, 2019) had mentioned that the cumulative reward results for a simple ML-Agent tutorial had not even come out of the negative results until after some eight million steps had occurred. They referenced training that ran for 21 days and had not even finished. The results were shown to be improving but at a very slow rate. It was estimated that it might take three hundred million or more steps to achieve the desired training. It was further considered that for some more complex examples it could run into the billions of steps required to achieve a desired level of inference model behavior. The analysis of Test 27-14 agreed with these estimates and concludes that an extensive amount of compute time and computer resources would be required to continue to use the visual sensor in this work.

The following time resource estimates, based on computational resources, are provided as linear extrapolations from the findings of Test 27-14:

- Actual: 700,000 steps took 14.5 hours to run.
- Estimate: 1,000,000 steps = 20.7 hours
- Estimate: 5,000,000 steps (expected number of steps to run) = 130.6 hours (4.3 days)
- Estimate: 100,000,000 steps = 2,070 hours (86 days, minimum limit)
- Estimate: 1,000,000,000 steps = 20,799 hours (862.5 days) (2.36 years, maximum limit)

Analysis of the estimates from the results at 700,000 steps, assuming a linear relationship, clearly indicated a very expensive compute time for using the visual sensor. The addition of expensive compute intensive hardware with parallel processing would be required to reduce the compute time to generate satisfactory results from a visual sensor. This assumes an equivalent experimental framework similar in nature to one used in this work. Based on the estimated extreme compute times when using visual sensors versus when a full training cycle runs without using a visual sensor, it was considered to take too long to run and thus out of scope for this work. Use of Unity's ML-Agents, as stated, had never provided any known examples in research literature or from internet-based tutorials which only used very simple tests. The few simple cases that did exist only provided minimal insight into the degree of computation required for the visual sensor. The internet tutorials references provided (Kelly, 2021; Schreurs, 2019), were recent additions and were not available at the decision time to move forward with Unity's ML-Agents as the backbone for this work's experimental framework.

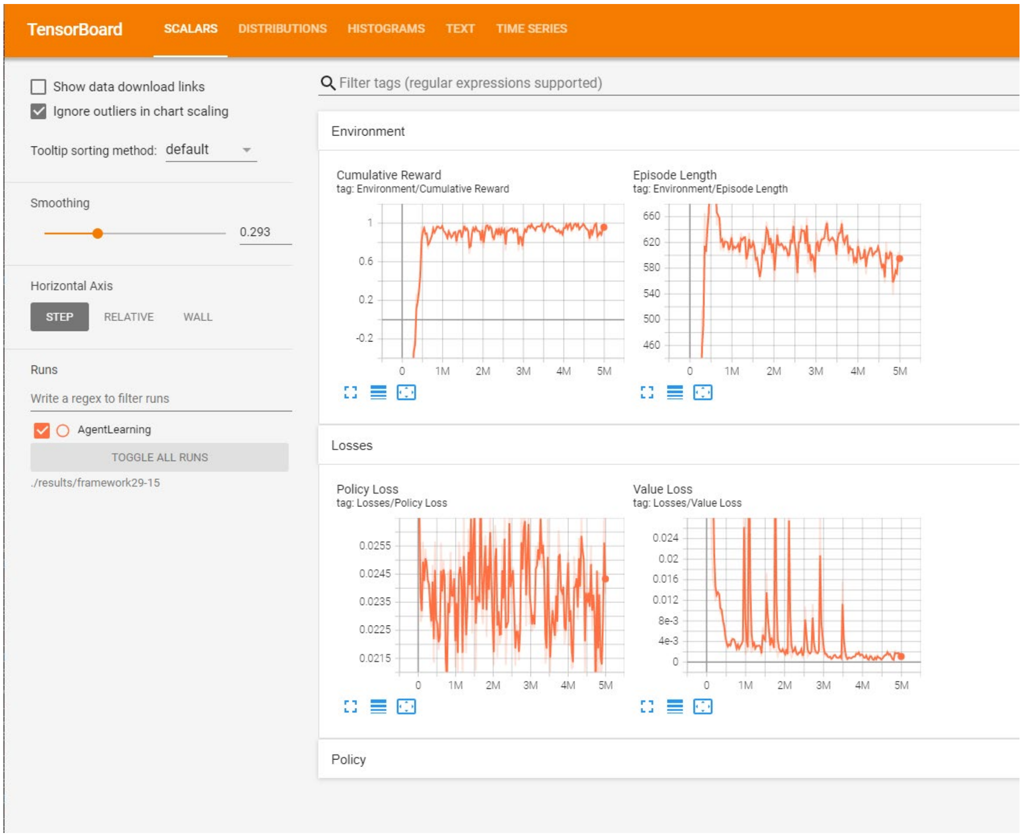
The implication from these results indicated that the visual sensor for the given drone racecourse use case would provide no benefit. This implied that without the use of the visual sensor, the naturalistic domain randomizations would not provide any valid experiments since no visual metric was measured. It is worth noting that the experimental framework for the city scene did provide the means to explore the path which led to this finding of these results. From a non-visual standpoint, the SDR+ technique still holds value for exploration for more physical-based domain randomizations such as the movement, placement, shape, size, and partial obscuration of the elements in the scene. Primary physical element considered were the gates, race path and any path obscurations along with methods to modify these within the synthetic environment.

This course of action, the use of physical-based domain randomizations, was planned for the remaining experiments. The city scene was reset to provide an initial level of racecourse modifications to be used in this manner. The original approach of this work planned for physical-based domain randomizations, therefore the city scene environment could be reused and reconfigured for these changes. The initial set of experiments were performed to validate the effect of physical domain randomization modifications on the baseline agent. The second set of experiments were performed in the city scene with the goal to explore the physical-based domain randomizations applied during the training of a SDR+ agent and to then to compare its trained results to those of the baseline agent results.

City Scene Physical DR Modifications Effects Applied to Baseline Agent

The city scene environment was utilized again to test the baseline agent with the inferred model. The purpose for that test was to evaluate the baseline agent performance for physically based variations in the environment. Several more experiments were performed to evaluate if this physical-based variance to the environment had any degradation effect on the baseline agent's performance. The premise was that if some degree of degradation to the baseline agent is seen under these conditions with the baseline agent, a comparison could then be made to agents trained using the physically based SDR+ to validate both the methodology and assumptions.

The first test, Test 31-07, made adjustment to the framework to remove 12 of the 27 gates from the racecourse. Each of the remaining gates was placed in the same location for 15 of the gates used during training. The objective was to see if variance through fewer gates caused degradation. Several preceding tests were performed until the logic of the framework performed correctly when moving from a training environment to the validation environment. An inferred model was developed in Framework 29-15. The results for this training session are seen in Figure 20.



□ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.293
Horizontal Axis: STEP RELATIVE WALL
Runs
Write a regex to filter runs
☑ ○ AgentLearning
./results/framework29-15

Figure 20: Additional training of baseline agent

Some additional measures to improve upon the inferred model were attempted but did not produce results better than in Framework 20-6. The trained model, however, reached a cumulative reward of 1.0 at several points but took much longer to get to the point and a lot of noise was observed as the model continued to train. This was then used in Test 31-07 to determine how well the baseline agent performed with the variance of changing the number of gates in lap as well as the spacing between those gates. A larger spacing was always applied because it increased the level of difficultly to reach and pass through the gates. The inference model of Test 29-15 was used in this case, and it did not really differ that significantly even if Test 20-06 was used. It is noted that baseline inference model from Test 20-06 was used in the remainder of the tests for the influence of environmental variation on the baseline agent's performance.

The results from Test 31-7 are listed in the table below:

- Consecutive completed laps: 0
- Completed laps of 15 gates: 2 times
- Maximum number of consecutive gates: 15, occurred 2 times
- Minimum number of gates entries: 1 which occurred 11 times
- Two consecutive gates passed: 13 times
- Fourteen consecutive gates passed: 3 times
- Total number of crashes occurring before 108 gates completed: 272
- Distribution of consecutive crashes (number of consecutive crashes): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 29 and 41

Analysis of these results indicate a definite degradation in performance. The baseline agent ran 109 consecutive gate passes without a single crash. In this case, 272 crashes occurred with near even distribution for the consecutive number of crashes that occurred up to about 17 crashes. Two outlier cases were for 29 and 41 consecutive crashes. There were two cases of full laps completed and three cases short of one gate from being a full lap.

The next test, Test 32-2, examined the effect of relocating the same gates used in the training to slightly new locations. Each gate in the racecourse was relocated either further down or up the racecourse path along with movement to the left or right and some rotations of gates. The premise was that this change in gate positions would provide location-based variance to that of the baseline agent training environment. This test, like all tests in this section, were performed under the condition of isolating types of variations. This was done to provide a better understanding each type of physical domain randomization that was applied.

The results for Test 32-2 are better in terms of agent performance than the results for Test 31-07, however, still fall short when compared to the baseline agent running the same course it was trained in. The results of Test 32-2 are shown in the following list:

- Consecutive completed laps: 0
- Completed laps of 27 gates: 0
- Maximum number of consecutive gates: 22, occurred 1 time
- Minimum number of gates passed through 1, occurred 2 times
- The number of consecutive gates ranged from 1 to 22 which a even distribution for consecutive gate passages (1, 3, 7, 10, 14, 16, 21, 22)
- Total number of crashes occurring before 108 gates completed: 23
- Distribution of consecutive crashes had a maximum of 5 and minimum of 1. 70% of consecutive crashes came in pairs.

Analysis of these results indicated a definite degradation in performance, however, better than the performance of Test 31-7. A 91.54% improvement in crashes over Test 31-7 occurred with total of 23 versus 272 crashes. The agent in this test did not complete even one lap but was close to doing so twice. It is also noted there were 27 gates per lap. Fewer crashes occurred but less in number of consecutive gate pass throughs happened. These results indicate that the relocations of gates also provided degradation to the agent’s performance. Given these results, it was not necessary to test for dynamic movement of the gates since this case would provide worse results.

Test 33-3 provided the test for the effects of physical-based domain randomizations. The tests in this subsection are solely to evaluate if an agent is affected by these randomizations. This sets the case for the reality gap also being illustrated within a synthetic environment. Test 33-3 adjusted the synthetic environment by changing out the shape of the gates. A total of three new shapes were utilized; a hexagon, a rectangle, and a rectangle with a 45 degrees rotation or diamond shaped. Each of these gates maintained an equivalent entrance area for the drone to pass through the portal of the gate. An assumption was made, for this gate shape variation, that its effect would be the least of all the degradation tests. It was clear that this affect would be more impactful if the visual sensor was being used. The change in shape and color and size of the gate is more applicable in the visual range versus ray casting. The results of Test 33-3 are listed below:

- Consecutive completed laps: 4
- Completed laps of 27 gates: 4
- Maximum number of consecutive gates: 108
- Minimum number of gates passed through:108
- Total number of crashes occurring before 108 gates completed: 0

These results indicated that no changes occurred in the baseline agent's performance when the shape of the gates were modified as described. There is no impact for just changing the shape of the gate unless it causes a change it the actual passage causing the drone to learn to find the spot in the portal were no collisions with the edges of gates occur.

The next baseline agent validation test, Test 35-01, was to examine if the size of the pass-through portal of the racecourse gates may make some difference in performance. The assumption was made that larger portal sizing would not provide any potential degradation since the inference model performed at 100% at the current portal size. A decrease in portal size while still allowing room for passage of the drone agent was used as a basis. Three different portal size reductions were performed: at 75%, 60%, and 50%. Test 35-01 was run three times for each change in size to the portal. A good balance at the degree of performance degradations was observed within the three size reductions. Table 2 provides the data results from these tests. For a 75% reduction in gate or portal entry size resulted in a performance of 100%. This is equivalent to the baseline agent performance in the original synthetic environment. At 60% size reduction in the portal the degradation in performance became apparent although in the beginning state. A total of 5 collisions occurred with 4 of them being just past the first gate and the other just after completing just over one lap through all gates in the lap plus one. One more crash occurred then it ran through almost 3 full laps. The data indicates that run performed just below 100%. This indicated the beginning of degradation. A repeat of the test at 50% size reduction in the gate portal provided worse performance results. A total of 94 crashes occurred, most of which occurred after only one portal pass through occurred. No laps were completed There were a series of 3, 8 and 8 consecutive portals pass throughs that occurred during the test. The size reduction of 50% was indicative of the lower bound limit while slightly higher that 60% was the upper limit on performance.

Percent Gate Reduction	Full Laps Completed	Consecutive Gates (>1)
50%	0	3, 8, 8
60%	2 (1 gate short of 3)	28, 80
75%	4	109

Table 2: Gate size reduction metrics for baseline agent

The next test to validate degradation due to variation in the synthetic environment was Test 36-01. In this test path-blocking objects or hinderances were placed in or near the expected drone path. The concept was to place objects in the scene that covered a portion of the ground truth path while still allowing room for the drone agent to navigate around the object. In this test the goal was to determine if adding these path blocking objects into the scene would change the performance of the baseline agent. Two tests were run to verify the results. Each test indicated failure of the baseline agent to avoid the obstacles if the obstacles were placed closer to the trained flight path of the agent. The second run was performed to help verify this assumption. Placements of the obstacles which caused crashes were placed more directly in the path the agent. The second attempt moved these obstacles further out of the way. These results confirmed that the agent would pass by all objects which were sufficiently offset from the ground truth path while those placed directly in the path caused the baseline agent to crash. The analysis confirms that the baseline agent did not learn to avoid obstacles that were not originally in the training environment and place directly in the path.

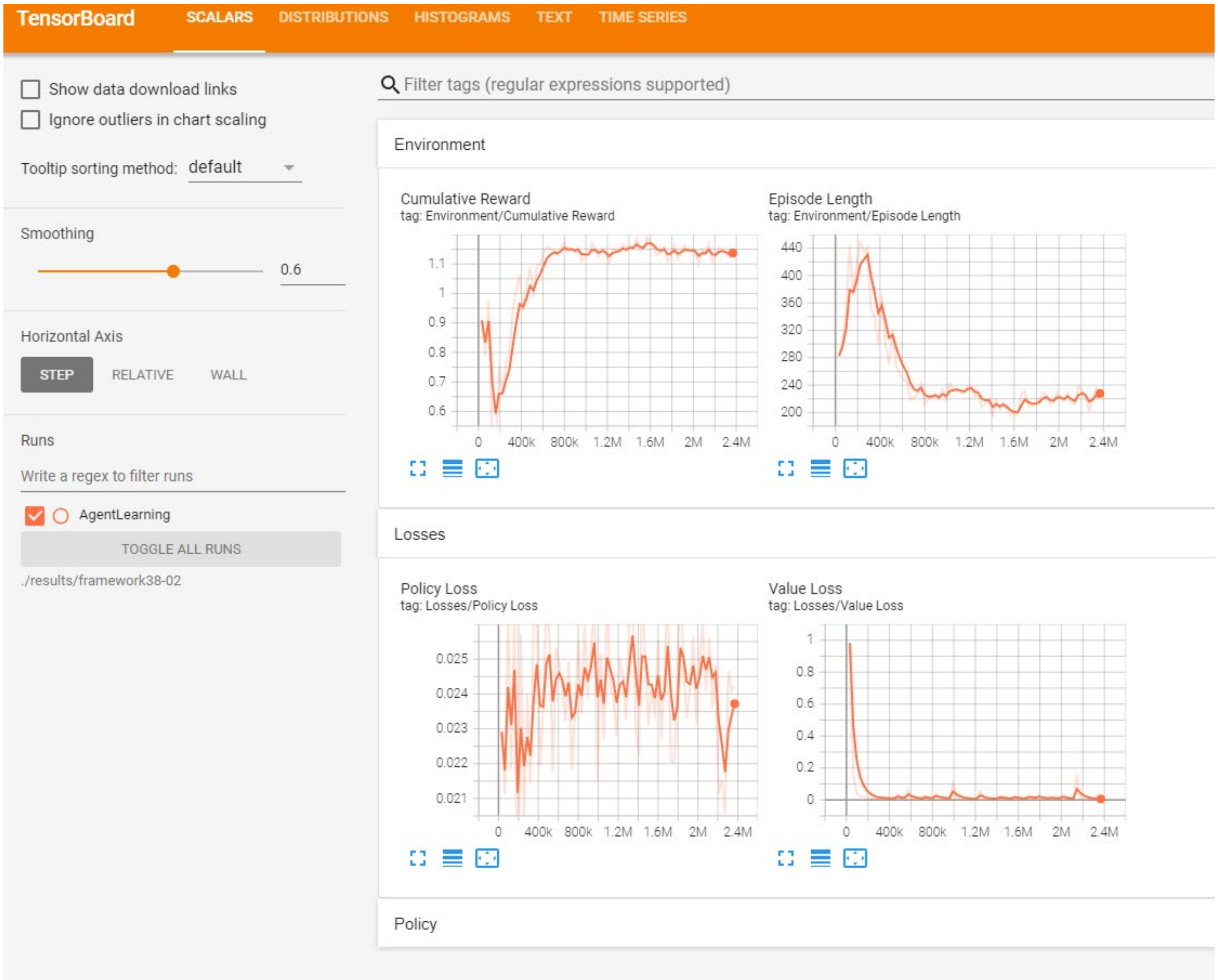
In summary, this section covered several tests to verify that variations to the trained environment caused a baseline agent to degrade in its performances. This provided the values to compare against when the SDR+ techniques are applied during training to an equivalent agent. This section covered tests, results, and analysis for the baseline agent as follows:

- Modifying the number of gates and spacing between gates: degraded behavior
- Modifying the locations/path static: degraded behavior (extends to dynamic case)
- Modifying gate shape: no degradation of behavior (considered for visual sensors)
- Modifying gate entry size: degraded behavior
- Adding obstacles in flight path: degraded behavior

City Scene Physical DR Modifications Training of SDR+ Agents

This section covers the training, testing, and comparison of results for the SDR+ trained agents. Comparisons were made against the baseline agent of the previous sections. The focus was applied to physical-based variations within the city scene synthetic environment. These physical-based variation were applied to vary five elements: 1) gate size, 2) placement of obstacle in the flight path of the drone, 3) number of gates for a single lap, 4) spacing between the gates, and 5) location of the gates and paths.

Framework 20-06 was modified to allow variance in the gate portal size during training. This was retitled as Framework 38 which was a common practice to preserve a working copy for each modification or test applied in this work. The original gate portal size was at 100%. The gate portal reduction was evaluated in reductions of 10% from the original size. The number of gates in the drone racecourse remained the same at 27 gates. During the training the sizing of the portal was modified between 100% down to 50%. This range was selected based upon the performance seen in the baseline agent. The results of the training for Framework 38 are illustrated in Figure 38-02. The training initially started out with high reward values but dropped by more than 30%. This was followed by a quick rise and achieve a steady state reward of 1.15 at about 800,000 steps. The training was run to about 2.4 million steps to verify. Episode length, policy loss and value loss plots were all in accordance with the results observed. The training was a bit more extensive by covering the range of gate portal size reduction from 100% to 50%. The baseline agent was examined only at 100%, 60% and 50%. No further data would be required for the baseline agent since the point of degradations only occurred in the 60% and lower range. Training for Framework 38 was modified to be able to run the 10% increments in reduction automatically during training and the results are shown in Figure 21.



□ Show data download links
□ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: ☑ AgentLearning
./results/framework38-02

Figure 21: SDR+ agent training for gate portal reduction

Test 38-02 provided results of 100% performance, all gates passed through for entire test, for portal sizes ranging from 100% down to 60%. At 60% size reduction the SDR+ agent ran through all 108 gates, or 4 laps, without a single crash. The baseline was below this level as illustrated in Table 3.

Consecutive Gate Passes	28	80	109	Laps
Baseline	1	1	N/A	2
SDR+	N/A	N/A	1	4

Table 3: Comparison of baseline to SDR+ agents at 60% gate reduction

From Test 38-02 for 60% gate portal reduction, the number of consecutive laps was half as much for the baseline agent. The baseline agent performance did have 80 consecutive laps along with one set of 28 consecutive gate passages. Test 38-02 also included a test for 50% gate portal reduction. For this case, degradation in the SDR+ agent was obtained in the results. These results are illustrated in Table 4.

Consecutive Gate Passes	1	2	3	4	5	7	8	9	Laps
Baseline	88	1	1	0	0	1	1	0	0
SDR+	17	1	4	2	4	1	3	2	0

Table 4: Comparison of baseline to SDR+ agents at 50% gate reduction

The SDR+ agent for the 50% reduction in portal size test failed to achieve any complete laps of the drone racecourse. In Table 38-02-50 the SDR+ agent achieved a better distribution for consecutive gate passages than that seen by the baseline agent under the same conditions. Figure 22 illustrates this distribution.

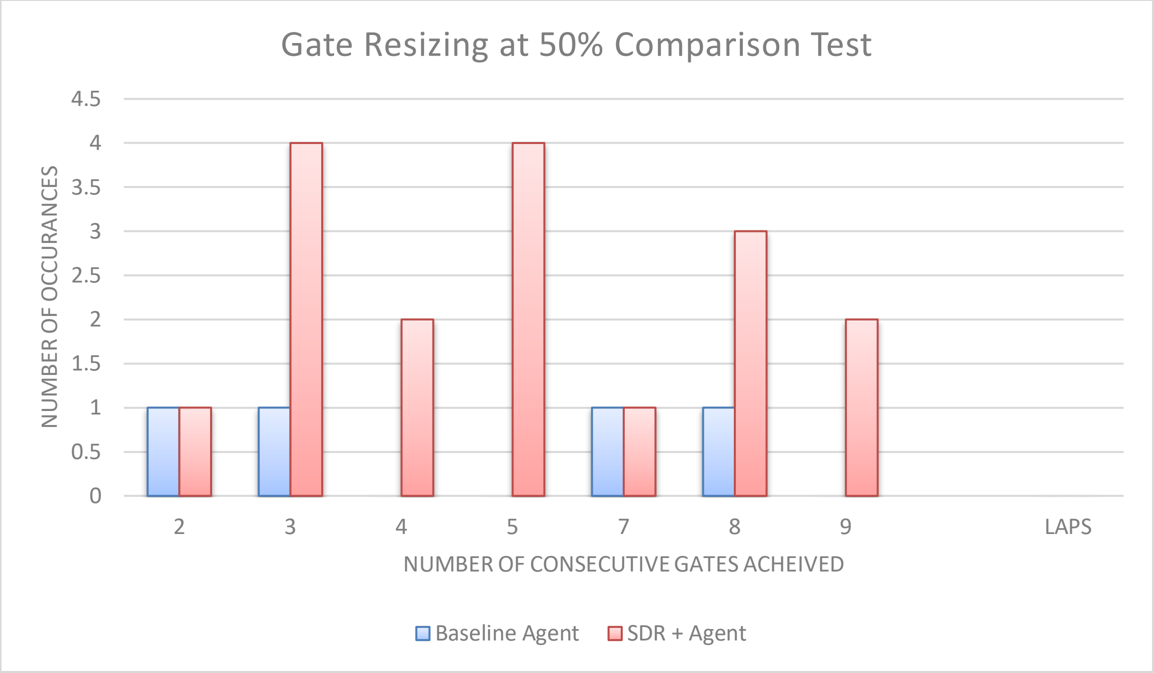
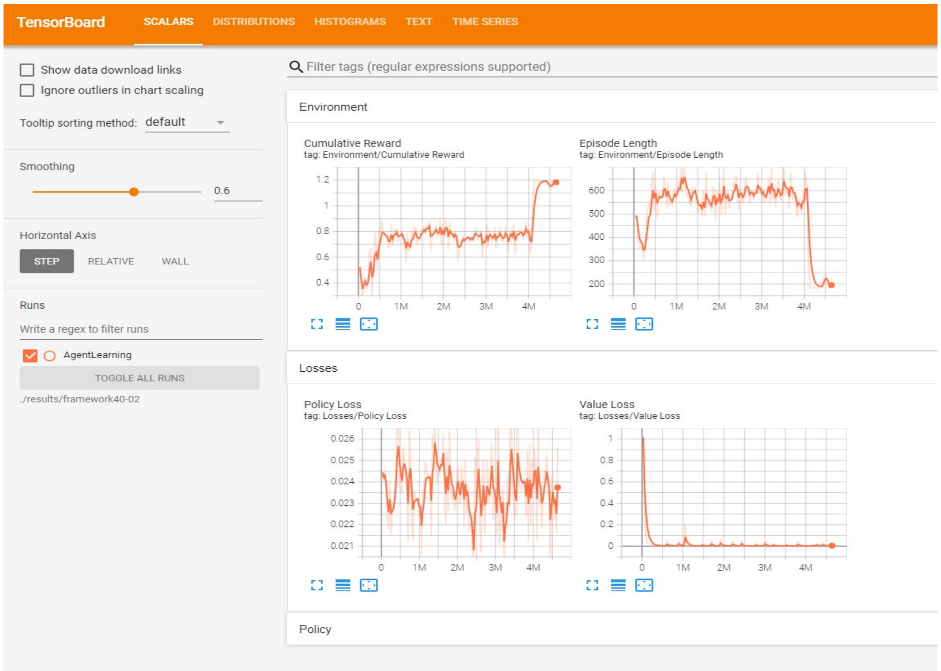


Figure 22: Comparison of SDR+ and baseline agents for 50% gate portal reduction

The baseline agent had 88 single gate passages. This data value demonstrates a crash after going through only one gate. This was reduced by 80.68% by the SDR+ agent which had only 17 occurrences of a single gate passage. That factor of roughly a 3.38 times improvement along with a better distribution at higher numbers of consecutive gate passages provides insight into the improved performance achieved training variance is applied.

The next physical-based DR variation was the placement of obstacles in the path of the drone agent. Framework 40-02 was used to train the SDR+ agent with variation based on obstacles. The training at approximately 500,000 steps reached a fluctuating plateau around a cumulative reward of 0.75. For this case, the accumulative reward for peak performance was expected to reach a cumulative reward of 1.2. As the training approached 4 million steps the cumulative reward had still maintained a cumulative reward of 0.75. Inspection of the visual representation of the virtual environment provided the indicators for the lower reward level. The drone agent would continually crash into certain gates which were directly in the path of the agent. This was confirmed during Test 40-02 by moving said obstacles off to one side or the other or less directly in the path between gates. Once this live training action took place the cumulative reward quickly increased to the expected reward level of 1.2. This is shown in Figure 23.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

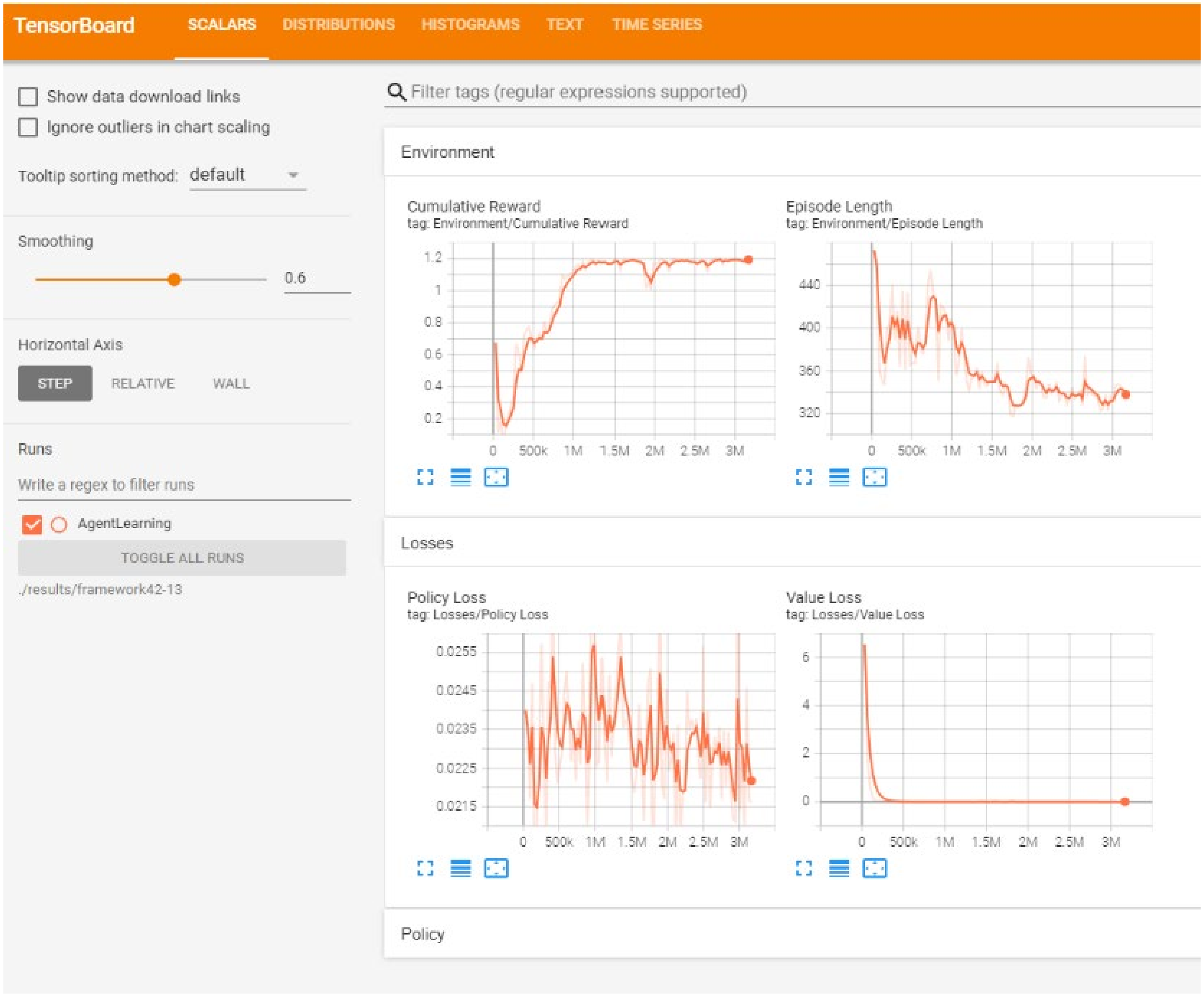
☑ ○ AgentLearning

./results/framework40-02

Figure 23: SDR+ agent training with obstacles and live training adjustments

The results of Test 40-02 were similar in nature to Test 36-01 for the baseline agent under the same use of obstacles. The data prior to the live training adjustments did show improvement over the performance of the baseline agent for the same conditions. There was a total of 8 obstacles placed in the agent's path. Under certain circumstances these obstacles were directly in the path and in other instances partially in the path or just an edge covering the ground truth path. The results for the baseline agent had 5 of the 8 obstacles that prohibited the agent from further advancement. For the case of the SDR+ agent only 3 of the 8 obstacles provided the same behavior. Examination of the visual representation of the virtual environment, it was seen that obstacles on the edge of the primary path caused problems for the baseline agent, however, the same was not true for the SDR+ agent. The SDR+ agent was able to navigate around obstacles these edge cases, thus the improvement seen in the results.

The next training, testing, and validation in the series of physical-based domain randomization was for the case of reduction of the number of gates used in each lap and corresponding changes in separation in the spacing between gates. Training was run on Framework 42-13 and tested and evaluated with Test 43-01. The results of the training are shown in Figure 42-13. The cumulative reward graph was very similar in response to the case for obstacles placed into the path of the drone agent. The reward reached steady state at about 1.2 million steps. This training took longer to reach a steady state than other training cases. The results also indicated a small dip in the reward just before 2 million steps, however, it recovered and returned to the approximate cumulative reward of 1.2. Episode length illustrated this slight change in reward value, but it did not provide any indications of degradation of reward during training rather more of a small perturbation in the data most likely due to stained or overtaxed compute resources.



□ Show data download links
□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ AgentLearning

TOGGLE ALL RUNS

./results/framework42-13

Figure 24: SDR+ agent training with reduced number of gates and spacing changes

Test 43-01 was run to validate the training for reduction in gates and changing of their spacing. The performance for the SDR+ agent was at 100%. It completed all 4 laps consecutively which included 109 consecutive passages through the gates. This was one of the expected results when running a trained agent within the same environment and conditions in which it was trained. In comparison, the results for the baseline agent did not perform as well. It is noted that with gate reduction a single lap consisted of 15 gates versus the 27 gates used in the other evaluations. Figure 25 shows the comparison between the baseline and SDR+ agents.

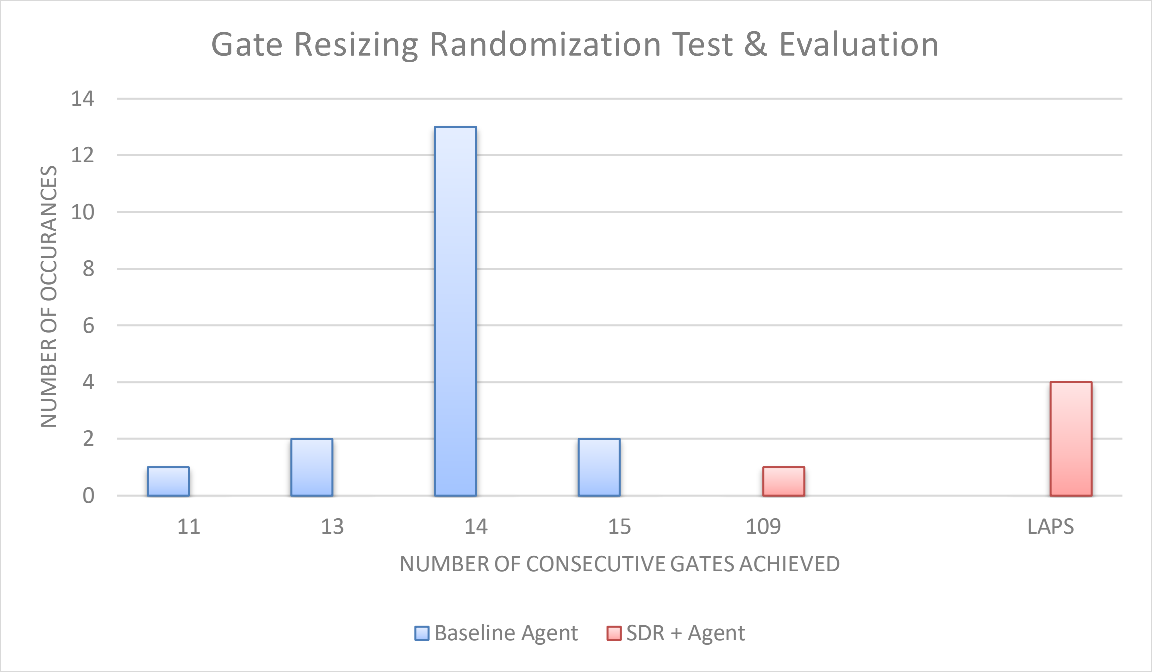
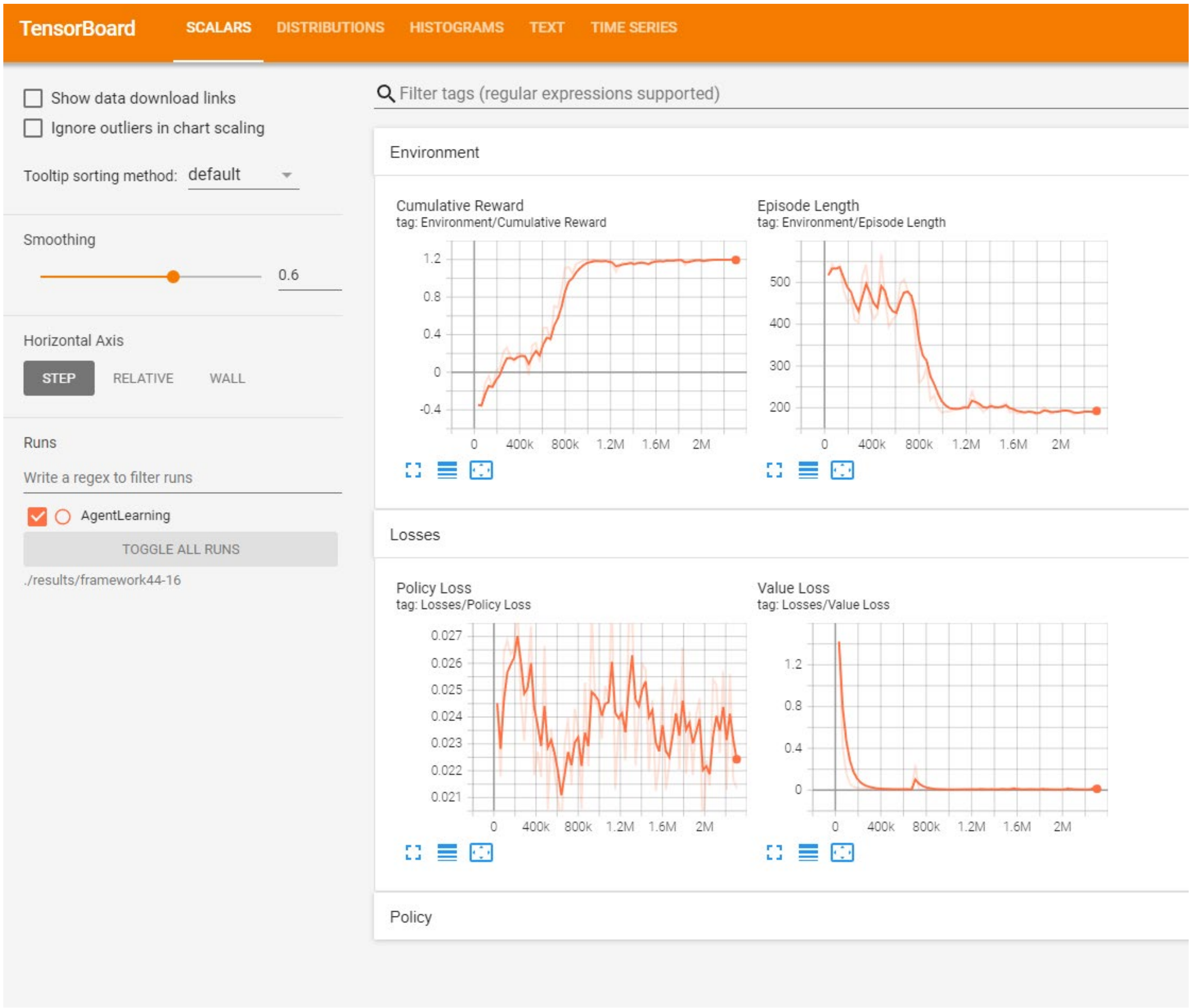


Figure 25: Comparison of SDR+ and baseline agents for gate reduction and spacing

Figure 25 illustrates an improved performance for the SDR+ agent because of the variance of gate reduction and increasing gate spacing during training. For this case the improvement was significant compared to the other physical-based DR tests.

The final physical-based DR training and evaluation experiment was for the movement of gates which also changed the path of the drone racecourse. Framework 44-16 was constructed for this experiment along with Test 45-01 to test and validate the experiment. Gate movement was controlled and reset at each episode. To maintain consistency and prevent the movement of gates into non-assessable locations the experiment moved each gate to one of four different gate positions. Each movement was then followed by a movement back to the original gate locations and path. This was repeated for a total of four new positions represented by changing the physical location either up or down, to the left or right, and then forward or backward along the course. Testing required changing from every episode to once every 100 updates during the simulation. Figure 26 illustrates the training of the SDR+ agent for gate and path movement.



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

☑ ○ AgentLearning

./results/framework44-16

Figure 26: SDR+ agent training with gate and path movement

The cumulative reward for this training resulted in a plateau reward value around 1.2. The training rate to reach this plateau took approximately 1 million steps. The training for cumulative reward was consistent with just some noise related changes in the values which were also seen in the episode length graph. The value loss graph also indicated one jump in response just before the main increase in reward values during training. Test 45-01 was used to test and validate the training under similar conditions in which the agent was trained. The SDR+ agent performance was slightly degraded in the evaluation of the inference model in the same training environment. Three laps were completed along with two large consecutive gate passages of 58 and 39. Total number of collisions during Test 45-01 was 8 with 6 of those occurring at the first gate at the beginning of the test.

A comparison of the baseline agent to the SDR+ agent was conclusive that adding gate/path movement variations in training improved the performance. From Figure 45-01 the training for gate movement was improved by achieving a higher number of consecutive gates along with completion of 3 consecutive laps. Analysis further indicated that a higher number of collisions for a single gate occurred at the beginning gate at the start of the test for both the baseline and the SDR+ agents. The baseline agent did not complete any successful laps and had a widespread distribution seen in the number of consecutive gates passages in Figure 27. The number of collisions for both were approximately equal at the starting of the test providing further indication of a problem when initiating the test.

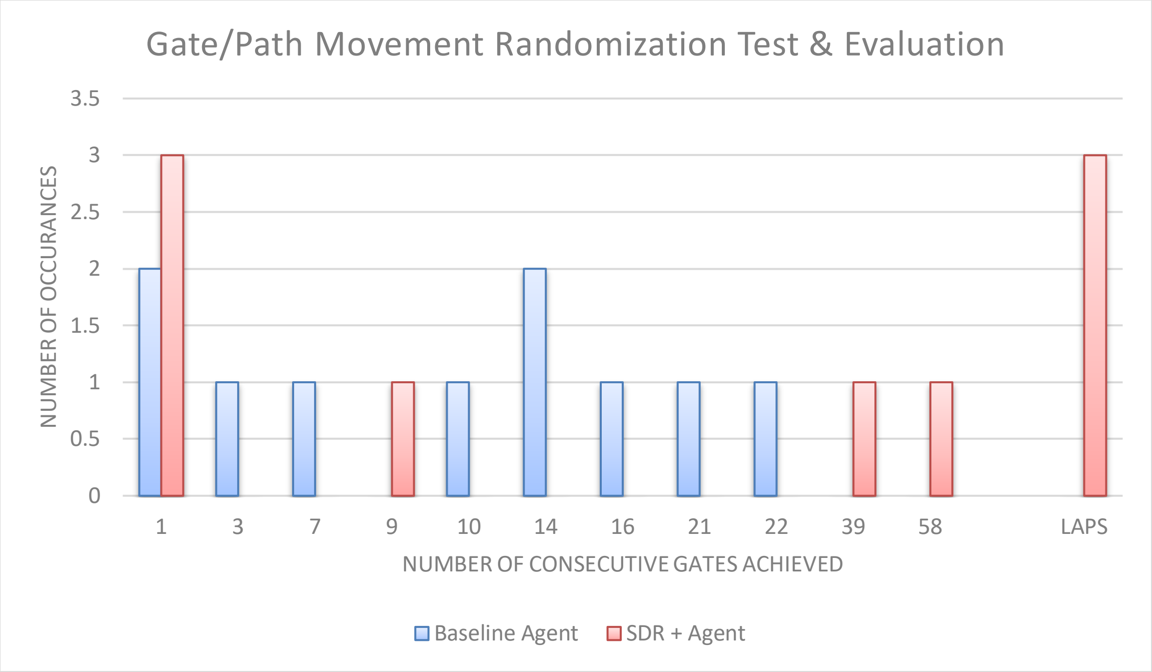


Figure 27: Comparison of SDR+ and baseline agents for gate and path movement

The findings for gate and path movement training indicated improvement in performance for training with this type of variation. The analysis of the data did show some degradation of the SDR+ agent during the test in the same training environment. One factor that may explain the difference was the problem at the start of the test at the first gate. If this is discounted, then only three collisions occurred which kept the SDR+ agent achieving 100% performance. All the SDR+ agent tests were run only once if it was a successful run. Multiple runs and averaging out the data may provide slightly different results, but it was assumed the same general pattern would be followed. With this assumption, no repeated tests were performed in this work.

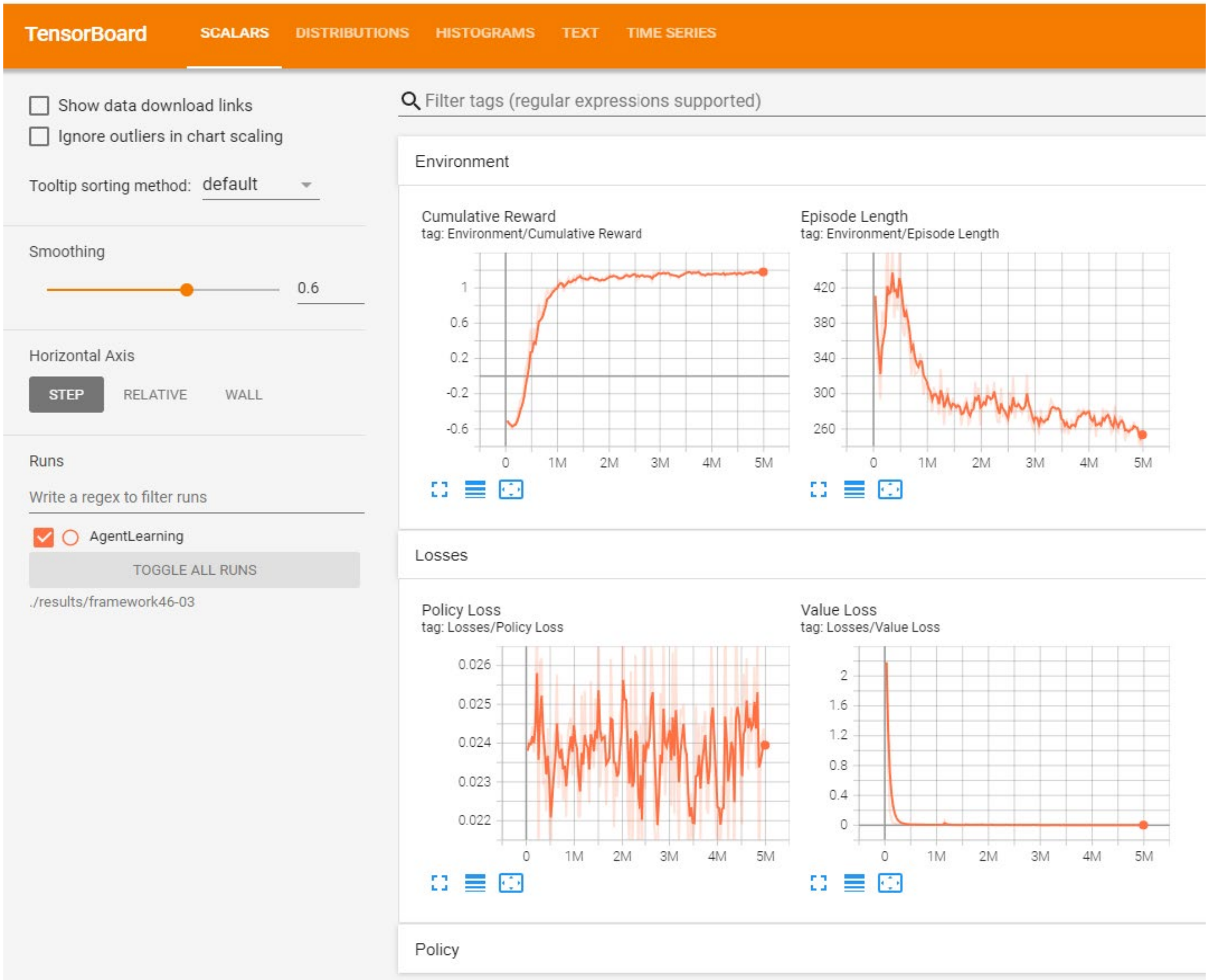
The next section examines if this can be generalized from one environment to another. Results showed that the experiments provide positive results when it was tested within the same environment in which was trained. The experiment of the next section explores whether this generalizes, or the agent only performs well under conditions for which it was trained.

Non-City Scene Combined Physical DR Modifications Generalization Comparison

This section discusses the results, analysis and findings of experiments leading toward testing for indications of generalization through use of the SDR+ technique. Prior sections in this chapter covered the results, analysis and findings of the experiments that could now be combined and then test in a different environment. Prior experiments provided the foundation for this experiment. The original argument for this work included naturalistic domain randomization. It is still assumed to play a significant part in the SDR+ methodology, however as stated, insufficient computing resources to overcome the amount of processing for visual based sensing would not allow further investigations down the naturalistic domain randomization path.

Based on the finding of the previous experiments for physical-based domain randomization, another set of experiments were also run in the city scene virtual environment to provide a SDR+ training baseline. This was then tested and validated for some portion of generalization in a new and different synthetic environment. First the new training framework was created which combined the effects of environment variation for: 1) gate portal resizing, 2) gate and path relocations and 3) number of gates used and changed spacing between gates. The obstacle blocking experiment was not included based on the findings. The modification of gate appearance and shape was also excluded because of a previous conclusion that it was tied closer to naturalistic domain randomization than to physical-based domain randomization.

Framework 46-03 was used to train a SDR+ agent in the city scene virtual environment. Dynamic gate portal resizing, dynamic gate and path placement, along with a reduced number of gates which were spaced at various distances were used as the physical-based domain randomization parameters. The results for this training are shown in Figure 28.



□ Show data download links
□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ AgentLearning

./results/framework46-03

Figure 28: Combined physical domain randomization training in the city scene

The training did not require much more in terms of training steps to achieve a plateau of near constant cumulative reward value of 1.18. The training was allowed to train to the pre-defined termination point at 5 million steps. A steady state reward value was achieved approximately after 1.2 million training steps. The episode length, policy loss and value loss graphs also corresponding to the expected training pattern. Overall, the training for the combined variance effects ran better than expected when it was a combination of dynamic physical-based domain randomizations.

The trained combined inference model was tested and validated first in the same city scene training environment under the same conditions. Framework 47 and Test 47-09 was used to perform the testing and validation of the combined environmental changes in the city scene using the same conditions in training. Prior attempts at the validation of inference model 46-03 indicated some results did not appear to provide the expected performance. Upon analysis of the testing and validation it was discovered that the rate of the changes in the physical-based domain randomizations was occurring at too fast a rate. In training the rate of change was based upon episode length, however, during training and validation the rate is determined by updates to the simulation. Several runs of Framework 47 were attempted, each had an increasing number of updates prior to domain randomization until an acceptable amount of time passed before the combined domain randomizations occurred.

Figure 29 illustrates the results of the testing and validation of inferred model for both SDR+ training test 47-09 and baseline training test 20-06 in the city scene environment. Analysis of these results indicate that the trained SDR+ model did not provide the perfect performance of 100% through the course. A total of 4 laps of a 20 consecutive gate racecourse were achieved with an occurrence of 60 consecutive gate passages and then 20 consecutive gates passages. Analysis and an understanding of the training explained the amount of degradation in performance as shown from the data in Figure 47-09. When a test or validation is running it assumes the entire course continues to run without stopping. During training an episode is reset when it has achieved a certain level of reward and then it resets to a new location in the course with the intent to provide training over the entire racecourse. This means that during training the same behavior could have been happening but not exhibited because of the need to reset episodes periodically during training. The analysis also shows the SDR+ agent outperformed the base line agents for 4 completed laps versus 1 as well as a distribution of consecutive gate passages at the higher end of the scale, especially for the case of 60 consecutive gate passages.

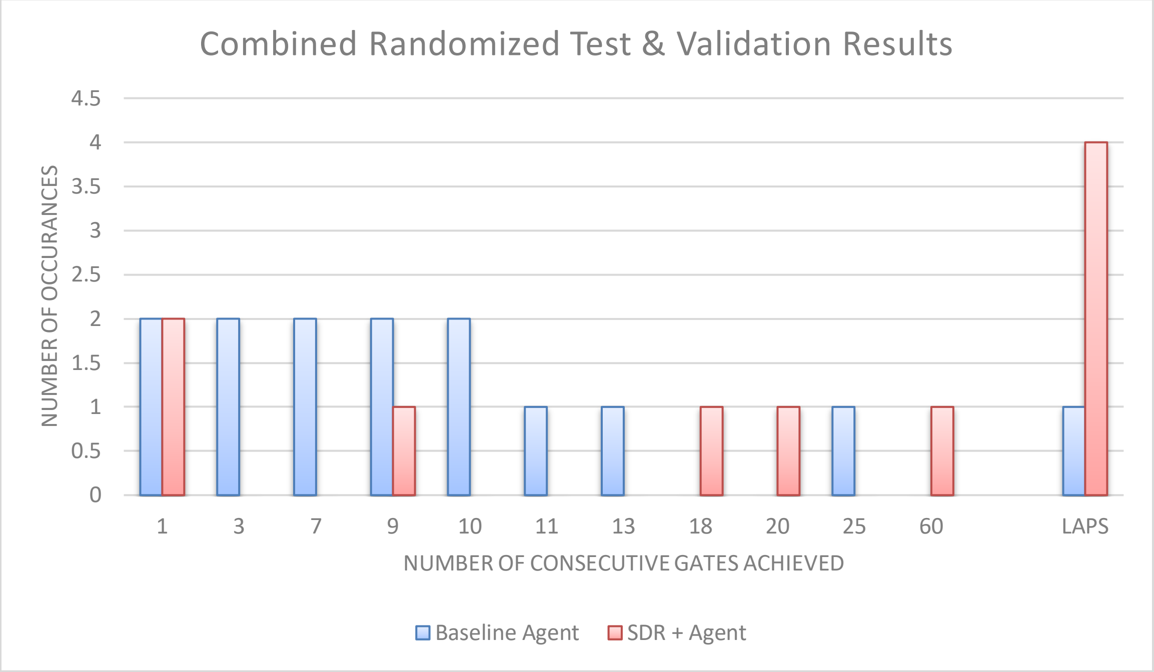


Figure 29: SDR+ and baseline agent test and validation for combined randomizations

The next experiment performed took the inference models of the baseline and SDR+ agents and inserted them into an entirely new synthetic environment. The same modified physical-based domain randomizations were used; however, some were modified from domain randomizations that existed in the city scene environment. The first change was the overall synthetic environment, which was changed from a city scene to a small mountainous island.

Images of these are provided in Appendix A. Boundaries were maintained as a rectangular cube around the island sides, bottom and top. The racecourse path was much longer and increased and decreased in elevation along the slopes of the island mountain much more than with the city scene. This added a new physical-based domain randomization to the course not used previously. This change was due to increased positive and negative slope changes in the racecourse, but also due to the gate orientations. The gates were reoriented to align with the new three-dimensional space in the mountainous island scene. This varied the city scene where gate placement and orientation were planar and only rotating around one axis as in the city scene. The same number of gates were used with the same gate portal size changes, gate location and path movement. The spacing of the gates were spaced further apart across the new mountainous island scene.

The trained baseline inference model from Framework 20-06 and the trained SDR+ inference model from Framework 46-03 were tested and validated in this new synthetic environment. The results are shown in both Table 5 and Figure 30.

Consecutive Gate Passes	1	2	3	4	5	6	7	8	9	18	laps
Baseline	82	8	2	0	1	0	0	0	0	0	0
SDR+	13	5	4	1	3	1	2	1	1	1	0

Table 5: Data result for generalization test of baseline and SDR+ agents

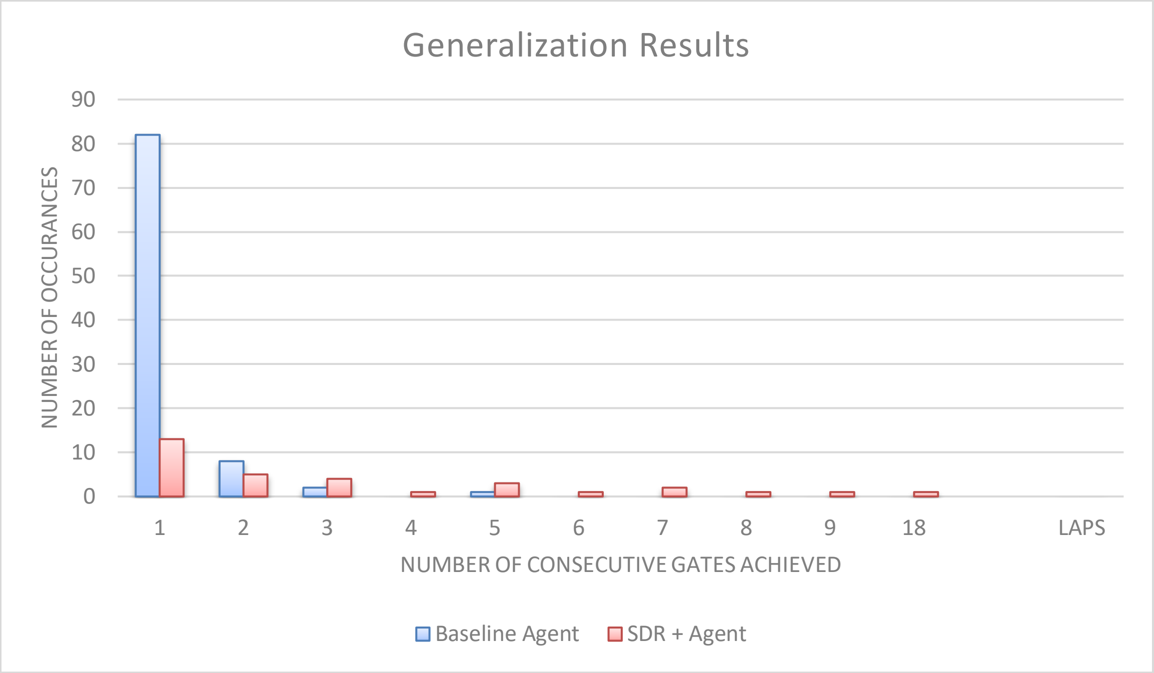


Figure 30: Generalization test of SDR+ and baseline inference models

The total number of a single gate passage occurring in the baseline inference model was an order of magnitude greater than any of the other occurrences. Analysis indicated this was due to the greater difficulty that the baseline agent had in getting through most of the gates. The baseline agent did not get beyond 5 consecutive gate passages. The distribution for the SDR+ inference model was much better and extended up to 18 consecutive gate passages. Neither inference model was able to complete one or more laps of the course. Table 49-10 was presented alongside of Figure 30 for clarity on the results. For this case Figure 31, shown below, was created to better illustrate the difference in findings between the SDR+ and baseline agents.

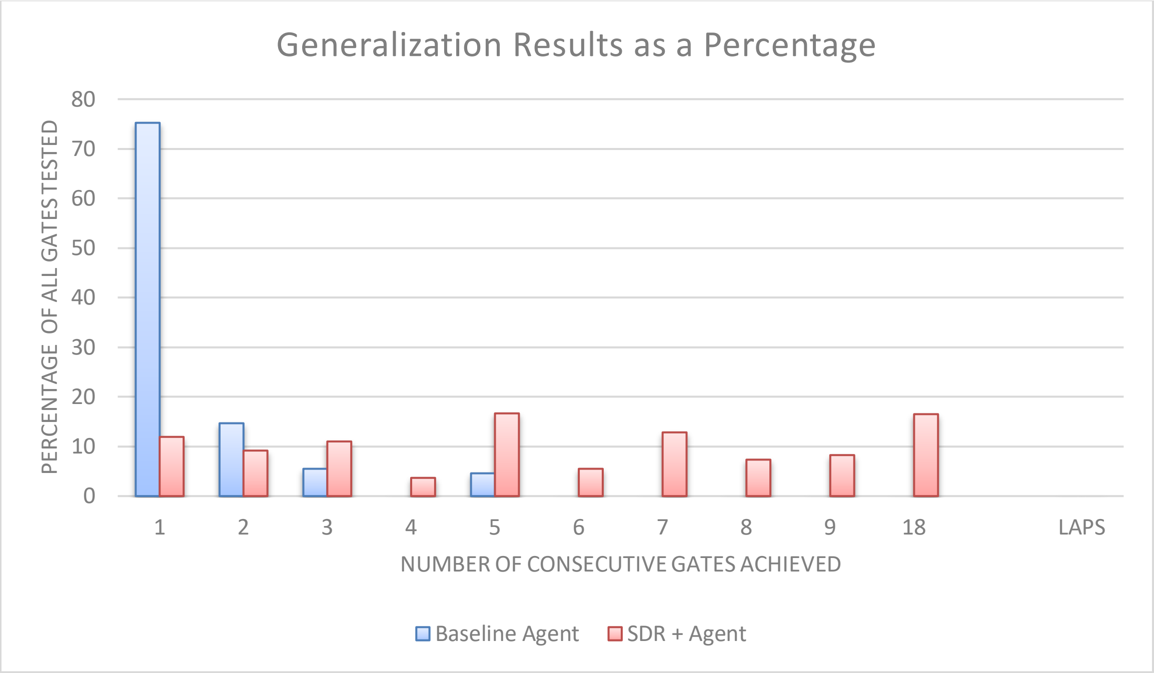


Figure 31: Generalization test of SDR+ and baseline inference models based on percentage

Figure 31 illustrates a higher and better distribution of consecutive gate passages for the SDR+ inference model when view as a total percentage of the gate passages that occurred in the test. A very high percentage of just single gate passages occurred for the baseline inference model. Analysis indicates that the baseline agent performed poorly in the new synthetic environment with the physical-based domain randomizations used. Analysis for the SDR+ inference model illustrated a better performance over the baseline inference model; however, consecutive gate passages were only at 66.7% of completing even one successful lap of the racecourse. Improvement was seen by training for some of the variations presented in the environment, but that data is in a range between being inconclusive to almost no findings which could verify that some form of generalization occurred when using the SDR+ technique.

Summary of Results

The results of this work provided some interesting insights about the constitution of DRL PPO-based algorithms, the integrations of ML-Agents with Unity's application with regards to synthetic training environments, and the SDR+ technique to support the hypothesis of bridging the reality gap and improving generalization for synthetically trained agents. An experimental workflow followed a path in the development of a series of frameworks that permitted the training, testing, and validation of a SDR+ agent. The workflow began with the experiments of a simple example of an agent trying to reach a target in constrained space. The experiments supported which set of configurations would be used in this work. A set of synthetic environments were constructed which allowed elements of the environment to be varied for training, testing, validation, and comparison purposes. A generic synthetic environment constructed from a city scene was used to provide a baseline for comparison of the variational effects on both an agent trained in such an environment and a series of agent trained with these variations. The workflow proceeded to develop, test, refine and modify both the baseline and SDR+ agents as part of the process to determine if the hypothesis of the SDR+ techniques was supported or lacked the support to validate the hypothesis. The results obtained through this experimental workflow constitutes the remainder of this section.

The experimental workflow started with the simple example of the agent locating an object and moving to where the object was located. The purpose was to determine the configuration to use for the DRL PPO-based agents. It was determined from these experiments that the best options were a combination of the environment observations, ray cast sensing and visual sensing as inputs into the agents learning. It was determined that use of ML-Agents does require environment observations at a minimum to work properly. Various forms of neural networks were available for the visual sensor, but the ML-Agents recommended simple neural network was selected because it performed as well as the other types of versions of neural networks and was the recommended configuration.

The simple example framework was also used to determine if context-aware elements could add to the agent's learning ability. An experiment based on the level of reward by the distance to the target had an unexpected result of the agent learning to move back and forth across the distance boundary to gain more reward. The agent learned to vacillate on the reward boundary to get reward versus moving closer to the target. Adjustments were made to use Unity's box collider which made no difference. The box collider became a blocker between the agent and the target. Ray cast sensing failed because of these objects. Plane colliders were also tried with the same results.

A re-examination of the simple example framework was performed and allowed to run for a longer period. The previous results and the extended period unveiled another anomaly of overfitting. Overfitting is a problem with machine learning and is less common for DRL agents. Additional literature research revealed that DRL overfitting does occur, and this fact matched up to the data. The modification to the context-aware elements did help to stabilize the overfitting but still provided lower cumulative reward values.

Further experiments were formulated to determine if a new approach to using the context-aware element could limit overfitting while achieving a better performance of the training. This methodology moved part of the reward received by the agent away from the context-aware elements and changed them to provide just direction information for the agent. The simple example experiment was also extended by moving from two-dimensional to three-dimensional space. This provided a closer simulated movement with regards to the use case of the drone racecourse. Additionally, the reward and penalty values were adjusted to finally arrive at more consistent and acceptable results for the trained agent.

With a better configuration for the agent, the experimental workflow moved to the synthetic environment of interest called the city scene. The city scene was set up to adjust the environment for several naturalistic domain randomization settings along with a few physical-based domain randomization methods. In porting the agent to this new synthetic environment, several conditions prohibited better training, testing, and evaluation of the learning agent. These conditions were: improper configuration settings, programing logic errors, and overriding of the number of steps to take before resetting an episode.

With a valid baseline agent, further experimentation was performed to determine the effect of environment domain randomizations on agents. The baseline agent was trained in the springtime city scene environment which provided good lighting with minimal naturalistic domain randomization effects. The baseline agent was then subjected to an autumntime season which included changes in coloring, sky type, falling leaves and different lighting in the scene. A rain-based environment provided obscuration of rain with dark cloud cover and different lighting in the scene. A snow environment provided the obscuration of snow, a change in lighting and the larger visual effect of scene objects covered in snow. The final environment change was to a night scene with city lights turned on providing a very contrasting lighting in the scene. Each of these conditions were run and the baseline agent had a 100% performance rating for passing through all gates over the course of four laps. These results did not align with the hypothesized outcome. No degradation in the agent was observed. The analysis determined that the visual sensor used was not sufficiently trained and thus did not provide the behavioral influence upon the agent. Essentially all the training influence was due to the environment observations and the ray cast sensor. Further analysis and more current literature research revealed that the ML-Agents visual sensing capability would require resources and training times that would far exceed the scope of this work. The visual sensing capability was deemed ineffective and unusable for this work. Naturalistic domain randomization relies heavily upon the visual component of the agent. Given this finding, the use of naturalistic domain randomizations was dropped from this work along with the use of the ML-Agents visual sensor. Further examination did reveal that the work could effectively continue with just the use of physical-based domain randomizations such as changing the gate size area where the agent must pass through, the location and orientation of the gates, a change in the path, a change in the number of gates, spacing of the gates, and adding obstacles in the flight path.

With these experimental changes the workflow was adjusted for the use of just physical-based domain randomizations. The baseline agent was then tested in the same city scene synthetic environment with the specified variations. Each of these physical-based domain randomizations provided degradation in the performance of the baseline agent. This aligned with the hypothesized outcome for applying environmental variations to an agent not trained for those variations. It was noted that the experiment for obstacles placed in the flight path failed for the baseline agent. This trained agent was unable to navigate around the obstacles. It was assumed this was partly due to the lack of ability given to the agent to perform better navigation. Navigational capabilities of better adjustment to speed, ability to reverse direction and the ability to move away from objects and then return to the computed path could have supported better results. Such navigational adjustments were inhibited by the black box nature of the ML-Agents DRL algorithm. The data feedback required to adjust the navigation was inaccessible.

The experimental workflow then moved to the training of agents where they were subjected to various forms of physical-based domain randomizations during training. This SDR+ agent was first trained in the city scene environment to establish an inference model. The physical-based domain randomization incorporated dynamic gate opening resizing, relocation of gates and paths, and changing the number of gates used in a lap as well as their spacing. All these effects were used simultaneously during training. Good training results were obtained which then allowed the inference model to be tested and validated in this same training environment. It was determined from the results for testing and validation of the SDR+ agent, that it had a lower performance than expected as was seen in the case when testing and validating the baseline agent. Analysis indicated a portion of the degradations was due to the difference between the episodic training of the agent and the simulation updates for testing and validation under the physical-based domain randomization settings. In training, episodes are reset once the agent reaches a certain cumulative reward or penalty value during that episode. It then resets and moves the agent to a new location on the racecourse and continues with the same training to gather experience over the entire racecourse. During testing and validation, the use of episodes is replaced by the agent running all the time and not being reset. The agent when learning would pass through several gates and when successfully achieved the proper reward value it would reset. The agent was learning the racecourse in a piecewise fashion. When testing or evaluation the agent, the agent must continue to fly through all the gates without any resetting. An assumption was made that these two distinct behaviors caused the fault of the agent to pass through as many consecutive gates. It had not learned to traverse the entire course rather just portions of it at a time. This same condition was also seen in the baseline agent.

The final step in the experimental workflow was to move the SDR+ agent with its inference model into a totally new synthetic environment. A mountainous island was used for the synthetic environment. Testing and validation of the SDR+ agent was performed in this new environment. The steeper slopes of the island's mountains provided an additional physical-based domain randomization not applied in the city scene. The environment required the gates to be oriented in a three-dimensional fashion versus the more two-dimensional orientation in the city scene. The agent was made to navigate over steeper uphill and downhill slopes. The gate spacing was also rearranged with a longer path and longer and shorter spacing of the gates. Both the baseline agent and the SDR+ agent was tested and validated in this dynamically varied environment. The results of the baseline agent indicated a poor performance. A large portion of the gate passages occurred for only one gate and the highest number of consecutive gates passed through was just five. The SDR+ agent also performed relatively poor, however, much better than the baseline agent. There was a better distribution of consecutive gate passages and at higher levels of consecutive gate passages. The performance was below the performance seen in the city scene where it was trained. The attributed reason for the SDR+ agent performing better than the baseline agent was due to the additional training of physical-based domain randomizations. It is inconclusive and more likely doubtful that an improvement in generalization of the agent's behavior occurred. The training helped the SDR+ agent's performance in the new environment but there was no way to determine if even a small incremental improvement in generalization occurred based on the outcome of the results.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The results and analysis of this work provide several conclusions regarding the use of the SDR+ technique to further bridge the reality gap. The reality gap which occurs when AI agents are trained in a synthetic environment. Several conclusions are presented.

The use of synthetic environments to train agents provides an innovative method to apply modifications to the environment and explore the agent's reaction to these variations. A good number of synthetic environments were explored in this work. The city scene environment provided an encapsulated area for a DRL agent to explore and exploit which is a common behavior of DRL algorithms. Use of synthetic environments made it easier to modify for a new set of conditions than would be possible if limited to real-world environments. This was illustrated in the city scene environment. In this environment, five naturalistic environments were established which include visual changes to the racecourse gates. There were four physical-based items which were also easily modified in a dynamic fashion during training, testing, and evaluation. Switching between such variations in the environment was readily supported by to the Unity application and its C# scripting API. The ability to provide these variations in the environment was critical to this work because of its dependencies upon domain randomization to achieve the desired outcome.

The use of the existing DRL PPO-based algorithm, which was not modified to achieve results, provided a stable baseline for the agent's capabilities. It was determined that the configuration parameters associated with the algorithm needed to be determined. This was successfully accomplished through a series of experiments on a simplified example. In the experiments of the simplified example, it became clear that environment observations were an essential element in the configuration. Including ray cast sensors and a visual sensor with the environment observations provided the best overall results. These configuration parameters were successfully transferred to the full model of the simulated drone racecourse.

Context-aware elements were proven to be successful when used as guides and for the improvement in the rate of training for the agents. Complications with their use was discovered during experimentation. A common phenomenon seen in DRL-based agents occurs when the agent learns to gain reward in a manner that does not contribute to the objective. This was exhibited in the simple example where the agent learned to vacillate over reward boundaries to increase its cumulative reward. Generally, this behavior would prohibit the agent from performing the intended goal. This work explored the ability to modify the context-aware elements by removing a good portion of the reward it provided and instead providing guidance or direction for the agent's next action. Guidance came in the form of providing a corrective direction towards the next target which was the gate or path in the racecourse. Guidance was also used in the ray casting sensor to determine if certain objects in the environment would cause a penalty rather than a reward to the agent. This change in the structure of the context-aware elements proved to be successful and the erroneous training behavior ceased to be observed. Another problem discovered with context-aware elements was their effect on DRL agents leading to overfitting. When rewards were provided to the agent via the context-aware elements and the training was allowed to run sufficiently long, a drop in the performance was observed. The added reward by the context-aware elements were generating the case for overfitting. Overfitting was not observed after the change in both the guidance and the adjustment to values for rewards and punishments were made to the context-aware elements.

The validation of the experiments of this work relied upon the development of a baseline agent. It was hypothesized that the baseline agent, when trained in a generic synthetic environment, it could reach the same performance level achieved during training. This was accomplished by using its inference model in the same training environment. The baseline agent successfully achieved equivalent behavior when re-run in the same environment using the inference model. This met the criteria of establishing a baseline metric so that experiments which incorporated domain randomization could verify their performance relative to that of the baseline agent. The baseline agent's performance in a domain randomized experiment exhibited degraded performance. The performance exhibited by the baseline agent aligns effects commonly seen in the reality gap problem for synthetically trained agents. The type of behavior by the baseline agent supports the concept that it was used as a metric to when compared to the performance of the SDR+ agent. It also supports the concept of being able to verify these results all within a synthetic environment and avoid porting it to real world experiments for validation.

The Unity applications along with its programing interface successfully allowed the development of experimental frameworks which were able to inject variations into the environment consistent with the domain randomization principle. This allowed training of SDR+ agents using one or more types of domain randomizations. This work examined both single and combined applications of domain randomization during experimentation to examine the effect of each and in combination. The results demonstrated that the domain randomization training improved performance of the SDR+ agent over the baseline agent. In some experiments some degradation in performance of the SDR+ agent was observed. Analysis indicated that the differences were due to problems with the agent behavior at the beginning of the simulation as well as the navigation capabilities of the agent. The problem associated with the beginning of the simulation was determined to be caused by the initial placement and orientation of the agent. There existed several layers of transforms within the agent and all its components including the camera object which was attached to provide a view of the agent moving through the course. An initial setting of the transforms for position and orientation provided an initial disorientation behavior in the agent. This effect only applied behavior at the start and the first gate in the course. Proper adjustments to the initial position and orientation transforms removed this problem. Regarding the navigational issues it was determined ML-Agents is considered a "black box" in terms of getting feedback from its sensors to better control the agent's navigational capabilities. This prohibited changing the agent navigation capabilities to adjust for finer movements such as learning when to slow down or speed up, when to turn sharply or not so sharply, and the ability to stop and reverse the motion to avoid some obstacles. Such navigational abilities are left to future work albeit an important capability based on the outcome of the results of this work.

There were two main conclusions to this work. First, that a synthetic environment can be constructed which allows the successful training of agents when domain randomization and context-aware elements are used. Second, which was unsuccessful, was the attempt to determine if an agent could generalize by learning to accept and adapt to variability not seen in the training. This implies that it was trained with various forms of domain randomization but learned from that to accept and adapt to others via that training. These conclusions are interdependent in the sense that one builds upon the other. Each of these conclusions were be addressed individually, however, what constitutes the makeup of these conclusions is addressed first.

Both conclusions are derived from the experiments performed. There were four primary types of experiments. The first experiment type was for training. A baseline for training an agent was established with the springtime city scene environment which provided the most basic environment. The performance of the trained baseline agent was then subjected to the second type of experiment. This second type took the inference model obtain in the first type and tested it in the same training environment. This determined the performance level achieved by the baseline agent during the training process. This same process was repeated for the SDR+ agent with the addition domain randomizations and context aware element guidance used during the training. The test and evaluation experiments for the SDR+ agent was performed under the same training conditions which included the domain randomization. The third type of experiment tested the baseline agent in the test and evaluation environment of the SDR+ agent. The purpose was to determine if the baseline agent's performance did degrade under the influence of domain randomization. The final type of experiment provided a new and different synthetic environment for testing and evaluation of both the baseline and SDR+ agent. This environment combined and added more domain randomization effects to arrive at the second conclusion. The inference models form the city scene training environment were used in this new environment.

The first conclusion was shown in the results and analysis that a synthetic environment can be constructed which allows the successful training of agents when domain randomization and context-aware elements are used. Building the framework on Unity and its ML-Agents module provided most of the heavy lift which permitted not only the construction of the virtual environment but the ability to manipulate the environment with domain randomizations and context-aware elements capabilities. ML-Agents along with C# scripting allowed the integration of a DRL PPO-based algorithm to successfully interface to a drone agent and allowed that agent to interact with the environment. Result data was analyzed proven to show that framework worked for each of the experiment types stated. It also showed the training worked under normal conditions and when conditions were modified via domain randomizations. The baseline agent performed well in its own training environment. It performed worse when run in the same environment with domain randomization applied. The SDR+ agent performed well within its own training environment. These results the successful construction use for experimentation of the framework.

The second conclusion, which was unsuccessful, attempted to determine whether an agent could generalize by learning to accept and adapt to variability not seen in the training. This conclusion was built upon the first conclusion and associated experiments. In this case the experiment was used to determine if any form of generalization was achieved by training with domain randomizations but being tested with other domain randomization beyond the training environment. The poor performance by the baseline agent was worse than seen in the city scene environment with domain randomizations applied. This validated the new environment and the change in domain randomizations. Basically, the baseline agent performance was further degraded to almost the point of being non-functional. The SDR+ agent's performance was also degraded but remained functional. It was expected that the SDR+ agent's performance would degrade in this new environment but not to the level indicated in the results. It is noted that it was also inconclusive that some form of generalization did occur. It was assumed than the agent performed well under the domain randomizations used in training in this new environment. It is also unknown if disparities between the city scene and the new environment had effects that degraded the agent's behavior. It is also uncertain if the episode resetting during training caused the agent to perform even worse when it only was training over small segments of the course during training. The episodic reset during training was understood to be the proper form of training under ML-Agents. Future work might study methods which increases training coverage to allow the agent to train over the entire course as well as multiple laps.

Implications

The emphasis of this research work was to examine and leverage an existing work of SDR (Prakash et. al., 2019) with the goal to demonstrate some level of improvement in the generalization of DRL agents. This work modified the SDR approach by adjusting the learning mechanism during training and provided another use case example of drone navigation. The SDR and DR (Tobin et. al., 2017) works are recent research works and limited to the study of robotics and image recognition. This work expanded upon this via the application for a drone agent navigation through a racecourse. Adding additional use cases to the body of work supports the concept of domain randomization scalability to other applications.

This work was solely based on experiments performed in synthetic environments to support the validity of their use in training of agents. The workflow process, the number of experiments performed, and the adjustments required demonstrated the utility of using synthetic training. The ability to modify, adapt and perform a variety of features or functions was demonstrated in the development and use of the experimental frameworks in this work. Such capabilities would either be too costly and difficult to perform in the real world.

This work added evidence that domain randomization use during training does allow an agent to learn behaviors and adapt to variations of the same kind. In the DR work (Tobin et. al., 2017) this was demonstrated by the improvement in a robotic arm to determine the location and manipulation of objects. For the case of the SDR work (Prakash et. al., 2019), this same principle was applied to the identification of cars in videos. This work demonstrated that using domain randomizations during training also provided conditions in improvement of a drone agent to navigate through a racecourse when such variations were present. Again, this implies another addition to the body of work in demonstrating the effectiveness of using domain randomization in training.

This work demonstrated that the use of domain randomization during training could help the agent to learn to accept variability when the variability was trained via domain randomization methods. The results did not indicate that a form of generalization occurred where the agent learns to adapt to variability in the environment for cases where no training was applied.

One problematic finding of this work was the inability to use a visual sensor. The problem was exhibited during experimentation. Analysis determined that the amount of computation time and resources required for visual sensing was too excessive. It remains to be seen whether an agent's ability to learn from and interact with its environment would improve when combined with ray cast sensors and environmental observations but permitted to complete the extra amount of training required for use with the visual sensor. The primary implication is not just with the computational capability of the hardware but also lies with the structure and working of ML-Agents. In its current development state, ML-Agents is insufficient to meet these requirements. It is unclear if Unity will continue to improve ML-Agent in this regard or if some other tool would be required.

The findings of this work have several implications for future research. First, improvement to the modeling for drone navigation and flight dynamics including sensor feedback from the DRL algorithm would provide a richer set of actions the agent could use during training. These navigational improvements would allow improvement in the agent's ability to learn to redirect itself around obstacles, adjust speed to improve rate of traversing the course, slow down to pass through more difficult gates, and to reorient position and location with guidance from the context-aware elements. Second, a different tool other than Unity and ML-Agents might prove more beneficial in the development the framework which supports all aspects of this work. Third, in this work no algorithm modifications were made. The assumption was that some form of learned acceptance and adaption to environment variabilities could be achieve through just modifications to the environment during training. A future effort could determine what modifications to the algorithm might lead to these intended results. The final implication for future work would be the ability to successfully integrate the visual sensor either through the computing resources, improvement to the existing ML-Agents model, or finding other tools which are computationally efficient to reduce the computation costs to an acceptable level. This work still considers the visual sensing for the DRP PPO-based agent to be an essential part of the work. It is possible that some form of generalization can be found when this is included in the learning process of the agent.

Recommendations

There are several recommendations for future research specific to this SDR+ technique. Better agent navigation capabilities are required. The learning process should allow feedback from ML-Agents to permit the agent to perform more distinct and required movements in line with the agent's state. These improvements would include better control of the agent's speed which includes the ability to stop and reverse direction. The navigation also needs to apply to finer control over side to side, up and down movement. The assumption is that by having an improved feedback mechanism from ML-Agents would allow better navigation control during training which would then be instilled into the inference model.

It was shown that domain randomization does work, and the agent learns from training which incorporates domain randomization. Further exploration could be made in the use of other types of domain randomizations. The literature cites other application areas in which domain randomization is effective.

New research could determine improvements to the context-aware elements. In this research an erroneous learning behavior and an overfitting problem were observed in conjunctions with context-aware elements. This work's results indicated that changes from a reward to guidance-based application inhibited the learning problems previously observed. Continued research in this area could prove useful in support of improved and more rapid training techniques. Additionally, if context-aware elements could be associated with visual sensing, it is assumed this might help reduce the computational costs associated with this sensing method. The field of real time ray tracing provides some interesting techniques with regards to the use of machine learning to speed up the ray tracing process. An extension of this concept could apply in speeding up the visual sensing learning rate.

One of the main hypotheses of this works was that an agent could also learn to understand and adapt to any type of variability seen in the environment. This work barely breached this concept. It was inconclusive but doubtful from the results of this work that modifications to the synthetic environment alone could generate such a condition in the inference model. This implies that additional research in this generalization technique is required to understand if it is possible, and if so, what would be required. Many of these techniques might also be addressed by either using tools other than Unity with its ML-Agents or, perform modifications to Unity's ML-Agents if the code is accessible.

Summary

The concept of training AI agents in synthetic environments becomes advantageous when considering the costs associated with real world training. The advantages also include the flexibility and adjustments that can be obtained through synthetic training, testing, and validation in these environments. There exists a reality gap between the synthetic training to that of training in a real-world environment. This has been considered one of the main drawbacks to synthetic training, however, the costs associated with methods which can bridge this reality gap far out way the costs of doing so under real-world circumstances. The goal of this research work was to address this issue.

The goal consisted of building a framework composed of synthetic environments with the ability to apply domain randomizations in both a naturalistic setting as well as physical-based modifications. An application of context-aware elements was applied to the synthetic environment to provide guidance and improve training speed for the agent. The framework incorporated an existing DRL PPO-based algorithm via a Python interface. It also included a C# programming interface to allow control and configuration of both the synthetic environment, the control of the agent, an interface into ML-Agents, and the means to apply domain randomizations to the synthetic environment.

This work proposed two research questions. The first was whether this methodology could improve upon the agent's ability to better generalize to conditions unseen in the training. The second research question was to determine if the first question could be answered without modifying or adding to the DRL PPO-based algorithm. Two hypotheses were established. The first hypothesis stated that use of naturalistic and physical-based domain randomizations was just as effective as the more drastic effects used in other research works. The second hypothesis stated that context-aware element was effective in providing guidance to the agent during training.

An extensive literature review was performed to better understand the various forms of existing research related to this work. It also permitted the selections of specific research works and potential gaps which could then be extended for this work. The seminal work of domain randomization (Tobin et al., 2018) with the follow-on research in structured domain randomization (Prakash et al., 2019) were selected. This provide the foundation for this work; however, a more detailed literature review was examined to address the various components that constitute the concepts of this work. The literature review covered works on generalization, DRL algorithms, synthetic-based training, the reality gap, domain randomization and works specific to using learning algorithms in the use case of drone racing. A gap in the research literature was determined which specifically addressed the approach used in this work. This work attempted to help fill that gap in the research literature.

A methodology was established to meet the stated gap and goals. Initially, the methodology addressed the selection of a DRL PPO-based algorithm with the constraint of not having to modify any part of the algorithm. A framework of a simple example was selected as the means to explore the configuration settings of the DRL PPO-based algorithm. This same framework was extended to several frameworks performing different studies. Experiments were run for training, testing, and validation within the specified synthetic environment. The synthetic environments, except for the simple example, were built around the use case of a drone flying through a specified racecourse. Unity, a 2D and 3D game development application, was used as the basis for the framework. Unity provides a module called ML-Agents which incorporates machine learning agents into the synthetic environments. ML-Agent came with a DRL PPO-based algorithm built in and came with a C# scripting or programming interface. ML-Agents provided three methods for the agent to sense the environment: environment observations, ray cast sensing and visual sensing. The framework, based on Unity's product, provided the means to build, generate or incorporate synthetic elements in a three-dimensional space which could be modified to meet the requirements of domain randomization and inclusion of context aware elements. The C# scripting interface also allowed the integration of data collection elements based on the specified metrics for the experiments performed. The analysis of the collected data specifically looked at the agent's performance, but also examined through ablation studies the effect of each component had on the agent.

The methodology required the establishment of a baseline agent. An agent was trained in a stable and common synthetic environment with the purpose using its performance as a baseline in comparison to agents trained under the SDR+ technique. The training of the SDR+ agent required a scripted change in the synthetic environment to adjust both the naturalistic and physical-based domain randomizations. The methodology required that both the baseline agent and the SDR+ agent were trained and then tested and validated in that same training environment. The baseline agent was then tested an evaluated in the synthetic environment with the domain randomizations applied. This was the method to determine the amount of performance degradation the baseline agent had when variance in the environment was introduced. The final element of the of the methodology was to subject both the baseline and SDR+ agents to a completely different synthetic environment with all previous domain randomization applied along with several new ones. The purpose was to evaluate the generalization of the SDR+ agent. An outline of this methodology is shown in Figure 32.

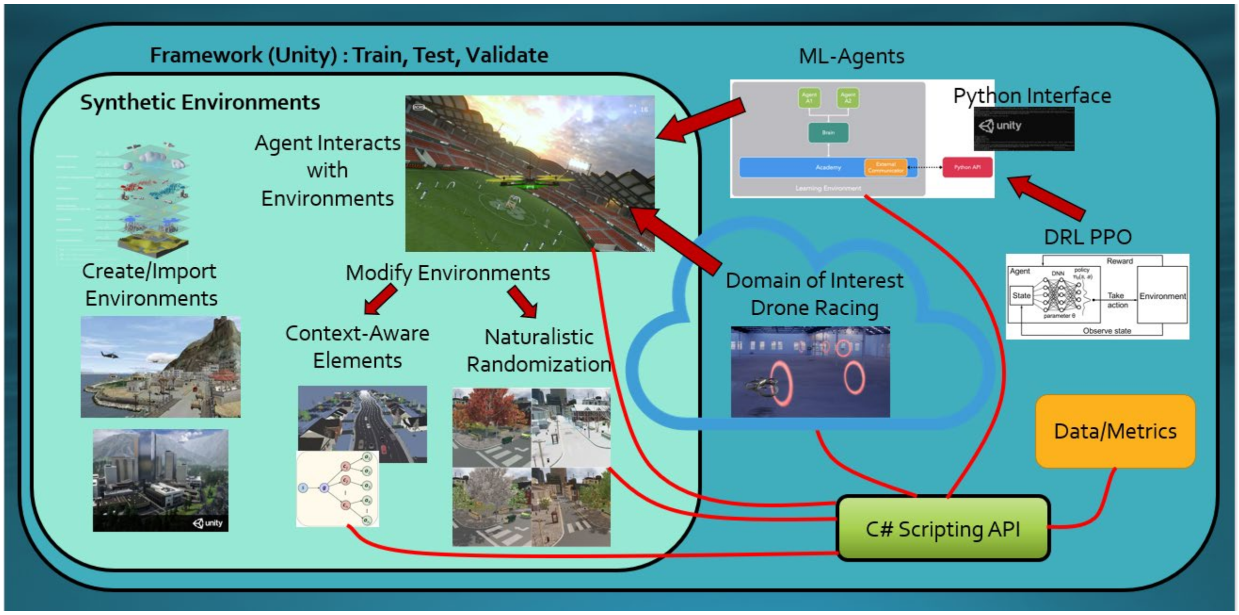


Figure 32: Methodology overview for summary

Multiple experiments were performed for this work. The first experiments involved a framework using a simple example of an agent detecting and then moving to where a target was located. This framework was first used to determine the configuration settings for the DRL PPO-based algorithm. Analysis indicated that all settings including environment observations, ray cast sensing and visual sensing, provided the best results for that agent. It was observed that environment observations were required in ML-Agents otherwise it would not function properly. The length of training was also applied in the configuration settings. Tensorboard, a utility to view the results, was used to examine the cumulative reward, episode length, policy loss and value loss. All these graphs supported the analysis of the results for each experiment.

The experiments with the simple example provided insight into two specific problems that can occur when training DRL agents. Unexpected results indicated a drop off in cumulative reward values during training. These training discrepancies were observed when the agents would vacillate on a reward boundary to gain reward. The agent would ignore the true goal of reaching the target perform a greedy behavior in search of reward. The second discrepancy was overfitting caused by context aware element providing too much reward. Solutions to these problems involved using guidance for the context-aware elements versus some reward value and controlling the reward value provided to the agent at each reward or punishment trigger point.

The training environment for the baseline agent was a city scene synthetic environment. This same environment was used to test and evaluate the inference model. Five different naturalistic environments were built into the environment. The springtime version of the environment provided the most generic environment to perform experiments for the baseline agent. A valid inference model was obtained. With the inference model integrated into the baseline agent, it performed as expected during training and evaluation experiments. This same baseline agent inference model was then used in an experiment which modified the environment with the naturalistic domain randomizations one at a time. The visual based domain randomization effects of these experiments had no effect on the trained baseline agent. Upon further inspection, analysis, and additional research, it was discovered that the ML-Agents visual sensor was providing no value to the inference model. The visual sensor requires more compute power and compute time at a level which became unusable for this work. The visual sensor of ML-Agents was abandoned with that discovery. This also implied that naturalistic domain randomization could not be evaluated. Experimentation at this point turned to the use of just physical-based domain randomizations. The baseline agent was retested for the various physical-based domain randomizations and its performance degraded as expected.

The SDR+ agent was trained in the city scene with just physical-based domain randomizations which included gate size reduction, gate/path changes to position and orientation, and the number of gates used to include their spacing. The training generated adequately performing inference models for each of the physical-based domain randomizations including when they were combined within the same experiment. Some degradation in the performance was caused because the required resetting of episodes during training. This caused the agent to train over small segments of the course and never learned to traverse the entire course.

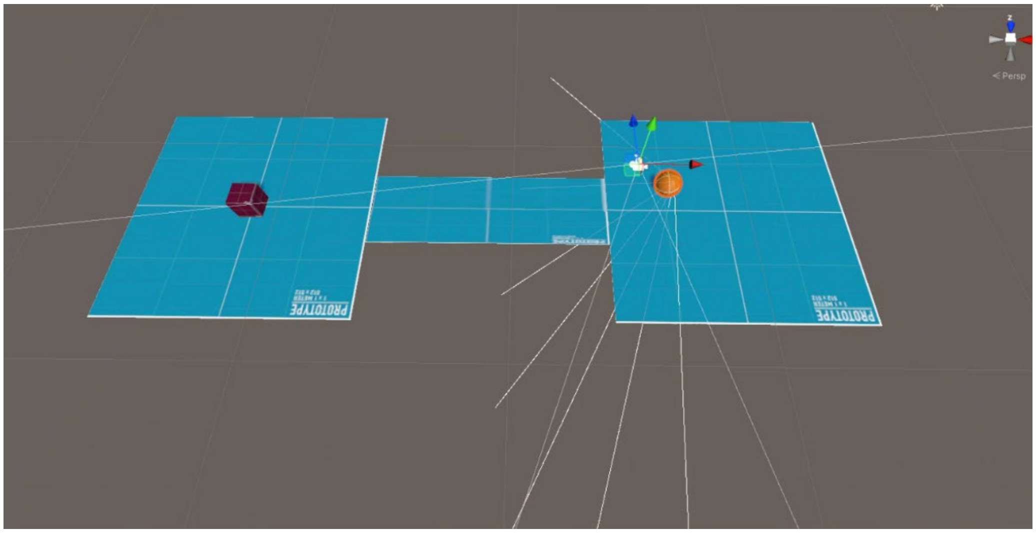
A new and different synthetic environment was created for the final experiment. The concept was to introduce the inference models for both the baseline and SDR+ agents into a new environment which not only contained the same domain randomizations applied during training but also combined and applied modifications to these domain randomizations. New domain randomizations were also introduced. This permitted an environment which differed and was dynamically modified in ways not seen during training. The baseline agent was tested first, and the results showed a poor performance. This was as expected since this experiment provided a greater amount of variation none of which the baseline agent had been trained. The SDR+ agent was tested, and its performance was also degraded, but was much better than the baseline agent. Analysis revealed that any type of generalization most likely did not occur. Improvement in performance of the SDR+ agent over the baseline agent was attributed to the training applied to the SDR+ agent. This implies that the agent performs well when it is trained for the variations which might occur but, there is no clear indications it learns to adapt to variability in the environment for untrained variations.

In summary, the framework and its use in associated experiments were successful in the training, testing, and evaluation of DRL PPO-based agents in synthetic environments. The framework allowed the integration of not just the DRL PPO-based agent but allowed modification of the environment to support the successful use of domain randomization in various forms. The environment was also modified with context-aware elements which successfully provide targeted guidance to the agent and supported the ability to increase the agent's learning rate. Initial experimentation with a simple example in the framework provided the means to determine optimized setting for the DRL PPO-based algorithm. These simple example experiments were also used to find and then solve unexpected agent behaviors such as vacillating over a reward boundary and overfitting during training. This provided a solid foundation which was ported to the drone use case. Experiments were successfully performed to establish an agent to serve as a baseline. Experimentation with the baseline agent generated the metric to use for a baseline performance. This same baseline agent was used to validate that the effect of the reality gap was observed when it was tested under various forms of domain randomization. The SDR+ technique of this work was successfully applied to an agent. It was successfully trained and tested under various forms of domain randomization. In all experiments the SDR+ agent performed better than the baseline agent proving the benefit of applying domain randomization during training. Essentially, an agent trained with domain randomizations applied was able to adjust for the domain randomizations it experienced in training. When the SDR+ agent was subjected to an environment which extended beyond its training environment it did not perform as expected. This indicated that generalization in the form of the agent learning to generalize from a specific set of domain randomizations was not observed in the results. Future work using other modifications or options which may provided better results for generalization.

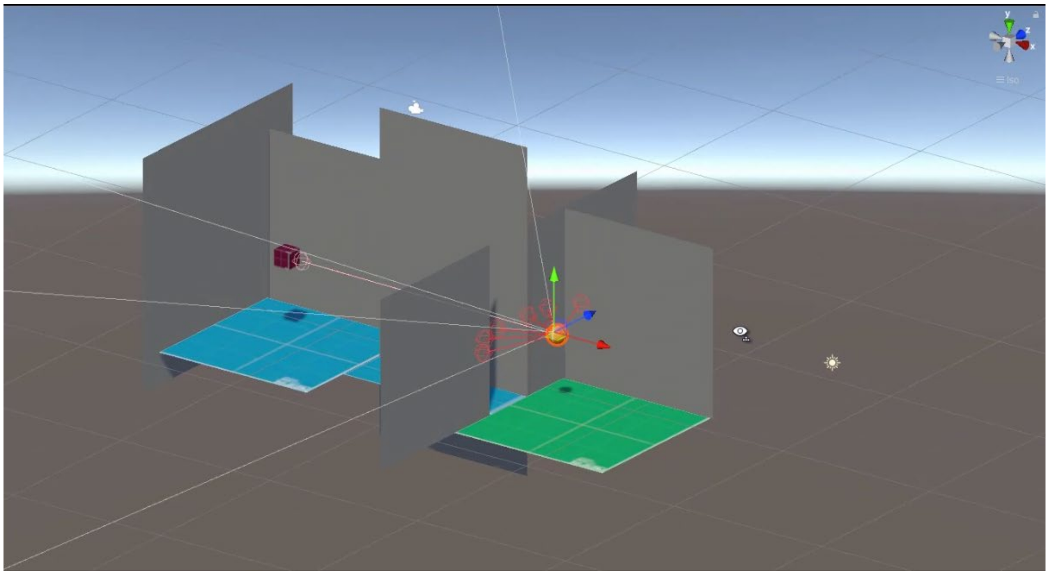
Such modifications are better agent navigation capability, better integration with the DRL algorithm and the use of visual sensing.

Appendix A

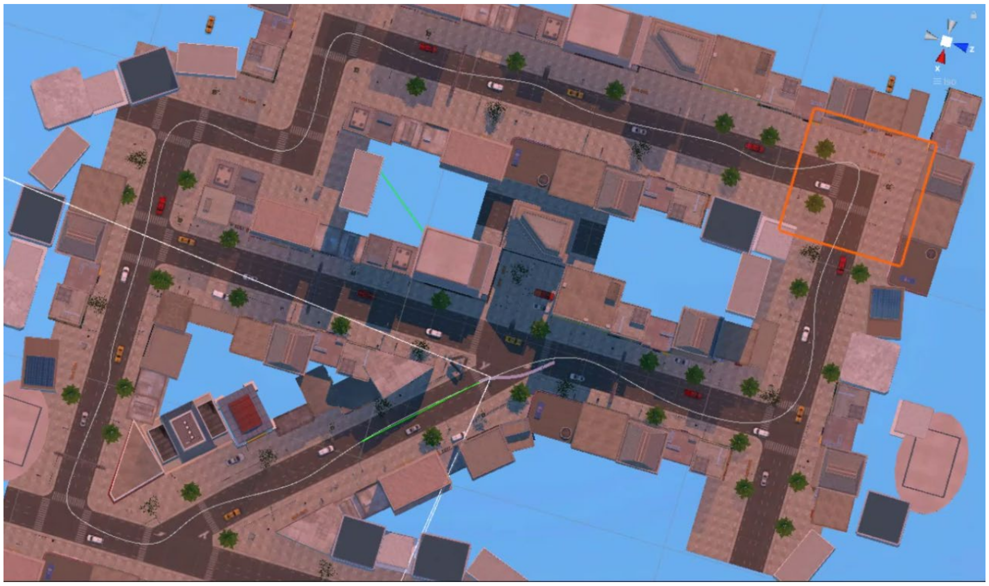
Images of Synthetic Training Environments



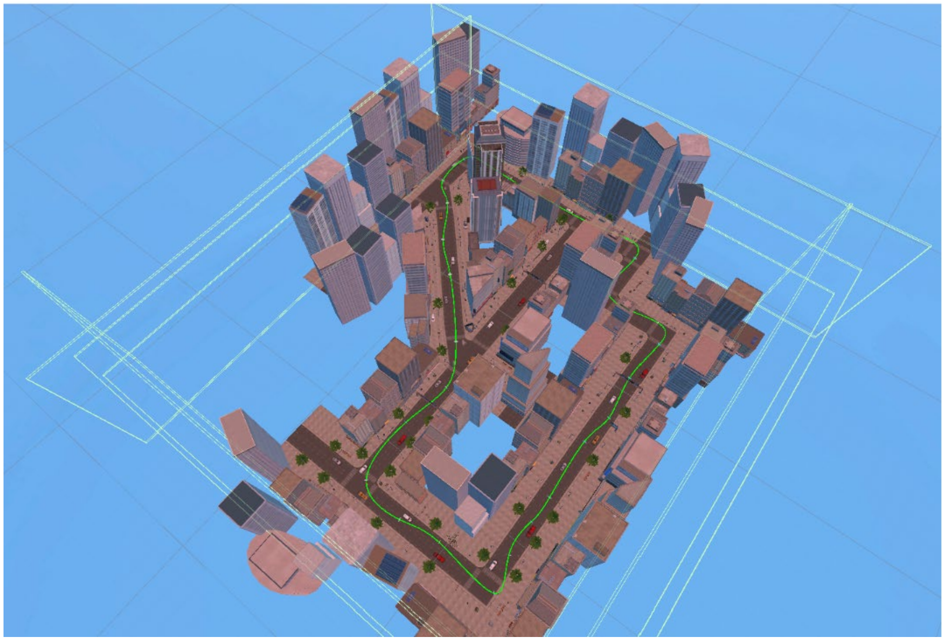
Simple framework example agent getting to the target without falling off edges



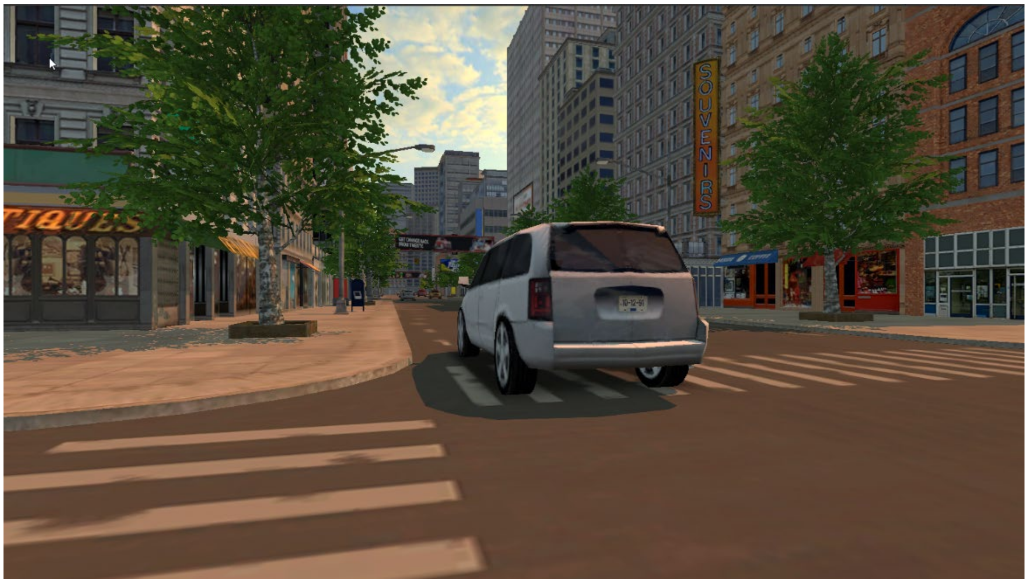
Simple framework example extending to three dimensions



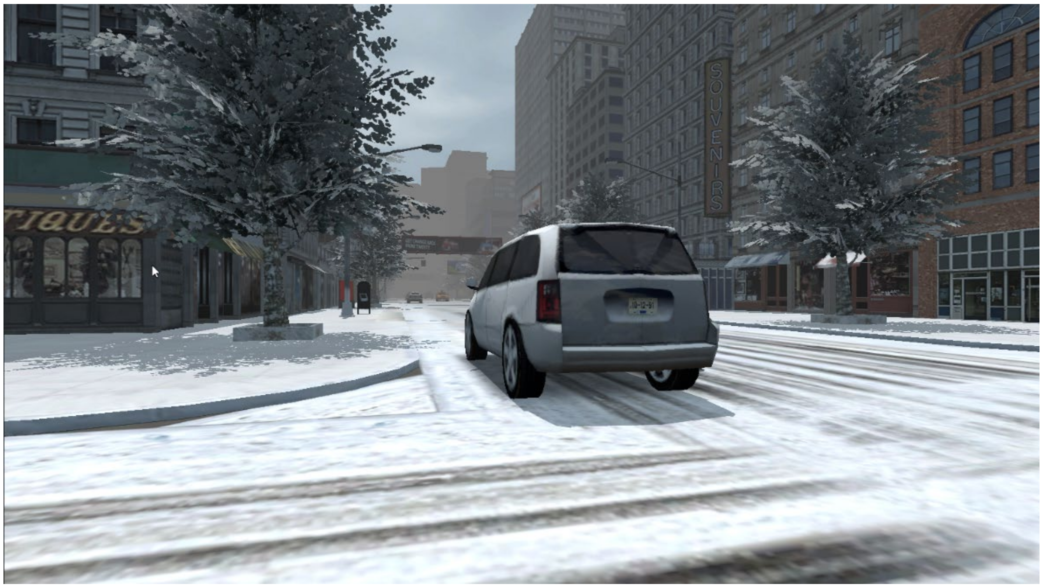
Top-down view of city scene synthetic environment



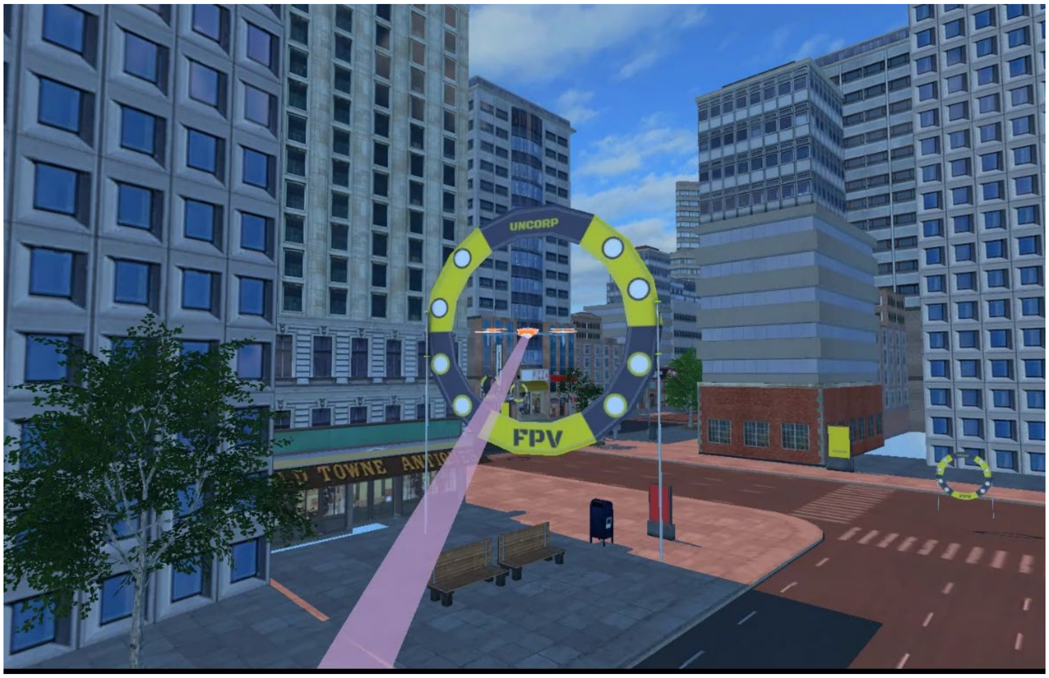
Angled view of city scene synthetic environment with limit boundaries and flight path



City scene synthetic environment showing springtime domain randomization (baseline)



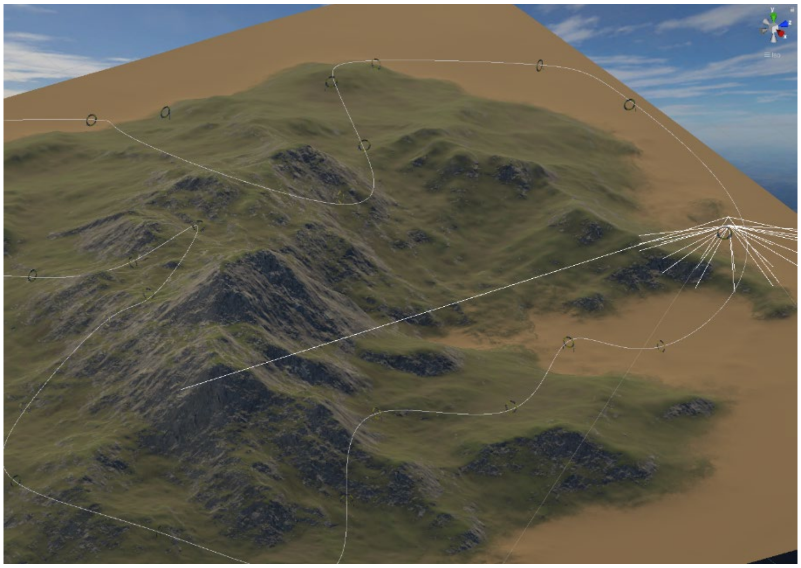
City scene synthetic environment showing winter domain randomization



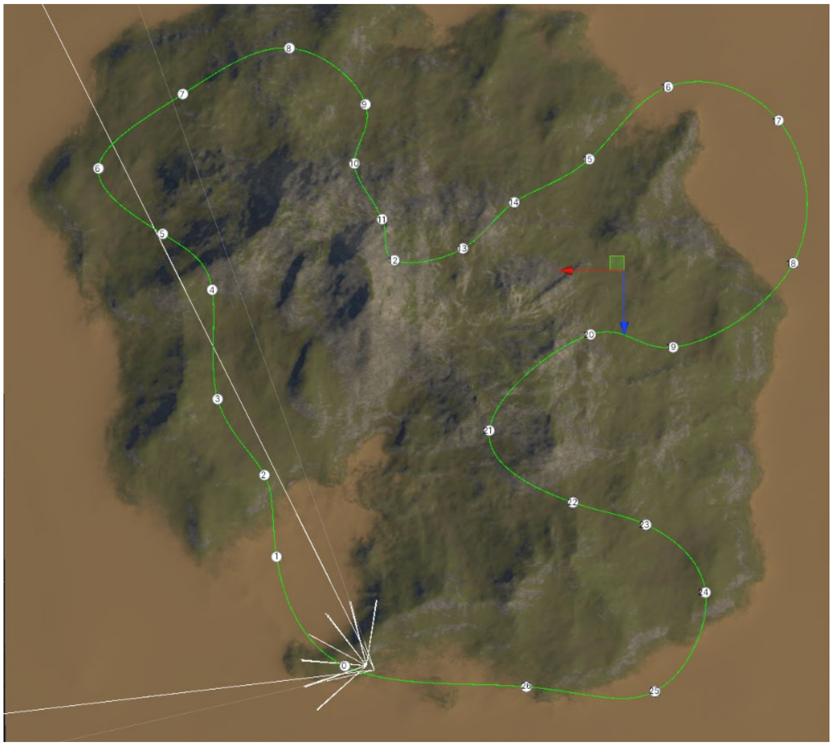
Agent training in city scene synthetic environment – view from following camera

JUST £6
ONE-WAY

Additional view of agent training in city scene environment



View of mountainous island with agent showing sensors during generalization test



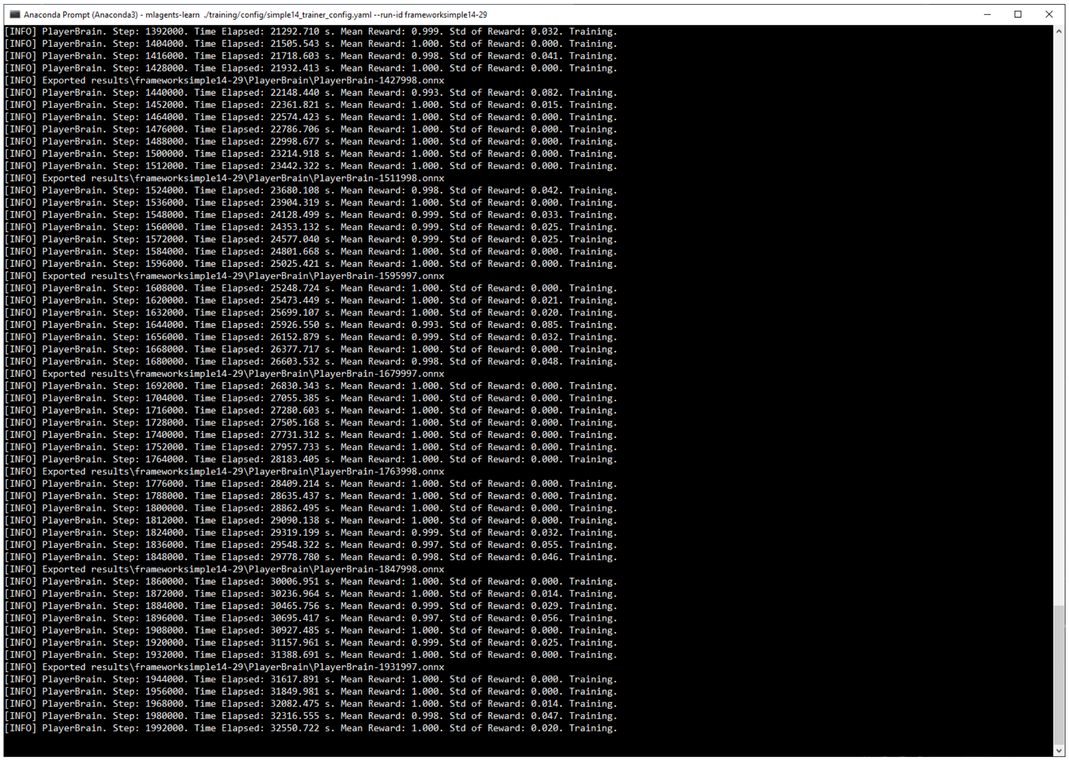
Top-down view of mountainous island showing racecourse and agent with sensors

FPV
FPV
mtnIlsandAgentView

View of agent generalization test in mountainous island synthetic environment

Appendix B

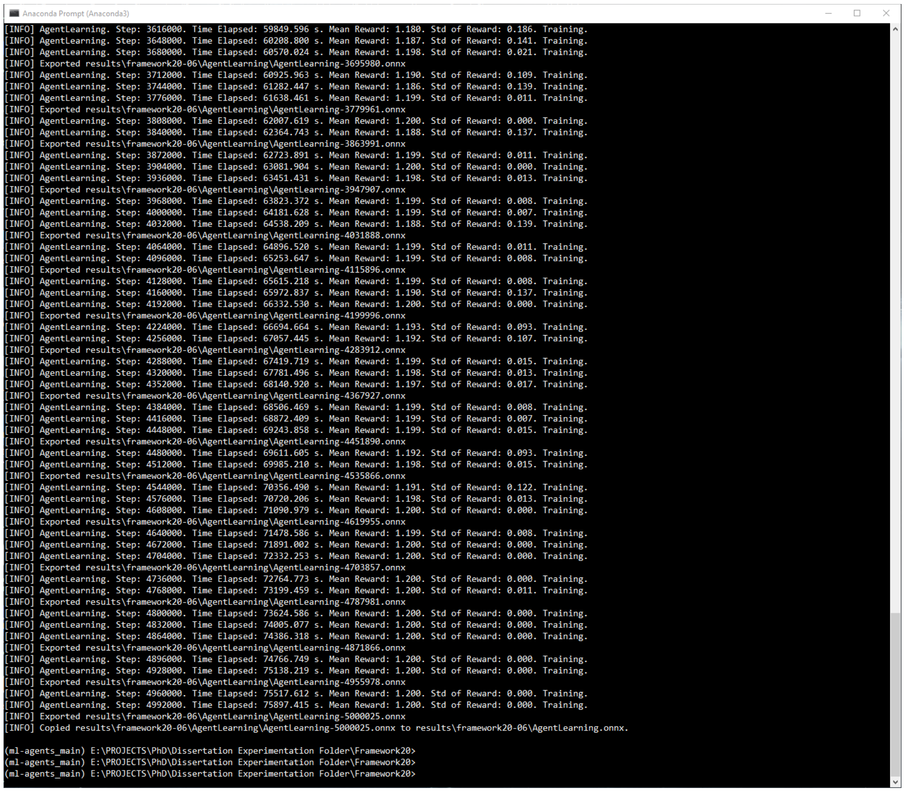
Sample Data from Training Experiments



Anaconda Prompt (Anaconda3) - mlagents-learn ./training/config/simple14_trainer_config.yaml --run-id frameworksimple14-29

```
[INFO] PlayerBrain. Step: 1392000. Time Elapsed: 21292.710 s. Mean Reward: 0.999. Std of Reward: 0.032. Training.
[INFO] PlayerBrain. Step: 1404000. Time Elapsed: 21505.543 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1416000. Time Elapsed: 21718.603 s. Mean Reward: 0.998. Std of Reward: 0.041. Training.
[INFO] PlayerBrain. Step: 1428000. Time Elapsed: 21932.413 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1427998.onnx
[INFO] PlayerBrain. Step: 1440000. Time Elapsed: 22148.440 s. Mean Reward: 0.993. Std of Reward: 0.082. Training.
[INFO] PlayerBrain. Step: 1452000. Time Elapsed: 22361.821 s. Mean Reward: 1.000. Std of Reward: 0.015. Training.
[INFO] PlayerBrain. Step: 1464000. Time Elapsed: 22574.423 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1476000. Time Elapsed: 22786.706 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1488000. Time Elapsed: 22998.677 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1500000. Time Elapsed: 23214.918 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1512000. Time Elapsed: 23442.322 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1511998.onnx
[INFO] PlayerBrain. Step: 1524000. Time Elapsed: 23680.108 s. Mean Reward: 0.998. Std of Reward: 0.042. Training.
[INFO] PlayerBrain. Step: 1536000. Time Elapsed: 23904.319 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1548000. Time Elapsed: 24128.499 s. Mean Reward: 0.999. Std of Reward: 0.033. Training.
[INFO] PlayerBrain. Step: 1560000. Time Elapsed: 24353.132 s. Mean Reward: 0.999. Std of Reward: 0.025. Training.
[INFO] PlayerBrain. Step: 1572000. Time Elapsed: 24577.040 s. Mean Reward: 0.999. Std of Reward: 0.025. Training.
[INFO] PlayerBrain. Step: 1584000. Time Elapsed: 24801.668 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1596000. Time Elapsed: 25025.421 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1595997.onnx
[INFO] PlayerBrain. Step: 1608000. Time Elapsed: 25248.724 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1620000. Time Elapsed: 25473.449 s. Mean Reward: 1.000. Std of Reward: 0.021. Training.
[INFO] PlayerBrain. Step: 1632000. Time Elapsed: 25699.107 s. Mean Reward: 1.000. Std of Reward: 0.020. Training.
[INFO] PlayerBrain. Step: 1644000. Time Elapsed: 25926.550 s. Mean Reward: 0.993. Std of Reward: 0.085. Training.
[INFO] PlayerBrain. Step: 1656000. Time Elapsed: 26152.879 s. Mean Reward: 0.999. Std of Reward: 0.032. Training.
[INFO] PlayerBrain. Step: 1668000. Time Elapsed: 26377.717 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1680000. Time Elapsed: 26603.532 s. Mean Reward: 0.998. Std of Reward: 0.048. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1679997.onnx
[INFO] PlayerBrain. Step: 1692000. Time Elapsed: 26830.343 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1704000. Time Elapsed: 27055.385 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1716000. Time Elapsed: 27280.603 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1728000. Time Elapsed: 27505.168 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1740000. Time Elapsed: 27731.312 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1752000. Time Elapsed: 27957.733 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1764000. Time Elapsed: 28183.405 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1763998.onnx
[INFO] PlayerBrain. Step: 1776000. Time Elapsed: 28409.214 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1788000. Time Elapsed: 28635.437 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1800000. Time Elapsed: 28862.495 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1812000. Time Elapsed: 29090.138 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1824000. Time Elapsed: 29319.199 s. Mean Reward: 0.999. Std of Reward: 0.032. Training.
[INFO] PlayerBrain. Step: 1836000. Time Elapsed: 29548.322 s. Mean Reward: 0.997. Std of Reward: 0.055. Training.
[INFO] PlayerBrain. Step: 1848000. Time Elapsed: 29778.780 s. Mean Reward: 0.998. Std of Reward: 0.046. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1847998.onnx
[INFO] PlayerBrain. Step: 1860000. Time Elapsed: 30006.951 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1872000. Time Elapsed: 30236.964 s. Mean Reward: 1.000. Std of Reward: 0.014. Training.
[INFO] PlayerBrain. Step: 1884000. Time Elapsed: 30465.756 s. Mean Reward: 0.999. Std of Reward: 0.029. Training.
[INFO] PlayerBrain. Step: 1896000. Time Elapsed: 30695.417 s. Mean Reward: 0.997. Std of Reward: 0.056. Training.
[INFO] PlayerBrain. Step: 1908000. Time Elapsed: 30927.485 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1920000. Time Elapsed: 31157.961 s. Mean Reward: 0.999. Std of Reward: 0.025. Training.
[INFO] PlayerBrain. Step: 1932000. Time Elapsed: 31388.691 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] Exported results\frameworksimple14-29\PlayerBrain\PlayerBrain-1931997.onnx
[INFO] PlayerBrain. Step: 1944000. Time Elapsed: 31617.891 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1956000. Time Elapsed: 31849.981 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
[INFO] PlayerBrain. Step: 1968000. Time Elapsed: 32082.475 s. Mean Reward: 1.000. Std of Reward: 0.014. Training.
[INFO] PlayerBrain. Step: 1980000. Time Elapsed: 32316.555 s. Mean Reward: 0.998. Std of Reward: 0.047. Training.
[INFO] PlayerBrain. Step: 1992000. Time Elapsed: 32550.722 s. Mean Reward: 1.000. Std of Reward: 0.020. Training.
```

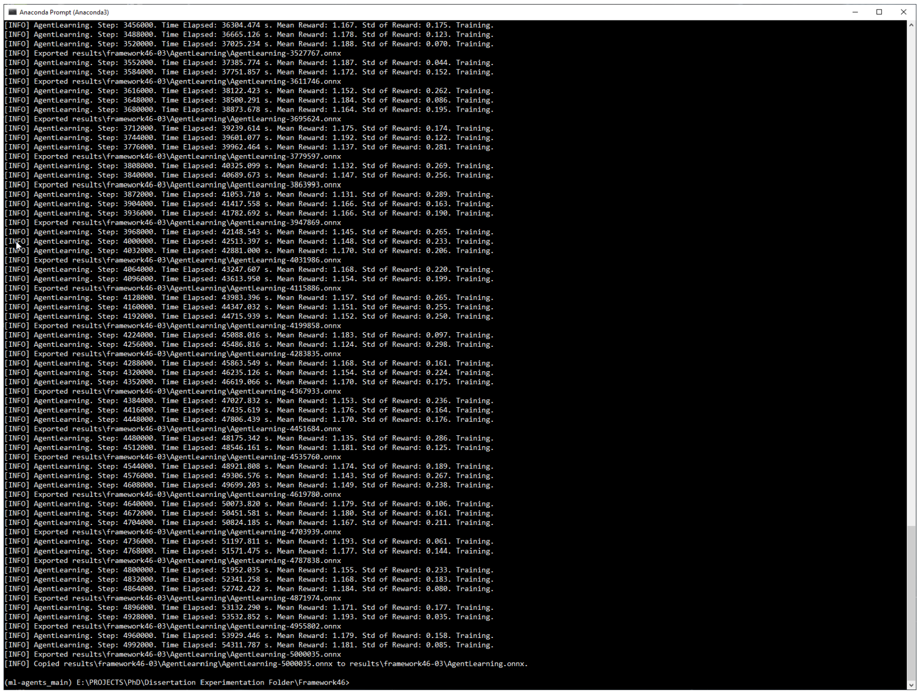
Sample data output from during simple example training



```
[INFO] AgentLearning. Step: 3616000. Time Elapsed: 59849.596 s. Mean Reward: 1.180. Std of Reward: 0.186. Training.
[INFO] AgentLearning. Step: 3648000. Time Elapsed: 60208.800 s. Mean Reward: 1.187. Std of Reward: 0.141. Training.
[INFO] AgentLearning. Step: 3680000. Time Elapsed: 60570.024 s. Mean Reward: 1.198. Std of Reward: 0.021. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-3695980.onnx
[INFO] AgentLearning. Step: 3712000. Time Elapsed: 60925.963 s. Mean Reward: 1.190. Std of Reward: 0.109. Training.
[INFO] AgentLearning. Step: 3744000. Time Elapsed: 61282.447 s. Mean Reward: 1.186. Std of Reward: 0.139. Training.
[INFO] AgentLearning. Step: 3776000. Time Elapsed: 61638.461 s. Mean Reward: 1.199. Std of Reward: 0.011. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-3779961.onnx
[INFO] AgentLearning. Step: 3808000. Time Elapsed: 62007.619 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 3840000. Time Elapsed: 62364.743 s. Mean Reward: 1.188. Std of Reward: 0.137. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-3863991.onnx
[INFO] AgentLearning. Step: 3872000. Time Elapsed: 62723.891 s. Mean Reward: 1.199. Std of Reward: 0.011. Training.
[INFO] AgentLearning. Step: 3904000. Time Elapsed: 63081.904 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 3936000. Time Elapsed: 63451.431 s. Mean Reward: 1.198. Std of Reward: 0.013. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-3947907.onnx
[INFO] AgentLearning. Step: 3968000. Time Elapsed: 63823.372 s. Mean Reward: 1.199. Std of Reward: 0.008. Training.
[INFO] AgentLearning. Step: 4000000. Time Elapsed: 64181.628 s. Mean Reward: 1.199. Std of Reward: 0.007. Training.
[INFO] AgentLearning. Step: 4032000. Time Elapsed: 64538.209 s. Mean Reward: 1.188. Std of Reward: 0.139. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4031888.onnx
[INFO] AgentLearning. Step: 4064000. Time Elapsed: 64896.520 s. Mean Reward: 1.199. Std of Reward: 0.011. Training.
[INFO] AgentLearning. Step: 4096000. Time Elapsed: 65253.647 s. Mean Reward: 1.199. Std of Reward: 0.008. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4115896.onnx
[INFO] AgentLearning. Step: 4128000. Time Elapsed: 65615.218 s. Mean Reward: 1.199. Std of Reward: 0.008. Training.
[INFO] AgentLearning. Step: 4160000. Time Elapsed: 65972.837 s. Mean Reward: 1.190. Std of Reward: 0.137. Training.
[INFO] AgentLearning. Step: 4192000. Time Elapsed: 66332.530 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4199996.onnx
[INFO] AgentLearning. Step: 4224000. Time Elapsed: 66694.664 s. Mean Reward: 1.193. Std of Reward: 0.093. Training.
[INFO] AgentLearning. Step: 4256000. Time Elapsed: 67057.445 s. Mean Reward: 1.192. Std of Reward: 0.107. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4283912.onnx
[INFO] AgentLearning. Step: 4288000. Time Elapsed: 67419.719 s. Mean Reward: 1.199. Std of Reward: 0.015. Training.
[INFO] AgentLearning. Step: 4320000. Time Elapsed: 67781.496 s. Mean Reward: 1.198. Std of Reward: 0.013. Training.
[INFO] AgentLearning. Step: 4352000. Time Elapsed: 68140.920 s. Mean Reward: 1.197. Std of Reward: 0.017. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4367927.onnx
[INFO] AgentLearning. Step: 4384000. Time Elapsed: 68506.469 s. Mean Reward: 1.199. Std of Reward: 0.008. Training.
[INFO] AgentLearning. Step: 4416000. Time Elapsed: 68872.409 s. Mean Reward: 1.199. Std of Reward: 0.007. Training.
[INFO] AgentLearning. Step: 4448000. Time Elapsed: 69243.858 s. Mean Reward: 1.199. Std of Reward: 0.015. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4451890.onnx
[INFO] AgentLearning. Step: 4480000. Time Elapsed: 69611.605 s. Mean Reward: 1.192. Std of Reward: 0.093. Training.
[INFO] AgentLearning. Step: 4512000. Time Elapsed: 69985.210 s. Mean Reward: 1.198. Std of Reward: 0.015. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4535866.onnx
[INFO] AgentLearning. Step: 4544000. Time Elapsed: 70356.490 s. Mean Reward: 1.191. Std of Reward: 0.122. Training.
[INFO] AgentLearning. Step: 4576000. Time Elapsed: 70720.206 s. Mean Reward: 1.198. Std of Reward: 0.013. Training.
[INFO] AgentLearning. Step: 4608000. Time Elapsed: 71090.979 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4619955.onnx
[INFO] AgentLearning. Step: 4640000. Time Elapsed: 71478.586 s. Mean Reward: 1.199. Std of Reward: 0.008. Training.
[INFO] AgentLearning. Step: 4672000. Time Elapsed: 71891.002 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4704000. Time Elapsed: 72332.253 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4703857.onnx
[INFO] AgentLearning. Step: 4736000. Time Elapsed: 72764.773 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4768000. Time Elapsed: 73199.459 s. Mean Reward: 1.200. Std of Reward: 0.011. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4787981.onnx
[INFO] AgentLearning. Step: 4800000. Time Elapsed: 73624.586 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4832000. Time Elapsed: 74005.077 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4864000. Time Elapsed: 74386.318 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4871866.onnx
[INFO] AgentLearning. Step: 4896000. Time Elapsed: 74766.749 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4928000. Time Elapsed: 75138.219 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-4955978.onnx
[INFO] AgentLearning. Step: 4960000. Time Elapsed: 75517.612 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] AgentLearning. Step: 4992000. Time Elapsed: 75897.415 s. Mean Reward: 1.200. Std of Reward: 0.000. Training.
[INFO] Exported results\framework20-06\AgentLearning\AgentLearning-5000025.onnx
[INFO] Copied results\framework20-06\AgentLearning\AgentLearning-5000025.onnx to results\framework20-06\AgentLearning.onnx.

(ml-agents_main) E:\PROJECTS\PhD\Dissertation Experimentation Folder\Framework20>
(ml-agents_main) E:\PROJECTS\PhD\Dissertation Experimentation Folder\Framework20>
(ml-agents_main) E:\PROJECTS\PhD\Dissertation Experimentation Folder\Framework20>
```

Sample data output during baseline agent training



```
[INFO] AgentLearning. Step: 3456000. Time Elapsed: 36304.474 s. Mean Reward: 1.167. Std of Reward: 0.175. Training.
[INFO] AgentLearning. Step: 3488000. Time Elapsed: 36665.126 s. Mean Reward: 1.178. Std of Reward: 0.123. Training.
[INFO] AgentLearning. Step: 3520000. Time Elapsed: 37025.234 s. Mean Reward: 1.188. Std of Reward: 0.070. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3527767.onnx
[INFO] AgentLearning. Step: 3552000. Time Elapsed: 37385.774 s. Mean Reward: 1.187. Std of Reward: 0.044. Training.
[INFO] AgentLearning. Step: 3584000. Time Elapsed: 37751.857 s. Mean Reward: 1.172. Std of Reward: 0.152. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3611746.onnx
[INFO] AgentLearning. Step: 3616000. Time Elapsed: 38122.423 s. Mean Reward: 1.152. Std of Reward: 0.262. Training.
[INFO] AgentLearning. Step: 3648000. Time Elapsed: 38500.291 s. Mean Reward: 1.184. Std of Reward: 0.086. Training.
[INFO] AgentLearning. Step: 3680000. Time Elapsed: 38873.678 s. Mean Reward: 1.164. Std of Reward: 0.195. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3695624.onnx
[INFO] AgentLearning. Step: 3712000. Time Elapsed: 39239.614 s. Mean Reward: 1.175. Std of Reward: 0.174. Training.
[INFO] AgentLearning. Step: 3744000. Time Elapsed: 39601.077 s. Mean Reward: 1.192. Std of Reward: 0.122. Training.
[INFO] AgentLearning. Step: 3776000. Time Elapsed: 39962.464 s. Mean Reward: 1.137. Std of Reward: 0.281. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3779597.onnx
[INFO] AgentLearning. Step: 3808000. Time Elapsed: 40325.099 s. Mean Reward: 1.132. Std of Reward: 0.269. Training.
[INFO] AgentLearning. Step: 3840000. Time Elapsed: 40689.673 s. Mean Reward: 1.147. Std of Reward: 0.256. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3863993.onnx
[INFO] AgentLearning. Step: 3872000. Time Elapsed: 41053.710 s. Mean Reward: 1.131. Std of Reward: 0.289. Training.
[INFO] AgentLearning. Step: 3904000. Time Elapsed: 41417.558 s. Mean Reward: 1.166. Std of Reward: 0.163. Training.
[INFO] AgentLearning. Step: 3936000. Time Elapsed: 41782.692 s. Mean Reward: 1.166. Std of Reward: 0.190. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-3947869.onnx
[INFO] AgentLearning. Step: 3968000. Time Elapsed: 42148.543 s. Mean Reward: 1.145. Std of Reward: 0.265. Training.
[INFO] AgentLearning. Step: 4000000. Time Elapsed: 42513.397 s. Mean Reward: 1.148. Std of Reward: 0.233. Training.
[INFO] AgentLearning. Step: 4032000. Time Elapsed: 42881.000 s. Mean Reward: 1.170. Std of Reward: 0.206. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4031986.onnx
[INFO] AgentLearning. Step: 4064000. Time Elapsed: 43247.607 s. Mean Reward: 1.168. Std of Reward: 0.220. Training.
[INFO] AgentLearning. Step: 4096000. Time Elapsed: 43613.950 s. Mean Reward: 1.154. Std of Reward: 0.199. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4115886.onnx
[INFO] AgentLearning. Step: 4128000. Time Elapsed: 43983.396 s. Mean Reward: 1.157. Std of Reward: 0.265. Training.
[INFO] AgentLearning. Step: 4160000. Time Elapsed: 44347.032 s. Mean Reward: 1.151. Std of Reward: 0.255. Training.
[INFO] AgentLearning. Step: 4192000. Time Elapsed: 44715.939 s. Mean Reward: 1.152. Std of Reward: 0.250. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4199858.onnx
[INFO] AgentLearning. Step: 4224000. Time Elapsed: 45088.016 s. Mean Reward: 1.183. Std of Reward: 0.097. Training.
[INFO] AgentLearning. Step: 4256000. Time Elapsed: 45486.816 s. Mean Reward: 1.124. Std of Reward: 0.298. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4283835.onnx
[INFO] AgentLearning. Step: 4288000. Time Elapsed: 45863.549 s. Mean Reward: 1.168. Std of Reward: 0.161. Training.
[INFO] AgentLearning. Step: 4320000. Time Elapsed: 46235.126 s. Mean Reward: 1.154. Std of Reward: 0.224. Training.
[INFO] AgentLearning. Step: 4352000. Time Elapsed: 46619.066 s. Mean Reward: 1.170. Std of Reward: 0.175. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4367933.onnx
[INFO] AgentLearning. Step: 4384000. Time Elapsed: 47027.832 s. Mean Reward: 1.153. Std of Reward: 0.236. Training.
[INFO] AgentLearning. Step: 4416000. Time Elapsed: 47435.619 s. Mean Reward: 1.176. Std of Reward: 0.164. Training.
[INFO] AgentLearning. Step: 4448000. Time Elapsed: 47806.439 s. Mean Reward: 1.170. Std of Reward: 0.176. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4451684.onnx
[INFO] AgentLearning. Step: 4480000. Time Elapsed: 48175.342 s. Mean Reward: 1.135. Std of Reward: 0.286. Training.
[INFO] AgentLearning. Step: 4512000. Time Elapsed: 48546.161 s. Mean Reward: 1.181. Std of Reward: 0.125. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4535760.onnx
[INFO] AgentLearning. Step: 4544000. Time Elapsed: 48921.808 s. Mean Reward: 1.174. Std of Reward: 0.189. Training.
[INFO] AgentLearning. Step: 4576000. Time Elapsed: 49306.576 s. Mean Reward: 1.143. Std of Reward: 0.267. Training.
[INFO] AgentLearning. Step: 4608000. Time Elapsed: 49699.203 s. Mean Reward: 1.149. Std of Reward: 0.238. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4619780.onnx
[INFO] AgentLearning. Step: 4640000. Time Elapsed: 50073.820 s. Mean Reward: 1.179. Std of Reward: 0.106. Training.
[INFO] AgentLearning. Step: 4672000. Time Elapsed: 50451.581 s. Mean Reward: 1.180. Std of Reward: 0.161. Training.
[INFO] AgentLearning. Step: 4704000. Time Elapsed: 50824.185 s. Mean Reward: 1.167. Std of Reward: 0.211. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4703939.onnx
[INFO] AgentLearning. Step: 4736000. Time Elapsed: 51197.811 s. Mean Reward: 1.193. Std of Reward: 0.061. Training.
[INFO] AgentLearning. Step: 4768000. Time Elapsed: 51571.475 s. Mean Reward: 1.177. Std of Reward: 0.144. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4787838.onnx
[INFO] AgentLearning. Step: 4800000. Time Elapsed: 51952.035 s. Mean Reward: 1.155. Std of Reward: 0.233. Training.
[INFO] AgentLearning. Step: 4832000. Time Elapsed: 52341.258 s. Mean Reward: 1.168. Std of Reward: 0.183. Training.
[INFO] AgentLearning. Step: 4864000. Time Elapsed: 52742.422 s. Mean Reward: 1.184. Std of Reward: 0.080. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4871974.onnx
[INFO] AgentLearning. Step: 4896000. Time Elapsed: 53132.290 s. Mean Reward: 1.171. Std of Reward: 0.177. Training.
[INFO] AgentLearning. Step: 4928000. Time Elapsed: 53532.852 s. Mean Reward: 1.193. Std of Reward: 0.035. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-4955802.onnx
[INFO] AgentLearning. Step: 4960000. Time Elapsed: 53929.446 s. Mean Reward: 1.179. Std of Reward: 0.158. Training.
[INFO] AgentLearning. Step: 4992000. Time Elapsed: 54311.787 s. Mean Reward: 1.181. Std of Reward: 0.085. Training.
[INFO] Exported results\framework46-03\AgentLearning\AgentLearning-5000035.onnx
[INFO] Copied results\framework46-03\AgentLearning\AgentLearning-5000035.onnx to results\framework46-03\AgentLearning.onnx.

(ml-agents_main) E:\PROJECTS\PhD\Dissertation Experimentation Folder\Framework46>
```

Sample data output during SDR+ agent training

References

Abbeel, P., Quigley, M. & Ng, A. Y. (2006). Using Inaccurate Models in Reinforcement Learning. *ACM Proceedings of the 23rd International Conference on Machine Learning*; 1-8.

Abbeel, P. & Schulman, J. (2016). Deep Reinforcement Learning through Policy Optimization. *NIPS 2016.*

Amini, A., et al. (2020). Learning Robust Control Policies for End-to-End Autonomous Driving from Data-Driven Simulation. *IEEE Robotics and Automation Letters, 5(2); 1143-1150, April 2020*. doi: 10.1109/LRA.2020.2966414.

Anderlini, E., Parker, G.G. & Thomas, G. (2019). Docking Control of An Autonomous Underwater Vehicle Using Reinforcement Learning. *Journal of Applied Sciences, 9(13); 3445. August 2019*. doi: 10.3390/app9173456.

Antonova, R., Cruciani, S., Smith, C. & Kragic, D. (2017). Reinforcement Learning for Pivoting Task. arXiv:1703.00472.

Arulkumaran, K., Deisenroth, M.P., Brundage, M. & Bharath, A.A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine, 34(6)*; 26-38, November 2017. Doi: 10.1109/MSP.2017.27443240.

Bacon, P., Harb, J. & Precup, D. (2017). The Option-Critic Architecture. *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI '17)*; 1726-1734. doi:10.555/3298483.3298491.

Bahdanau, D., et al. (2016). An Actor-Critic Algorithm for Sequence Prediction. *Journal of Computing Research Repository, July 2016*. arXiv: 1607.07086.

Barto, A.G., Sutton, R.S. & Anderson, C.W. (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man and Cybernetics, 13(5); 834-846. doi: 10.1109/TSMC.1983.6313077.

Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics, 6(4)*; 679-684.

Bello, I., Pham, H., Le, Q.V., Norouzi, M. & Bengio, S. (2017). Neural Combinatorial Optimization with Reinforcement Learning. *26th International Conference on Learning Representations (ICLR '17) Workshop*. arXiv: 1611.09940.

Bengio, Y., LeCun, Y. & Hinton, G. (2015). Deep Learning. *Nature*, 521(7553); 426-444. doi: 10.1038/nature14539.

Beyer, H. & Schwefel, H. (2002). Evolution Strategies – A Comprehensive Introduction. *Natural Computing, 1(1)*; 3-52.

Borrego, J., et al. (2018). Applying Domain Randomization to Synthetic Data for Object Category Detection. arXiV:1807.09834.

Brooks, R. A., (1992). Artificial Life and Real Robots. *Proceedings of the First European Conference on Artificial Life*; 3-10.

Brown, N. & Sandholm, T. (2017). Libratus: The Superhuman AI for No-Limit Poker. *International Joint Conference on Artificial Intelligence (IJCAI '17)*; 5226-5228. doi: 10.24963/ijcai.2017/772.

Burda, Y., Edwards, H., Storkey, A. & Klimov, O. (2018). Exploration by Random Network Distillation. *Journal of Computing Research Repository, November 2018*. arXiv:1810.12894.

Butler, D. J. & Wulff, J. Stanley, G. B. & Black, M. J., (2012). A Naturalistic Open-Source Movie for Optical Flow Evaluation. European Conference on Computer Vision (ECCV '12); 661-625.

Chaoxing, H. & Li, T. (2017). Deep Reinforcement Learning Based Optimal Trajectory Tracking Control of Autonomous Underwater Vehicle. *Proceedings of the 36th Chinese Control Conference (CCC '17)*; 4958-4965. doi: 10.23919/ChiCC.2017.8028138.

Cutler, M. & How, J. P. (2015). Efficient Reinforcement Learning for Robots Using Informative Simulated Priors. 2015 IEEE Conference on Robotics and Automation (ICRA '15); 2605-2612.

Cutler, M., Walsh, T. J. & How, J. P. (2014). Reinforcement Learning with Multi-Fidelity Simulators. 2014 IEEE International Conference on Robotics and Automation (ICRA '14); 3888-3895.

Dia, T., et al. (2020). Analyzing Deep Reinforcement Learning Agents Trained with Domain Randomization. arXiv:1912.08324v2.

Dai, W., Xu, Q., Yu, Y. & Zhou, Z. (2018). Tunneling Neural Perception and Logic Reasoning Through Abductive Learning. arXiv:1802.01173.

Dayan, P. & Hinton, G.E. (1992). Feudal Reinforcement Learning. *Neural Information Processing Systems (NIPS '92*); 271-278.

Dehban, A., et al., (2019). The Impact of Domain Randomization on Object Detection: A Case Study on Parameter Shapes and Synthetic Textures. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '19); 2593-2600. Doi: 10.1109/IROS40897.2019.8968139.

Deng, Y, Boa, F., Kong, Y., Ren, Z. & Dai, Q. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS '16), 28(3)*; 653-664. doi: 10.1109/TNNLS.2016.2522401.

Deng, J., et al. (2009). ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR '09); 248-255.

Donahue, J., et al., (2016). Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMLI '16),* 39(4); 677-691. Doi: 10.1109/TPAMI.2016.2599174.

Dosovitskiy, A., et al. (2015). FlowNet: Learning Optical Flow with Convolutional Networks. IEEE International Conference on Computer Vision (ICCV '15).

Duan, Y. Chen, X., Houthooft, Schulman, J. & Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. *Proceedings of the 33rd International Conference on Machine Learning (ICML '16), 48*; 1329-1338. doi: 10.5555/3045390.3045531.

DroneSim Pro (2020). *Retrieved from: https://dronessimpro.com/*, April 2020.

DRL Simulator (2020). *Retrieved from: https://thedroneracingleague.com/simulator*, April 2020.

Espeholt, L., et al. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. arXiv: 1802.01561.

Evans, R. & Grefenstette, E. (2018). Learning Explanatory Rules from Noisy Data. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18*); 5598-5602.

Everingham, M., et al., (2015). The Pascal Visual Object Classes Challenge: A Retrospective. International Journal of Computer Vision (IJCV '15); 98-136.

Fazel-Zarandi, M., et al. (2017). Learning Robust Dialog Policies in Noisy Environments. *Journal of Computing Research Repository, December 2017.* arXiv: 1712.04034.

Ferrante, E., Lazaric, A. & Restelli, M. (2008). Transfer of Task Representation in Reinforcement Learning Using Policy-Based Proto-Value Functions. In AAMAS, 2008.

Finn, C., Abbeel, P. & Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaption of Deep Networks. *Proceedings of the 34th International Conference on Machine Learning (ICML '17), 70*; 1126-1135.

Florensa, C., Duan, Y. & Abbeel, P. (2017). Stochastic Neural Networks for Hierarchical Reinforcement Learning. *Proceedings of the International Conference on Learning Representations (ICLR '17)*.

Fortunato, M., et al. (2018). Noisy Networks for Exploration. *International Conference on Learning Representations (ICLR '18)*, February 2018.

FPV FreeRider (2020). Retrieved from: https://fpv-freerider.itch.io/fpv-freerider

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V. & Birattari, M. (2014). Automode: A Novel Approach to the Automatic Design of Control Software for Robot Swarms. *Swarm Intelligence*, 8(2); 89-112.

Francois-Lavet, V. (2016). *Contributions to Deep Reinforcement Learning and its Application in Smart Grids*. University of Liege, Department of Electrical & Computer Science, liege, Wallonia, Belgium. In: *European Conference on Artificial Life.* Springer. 704-720.

Francois-Lavet, V., et al. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning 11(3-4)*. doi: 10.1561/2200000071.

Frutos-Pascual, M. & Zapirain, B.G. (2017). Review of the Use of AI Techniques in Serious Games: Decision Making and Machine Learning. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG '17), 9*; 133-152. doi: 10.1109/TCIAIG.2017.2512592.

Gaidon, A., Wang, Q., Cabon, Y. & Vig, E. (2016). Virtual Worlds as Proxy for Multi-Object Tracking Analysis. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*; 4340-4349. doi: 10.1109/CVPR.2016.470.

Gauci, J., et al. (2019). Horizon: Facebook's Open Source Applied Reinforcement Learning Platform. *36th International Conference on Machine Learning (ICML '19), Workshop RL4RealLife, May 2019*. arXiv: 1811.00260.

Geiger, A., Lenz, P. & Urtasun, R. (2012). Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In CVPR,2012.

Ghadirzadeh, A., Maki, A., Kragic, D. & Bjorkman, M. (2017). Deep Predictive Policy Training using Reinforcement Learning. arXiv:1703.00272.

Graves, A., et al. (2016). Hybrid Computing Using a Neural Network with Dynamic External Memory. *Nature, 538(7626)*; 471-476. doi: 10.1038/nature20101.

Grun, S., Honinger, S., Scheikl, P. M., Hein, B. & Kroger, T. (2019). Evaluation of Domain Randomization Techniques for Transfer Learning. 2019 19th International Conference on Advanced Robotics (ICAR '19); 481-486. Doi: 10.1109/ICAR46387.2019.8981654.

Gu, S. Holly, E., Lillicrap, T. & Levine, S. (2017). Deep Reinforcement Learning for Robotic Manipulations with Asynchronous Off-Policy Updates. *2017 IEEE International Conference on Robotics and Automation (ICRA '17)*; 3389-3396. doi: 10.1109/ICRA.2017.7989385.

Gudmundsson, S. E., Eisen, P., Poromaa, E. & Nodet, A. (2018). Human-Like Playtesting with Deep Learning. 2018 IEEE Conference on Computational Intelligence and Games (CIG '18); 1 – 8. doi: 10.1109/CIG.2018.8490442.

Haarnoja, T. et al. (2018). Soft Actor-Critic Algorithms and Applications. *arXiv preprint, December 2018.* arXiv:1812.05905.

Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S. & Cipolla, R. (2016). SceneNet: Understanding Real World Indoor Scenes with Synthetic Data. CVPR 2016.

Heess, N. et al. (2017). Emergence of Locomotion Behaviors in Rich Environments. arXiv:1707.02286.

Hinterstoisser, V. et al., (2013). Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. Asian Conference on Computer Vision (ACCV '13); 548-562.

Hinterstoisser, S., Lepetit, V., Wohlhart, P. & Konolige, K. (2017). On Pre-Trained Image Features and Synthetic Images for Deep Learning. arXiv:1710.10710.

Hinton, G. & Sejnowski, T. (1999). *Unsupervised Learning: Foundations of Neural Computation*, MIT Press, ISBN 978-0262581684.

Holcomb, S.D., Porter, W.K., Ault, S.V., Mao, G. & Wang, J. (2018). Overview on DeepMind and Its AlphaGo Zero AI. *2018 Proceedings of the International Conference on Big Data and Education*; 67-71. doi: 10.1145/3206157.3206174.

Hu, Z., Ma, X., Liu, Z., Hovy, E. & Xing, E. (2016). Harnessing Deep Neural Networks with Logic Rules. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 1*; 2410-2420. doi: 10.18653/v1/P16-1228.

Hu, Y., Qian, H. & Yu, Y. (2017). Sequential Classification-Based Optimization for Direct Policy Search. In AAAI, 2017.

Huang, H., et al., (2019). Deep Reinforcement Learning for UAV Navigation Through Massive MIMO Technique. *IEEE Transactions on Vehicular Technology*. doi: 10.1109/TVT.2019.2952549.

Humr, S. (2019). Augmented Intelligence Warrior: An Artificial Intelligence and Machine Learning Roadmap for the Military. *Modern War Institute at West Point,* retrieved from https://mwi.usma.edu/augmented-intelligence-warrior-artificial-intelligence-machine-learning-roadmap-military/

Jaderberg, M. et al. (2017). Population Based Training of Neural Networks. arXiv:1711.09846, 2017.

Jakobi, N., Husbands, P. & Harvey, I. (1995). Noise and the Reality Gap: The use of Simulation in Evolutionary Robotics. In: *European Conference on Artificial Life (ECAL '95)*. doi: 10.1007/3-540-59496-5_337.

James, S. & Johns, E. (2016). 3D Simulation for Robot Arm Control with Deep Q-Learning. arXiv:1609.03759.

Johnson-Roberson, M., et al. (2017). Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks? 2*017 IEEE Conference on Robotics and Automation (ICRA '17)*; 746-753. doi: 10.1109/ICRA.2017.7989092.

Kaelbling, L., Littman, M. L. & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research, 4*; 237-285. arXiv: cs/9605103. Doi: 10.1613/jair,301.

Kalashnikov, D., et al. (2018). Scalable Deep Reinforcement Learning for Vision-based Robotic Manipulation. *Proceedings of Machine Learning Research (PMLR '18), 87*; 651-673. arXiv:1806.10293.

Kang, K., Belkhale, S., Kahn, G., Abbeel, P. & Levine, S. (2019). Generalization Through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight. arXiv:1902.03701.

Kelly, A. (2019). Camera Vision Unity ML-Agents. *Retrieved From: https://youtube.com/watch?v=7FHyqzUBzZ0, Immersive Limit, March 2021.*

Kersandt, K., Munoz, G. & Barrado, G. (2018). Self-Training by Reinforcement Learning for Full-Autonomous Drones of the Future. *Proceedings of the 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), London, UK*; 1-10. doi: 10.1109/DASC.2018.8569503.

Khan, A. (2018). Deep Reinforcement Learning Based Tracking Behavior for Underwater Vehicles. *Master's Thesis from Norwegian University of Science and Technology,* *Department of Marine Technology*. Retrieved from *https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2564506/19456_FULLTEXT.pdf?sequence=1.*

Khirodkar, R., Yoo, D. & Kitani, K. (2019). Domain Randomization for Scene-Specific Car Detection and Pose Estimation. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV '19); 1932-1940. Doi: 10.1109/WACV.2019.00210.

Kolter, J. Z. & Ng, A. Y. (2007). Learning Omnidirectional Path Follow using Dimensionality Reduction. Robotics: Science and Systems.

Koos, S., Mouret, J. & Doncieux, S. (2013). The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics. IEEE Transactions on Evolutionary Computation, 17(1);122-145.

Krizhevsky, A., Sutskever, I. & Hinton, G.E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, May 2017, 60(6); 84-90. Doi: 10.1145/3065586.

Kumar, S.A., Bharathi, J.R.D. & Indhumathi, R. (2019). Ship Detection for Pre-Annotated Ship Dataset in Machine Learning using CNN. *International Research Journal of Engineering and Technology (IRJET '18), 6(3)*; 2160-2163. e-ISSN: 2395-0056, p-ISSN: 2395-0072.

Lazaric, A., Restelli, M. & Bonarini, A. (2008). Transfer of Samples in Batch Reinforcement Learning. In ICML, 2008.

Leclerc, M., (2018). Ship Classification using Deep Learning Techniques for Maritime Target Tracking. *21st International Conference on Information Fusion (FUSION '18)*; 737-744. doi: 10.23919/ICIF.2018.8455679.

Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11); 2278-2342. Doi: 10.1109/5.726791.

Lester, J.C., Ha, E.Y., Lee, S.Y., Mott, B.W., Rowe, J.P. & Sabourin, J.L. (2013). Serious Games Get Smart: Intelligent Game-Based Learning Environments. *Association for the Advancement of Artificial Intelligence*; 31-45. doi: 10.1609/aimag.v34i4.2488.

Levine, S. & Abbeel, P. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. NIPS, 2014.

Levine, S., Finn, C., Darrell, T. & Abbeel, P. (2016). End-to-End Training of Deep Visumotor Policies. JMLR, 17(39); 1-40.

Levine, S. & Koltun, V. (2013). Guided Policy Search. ICLR 2013.

Li, Y. (2017). Deep Reinforcement Learning: An Overview. arXiv:1701.07274, 2017.

Liaq, M. & Byun, Y. (2019). Autonomous UAV Navigation Using Reinforcement Learning. *International Journal of Machine Learning and Computing, 9(6*); 756-761. doi: 10.18178/ijmlc.2019.9.6.869.

Liftoff (2019). Liftoff by LuGus Studios. *Retrieved from: http://www.liftoff-game.com/.*

Loquerico, A., et al. (2019). Deep Drone Racing: from Simulation to Reality with Domain Randomization. *IEEE Transactions on Robotics, 36(1)*; 1-14. doi: 10.1109/TRO.2019.2942989.

Mayer, N., et al. (2016). A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. CVRP 2016.

Mayer, N., et al. (2018). What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation. IJCV, 126; 942-960.

McCormac, J., Handa, A. & Leutenegger, S. (2017). SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth. ICCV 2017.

Mehta, B., Diaz, M., Golemo, F., Pal, C.J. & Paull, L. (2019). Active Domain Randomization. arXiv:1904.04762v2, July 2019.

Miglino, O., Lund, H. H. & Nolfi, S. (1995). Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, 2(4);417-434.

Mitash, C., Bekris, K. E. & Boularias, A. (2017). A Self-Supervised Learning System for Object Detection using Physics Simulation and Multi-View Pose Estimation. arXiv:1703.03347.

Mnih, V., et al. (2015). Human-level Control Through Deep Reinforcement Learning. *Nature*, 518(7540); 529-533. doi: 10.1038/nature14236.

Mnih, B. et al. (2016). Asynchronous Methods for Deep Reinforcement Learning. ICLR

Moravcik, M., et al. (2017). DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker. *Science, 356(6337)*; 508-513. doi: 10.1126/science.aam6960.

Mueller, M., Casser, V., Lahoud, J., Smith, N. & Ghanem, B. (2017). Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications. arXiv:1708.05869.

Munoz, G., Barrado, C., Cetin, E. & Salami, E. (2019). Deep Reinforcement Learning for Drone Delivery. *Journal of Drones 2019, 3(3)*; 72. doi: 10.3390/drones3030072.

Muratore, F., Treede, F., Gienger, M. & Peters, J. (2018). Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment. *Proceedings of the 2nd Conference on Robotic Learning, 87*; 700-713.

Nikolenko, S. I. (2019). Synthetic Data for Deep Learning. arXiv:1099.11512v1.

Oh, J., Chockalingam, V., Singh, S. & Lee, H. (2016). Control of Memory, Active Perception, and Action in Minecraft. ICLR.

Packer, C., et al. (2019). Assessing Generalization in Deep Reinforcement Learning. arXiv:1810.12282v2.

Parr, R. & Russell, S.J., (1997). Reinforcement Learning with Hierarchies of Machines. *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems (NIPS '97), 10*; 1043-1049.

Pathak, D., Agrawal, P., Efros, A. A. & Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. In ICML, 2017

Patki, N., Wedge, R. & Veeramachaneni, K. (2016). The Synthetic Data Vault. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA '16)*; 399-410. doi: 10.1109/DSAA.2016.49.

Peirelinck, T., Hermans, C., Spiessens, F. & Deconinck, G. (2020). Domain Randomization for Demand Response of an Electric Water Heater. *IEEE Transactions on Smart Grid*. Doi: 10.1109/TSG.2020.3024656.

Peng, X.B., Andrychowicz, M., Zaremba, W. & Abbeel, P. (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA '18)*; 3803-3810. doi: 10.1109/ICRA.2018.8460528.

Peng, X., Sun, B., Ali, K. & Saenko, K. (2015). Learning Deep Object Detectors From 3D Models. *Proceedings of the 2015 International Conference on Computer Vision (ICCV '15)*; 1278-1286. doi: 10.1109/ICCV.2015.151.

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W. & Abbeel, P. (2017). Asymmetric Actor Critic for Image-Based Robot Learning. *2018 Conference on Robotics Science and Systems*. doi: 10.15607/RSS.2018.XIV.008.

Planche, B., et al., (2017). Depthsynth: Real-time Realistic Synthetic Data Generation from CAD Models for 2.5 D Recognition. arXiv:1702.08558.

Plappert, M., et al. (2018). Parameter Space Noise for Exploration. *International Conference on Learning Representations (ICLR '18)*, January 2018.

Polvara, R. et al. (2017). Autonomous Quadrotor Landing Using Deep Reinforcement Learning. *arXiv preprint, February 2018* arXiv: 1709.03339v3.

Pong, V., Gu, S., Dalal, M. & Levine, S. (2018). Temporal Difference Models: Model-Free Deep Reinforcement Learning for Model-Based Control. *International Conference on Learning Representations (ICLR '18)*.

Pouyanfar, S., Saleem, M., George, N. & Chen, S. (2019). ROADS: Randomization for Obstacle Avoidance and Driving in Simulation. *Proceedings of the IEEE Conference of Computer Vision and Pattern Recognition (CVPR '19) Workshops*; 78-87.

Prakash, A. et al. (2018). Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. *2019 International Conference on Robotics and Automation (ICRA '19*); 7249-7255. doi: 10.1109/ICRA.2019.8794443.

PUN (2020). The Photon Unity Networking Engine. *Retrieved from: https://photonengine.com/pun.*

Qiu, W. & Yuille, A. (2016). UnrealCV: Connecting Computer Vision to Unreal Engine. arXiv:1609.01326.

Qian, H., Hu, Y. & Yu, Y. (2016). Derivative-Free Optimization of High-Dimensional Non-Convex Functions by Sequential Random Embeddings. In IJCAI, 2016.

Raginsky, M., Rakhlin, A. & Telgarsky, M. (2017). Non-Convex Learning via Stochastic Gradient Langevin Dynamics: A Nonasymptotic Analysis. In COLT, 2017.

Ramos, F., Possas, R.C. & Fox, D. (2019). BayesSim: Adaptive Domain Randomization via Probabilistic Inference for Robotics Simulators. *Robotics: Science and Systems 2019*.

Richter, S. R., Vineet, V., Roth, S. & Koltun, V. (2016). Playing for Data: Ground Truth from Computer Games. ECCV, 2016.

Ros, G., Sellart, L., Materzynska, J., Vazquez, D. & Lopez, A. (2016). The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. CVPR, June 2016.

Ranzato, M.A., Chopra, S., Auli, M. & Zaremba, W. (2015). Sequence Level Training with Recurrent Neural Networks. arXiv:1511.06732.

Real Drone Simulator (2020). Real Drone Simulator. *Retrieved from: http://www.realdronesimulator.com/.*

Ren, X., et al. (2019). Domain Randomization for Active Pose Estimation. *2019 International Conference on Robotics and Automation (IRCA '19*); 7228-7234. doi: 10.1109/ICRA.2019.8794126.

Richter, S.R., Vineet, V., Roth, S. & Koltun, V. (2016). Playing for Data: Ground Truth from Computer Games. *European Conference on Computer Vision (ECCV '16), 9906*; 102-118.

Rodriguez-Ramos, A., Sampedro, C., Bavle, H., de la Puente, P. & Compoy, P. (2018). A Deep Reinforcement Learning Strategy for UAV Autonomous Landing on a Moving Platform. Journal of Intelligent & Robotic Systems, 39; 351-366. Doi:10.1007/s10846-018-0891-8.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1988). Learning Representations by Back-Propagation Errors. *Cognitive Modeling* 5(3); 1.

Russel, S. J. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, Third Edition, Prentice Hall ISBN 978-9136042594.

Rusu, A. A., et al. (2016). Sim-to-Real Robot Learning from Pixels with Progressive Nets. arXiv:1610.04286.

Sadeghi, F. & Levine, S. (2017). CAD2R: Real Single-Image Flight Without a Single Real Image. *arXiv preprint, June 2017.* arXiv:1611.04201v4.

Sahni, H., Kumar, S., Tejani, F. & Isbell, C. (2017). Learning to Compose Skills. arXiv:1711.11289, 2017.

Salimans, T., Ho, J., Chen, X. & Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864.

Schraml, D. (2019). Physically Based Synthetic Image Generation for Machine Learning: A Review of Pertinent Literature. Proceeding of Photonics and Education in Measurement Science, 111440. doi: 10.1117/12.25333485.

Schreurs, B. (2019). Getting Started with Visual Observations in Unity Machine Learning Agents. *Retrieved From: https://youtube.com/watch?v=TvkK5kq138, Draco Software, April 2019.*

Schulman, J., Levine, S., Abbeel, P., Jordan, M.I. & Moritz, P. (2015). Trust Region Policy Optimization. *Proceedings of the 31st International Conference on Machine Learning (ICML '15), 37.*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint, August 2017.* arXiv: 1707.06347.

Shah, S., Dey, D., Lovett, C. & Kapoor, A. (2017). AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *Field and Service Robotics, Springer Proceedings in Advanced Robotics, 5.* doi: 10.1007/978-3-319-67361-5_40.

Sheckells, M., Garimella, G., Mishra, S. & Kobilarov, M. (2019). Using Data-Driven Domain Randomization to Transfer Robust Control Policies to Mobile Robots. 2019 International Conference on Robotics and Automation (ICRA '19); 3224-3230. Doi: 10.1109/ICRA.2019.8794343.

Shi, J., Yu, Y., Da, Q., Chen, S. & Zeng, A. (2018). Virtual-Taobao: Virtualizing Real-World Online Retail Environment for Reinforcement Learning. *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI '19), 33(1)*. doi: 10.1609/aaai.v33i01.33014902.

Silver, D., et al., (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature, 529(7587)*; 484-489. doi: 10.1038/nature16961.

Singh, S., Barto, A. G. & Chentanez, N. (2004). Intrinsically Motivated Reinforcement Learning. NIPS 2004

Slaoui, R.B., Clements, W.R., Foerster, J.N. & Toth, S. (2019). Robust Domain Randomization for Reinforcement Learning. *Proceedings of the 80th International Conference on Learning Representations (ICLR '20).*

Snoek, J., Larochelle, H. & Adams, R.P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS '12), 2*; 2951-2959.

Song, X., Jiang, Y., Tu, S., Du, Y. & Neyshabur, B. (2019). A Study on Overfitting in Deep Reinforcement Learning. 20 April 2018. arXiV:1804.06893v2.

Such F.P., et al. (2018). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. arXiv:1712.06567.

Sutton, R.S., Precup, D. & Singh, S.P. (1999). A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence, 112(1-2)*; 181-211, 1999.

Suwajanakorn, S., Snavely, N., Tompson, J. & Norouzi, M. (2018). Discovery of Latent 3D Keypoints via End-to-End Geometric Reasoning. NIPS 2018.

Tamar, A., Wu, Y., Thomas, G., Levine, S. & Abbeel, P. (2016). Value Iteration Networks. *Proceedings of the30th International Conference on Neural Information Processing Systems (NIPS '16).*

Tesauro, G., (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM, March 1995, 38(3)*; 58-68. doi: 10.1145/203330.203343.

Tobin, J., et al. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '17)*; 23-30. Doi: 10.1109/IROS.2017.8202133.

Tobin, J., et al. (2018). Domain Randomization and Generative Models for Robotic Grasping. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '18)*; 3482-3489. doi: 10.1109/IROS.2018.8593933.

Tremblay, J., et al. (2018). Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '18)*; 1082-1088. doi: 10.1109/CVPRW.2018.00143.

Tzeng, E., et al. (2016). Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. Workshop on the Algorithmic Foundations of Robotics (WAFR '16).

Unity (2019), Retrieved from https://unity.com.

Van Den Berg, J., et al. (2010). Super-Human Performance of Surgical Tasks by Robots using Iterative Learning from Human-Guided Demonstrations. 2010 IEEE International Conference on Robotics and Automation (ICRA '10); 2074-2081.

Velocidrone (2020). Velocidrone by BatCave Games. *Retrieved from: https://www.velocidrone.com/.*

Vezhnevets, A.S., et al., (2017). Feudal Networks for Hierarchical Reinforcement Learning. *Proceedings of the 34th International Conference on Machine Learning (ICML '17)*, 70; 3540-3549.

Vinyals, O., et al. (2017). StarCraft II: A New Challenge for Reinforcement Learning. *Journal of Computing Research Repository, August 2017.* arXiv:1708.04782.

Vuong, Q., Vikram, S., Su, H., Gao, S. & Christensen, H.I. (2019). How to Pick the Domain Randomization Parameters for Sim-to-Real Transfer of Reinforcement Learning Policies. arXiv:1903.11774v1

Yang, B. & Liu M. (2018). Keeping in Touch with Collaborative UAVs: A Deep Reinforcement Learning Approach. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*; 562-568. doi: 10.24963/ijcai.2018/78.

You, S., Diao, M. & Gao, L. (2019). Deep Reinforcement Learning for Target Searching in Cognitive Electronic Warfare. *IEEE Access, 7*; 37432-37447. doi: 10.1109/ACCESS.2019.2905649.

You, Y., Pan, X., Wang, Z. & Lu, C. (2017). Virtual to Real Reinforcement Learning for Autonomous Driving. *Journal of Computing Research Repository, September 2017.* arXiv: 1704.03952.

Yu, Y. (2018). Towards Sample Efficient Reinforcement Learning. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, July 2018, 31; 5739-574.

Yu, Y., Chen, S., Da, Q. & Zhou, Z. (2018). Reusable Reinforcement Learning via Shallow Trails. *IEEE Transactions on Neural Networks and Learning Systems, 29(6)*; 2204-2215. Doi: 10.1109/TNNLS.2018.2803729.

Yue, X., et al. (2019). Domain Randomization and Pyramid Consistency: Simulation-to-Real Generalization without Accessing Target Domain Data. *2019 IEEE/CVF International Conference on Computer Vision (ICCV '19)*; 2100-2110.

Wang, H., Qian, H. & Yu, Y. (2018). Noisy Derivative-Free Optimization with Value Suppression. In AAAI, 2018.

Wang, L., Wu, Q., Liu, J., Li, J. & Negenborn, R.R. (2019). State-of-the-Art Research on Motion Control of Maritime Autonomous Surface Ships. *Journal of Marine Science and Engineering, 7(12)*; 438. doi: 10.3390/jmse7120438.

Weber, T., et al., (2018). Imagination-Augmented Agents for Deep Reinforcement Learning. *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS '17)*; 5694-5705.

Werbos, P.J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Technical Report, Harvard University, Applied Mathematics.

Whiteson, S. (2012). Evolutionary Computation for Reinforcement Learning. *In Macro Wiering and Martijn van Otterlo, Reinforcement Learning: State-of-the-Art*; 325-355. Springer, Berlin, Heidelbert, 2012.

Zeng, J., Ju, R., Qin, L., Hu, Y., Yin, Q. & Hu, C. (2019). Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning. *Sensors, 19(18), September 2019.* doi: 10/3390/s19183837.

Zephyr-Sim, (2020). Retrieved from: https://zephyr-sim.com. March 2020.

Zhang, X., Chengbo, W., Liu, Y. & Chen, X. (2019). Decision-Making for Autonomous Navigation of Maritime Autonomous Surface Ships Based on Scene Division and Deep Reinforcement Learning. *Sensors, 19(18), September 2019.* doi: 10/3390/s19184055.

Zhang, C., Vinyals, O., Munos, R. & Bengio, S. (2018). A Study on Overfitting in Deep Reinforcement Learning. 20 April 2018. arXiV:1804.06893v2.

Zhang, Y., Qiu, W., Chen, Q., Hu, X. & Yuille, A. (2016). UnrealStereo: A Synthetic Dataset for Analyzing Stereo Vision. arXiv:1612.04647.

Zhang, C., Yu, Y. & Zhou, Z. (2018). Learning Environmental Calibration Actions for Policy Self-Evolution. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*; 3061-3067. doi: 10.24963/ijcai.2018/425.

Zhang, F., Leitner, J., Upcroft, B. & Corke, P. I. (2016). Vision-based Reaching using Modular Deep Networks: From Simulation to the Real World. CoRR, abs/1610.06781.

Zhu, Y. et al. (2017). Target-driven Visual Navigation in Indoor Scenes Using Deep Reinforcement Learning. *2017 IEEE International Conference on Robotic and Automation (ICRA '17)*; 3357-3364. doi: 10.1109/ICRA.2017.7989381.

Zhu, Y., Urtasun, R., Salakhutdinov, R. & Fidler, S. (2019). segDeepM: Exploiting Segmentation and Context in Deep Neural Networks for Object Detection. *2015 IEEE Conference of Computer Vision and Pattern Recognition (CVPR '15)*; 4703-4711. doi: 10.1109/CVPR.2015.7299102.

Zuo, G., Zhang, C., Zheng, T., Gu, Q. & Gong, D. (2019). Three-Dimensional Localization of Convolutional Neural Networks Based on Domain Randomization. 2019 IEEE International Conference on Robotics and Biometrics (ROBIO '19); 3042-3047. Doi: 10.1109/ROBIO49542.2019.8961846.

NSU NOVA
SOUTHEASTERN
UNIVERSITY

Graduate School of Computer and Information Sciences

Certification of Authorship

Submitted to (Name of Instructor): Dr. Michael J. Laszlo

Submitted to (Name of Course): Doctoral Dissertation Report, CISD 920, Winter 2022
Student’s Name: Bryan L. Croft
Date of Submission: 28 April 2022

Purpose and Title of Submission: Dissertation Report –Modified Structured Domain Randomization in a Synthetic Environment for Learning Algorithms.

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this document was prepared by me for this purpose.

Student's Signature: Bryan L Croft signed on 28 April 2022