Towards Learning Autonomous Spacecraft Path-Planning from Demonstrations and Environment Interactions

by

Kanak Parmar

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 7, 2022

Keywords: machine learning, cooperative human-AI learning, interactive path planning, optimal control

Approved by

Davide Guzzetti, Chair, Assistant Professor of Aerospace Engineering
Ehsan Taheri, Assistant Professor of Aerospace Engineering
Masatoshi Hirabayashi, Assistant Professor of Aerospace Engineering

Abstract

An increasing interest in the robotic exploration of uncharted space environments is warranting a spacecraft path-planning approach that is capable of producing not only autonomous, but also generalized solutions. Though current methods allow for some degree of autonomy in specific phases of the mission timeline, such as proximity operations and "touch-and-go" maneuvers, they are not currently extended. Traditional spacecraft path-planning solutions rely on rigorous baseline strategy design, which inherently requires *a priori* knowledge of the state dynamics, which can be quantified via relevant dynamical parameters. This assumption may break down in the context of unknown environmental dynamics, as is the case with the exploration of uncharted environments. Alternative to approaches that require a baseline, path-planning methods driven by machine learning methods may offer solutions that are more generalized and autonomous. However, the extent to which the performance of a machine learning model compares to an optimal solution has not been extensively evaluated, and initial implementation provided in this work aims to provide further insight into this domain. Path-planning methods based on optimal control theory, pure machine learning methods, as well as the novel field of cooperative human-AI interaction are explored in order to not only analyze the advantages and inherent drawbacks of each distinct approach, but also how each method may be leveraged to construct a synergistic framework that is capable of producing generalized and autonomous solutions. Additionally, extensive exploratory work regarding the implementation of cooperative human-AI interaction as applicable to spacecraft path-planning may serve as the first, empirical observations of behavioral cloning within higher fidelity dynamics, as representative of more realistic scenarios, as well as provide early identification of challenges in training fully autonomous agents for a multi-body dynamics path planning problem.

Acknowledgments

The work presented as part of this manuscript would not have been possible without the support of numerous individuals. First and foremost, I would like to thank my parents for always supporting me throughout my academic endeavours, and never once doubting my childhood dream of wanting to have a career in the space exploration industry.

Of course, the support of my thesis supervisor, Dr. Davide Guzzetti, cannot be understated as being instrumental to the success of this work, as well as my continued development as a researcher. The journey through graduate school is never easy, and words cannot easily describe how instrumental the advisor's support is during this time; I could not be more grateful for being given the chance of working with Dr. Guzzetti as not only a student, but a researcher.

Additionally, the support and technical expertise offered by my committee members not only allowed for me to better implement the relevant technical details required for this work, but they also never hesitated to answer any questions or address any confusions I had regarding the relevant subject material. This consequently allowed me to gain a better technical understanding that goes beyond the material covered in the classroom.

Furthermore, I would also like to acknowledge the Department of Aerospace Engineering at Auburn University, for giving me the opportunity to purse a higher education in a field that I am genuinely interested in, and allowing me to ambitiously pursue my career goals.

Lastly, I would like to acknowledge the support of all my colleagues, some of which have turned into close friends, during our journey together; your support and friendship cannot be understated. Though some of us have gone our separate ways due to career advancements, I am sure we will meet again, hopefully sooner than later.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study and exploration of space environments has been a scientific endeavour spanning centuries. Starting with ancient astronomers and philosophers, the constant scientific studies and missions have yielded a fluid evolution in our understanding of not only the solar system, but also our place in it. By leveraging modern technological capabilities, and the increasing synergies between the industry and academia, we are now able to more autonomously explore the solar system with a greater fervor than before.

This rise in exploration interest also brings new technological needs, including the capability of more autonomous and generalized path planning that the spacecraft itself can execute in real-time. Current methods for path planning heavily rely on the concept of designing baseline solutions, which are tailored to adhere to the local environment dynamics [5]. These solutions can be generated via an analytic and/or heuristic multiple-stage process, which generally may begin with the realization of the relevant environment parameters that drive the underlying dynamical motion. These parameters may then be used by human experts, along with corresponding algorithms, at ground control in order to reconstruct the dynamical models, which are subsequently utilized to design the baseline path planning solutions. Though this may seem like a logical path-planning development process, there are some inherent drawbacks.

These drawbacks may be highlighted by shifting scientific interests; in recent years, there has been a growing interest by numerous space agencies in regards to exploring small body environments such as asteroid systems and icy moons [6]. However, in such cases, the relevant dynamical system parameters may not be fully realized until first rendezvous. This will additionally impose a requirement of a phase of the mission timeline in which the

spacecraft takes measurements of the local dynamical environment in order to initiate the baseline solution design process. Furthermore, additional delays in communication between spacecraft and the ground stations may reduce the ability of rapid path-planning adjustments in the presence of strong dynamical perturbations, which consequently may result in the trajectory quickly diverging from the baseline solution in a manner that is unaccounted for within any included margins of error.

The inherent requirements of baseline driven path-planning solutions may serve as a major drawback of such approaches, and therefore a more autonomous approach may be warranted. Baseline driven methods may be characterized by reduced spacecraft reliance on ground control in order to design path-planning solutions, and provide this capability to the spacecraft as a continually present, on-board computational process. Though there are instances of spacecraft being provided full autonomous path-planning ability, such as autonomous landing for asteroid sample collection [7] or short-range proximity operations [8].

This ability of autonomy for long term path-planning, which is also capable of generalizing and adapting to environmental perturbations, may be yielded from the application of machine learning methods. Though machine learning is traditionally related to the field of computer science, with applications in image recognition, text prediction, and robotic manipulations and path planning mobility, their concepts may be applicable within an astrodynamical context. Recent research trends have explored the application of machine learning based methods in the context of spacecraft stationkeeping [9], multibody transfers [10, 11, 12, 13, 14], and real-time guidance schemes [15, 16, 17, 18].

Though these applications yielded promising results, it should be noted that most applications were for specific scenarios and, rarely, included the modeling of a wide range of dynamics in order to achieve generalized solutions. Machine learning based models inherently require training times that may exponentially increase as the problem complexity increases; in other words, increasing the dynamical ranges of the path planning problem may significantly increase the training time required by a machine learning model. Alternatively,

it may be hypothesized that rather than relying on a machine learning agent independently attempting to learn a path-planning solution for a large range of dynamical environments, the training may be reduced by "bootstrapping" the agent's behavioral policy with already existing domain knowledge. This modification may stem from leveraging innate human intuition. Due to their ability of building causal models, as well as the capability of rapidly adapting a general strategy to fit a changing environment, it may be possible to task a human with generating generalized path-planning demonstrations within a wide range of dynamics. These solutions can then be used to train machine learning agents in an attempt to achieve both autonomous and generalized path-planning strategies. However, due to the novelty of the approach, investigations into not only whether a human would be able to successfully navigate a spacecraft in varying dynamical environments, as well as being able to leverage these solutions to train machine learning models, may have only been explored within the investigations presented in this work. To the best of our knowledge, this exploratory work resulted in not only the preliminary establishment of human logic driven autonomous path-planning strategies, but also the advancements in state of art that would be required in order to achieve a feasible implementation of the approach.

In summary, there are several methods that may be applicable for designing path-planning solutions, ranging from the traditional baseline driven methods to the more recent trends of leveraging machine learning models. This work aims to gain a preliminary understanding of not only the relative performance of the methods, but to also analyze the extent to which the performance of a trained machine learning agent compares to an optimal path-planning solution.

The structure of this manuscript is as follows: Chapter 2 summarizes the existing literature and state of art for the path-planning methods that are considered within this investigation (i.e. baseline solution design via optimal control methods, pure machine learning methods, and human driven machine learning methods), Chapter 3 provides the technical theory and mathematical fundamentals of all components for each of these path-planning methods,

Chapter 4 details the benchmarking path-planning problem that all methods were tasked with solving, Chapter 5 discusses the results of the benchmarking investigation, Chapter 6 details the additional numerical investigations conducted that further explored the human driven machine learning method, and Chapter 7 summarizes the conclusions of the work, as well as possible investigations that may be considered for future work.

Chapter 2

State of the Art

2.1 Baseline Driven Spacecraft Path-Planning

The traditional method at spacecraft path planning in relatively uncharted environments is driven by a generalized procedure that is visualized in Figure 2.1. Before the spacecraft has been launched towards it's destination, it may be possible that mission designers have, at best, an estimate of the dynamical environment that the spacecraft will be operating in. Based on these lower fidelity dynamical models, mission analysts design preliminary approach strategies for the initial rendezvous phase of the mission, wherein the spacecraft will encounter the destination celestial body for the first time. After first rendezvous, the mission may generally proceed in the sequential manner that is illustrated in Figure 2.1. First, while on the initially chosen approach trajectory (or baseline orbit in some cases), the spacecraft will take measurements of the local dynamical environment; this data will then be downlinked to mission control, where human experts will begin to reconstruct the dynamical environment at a higher fidelity, and design baseline path planning solutions. These solutions will then be uplinked to the spacecraft, subsequent to which the mission timeline will proceed as planned.

It may be inferred that these traditional methods largely rely on human insight and insight for autonomous space exploration, with the possibility of autonomous execution during specific mission phases, such as autonomous landing for asteroid sample collection [7] or short-range proximity operations [8]. Most of the traditional path planning methods depend on the ability to recreate the environment dynamics, after which human insight is needed to design tailored solutions. Furthermore, not only must the guidance law be suited to the specific dynamics of the orbital environment, it must also take into consideration restrictions

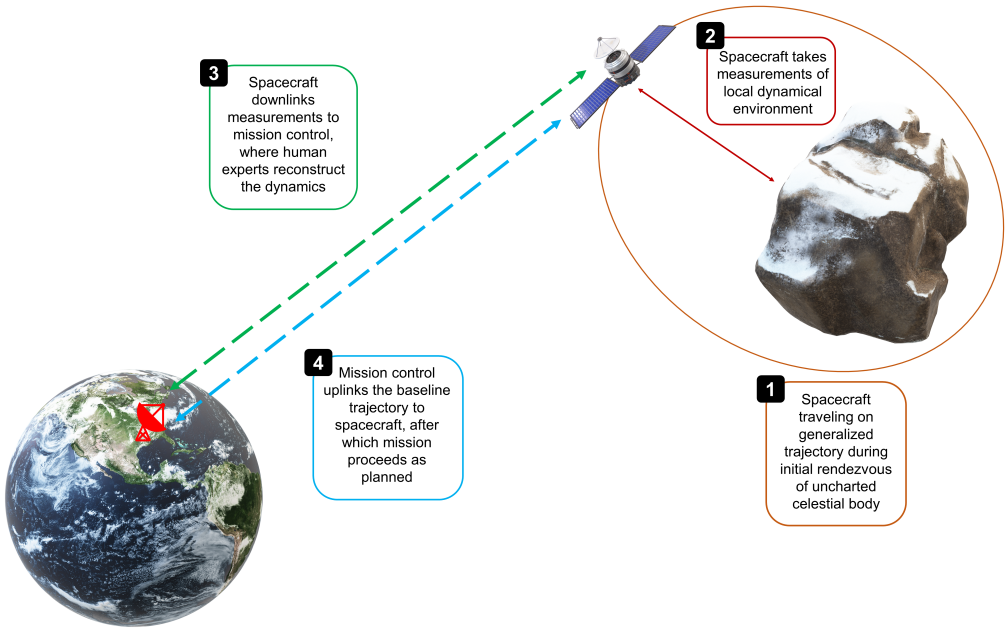


Figure 2.1: Visualization of the generalized path planning pipeline for the traditional, optimal control driven methods.

created by the spacecraft design [5]. As a consequence, missions striving to achieve similar scientific objectives may have vastly differing path planning strategies. For example, JAXA's Hayabusa2 and NASA's OSIRIS-REx are both asteroid sample return missions, but at different destinations. Hayabusa2 has an actively controlled proximity operations strategy, with stationkeeping maneuvers executed daily to maintain spacecraft position about the asteroid Ryugu within predetermined ranges [19]. Conversely, NASA's OSIRIS-REx mission has an initial Sun-Terminator frozen orbit around asteroid Bennu to provide stability against solar pressure [20]. Furthermore, the dependency on baselined solutions can have drastic influence on the spacecraft design; having minimal to no prior knowledge of the dynamical environment could lead to increased design costs of the spacecraft, especially if related to the navigation and control aspects [5]. If baseline trajectories are more yielding to the system dynamics as a consequence of higher sensitivity to errors and uncertainties, overcoming undesired dynamics with adaptive feedback control may become problematic due to spacecraft limitations, such as ΔV constraints. As a result, the adaptive capabilities of a control schemes may not be

exploited in practice; the spacecraft may be, instead, confined to dynamically calmer regions along safe, pre-determined paths, with autonomy only appearing in terminal operations [19].

Baseline driven methods are the *de facto* standard for space exploration missions within the solar system. However, environmental dynamics and system parameters for small bodies cannot be fully reconstructed from ground observations. The lack of accurate information about the dynamics and system parameters may impose restrictions on both the spacecraft design and resulting proximity dynamics [5]. Furthermore, environments such as small body systems may be characterized by chaotic spacecraft dynamics, which may result in a strong amplification of small perturbations and deviations from a nominal trajectory. When considering small-body destinations, a general range of the dynamics and system parameters may be known at the time of launch, but accurate values may only be acquired during initial rendezvous. As a consequence, both the spacecraft bus design and the initial approach trajectory must be equipped to initially handle a general case scenario [5]. However, despite these challenges, the derivation and application of baseline driven path planning in unknown dynamics, such as binary asteroid systems, is an active field of research [21, 22, 23, 24, 25, 26, 27].

Furthermore, the accuracy of current spacecraft path-planning algorithms is largely determined by the ability to estimate a model for the spacecraft dynamics [28]. Spacecraft path-planning algorithms generally rely on analytical predictors, numerical targeting, and closed-loop control. Analytical prediction uses approximate solutions of the orbit dynamics to predict the future evolution of the orbit; it works best when the spacecraft dynamics can be modeled using averaging techniques [28]. For regimes where analytical approaches no longer describe the dynamical realities, the numerical targeting approach is utilized. This method requires defining target locations, events or constraints and subsequently developing the numerical procedures to satisfy these conditions; common applications of this approach include B-plane targeting and the computation of periodic orbits [28]. The output of both analytical prediction and numerical targeting is a deterministic baseline trajectory. During flight, orbit determination errors and small perturbing forces that are not included in the

system model may cause small deviations from the baseline. Such deviations can be reasonably rendered as small random perturbations and corrected with existing stationkeeping and closed-loop control algorithms. With ground station supervision and intervention, the standard sequence comprised of model identification, baseline design with known model parameters and baseline tracking under small perturbations enables spacecraft operations at unexplored destination.

While individual elements of the sequence visualized in Figure 2.1 may be executed autonomously with current technology, it is challenging to automate such path-planning processes as a whole. Among these sequence elements, baseline design may be considered one of the main obstacles to a fully autonomous system. In fact, application of baseline tracking schemes, even with robust control properties, may become infeasible or highly inefficient if the selected baseline is not representative of the natural flow. Conversely, the design of natural-flow-compliant baselines is challenging if system parameters of the dynamical model are poorly known, such as during exploration of uncharted destinations. Furthermore, the probability of occurrence of baseline trajectories that are not fully compliant with real system dynamics increases within systems that exhibit chaotic behavior. In chaotic systems, small variations of the system parameters may yield large variations of the dynamical flow structure. Therefore, baseline trajectories that are valid representations of the natural flow under a prescribed set of system parameters may become a poor description of the natural flow for a small variation of the parameters. The existence of chaos and bifurcations within spacecraft dynamics is also an obstacle to mapping point-design solutions to new applications, ones that are governed by identical equations of motion but different system parameters. Bringing the variation of system parameters to an extreme, an ideal autonomous spacecraft guidance algorithm should, instead, be capable of independent path-planning when the system parameters are randomly assigned from a large range of possible values. In summary, the ability to design autonomous path-planning algorithms for space-based assets is intimately linked to planning the motion of an object within chaotic dynamics.

2.2 Shift towards machine learning methods

Rather than a continued reliance on baseline trajectories, the implementation of machine learning based methods may yield both generalized and autonomous path planning solutions, which consequently results in higher mission adaptability for robotic exploration of uncharted destinations. The machine learning methods that have been explored for astrodynamics and spacecraft path planning applications can generally be divided into two categories: demonstration based learning and environment based/interactive learning. Each of these categories is further discussed below.

2.2.1 Demonstration Based Learning

Demonstration based learning is a form of supervised learning within the context of machine learning. The artificial learning agent, which is represented by a neural network, is provided a training dataset that contains some tuples of state/observation and action labels. The agent then uses this pre-labeled data to learn the underlying patterns and train itself to provide outputs that mimic these inherent trends. Demonstration based learning can be further divided into two categories: training via database generation, and human inspired; each of these two approaches is discussed below.

2.2.1.1 Training via database generations

The training via database generation approach is intuitively simple, with the generalized pipeline visualized in Figure 2.2. First, the training dataset is generated via some form of stochastic solution generation, such as Monte Carlo trials with perturbations on initial conditions, randomly selecting dynamical parameters and attempting to re-converge the solution. Known path planning solutions, such as two-body Hohmann transfers and interplanetary trajectories [29], as well as optimal control solutions stemming from indirect methods, may be used to generate these datasets, which can comprise of thousands to millions of solutions.

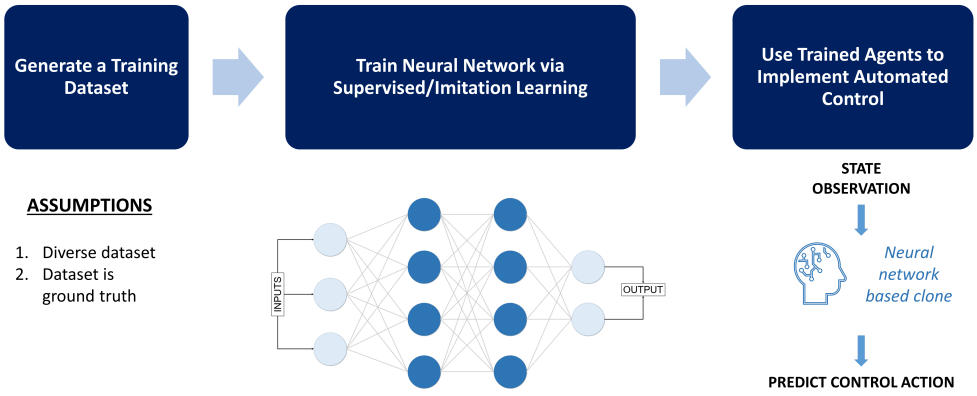


Figure 2.2: Visualization of a path planning pipeline for training an artificial learning agent from database generations.

This training dataset is then used to train an artificial learning agent, which is represented by a neural network. The topic of neural networks itself can be vast, and the technical details are further discussed in later sections of this manuscript; however, they are briefly introduced here in order to provide the appropriate context of this method. Neural networks were developed from biological inspiration from observing the human brain [30, 31], and are comprised of three primary types of layers through which the information flows sequentially. The goal of neural networks is to learn a mapping from a given input, which may represent a state observation (such as position and velocity vectors), to an output prediction (such as the direction of the thrust control vector). The three primary layers are the input, hidden, and output layers; each of these layers is further comprised of a certain amount of computational units that are referred to as "neurons" or "nodes". Between each combinatorial pair of nodes in adjacent layers are connections that are referred to as "synapses". The synapses represent scalar weights, which are used in the decision making process of the neural network output. As the neural network trains on the information it is provided, the values of these weights will be updated as function of the error, which in itself is driven by the computation of the

"loss" value (i.e. the measure of difference between that the network has predicted versus what the true output should be) and the numerical optimization algorithm that has been specified. The numerical optimization algorithm determines the extent to which the weights will be updated. As a common metric to verify the neural network's learning process, the temporal history of the loss value is plotted; generally, and for datasets with minimal outliers, the loss value smoothly decaying and converging to a minimum value is an indication of a stable learning process for the neural network.

Within the context of training via database generation methods, as the artificial agent has access to a labeled dataset, the training process of the neural network can proceed by direct comparison of the predicted output and the true output; standard metrics such as mean square error, are commonly utilized in these cases. Through enough training iterations, the artificial neural network may be able to replicate the true optimal solutions, though the accuracy is heavily reliant on the quality and quantity of data that is provided [32]. Eventually, a relatively accurate trained neural network may be implemented as a surrogate, or proxy, for an optimal controller as it would be able to take in a state observation as the input, and predict the optimal control action that should be taken. This may eventually reduce both the on-board computational footprint of a path planning strategy, as compared to on-board execution of optimization routines.

Although this method may seem as a promising alternative, there still exist some drawbacks of this approach. First, and perhaps most imperative, is that this method is based on the supervised learning branch of machine learning; this means that the trained neural network will only be as accurate as the quality of the data it is provided. More specifically, supervised learning methods train a neural network to learn the average pattern (which may alternatively be referred to as the "behavior" or "policy" within the machine learning terminology). Therefore, if the training dataset is diverse with varying solution patterns, then the supervised learning approach will require additional modifications in order to better replicate the solutions. For example, a training database that is generated from application

of Hohmann transfers, which have singular arc geometries, may produce a more accurate agent as compared to one that is trained on a dataset that has been generated from more rigorous methods and contains multiple types of solution geometries (i.e. both singular arc and multi-revolution transfers for the same problem). Additionally, the type of action space that is modeled, such as impulsive maneuvers (which can be modeled as a discrete action space) in comparison to continuous control (which must be modeled via the generation of probability density functions) are understood to increase the complexity of the task that the neural network is required to learn, and may also affect the performance of a trained agent.

Furthermore, if the network receives an input that is vastly different from the range of those that were included in the additional training dataset, the network may not be able to predict accurately, hence leading to a compounding of errors, especially within the context of path-planning. Secondly, the task of path-planning itself can be categorized as a sequential prediction problem, where previous decisions will affect the current state. Therefore, a slight mistake in the prediction at a previous decision will also lead to a compounding of errors as the control decision are implemented via numerical propagation, and in some cases recovery may not be possible. There exist some machine learning methods that may be able to counter this phenomenon, such as using recurrent neural networks (which are designed for sequential prediction tasks) in lieu of the standard feed forward networks, or shifting to alternative supervised learning algorithms, such as Dataset Aggregation (DAgger) [33], that are specifically tailored to solve sequence prediction problems.

Lastly, the implementation of any neural network driven machine learning method requires an expertise from the designer. These methods may be perceived as "black box" approaches [34], and many critical details such as minor algorithm adjustments and unspoken rules and best practices are generally omitted from the standard literature publications. This leads to a heavy burden on a novice user that may have little to no prior experience with neural networks, and may inadvertently lead to ill-formulated implementations of a neural network, which subsequently will result in low accuracy artificial agents.

In regards to the application of supervised learning methods to spacecraft path planning and astrodynamics applications, there have been recent investigations focusing on interplanetary transfers [29], improving orbit prediction accuracies [35], and real-time control during landing problems [36], with further advances in the state of art being detailed in [37]. Though these applications required the generation of large training databases, from which neural networks of the appropriate architectures were designed and implemented, the results reflected an ability to solve the specified path-planning and astrodynamics problem to an acceptable degree of accuracy.

2.2.1.2 Learning from Demonstrations

As mentioned in Section 2.1.1, the ability to autonomously explore an uncharted environment, which underlying dynamics may not be fully realized until first rendezvous, will require that the path-planning logic have a capability to generalize and adapt in response to changing environments. Achieving generalized solutions may be stem imposing stochasticity in the dynamics; in other words, by perturbing the relevant dynamical parameters over a given range and randomly solving each run via independent Monte Carlo trials, a training database via rigorous methods may be generated. However, this may take a considerable amount of time to convergence given the right combination of parameters. In such cases, it may be more intuitive to leverage the innate human ability at finding generalized solutions for a complex task. This hypothesis is the driving factor behind the research field of learning from demonstrations (or LfD as commonly referred to in machine learning literature). Though a majority of the LfD experiments have been centered on robotic movement and multi-link arm control [38], the concepts may be applicable to spacecraft path-planning as well. To the best of our knowledge, the investigations presented as part of this work represent the first instance of leveraging human logic to generate a database of generalized spacecraft path-planning solutions [39, 4], but to also attempt autonomous cloning via supervised learning of this knowledge [40].

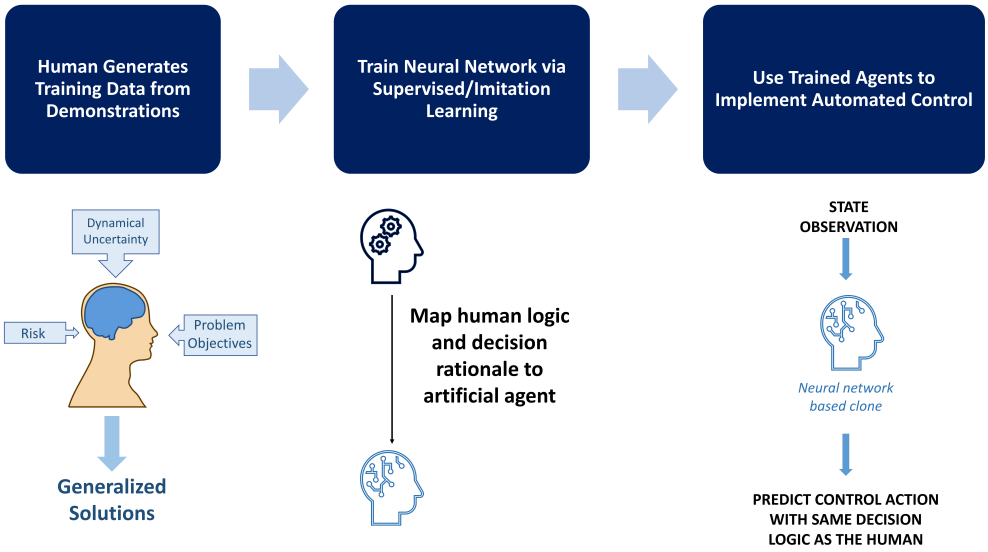


Figure 2.3: A visualization of the generalized path planning pipeline for training an artificial learning agent from human demonstrations

The general path planning development pipeline based on human driven demonstrations is illustrated in Figure 2.3. For path planning problems in extremely complex and varying dynamics, the innate human ability to build causal models and quickly adapt strategies may be leveraged to achieve generalized solutions that inherently balance the task objectives, risk, and dynamical variances. This dataset can then be used to train an artificial agent, similar to the path planning pipeline for training via database generation that was previously discussed.

There are considerable challenges that arise with using a human to generate the demosntrations for the training dataset. First, as mentioned in the previous section, when training an artificial agent with a training dataset that contains labels of the correct outputs, there is an assumption that the data represents ground truth. However, humans are not perfect and can have mistakes as they are generating their demonstration data. This leads to a severe reduction in quality of data, which will ultimately impact the accuracy of the trained agent. An additional challenge is combating the difference in perception between human logic and artificial logic [30, 31, 41]; in other words, even though both the human

demonstrator and artificial agent were asked to accomplish the same task, they may learn how to do so in different ways; this may be attributed to varying internal representations of the task, as well as variance in the perception (i.e. human's can accomplish the task from a visual input, whereas artificial agents may only have numeric data) [41]. The challenge of getting both the human demonstrator and artificial agent to not only think alike, but to do so while perceiving in the same manner, is currently an open field of academic research, especially in the context of spacecraft path planning. There exist various methods, such as domain transfer and data augmentation [42], that may be used to map the information from the human demonstrator's knowledge domain to the artificial agent's knowledge domain, in order to combat the differences in perception. This step may be implemented prior to the training of the true neural network.

To summarize, when wanting to shift towards machine learning based methods to achieve greater path planning autonomy, the implementation of those that rely on the generation of training datasets may seem the most logical route. However, as this approach may be classified as supervised, or imitation, machine learning, the achieved accuracy of the artificial agent is largely dependent on the quantity, quality and diversity of the dataset. Additionally, during the policy rollout, if the agent encounters an observation that is has never experienced before, as part of the original training dataset, then it may not be able to produce an accurate prediction, leading to a compounding of errors as the agent will then begin to experience states it has never encountered before. This phenomenon may be alleviated by generating a vast dataset that encompasses a large range of possible scenarios, but this may be further limited by the amount of computational time. To combat this, it may be logical to leverage the innate human intuition at solving a path planning problem, and ask a human demonstrator, in an intuitive manner, to generate the training dataset across a wide range of dynamics. However, using demonstrations provided by a human will require additional measure to be taken to account for mistakes and uncertainties, as well as successfully being

able to transfer perception logic between the human demonstrator and the artificial learning agent.

2.2.2 Environment based and Interactive learning

2.2.2.1 Reinforcement Learning

Reinforcement learning (RL) is one of the major branches of the machine learning taxonomy, along with supervised and unsupervised learning. However, in comparison to the latter two, which require the *a priori* existence of a dataset, RL methods are based on agent-environment interactions, which are then used to train the learner agent policy [43]. The fundamental framework of reinforcement learning can be visualized in Figure 2.4. An agent, which may be represented by a neural network(s), interacts with a given environment by observing the current state, and predicting the most appropriate action given that state. After the agent's selected action has been implemented, the agent will receive feedback in the form of a scalar reward that quantifies how "good" the action was. The cumulative rewards, after a certain amount of interactions, or steps, will be then used to update the agent's behavior, or policy. The exact method of these updates vary depending on the algorithm that may be chosen, with the relevant details being included in the corresponding publications of the algorithm.

As mentioned before, supervised learning based methods require the existence of a training dataset, whereas RL based methods do not. Rather, RL algorithms build their own training datasets by continual interactions with the environment, making RL based methods a subject of increasing research interest. Within the realm of RL, the importance of the reward function formulation as the driving factor behind the agent's learning process cannot be understated. The reward function can be thought of as a quantified representation of the goal, or task, that the agent is being asked to accomplish. For example, moving towards a certain target point may result in increasingly positive reward, whereas moving away from the target point may result in increasingly negative rewards. It is these scalar feedback

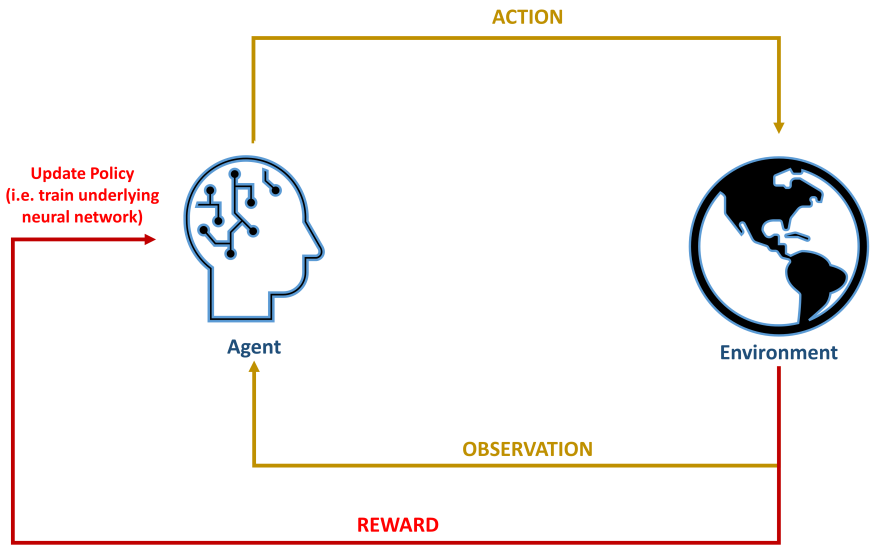


Figure 2.4: A generalized visualization of the reinforcement learning method

signals that guide the agent towards the desired behavior, and ill-defined reward functions can have disastrous consequences during the learning process; hence, great care must be placed when designing the reward function for a given task for which the RL agent is being implemented for. Furthermore, a majority of the RL algorithms that may be applied to more complex problems are heavily reliant on the learning hyperparameters itself, which change for each application of the problem. Within the context of machine learning, the term hyperparameters refers to the collective set of scalars that control the learning process of the RL algorithm. Some examples of hyperparameters include the learning rate, the size of the neural network architecture, and any additional parameters that are used in conjunction with the cumulative rewards during the policy update process.

Along with selecting the appropriate hyperparameters, the user must also fully understand the "collaborative" nature of the hyperparameters as well. For example, selecting a large learning rate, which essentially controls the magnitude of the updates to the neural network, but having a relatively small neural network architecture, are known to lead to overtrained RL agents. Conversely, defining a small learning rate and having a significantly

large neural network architecture will result in increasingly long training times, even more so when the given task for the RL agent to solve is dynamically complex. These underlying synergies are generally not extensively discussed in machine learning literature, and instead may rather develop from user experience and intuition. Therefore, RL methods, especially in the context of dynamically challenging problems, may require the most user expertise and intuition as compared to supervised learning approaches.

Despite these challenges, numerous researchers have investigated the application of RL methods to spacecraft path planning in multi-body environments, which are inherently dynamically complex problems. Recent investigations have analyzed the application of RL techniques to spacecraft stationkeeping [9], multibody transfers [10, 11, 12, 13, 14], and real-time guidance schemes [15, 16, 17, 18]; additional applications are briefly discussed in the survey paper provided by [44]. Though these applications also provided promising results in regards to the accuracy of the agents, as well as their feasibility in application, it should be noted that these results were characterized by long training times. Though they may offer more robust solutions, RL based methods are known to also generally require extensive training times, along with strategic modifications to the underlying algorithm chosen, for the most appropriate application to the problem at hand.

2.2.2.2 Real-Time Human-Agent Interaction

In the previous sections, the distinctions between supervised learning and reinforcement learning were introduced and discussed as separate and independent entities. However, their fundamental concepts can be merged into a synergistic framework, which may be referred to as "online human-agent interaction", or "cooperative human-agent learning". This framework has an agent accomplishing a task in real time, under the supervision of a human expert. If the human deems that the agent is proceeding in a risky, or unsafe manner, then the human can intervene via a control authority gating switch, and impose a corrective control maneuver that can guide the agent back towards operating in a safe or desired manner. The

human expert's corrections are then stored in the running dataset that the agent is collecting during rollout (the dataset generally consists of tuples comprising of state/observation and action pairs). If the human expert intervened, then the corrective action is recorded in the data tuple, otherwise the agent's own selected actions are recorded. This dataset is then used to retrain the agent via supervised learning, as the data tuples are already labeled with the corrective actions.

This field of research may be regarded as being relatively novel in comparison to the independent research areas of supervised and reinforcement learning themselves; however, there have been some applications of the cooperative human-agent learning framework [45, 46, 47, 48, 49, 50, 51, 52]. Some applications have been for low dimensional state space problems, such as the cart pole problem and simulated car racing [45, 46, 47, 48, 49], whereas others have been applied to higher dimensional scenarios such as Atari games and simulated multi-link manipulations [50, 51, 52]. To the best of our knowledge, the application of this framework specifically for astrodynamical applications is a novel contribution of this work.

The utilization of cooperative human-agent learning frameworks requires the understanding of the "control authority gating function". In essence, this gating function determines the instances when the human can override the agent's actions in real-time, and execute explicit control. This gating function can be vastly variant for each problem application, and can be regarded as an additional heuristic within the learning framework, as there is seemingly no set approach at designing the gating function. For example, the human can execute control authority whenever they deem fit, in order to guide the agent back to a safe behavioral mode [45]; conversely, the human's control authority can only activate when the agent's prediction confidence falls below a certain threshold [52]. Therefore, the appropriate method at designing the control authority switching function will be dependent on the nature of the problem, and additional user expertise and intuition will be required for an appropriate formulation.

The use of the control authority switching function implies that the human expert can theoretically intervene during the entire training process. However, if the desired result is a fully autonomous machine learning based agent that does not require and human feedback/corrections during final execution, then an alternative approach may be required. One possible alternative to the control authority switching function may be the provided via the "cycle of autonomy" framework [53], in which the authors investigated the utilization of different forms of real-time human interaction during the learning process in order to safely train an autonomous agent. The framework consisted of three primary phases: learning from pure human demonstrations, real-time cooperative learning between agent execution and corrections from the human, and fully independent execution and re-training of the agent via reinforcement learning. The transitions between each phase were internally triggered when the agent consistently achieved increasing performance metrics [53].

To summarize, the distinct methods of supervised learning and reinforcement learning can be merged into a synergistic framework in which a human expert provides real-time corrections to an autonomous agent, which are then used during the iterative updates to the underlying neural networks. However, incorporating a human in the learning framework, as well as providing them with control authority at certain instances, may require additional considerations as to when human intervention is deemed necessary, along with the appropriate formulation of the control authority switching function. Though there have been some applications of this framework for autonomous robotics and simulated, low dimensional environments, along with the application to spacecraft path-planning are presented in this work, the method of real-time human-agent interaction remains a considerably novel approach that is an active research area.

2.3 Investigation of Synergies between Distinct Path-Planning Methods

Both the baseline driven and machine learning based method at achieving path-planning solutions have generally been regarded and investigated as distinct approaches.

However, the possibility of leveraging their innate synergies in order to create more robust frameworks and autonomous solutions may have only been explored to a minimal degree. It may be possible that one singular approach may not be the most applicable for developing fully autonomous path-planning strategies, and therefore the utilization of components from each of these methods may be warranted. Therefore, understanding the inherent advantages and drawbacks of each method (baseline design via optimal control methods, pure machine learning methods, and human inspired machine learning) will help determine the most appropriate synergistic path-planning framework that not only yields autonomous, but also generalized, solutions.

This work aims to assess each of these distinct methods, and their ability at solving a benchmarking path-planning task. Quantifications of the results, along with the performance of the trained machine learning models, can then be utilized to assess the inherent advantages, drawbacks and possible synergistic frameworks. The following chapter discusses the technical theory and mathematical formulations for each of the methods implemented.

Chapter 3

Technical Background

3.1 Primary Components for Developing Spacecraft Path-Planning Solutions

The primary component needed for the development of a spacecraft path-planning solution is the underlying dynamical model, which quantifies the temporal evolution of the orbital state. There are numerous models that can be used to represent the environmental dynamics, each representative of differing fidelities. The simplest model is the two-body model, which assumes a spacecraft of negligible mass orbiting a central, gravitationally attracting body. Conversely, higher fidelity dynamical modeling can be provided by the n-body, or the ephemeris, model, which takes into consideration the gravitational accelerations imparted on a spacecraft by any number of celestial bodies. Further modifications to account for non-spherical gravitational fields, solar radiation pressure (SRP), and errors in state determination can also be included to increase the fidelity. This work primarily utilizes the Circular Restricted 3-Body Problem (CR3BP), which is further discussed below.

3.1.1 Circular Restricted 3-Body Problem (CR3BP)

The CR3BP dynamical model quantifies the motion of a spacecraft of negligible mass under the influence of two gravitationally attracting celestial bodies, which are commonly referred to as the "primary" bodies. The CR3BP model is derived in the rotating frame $(\hat{x}_{\text{CR3BP}}, \hat{y}_{\text{CR3BP}}, \hat{z}_{\text{CR3BP}})$, with $\hat{x}_{\text{CR3BP}}$ extending from the larger primary P_1 through the smaller/secondary primary P_2, $\hat{z}_{\text{CR3BP}}$ orthogonal to the mutual orbital plane, and $\hat{y}_{\text{CR3BP}}$ completing the right handed coordinate system [54]. The origin of the rotating frame is located at the barycenter of the system; this frame is illustrated in Figure 3.1. The dynamics

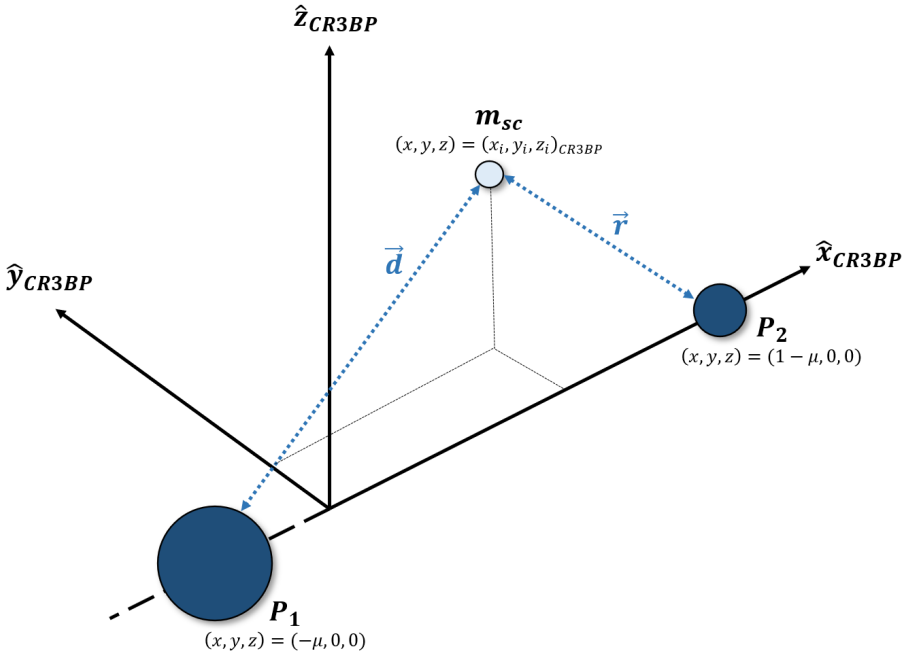


Figure 3.1: A visualization of the CR3BP dynamical model

can be determined by a single, nondimensional parameter μ, which denotes the ratio of the mass of the smaller primary relative to the total mass of the system

$$\mu = \frac{m_{\mathrm{P2}}}{m_{\mathrm{P1}} + m_{\mathrm{P2}}} \tag{3.1}$$

where m_{P1} and m_{P2} are the masses of the larger and smaller primaries, respectively. Given this quantity, the locations of the primaries can be denoted as

$$(x, y, z)_{P1} = (-\mu, 0, 0) \tag{3.2}$$

$$(x, y, z)_{P2} = (1 - \mu, 0, 0) \tag{3.3}$$

and the underlying equations of motion can be derived as

$$a_x - 2\dot{y} = \frac{\partial U^*}{\partial x} \tag{3.4}$$

$$a_y + 2\dot{x} = \frac{\partial U^*}{\partial y} \tag{3.5}$$

$$a_z = \frac{\partial U^*}{\partial z} \tag{3.6}$$

where U^* is the pseudopotential function

$$U^* = \frac{1-\mu}{d} + \frac{\mu}{r} + \frac{1}{2}(x^2 + y^2) \tag{3.7}$$

$$d = \sqrt{(x+\mu)^2 + y^2 + z^2} \tag{3.8}$$

$$r = \sqrt{(x-1+\mu)^2 + y^2 + z^2} \tag{3.9}$$

d and r are the distances of the third particle from the larger and smaller primaries, respectively [54].

Within the CR3BP model, there exist five equilibrium points at which the state has zero relative velocity and acceleration, in the rotating frame, relative to the primaries. These points are commonly referred to as the Lagrange points, and their locations within the system can be computed. Three Lagrange points are located along the $\hat{x}_{\text{CR3BP}}$ axis and are termed as the "collinear Lagrange points"; their locations can be numerically computed by solving the equation

$$\frac{\partial U^*}{\partial x} = \frac{1-\mu}{(x+\mu)^3}(x+\mu) + \frac{\mu}{(x+\mu-1)^3}(x+\mu-1) - x \tag{3.10}$$

with a root finding iterative scheme, such as Newton-Raphson method [54]. As the collinear Lagrange points are located along the $\hat{x}_{\text{CR3BP}}$ axis, their locations will only have a non-zero component for the x position. Therefore, Equation (3.10) is solved in terms of x, and the

locations of these points are provided as

$$(x, y, z)_{\text{L1}} = (x_1, 0, 0) \tag{3.11}$$

$$(x, y, z)_{\text{L2}} = (x_2, 0, 0) \tag{3.12}$$

$$(x, y, z)_{\text{L3}} = (x_3, 0, 0) \tag{3.13}$$

The remaining two Lagrange points are referred to as the triangular equilibrium points, and are located between the two primaries. Their locations can be analytically computed via

$$(x, y, z)_{\text{L4}} = \left(0.5 - \mu, \frac{\sqrt{3}}{2}, 0\right) \tag{3.14}$$

$$(x, y, z)_{\text{L5}} = \left(0.5 - \mu, \frac{-\sqrt{3}}{2}, 0\right) \tag{3.15}$$

It can be shown that from Lyapunov stability analysis, the collinear Lagrange points are unstable, whereas the triangular Lagrange points are stable; these stability characteristics may be leveraged for path-planning purposes.

3.2 Traditional Methods: Optimal Control Theory

3.2.1 Generalized Overview

Conventional approaches for solving the spacecraft path planning problem stem optimal control methods, which in turn are analytically derived from concepts found in Calculus of Variations. More specifically, the common methods of optimal control schemes are driven by the Euler-Lagrange equation [55], which is of the form

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\left(\frac{\partial F}{\partial y'}\right) = 0 \tag{3.16}$$

where F is a function with continuous first and second order partial derivatives with respect to all arguments. The Euler-Lagrange equation can be used to formulate the corresponding

Euler-Lagrange Theorem, which is formulated as follows. Let $J[y]$ be a functional of the following form

$$J[y] = \int_a^b F(x, y, y')\, dx \tag{3.17}$$

defined on a set of function *y(x)* which has continuous first order derivatives in the interval $[a, b]$ and satisfies the boundary conditions $y(x_1) = y_1$ and $y(x_2) = y_2$. The necessary conditions for the functional $J[y]$ to have a weak extremum for a given function *y(x)* is that *y(x)* satisfies the Euler-Lagrange Equations. It should be noted that the Euler-Lagrange Equation is a necessary condition for weak extrema, but as every strong extremum is also a weak extremum, the Euler-Lagrange equation is also a necessary condition for strong extremum. Additionally, the Euler-Lagrange Equation is only a necessary condition and not a sufficient condition.

3.2.2 Indirect Methods

Indirect Methods, also referred to as the variational approach, is formulated as follows. Consider a functional of the following form

$$J[x(t)] = \phi(x(t_f)) + \int_a^b L(\vec{x}, \dot{\vec{x}}, t)\, dt \tag{3.18}$$

where ϕ is the terminal cost at the final epoch t_f, the integral term represents the running cost along the solution trajectory, $\vec{x}$ represents the state vector, and $\dot{\vec{x}}$ represents the time derivative of the state vector. The terminal and running costs can alternatively be denoted as the Mayer and Lagrangian cost, respectively. Furthermore, we also consider that $\dot{\vec{x}}$ is not arbitrary, but constrained by the dynamics of the system; in other words,

$$\dot{\vec{x}} = \vec{f}(\vec{x}(t), \vec{u}(t), t) \tag{3.19}$$

where $\vec{u}(t)$ is some control vector and $\vec{f}$ is the representation of the state dynamics. The optimal control formulation will proceed to minimize the cost functional $J[x(t)]$ with respect to the state dynamics in order to achieve a solution.

The unconstrained indirect method formulation (i.e. one that does not take into consideration state-path constraints, and only considers the cost functional and the boundary constraints) can then be formulated in the following manner, via an analytical derivation process driven by calculus of variations. Given the set of state dynamics (i.e. $\dot{\vec{x}} = \vec{f}(\vec{x}(t), \vec{u}(t), t)$), we first formulate the Hamiltonian as [55]

$$H = L + \vec{\lambda}^T \vec{f} \tag{3.20}$$

where L is the Lagrangian cost term of the full cost functional $J[x(t)]$. The terms $\vec{\lambda}^T$ represent the co-state variables, mathematically denoted by Lagrange multipliers and represented by differential equations. The co-state equations can be interpreted as follows: if the set of differential equations that represent the constraints of the minimization problem are provided by $\dot{\vec{x}} = \vec{f}(\vec{x}(t), \vec{u}(t), t)$, then the co-states represent the marginal cost of violating these constraints. There is one co-state corresponding to each state variable in the dynamical model; therefore, if the state vector is represented by three position and three velocity terms, then there will be six total co-states within the indirect method formulation.

After the Hamiltonian has been derived, the differential equations for the co-states can be computed via the partial derivative(s) [55]

$$\vec{\lambda}^T = \frac{\partial H}{\partial \vec{x}(t)} \tag{3.21}$$

and the extremal control $\vec{u}(t)$ can be shown to have the form

$$\frac{\partial H}{\partial \vec{u}(t)} = 0 \tag{3.22}$$

wherein the above equation is then solved for the control term. It may be observed that this approach converts the optimization problem into a two point boundary value problem (TPBVP), as the initial and final boundary conditions are explicitly respected.

Once the unconstrained indirect method formulation has been derived, the implementation procedure in order to obtain a solution is relatively simple. First, the set of differential equations that represent the state dynamics (i.e. $\dot{\vec{x}}$) are expanded to include the derived co-state differential equations (i.e. $\vec{\lambda}$). In other words, along with propagating the set of differential equations that represent the state dynamics, we also propagate the derived differential equations for each of the co-states. This total set of differential equations are then implemented into an optimization routine, such as `fsolve` in MATLAB, which iteratively attempts to solve the minimization problem. Embedded within the optimization routine is a numerical integration routine, such as 4th order Runge-Kutta, that propagates the augmented state differential equation vector from the initial conditions (i.e. the initial boundary constraint), with a random initial guess of the costates. This initial augmented state vector is numerically propagated forward in time, after which the residual relative to the final boundary constraint is computed. At minimum, this residual is comprised of the error of the propagated state at the final epoch, relative to the final boundary constraint. There can be additional components included in the residual that are representative of stationarity and/or transversality conditions. If the norm of the residual term is less than some pre-defined tolerance, then the solution is assumed to have converged to optimality. If the residual does not fall below the tolerance, then the values of the initial guess of the co-states are updated such that the threshold criterion is met.

3.2.3 Direct Methods

Though the indirect methods may seem like a logical approach to solving optimal control problems, they may suffer from drawbacks, some of which can be summarized as follows:

1. The initial value of the co-states is determined heuristically, and not via any sufficient approximations. Therefore, the method can be informally regarded as a "trial and error" approach in order to achieve a solution, which may take a considerable amount of iterations should the problem complexity increase

2. If the state dynamics and/or the cost functional is changed, then all co-state equations must be re-derived before implementation

3. The simplest form of the indirect method, which is the unconstrained indirect method, does not include and equality/inequality (i.e. state-path) constraints along the trajectory. The solution only considers and satisfies the two boundary constraints.

In comparison to the unconstrained indirect method formulation, the direct method (which may be represented by the direct collocation formulation) do not require the extensive derivation of co-state differential equations [55].

The direct collocation methods converts the minimization problem into a discretized formulation, which is then optimized. The goal of the direct method can therefore be summarized as to transcribe the optimal control problem to a parameter optimization problem so that a non-linear programming (NLP) solver can be used. Though there are various approaches for the direct method, the commonly implemented formulation discretizes the continuous control problem such that the states and control variables are only known at certain times [55].

The direct collocation method is formulated as follows. We first discretize a set of continuous states $\vec{x}(t)$ and controls $\vec{u}(t)$ into N nodes such that

$$\vec{x}(t) \longrightarrow [\vec{x}(t_0), \vec{x}(t_1), \vec{x}(t_2), ..., \vec{x}(t_N)] \tag{3.23}$$

$$\vec{u}(t) \longrightarrow [\vec{u}(t_0), \vec{u}(t_1), \vec{u}(t_2), ..., \vec{u}(t_N)] \tag{3.24}$$

The boundary constraints of the problem are applied at the initial and final discretization nodes, and the state-path constraints (represented by equality/inequality constraints) are applied at all nodes. The direct method then proceeds to solve the optimization problem by solving the dynamics and control at each node.

Though we have transcribed the continuous control problem into a discretized formulation that is evaluated at each node, the overall solution must still result in a smooth approximation; in other words, there cannot be any discontinuities between the state variables between nodes. This feature of smooth approximation is ensured via the implementation of the defect equations as follows [56]. First, the solution at each node can be represented as

$$\vec{x}_{k+1} = \vec{x}_k + \beta \text{ for all } k \, \epsilon \, [0, ..., N-1] \tag{3.25}$$

where β is a linear/polynomial approximation of the Lagrangian component of the cost functional (this approximation is commonly done via trapezoidal or Hermite-Simpson methods). Re-arranging the above equation yields

$$\vec{x}_{k+1} = \vec{x}_k + \beta \longrightarrow \vec{x}_{k+1} - \vec{x}_k - \beta = 0 \tag{3.26}$$

which represent the defect equations as equality constraints. Given this formulation, the direct method can then utilize an NLP solver and achieve a converged solution.

3.3 Fundamentals of Machine Learning

Recent trends in shifting research focus and published literature are indicative of increasing interest in the use of machine learning methods in synergy with optimal control methods. The following sections briefly discuss the generalized taxonomy of neural network architectures, as well as how they can be designed and trained. Additionally, a section that provides a high-level, generalized overview of the machine learning taxonomy as a whole is also included.

3.3.1 Training a Neural Network

Prior to discussing the general types of neural networks that are used for machine learning models, it perhaps may make intuitive sense to first understand how neural networks in general are trained. The training process is generally common throughout all types of neural network based machine learning models, but each step in this process may lead to challenges that must be addressed by neural network architectures, hence leading to the varying architectures seen throughout modern literature which may be specifically designed to counter some of these issues.

The standard procedure for training a neural network may best be explained by considering a simple, fully connected neural network, as illustrated in Figure 3.2. All neural network architectures comprise of input, hidden/computational, and output layers, with the flow of information occurring in the stated sequence. Within each layer are individual units that are referred to as "nodes" or "neurons". All the nodes in one layer are combinatorially connected to each node in the immediate next layer; this is visualized in Figure 3.2 by black arrows. These connections are referred to as "synapses" and essentially denote a scalar weight that is utilized during the computation process. Additional to the weights, there are also biases between two immediate layers, which are also combinatorially connected to each node in the next layer. These biases are denoted by green circles in Figure 3.2. Within each hidden/computational and output layer, each node will perform some algebraic function computation of the incoming data provided by the synapses. The result of this function computation(H_i and O_i) is then sent to an activation function, which can be a linear or nonlinear function, to compute the activated function computation ($H_{i,\text{act}}$ and $O_{i,\text{act}}$). The final output of the neural network (i.e the $O_{i,\text{act}}$) is then utilized in some criterion measurement that represents a minimization problem, which is conversely defined and set by the user. The objective of the neural network training process is to incrementally update the values of the weights and biases such that the output prediction minimizes some criteria, such as the mean square error.

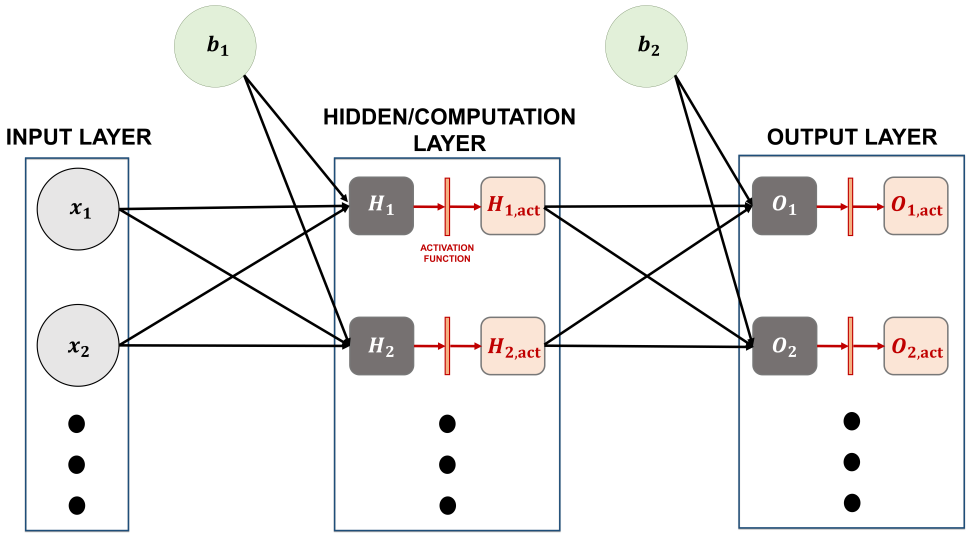


Figure 3.2: A simple fully connected neural network architecture (referenced from [1])

3.3.1.1 Forward Pass and Gradient Backpropagation

The training process of a neural network proceeds in an iterative manner, with the procedural loop repeating for a certain amount of epochs m_{train}, an additional parameter that the user defines.

At each epoch of the training process, the inputs from the dataset are passed to the neural network, which performs the necessary transformations and produces a predicted output. This process is referred to as a "forward pass" of the neural network. To empiricize the underlying forward pass computations, consider a small neural network architecture than the one previously illustrated in Figure 3.2. More specifically, consider a neural network that comprises of two inputs (x_1, x_2), one hidden/computational layer with two computational nodes, and an output layer with two outputs (O_1, O_2). This is illustrated in Figure 3.3.

Each forward pass through the neural network is computed as follows: given the set of inputs (x_1, x_2), their corresponding weights w_i, and the bias, each computational node in

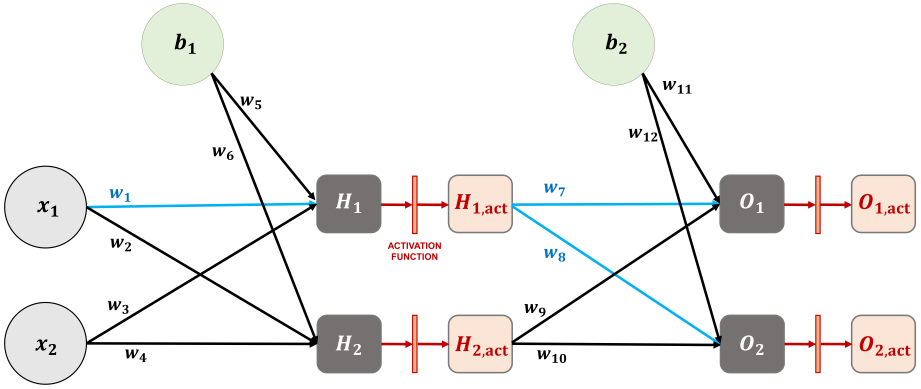


Figure 3.3: A basic fully connected neural network comprising of two inputs, one hidden layer with two computational nodes, and two outputs (referenced from [1]).

the hidden layer will compute the function approximation as the algebraic summation

$$H_1 = x_1 w_1 + x_2 w_3 + b_1 w_5 \tag{3.27}$$

$$H_2 = x_1 w_2 + x_2 w_4 + b_1 w_6 \tag{3.28}$$

These function approximations are then passed through an pre-defined activation function f_{act} to obtain

$$H_{1,\text{act}} = f_{\text{act}}(H_1) \tag{3.29}$$

$$H_{2,\text{act}} = f_{\text{act}}(H_2) \tag{3.30}$$

The utilization of activation functions is critical within the neural network architecture, as they essentially determine which nodes are "activated" (i.e. numerically contribute to the prediction computational process). Further details and examples of common activation functions are provided in a later section. Subsequently, the activated function approximations

$H_{i,\text{act}}$ are sent to the output layer, wherein each node computes the algebraic summation

$$O_1 = H_{1,\text{act}} w_7 + H_{2,\text{act}} w_9 + b_2 w_{11} \tag{3.31}$$

$$O_2 = H_{1,\text{act}} w_8 + H_{2,\text{act}} w_{10} + b_2 w_{12} \tag{3.32}$$

after which the output function approximations are also passed through an activation function to obtain

$$O_{1,\text{act}} = f_{\text{act}}(O_1) \tag{3.33}$$

$$O_{2,\text{act}} = f_{\text{act}}(O_2) \tag{3.34}$$

The values of $O_{i,\text{act}}$ are considered as the final output of the neural network, and a forward pass is considered to be completed once these quantities are computed. The final outputs are then used in the defined criterion, which represents the minimization objective. In the case of supervised learning (SL), the minimization objective is defined to be the error between the prediction and true observation (note that this error is also referred to as "loss" in ML literature). For simplicity, we assume the error e to be denoted as the mean square error between the final neural network outputs $O_{i,\text{act}}$ and the true observation y_i

$$e = \frac{1}{n_{\text{inputs}}} \sum_{i=1}^{2} (y_i - O_{i,\text{act}})^2 \tag{3.35}$$

where n_{inputs} is the number of inputs in the training dataset. Once the error e has been computed, the neural network uses this information to update the weights accordingly, via the process of gradient backpropagation. For simplicity, we assume that the weight update equation is provided by that of stochastic gradient descent (SGD) numerical optimization; therefore, at epoch k, the incremental weight update is computed as

$$w_i^{k+1} = w_i^k - \eta \frac{\partial e}{\partial w_i^k} \tag{3.36}$$

where η denotes the step size of the numerical optimization algorithm (in ML terminology, this term is referred to as the learning rate of the neural network). The partial derivative term in the above equation can be expanded via chain rule to obtain the full equation for all weight updates in the neural network. For example, consider the weight update for w_1; the expansion of the partial derivative term yields

$$\frac{\partial e}{\partial w_1^k} = \left[\frac{\partial e}{\partial O_{1,\text{act}}}\frac{\partial O_{1,\text{act}}}{\partial O_1}\frac{\partial O_1}{\partial H_{1,\text{act}}}\frac{\partial H_{1,\text{act}}}{\partial H_1}\frac{\partial H_1}{\partial w_1}\right] + \left[\frac{\partial e}{\partial O_{2,\text{act}}}\frac{\partial O_{2,\text{act}}}{\partial O_2}\frac{\partial O_2}{\partial H_{2,\text{act}}}\frac{\partial H_{1,\text{act}}}{\partial H_1}\frac{\partial H_1}{\partial w_1}\right] \quad (3.37)$$

and the update to weight w_1 is therefore

$$w_1^{k+1} = w_1^k - \eta\frac{\partial e}{\partial w_1^k} \quad (3.38)$$

This process is repeated for each weight w_i in the neural network architecture at each epoch (i.e. after each forward pass of all inputs within the training dataset). Given this brief overview, the generalized algorithm for training a neural network is detailed in Algorithm 1, and each step is briefly discussed.

3.3.1.2 Defining the Neural Network Architecture

First, the user must define the neural network architecture that is to be used for the machine learning model. As previously mentioned, neural network architectures pass information through sequential layers that are categorically referred to as the input, computational/hidden, and output layers. It is well known that the architecture size and shape (i.e. how many computational/hidden layers, and how many computational units/neurons/nodes within each of these layers respectively) drives the performance of the machine learning model. The generally accepted guiding principle is that shallow neural networks train faster, but may also result in less accurate solutions during training; conversely, larger and deeper neural networks require significantly more training time but may provide more accurate

Algorithm 1 Generalized Training Process for a Neural Network

1. Define a neural network architecture that follows the structure of input, computational/hidden, and output layers
2. Define the numerical optimization scheme and learning rate (along with any other hyperparameters for more complex learning algorithms)
3. Define the loss criterion
4. For each iteration for a maximum number of epochs m_{train}
   (a) Pass the training dataset as inputs to the neural network
   (b) Compute the loss (i.e. a quantification of the error) between the network's predicted outputs and the true outputs
   (c) Compute the gradients
   (d) Perform gradient backpropagation to update the neural network weights
5. Repeat Step 4 until the learning has converged

solutions. Furthermore, the specific design of the size and shape of the neural network architecture becomes an exercise of user expertise and intuition, as there is currently no general universally accepted method to designing an architecture that is most applicable to the problem or task that the user intends to solve.

3.3.1.3 Selecting a Numerical Optimization Scheme, Learning Rate, and Loss Criterion

After the neural network architecture has been defined, the numerical optimization scheme must be defined. Mathematically, optimizers function as their name indicates, in which they solve an optimization problem by minimizing some cost function that is defined by the user. As mentioned before, neural networks learn by the continual updates of weights, which are represented as the combinatorial connections between each computational node in adjacent hidden layers. The changing of these weight values will progress the learning process of the neural network; how the weight values are updated are heavily driven by the numerical optimization routine that is chosen.

The numerical optimization routine will minimize the defined loss criterion; the loss

criterion can be thought of as analogous to the cost functional in the context of optimal control. The form of these loss functions can vary depending on the type of algorithm that is implemented. For example, training via supervised learning can occur via use of the simple mean square error between the predicted and true outputs. Conversely, reinforcement learning based algorithms must utilize specific formulations of the loss criterion; the full derivation of these specific loss functions, as well as the significance of each component, are generally included in the relevant publications that first introduce new RL algorithms.

Consider a neural network with some set of weights $\mathbf{W}_{\text{ij,current}}$, which is a matrix representation of the current weights of each node j at each layer i. At each iteration during the training process, the weights are updated from the current set $\mathbf{W}_{\text{ij,current}}$ to a new set $\mathbf{W}_{\text{ij,new}}$ via the mathematical transformation

$$\mathbf{W}_{\text{ij,new}} = \mathbf{W}_{\text{ij,current}} - \eta f_{\text{optimizer}}(Q(\mathbf{W}_{\text{ij,current}})) \tag{3.39}$$

where η represents the learning rate, or step size, $Q(\mathbf{W}_{\text{ij,current}})$ represents the value of the objective, or cost function that is to be minimized with respect to the weight values, and $f_{\text{optimizer}}$ represents the numerical optimization routine. As can be seen from this equation, the size of the learning rate proportionally drives the weight updates; therefore, great care must be exercised when selecting this parameter value as too large of a learning rate will result in wildly varying weight updates and an unstable neural network, whereas too small of a learning rate will result in significantly larger training times. There are various numerical optimization algorithms that are available, but two of the commonly used algorithms are Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam).

1. **Stochastic Gradient Descent (SGD)**: Stochastic Gradient Descent is an extension of the base Gradient Descent algorithm, and may reduce the memory requirement that is warranted by the latter. Whereas Gradient Descent computes the first order derivatives over all points in the dataset, SGD computes the derivatives of the error

(i.e. loss) at each point. However, though this may reduce the computational memory required, the SGD optimization routine may also require longer times to convergence, and may also suffer from being unable to escape from local minima and saddle points. The SGD numerical algorithm updates the weights within a neural network via

$$\mathbf{W}_{\text{ij,new}} = \mathbf{W}_{\text{ij,current}} - \eta \frac{\partial e}{\partial \mathbf{W}_{\text{ij,current}}} \tag{3.40}$$

where e represents the error/loss, η represents the learning rate, and $\frac{\partial e}{\partial \mathbf{W}_{\text{ij,current}}}$ represents $f_{\text{optimizer}}()$

2. **Adam**: Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions that is based on adaptive estimates of lower-order moments. The algorithm updates exponential moving averages of the gradient ($\hat{m}_t$) and the squared gradient ($\hat{v}_t$), and the algorithm hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages [57]. The moving averages themselves are estimates of the mean and the uncentered variance of the gradient [57]. These moment vectors which are then included within the weight update equation via

$$\mathbf{W}_{\text{ij,new}} = \mathbf{W}_{\text{ij,current}} - \eta \frac{\hat{m}_t}{\hat{v}_t + 10^{-8}} \tag{3.41}$$

where the term 10^{-8} is included in the denominator to avoid any zero quantities that would result in gradient explosions during the backpropagation. Further details and the full derivation of the algorithm can be found in [57]. Compared to SGD, the Adam optimization algorithm has been found to accelerate the learning process by decreasing the number of function evaluations required to reach the minima.

3.3.2 General Categories of Neural Networks

It is imperative to note that the selection of the appropriate neural network architecture is critical to the development of a successive machine learning model. Though there are various architectures of neural networks that can be selected from, they can be broadly categorized into three distinct categories, though all types are trained in the same method as summarized in Algorithm 1. The three distinct categories of neural networks are generally referred to as artificial/feed forward, convolutional and recurrent neural networks; this work explored architectures belonging to each category. A brief overview of each type is further discussed below.

3.3.2.1 Artificial/Feed Forward Neural Networks (ANN/FFNN)

As denoted by the name, the feed forward neural network is an architecture that only moves the information forward. Consider the general form of a feed forward neural network, as illustrated in Figure 3.4; it can be observed that the feed forward neural network has the same architecture as that of the network in Figure 3.3 that was utilized to demonstrate the training process. There are three main layers that comprise a standard feedforward network: the input, hidden, and output layers. The input layer represents the initiation of the workflow of the neural network by accepting the raw input values and passing them to the internal layers for further processing. The hidden layer performs the appropriate nonlinear transformations of the inputs; depending on the complexity of the problem, the architecture may have multiple hidden layers. Lastly, as the name implies, the output layer takes the transformed information from the subsequent hidden layers and processes it for the desired output, such as predicting the category that the input data belongs to, or a scalar value for predictive regression tasks.

Given this general overview of the feed forward network architecture, the mathematical theory behind neural networks can now be briefly discussed. As illustrated in Figure 3.4, each layer in the architecture is comprised of nodes/neurons (denoted by circles) and synapses

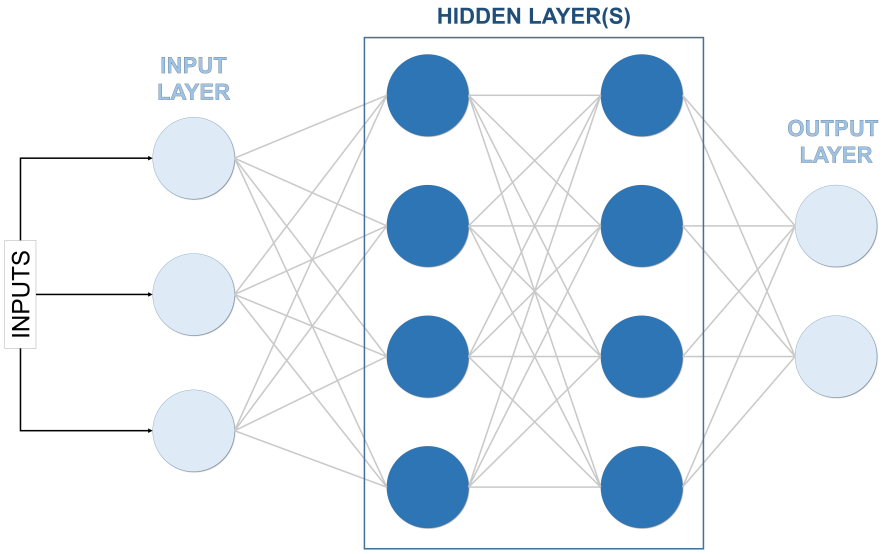


Figure 3.4: Conceptual architecture for a feed forward neural network. The three main layers are the input, hidden, and output layers, with the flow of information proceeding from left to right as illustrated.

(denoted by straight lines connecting each node in one layer to all the nodes in the next layer). The relationship between each node and synapse is one of the driving factors behind how a neural network learns a given task; to further understand this relationship, consider a single synapse feeding into a single node, as visualized in Figure 3.5. Each synapse n will bring to the node some output scalar, x_n, from a previous node, multiplied by a weight $\mathbf{W}_{ij}$. In other words, the synapse will take the output from a node in the previous layer and apply some weight via multiplication before passing the information to the next node. The recipient node (denoted by blue circle in Figure 3.5) will compute the sum of the weighted inputs, and apply a bias, which are used to adjust the intermediary output Y_{node} via

$$Y_{node} = \sum_{1}^{n} \boldsymbol{W}_{ij} x_n + \mathrm{b}_{ij} \tag{3.42}$$

This intermediary output Y_{node} is then passed as the mathematical argument to some user defined activation function, the output of which is passed to the outgoing synapse, subsequent

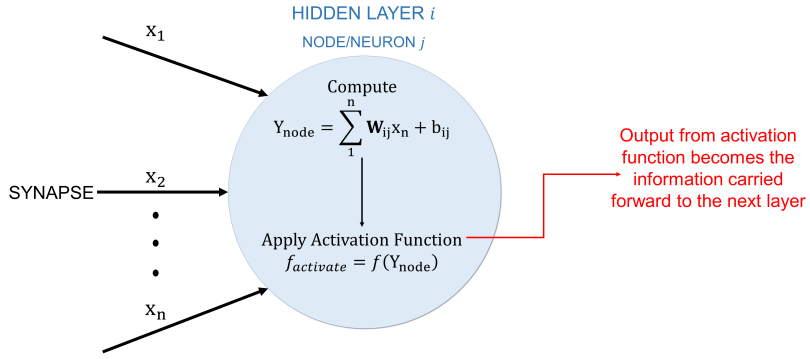


Figure 3.5: A visual representation of the inner-workings of a node in a neural network.

to which the process repeats for each node-synapse relationship in the architecture. The use of activation functions in any neural network are required as they can help the architecture recognize complex patterns within the data. There are multiple activation functions that can be used (such as hyperbolic tangent, sigmoid, etc.), which are further discussed in Section 3.3.2.4.

3.3.2.2 Convolutional Neural Networks (CNN)

Convolutional neural networks are the most prevalent for computer vision based tasks, with the goal being to enable the machine to both view and perceive the world in a similar manner to humans. Along with image processing, CNN can be used for classification and segmentation tasks. A generalized convolutional neural network architecture, with the primary layers, is illustrated in Figure 3.6. The layers of the typical CNN architecture can be divided into those that focus on feature extraction, and those that focus on classifying the input into one of multiple categories.

Following the input to the neural network, which may be in the form of a multi-

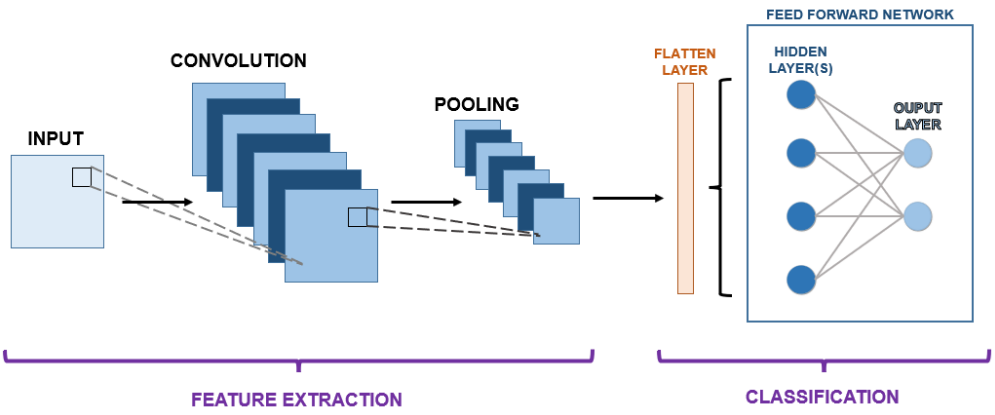


Figure 3.6: A generalized visualization of the convolutional neural network architecture, with the commonly used layers labeled.

dimensional tensor to represent a greyscale or color image, the CNN architecture then applies a series of convolutional layers across the data. Each convolutional layer contains a series of filters that is referred to as convolutional kernels; each filter is a matrix of integers that is used on a subset of input pixel values. Each pixel is linearly scaled by the corresponding value of the kernel, with all results being summed up for a single value representing a grid cell in the output feature map. The amount of convolutional kernels applied within the convolutional layer represents the amount of feature maps that are being created for learning the input feature representations.

Following the convolutional layer is the pooling layer, which is utilized to reduce the spatial size of the convoluted features computed in the previous layers. This decreases the computational power required to process the data, and may also be useful for extracting dominant features from the convolutional layer feature maps. There are two general types of pooling layers that can be utilized: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the kernel, whereas the Average Pooling returns the average of all values from the portion of the image covered by the kernel. This sequential pattern of convolutional and pooling layers is repeated as many times as necessary, subject to the judgement of the user.

Subsequent to the pooling layer, which is the last of the feature extraction layers of the

CNN architecture, we must prepare the resultant data for the classification portion. This requires the use of a flattening layer, in which we transform the multidimensional output from the pooling layers into a one dimensional column vector representation that can be easily passed into a standard feed forward architecture.

3.3.2.3 Recurrent Neural Networks

Within the context of machine learning, recurrent neural networks (RNN) have increasingly gained the reputation of being the most promising and capable type of network that may be implemented for sequential/temporally driven datasets. Compared to the standard feed forward and convolutional neural networks, recurrent neural networks may be better equipped to recognize underlying patterns within sequential/temporal data inputs, and subsequently use them to make accurate forecasting based predictions. This is made possible by the underlying architecture of the RNN having an internal memory capability, which allows the network to form an inner representation of given sequential data.

The primary difference between an RNN and a standard feed forward neural network can be visualized in Figure 3.7. As mentioned before, standard feed forward neural network simply passes the information forward through each computational layer of nodes. In comparison, each of the nodes in the recurrent layer of the RNN not only pass the information forward to the next layer, but the output of each node is looped back into the node. Therefore, each of the computational nodes in a RNN have information of both the current and recent past, allowing it to perform significantly better on sequential prediction tasks. Aside from this difference in the architecture, the RNN architecture is also trained in the same manner as the CNN and the feedfoward by updating the weights of each synapse during the gradient backpropagation.

There are four primary types of RNN that can be implemented, the categorization of which is driven by the types of inputs provided to the network, and the type of output that is being requested:

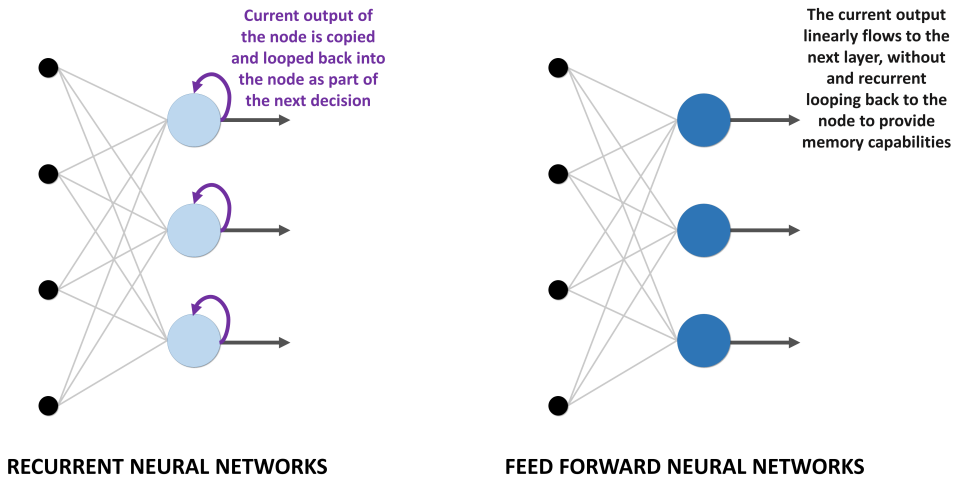


Figure 3.7: A high level comparison between the general architectures of a recurrent neural network and a feed forward neural network (referenced from [2])

1. **One to One**: the most traditional and simple form, the one to one architecture provides a single output for a single input

2. **One to Many**: given a single input, the prediction contains multiple outputs, which may represent multiple categories for a classification task

3. **Many to One**: given multiple inputs, the network will provide a single output

4. **Many to Many**: given multiple inputs, the network will provide multiple outputs

The above four labels can be commonly seen throughout the literature on RNN.

Though RNN architectures may seem to offer a promising approach, they also suffer from a few critical drawbacks. The standard RNN must be equipped to counter two major problems that may arise during the gradient backpropagation phase of the training process. As mentioned before, the gradient represents how much the output of a function changes when the inputs are only slightly changed. Within machine learning context, the gradient

measure the change of all weights with respect to the change in the prediction error. Therefore, a higher gradient will result in faster learning, whereas gradient values near zero, or in some cases exactly zero, result in significantly slow learning or no learning at all. Standard RNN neural networks suffer from two known gradient based problems in machine learning: exploding and vanishing gradients [58]. Exploding gradients can be a by-product of the accumulation of error gradients during an update of the neural network. These accumulations may result in very large gradients, which in turn results in large updates to the network weights and ultimately leading to unstable neural networks. Conversely, vanishing gradient occur when the neural network has a large number of hidden layers, and subsequently leads to the gradient partial derivative term to be near zero as the backpropagation process continues (i.e. the weights closer to the input layer of the architecture may only be updated by an immensely small amount, or not at all). To counter these issues, as well as to better enhance the ability of the network to recognize long term dependencies within temporal data, the Long-Short Term Memory (LSTM) network was developed as an extension of the standard RNN [59]. The general architecture of an LSTM unit, or cell, that is implemented within each node of the recurrent layer(s) is illustrated in Figure 3.8.

The fundamental components of an LSTM unit are the cell state, denoted as c_i in Figure 3.8, and the hidden state, denoted as h_i. Hidden states are analogous to a working memory, or short term memory, and carry a representation of the immediately previous input state. Cell states are analogous to a long term memory that is capable of remembering beyond the immediately previous input state, and is updated via an algebraic calculation that is a function of the network's weights, biases and the hidden state. The LSTM unit has the capability of adding or removing information within the cell state via the use of regulated gates.

The flow of information through an LSTM unit is textually described in the following paragraphs [3]; subsequently, an example is provided to clarify the dimensions of each computational stem within the LSTM cell [60]. Given an input x_t as some epoch t, the first

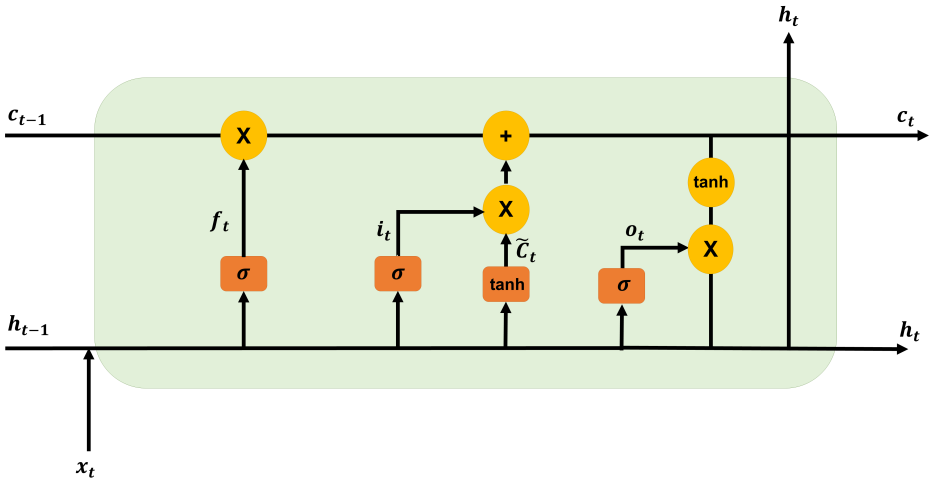


Figure 3.8: A generalized visualization of an LSTM unit architecture (referenced from [3])

computation occurs through the "forget gate layer". This initial step within the LSTM unit essentially decides the information that will be discarded from exclusion within the cell state. The forget gate f_t directly evaluates

$$f_t = \sigma(\mathbf{W}_f \times x_t + \mathbf{U}_f \times h_{t-1} + \mathbf{b}_f) \tag{3.43}$$

where $\mathbf{W}_f$, $\mathbf{U}_f$ and $\mathbf{b}_f$ are the weights and biases of the forget gate, h_{t-1} is the previous value of the hidden state, $\times$ indicates the matrix multiplication, and σ is the sigmoid function. If the forget gate computes a value close to zero, than the information is not passed to the cell state; conversely, if the value is close to one, then the information is deemed appropriate to include in the cell state.

The forget gate essentially decides if the information should be included in the cell state; the next step determines what information should be included. This decision occurs in two steps: first, the "input gate layer" decides what values in the cell state will be updated via the computation

$$i_t = \sigma(\mathbf{W}_i \times x_t + \mathbf{U}_i \times h_{t-1} + \mathbf{b}_i) \tag{3.44}$$

where $\mathbf{W}_i$ and $\mathbf{U}_i$ are the weights of the input gate, $\times$ indicates the matrix multiplication, and $\mathbf{b}_i$ are biases. Then, a hyperbolic tangent operation is utilized to create a vector of candidate values $\bar{C}_t$ that could potentially be added to the cell state

$$\tilde{C}_t = \tanh(\mathbf{W}_c \times x_t + \mathbf{U}_c \times h_{t-1} + \mathbf{b}_c) \tag{3.45}$$

Once the vector of candidate values has been computed, the cell state itself is updated via

$$c_t = f_t c_{t-1} + i_t \bar{C}_t \tag{3.46}$$

Finally, the output of the LSTM unit is computed as

$$o_t = \sigma(\mathbf{W}_o \times x_t + \mathbf{U}_o \times h_{t-1} + \mathbf{b}_o) \tag{3.47}$$

where $\mathbf{W}_o$ and $\mathbf{U}_o$ are the weights, and $\mathbf{b}_o$ are the biases of the output function. Once the output has been computed, the hidden state is updated via

$$h_t = o_t \tanh(c_t) \tag{3.48}$$

The dimesionality of each computational step within the LSTM cell can seem confusing at first sight, and therefore an example computation is provided in Table 3.1, with the assumed dimensions of the input and output vector being explicitly stated.

3.3.2.4 Activation Functions

As alluded to briefly in previous sections, the use of activation functions plays an important role in the performance of a neural network. Fundamentally, the activation function defines how the weighted sum of the input is transformed into an output from a node or nodes in a given layer of the network. The selection of the appropriate activation function is not only driven by the type of neural network, but also the type of layer (i.e. hidden or

Table 3.1: Example dimensionality for each computational step within an LSTM cell

Assumed Input: x_t	10x1		
Assumed Output: o_t	2x1		
LSTM Cell Layer	**Dimension**	**LSTM Cell Layer**	**Dimension**
Forget Gate: f_t	**2x1**	**Output Gate:** o_t	**2x1**
$\mathbf{W}_f$	2x10	$\mathbf{W}_o$	2x10
$\mathbf{U}_f$	2x2	$\mathbf{U}_o$	2x2
$\mathbf{b}_f$	2x1	$\mathbf{b}_o$	2x1
Input Gate: i_t	**2x1**	**Candidate Cell State Values:** $\tilde{C}_t$	**2x1**
$\mathbf{W}_i$	2x10	$\mathbf{W}_c$	2x10
$\mathbf{U}_i$	2x2	$\mathbf{U}_c$	2x2
$\mathbf{b}_i$	2x1	$\mathbf{b}_c$	2x1
Cell State Update: c_t	2x1		
Hidden State Update: h_t	2x1		

output) that the function is being applied on. There are various activation functions that can be used, but a few of the commonly used ones, along with the general scenarios in which they should be used, are discussed below [61]. We begin with introducing the activation functions that are commonly

1. **Rectified Linear Unit (ReLU)**: The ReLU activation is generally the most utilized for hidden layers as it is not only easy to understand and implement, but it is also less susceptible to the problem of vanishing gradients when compared to other activation functions for hidden layers. The ReLU activation is computed via

$$f_{\text{ReLU}} = \max(0, Y_{node}) \tag{3.49}$$

   where Y_{node} is the value computed by a node in a hidden layer. As can be discerned from the empirical formulation above, the ReLU activation function truncates all negative outputs to zero.

2. **Logistic**: Also referred to as Sigmoid activation, this function maps all inputs to within the bounded range of [0,1]. The algebraic formulation for the Logistic activation

function is given by

$$f_{\text{Logistic}} = \frac{1}{1 + e^{-Y_{node}}} \tag{3.50}$$

From this Equation, it can be seen that a more positive input Y_{node} will result in the activation function returning a value closer to one, whereas a more negative input Y_{node} will result in the activation function returning a value closer to zero.

3. **Hyperbolic Tangent (Tanh)**: The Hyperbolic Tangent activation function behaves similarly to the Logistic activation function, however the outputs are bounded between [-1,1] instead of [0,1]. The Hyperbolic Tangent activation function is formulated as

$$f_{\text{Tanh}} = \frac{e^{Y_{node}} - e^{-Y_{node}}}{e^{Y_{node}} + e^{-Y_{node}}} \tag{3.51}$$

The above three activation functions are generally utilized after each hidden layer; however, there also exist certain activation functions that can be used to transform the neural network output into the desired format. The following are some of the commonly used activation functions that can be additionally included after the output layer, though the user can choose to omit the inclusion of these transformations should they deem fit.

1. **Linear**: This activation function may alternatively be referred to as the "identity" activation function in some literature. The linear activation function does not perform any algebraic transformation on the values resulting from the output layer of the neural network, and returns the exact values that are predicted.

2. **Logistic (Sigmoid)**: Functioning in the same manner as the Logistic activation function for the hidden layers, the Logistic activation for the output layer binds the output within the range [0,1].

3. **Softmax**: The Softmax activation function binds the values from the output layer to a vector of values, the summation of which is always one; in other words, the Softmax

activation function is utilized to represent the probability of classification of the input to one of multiple, discrete output categorizations.

The selection of the appropriate activation function for the output may generally be driven by the type of problem that the machine learning model is being asked to solve [61].

3.3.3 Generalized Taxonomy of Machine Learning (ML)

Lastly, each of the three main categories of neural network architectures discussed above can be applied for various types of machine learning applications. The landscape of ML algorithms may seem daunting and hard to navigate, especially considering the vast amount of critical details that are generally not included in common literature, and instead left as an exercise to the user to realize and understand. Therefore, this section is included within this work for the purposes of intuitively explaining, at an extremely high level, the generalized taxonomy of machine learning methods.

Most of the algorithms can be classified as part of a single taxonomy, with the three main branches/categories being unsupervised learning, supervised/imitation learning, and reinforcement learning. A high level, extremely condensed taxonomy is provided in Figure 3.9, and the following sections briefly discuss each of the methods listed in the visualization.

3.3.3.1 Unsupervised Learning

Unsupervised learning (USL) is where the user provides an unlabeled dataset (i.e. state/observation inputs do not have corresponding output labels), and asks that the ML agent itself learn how to classify, label and/or group the data points. In other words, USL requires the ML agent to independently identify inherent patterns within the dataset.

1. **Dimensionality Reduction** [62]: Dimensionality reduction refers to the process of transforming high-dimensional data into a low dimensional representation, where the latter retains meaningful properties of the original data. Dimensionality reduction is commonly used to counter the curse of dimensionality, a well known phenomenon that

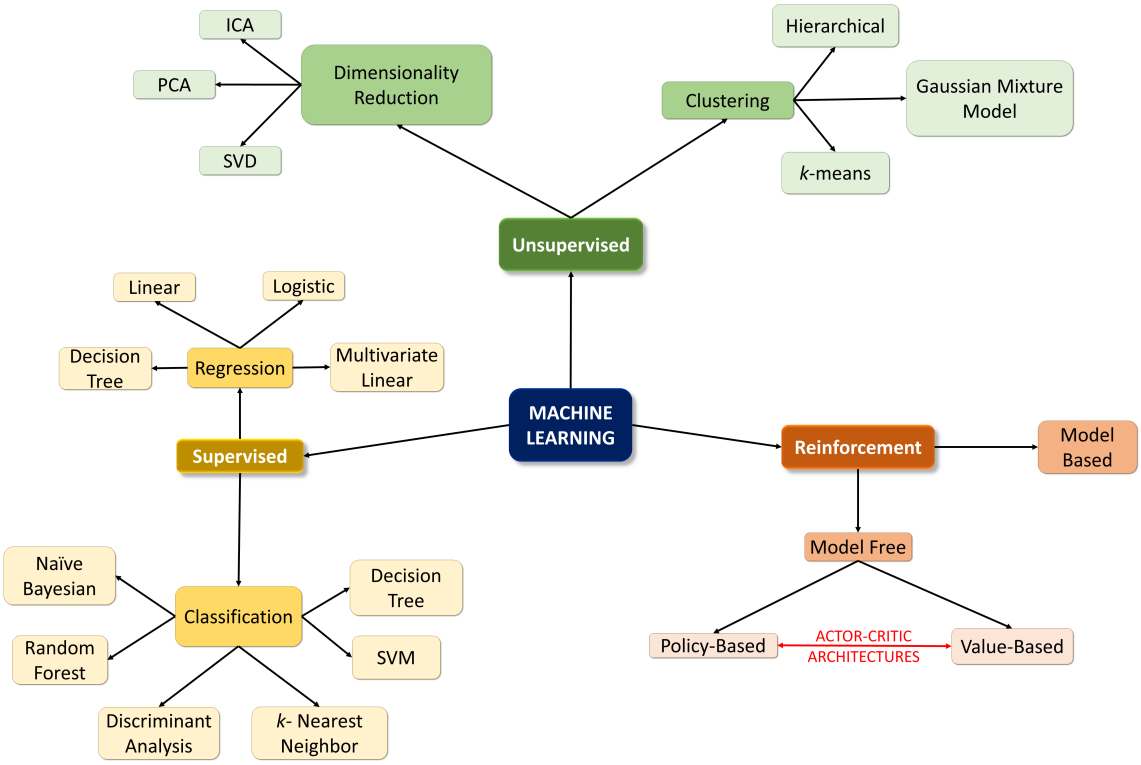


Figure 3.9: Generalized Taxonomy of Machine Learning Methods

arises when analyzing and organizing data in high-dimensional spaces. Additionally, these methods can be used for noise reduction, data visualization, cluster analysis, or as a data pre-processing step for other ML methods.

- **Independent Component Analysis (ICA) [63, 64]**: ICA is a linear dimension reduction method that transforms the dataset into columns of independent components. The method assumes that each sample of the data is a mixture of independent components, and aims to determine them.
- **Principal Component Analysis (PCA) [65, 66, 67]**: PCA is one of the more popular dimensionality reduction methods utilized in ML, and similar to ICA, is also a linear dimension reduction. PCA is a projection based method that transforms the data by projecting into a lower dimension via the eigenvalue analysis.

Appropriately implemented PCA will result in a high accuracy representation of high dimensional data via projection into a low dimensional space.

- **Singular Value Decomposition (SVD) [68, 69, 70, 71]**: SVD is a matrix factorization method that transforms the data linearly into independent, unrelated properties. SVD will provide all the independent components that can be used to represent high dimensional data, whereas PCA will select the most representative features. Therefore, PCA can be interpreted as a specific formulation of SVD.

2. **Clustering**: As the name implies, clustering is a machine learning technique that divides the dataset into groups, such that all data points within one group share some characteristic, and all created groups are characteristically different from each other. Often times, clustering may be a first step in the machine learning process, as it allows the user to understand their dataset and assess the need for any consequent modifications.

    - **Heirarchical [72]**: Also referred to as heirarchical cluster analysis, heirarchical clustering is an algorithm that groups similar objects into independent groups called clusters, within which each data sample is broadly similar to the others.
    - ***k*-means [73]**: These methods aim to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. The value of k is a user-defined parameter of the algorithm.
    - **Gaussian Mixture Model [74]**: Gaussian Mixture Models are probabilistic models that assume all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. Mixture models may be thought of as generalizing k-means clustering to incorporate information about the covariance structure of the data.

3.3.3.2 Supervised Learning

In contrast to USL methods, supervised learning (SL) methods are provided a training dataset that already contains the true data labels. The artificial agent is then trained by computing the error between the predicted and true labels, and using this information as part of learning updates. Within SL, there are two categories for predicting: classification and regression. Each of these is further explained below, along with some corresponding methods that are generally used.

1. **Classification**: Classification predictive models approximate a mapping function from input variables to discrete output variables, which can represent labels or categories. A classification algorithm can have both discrete and real valued variables, but it requires that the examples be classified into at least two classes.

    - **Decision Tree [75]**: This method builds a decision tree where every node of the tree is a test case for an attribute and each branch coming from the node is a possible value for the attribute.
    - **Support Vector Machine (SVM) [76]**: The objective of SVM algorithms is to find a hyperplane in an c-dimensional space (where c signifies the number of features) that distinctly classify the data points.
    - ***k*-nearest neighbor (*k*-NN [77])**: This algorithm assumes that similar things exist in close proximity to each other. It uses feature similarity for predicting values of new data points, and attempts to group similar data points together according to their proximity. The main goal of k-NN algorithms is to determine how likely it is for a data point to be part of a specific group.
    - **Discriminant Analysis**: Discriminant analysis is a classification problem, where two or more groups/clusters are known *a priori* and one or more new observations are classified into one of the known categories based on the measured characteristics.

- **Random Forest [78]**: This method builds a set of decision trees that are randomly generated from a subset of the main training set. The algorithm then aggregates outputs form all the different decision trees to decide on the final output prediction, which is more accurate than any of the individual trees.
- **Naive Bayesian [79]**: The Naive Bayesian classifier is a probabilistic model that is derived the the Bayes Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.52}$$

which computes the probability P of event A happening, given that event B has already occurred. This algorithm assumes that the features that drive the prediction determination are independent, and that one particular feature does not affect the other (i.e. there is no causal dependence between the features).

2. **Regression**: In contrast to classification tasks, where the learning agent is asked to categorize the input variable(s) into one of multiple, discrete categories, regressive SL algorithms are tasked with predicting continuous values, that may exist within some range. The goal of regression algorithms is to determine the best-fit line/curve between all the data points.
   - **Decision Tree**: Decision trees that are used for regression tasks may also be referred to as "regression trees"
   - **Linear**: A linear regression ML model will find the best fit linear line between the independent and dependent variables (i.e. it will find the linear relationship).
   - **Logistic**: Logistic regression models the probability of a discrete outcome given an input variable. Whereas linear regression is used for data which may vary between some range of values, logistic regression may be used for data that represents one of two values.

- **Multivariate Linear**: The multivariate regression technique learns a single regression model that is capable of predicting more than one outcome variable

3.3.3.3 Reinforcement Learning (RL)

Both the unsupervised and supervised learning methods assumed the existence of a dataset that is used to train a model for a prediction task. In comparison, RL methods are driven by rewarding desired behaviors and penalizing/punishing undesired ones. The general framework for RL is onen in which an agent continuously interacts with an environment via actions, receiving feedback, in the form of a reward, that quantifies how good that action/behavior was, and subsequently updates its learned policy in the direction of the desired behavior. Mathematically, the RL framework solves the Markov Decision Process (MDP), which in turn is based on the Bellman Equation [43]. The Bellman Equation outlines a framework that determines the optimal expected reward R at a current state s, and can be fundamentally described as

$$V(s) = \max_a \left(R(s,a) + \gamma V(s')\right) \tag{3.53}$$

where a is the action, V is the expected return, or value, γ is the discount factor (i.e. a hyperparameter of the RL algorithm, and varies in the range $\gamma \in [0,1]$. Setting γ closer to zero indicates that the model only considers immediate reward, whereas γ values closer to one also considers future rewards.), and s' is the next state. In more simpler terms, Equation (3.53) can be read as "the expected return at the current state s is the maximum value of any possible action a for the expected reward for taking action a at state s, plus the discount factor multiplied by the value of the next state" [43].

There are two general categories of RL methods: model based and model free. Each of these, along with their further sub-categories, is briefly summarized below.

1. **Model Based**: Model based RL methods learn optimal policies indirectly by learning a representation of the environment [43]. This is done by continually taking actions and observing the consequent outcomes, along with the value of the feedback signal quantified by the reward. These representations can be used to predict the outcomes of certain actions, and may be used as a proxy to the environment interactions in order to learn optimal policies

2. **Model Free**: Conversely, model free RL methods do not utilize the transition probability distribution associated with Markov Decision Processes, and only concern themselves with determining the appropriate action/behavior given a specific state that the agent is currently in [43]. Model free methods can be alternatively viewed as "trial and error" algorithms, and can be further divided as being either policy-based or value based.

   - **Policy-Based**: Policy-based methods explicitly learn a mapping π from a state s to an action a (i.e. the method maps $\pi : s \longrightarrow a$).
   - **Value-Based**: Conversely, value-based methods do not learn a mapping scheme, and only store information on the value function.

   Furthermore, model-free methods can also be classified as either on-policy (which evaluates and/or improves its current learned policy as it continually interacts with the environment) or off-policy (which evaluates and/or improves a policy that is different from the one used to interact with the environment and generate the training data) [43].

3. **Actor-Critic Architectures**: Actor-Critic architectures leverage both policy and value based RL algorithm approaches, by deploying two agents in parallel; both agents are modeled via independent neural networks [43]. The actor agent learns to predict the appropriate action for a given state, whereas the critic agent learns to compute how "good" the value of the action was for the given state. The value function computed by the critic agent is then used to update the policy of the actor agent's neural network.

Chapter 4

Experimental Formulation and Methodology

This chapter provides the specific details on the investigation that was implemented to compare path-planning solutions that were generated using various approaches. More specifically, the full derivations and appropriate machine learning model architectures are discussed in the sections below.

4.1 Numerical Experiment Framework

We evaluate the implemented approaches via a numerical formulation for a baseline path-planning problem. This numerical investigation serves two primary purposes: to explore the inherent advantages and drawbacks of the various methods that may be leveraged to achieve autonomous spacecraft path planning, as well as to initially establish a comparative strategy that can be used to further assess the underlying trade offs and possible synergies of each method.

The utilization of optimal control approaches, such as indirect and direct collocation methods, are generally regarded as one of the standard procedures for designing spacecraft path planning solutions. However, with increasing research interest in the application of machine learning, we aim to understand how data-driven approaches stand in comparison to the conventional optimal control methods within the context of spacecraft path-planning. Such an analysis may not only serve as an indication of how to merge these independent approaches into a synergistic framework, but may also bring to realization the necessary advances in the state of art required to apply machine learning methods to the scale of complex astrodynamical applications.

When designing a baseline trajectory, the solution must satisfy the mission and spacecraft constraints; for example, these constraints may be quantified by flyby velocities, radial distances from the neighboring primary celestial bodies, fuel consumption, and even the attitude pointing direction of the spacecraft. In order to meet these constraints, the optimal control approaches may require further modifications within the optimization framework. The same concept applies to machine learning methods; the underlying neural network and algorithm hyperparameters can be modified and tuned in order to achieve the best performance within the defined constraints. *Simply put, the best modification of the method will change depending on the problem that it is being applied to.* This subsequently leads to diminishing value of attempts at quantified comparison of the performance of the methods. Therefore, in this work, a choice is made to not implement any additional modifications that would bias the performance of a given method in relation to the others. Such modifications include the optimization of neural network architectures and hyperparameters, and additional inclusions in the optimal control methods that result in the converged solution to fit a preferred solution range. As a consequence of this choice, the comparison of results provided in this study are therefore limited.

Figure 4.1 illustrates all the methods of autonomous path planning that are explored within this work. The methods can be generally grouped into three primary categories: optimal control (i.e. indirect and direct collocation methods), supervised/imitation learning (i.e. cloning from a database of optimal solutions, and cloning from human demonstrations), and reinforcement learning. The synergistic combination of reinforcement and supervised learning yields the novel field of human-AI cooperative learning, which is further explored in Chapter 6 as an extended application of autonomous path-planning frameworks in random space environments. The formulation and implemented methodology of each of these methods, aside from that of human-AI cooperative learning, are further discussed in the following subsections.

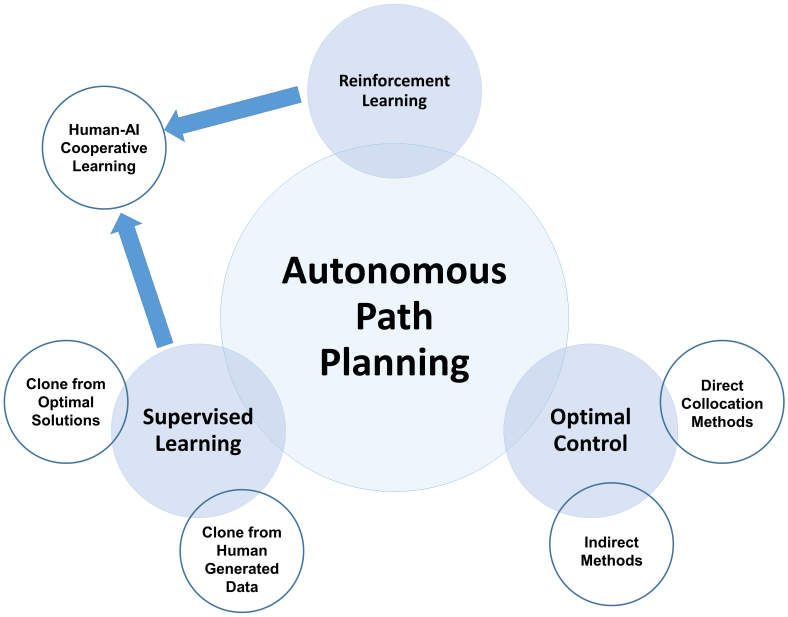


Figure 4.1: A visualization of possible methods for autonomous path planning methods for this investigative work

4.2 Problem Background

This section discusses the path-planning problem that was developed and implemented. The relevant technical details of the applied methods are previously discussed in Section 3.

4.2.1 Definition of the Problem Statement

We consider a specific path-planning task to test the selected methods via the implementation of a "sandbox environment" approach. The implemented path planning problem comprises of a low thrust, minimum time transfer within a binary asteroid system (these environments may be inherently characterized by chaotic dynamics, hence leading to an increasing complexity of the path planning problem). More specifically, the transfer is assumed to occur from the L4 Lagrangian point to the L5 Lagrangian point of the system (these points may also be referred to as the triangular Lagrangian points). These Lagrangian points were chosen not only because the spacecraft state at these locations is

explicitly known, but also because the triangular Lagrangian points may be able to serve as stationing candidates due to their dynamical stability [80].

This work utilizes the CR3BP dynamical model (equations previously provided in Chapter 3), with the additional simplifying assumption of planar dynamics. Therefore, the driving equations of motion for the planar CR3BP dynamical model are

$$\ddot{x} = 2\dot{y} + \frac{\partial U^*}{\partial x} \tag{4.1}$$

$$\ddot{y} = -2\dot{x} + \frac{\partial U^*}{\partial y} \tag{4.2}$$

where $\ddot{x}$ and $\ddot{y}$ represent the planar accelerations, and U^* is the pseudopotential function. However, as we are solving a control problem, the dynamical influence of the control term must be implemented within the base dynamics. The control term $\boldsymbol{u}$ is defined as

$$\boldsymbol{u} = [u_x, u_y]^\top = T_{\max}\delta[\cos\alpha, \sin\alpha]^\top, \tag{4.3}$$

where $\delta \in [0, 1]$ and $\alpha \in [0, 2\pi]$ denote the magnitude and direction of the control vector, respectively and $T_{\max}$ denotes the constant maximum thrust produced by the propulsion system. Therefore, the inclusion of the acceleration due to the thrust control vector yields the following equations of motion for the spacecraft

$$\ddot{x} = 2\dot{y} + \frac{\partial U^*}{\partial x} + \frac{T_{max}}{m}\eta u_x \tag{4.4}$$

$$\ddot{y} = -2\dot{x} + \frac{\partial U^*}{\partial y} + \frac{T_{max}}{m}\eta u_y \tag{4.5}$$

where m denotes the spacecraft mass. The modeling of the state estimation is implemented under deterministic assumptions; in other words, there are no additional perturbations or uncertainties in the spacecraft state

$$\vec{x} = [x, y, \dot{x}, \dot{y}, m_{\text{spacecraft}}]^T \tag{4.6}$$

which is explicitly computed through the numerical integration of the dynamical equations of motion via the computation of the time derivative of $\vec{x}$ as empiricized by

$$\dot{\vec{x}} = [\dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dot{m}]^T \tag{4.7}$$

The time rate of change of mass ($\dot{m}$) is given by

$$\dot{m} = -\frac{T_{\max}}{c}\delta, \tag{4.8}$$

where $c = I_{\text{sp}} g_0$ is the effective exhaust velocity and I_{sp} and g_0 denote specific impulse and sea-level gravitational, respectively. In this work, the specific impulse value remains constant along the extremal trajectories.

4.2.2 Dynamical Parameters and Simulation Constants

It is possible that the ability to achieve a solution, as well as the subsequent performance, may be inherently driven by the dynamical scenario that is encountered. In other words, it may become more difficult to achieve a path planning solution within certain dynamical environments. Therefore, to analyze the performance as a function of the underlying dynamics, each method was tested over different combinations of selected dynamical large range of scenarios that are randomly generated via independent Monte Carlo trials.

In particular, there are four dimensions of perturbations: the μ value, the maximum thrust value, and perturbations on both the initial position and velocity states. Each of these parameters were randomly selected from the ranges of these perturbations provided in Table 4.1; all other relevant dynamical values were fixed to constant scalars, whose values are given in Table 4.2.

Table 4.1: The perturbation ranges for the dynamical parameters of the baseline problem

Parameter	Perturbation Range
μ (ndim)	[0.001, 0.45]
Thrust (Newtons)	[1e-3, 50e-3]
Position Initial Condition (ndim)	[-0.2, 0.2]
Velocity Initial Condition (ndim)	[-0.2, 0.2]

Table 4.2: The fixed valued of dynamical parameters of the baseline problem

Parameter	Fixed Value
Characteristic Length (km)	3.75
Characteristic Time (hours)	4.19
P1 Diameter (km)	1.61
P2 Diameter (km)	0.49
Initial Spacecraft Mass (kg)	610
Isp (sec)	2000

4.3 Optimal Control Solution Formulation

The optimal control based analysis is implemented via a two tier controller that first implements the indirect method formulation, and then assesses the need to implement a direct collocation scheme on the generated solution in the event that the solution violates an inherent state path constraint. The derivation of each of these methods is further discussed in the subsequent sections, along with the implemented state path constraints and an overview of the hierarchical controller architecture.

4.3.1 Indirect Optimal Control Formulation

The first stage in the two tier controller implements an indirect method formulation, which requires the definition of a cost functional for the problem that is being implemented. For minimum-time trajectory optimization problems, the cost functional, J, can be written in the Lagrangian form as

$$\underset{\vec{x}\in\mathcal{X}(t)\ \&\ \vec{u}(t)\in\mathcal{U}}{\text{minimize}} \quad J = \int_0^{t_f} 1\, dt, \tag{4.9}$$

where $\mathcal{X}$ and $\mathcal{U}$ denote the set of admissible states and controls, respectively. Here, the initial time of maneuver is set to $t_0 = 0$ without loss in generality. This functional J can be interpreted as the running cost that must be minimized over the entire solution. The problem comprises of minimum time transfers, and therefore the integrand is unity and the optimization must occur in the temporal dimension by default. The set of differential equations $\vec{f}$ that propagate the state $\vec{x}$ forward in time are denoted by

$$\vec{f} = \begin{cases} \dot{x} \\ \dot{y} \\ \ddot{x} = 2\dot{y} + \frac{\partial U^*}{\partial x} + \frac{T_{\max}}{m}\delta\cos(\alpha) \\ \ddot{y} = -2\dot{x} + \frac{\partial U^*}{\partial y} + \frac{T_{\max}}{m}\delta\sin(\alpha) \\ \dot{m} = -\frac{T_{\max}}{c}\delta \end{cases} \tag{4.10}$$

For simplicity, we assume no throttling capabilities of the propulsion system, which results in the value of $\delta = 1$.

Let $\vec{\lambda} = [\lambda_x, \lambda_y, \lambda_{\dot{x}}, \lambda_{\dot{y}}, \lambda_m]^\top$ denote the costate vector associated with the state vector, $\vec{x}$. We can proceed by forming the variational Hamiltonian as

$$\begin{gathered} H = L + \vec{\lambda}^\top \vec{f} \\ \therefore H = 1 + \lambda_x\dot{x} + \lambda_y\dot{y} + \lambda_{\dot{x}}\left[2\dot{y} + \frac{\partial U^*}{\partial x} + \frac{T_{\max}}{m}\delta\cos(\alpha)\right] + \\ \lambda_y\left[-2\dot{x} + \frac{\partial U^*}{\partial y} + \frac{T_{\max}}{m}\delta\sin(\alpha)\right] - \lambda_m\frac{T_{\max}}{c}\delta \end{gathered} \tag{4.11}$$

where $L = 1$ is the Lagrangian cost function. The costate differential equations can be obtained using Euler-Lagrange equation as

$$\dot{\vec{\lambda}} = -\left[\frac{\partial H}{\partial \vec{x}}\right]^\top. \tag{4.12}$$

The optimal expression for $\sin(\alpha)$ and $\cos(\alpha)$ can be optimized using Pontryagin's minimum principle (PMP) and Primer Vector Theory of Lawden[81] as

$$\cos(\alpha^*) = -\frac{\lambda_{\dot{x}}}{\sqrt{\lambda_{\dot{x}}^2 + \lambda_{\dot{y}}^2}}, \qquad \sin(\alpha^*) = -\frac{\lambda_{\dot{y}}}{\sqrt{\lambda_{\dot{x}}^2 + \lambda_{\dot{y}}^2}} \tag{4.13}$$

Upon substitution of relations given in Eq. (4.13) into Eq. (4.11) and simplifying the Hamiltonian we have

$$H = H_0 - \frac{T_{\max}}{c}\left[\frac{c\sqrt{\lambda_{\dot{x}}^2 + \lambda_{\dot{y}}^2}}{m} + \lambda_m\right]\delta, \tag{4.14}$$

where the term in the bracket is the so-called thrust switching function. According to PMP, the optimal throttle, δ^*, has to minimize the Hamiltonian along an extremal trajectory. In general, the switching function may have multiple switches in its sign, which gives rise to bang-bang control profiles. In addition, it is also possible for the switching function to remain zero for a finite time interval which characterize *singular* control arcs (though singular arc controls rarely occur in space flight). Additionally, for minimum-time maneuvers, it is shown that the switching function remains positive along an extremal solution [82]. Thus, the optimal throttle becomes $\delta^* = 1$, which means that the propulsion system will operate at its maximum throttle setting. While not considered in this study, in the case of minimum-fuel trajectory optimization problems, it is possible to alleviate the numerical issue due to bang-bang control profiles by using a hyperbolic tangent smoothing (HTS) method [83] combined with a numerical continuation method.

The implemented problem consists of generating minimum-time transfers from the system's L4 to L5 Lagrangian points with their position vectors given in Equations (3.14) and (3.15) and with velocity vectors $\vec{v}_{\mathrm{L4}} = \vec{v}_{\mathrm{L5}} = \vec{0}$. Recall that the spacecraft mass is not fixed for the final time, and is in fact, a function of the time of flight, t_f. We are only concerned with generating transfer solutions between the triangular Lagrange points, and doing so with the shortest possible time of flight. There are two additional boundary constraints that must

be satisfied at the final time t_f. From stationarity conditions, the value of the Hamiltonian must satisfy $H(t_f) = 0$. In addition, the cost functional represents a minimum-time problem and it does not contain mass state explicitly; thus, the co-state associated with the mass value must also satisfy $\lambda_m(t_f) = 0$. Thus, the original optimal control problem has been reduced to solving a two-point boundary-value problem (TPBVP). In summary, the TPBVP associated with minimum-time maneuvers can be summarized with state dynamics, costate dynamics, extremal control expressions, Eq. (4.13), $\delta^* = 1$, and the boundary conditions $\Psi(t_i)$ defined as

$$\Psi(t_0 = 0) = \begin{bmatrix} \boldsymbol{r}(t_0) - \boldsymbol{r}_{L4} \\ \boldsymbol{v}(t_0) - \boldsymbol{v}_{L4} \\ m(t_0) - m_0 \end{bmatrix} = \mathbf{0}, \qquad \Psi(t_f) = \begin{bmatrix} \boldsymbol{r}(t_f) - \boldsymbol{r}_{L5} \\ \boldsymbol{v}(t_f) - \boldsymbol{v}_{L5} \\ \lambda_m(t_f) \\ H(t_f) \end{bmatrix} = \mathbf{0}. \tag{4.15}$$

The unknowns for the base minimum-time problem are seven initial values of the costate vector, $\vec{\lambda}(t_0)$, and the time of maneuver, t_f, which has to be determined. The resulting TPBVP does not have an analytical solution and has to be solved numerically. However, the problem is converted to a non-linear root-finding problem. In this paper, a single-shooting solution scheme based on Quasi-Newton method is used to solve different instances of the TPBVPs. Each instance corresponds to perturbation ranges per Table 1. MATLAB's built-in non-linear solver `fsolve` is used as the root-solver and its built-in numerical integrators `ode45` and `ode15s` are used for propagating the differential equations forward in time.

The implementation of this indirect method formulation echoes the standard approach that may be generally utilized for initial analysis and path planning design. However, it should be immediately noted that this formulation does not take into consideration any state path constraints, such as the trajectory remaining within certain bounds that can be geometrically empiricized by equality and/or inequality constraints. There exist methods that may allow for the inclusion of state path constraints within the indirect formulation [84],

but these are not implemented within the problem formulation as they would be considered additional modifications to the solution methodology, and bias the resulting analysis.

4.3.2 Direct Collocation Method Formulation

Given that the indirect method formulation may produce an infeasible solution, with feasibility being defined as there being no violations of the state path constraints, the solution must therefore be rectified via the direct collocation method. There are three state path constraints that the solution must adhere to, which are provided below, with the bolded scalars denoting additional safety distances from the primaries to ensure a more realistic solution.

1. *Constraint 1*: at any epoch t_i, the trajectory cannot go through the main primary P1, which has a radius R_{P1}

$$d(t_i)^2 + r(t_i)^2 \geq \mathbf{1.1} R_{P1}^2 \tag{4.16}$$

2. *Constraint 2*: at any epoch t_i, the trajectory cannot go through the secondary primary P2, which has a radius R_{P2}

$$d(t_i)^2 + r(t_i)^2 \geq \mathbf{1.5} R_{P2}^2 \tag{4.17}$$

3. *Constraint 3*: at any epoch t_i, the trajectory must remain within the outer bound of the binary system, which has a radius defined by the characteristic length (L^*), which is nondimensionalized for unit consistency

$$d(t_i)^2 + r(t_i)^2 \leq \mathbf{2} L^{*2} \tag{4.18}$$

In the above constraint equations, the terms $d(t_i)$ and $r(t_i)$ represent the distances from the larger and smaller primaries, respectively. The scalar values of **1.1**, **1.5**, and **2** were arbitrarily chosen for this analysis as a representation of safety constraints from a practical

context. These state path constraints are enforced at mesh points within the direct collocation method framework, which is based on the Hermite-Simpson transcription method [85] with the formulation details given in [86].

The full visualization of the two tier optimal controller is provided in Figure 4.2, and the flow of information proceeds as textually described above. The first stage of the controller is the indirect method formulation, which implements the derived co-state differential equations and generates an initial solution that adheres to all boundary constraints. Then, this solution is evaluated for feasibility, and if no state path violations occur, then the trajectory data is directly added to the solution database. If a state path violation is found to have occurred, then this solution is passed to the direct collocation method as the initial guess that requires corrections for the necessary constraints. Given that the problem scenario is modeling low thrust transfers, and the fact that the initial boundary conditions, system μ, and thrust value parameters will be randomly varied for each scenario, there is a possibility of the solution dataset comprising of a diverse topology. For example, certain combinations of perturbations may result in transfers that comprise of singular arcs, whereas other scenarios may result in multiple-revolution solutions. A solution with more revolutions may require more discretization nodes within the direct collocation scheme, and therefore a rough mesh refinement strategy is adopted within the two tier controller architecture. All solutions that are passed to the direct method for corrections are initially discretized with 75 nodes, after which the solution is re-converged. Subsequently, this solution is evaluated again to ensure that no arc segments between any nodes violate the state path constraints. If a violation is found to occur, then the number of discretization nodes is increased by 10, and the solution is re-converged. This iterative scheme continues until the solution is within the defined tolerances.

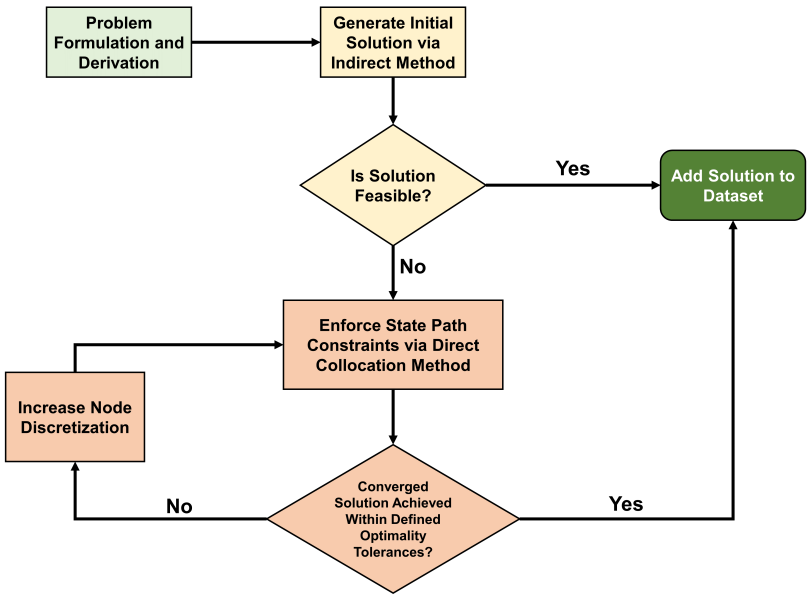


Figure 4.2: The two tier optimal controller architecture

4.4 Imitation of Optimal Control Solutions via Supervised Learning

4.4.1 Generation of an Optimal Solution Database

Multiple executions of the two tier controller architecture will result in the generation of optimal control solutions that represent minimum time transfers. As per the formulation of the two point boundary value problem. For each execution, not only are the non-dimensional mass parameter (μ) and thrust ($T_{\max}$) randomly selected from pre-defined ranges, and the initial state (both position and velocity) is also perturbed away from the true L4 state of the generated system. Therefore, the solution database will characterize a wide range of dynamical scenarios.

It was observed that the use of different numerical integration techniques yielded different solution geometries. More specifically, the use of standard 4th order Runge-Kutta integration resulted in more diverse trajectory geometries, such as multiple revolutions and close fly-by maneuvers of the larger primary. Conversely, the use of stiff numerical integration, such as `ode15s` in *MATLAB*, resulted in direct transfers around the smaller primary.

Taking this observation into account, approximately 10,000 solutions were generated using stiff numerical integration, and another approximately 1,500 solutions were generated using standard 4th order Runge-Kutta integration.

4.4.2 Training Clone Agents from Optimal Trajectory Solutions

The generated solution database can be directly utilized to train clone agents. As the trajectory solutions not only contain the state information, but also the control decisions, the dataset is fully labeled, therefore allowing us to use standard supervised learning methods to train clone agents.

We consider two options for the neural network architecture for the clone agents: the standard feed forward neural network, and the recurrent neural network modeled via the LSTM formulation. By the conventional formulation of the architectures, the feed forward network may only be able to consider the current state as the input in order to compute a control prediction. However, as all recurrent neural networks, and by relation the LSTM formulation, are capable of considering the temporal sequences as part of the training process. Additionally, during rollout of the LSTM based network, the input to the network can be either the current state only, or a rolling "lookback" window that not only contains the current state, but the immediately previous states for a certain amount of epochs. This latter characteristic may be of great benefit during the rollout of the trained agent.

For developing clone agents based on the optimal control solutions, both the feed forward and LSTM neural networks were utilized. However, an important modification must be made to the LSTM network prior to the training process. As previously discussed in Chapter 3, the LSTM network learns long term dependencies in temporal data via continual updates to the innate cell and hidden states. An important assumption that is made for all recurrent neural network architectures is the fact that the input data is characteristic of a single, temporally driven data stream. In our case, we have a solution database comprising of thousands of solutions, all of which are independent temporal data. Therefore, the

Table 4.3: The neural network architectures for the implemented feed forward model

Hyperparameter/Layer	**Version 1 (Deterministic Policy)**	**Version 1 (Stochastic Policy)**
State Feature Vector	$[x, y, \dot{x}, \dot{y}]$	$[x, y, \dot{x}, \dot{y}]$
Learning Rate	7e-4	7e-4
Optimizer	Adam	Adam
Criterion	Mean Square Error	Mean Square Error
Hidden Layer 1	Fully Connected, 250 nodes	Fully Connected, 250 nodes
Activation Function	tanh	tanh
Hidden Layer 2	Fully Connected, 125 nodes	Fully Connected, 125 nodes
Activation Function	tanh	tanh
Output Layer	1 node	1 node
Output Format	Predict α	Predict mean Sample α = Normal(mean, $\sigma = 0.1$)

Table 4.4: The neural network architecture for the implemented LSTM model

Hyperparameter/Layer	**Version 1 (Deterministic Policy)**	**Version 1 (Stochastic Policy)**
State Feature Vector	$[x, y, \dot{x}, \dot{y}]$	$[x, y, \dot{x}, \dot{y}]$
Learning Rate	7e-4	7e-4
Optimizer	Adam	Adam
Criterion	Mean Square Error	Mean Square Error
Hidden Layer 1	1 LSTM layer, 64 units	1 LSTM layer, 64 units
Activation Function	ReLU	ReLU
Hidden Layer 2	Fully Connected 200 nodes	Fully Connected 200 nodes
Activation Function	ReLU	ReLU
Output Layer	1 node	1 node
Output Format	Predict α	Predict mean Sample α = Normal(mean, $\sigma = 0.1$)

assumption behind recurrent neural networks becomes invalid, and modifications must be made to allow for this scenario. This modification comes in a surprisingly intuitive manner: after each independent temporal data has been passed through the LSTM network, and the training loss has converged to the minimal value, the internal memory of the LSTM units (i.e. cell and hidden states) are reset to zero before the next independent temporal data stream is passed through as the input. This modification will results in what is termed as the "stateless LSTM" network.

The full architectures of the implemented feed forward and LSTM networks are provided in Tables 4.3 and 4.4. For each of the two architectures, there were two variations: version 1 of each agent was trained to learn how to directly predict the thrust control direction α, whereas version 2 of each agent was trained to learn the prediction of a mean value, which is then utilized to sample the thrust control direction α from a Normal distribution with an assumed standard deviation of $\sigma = 0.1$. Additionally, during the training process for each variation of both networks, an early stopping criteria was implemented to avoid overfitting the model. All neural networks were coded using the Pytorch package in Python, with random weight initialization; the early stopping criteria was implemented via a hard-coded function routine

4.5 Human Driven Imitation Learning

4.5.1 Game Framework and Human Pilot Database Generation

The two tier optimal control architecture can be utilized as an analytical approach for generating trajectory solutions for a varying range of dynamics. However, posing this minimum time path planning problem to a human pilot, in the form on real time "gamified engineering" may also result in a diverse set of solutions. The innate human ability to not only build causal models of dynamics, along with the capability to immediately adapt a given behavior to a new scenario, may inspire complementary approached as compared to optimal control methods.

In order to evaluate the human ability at solving a minimum time transfer problem, a real time flight simulator was developed in Python that appropriately models the problem. Each run of this simulator is referred to as an episode; each episode begins with the random selection of the μ and thrust values, as well as the perturbations to the initial state. In other words, each episode models a different dynamical scenario and initial boundary conditions in the same method utilized to generate the solution database from the optimal control solution. The game interface is visualized in Figure 4.3, and provides the human pilot with

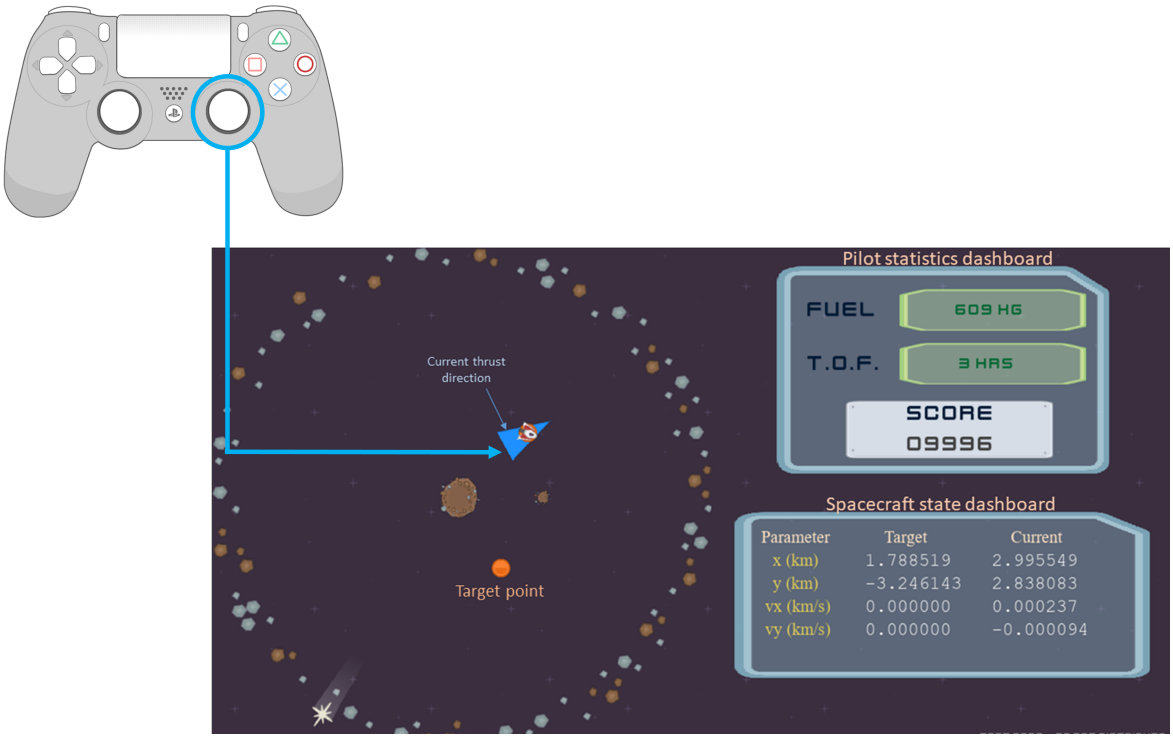


Figure 4.3: A screenshot of the developed, real-time flight simulator that models the minimum time L4-L5 transfer problem via the perturbations detailed in Table 4.1

relevant details that can be utilized as part of their path-planning decision process. Namely, the pilot statistics dashboard provides the current score, the time of flight, and the amount of fuel remaining. The spacecraft state dashboard provided the explicit quantifications of the current and target states. Lastly, the blue arrow located below the spacecraft icon denotes the direction of the thrust control vector relative to the positive tangent vector of the spacecraft.

The human pilot explicitly controls the direction of this thrust control vector via a joystick input on a PlayStation 4 controller. The angle β defines the orientation of the thrust control vector, relative to the current tangential velocity vector of the spacecraft. This angle β is directly implemented within the state dynamics in order to propagate the motion forward in time. Each control decision by the human pilot is propagated forward for 15 seconds in simulated time (though it may be small, this value was deemed to be the most appropriate through trial and error, as it was observed that and higher values resulted in

the inability of the human to appropriately control the spacecraft for high thrust scenarios).

In order to encourage the development of minimum time transfers, as well as achieving the final boundary condition as best as possible, the game interface includes an underlying scoring function that is directly visible to the human pilot. In particular, an inverse scoring scheme of the form

$$r(t_i) = s_{base} - s_1 t_i + \text{terminal bonus/penalty} \tag{4.19}$$

where t_i is the current epoch, s_{base} is a fixed scalar value of 10,000, s_1 is a scalar weight of 5, and the terminal bonus/penalty is determined by whether a termination criteria was detected. If the spacecraft crashed into either primary or left the asteroid system, the terminal penalty is -2,000; conversely, if the human was able to navigate the spacecraft to L5, and they deemed that they achieved the final state, then the terminal bonus is defined by the following scheme:

$$\text{terminal bonus} = \begin{cases} +500 \text{ if } \text{norm}(\vec{x}_{\text{final}} - \vec{x}_{\text{L5}}) \leq 0.25 \\ +1,000 \text{ if } \text{norm}(\vec{x}_{\text{final}} - \vec{x}_{\text{L5}}) \leq 0.1 \\ +1,500 \text{ if } \text{norm}(\vec{x}_{\text{final}} - \vec{x}_{\text{L5}}) \leq 0.05 \end{cases} \tag{4.20}$$

where $\vec{x}_{final}$ is the state at which the episode was terminated as the human pilot believed that they met the target state boundary condition.

Approximately 25 hours of training data from a single human pilot were collected; due to the small integration time step within the simulation dynamical propagate, this only yielded approximately 100 episodes of trajectory solutions.

4.5.2 Agent Training

In the same method utilized for training the clone agents from the solution database generated by optimal control methods, a clone agent was also trained from the human pilot

solutions. However, we only consider the LSTM formulation for the human pilot version, due to the fact that they may be better equipped to deconstruct the diverse human pilot path planning logic. Additionally, we only train one version of the clone, that learns to directly predict the value of the thrust control direction α. The architecture and hyperparameters utilized is the same as version 1 of the LSTM network utilized for the clone agents trained from the optimal control solution database (these values were previously provided in Table 4.4).

4.6 Reinforcement Learning: Proximal Policy Optimization (PPO)

4.6.1 Algorithm Overview and Hyperparameters

As previously discussed in Chapter 3, reinforcement learning (RL) agents have no direct knowledge of the underlying dynamics, and instead learn a behavior, or policy, by interacting with a simulated environment through predicted actions, and receiving scalar feedback signals quantifying "how good" the action was, in regards to either the new state or the previous state-action tuple. After a certain amount of interactions (also referred to as "steps" in RL terminology), the encountered states, the chosen action and the reward feedback signal are utilized to update the agent's policy through the computation of a loss function. The definition of the loss function for RL is different than that of supervised learning (which generally utilizes mean square error), and each RL algorithm has a distinct loss formulation that is specified in the corresponding literature. This process iteratively repeats until the policy has converged; this is generally visualized and quantified by plotting the total reward versus the number of interactions.

There are numerous RL algorithms that may be applicable within the context of spacecraft path planning. For the context of this work, we implement Proximal Policy Optimization (PPO) with clipped objective [87], which has garnered recent interest due to its robustness for astrodynamical path planning purposes. PPO with clipped objective is a policy gradient method in which the underlying neural network directly learns the policy

by updating its parameters appropriately [87]. Policy gradient methods are appropriate for continuous action space problems, and the minimum time transfer problem can be classified under this category.

PPO is comprised of an actor-critic architecture, with two different neural networks. During rollout of the policy, the agent gathers data about the encountered state, the action it has predicted, and subsequent reward; this information is then stored in a local, running memory. After a certain amount of steps, the data stored in the running memory is utilized to update the actor and critic architectures for a certain amount of epochs, subsequent to which the local memory is erased.

The actor neural network will dictate the action to take given a state, and the critic observes the actor's actions and provides feedback as to how good the decision was by calculating a value function. The agent will train by updating the neural network parameters via gradient descent according to the dictated loss function. To understand the unique loss function for a PPO with clipped objective agent, let us start with the following base equation:

$$L = E[log_{\pi_Q}(a_t|s_t)]A$$

where E is the expectation, $log_{\pi_Q}(a_t|s_t)$ is the log probability of taking a certain action a_t given a state s_t, and A is the advantage function, which simply tells us the advantage of selecting a certain action in a certain state. Given a policy Q, PPO first defines the probability ratio between the new and old policy as

$$r(Q) = \frac{\pi_Q(a_t|s_t)}{\pi_{Q_{old}}(a_t|s_t)}$$

If $r>1$, then the action is more probable in the current policy than the old policy.

Now, let's return to the advantage function, as it will drive the gradient updates in the neural network parameters. In essence, if the advantage is positive then we want to

Algorithm 2 Generalized PPO Algorithm Pseudocode with Clipped Objective

1. Define the neural network architectures for the actor and critic, along with all relevant algorithm hyperparamaters
2. Define the numerical optimization scheme
3. For each episode for a maximum number of episodes N
   (a) Execute the actor's learned policy and collect a dataset of trajectories.
   (b) Compute the reward received across each trajectory
   (c) Update the actor policy by maximizing the loss function defined by L^{PPO} and performing gradient backpropagation with the defined optimization scheme
   (d) Compute the mean square error between the critic's predicted state-action values, and the reward received. Use this error to update the critic weights via gradient backpropagation with the defined optimization scheme
4. Repeat Step 3 until the learning has converged

increase the probability of taking the action, but we want to contrain the policy shift to avoid overshooting in the direction of undesired behaviors. This is done by using clipped objective, which is used to define the clipped surrogate objective function as

$$L^{CLIP} = E[\min(r(Q)A), clip(r(Q), 1-\epsilon, 1+\epsilon)A]$$

where ϵ is a hyperparameter of the PPO algorithm, and min() indicates that L^{CLIP} selects the minimum of the two argument values. The inclusion of this clipped objective essentially ensures that the agent policy update is not too greedy and instead opting for smaller, and hence more stable, updates in the direction of good behaviors. The total loss function for a PPO agent is comprised of three terms

$$L^{PPO} = -E[L^{\text{CLIP}} - c_1 L^{\text{MSE}} + c_2 L^{\text{ENTROPY}}]$$

where L^{CLIP} is the clipped objective loss function defined above, L^{MSE} is the mean square error loss between the actor and critic predictions, and L^{ENTROPY} is the entropy bonus that measures the randomness of the agent's actions (good learning behavior means that the entropy decreases) [87]. The constants c_1 and c_2 are PPO hyperparameters and can be defined as desired. Furthermore, we note the inclusion of the negative sign in total loss

function. The neural network gradients are updated to minimize the loss, but we want to drive the learning in the direction dictated by the advantage function in L^{CLIP} (i.e. we wish to maximize) therefore the use of the negative sign is warranted. The summarized overview of PPO are given in Algorithm 2, with the explicit details being found in [87].

4.6.2 Environment Formulation and Reward Function Design

Though the terminology may be different, the environment that an RL agent are essentially represented by the underlying state dynamics. Therefore, each interaction, or step, is defined as follows:

1. At each epoch i, the agent receives an observation vector o_i that quantifies the current state

2. Given o_i, which is passed to the actor neural network as an input, the agent predicts the correct action a_t

3. The selected action a_t is then directly implemented in the state dynamical equations of motion, and the motion is propagated forward in time along the interval $[t_i, t_i + \Delta t]$, with the thrust control vector fixed in the direction predicted by the agent (i.e. we propagate the motion forward via a zero-order hold scheme). The value of Δt can be set as desired by the user, and for this implementation, $\Delta t = 30$ minutes unless the "endgame activation" phase is activated, which results in $\Delta t = 5$ minutes (further details on this phase are provided in the following paragraphs).

4. Given the propagated state s_{i+1} at $[t_i + \Delta t]$, the reward r_i is computed. Additionally, the propagated state is used to determine if a pre-defined termination criteria (such as a crash into a primary) has occurred

5. The reward r_i, information on whether a termination criteria was achieved, and the propagated state s_{i+1} is returned back to the agent, with the latter being used as the next observation vector.

For this investigation, there were four pre-defined termination criteria. Three of them are modeled as the state-path constraints previously given in Equations (4.16) - (4.18). The fourth termination criteria is derived from the implementation of what is referred to as an "endgame" zone. This zone is represented by a circle of some defined radial distance, with the origin centered on the target L5 position. If the agent's actions result in the trajectory entering this zone, then the agent enters the "endgame activation" phase, which results in a small bonus added to the reward function. However, should the agent select an action that results in the trajectory leaving this zone, then the episode is ended immediately with a heavy penalty in the reward function. It was observed that the inclusion of the "endgame" zone concept resulted in a rapid acceleration of the learning process, in comparison to cases where it was not included.

Given these termination criteria, the reward function for the PPO agent is defined as

$$r_i = -0.5t_i + \alpha_1 \text{norm}(e_{\vec{r}}) + \alpha_2 \text{norm}(e_{\vec{v}}) + \alpha_3 \tag{4.21}$$

where

$$\alpha_1 = \begin{cases} -10 \text{ if termination criteria detected} \\ +1 \text{ if endgame zone activated} \\ -5 \text{ at all other times} \end{cases} \tag{4.22}$$

$$\alpha_2 = \begin{cases} -5 \text{ if termination criteria detected} \\ +1 \text{ if endgame zone activated} \\ -2.5 \text{ if endgame zone activated and } e_{\vec{r}}) \leq 0.3 \text{ ndim} - 0.05 \text{ at all other times} \end{cases} \tag{4.23}$$

$$\alpha_3 = \begin{cases} -10,000 \text{ if termination criteria detected} \\ +100 \text{ if endgame zone activated} \\ +150 \text{ if endgame zone activated and norm}(e_{\vec{r}}) \leq 0.3 \text{ ndim} \\ +200 \text{ if endgame zone activated and norm}(e_{\vec{r}}) \leq 0.1 \text{ ndim} \\ +1000 \text{ if endgame zone activated and norm}(e_{\vec{r}}) \leq \text{1e-3 ndim} \\ 0 \text{ at all other times} \end{cases} \tag{4.24}$$

where norm($e_{\vec{r}}$) and norm($e_{\vec{v}}$) are the norm of the error between the current position and velocity vectors, with respect to the target L5 state. It was observed through numerous implementations that initially emphasizing the position error term, and then increasing the scalar weight value corresponding to the velocity error term only during the endgame activation phase resulted in the agent recognizing the desired path planning task at a faster rate. In other words, indicating to the agent that the first priority should be to get to the L5 state position, and then emphasize the braking maneuvers to achieve near zero velocity that is characteristic of the Lagrange points, was observed to result in a rapid learning process of the agent.

A final adjustment to the PPO algorithm was made, namely in the implementation of the continuous control aspect. As mentioned before, the standard approach to having any machine learning based model learn a continuous control task is to model the action space via Gaussian distribution, from which the action is sampled. To model any Gaussian distribution, the mean and standard deviation values are required, and it is possible to let the agent predict both of these parameters. However, it is a known observation that allowing the agent to also predict the standard deviation value may result in unstable learning, especially for complex dynamics. In such cases, an often preferred alternative strategy is to pre-define a standard deviation value, which is then linearly decayed to some minimum value as agent training progresses. Opting to implement this scheme also provides the user the ability to control the transition of the agent from an "exploration" phase (wherein the agent essentially

Table 4.5: The defined PPO hyperparameters and neural network architectures

Hyperparameter/Layer	**Actor**	**Critic**
Learning Rate	3e-4	5e-4
Criterion	PPO Loss	MSE
Optimizer	Adam	Adam
c_1	0.5	-
c_2	0.01	-
ϵ^{clip}	0.2	-
γ	0.99	-
K	10	10
Policy Update Interval	1,000 steps	1,000 steps
Hidden Layer 1	Fully Connected, 128 nodes	Fully Connected, 128 nodes
Activation	tanh	tanh
Hidden Layer 2	Fully Connected, 64 nodes	Fully Connected, 64 nodes
Activation	tanh	tanh
Output	1 node	1 node
Output Format	mean, used to sample α	scalar value of state-action pair

explores the environment) to the "exploitation" phase (in which the agent greedily selects the actions it has deemed to be the best/result in the highest reward). This transition between "exploration" and "exploitation" phases is a common characteristic of RL based methods.

Given the design of the reward function and the additional modifications given to the algorithm, the neural network architecture and the hyperparameters for the implemented PPO are provided in Table 4.5; the observation vector o_i, which essentially denotes the input to the actor neural network, is the same as that of the clone agent, and is defined to be $o_i = [x_i, y_i, \dot{x}, \dot{y}]$.

Chapter 5

Analysis of Results

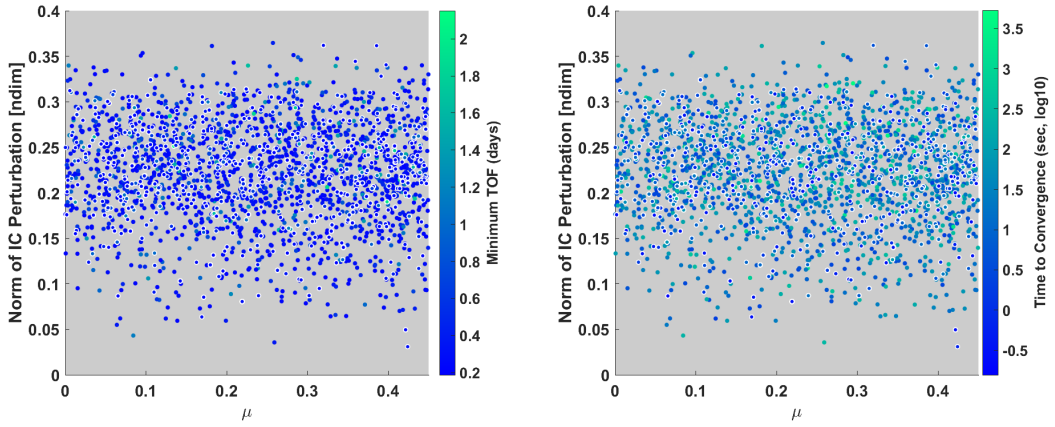
Each of the methods explained in the previous chapter were implemented as their formulations described. Their results are presented in each of the following subsections, with the results from the two tier controller presented first as they are the reference solutions that the ML approaches will be compared to.

5.1 Optimal Solution Database Generation

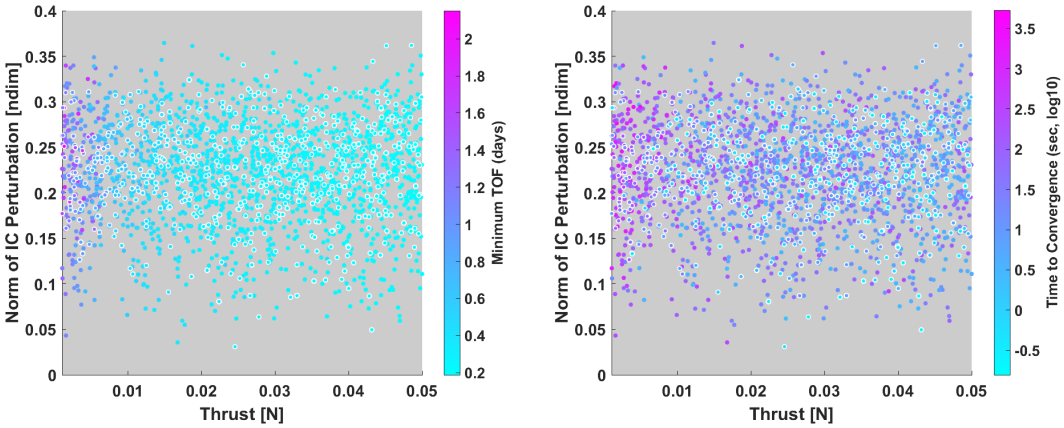
5.1.1 Two Tier Controller Solution Space

As mentioned before, the two tier controller was implemented with a stiff numerical integration routine to generate approximately 10,000 solutions, as well as with a 4th order Runge-Kutta routine to generate an additional approximately 1,500 solutions. These results are presented in Figures 5.1 and 5.2. In both figures, the quantity plotted along the y axis represents the norm of the perturbation of the initial boundary condition relative to the true L4 position of the randomly generated system, as quantified by the nondimensional μ parameters. Additionally, for both figures, the points outlined in white represent the solutions generated with the 4th order Runge-Kutta integration routine.

Figure 5.1 represents all the valid solutions that were generated by the indirect method formulation; in other words, no state path constraint violations were detected from the resulting trajectories. From the left subfigure in Figure 5.1a, it is observed that there is no discernible pattern regarding the time of flight as a function of the norm of the initial condition perturbation and the system μ value. Additionally, from the right subfigure in Figure 5.1a, it is also observed that there is no discernible pattern regarding the convergence



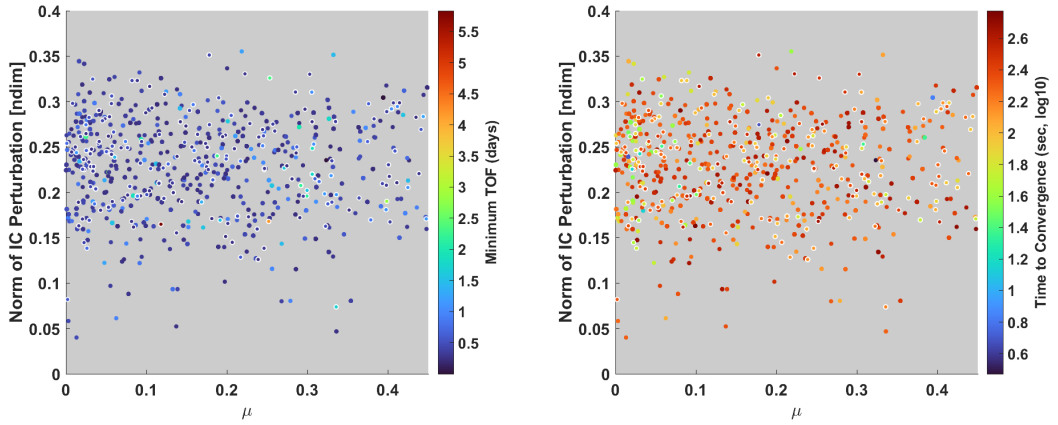
(a)



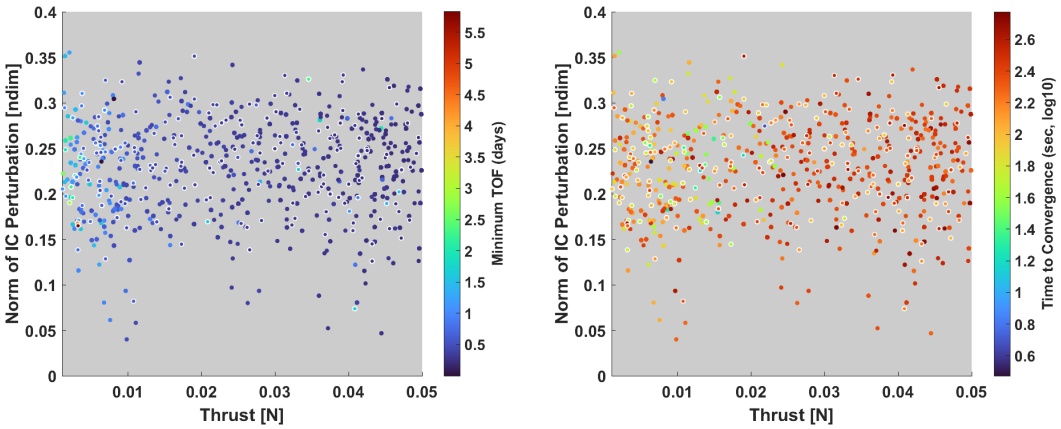
(b)

Figure 5.1: The solution space for trajectories that resulted in feasible solutions from the indirect method formulation. Scatter points outlined in white denote the solutions obtained from standard `ode45` integration, whereas all others denote solutions obtained from `ode15s`.

time of the solution as a function of the norm of the initial condition perturbation and the system μ value. However, when looking at both the time of flight and time of convergence as a function of the norm of the initial condition perturbation and the thrust value, a logical pattern emerges. Intuitively, low thrust values should result in higher time of flight, as they contribute to a lower total acceleration imparted by the propulsion system and do not become a dominating term within the dynamical equations of motion. As visualized in the left subfigure of Figure 5.1b. this observation is evidenced. Additionally, it is observed that low thrust scenarios also require higher times of convergence to a solution for the indirect



(a)



(b)

Figure 5.2: The solution space for trajectories that required path planning corrections from the direct collocation method. Scatter points outlined in white denote the solutions obtained from standard `ode45` integration, whereas all others denote solutions obtained from `ode15s`.

formulation.

In contrast to Figure 5.1, Figure 5.2 represents the invalid solutions that were generated from the indirect formulation, and required additional corrections via the direct collocation method and the defined state-path constraints. From Figure 5.2a, it can be observed that solutions within low μ dynamics required more corrections than those with high μ dynamics; from the right subfigure in Figure 5.2b it can be seen that solutions with high thrust resulted in higher times of convergence. Comparing the color gradients between this subfigure and the right subfigure in Figure 5.2a, it is observed that, overall, not only did scenarios with

high thrust and low μ values require direct method corrections more frequently than other cases, but these combinations also required longer times to converge to a solution. This observation follows dynamical intuition, as increasingly lower μ values result in the orbital motion resembling Keplerian dynamics rather than three-body dynamics. In such cases, having high thrust values results in the contributing acceleration of the propulsion system becoming the dominant component in the underlying equations of motion, allowing for a higher chance of compounding of errors during the propagation of the control actions. This consequently warrants a higher amount of corrections in order to achieve a valid solution.

5.1.2 Diverse Geometry of Solutions

In Chapter 4, it was noted that the use of 4th order Ringe-Kutta integration resulted in more diverse trajectory geometries as compared to the stiff numerical integration routines. From the approximately 1,500 solutions generated using the former integration scheme, at least eight different classes of trajectory geometries were visually identified, and are visualized in Figure 5.3; the Figure also provides the μ and thrust values that the solution corresponds to, along with the minimum time of flight that was achieved.

Though the diversity of solutions can be easily visualized, another interesting observation can be made in regards to the μ and thrust values. Consider trajectory classes 3 and 7, which had extremely similar μ values, but the thrust value for the solution belonging to class 7 was approximately half that of class 3. Given these parameters, the trajectory solution geometries were starkly different, through the minimum time of flight was only varying by a few hours. Additionally, it can be observed across all presented trajectory classes that extremely low thrust values result direct transfers that traverse around the smaller primary, whereas higher thrust values result in solutions that traverse around the main primary (either characterizing a direct transfer, or multi-revolution trajectories), as well as solutions that are able to navigate in the region between the primaries. Therefore, it may be possible that the thrust value of the propulsion system is a major contributing factor that determines

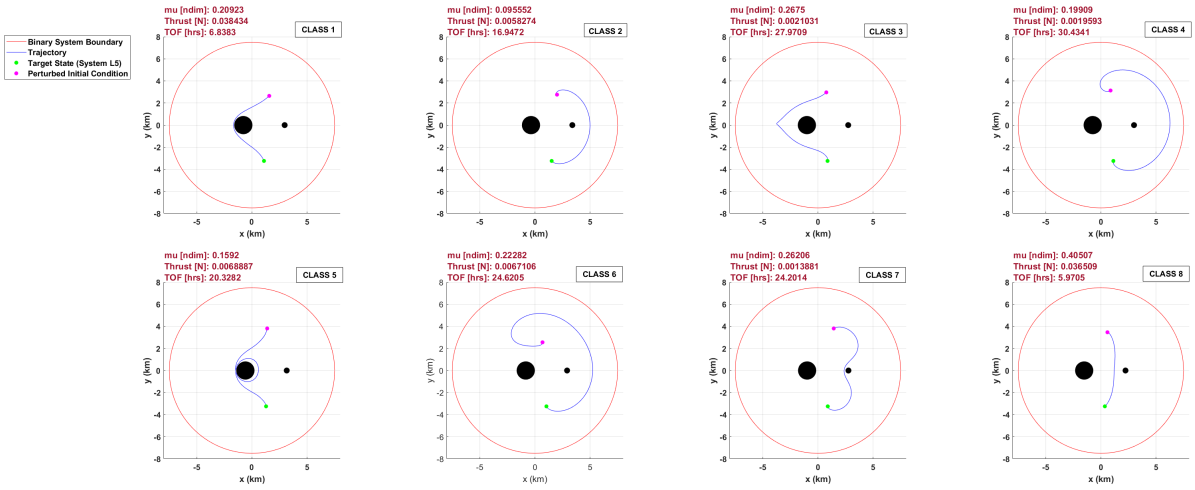


Figure 5.3: A visualization of some of the identified trajectory solution geometries within the dataset generated by the two tier controller

the trajectory class that is achieved; it is also possible that the converged values for the initial co-states determines the trajectory class. However, both of these hypotheses would warrant further investigations in order to achieve a definitive conclusion.

Though the existence of a diverse range of solution geometries within the generated dataset is an interesting observation, it may inadvertently impact the performance of any clone agents that use this information to train via supervised learning. More specifically, it was observed that across the entire dataset generated from the stiff numerical integration as the 4th order Runge-Kutta scheme, the trajectory geometry belonging to class 2 in Figure 5.3 was the dominant solution type, whereas the others occurred at a fewer number of instances. By their mathematical formulation, clone agents trained via supervised learning learn the average "behavior", or policy, that is present in the dataset; therefore if the trajectory geometry of class 2 is the dominant solution type, the agent would consider all other trajectory classes as outliers during the training process, thus leading to either minimal contribution, or no contribution at all, of these solutions to the agent's learning process. This subsequently would result in the agent only being able to provide path-planning strategies that resemble those of trajectory class 2. It may be possible to develop a training scheme that removes this bias in the dataset, and weighs all solutions equally in order to train a more generalized

Table 5.1: The initial state vector, and converged initial costate values for the visualized trajectory classes 1-4 in Figure 5.3. The vector $\vec{\lambda}_i = [\lambda_x, \lambda_y, \lambda_{\dot{x}}, \lambda_{\dot{y}}, \lambda_m]^T$, and $\vec{x} = [x, y, \dot{x}, \dot{y}, m_{init}]^T$ (all units are nondimensional except m_{init}, which is in kg)

Trajectory Class	$\vec{\lambda}_i(t_0)$	$\vec{x}(t_0)$	Correction Required by Direct Collocation Method?
1	$\begin{bmatrix} 0.3273 \\ 0.4765 \\ 0.0221 \\ 0.2421 \\ 9.0298 \times 10^{-4} \end{bmatrix}$	$\begin{bmatrix} 0.4181 \\ 0.7061 \\ -0.1288 \\ -0.0561 \\ 610 \end{bmatrix}$	Yes
2	$\begin{bmatrix} -1.8919 \\ -0.8813 \\ -0.8300 \\ -1.3381 \\ 0.0229 \end{bmatrix}$	$\begin{bmatrix} 0.5339 \\ 0.7360 \\ -0.1346 \\ 0.0664 \\ 610 \end{bmatrix}$	No
3	$\begin{bmatrix} 1.9637 \\ 6.5599 \\ -1.6941 \\ 2.5391 \\ 0.0060 \end{bmatrix}$	$\begin{bmatrix} 0.2026 \\ 0.7911 \\ -0.1354 \\ -0.1285 \\ 610 \end{bmatrix}$	No
4	$\begin{bmatrix} -3.4588 \\ -9.0187 \\ 3.5268 \\ -4.1938 \\ 0.0037 \end{bmatrix}$	$\begin{bmatrix} 0.2332 \\ 0.8357 \\ -0.9109 \\ -0.1212 \\ 610 \end{bmatrix}$	No

clone agent. However, this was not implemented within this investigation as this additional modification may skew the analysis in favor of the machine learning methods. Furthermore, the appropriate bias normalization method for the dataset may not be straightforward, hence requiring additional investigation.

5.2 Optimal Control Driven Imitation Learning

As mentioned before, the generated dataset from the two tier optimal controller was used to train two types of clone agents, which are represented by the standard feed-forward neural network, and the stateless LSTM neural network. The training of each of these agents was implemented with the network architectures and the hyperparameters previously presented in Tables 4.3 and 4.4. Subsequently, the performance of the trained agents was

Table 5.2: The initial state vector, and converged initial costate values for the visualized trajectory classes 5-8 in Figure 5.3. The vector $\vec{\lambda_i} = [\lambda_x, \lambda_y, \lambda_{\dot{x}}, \lambda_{\dot{y}}, \lambda_m]^T$, and $\vec{x} = [x, y, \dot{x}, \dot{y}, m_{init}]^T$ (all units are nondimensional except m_{init}, which is in kg)

Trajectory Class	$\vec{\lambda_i}(t_0)$	$\vec{x}(t_0)$	**Correction Required by Direct Collocation Method?**
5	$\begin{bmatrix} 1.5393 \\ 3.3254 \\ -0.8366 \\ 1.4926 \\ 0.0050 \end{bmatrix}$	$\begin{bmatrix} 0.3684 \\ 1.0160 \\ -0.0606 \\ -0.1832 \\ 610 \end{bmatrix}$	Yes
6	$\begin{bmatrix} -4.2860 \\ -9.4105 \\ 3.0786 \\ -3.9749 \\ 0.0067 \end{bmatrix}$	$\begin{bmatrix} 0.1826 \\ 0.6816 \\ 0.0868 \\ -0.1682 \\ 610 \end{bmatrix}$	No
7	$\begin{bmatrix} 10.9906 \\ 18.8830 \\ -6.9888 \\ 14.3663 \\ 0.0058 \end{bmatrix}$	$\begin{bmatrix} 0.3825 \\ 1.0153 \\ 0.1973 \\ 0.1359 \\ 610 \end{bmatrix}$	No
8	$\begin{bmatrix} -0.0609 \\ 0.5523 \\ -0.1972 \\ 0.2090 \\ 0.0011 \end{bmatrix}$	$\begin{bmatrix} 0.1579 \\ 0.9239 \\ -0.0067 \\ 0.1562 \\ 610 \end{bmatrix}$	No

analysed by the comparison of the agent's predicted path planning strategy with respect to a separate dataset of approximately 1,000 solutions, also generated using the two tier optimal controller, that were not utilized during the training process. In other words, the agent has no prior knowledge of the solutions contained within this validation dataset as it has not encountered the specific sequence of states during the training process.

5.2.1 Quantifying Performance via Target State Insertion Errors

The performance of each trained clone was analyzed as follows. For each of the solutions within this validation dataset, the total time of flight $[t_0, t_f]$ of the optimal solution is discretized into 5,000 equitemporal time steps of some increment δt. The numerical integration routine, that propagates the dynamical equations of motion forward in time, is initialized

with the initial state of the optimal solution. At each time step i, the input feature vector for the neural network (i.e. $\vec{x}_{input} = [x_i, y_i, \dot{x}, \dot{y}]$) is passed to the clone agent to obtain the control decision, which is then propagated forward on the interval $[t_i, t_i + \delta t]$, using the state dynamical equations of motion, until the final time t_f is achieved. The agent's path planning solution is then compared to the optimal solution by comparing the magnitude of the state error $e(t_f)$ at the final time t_f; in other words, the value

$$e(t_f) = \text{norm}\left(\vec{x}(t_f)_{agent} - \vec{x}(t_f)_{optimal}\right) \tag{5.1}$$

is explicitly computed. This simple metric may provide an insight into the performance and accuracy of the trained agents as it directly computes the error bound at which the final boundary constraint of the problem (i.e. the target L5 state) was achieved.

We begin by analyzing the results of the feed forward neural network based clone agents. Recall that there were two versions of the feed-forward based clone agents: version 1 directly learned to predict α, the direction of the thrust control vector, whereas version 2 learned to predict a mean value, that was used to construct the continuous action space modeled by a Normal distribution of the predicted mean and a fixed standard deviation value of $\sigma = 0.1$, from which the thrust control direction α was sampled from. After the agents were trained, their performance was evaluated using the validation dataset and the computation of the performance metric defined in Eq. (5.1)

The performance results of version 1 of the feed-forward based neural network clone are presented in Figure 5.4, which visualizes the norm of the state error at the final time with respect to the variation in μ and thrust values. From the left subfigure in Figure 5.4, there is no discernible trend regarding the performance of the agent as a function of the norm of the initial condition perturbation and the μ parameter space. However, as evidenced in the right subfigure in Figure 5.4, a performance trend emerges as a function of the norm of the initial condition perturbation and the thrust parameter space. It can be observed

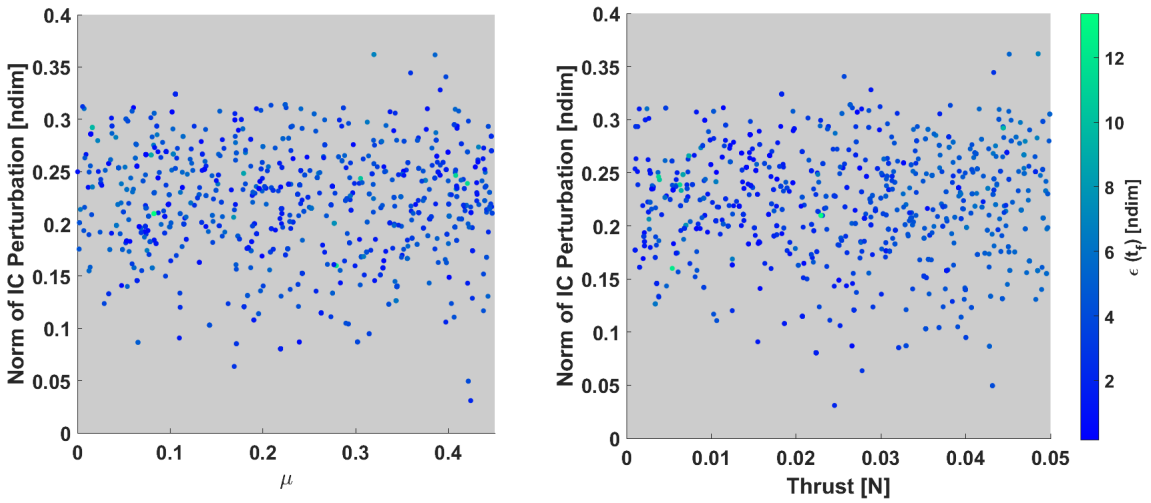


Figure 5.4: The performance results of version 1 of the feed-forward based optimal solution clone agent, as quantified by Eq. (5.1).

that scenarios comprising of high thrust values result in consistently inaccurate performance of the clone agents, whereas low thrust values generally result in the agent being able to navigate to the target L5 state with a higher accuracy. This observation may be driven by the dynamical contribution of the thrust value itself. More specifically, higher thrust values result may result in a higher dynamical contribution of the acceleration provided by the propulsion system; in instances where this dynamical term becomes dominant within the underlying equations of motion, an inaccurate prediction of the thrust control direction α by the clone agent would result in a compounding of errors from which recovery may be impossible. Conversely, low thrust values may result in a non-dominant contribution of the propulsion system, which may not only reduce the possibility of compounding of errors due to inaccurate control predictions, but also allow for the clone agent to attempt a recovery of the path-planning solution.

Along with the overall performance results, the individual path planning strategies generated by the agent can also be visualized. For example, consider the most accurate solution generated by version 1 of the feed forward clone agent; this solution is visualized in Figure 5.5. Within the same time interval of the minimum time of flight as computed by

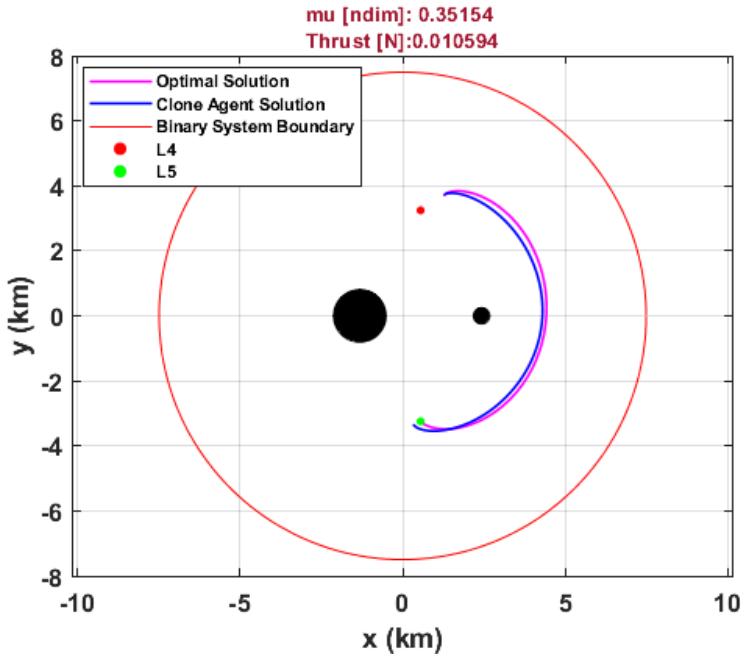


Figure 5.5: An example visualization of the clone agent path planning solution in comparison to the optimal solution

the optimal solution, the clone agent was able to achieve an extremely similar path-planning strategy with position and velocity insertion errors of 0.23 km and 0.04 m/s. The clone agent required approximately 6 seconds in order to generate a solution, whereas the optimal control solution only required 0.29 seconds to achieve a valid solution from indirect method formulation that respected both the boundary and state-path constraints. This analysis can be repeated for all the solutions generated by the clone agents.

The results from version 2 of the feed forward clone agent are presented in Figure 5.6; recall that whereas version 1 of the clone agent learned to directly predict the thrust control direction α, version 2 learned to predict a mean value that is then used to construct a Gaussian distribution from which α is sampled from. On direct comparison of Figures 5.4 and 5.6, it may be observed that version 2 of the agent performed at a lower accuracy than version 1. However, in both versions of the feed forward clone agent, the performance varying as a function of the norm of the initial condition perturbation and the thrust value remains similar, with high thrust values resulting in consistently inaccurate agent performance, and

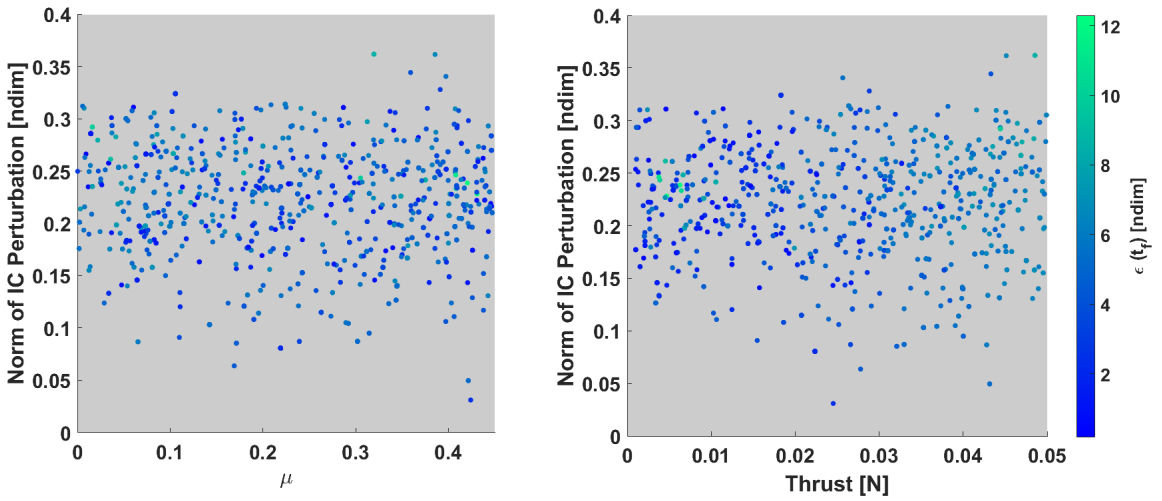


Figure 5.6: The performance results of version 2 of the feed-forward based optimal solution clone agent, as quantified by Eq. (5.1).

low thrust values resulting in generally higher accuracy performance.

Version 2 resulting in worse performing agents may be a logical observation from a machine learning perspective, as it is a commonly recognized feature of the manner that the action space is inherently modeled with. The implementation of any probability density function based schemes, from which the action is sampled, have often been observed to require considerably stronger neural network architectures and additional training modifications for the clone agents as compared to directly learning the prediction of the desired output; furthermore, these modifications may vary as a function of the dataset itself. However, as mentioned in Chapter 4, no additional modifications were imparted on any of the methods implemented in this investigation, which may inherently affect the performance of this version of the feed forward clone agent.

In the same manner as the feed-forward network based clone agents, there were also two versions of the LSTM network clone agents: version 1 directly learned to predict α, the direction of the thrust control vector, whereas version 2 learned to predict a mean value, that was used to construct the continuous action space modeled by a Normal distribution of the predicted mean and a fixed standard deviation value of $\sigma = 0.1$, from which the

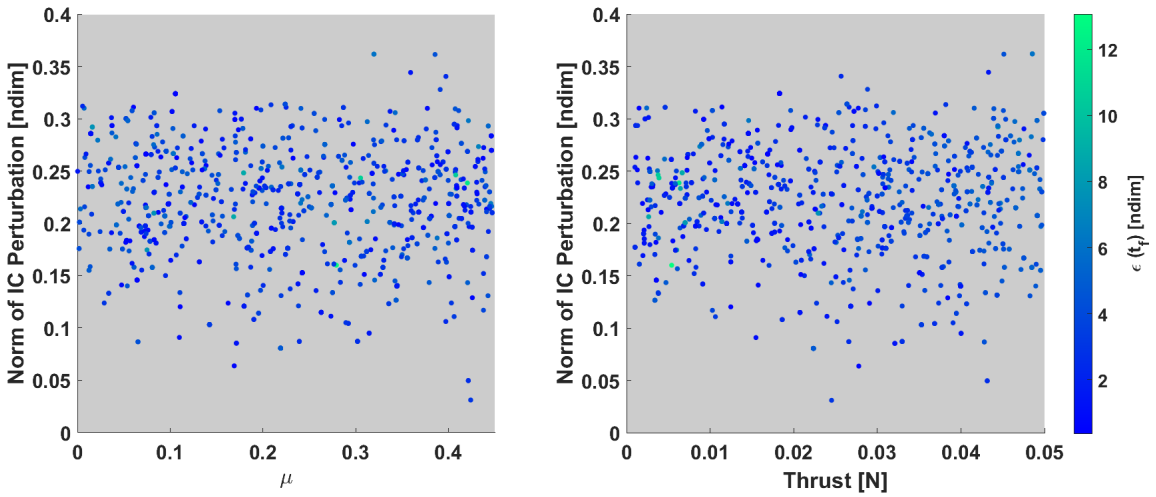


Figure 5.7: The performance results of version 1 of the LSTM based optimal solution clone agent, as quantified by Eq. (5.1).

thrust control direction α was sampled from. Once again, after the agents were trained, their performance was evaluated using the validation dataset and the computation of the performance metric defined in Eq. (5.1).

The performance results from version 1 of the LSTM clone agent are visualized in Figure 5.7. In direct comparison to the performance of version 1 of the feed-forward network clone agent (Figure 5.4), it can be observed that the utilization of a more rigourous neural network architecture yields better performance. More specifically, we can note that the LSTM based clone agent is capable of achieving more accurate results for some high thrust values, whereas the feed forward based clone was unable to do so for such cases. This observation may be directly related to the fact that LSTM networks are inherently able to recognize long-term temporal dependencies, which may prove critical to the path-planning, as this is essentially a sequence prediction task.

The performance results from version 2 of the LSTM network clone agent are visualized in Figure 5.8, and it an be observed that this formulation of the agent resulted in significantly worse performance as compared to version 1. Again, this may be attributed to the fact that version 2 of the clone agent predicts a mean value, which is then used to construct a Gaussian

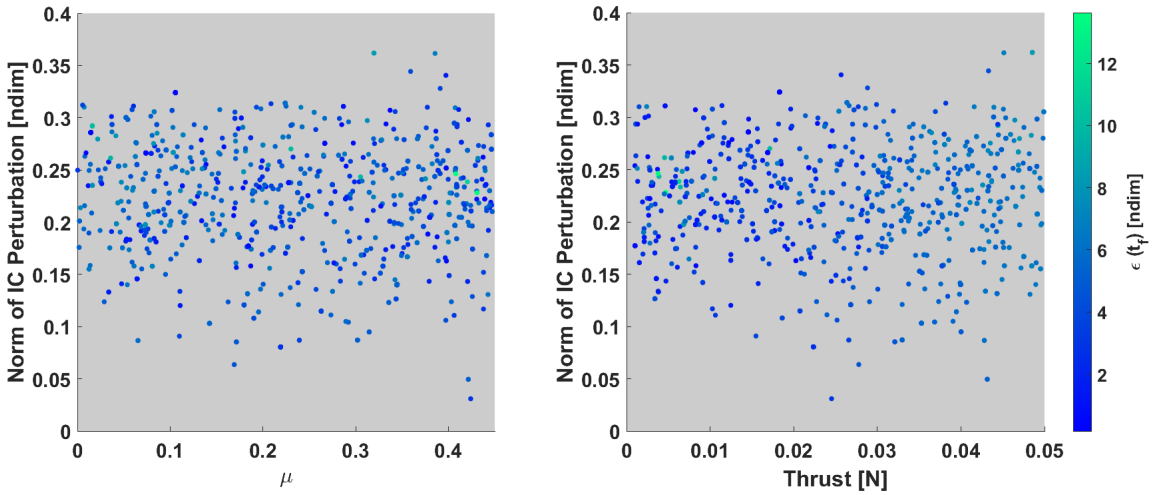


Figure 5.8: The performance results of version 2 of the LSTM based optimal solution clone agent, as quantified by Eq. (5.1).

distribution from which the action is sampled from. As mentioned before, opting to select the action in this manner may result in a more inaccurate, or unstable agent performance, as compared to directly learning to predict the appropriate action without the use of probability density functions. Though this version of the agent did not achieve as high accuracies as version 1, it is interestingly observed from initial comparisons between Figures 5.4 and 5.8 that a feed-forward network that learns to directly predict the thrust control direction α, and an LSTM network that selects α from a predicted Normal distribution may offer similar performance. However, further investigations are warranted in order to verify this initial observation.

5.2.2 Quantifying Performance via Optimal Flight Envelope Analysis

The previous section utilized a common performance metric of quantifying the final state insertion error in order to gain insights into the performance of a machine learning agent. However, tasking an autonomous agent to solve a *sequential* path-planning problem with explicitly defined boundary conditions is subject to the compounding of errors, as there will be a compounding of errors should the agent select an incorrect action for a given state.

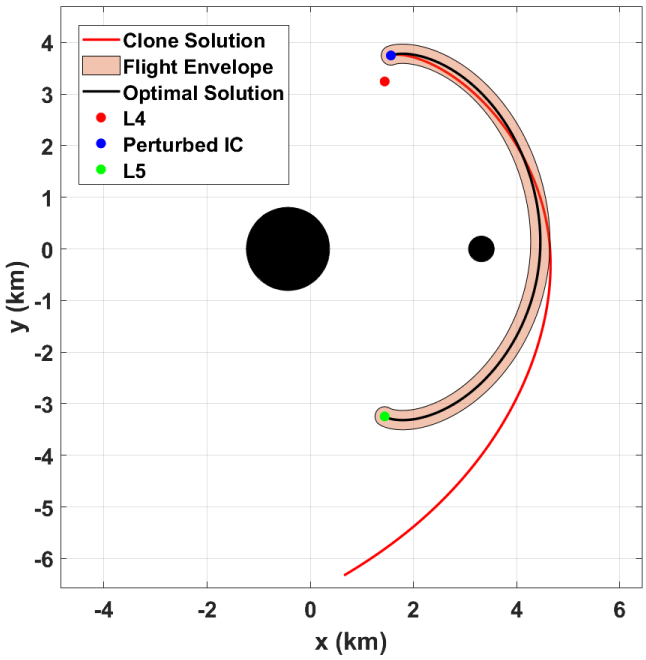


Figure 5.9: An example visualization of the flight envelope based analysis as a measure of performance of a clone agent ($\mu = 0.1158$, $T_{\max} = 0.0113$ Newtons). The clone solution demonstrated a flight envelope match of 49.58%.

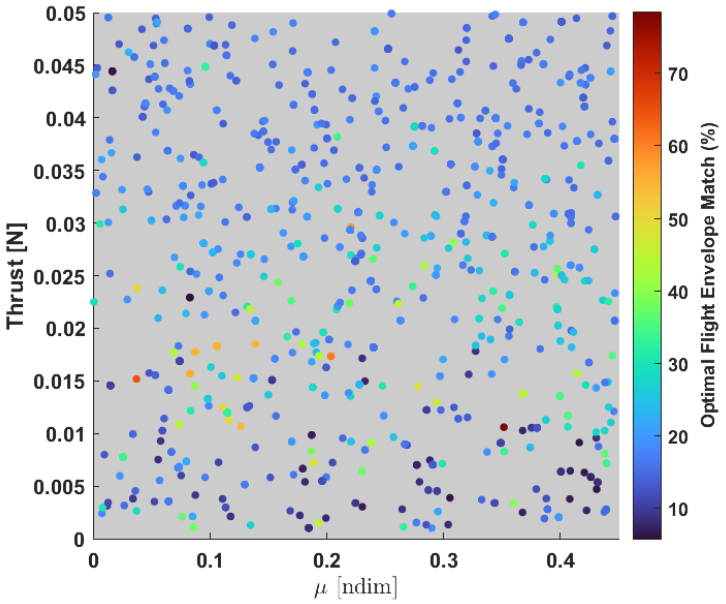
Therefore, it is possible that an agent initially provided a similar path-planning solution as that computed by an optimal control approach, but eventually diverged from the latter due to compounding of errors; this performance characteristic may not be fully captured by the quantification provided in Eq. (5.1).

Alternatively, insight into the duration at which the autonomous agent's path-planning solution resembled the optimal before diverging may be yielded by the implementation of a "flight envelope", which is defined as a spatial region with some pre-defined radius that surrounds the optimal trajectory solution. In this manner, and by overlaying the optimal solution, corresponding flight envelope, and the agent's path-planning solution, the total duration of the latter's flight time that was spent within the flight envelope can be computed. This computation is referred to as the "flight envelope match percentage" in the resulting analyses, and can provide more direct insight into how well the trained autonomous agent is

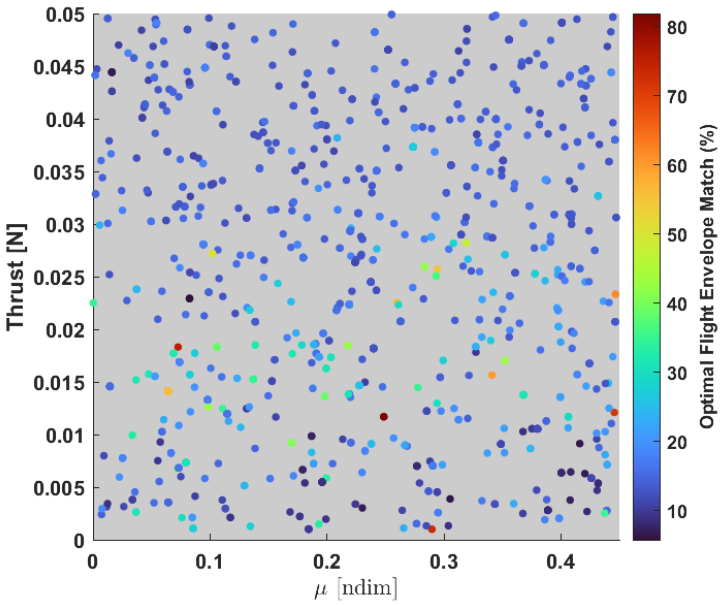
able to track an optimal solution. For the purposes of this analysis, the flight envelope was defined to be a radial distance of 5% of the binary asteroid system characteristic length value, which is quantified in Table 4.2. An example visualization of the optimal control solution with the corresponding flight envelope and the agent's path-planning solution is visualized in Figure 5.9, which demonstrates the optimal solution for a minimum-time transfer from a perturbed L4 initial condition to the system L5 state, given the parameters of $\mu = 0.1158$ and T = 0.0113 Newtons. Also visualized is the flight envelope surrounding the optimal solution, as well as the path-planning solution generated by a trained clone agent. It can be seen that the clone's solution is able to track the optimal solution for approximately half of the flight time, before diverging due to a compounding of errors in the predicted control profile.

Figure 5.10 visualizes the performance of both optimal solution based clone agents that were trained using the feed forward NN; recall that version 1 (Figure 5.10a) trains the agent to learn a deterministic policy, whereas version 2 (Figure 5.10b) trains the agent to learn a stochastic policy. It can be observed that version 2 of the clone provided more accurate performance as compared to its deterministic policy counterpart. This result may seem counter-intuitive, and may be attributed to the fact that the implementation of a stochastic policy may have allowed the clone to "get lucky" and accidentally achieve a more accurate solution.

Figure 5.11 visualizes the performance of both optimal solution based clone agents that were trained using the stateless LSTM NN; again, recall that version 1 (Figure 5.11a) trains the agent to learn a deterministic policy, whereas version 2 (Figure 5.11b) trains the agent to learn a stochastic policy. From these figures, it can be observed that version 1 of the stateless LSTM network based clone provided more accurate performance as compared to its stochastic policy counterpart. Furthermore, it can be seen that both versions of the LSTM based clones demonstrate similar or better flight envelope match percentages in comparison

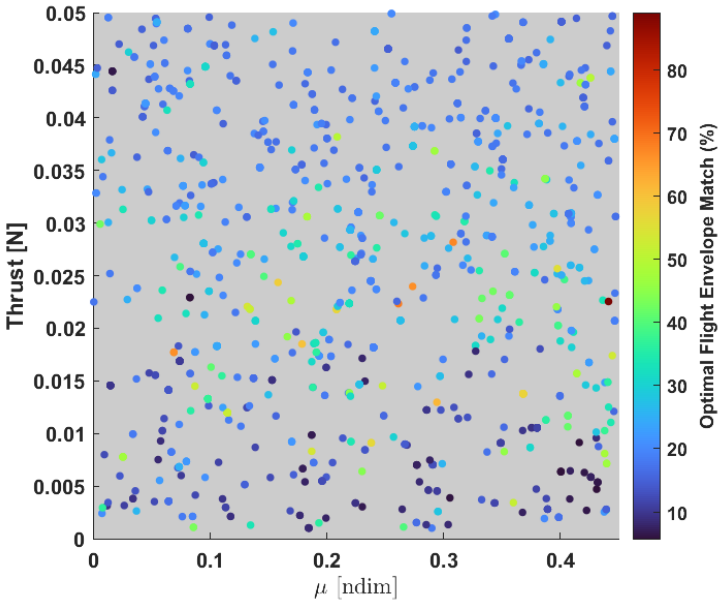


(a)

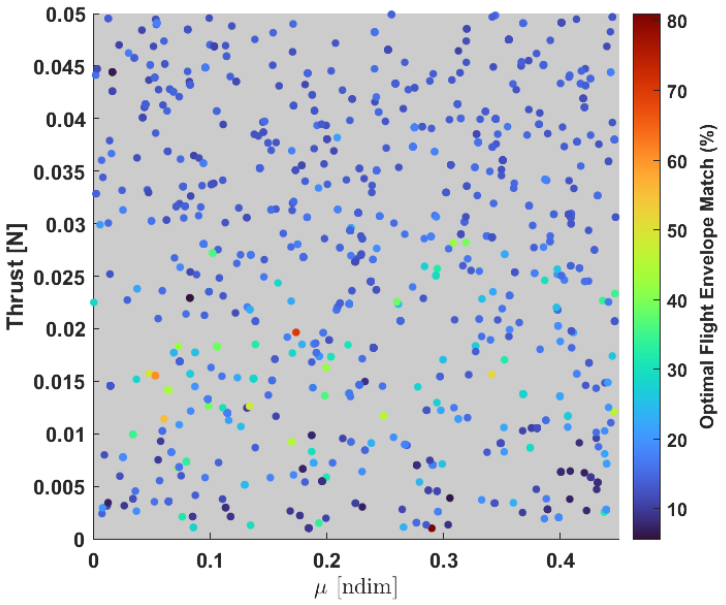


(b)

Figure 5.10: The flight envelope match percentages for both feed forward neural network based clones, trained from the optimal control databse generated by the two tier controller. Figure 5.10a represents the performance of version 1 of the clone (deterministic policy), and Figure 5.10b represents the performance of version 2 of the clone (stochastic policy).



(a)



(b)

Figure 5.11: The flight envelope match percentages for both stateless LSTM neural network based clones, trained from the optimal control database generated by the two tier controller. Figure 5.11a represents the performance of version 1 of the clone (deterministic policy), and Figure 5.11b represents the performance of version 2 of the clone (stochastic policy).

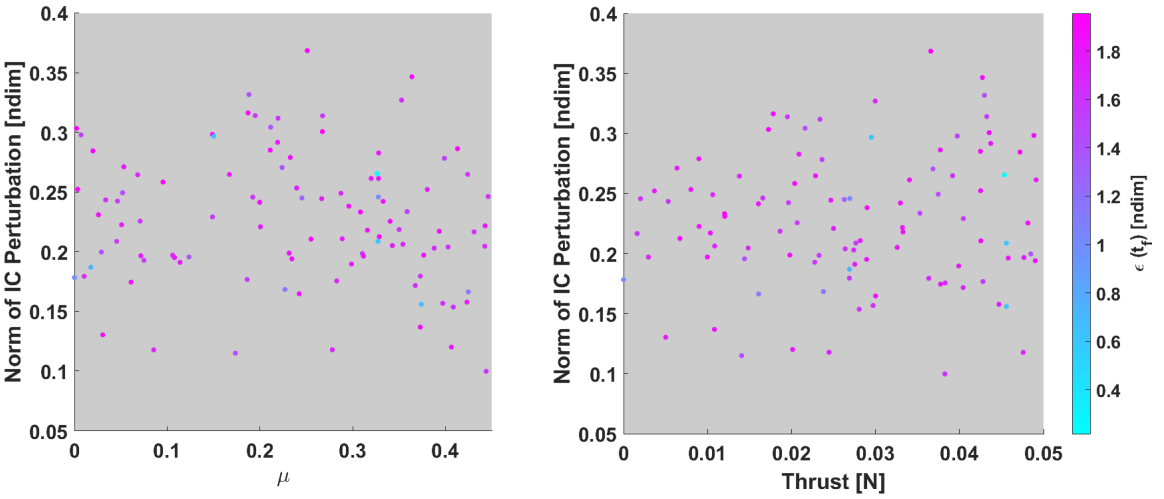


Figure 5.12: The performance results of the human pilot, as quantified by the definition of the final state error relative to the target state.

to the feed forward based clones, intuitively indicating that this NN architecture may be better suited for the sequential path-planning problem.

5.3 Human Driven Path Planning

5.3.1 Quantifying Performance via Target State Insertion Errors

Prior to training a clone agent from human demonstrations, a dataset of such solutions must first be generated. This was accomplished via the use of the interactive flight simulator. Recall that approximately 25 hours of data were collected from a single human pilot, which corresponded to approximately 100 independent demonstrations of path-planning solutions. Given this initial dataset, the performance of a human pilot can also be quantified by computing the final state error relative to the target state, as empiricized by Eq. (5.1). These results are plotted in Figure 5.12, and it can be immediately observed that for all generated solutions, the human pilot provided better performance that was approximately an order of magnitude more accurate than the clone agents directly trained on optimal solutions; this may be attributed to the interactive manner through which the human pilot was able to generate the solutions. By allowing the human to have full control authority in navigating the

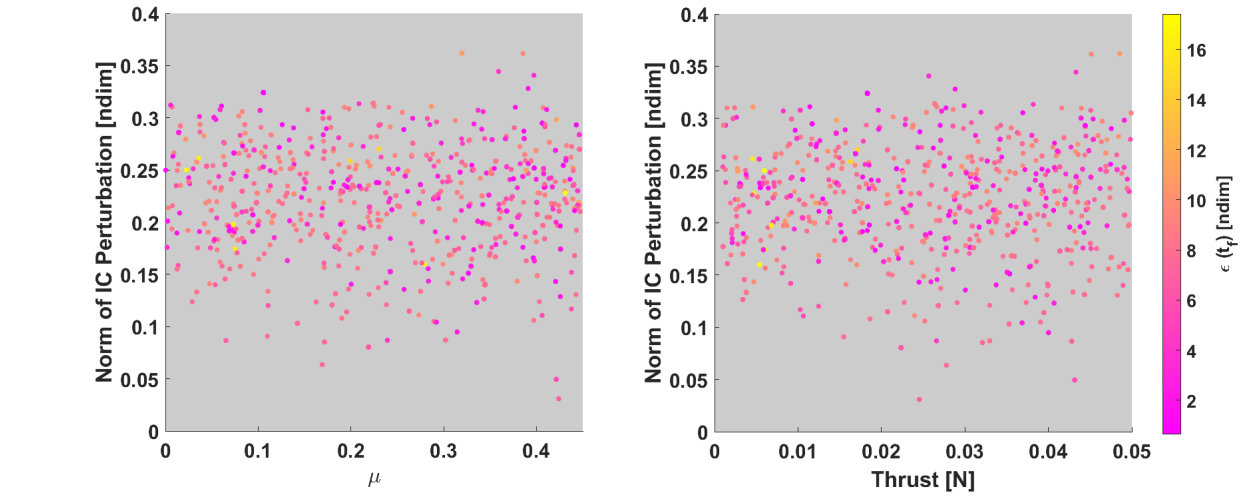


Figure 5.13: The performance results of an LSTM network clone agent that was trained on a human generated dataset of path planning solutions

spacecraft to accomplish the designated path-planning task, and presenting this information in a visual and intuitive manner, the human may be able to mitigate the compounding of errors due to an incorrect control action by subsequently implementing a corrective maneuver. This ability, along with the capability to adapt the path-planning strategy in real-time, as a response to environment interactions, can greatly influence the accuracy of the generated path-planning solution. However, it should be noted that the human pilot was unable to provide perfect demonstrations, as there were errors in the achieved final state relative to the target final state. Therefore, this dataset may not be representative of perfect demonstrations, and the quality will directly impact the performance of any clone agent trained via supervised learning.

Following the data generation process utilizing human logic, as previously explained in Section 4.5, a clone agent was trained from these path-planning solutions. However, only the LSTM network driven clone is considered and implemented, with the same architecture and training parameters as that of version 1 of the LSTM clone agent trained from the optimal control solutions (these values were previously provided in Table 4.4. In other words, only one LSTM network clone agent was trained from the human generated dataset, which

learned to directly predict the direction of the thrust control vector α, instead of sampling α from a predicted Gaussian distribution.

The performance results of the human data based clone are visualized in Figure 5.13, and it can be immediately observed that there is no discernable pattern in regards to the performance of the clone as a function of any of the dynamical variations. This may be attributed to multiple factors, two of which are briefly discussed below:

1. **Quantity and quality of solutions generated by the human demonstrator**: As mentioned before in Section 4.5, a single human pilot generated approximately 100 path planning solutions within the pre-defined data collection time frame of 25 hours. However, this amount of time may not be sufficient in order to generate high quality solutions, as the human pilot may still be learning how to navigate the spacecraft, within perturbed dynamics, for minimum time path planning task. This consequently results in a lower quality dataset, which results in a low performing clone agent

2. **Domain transfer between human and agent**: The utilization of human intuition for a complex task may warrant the use of domain adaptation techniques, in which we ensure that the agent takes into consideration the same decision variables as the human in order to make a path planning prediction. Determining the true causal variables of the human decision process may be achieved by feature extraction and/or feature selection techniques, in order to construct the appropriate input vector for the clone agent neural network.

5.3.1.1 Quantifying Performance via Optimal Flight Envelope Analysis

Similar to before, the flight envelope based analysis can be repeated for both the human pilot, as well as the corresponding clone agent in order to provide further analysis in how well these agents are able to relatively adhere to an optimal path-planning solution. The flight envelope match percentage for both the human pilot, as well as the human clone, is visualized in Figures 5.14 and 5.15, respectively. In comparison to the clone agents trained

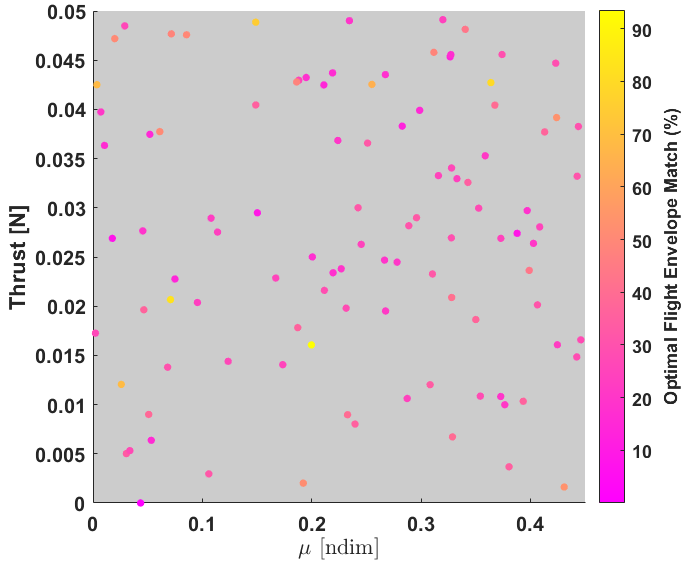


Figure 5.14: The flight envelope match percentage for all solutions generated by the human pilot.

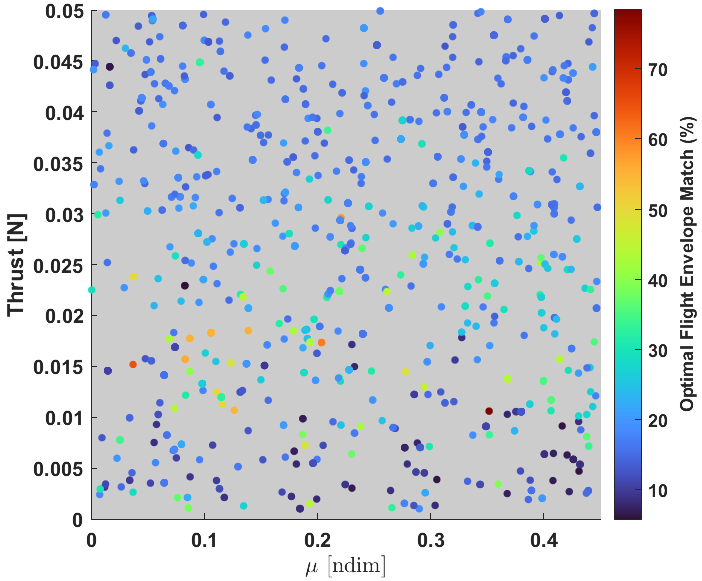


Figure 5.15: The flight envelope match percentage for the stateless LSTM NN based clone agent trained on all solutions generated by the human pilot.

from optimal solutions, which demonstrated a flight envelope match percentage of upwards of 80% (Figure 5.10 and 5.11), the human pilot was able to demonstrate flight envelope match percentages upwards of 90%. It should be noted that the optimal clone agents had

direct knowledge of the optimal solutions via the respective training database, whereas the human pilot had no knowledge of the optimal behavior and was instead generating path-planning solutions solely via environment interactions. This may be indicative of human driven solutions being a reasonable source of domain knowledge, though it should be noted that the inclusion of humans within any frameworks need to consider the risk of mistakes, biases, and inaccuracies. As there were no modifications made within the machine learning framework to address these concerns during training, the performance of the human clone agent is seemingly the worst of all autonomous agents. This effect can be directly visualized in Figure 5.15, which demonstrates that this clone provides performance that is well below 60% of the flight envelope match metric.

5.4 Summarized Performance Characteristics of all Path-Planning Agents

The previous sections separately discussed the optimal control clones, human pilot, and the human based clone path planning solutions. To allow for a more concise comparison of performance, the standard statistical metrics of the flight envelope match percentages of all agents are presented in Table 5.3. As previously mentioned, rather than considering the error of the final state relative to the target state, the flight envelope based analysis may offer a better insight into the agent performance as it is able to capture the duration to which the generated path-planning solution remains in the vicinity of the optimal before diverging due to the compounding of errors.

From the statistics presented in Table 5.3, it can be seen that the human pilot provided the most accurate path-planning solutions relative to the corresponding optimal strategy. Though the performance may greatly vary as a function of the dynamics that were encountered, the human pilot was able to provide, on average, approximately 8.6% more accurate path-planning solutions in comparison to the most accurate machine learning based agent, which was the stateless LSTM clone, trained from the optimal solution database, which learned to predict a deterministic control strategy. This observation may be indicative of

Table 5.3: Summarized performance statistics of both the human pilot and all trained clone agents.

	Flight Envelope Match Statistics (%)		
Agent Type	**Min**	**Max**	**Average**
Human Pilot	7.90	93.55	31.16
Human Pilot Clone	4.96	52.24	12.76
Optimal Control Clones			
Feed Forward, Deterministic Policy	5.86	78.5	20.06
Feed Forward, Stochastic Policy	5.72	81.84	17.17
Stateless LSTM, Deterministic Policy	5.76	89.06	22.54
Stateless LSTM, Stochastic Policy	5.7	81.02	16.52

the existence of a less computationally expensive domain source which can be leveraged to subsequently train autonomous agents. Rather than generating extremely large training databases of path-planning solutions rigorously computed from optimal control methods, reformulating and posing the problem in an intuitive, interactive, and less rigorous manner may allow for human demonstrators to generate demonstration datasets which can then be used to train autonomous agents. However, it should also be noted that the use of humans as a domain source will require the inclusion of appropriate modifications within the corresponding machine learning framework that would be able to account for mistakes and biases. The extent and nature of these modifications were not addressed within this exploratory work, and can instead serve as an extension of future works.

5.5 Proximal Policy Optimization (PPO)

The final method that was implemented was the PPO algorithm, which was previously explained in Section 4.6. Prior to implementing PPO on the full perturbation range that was considered within this investigation, the algorithm was first tested on a deterministic scenario, with no perturbations on the initial L4 state, and the fixed values of $\mu = 0.022848$ and a thrust value of T = 5 mN (the values of μ and T were randomly chosen from the ranges defined in Table 4.1).

The PPO agent predicts a mean value that is used to construct a Gaussian distribution,

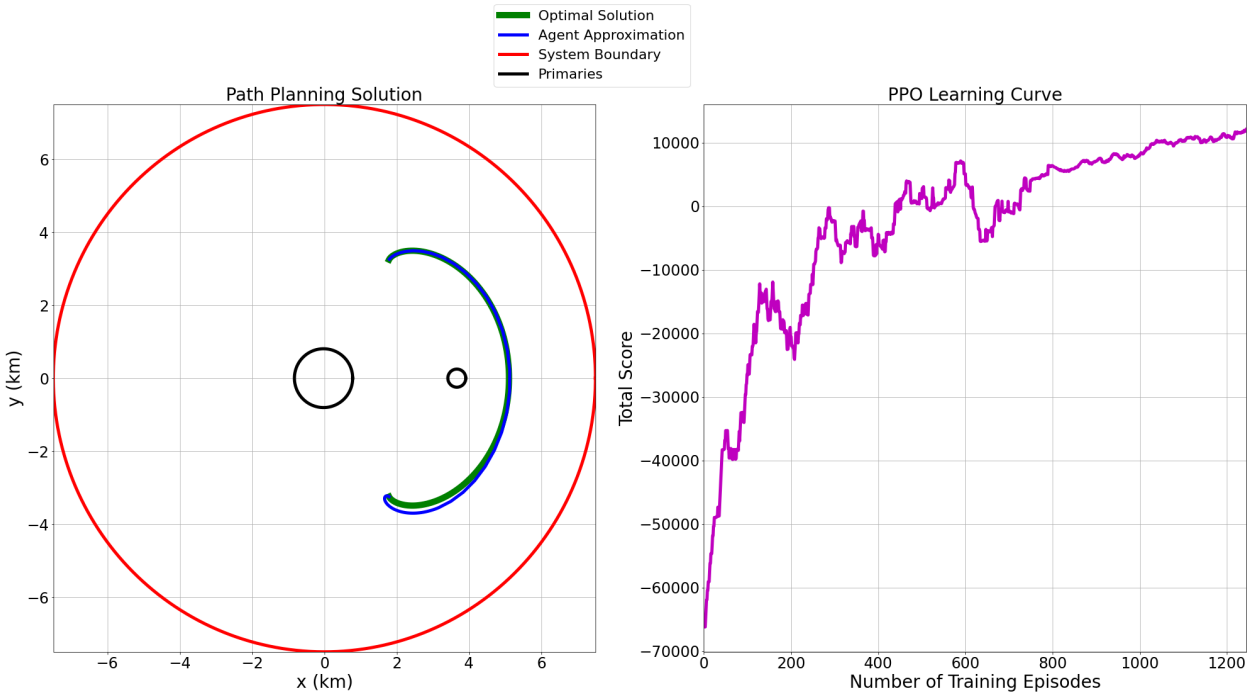


Figure 5.16: The path-planning solution for a deterministic L4-L5 minimum time transfer achieved by the PPO agent. The left subfigure shows the predicted transfer from L4-L5 as predicted by the agent, and the right subfigure shows the moving average of the learning curve (window size = 50 episodes).

along with a standard deviation value σ; the thrust control direction α is then sampled from this distribution. The value of σ is initialized to be $\sigma = 0.15$, and is linearly decayed by an increment of 0.01 every 60,000 steps, until the minimum value $\sigma_{min} = 0.1$ is achieved (it should be noted that every 60,000 steps corresponded to approximately 100 independent episodes). At each step, the motion driven by the predicted direction of the thrust control vector is propagated forward in time for 30 minutes via a zero-order hold. However, if the "endgame" phase is determined to be activated, then the time interval for forward motion propagation is reduced to 5 minutes (recall that the concept of "endgame" was previously discussed in Section 4.6). For this deterministic scenario, the endgame zone radius is defined as 2 km. All other learning hyperparameters implemented are the same as those included in Table 4.5.

Figure 5.16 visualizes the converged path planning solution achieved from training the PPO algorithm via the neural network architecture and learning hyperparameters previously

discussed above; the left subfigure shows the transfer from L4-L5 as generated by the PPO agent, whereas the right subfigure shows the moving average (window size = 50 episodes) of the learning curve as a function of the total reward achieved in each episode. The latter is a common metric that is monitored during the training process of any RL based agent, as it provides direct insight into the learning progression. For reference, the optimal control solution is also provided in Figure 5.16. It is seen that PPO was able to generate a valid (i.e. no state-path constraints violated), minimum time transfer solution within approximately 1,250 episodes (the full training time required to achieve this solution was approximately 21 minutes; a single core on a Linux server, with specifications provided in Section 5.6.1, was utilized to train). Additionally, it is evidenced that the PPO agent solution generally overlaps with that of the optimal solution, only slightly diverging near the target state; this results in position and velocity insertion errors at the target L5 state of 0.8 meters and 3.9 cm/s, respectively. As a consequence, the total time of flight of the PPO generated solution is 19.42 hours, compared to 18.29 hours as achieved by the optimal control solution.

Along with the visualization of the learning curve during the PPO training process, the evolution of the path planning logic can also be recorded and visualized. For the deterministic PPO implemented above, the generalized evolution of the path-planning logic is illustrated in Figure 5.17.

Though the above specified formulation of PPO was found to quickly converge to a path planning solution for a deterministic scenario, it was unable to demonstrate any sort of learning capability for the full perturbation range previously defined in Table 4.1, and consistently implemented for all previous methods. Minor adjustments to the learning hyperparameters and underlying neural network architecture were investigated, though they yielded no considerable results. Additionally implementations that reduced the full perturbation ranges into smaller increments also yielded no considerable results of PPO in regards to both the learning capability of the agent as well as the resultant path planning logic. It was therefore concluded that this formulation of the PPO algorithm may be unsuitable to

be applied for generating path planning solutions for a wide range of dynamical parameters. It is possible that considerable modifications to the structure of the reward function, along with the utilization of an LSTM layer within the neural network architectures may yield better results; however, these modifications require significant further investigations.

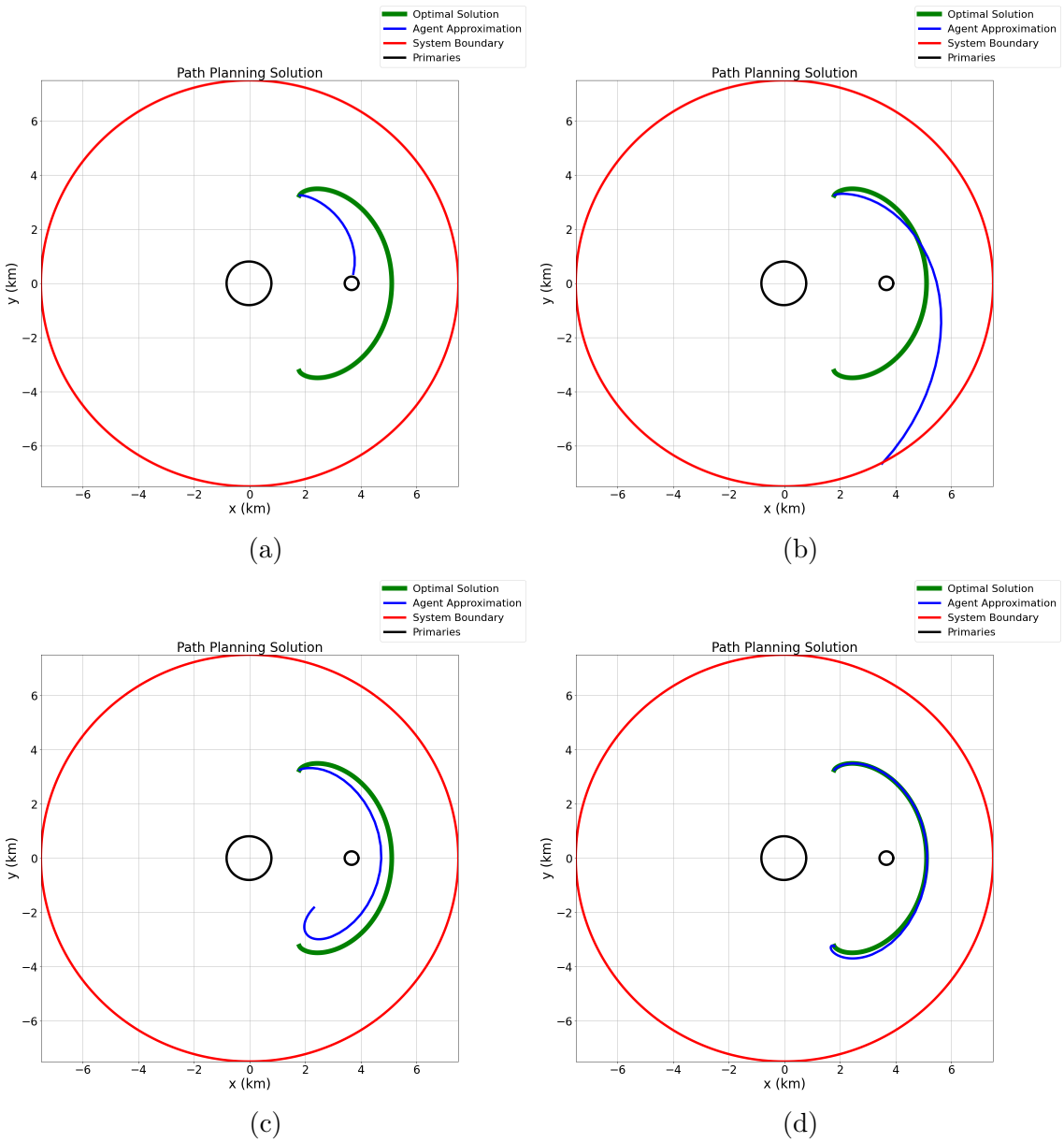


Figure 5.17: Visualization of the evolution of the path-planning policy as during the progression of the PPO training process. The policy evolution follows the alphabetical order as indicated in the subcaptions.

5.6 Discussion of Quantifiable Metrics

Given the above summaries of results for each of the implemented path-planning methods, a discussion on the inherent advantages, drawbacks and possible synergies can be analysed. For the purposes of this investigation, five metrics are defined that may allow for a discussion of the relative performance of the each implemented method. Four of these metrics (runtime, optimality, convergence, and robustness of solution) are quantifiable, whereas the fifth one (user expertise required during implementation) is qualitative. Each of the following subsections independently discusses these defined metrics, for all the methods that were implemented for this investigation.

5.6.1 Runtime

The quantity of runtime can easily be measured by the duration of time needed to complete the coded simulation. For the generation of the optimal control solution database, via the two tiered controller, as well as training all clone agents (optimal control and human demonstration based) and the PPO algorithm, a Linux server with the following specifications was utilized:

- CPU: 24 cores, AMD Threadripper
- Memory: 64 GB RAM
- GPU: 4 GeForce RTX 2080 Ti
- 2TB storage drive
- 512 GB solid state drive

The training of each clone agent only utilized a single core, whereas the two tier controller utilized multiple cores via parallel processing in order to generate the large dataset of solutions. A summary of the computational power utilized, along with the approximate runtimes,

Table 5.4: The computational power utilized for each method, along with the approximate runtimes.

Method	Computational Power Utilized	Appx. Runtime
Two-Tier Optimal Controller	16 cores (parallel processing)	3.5 days
Optimal Control Based Clones		
Feed Forward Version 1	1 core	5 hours
Feed Forward Version 2	1 core	8 hours
LSTM Version 1	1 core	2 days
LSTM Version 2	1 core	2 days
Human Database Clone		
LSTM Network	1 core	2 hours
PPO (deterministic scenario)	1 core	21 minutes

of each method is provided in Table 5.4.

From Table 5.4, it can be seen that the two tiered optimal controller required the longest runtime of approximately 3.5 days. This can be attributed to the low tolerances defined within the optimization schemes (which were defined to be less than 10^{-10}), but also to how the indirect method formulation generates a solution. Recalling from the discussion of indirect methods in Chapter 3, the solver must first guess the appropriate initial values of the co-states that result in all initial and final boundary constraints being respected. If the solver is unable to converge to the appropriate initial values within a certain amount of iterations, then the solver attempts to re-converge with another initial guess of values. Therefore, it is possible that the solver requires longer convergence times if the initial guess of the co-states is not close to any local minima.

Conversely, the feed forward network clones, that were trained using the database generated by the two tier controller, approximately required a runtime on the order of hours in order to fully train via the respective training frameworks. From a machine learning perspective, these runtimes are logical, as the underlying neural network architectures are not deep/complex (i.e. they do not have more than 2 hidden layers, each comprising of hundreds on computational units, such as more than 500). It is a generally known observation that

deeper neural network architectures significantly require longer training times due the computation of the gradients during the backpropagation and weight update process. However, as the implemented feed forward networks for this investigation can be classified as being relatively simpler, the training time required is considerably reduced.

In comparison to the feed forward network clones, the LSTM network clones required approximately two days to train the agents. Again, from a machine learning perspective, this is a logical observation as any type of RNN architectures are characterized by long training times. Additionally, having dense temporal data has also been shown to increase training times for any RNN based network.

Though the clone agent trained from the human pilot database also utilized an LSTM network, it only required approximately two hours of training time in comparison to a runtime on the order of days for the LSTM agents trained from the optimal solution database. This can not only be attributed to a significantly lower number of trajectories within each dataset (approximately 100 solutions from the human generated dataset as compared to over 10,000 solutions within the dataset generated by the two tier optimal controller), but also to the resolution of the solutions. The human pilot based solutions were generated with a framework that propagated motion forward by a fixed time interval at each step, which may result in a lower resolution solution as compared to those generated by the two tier controller.

Finally, the deterministic PPO implementation, as previously discussed in Section 5.4, only required 21 minutes using a single core in order to achieve a converged path-planning solution that closely resembled the optimal solution to a high degree. From numerous implementations of variations on the PPO training framework that was discussed in Section 5.4, it was observed that this considerably low runtime can be directly attributed to the inclusion of the "endgame" concept as part of both the reward function as well as the termination criteria. However, given that the current formulation of the PPO algorithm was unable to learn path-planning strategies on the full perturbation range implemented, it can be inferred

that the learning process may need a considerable amount of time that was beyond the defined time frame of this investigation.

Lastly, it should be noted that even though some of the machine learning based methods may have required long training times, the rollout of the learned policy occurs within the order of seconds. Therefore, if the metric of runtime considers both the training times required and the execution of the learned path-planning strategy, then the machine learning approach that has trained an agent on a wide range of dynamical possibilities may be appropriate for rapid solution generation, albeit with a potential loss in accuracy, in comparison to analytical generation of optimal solutions that account for all possible scenarios.

5.6.2 Convergence

The metric of convergence is defined as whether the method was able to achieve any sort of path planning solution that adhered to the relevant boundary conditions and the state-path constraints. Additionally, the time required to achieve the solution can greatly vary on the order of seconds to tens of minutes as a function of the thrust value, as evidenced in Figures 5.1 and 5.2.

All clone agents, as well as the deterministic PPO implementation, were also able to achieve path planning solutions, though the accuracy greatly varied. However, it can be inferred that the human driven cloning approach may require the longest convergence times, simply due to the unavoidable fact that the human will exhibit their own learning curve during the solution generation process. In a manner similar to RL methods, the human will exhibit instances where they experiment with different actions and observe the consequences as part of their process of building generalized solutions. This learning time required for a human demonstrator can also increase significantly if the complexity of the task increases, and may reach a point of infeasibility. Therefore, modifications to the machine learning model that can extrapolate from sparse amounts of demonstration data, which may also contain uncertainties and mistakes, are greatly warranted.

5.6.3 Optimality

The measure of optimality of each method can be directly compared to that of the solutions generated using the two tier controller. However, recall that this controller architecture first generates an initial solution via the indirect method formulation that is later evaluated for validity and corrected if needed. The indirect method formulation results in a locally optimal solution due to the way that it is derived, and therefore all solutions, both valid and those that required corrections via direct collocation method, are assumed to be locally optimal solutions.

Insights on the optimality of the solutions generated from each of the remaining methods can be obtained via the computation of the relevant performance metrics. Recall that for each clone agent trained, the magnitude of the state error at the final time was directly computed via the performance metric given in Eq. (5.1), and utilized as part of the corresponding result visualizations.

From the relevant figures that visualized the performance of each of the clone agents, it was directly observed that the LSTM network based clone that learned a deterministic policy resulted in the most accurate, clone based, path-planning solutions; however, there was still an accumulation of errors that consequently resulted in the clone unable to perfectly achieve the target state. It was also observed that the feed forward network clone that learned to directly predict α, and the LSTM network clone that sampled α from a predicted Gaussian distribution resulted in similar performance during validation. This may be attributed to the fact that action spaces modeled via continuous distributions are known to be more difficult for machine learning models to train on. As an unfortunate consequence, the performance of the trained agent may be decreased and further modifications to the training architecture are explicitly warranted. Lastly, it was observed that the human demonstration data based clones had the worst performance, which can not only be attributed to the quality but also the quantity of the training dataset. An improvement in this formulation may be achieved by more solution generations, or via the implementation of rigorous domain adaptation methods

that better represent the human's complex, and possibly unintuitive, path-planning logic in a manner that a machine learning agent can both understand and replicate.

Lastly, the deterministic implementation of PPO seemingly offerred the most optimal path planning solution from the machine learning based methods, with a position and insertion error that was considerably less than those achieved by even the best performing clone instances. This may be attributed to the known convergence and robustness properties of the PPO algorithm.

It is possible that the optimality of a machine learning based path-planning solution can be augmented with an optimal controller in order to generate a more accurate solution. For example, a tiered architecture where a machine learning model generates an initial path-planning solution, which is then corrected via direct collocation methods to account for strict adherence to boundary conditions and state-path constraints, can be synergistically implemented to achieve more autonomous and generalized strategies.

5.6.4 Robustness

The metric of robustness is defined as whether or not a given method is able to generate any path planning solution for general applications; in other words, we assess whether the applicability of the method detiorates in the presence of a large range of perturbations.

Again, by the underlying mathematical formulation that drives the indirect and direct collocation methods, the two tier controller can be assumed to be robust to all perturbations at the boundary conditions, as well as the relevant dynamical parameters. However, it should be noted that any change in the aforementioned variables would warrant the re-generation of a solution that is tailored to the specific dynamics and boundary conditions.

The clone agents trained from the optimal solution database display characteristics of robustness, as they achieved path-planning solutions for all the scenarios in the validation dataset. However, in some cases, the solutions generated may be extremely inaccurate with respect to achieving the final, target state of L5, and may require some sort of corrections.

This observation was greatly evident in the clones trained using the simple feed forward network architectures, where the performance accuracy of the agent was only present in cases of low thrust values. Conversely, the LSTM network based clone that learned to directly predict the direction of the thrust control vector provided more accurate path planning solutions across a wider range of scenarios. Therefore, it is possible that the utilization of stronger neural network architectures can increase the robustness of the solution, though the specific formulation required would depend on the application at hand.

The clone agent trained from the current quantity and quality of the human dataset may not be as robust as the optimal solution based clones. Again, can be directly attributed to the low amount of solutions generated by the human pilot, as well as the quality of the solutions. It is also possible that the human pilot may not be able to achieve path-planning solutions for the full range of perturbations that were considered for this investigation; however, considerably more solutions would have to be interactively generated via the real-time simulator in order to assess this hypothesis.

Lastly, though PPO rapidly converged to a solution within a deterministic scenario, it was unable to display any learning evolution for any range of perturbations in the initial boundary conditions, the μ , and thrust values. Therefore, a simple implementation of PPO may not be robust to the level of perturbations that were considered as part of this investigation, and considerable modifications to the algorithm may be greatly warranted. These modifications may stem from using other neural network architectures, such as LSTM networks, rather than the feed forward networks that were implemented for this work. Additional investigations into the reward function formulation and the PPO algorithm hyperparameters may result in better learning ability and convergence to a path planning solution.

It should be noted that this work did not implement any stochastic perturbations along the trajectory during the propagation of the selected control actions. At most, only the initial boundary condition (i.e. the true L4 state) was augmented with both position and velocity perturbations that were randomly selected from the ranges provided in Table 4.1.

Therefore, an assessment on the robustness of the solution as related to perturbations along the path-planning trajectory cannot be made with the current set of results.

5.6.5 User Expertise During Implementation

Lastly, the metric of *a priori* user expertise required for implementing each of these methods should be briefly considered. Though this may not necessarily be a quantifiable value, it is a critical point of discussion that should be considered as part of this analysis.

The implementation of the two tier controller, which utilized both the indirect and direct method formulations, greatly requires that the user understand the underlying mathematical theory that these methods are derived from. Additionally, the determination of the additional boundary constraints that may arise due to stationarity and transversality conditions, as well as the form of the optimization cost functional.

The implementation of any machine learning based methods inherently require that the user not only understand the underlying mathematical framework, as well as the training process for neural networks, but also an intuition on whether the model architecture has been chosen appropriately. Additional intuition, as well as the ability, to tune the model architectures may also be greatly warranted if the application of machine learning based methods for path-planning in dynamically complex environments is to be considered.

Lastly, the implementation of human in the loop training frameworks, that leverage human generated solution datasets, in order to train clone agents, require that the user understand the quality and quantity of the training data. Additionally, as the state of art of this method, especially for spacecraft path-planning application, only exists within the context of this work, the extent of user expertise required cannot be currently assessed.

5.7 Development of Benchmarking Schemes and Identification of Possible Synergies

As mentioned at the beginning of Chapter 4, the purpose of the implemented benchmarking investigation was two-fold: to explore the inherent advantages and drawbacks of the various methods that may be leveraged to achieve autonomous spacecraft path planning, as well as to initially establish a comparative strategy that can be used to further assess the underlying trade offs and possible synergies of each method.

The aspect of the inherent advantages and drawbacks of the various methods can be characterized by the assessment of the previously discussed quantifiable metrics. This discussion noted that the optimal control based methods are not only robust, but can be applicable for the generation of path-planning solutions given that the underlying dynamical equations of motion do not change. In other words, the utilization of the same orbital dynamical model, but varying the dynamical parameters such as μ, thrust, and including perturbations at the boundary constraints, may not warrant the re-formulation of the optimal control based methods. However, if for example, the underlying state dynamical equations of motion are changed from the currently assumed CR3BP model to a higher fidelity, then the entire optimal control method will have to be re-formulated, as the differential equations for the co-state equations are no longer valid. This characteristic of optimal control methods may be an inherent drawback, as the method is not easily generalizable.

Conversely, the utilization of machine learning methods may be able to produce more generalized path-planning solutions. However, in order to do so, supervised learning based approaches would require extensive datasets that encompass all possible types of solutions that can be realized for a wide range of dynamics. Furthermore, if the solutions geometries are different (i.e. direct transfers versus multiple-revolution solutions), the machine learning training process must be modified to consider all geometries equally, instead of only focusing on the dominant solution type and possibly treating all else as outliers. In contrast, RL based methods may be able to achieve generalized solutions without requiring extensive training

datasets, though the training time may exponentially increase.

In order to remove the extensive training times required by an RL agent, the learning process can be accelerated by using human generated path-planning solutions that already characterize generalized strategies. However, in this case, not only must the time required to generate these solutions be considered as a limit, but the quality of the demonstrations is a vital determinant in the learning augmentation process. As mentioned before, the utilization of human generated data, especially in the context of complex problems, will require modifications in the machine learning training model that accounts for mistakes, uncertainties, and even the sub-optimality of the solutions generated.

Though this investigation only implemented the simple path-planning problem of minimum time L4-L5 transfers, further simplifying the solution search space by only considering planar dynamics, the notable observations summarized above were yielded. However, though the benchmarking problem was simple, the consequences of not including additional modifications to the methods was clearly evident. Recall that at the beginning of Chapter 4, it was discussed that the best formulation for each of the methods, especially those based on machine learning, can always be modified in order to be best applicable to the problem. These modifications are directly driven by the nature of the problem and have to be implemented via a heuristic process in which numerous options are considered. For example, the underlying neural network architectures can be rigorously optimized via open source simulation packages such as Optuna, and multiple variations of the training framework that implement variations of the originally defined hyperparameters can also be evaluated. These additional modifications were currently omitted from this investigation.

The exact performance and accuracy of these synergistic frameworks is considered the subject of future work, but an example formulation could be as follows. A machine learning based method, such as RL can be trained on a wide range of dynamical scenarios such that it is able to generate initial path-planning solutions. Given the fact that the training time of the RL agent would increase as a function of the problem complexity, the learning process

can be accelerated by tasking a human demonstrator to provide generalized path planning solutions that are applicable to varying dynamics. Subsequently, a fully trained RL agent can be implemented on-board the spacecraft computer, and generates initial path-planning solutions in real-time. These solutions are then corrected for any state-path constraints violations using an optimal control scheme, before they are executed. This heirarchy can be seen as similar to the implemented two-tier optimal controller, with the initial solution being formulated by a machine learning model rather than the indirect method. The corrections required for the solutions generated by the machine learning model can finally be utilized to update the neural network policy via supervised learning in order to increase robustness. Though this is just one possible synergistic framework that could be implemented, further work is greatly warranted to better determine the applicability of such methods.

Chapter 6

Extended Application: Asteroid Squadron

This Chapter further discusses the exploratory work conducted in regards to human-AI cooperative learning, with the particular application of spacecraft path-planning in random binary asteroid system environments. To the best of our knowledge, this work provides the first empirical observations of not only the human ability to navigate a spacecraft in such environments, but also initial implementations of cloning such behavior in a multi-body space environment.

6.1 Asteroid Squadron

This exploratory work heavily utilized the Asteroid Squadron framework, which is a python based interactive flight simulator where a human is tasked with navigating a spacecraft within a given binary asteroid system. The flight simulator employs a "gamified engineering" approach by posing a spacecraft path planning problem in a chaotic and complex dynamical environment.

Along with surviving as long as possible, the human is also asked to complete navigational and regional exploration goals within the system; these goals are incorporated into the total score via a multi-objective scoring function that is further described below.

6.1.1 Orbit Model

The propagation of spacecraft motion is implemented via numerical integration of the chosen dynamical model. The spacecraft state vector at epoch t_i is defined as

$$\vec{x}(t_i) = [\vec{r}(t_i); \dot{\vec{r}}(t_i)]^T \tag{6.1}$$

and quantifies the current position ($\vec{r}(t_i)$) and velocity ($\dot{\vec{r}}(t_i)$). The dynamical variation of these two state vectors due to dynamical motion are governed by the numerical propagation model of the general form

$$\dot{\vec{x}} = \vec{F}(\vec{x}, t, \vec{p}) \tag{6.2}$$

where $\vec{F}$ is a vector function of the current state vector, the current time t, and the system parameters $\vec{p}$. In other words, the vector function $\vec{F}$ can be used to mathematically denote the underlying dynamical model.

To better describe the spacecraft dynamics within binary asteroid systems, there is a greater emphasis placed on higher fidelity dynamical modeling. In general, a binary asteroid system can be fundamentally described as two primaries in mutual orbit about the system barycenter, which consequently influences the motion of the third mass (i.e. the spacecraft). This regime of dynamics can be described by the Circular Restricted 3-Body (CR3B) model; however, in order to better characterize the dynamics of binary asteroid, the Elliptical Restricted 3-Body (ER3B) model is implemented.

The ER3B model describes the spacecraft dynamics under the gravitational influence of two point-mass attractors that are in mutual Keplerian orbit with semi-major axis (also referred to as the system characteristic length L^*), eccentricity e, orbit period $2\pi T^*$ (where T^* is the system characteristic time), and initial true anomaly angle ϕ_e. Unchanged from conventional notation used in the CR3B formulation, the system mass parameter is denoted by μ. Additionally, the base ER3B model is modified to include irregular gravity modeling caused by a rotating homogenous tri-axial ellipsoid with semi-axes $R_{1x} - R_{1y} - R_{1z}$ and spin rate ω_1 (where the spin axis is orthogonal to the asteroid mutual orbit plane). This ellipsoid models the main primary P1, whereas the secondary primary, P2, is instead modeled as a non-rotating homogenous sphere with radius R_2. Furthermore, the augmented dynamical model also includes the effect of solar radiation pressure on the spacecraft, which are modeled via the cannon-ball formulation.

The motion of the spacecraft itself is described in the rotating, non-pulsating (RNP)

and non-dimensional frame given by unit vectors $(\hat{x}, \hat{y}, \hat{z})$ [88] ($\hat{x}$ is directed along the line extending from P1 through P2, is orthogonal to the mutual orbit plane, and completes the right handed coordinate system).

6.1.2 Real Time Propagation

The propagation of the spacecraft motion is based on the concept of frame epochs, t_i, that are defined equitemporally. Therefore, the spacecraft motion is propagated between successive frame epochs

$$t_{i+1} = t_i + \Delta T \tag{6.3}$$

where ΔT is the real world time interval that has elapsed between two system states.

The flight simulator also renders the effects of random perturbations of the spacecraft state via the implementation of a time scaled covariance matrix within the propagation of the spacecraft state. This scaling is controlled by the following equations

$$K_i(t) = \lambda_i(t) K_0 \tag{6.4}$$

where

$$\lambda_i(t) = \max\left(\boldsymbol{\lambda}(\phi_{x_i}(t, t_i)^T \phi_{x_i}(t, t_i)\right) \tag{6.5}$$

and $\phi_{x_i}(t, t_i)$ is the state transition matrix between $x_i = \mathrm{x}(t_i)$ to *x(t)*, and $\boldsymbol{\lambda}$ is an operator that extracts the eigenvalues of the argument matrix. The scaling factor $\lambda_i(\mathrm{t})$ can be physically interpreted as the estimate of the maximum growth rate for state variation at the initial time [89].

For an initial frame at epoch t_i, the spacecraft state is quantified by a Normal probability distribution $x_i \sim \mathcal{N}(\mathbf{x}|\bar{x}_i, K_0)$ with expected value $\bar{x}_i$ and diagonal covariance matrix K_0. Subsequently, the spacecraft motion is numerically propagated forward in time using the expected state in accordance with the dynamics described in Equation (6.2). As we are implementing a temporal driven covariance matrix, the uncertainty in the state will also scale

proportionally. Therefore, additional measurements are necessary in order to re-establish an adequate level for the spacecraft state prediction accuracy. These measurements are introduced every m frames, where the spacecraft state expected value is updated by randomly drawing a value from the distribution

$$\bar{\boldsymbol{x}}_{i+m} = \boldsymbol{x}(t_i + m\Delta T) \sim \mathcal{N}(x|\bar{x}_i(t_i + m\Delta T), K_i(t_i + m\Delta T)) \tag{6.6}$$

and the state probability distribution is reset to its original form

$$x_{i+m} \sim \mathcal{N}(\boldsymbol{x}|\bar{\boldsymbol{x}}_{i+m}, K_0) \tag{6.7}$$

Due to these periodic resets, the simulated final trajectory may not appear as a continuous curve during data analysis. Using a smaller number of frames m will provide better estimations of the spacecraft state, albeit at increased computational costs.

6.1.3 Online Trajectory Steering

When contextualizing the exploration of small body systems, it is imperative to realize that impulsive maneuvers are considered an appropriate form of control, as small ΔV actions may yield large trajectory variations due to weak gravity fields [90, 91].

Within the flight simulator, there are six different control actions that can be implemented in real time; there are two control actions per maneuver direction axis. To define these maneuver directions, we utilize the tangent $\hat{\boldsymbol{e}}_{\boldsymbol{t}}$, normal $\hat{\boldsymbol{e}}_{\boldsymbol{n}}$ and binormal $\hat{\boldsymbol{e}}_{\boldsymbol{b}}$ unit vectors relative to the trajectory in the RNP frame. These unit vectors are computed via

$$\begin{aligned} \hat{\boldsymbol{e}}_{\boldsymbol{t}} &= \frac{\boldsymbol{v}_i}{||\boldsymbol{v}_i||} \\ \hat{\boldsymbol{e}}_{\boldsymbol{b}} &= \frac{\boldsymbol{v}_i \times \dot{\boldsymbol{v}}_i}{||\boldsymbol{v}_i \times \dot{\boldsymbol{v}}_i||} \\ \hat{\boldsymbol{e}}_{\boldsymbol{n}} &= \hat{\boldsymbol{e}}_{\boldsymbol{b}} \times \hat{\boldsymbol{e}}_{\boldsymbol{t}} \end{aligned} \tag{6.8}$$

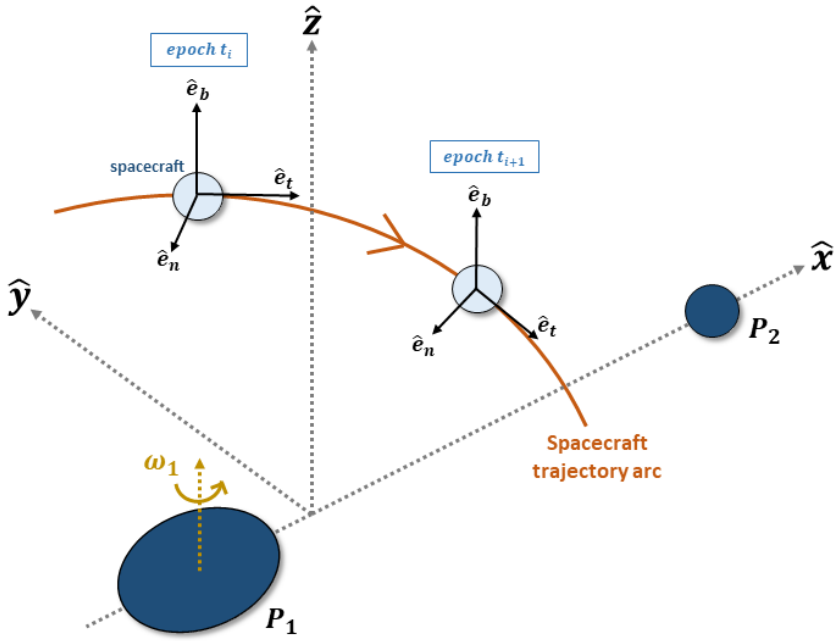


Figure 6.1: The orientation of the spacecraft fixed tangent-normal-binormal frame, computed at consecutive epochs during numerical propagation (scaled for visual clarity)

where $\boldsymbol{v}_i = [\dot{x}\ \dot{y}\ \dot{z}]$ at given epoch t_i. This results in a spacecraft fixed coordinate system that is visualized in Figure 6.1, in relation to the rotating, tri-axial ellipsoid representing the main primary P1 and the non-rotating, spherical secondary primary P2 of a given binary asteroid system. Using Equation (6.8), each of the six impulsive control maneuvers can be

explicitly written as

$$\begin{aligned}
+\text{tangent} &: \boldsymbol{v}_i = \boldsymbol{v}_i + \Delta V \hat{\boldsymbol{e}}_t \\
\text{-tangent} &: \boldsymbol{v}_i = \boldsymbol{v}_i - \Delta V \hat{\boldsymbol{e}}_t \\
+\text{normal} &: \boldsymbol{v}_i = \boldsymbol{v}_i + \Delta V \hat{\boldsymbol{e}}_n \\
\text{-normal} &: \boldsymbol{v}_i = \boldsymbol{v}_i - \Delta V \hat{\boldsymbol{e}}_n \\
+\text{binormal} &: \boldsymbol{v}_i = \boldsymbol{v}_i + \Delta V \hat{\boldsymbol{e}}_b \\
\text{-binormal} &: \boldsymbol{v}_i = \boldsymbol{v}_i - \Delta V \hat{\boldsymbol{e}}_b
\end{aligned} \tag{6.9}$$

Each episode begins with the same amount of available spacecraft ΔV, and each maneuver reduces this reserve by a constant amount. Once the propellant budget is depleted, no further actions are possible.

6.1.4 Range of System Parameters

A summary of the orbit model and the incorporated parameters is given in Table 6.1. Depending on the value assigned to the system parameters in Table 6.1, a large range of spacecraft dynamics within binary asteroid systems may be simulated. The upper and lower bounds for each parameter within the dynamical model are provided in Table 6.2.

6.1.5 Scoring System and Termination Criteria

A realistic space mission will be comprised of multiple science goals or exploration/navigation tasks. Within the context of the flight simulator, there are seven primary mission goals, each corresponding to a different reward function that contributes to the total episode score, inherently guiding the human pilot to developing path-planning strategies. This scalar value may subsequently be used to measure the agent learning progress.

The final score is a weighted summation of intermediate scoring functions, s^n, or subscores, where n represents the corresponding mission goal. The first intermediate scoring

Table 6.1: Asteroid system orbit model components and parameters

Component Type	**Model Component**	$\boldsymbol{p}$	**Description**
Point mass 3 body interaction	ER3BP	e	mutual orbit eccentricity
		L^*	mutual orbit semi-major axis
		T^*	mutual orbit period divided by 2π
		μ	three-body problem mass ratio
		ϕ_e	initial true anomaly for mutual orbit
SRP	Cannon Ball with Sun in apparent circular, coplanar orbit	β	SRP acceleration
		ϕ_β	Sun initial phase in RNP frame
		ω_β	Sun angular velocity in RNP frame
Primary asteroid shape model	Spinning homogeneous tri-axial ellipsoid	R_{1x}	asteroid semi-major axis
		R_{1y}	asteroid first semi-major axis
		R_{1z}	asteroid second semi-major axis
		ϕ_1	initial orientation in the RNP frame
		ω_1	spin rate in the RNP frame
Secondary asteroid shape model	Homogeneous sphere	R_2	asteroid radius

function, s^1, rewards the agent for achieving longer time of flight (TOF) within the system. The second and third intermediate scoring functions, s^2 and s^3 reward the agent for maintaining close proximity orbital motion at the equatorial regions of both asteroids, respectively. The fourth and fifth intermediate scoring functions, s^4 and s^5 reward the agent for maintaining orbital motion at high latitudes (i.e. northern and southern polar) observations of the larger asteroid, respectively. The sixth and seventh intermediate scoring functions s^6 and s^7 reward the agent for maintaining close proximity orbital motion at high latitudes around the larger asteroid. Table 6.3 summarizes the conditions and associated rewards in points per day (PPD) for each of the seven scoring functions. Rewards are a function of the current frame epoch, t_k, the distances from the larger and smaller asteroids, r_1 and r_2 respectively, and the radii of the larger and smaller asteroids, R_1 and R_2 respectively.

Table 6.2: Ranges for $\boldsymbol{p}$ for the asteroid system, denoted by lower and limits p_l and p_u. [4]

Parameter	p_l	p_u	**Parameter**	p_l	p_u
L^* [km]	0.5	7	μ [ndim]	0.0001	0.1698
R_1 [km]	$\frac{L^*}{5}$	$\frac{L^*}{1.5}$	R_{1y} [ndim]	$0.5R_{1x}$	R_{1x}
R_2 [km]	$\frac{R_1}{10}$	$\frac{R_1}{2}$	R_{1z} [ndim]	$0.5R_{1x}$	R_{1x}
P_{mutual} [days]	0.2	2	β [km/s^2]	10^{-12}	10^{-8}
T^* [days]	0.0318	0.318	P_β [years]	0.5	4
e [ndim]	0	0.5	ϕ_1 [degrees]	0	360
ϕ_β [degrees]	0	360	ϕ_e [degrees]	0	360
Derived Quantities	R_{1x} [ndim] = $\frac{0.5R_1[km]}{L^*[km]}$				
	ω_β [ndim] = 1 - $\frac{2\pi T^*[years]}{P_\beta[years]}$				
	ω_1 [ndim] = $\frac{2\pi T^*[hours]}{P_1[hours]}$ - 1				

Table 6.3: Definition of the scoring metrics and associated conditions

Function	**Condition**	**Reward**
s^1	$t_k > t_0$	2 PPD
s^2	$r_1(t_k) \leq 4R_1$	20 PPD
s^3	$r_2(t_k) \leq 4R_2$	50 PPD
s^4	$\arcsin(\frac{z(t_k)}{r_1(t_k)}) \geq \pi/4$	2 PPD
s^5	$\arcsin(\frac{z(t_k)}{r_1(t_k)}) \leq -\pi/4$	2 PPD
s^6	$\arcsin(\frac{z(t_k)}{r_1(t_k)}) \geq \pi/4$, $r_1(t_k) \leq 4R_1$	40 PPD
s^7	$\arcsin(\frac{z(t_k)}{r_1(t_k)}) \leq -\pi/4$, $r_1(t_k) \leq 4R_1$	40 PPD

It is possible to achieve multiple mission goals during an episode. To reward agents that do so, a double reward system is also included. Complementary scoring functions are paired to describe prominent areas of exploration: (s^2, s^3) for equatorial regions, (s^4, s^5) for the distant polar regions, and (s^6, s^7) for near polar regions around the primary asteroid. The distinction between regular and double reward is described by a scalar threshold corresponding to one of the scoring functions of a complementary pair; when one of the scoring functions crosses the threshold, the agent receives double reward for pursuing the

complementary objective. Therefore, the total episode score is described by the weighted summation

$$s = \alpha_1 s^1 + \alpha_2 s^2 + \alpha_3 s^3 + \alpha_4 s^4 + \alpha_5 s^5 + \alpha_6 s^6 + \alpha_7 s^7$$

where the weights α define the double reward system based on the threshold values

$$\alpha_2 = \begin{cases} 1 & s^2 \leq 100 \text{ points} \\ 2 & s^2 > 100 \text{ points} \end{cases}$$

$$\alpha_3 = \begin{cases} 1 & s^3 \leq 50 \text{ points} \\ 2 & s^3 > 50 \text{ points} \end{cases}$$

$$\alpha_4 = \begin{cases} 1 & s^4 \leq 20 \text{ points} \\ 2 & s^4 > 20 \text{ points} \end{cases}$$

$$\alpha_5 = \begin{cases} 1 & s^5 \leq 20 \text{ points} \\ 2 & s^5 > 20 \text{ points} \end{cases}$$

$$\alpha_6 = \begin{cases} 1 & s^6 \leq 20 \text{ points} \\ 2 & s^6 > 20 \text{ points} \end{cases}$$

$$\alpha_7 = \begin{cases} 1 & s^7 \leq 20 \text{ points} \\ 2 & s^7 > 20 \text{ points} \end{cases}$$

The scalar weight α_1, which corresponds to the TOF in the system, is always equal to 1 and is not included in the double reward system.

In order to better simulate realistic dynamical conditions, the inclusion of catastrophic events within the flight simulator is required. From an orbital dynamics perspective, there are three primary events that can immediately terminate the mission: crashing into either asteroid or escaping the binary asteroid system. We model primary catastrophic events

Figure 6.2: The interface for the real time flight simulator that visualizes spacecraft motion (1), primary and secondary asteroids (2), outer ring representing the termination criteria for escape (3), apparent sun position (4), and the interface dashboard (5).

using conservative termination conditions. Within the flight simulator, a crash into the larger asteroid occurs when $r_1(t_i) \leq 2R_1$, a crash into the smaller asteroid occurs when $r_2(t_i) \leq 2R_2$, and the system escape criterion is $r_{BC} \geq 3L^*$, where r_{BC} denotes the spacecraft distance from the asteroid system center of mass. If any of the termination criteria are met, the episode is immediately interrupted but without penalty on the accumulated total score.

6.1.6 Flight Simulator Interface

The flight simulator is designed such that real time spacecraft motion and other relevant information is intuitively presented to the human agent; Figure 6.2 illustrates this graphical interface. At any given instant, the agent can observe the spacecraft motion as approximated by the real time propagation routine within the RNP frame. Both asteroids and their dynamical motion are also displayed, along with an outer ring that marks episode termination conditions by system escape. The current Sun angular position is also represented by a rotating star that travels along the outer ring.

The interface dashboard is to the right of the binary system graphical representation, and presents the agent with mission progress and other relevant criteria. Along with the remaining fuel reserve level, the episode TOF and latitude measurements are made available

Table 6.4: Fixed parameters for the simulation framework.

Initial V	30 m/s
Maneuver Size V	0.03 m/s
Position Navigation Error	10 m
Velocity Navigation Error	1 mm/s
Measurement Update Interval	30 minutes
Frame Prop. Interval	2 minutes
Visualization Update Interval	20 fps

to the agent. The spacecraft z coordinate is represented by the yellow sliding bar along the left side of the dashboard. The agent is presented with warning signals if the current trajectory will result in a termination criteria; the creation of warning signals is described in previous work [92]. Lastly, both accumulated and primary mission sub-scores are visible. Progress on each of the mission sub-scores is also made available.

6.2 Human Pilot Database

6.2.1 Data Collection Experiment

Each episode of human pilot training, which can also simply be referred to as an episode, represents a single flight simulation. Each episode begins at the L4 Lagrangian point of the system, with zero relative spacecraft velocity. Due to the inherent stability properties [80], this equilibrium point may be a good home position candidate for a spacecraft within binary asteroid systems. Fixed simulation parameters are listed in Table 6.4; these include spacecraft-related properties (such as navigation errors and maneuvering capabilities) as well as values for configuring the simulation interface.

6.2.2 Emergent Path Planning Strategies

The following analyses presented are based on data collected from a single human pilot agent. The collected data corresponds to different, sequential episodes, where each one comprised of binary asteroid systems generated by the random selection of parameters from

the ranges presented in Table 6.2. Additionally, during post-processing of the generated database, a small number of episodes that correspond to outlier scores are removed from the dataset to avoid statistical biases.

6.2.2.1 Human Agent Learning Curve

Analyzing the human agent learning curve may offer insight into the improvement of the agent's performance as more experience is gained. Figure 6.3 illustrates the learning curve for all episodes in the collected data, as represented by a moving average of the total episode scores (window size is set to 100 episodes, with end effects removed). The learning curve visualizes the ability of the human agent to learn generalized path planning strategies that are applicable to a large range of asteroid system types. The vertical axis represents the final, or total, score obtained by the end of each episode. The total score can be used as a measure of mission success as higher values correspond to simulations having longer flight times and accomplishing greater primary mission goals, as denoted by the definition of the underlying game scoring function.

From Figure 6.3, we can see that as more experience is gained by the human pilot, the total score follows a generally increasing trend that may be approximated by a 4th order polynomial. Deviations from this best fit polynomial can be attributed to an alternation of the infancy-exploration-exploitation phases that are characteristic of human learning. The human pilot appears to have followed a sequence of repeated cycled of exploration and exploitation (which are represented by deviations below and above the approximation polynomial function, respectively). This observation is in stark contrast to previous similar work, in which the infancy-exploration-exploitation phases were more distinctly observed [92].

The behavior of the learning curve may be linked to the human agent gradually exploring different regions of the generated binary asteroid systems. As the number of training episodes increases, the human agent begins to face a tradeoff between further exploring the

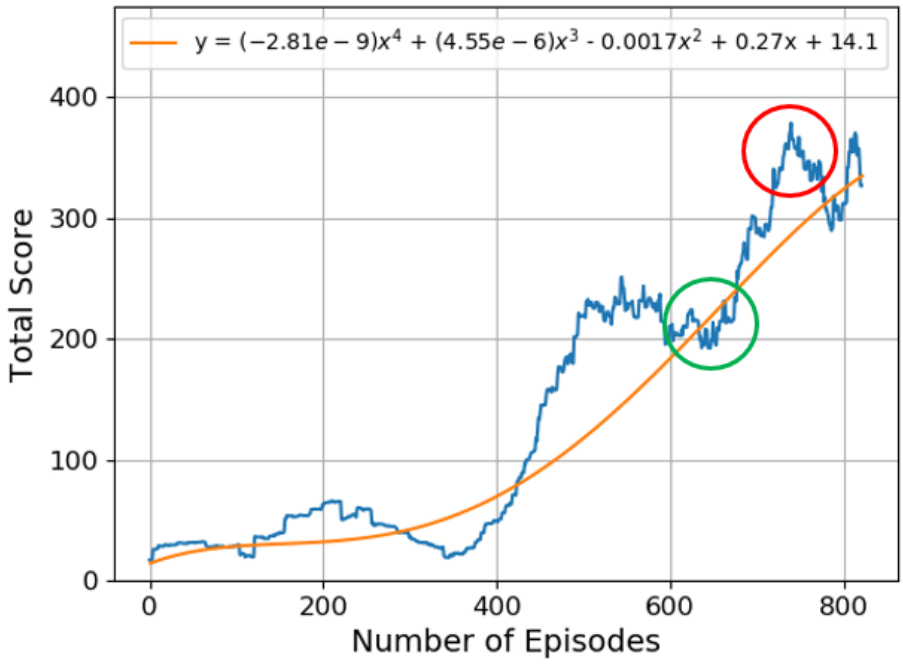


Figure 6.3: The moving average of the human agent learning curve (blue) overlayed with a polynomial fitted curve (orange)

system and exploiting a known strategy that guarantees a certain amount of points. A distinction between different episode strategies may be based on the ΔV consumed and the time of flight (TOF). Each episode begins with a fixed amount of fuel, with each maneuver depleting the reserve by a constant amount. If the agent is exploiting a known strategy, then the may be able to predict the spacecraft motion and implement a developed path-planning strategy that yields longer episode duration. On the contrary, an agent that is exploring unfamiliar system dynamics might not be fully aware of the effects of an action. Consequently, catastrophic events such as crashing into either asteroid or leaving the asteroid system, are more likely to occur. As the episode ends quickly, a lower amount of fuel is consumed. In this sense, the ΔV consumed and the TOF become an index, among others, to discern between exploration and exploitation phases.

Identification of exploration and exploitation phases on the bases of ΔV consumed and the total TOF is demonstrated in the following analyses of the s_2 and s_3 subscore trends. Though only two subscore influences are presented in this analysis, similar studies may be

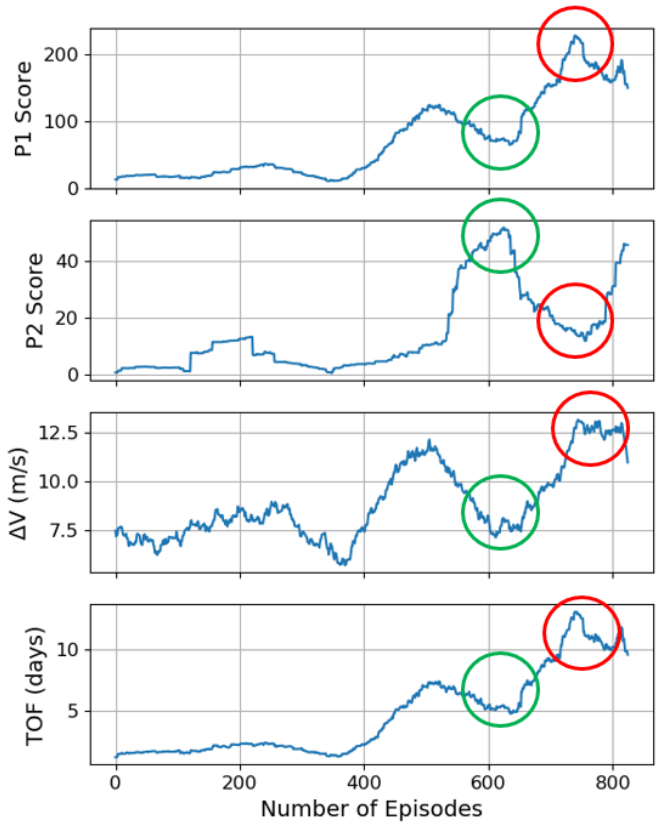


Figure 6.4: Sub-score influence on the visible peaks and dips in the human agent learning curve

performed in consideration of the remaining sub-scoring functions.

Figure 6.4 illustrates the moving average for the P1 and P2 sub-scores (i.e. near orbital plane motion around each of the primaries), as well as the ΔV consumed and TOF (all moving average window sizes are set to 100 episodes, and end effects are removed). Episodes within the first range of interest are designated by green circles in both Figures 6.3 and 6.4, and occur around the 650 episode mark. In this range, the total score moving average drops below the approximation, and there is also a correlated decrease in ΔV consumed, TOF duration, and P1 scores; simultaneously, there is a substantial increase in the P2 scores. Such trends may be interpreted as the human agent primarily exploring the region around P2.

Episodes within a second range of interest are designated by red circles in Figures 6.3

and 6.4, and occur around the 750 episode mark. In this range, the total scores peak above the approximation, and P1 scores, TOF duration, and ΔV consumed also increase; there is a correlated decrease in the P2 scores. The trends in this range may indicate that the agent is human pilot is choosing to exploit a known strategy for orbiting P1 bear the orbital plane rather than exploring other regions of the asteroid system.

Conversely, another measure of improvement of the human pilot s performance can stem from analyzing the episode efficiency trends. For a given episode, the episode efficiency can be quantified by the parameter $\eta_{episode}$ such that

$$\eta_{episode} = \frac{\Delta V_{consumed}}{\text{total score}} \tag{6.10}$$

Therefore, by definition, a lower $\eta_{episode}$ value correlates to a lower fuel cost per unit score and consequently a higher episode efficiency. Figure 6.5 illustrates the $\eta_{episode}$ value for the collected data, represented by the moving average (window size of 100 episodes, with end effects and episodes with zero total score removed). The $\eta_{episode}$ value decreases in a generally linear fashion as more experience is gained. This is indicative of the human agent learning more efficient ways to navigate the spacecraft within different asteroid system types. From both Figures 6.3 and 6.5, it is evident that the human agent is acquiring control of the spacecraft orbit dynamics within randomly generated binary asteroid systems. As experience is gained, the total score increases as does the episode efficiency, implying that the human agent is learning to achieve higher mission return via more efficient control strategies.

6.2.2.2 Maneuver Analysis

Actions that are called upon more frequently can prove vital in understanding the control of the resulting trajectory. As previously described in Section 6.1, there are six possible impulsive control maneuvers available within the game framework; each maneuver will deplete the available fuel reserve by a fixed amount. From the generated human pilot database,

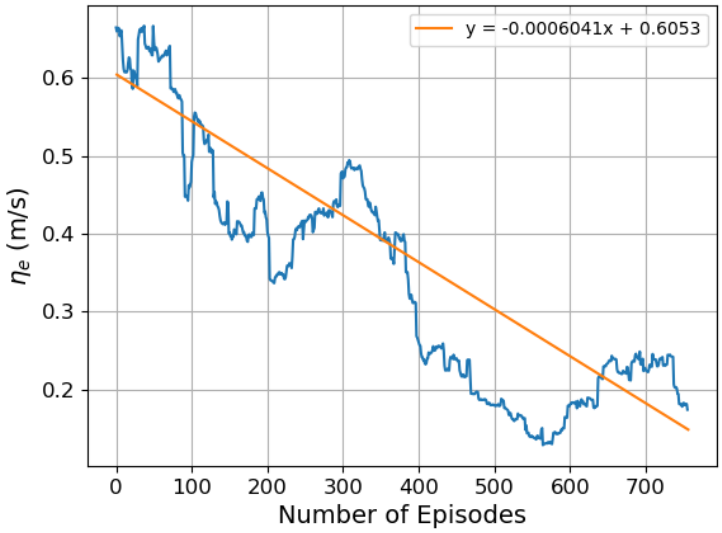


Figure 6.5: The moving average of the mission efficiency (blue) overlayed with a liner fitted curve (orange)

we can extract the percentage decomposition for each control maneuver; this data is presented in Table 6.5. Additionally, the action frequencies within the x-y and x-z parameter spaces are presented in Figures 6.6a and 6.6b respectively. These frequency maps are generated by plotting the spatial distributions of called actions within a given configuration space. The parameter space distances were normalized by the episode's characteristic length L^*.

From the frequency maps in Figure 6.6a, it is observed that positive and negative normal control actions are the most frequently utilized. Given the orientation of the spacecraft coordinate system as illustrated in Figure 6.1, the normal unit vector is aligned in the direction towards the center of curvature of the trajectory arc. High frequencies of both these actions occurring near the asteroids indicate that they were employed to maintain orbital motion about both P1 and P2. There is also a high frequency of positive and negative around L4 as each new episode begins with the spacecraft located at this equilibrium point. The negative tangent action is used to decrease the spacecraft mechanical energy relative to the asteroid, resulting in a lower semi-major axis and consequently allowing for a temporary orbital capture . There is a relatively even frequency distribution for this action. The positive

Table 6.5: The percentage decomposition for all control maneuver actions imparted in the collected data.

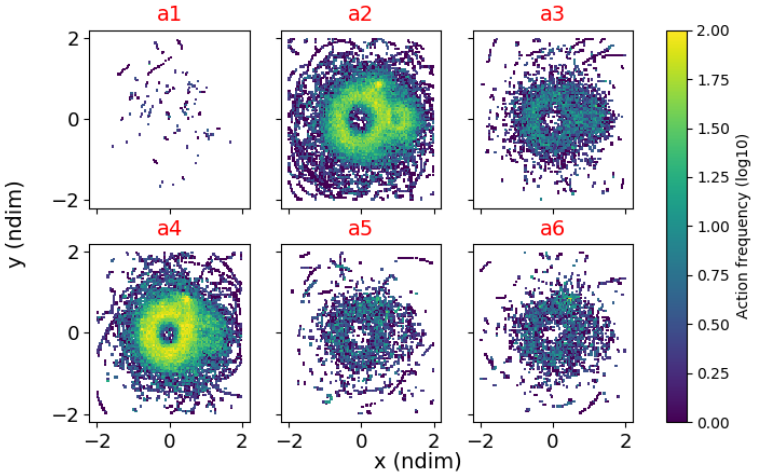
Total Maneuvers: 209,989	
+ tangent (a_1)	0.18 %
- tangent (a_3)	6.19 %
+ normal (a_2)	38.58 %
- normal (a_4)	47.14 %
+ binormal (a_5)	4.05 %
- binormal (a_6)	3.86 %

tangent is used least frequently as this would increase the mechanical energy of the spacecraft, which would increase the semi-major axis and decrease chances of temporary orbital capture. Lastly, control maneuvers along the binormal axis are used to move out of plane; these actions also have a relatively even distribution but low frequencies when compared to actions imparted along the normal axis.

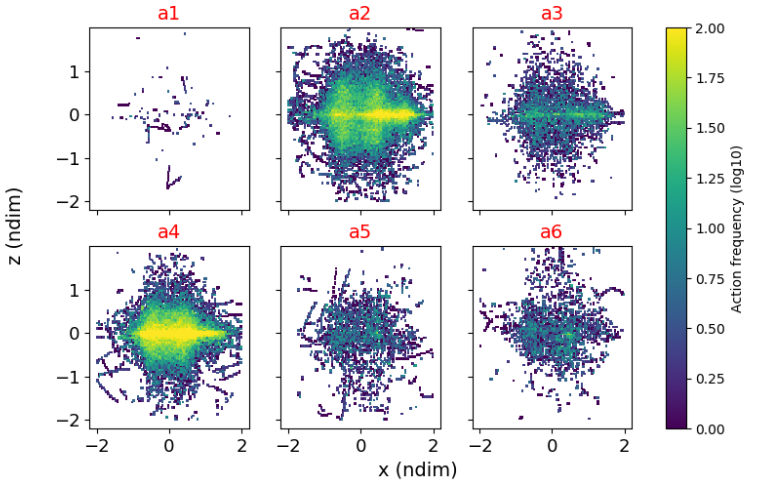
Frequency maps given in Figure 6.6b, which represent action spatial distributions in the x-z parameter space, offer a different perspective. Actions along the normal axis were utilized for out of plane motion, but there remains a heavy concentration at and near the orbital plane. The negative tangent action is seen to be utilized primarily near the orbital plane as a ΔV maneuver in this direction would decrease energy, thereby reducing the semi-major axis and bringing the spacecraft closer to the asteroid. Finally, actions along the binormal axis have relatively even distributions but low frequencies.

6.2.2.3 Controlled and Natural Motion

Analyzing the spacecraft's controlled and natural motion within the asteroid systems may reveal inherent dynamical pathways. More specifically, analyzing the action spatial distributions along the resultant trajectory may offer insight into the nature of the spacecraft's motion. The data presented in this section is from a single episode, which is chosen as representative case study. Figure 6.7a illustrates a trajectory arc (in pink) that captures the spacecraft transit from motion near P2 to motion near P1. Figure 6.7b portrays, instead, a



(a) Action frequencies in the x-y parameter space

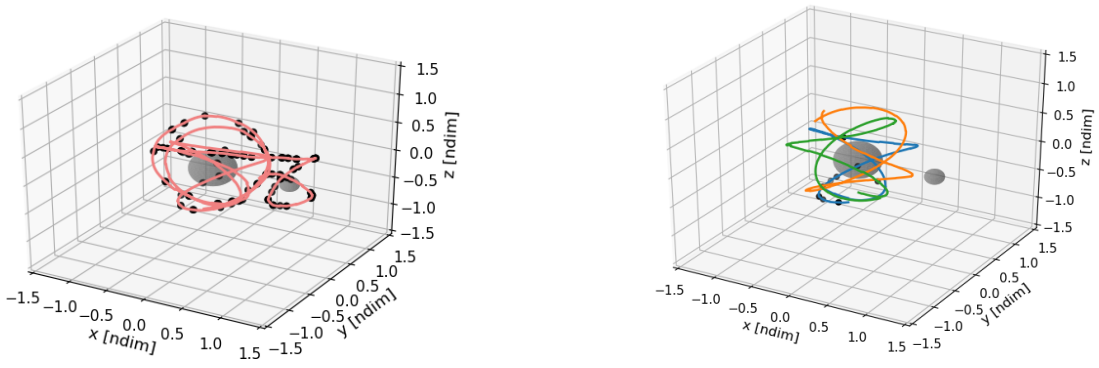


(b) Action frequencies in the x-z parameter space

Figure 6.6: Spatial distribution maps for the action frequencies

trajectory near P1 that occurs after the transition illustrated in Figure 6.7a. In both Figures 6.7a and 6.7b semi-transparent spheres represent P1 and P2 (semi-transparent in order to observe motion behind each asteroid), and the occurrence of ΔV maneuvers is indicated by black dots located at the point of execution of the corresponding maneuver.

From observing the frequency of ΔV maneuvers in Figure 6.7a, motion about P2 appears heavily controlled, as is the transfer from P2 to P1. Furthermore, the motion about



(a) (b)

Figure 6.7: Emerging differences in the spacecraft path planning, denoting instances of controlled and natural motion, with imparted actions represented by black circles. Subfigure (a) represents a selected transfer trajectory (pink) from P2 to P1. Subfigure (b) represents the selected sequential motions (blue, orange, and green respectively) about P1

both P2 and P1 during this phase appears to be vertical in nature. In Figure 6.7b, a transition from predominantly controlled to predominantly natural motion may be observed. The trajectory about P1 may best be described as successive, vertical, eight-shaped arcs, which are colored in blue, orange, and green respectively in Figure 6.7b. Each vertical, eight-shaped arc is distinguished by a self-intercepting point in configuration space. After the spacecraft has completed a vertical, eight-shaped arc, it transfers to a new arc and the self-intercepting point moves to a new location; the resulting spacecraft motion near P1 may be described as a combination of vertical, eight-shaped oscillations and precession of the self-intercepting point. During such motion, the only actions imparted are along the beginning of the blue arc near the southern hemisphere of P1 (see Figure 6.7b). Motion represented by the orange and green arcs may be a result of the natural dynamics of the system as there were no actions employed during these periods.

6.3 Automating Human Path-Planning Logic: Interactive DAgger

Given the generation of human generated solutions, the resultant dataset can be utilized to train machine learning methods in an attempt to automate the underlying path-planning logic. As mentioned previously, and to the best of our knowledge, this is the first instance in which an automated replication of human demonstration data for spacecraft path-planning is attempted. Therefore, due to the lack of state of art and an initial lack of knowledge of the required modifications to the underlying machine learning models, a relatively intuitive and straightforward approach is implemented. The chosen formulation is further discussed in the sections below.

6.3.1 Dataset Aggregation (DAgger)

The generation of path-planning strategies is a sequential prediction problem, wherein future observations are directly dependent on previous actions. The agent's behavior prediction directly influences future states that are encountered, the critical independent and identically distributed, or i.i.d., assumption that underlies statistical learning is inherently violated. Therefore behavioral clones purely based on statistical learning typically experience a compounding of errors after an incorrect prediction, as the future distribution of states will be different from those in expert demonstrations. Differences in state distributions between those in expert demonstrations, and those encountered during training is known as distributional shift [93]. Various solutions have been proposed that allow the agent to overcome distributional shifts, with the most popular method being DAgger. Dataset Aggregation (DAgger) reduces the structured prediction problem to that of an online learning setting [33] and may facilitate the application of statistical learning to sequential prediction problems. Online learning describes methods in which data becomes available to the agent at each step, and is used to update the model and predict the future action, without necessarily having to retain the observation. Conversely, offline learning is when agents train on repositories of data trajectories in order to build models.

The DAgger algorithm has a straightforward training architecture based on the concept of training batches for data collection. We begin by first training a behavioral clone from numerous expert demonstrations, commonly termed as the expert policy π_e [33]. The first batch consists of multiple episodes in which only π_e is used to generate trajectories; the number of episodes in each batch can be explicitly defined for each application. During the rollout of each episode, tuples of state-action pairs are recorded. DAgger requires us to query the expert at each step in order to assign the action labels, making the algorithm one that has an active setting. At the end of each batch, all tuples are aggregated to the main dataset used to retrain the policy of the learner agent π_l before its next execution. As more batches i are iteratively used to collect training data, we begin to use to use the learner policy more frequently by specifying the executed policy π to be a combination of two components

$$\pi = \beta_i \pi_e + (1 - \beta_i)\pi_l$$

where β_i is a chosen scalar parameter less than 1. The value of β_i differs for each problem, with the only requirement being that $\beta_i \longrightarrow 0 \, as \, i \longrightarrow \infty$.

6.3.1.1 Active Imitation Learning: Coached IL

Though DAgger is a commonly utilized algorithm for IL, it still suffers from inherent drawbacks that can severely limit the agent's performance. One of the main features of DAgger is its use of stochastic policy mixing of the expert and learner policies, implemented via use of the β_i parameter. This shifting of control authority of the executed policy to the learner agent at certain times is commonly termed as "Robot-Centric (RC)" sampling [52]. As a result, the resultant trajectory of states visited may not only be greatly deviated from the nominal trajectory of the expert, but it may have also been influenced by a suboptimal learner policy.

Another potential limitation of passive IL algorithms is the reliance on an expert policy,

which may not fully capture the intricate logic of the human expert [94]. Differences in perception (what the human expert observed versus what the agent is able to observe) used to make a decision may further hinder the development of an autonomous policy that is purely based on cloning behavior. The development of a policy using imitation learning relies on the use of certain input parameters. However, only some of these parameters directly influence the action, whereas the rest do not contribute to the decision logic [93]. If the agent learns a policy that heavily relies on "nuisance" variables, it will lead to poor performance during testing [93]. There exist methods, such as causal-graph parameterized policy learning [93], that allow for the differentiation between the true causal and nuisance variables, although they have only been validated on low dimensional state spaces.

Real time human interaction while training a learner policy, a technique known as interactive imitation learning, may mitigate some of the challenges in passive imitation learning. There exist different levels of control authority that allow for an interactive expert to aid during training of the learner policy. The first logical option is that the interactive expert can intervene and impose control authority whenever deemed necessary, and this approach is utilized in the COrrective Advice Communicated by Humans (COACH) interactive training framework [45]. Given that the COACH framework allows for the interactive expert to take control at any time, Pérez-Dattari et al. designate two types of coaching feedback that can be used: corrective and evaluative [45]. Corrective feedback is preferred when the interactor wants to teach the agent how to avoid catastrophic events that would terminate the episode [45]. However, although the agent may learn to avoid terminal events, it may still be unable to learn the overall desired behavior if the given problem lies in a high dimensional space. Conversely, evaluative feedback is similar to an RL approach in that the interactor directly provides feedback on the desired behavior of the agent [45].

Another approach at incorporating an interactive human expert relies on a gating function within the control logic [94, 52]. If the agent predicts an action that falls below a defined confidence threshold, then the control authority is switched to the human interactor, who

proceeds to control and gather data until the next instance the agent can perform confidently. This confidence threshold can be defined differently for separate problems, from using the covariance of the predictions given by using parallel neural networks to predict the agent actions [52], to having the interactive coach slowly build up to executing the full desired behavior and allowing the learner to incrementally learn good behavior [94]. The choice of coaching and the control authority of the interactive expert differs for each problem.

6.3.2 Agent Training Architecture

The underlying architecture chosen to train an agent that would play Asteroid Squadron is based on the original DAgger algorithm; DAgger relies on the use of a behavioral clone for its stochastic policy mixing during execution of each batch. Therefore, a behavioral clone was first trained using high scoring episodes from the human expert dataset; during evaluation the behavioral clone had an action prediction accuracy of 96.1%. It should be noted that more than 90% of all actions taken by the human expert were null actions, in which no control maneuvers were actuated. That caused a bias in the data set with significantly longer periods of spacecraft coasting versus instances of control, but the bias was not removed for any portion of the training within this experiment. Instances of explicit control strategy that separate the periods of null actions are representative of the human pilot's learned situational decision making logic, and removing this biases may inadvertently affect the desired performance of the trained agent.

6.3.2.1 Neural Network Architecture

The behavioral clone, as well as all trained agents, used the same underlying neural network architecture and input parameters; this architecture is visualized in Figure 6.8. We currently choose to employ a 2D convolutional neural network (CNN) in this investigation, for a few different reasons. First, this neural network architecture is the simplest

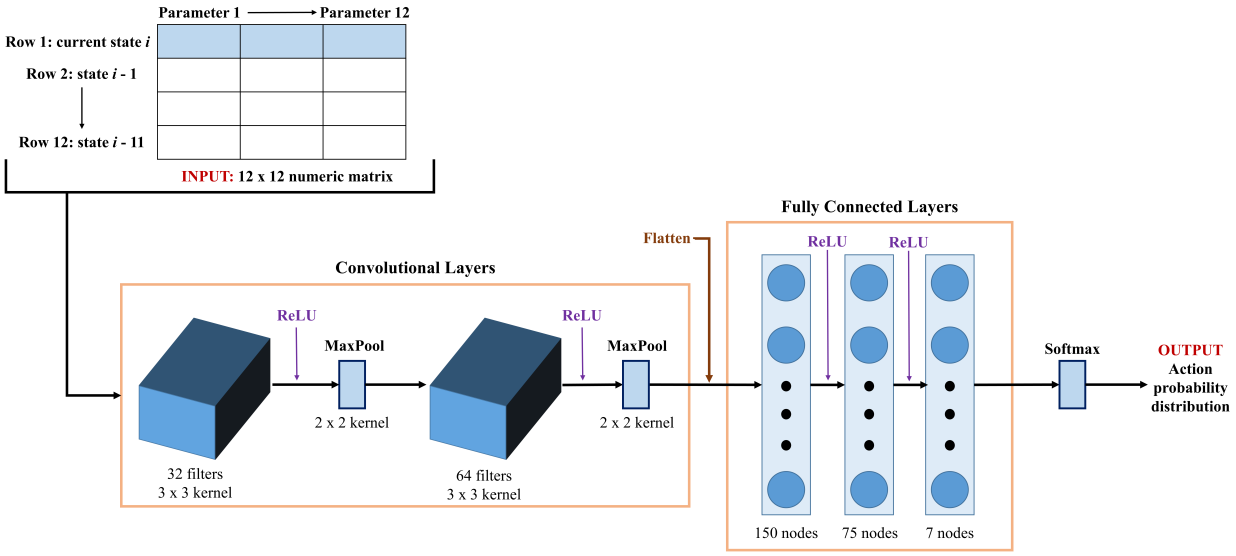


Figure 6.8: 2D convolutional neural network architecture utilized for training all agents.

formulation that may be able to solve a time series dependent, sequence prediction problem without relying on any characteristics that are typical of a recurrent neural network (RNN). Though RNN's may provide more powerful solutions, they can also suffer from the well-known phenomenon of the exploding and vanishing gradients [95]. Exploding gradients arise when large error gradients accumulate, resulting in significant updates to the neural network weights following the gradient backpropagation and resulting in an unstable architecture. Conversely, vanishing gradients refers to when the the norm of the gradient during backpropagation begins to converge to zero. This results in the neural network having great difficulty in learning the correlation between temporally distant events [95].

The use of CNN's as an alternative to RNN's has been explored, and these architectures may possibly serve as an alternative approach at sequence modeling and prediction [96]. Before the widespread use of RNN's, CNN architectures were previously applied to sequential tasks, and recent advancements in the development of temporal CNN architectures have been shown to avoid the problem of vanishing gradients, as the backpropagation occurs in a direction that is different from the temporal direction of the sequence [96]. Conversely, the

use of the Rectified Linear Unit (ReLU) activation function within a CNN may also mitigate the problem of vanishing gradients, and reduce the need for a complex, temporal CNN [97].

The second reason as to why we implement a CNN is that is allows for the input to be numeric matrices; this can be interpreted as allowing the input to be a rolling lookback window of the state history for the chosen state feature vector parameters. Providing this rolling lookback may aid the CNN in regards to short term memory retention during the training process.

Leveraging the idea of a rolling lookback window, our input to the implemented CNN is as follows. For each time step, twelve parameters are used as inputs by the agent: the spacecraft position, velocity, total score, fuel remaining, the x positions of both asteroids, and the spacecraft's distance from each asteroid. Each of these parameters is a designated column in the input matrix, and each row is the state at a specific time step. The first row is the current state i that the agent is experiencing, and each row thereafter is the state at the previous step. Currently, the input matrix is 12x12, meaning that along with the current state, the agent is recalling the past eleven states it has visited, in an effort to provide trajectory history and possible situational awareness.

The 12x12 input matrix passes through two convolutional layers to extract the underlying features, with each layer being followed by a ReLU activation function and a MaxPool layer. The MaxPool layer is included to create a summarized version of the features detected in the input matrix, and may address the issue of variance in the input matrix data (note that each episode of the simulator is a randomly generated binary asteroid system, and therefore variances in the input matrix are to be expected, and may be mitigated by the inclusion of a MaxPool layer). Following the convolutional layers, there are three hidden layers in the subsequent feed forward, fully connected network. The first two layers have 150 and 75 hidden nodes, respectively, with ReLU activation functions. The final hidden layer is 7 nodes, with each node corresponding to one of the actions in the total action space for this problem (1 null action, and 6 impulsive maneuvers). This layer has a Softmax activation

function, which results in the final output of the implemented CNN to be the probabilities of choosing each action. These probabilities are then used to randomly select an action, from our discrete action state space, that is then implemented to control the spacecraft.

6.3.2.2 Neural Network Training Parameters

From looking at the state input variables, we note that as the agent interacts with the environment, no information regarding the asteroid physical properties (e.g., mass and shape) is given. For this experiment, no additional considerations were taken to identify the possibility of "nuisance" variables that result in causal confusion, due to the high dimensionality of the problem. Each agent, as well as the behavioral clone, was trained on the respective datasets twice, using shuffling and batch sizes of 25,000 data points; the neural network in Figure 6.8 was utilized as the common architecture for both the behavioral clone and the agents.

The interactive PROPAGATE (ex**P**e**R**t c**O**aching for **PA**th plannin**G** **A**nd orbi**T**al stationk**E**eping) training architecture is illustrated in Figure 6.9. As mentioned before, the DAgger algorithm is used as the primary basis for agent training. Additional modifications allow for a real time human monitor to observe the agent's behavior at controlling the spacecraft motion, as well as providing corrections in real time. When aggregating the dataset for retraining the learner policy, the human monitor's response is used: if the monitor did not explicitly specify a control maneuver, and instead agreed with the agent's chosen behavior, then the agent's actions are used for labeling. Otherwise, the control maneuver dictated by the human monitor is utilized. For this experiment, the human monitor is the same human pilot that gathered the training data used to train the behavioral clone. Corrections provided to the agents by the human monitor are based on the heuristic path planning strategies learned during the 60-hours-long piloting experience with the original Asteroid Squadron simulation interface.

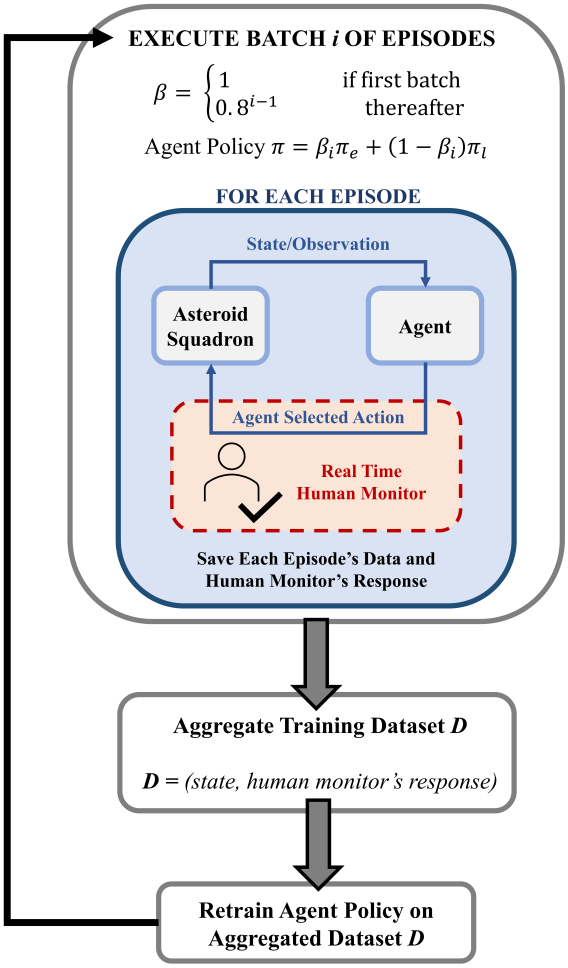


Figure 6.9: The interactive PROPAGATE training architecture

The application of heuristics by the human monitor may introduce multimodal behavior in the learning process. Multimodal behavior may be mitigated by a careful selection of the state feature vector that is utilized as neural network input. Expanding the state feature vector may results in comparatively less frequent multimodal behavior episodes, at the cost of increasing problem dimensionality and decreasing state-action pairs sampling. The analysis of multimodal behavior during learning is a subject for future expansions of this work.. Given that the PROPAGATE architecture allows for direct interaction between the agent and the human monitor, different types of coaching can be employed during training, resulting in multiple agents. It should be noted that, while the total score is passed to the neural network as an input for behavioral cloning, the purpose of the reward function is to

drive the behavior of the human pilot, whose learned path planning strategies may then be transferred to the agent during the interactive training.

6.3.3 Agent Versions and Control Authority Switch

Using the PROPAGATE training architecture, three separate agents were trained. The first is a pure DAgger agent, in which the human monitor has no control authority in the online trajectory steering for spacecraft control. Any corrections provided by the monitor are only recorded to be used as part of the learner policy training dataset. The representative algorithm and control logic for this agent is given in Algorithm 1 and Figure 6.10. The other two agents represent coached DAgger implementations, one for corrective coaching, and the other for evaluative coaching. In both scenarios, the human monitor's corrections were both recorded and implemented in real time. With the corrective coaching agent, the human monitor only imposed control authority in order to steer the agent away from a terminal condition (crash into either asteroid, or escape from the binary asteroid system). Conversely, with the evaluative coaching agent, the human monitor imposed control authority whenever they deemed fit in order to prompt desired behavior that is reminiscent of the control strategies in the original human pilot dataset. The representative algorithm and control logic for both of these agents is given in Algorithm 2 and Figure 6.11.

Algorithm 3 Pure DAgger Formulation

1. Initialize behavioral clone/expert policy π_e, learner policy $\pi_l \longrightarrow 0$, training dataset $D \longrightarrow 0$, $\beta \in (0, 1)$
2. For episode i in episode batch n = 1:N
   (a) Execute policy $\pi = \beta^{n-1}\pi_e + (1 - \beta^{n-1})\pi_l$
   (b) Record dataset tuples $D_i = \{(s, a^*)\}$ of visited states s and action labels a^*
   (c) Append all generated tuples to existing training dataset $D \longleftarrow D \cup D_i$
3. Retrain π_l on expanded D

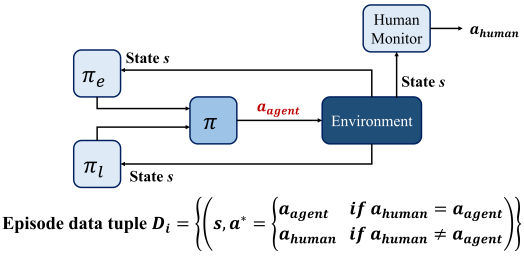


Figure 6.10: Control logic for the pure DAgger agent

Algorithm 4 Coached DAgger Formulation

1. Initialize behavioral clone/expert policy π_e, learner policy $\pi_l \longrightarrow 0$, training dataset $D \longrightarrow 0$, $\beta \in (0, 1)$
2. For episode i in episode batch n = 1:N
   (a) Execute policy
   $$\pi = \begin{cases} \text{if } a_{human} = a_{agent} \\ \text{if } a_{human} \neq a_{agent} \end{cases}$$
   (b) Record dataset tuples $D_i = \{(s, a^*)\}$ of visited states s and action labels a^*
   (c) Append all generated tuples to existing training dataset $D \longleftarrow D \cup D_i$
3. Retrain π_l on expanded D

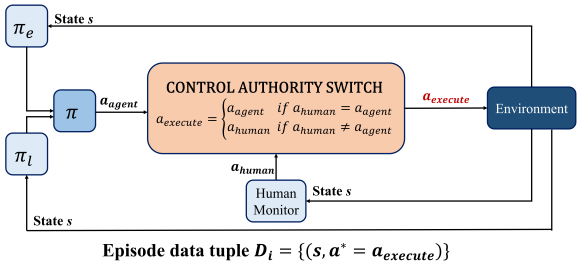


Figure 6.11: Control logic for both coached DAgger agents

Though the Asteroid Squadron flight simulator was the environment that each of the agents were interacting with, the range of the asteroid system parameters was not reduced. Rather, the same ranges for the parameters, as presented in Table 6.2, were utilized as the goal of this investigation is to assess the ability of an interactive, human-machine learning

framework at solving a complex and high dimensional problem. If the ranges of the parameters were reduced, then the complexity of the problem would effectively be reduced as well, resulting in the investigation of a scenario that is fundamentally different to the primary goal.

6.4 Results and Discussion of Interactive DAgger

6.4.1 Learning Curves

Each of the three DAGGER agents was trained for a benchmark of 20 hours (this represents the time given for the interactive data collection, and excludes the time needed to retrain the agents at the end of each batch). This amount was chosen in order to represent a scenario in which there may only be a short term availability of access to a human expert. Therefore, as the amount of interactive training hours is fixed, it is logical to expect that the total number of episodes played may vary for each agent. This observation is noted for the trained agent results, as further discussed below. A common metric to analyze the performance of a trained machine learning agent is the learning curve, which in turn can be defined by a chosen figure of merit. This figure of merit can range from fuel consumption to time of flight. For this experiment, we are aiming to train a machine learning agent so that it is able plan and execute path planning strategies via use of impulsive maneuvers. At the core of the training architecture is the Asteroid Squadron simulator, where each episode represents a fixed mission scenario with regional exploration goals that contribute to the total score earned. These regional exploration goals represent subtasks for the agent. The total score is defined such that higher values indicate that the underlying subtasks are being better accomplished [39].

The learning curves for each of the trained agents (corrective and evaluative coaching with DAgger, original DAgger, and pure behavioral clone) are illustrated in Figure 6.12 as moving averages. For comparison, the learning curve (blue) for the human pilot is also illustrated in Figure 6.13, along with its polynomial approximation (orange). The human

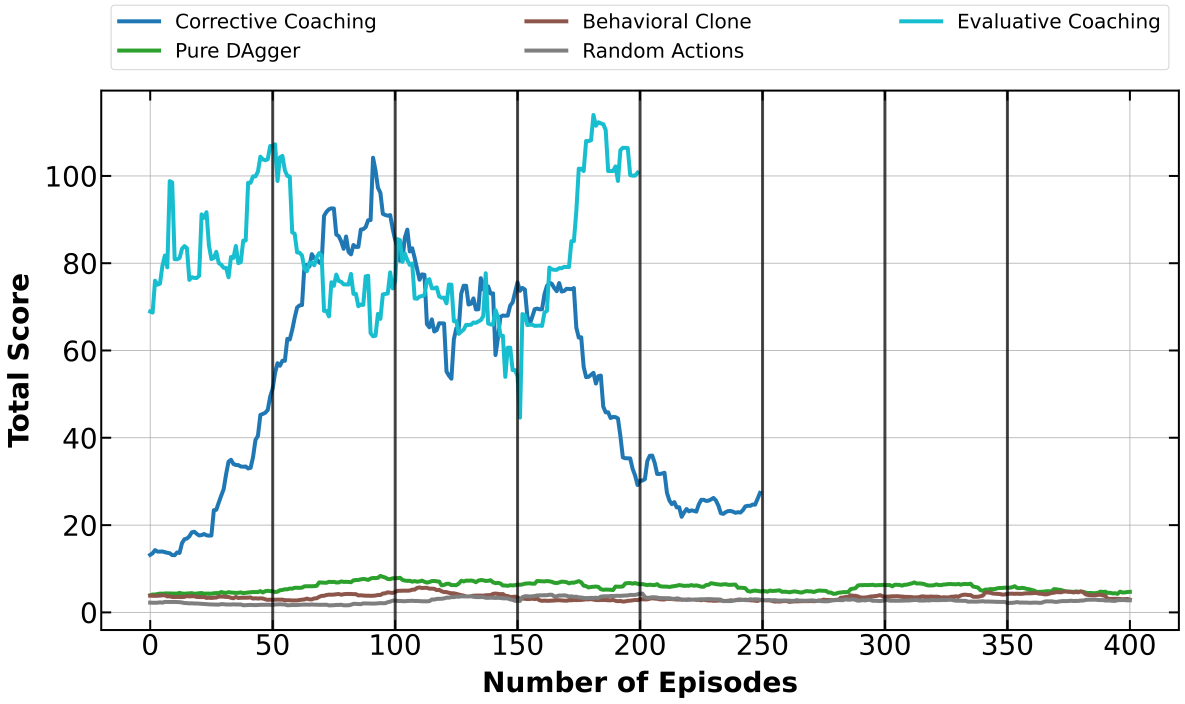


Figure 6.12: Learning curves of the agents represented via their moving average (window size = 50 episodes, for all agents, with end effects removed). The end of each batch, after which the learner agent was retrained, are visualized by vertical black lines.

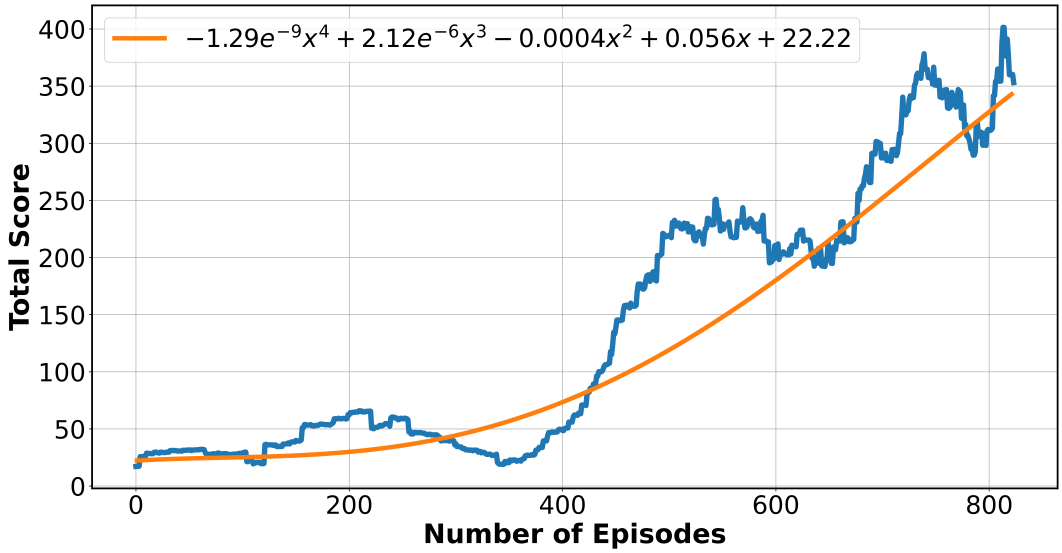


Figure 6.13: Learning curve for the human pilot, whose data was used to initialize the agents (moving average window size = 100 episodes, with end effects removed). The learning curve denotes the individual total score achieved by the human pilot in a given episode, and is not an explicit accumulation of the scores achieved in past episodes

learning curve for the human pilot in Figure 6.13 denotes the moving average of the total score achieved in each, individual episode the user played. Further analysis on the human's results and strategies is provided in [4] (the inclusion of the polynomial approximation is for visual purposes, to allow for a better relation between the human pilot's learning ability and the amount of simulation experience they have gained). It should be noted that within the designated 20 hours of interactive training time, each of the three DAgger agents completed a different amount of episodes, hence leading to a difference in the total number of episodes as a metric to measure the learning curves. Results from a random agent, which chooses actions with equal probability, are also included for comparison. Within the designated training time benchmark, the original DAgger agent completed 450 episodes, whereas both coached agents completed 150. Given the score trends and coaching experience during training, both coached agents were trained for an additional 20 hours; the corrective coaching agent completed another 150 episodes, while the evaluative coaching agent completed 100.

The random action agent should demonstrate the lowest learning performance, as it does not embody any control logic. Furthermore, as this agent's policy is not being updated during execution, we should expect that the learning curve remains relatively constant. From Figure 6.12, we see that this behavior holds; the random action agent generally demonstrated the lowest performance, as quantified by low total scores and a relatively constant learning curve.

All of the DAgger agents use this behavioral clone as the expert policy for the training algorithm. It is known that the use of an expert policy unable to capture the true logic of the human pilot, or oracle, can lead to unstable learning behaviors using DAgger. This can be observed in the original DAgger learning curve, which slightly increases during the initial phases of training before stagnating. As mentioned before, introducing coaching during training may improve learning. The corrective coaching learner demonstrates a sharp increase of the total score during the early learning phase, before the performance drops to lower total score values. This coached agent only received intervention from the human

monitor when one of the episode termination criteria was imminent. Conversely, the evaluative coaching learner seemingly provided the best learning trend. The learning curve slightly increased during the initial training batches, followed by a small decrease and subsequent, sharp increase. Recall that for this agent, the human monitor may arbitrarily impose control authority that may improve the learning progress.

6.4.2 Subscore Learning Curves

The learning curves illustrated in Figure 6.12 use the total score as the parameter to quantify the agent's learning ability. Recall that the total score is comprised of seven components, each representative of realistic mission goals during an asteroid exploration mission, with the scoring metrics previously detailed in Table 6.3. This scoring function therefore formulates the posed problem as a multi-objective task for the agent, which may increase the difficulty of achieving a converged machine learning solution. Consequently, assessing the agent's learning ability on all the underlying subtasks is critical in evaluating the total performance. In addition, understanding whether the subtasks are learned sequentially or in parallel may aid to interpret fluctuations in the total scoring function and inform the development of better training approaches. For example, during human training with the flight simulator, gaining experience in new sub-tasks negatively impacted the total score, displaying classical exploitation-exploration learning mechanics. Figure 6.14 illustrates the learning curves for each of the subscores for the three DAgger agents. The P1, P2, system north, system south, P1 north, and P1 south subscore labels respectively correspond to the s_2, s_3, s_4, s_5, s_6, and s_7 scoring conditions provided in Table 6.3.

Beginning with the original DAgger, we see that most of the subscore learning curves remain constant at low values. There are slight peaks and dips in the learning curves for the system north and south scores, indicating that the agent is going out of plane and possibly escaping the asteroid system, leading to a termination condition that prematurely ends the episode.

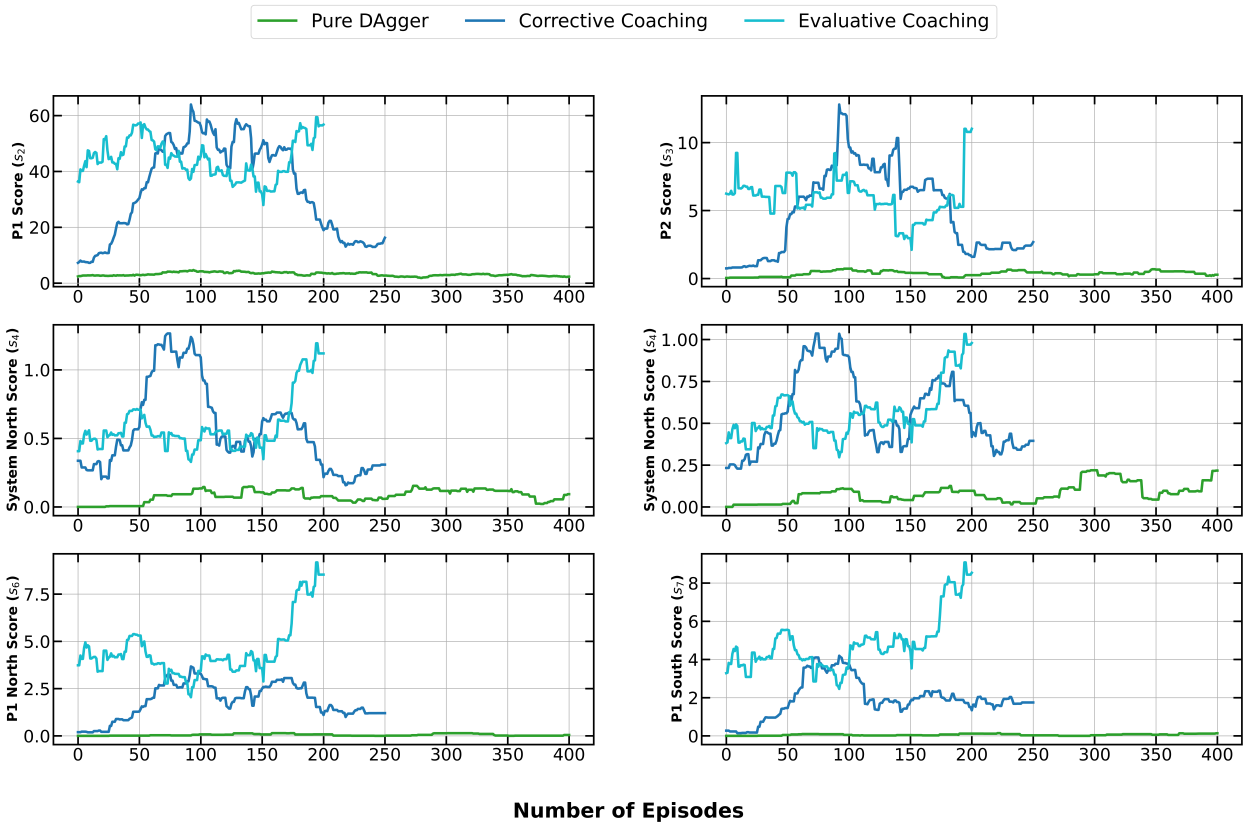


Figure 6.14: The learning curves for six of the seven component scoring function in Asteroid Squadron (moving average window size = 50 episodes, for all agents, with end effects removed).

The corrective coaching agent initially demonstrates an increase in learning behavior across all of the subscores, followed by a decrease. Upon further analysis, we observe that the learning curve for the P1 score most closely resembles the total score learning curve illustrated in Figure 6.12; this indicates that the learning for the corrective coaching agent is primarily influenced by an underlying logic aimed at path planning around P1, at altitudes at or near the mutual orbit plane of the asteroid system.

Analyzing the evaluative coaching agent leads to a slightly different insight into the learning style of the agent. From episodes 0-50, we see that all the subscore learning curves slightly increase or remain constant, except for that of the P2 score. This indicates that the agent's current path planning strategy is generally focused in regions around P1 rather than P2. From episodes 50-100, we observe a decrease in the P1, and P1 north and south scores

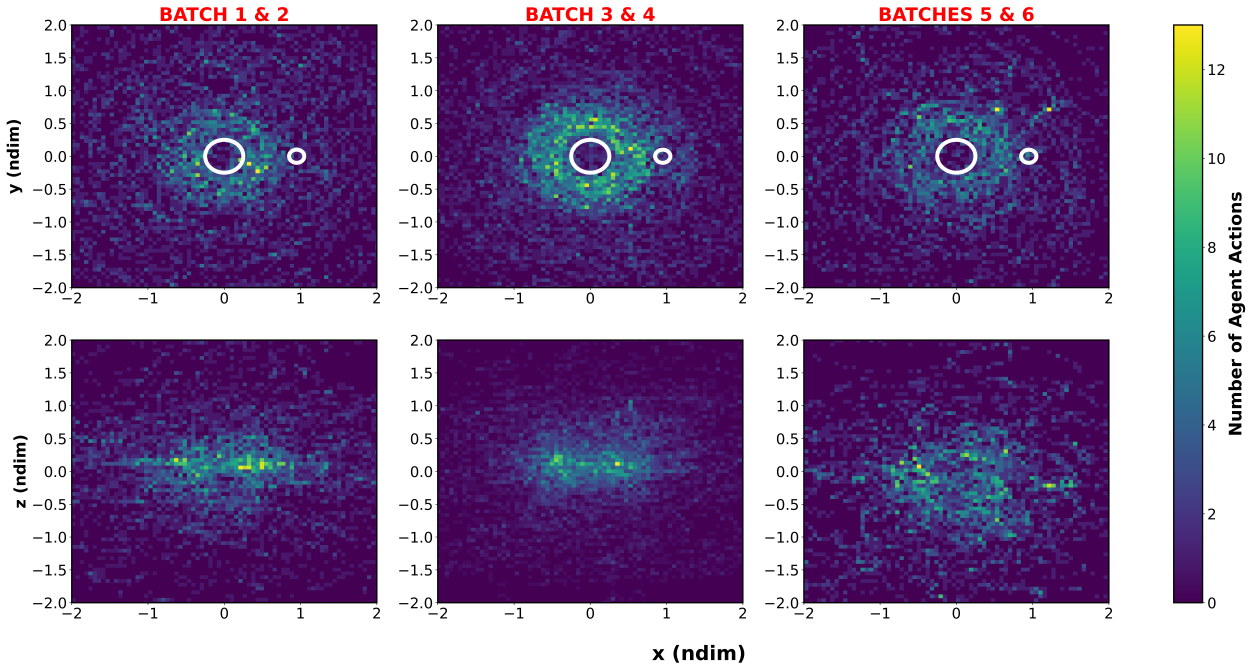
while noting an increase in the P2 score. This indicates that the agent's focus has slightly shifted to include orbital motion about P2. From episodes 100-150, we see a rise in the scores for the system north and south, as well as the P1 north and south subscores. Finally, from episodes 150-200, we see a sharp increase in all of the subscore learning curves, indicating that the agent is able to accomplish multiple regional exploration goals within an episode instead of solely focusing on one. This observation may also be an early indication that the current agent configuration learns the posed subtasks simultaneously, rather than leveraging an alternation of exploration-exploitation phases as the human pilot. While simultaneous multi-task learning is an appealing feature when interactive training time with a human expert is limited, a sequential approach to sub-task learning may yield overall better performance.

6.4.3 Maneuver Frequency Maps

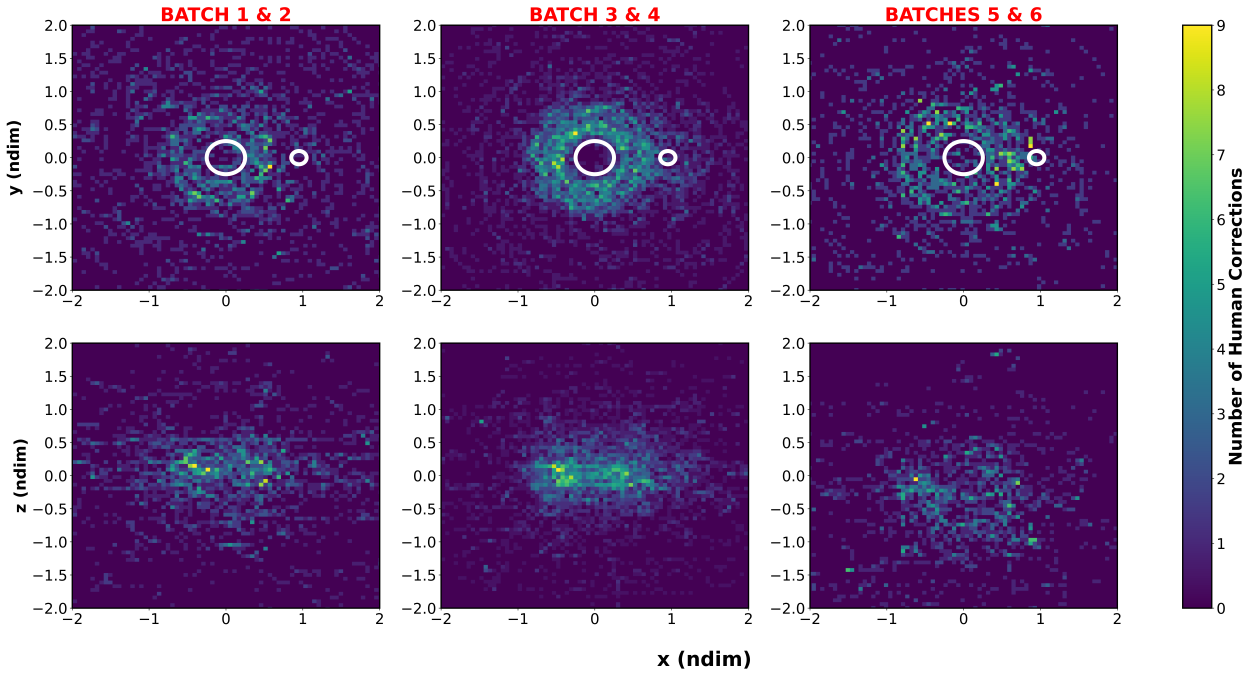
In each episode the asteroid environment is randomly generated by drawing parameters from uniform probability distributions (ranges are presented in Table 6.2). Large variations of orbit dynamics parameters result in a wide array of orbital environments that are dynamically different from each other. Therefore, the agent should not only learn a multi-objective task, but also develop general behavior (i.e. a path planning strategy learned for one type of binary asteroid system should transfer to other systems). One element enabling effective generalized policies is the identification of state features that minimize context variation across different asteroid environments. Recalling that orbit dynamics are nondimensional, nondimensional position states may offer an initial option to anchor learning to a context that is shared across different asteroid system. Then, one possible option to identify the emergence of generalized control heuristics is to visualize spatial distribution of control maneuvers as training progresses. Such a visualization may supply insight into where control actions are needed with an average representation that is agnostic relative to the specific asteroid system.

We currently choose to focus on the coached and evaluative learners due to their promising learning curve behaviors, recognizing that this analysis can be extended to all the agents trained. Figure 6.15 illustrates the spatial distribution of stationkeeping actions for the corrective coaching agent. Figure 6.15a represents actions taken by the agent itself, and Figure 6.15b represents actions demonstrated by the oracle during intervention. In the first 100 episodes (i.e. batches 1 and 2), the agent is executing maneuvering actions in the general vicinity of P1, at altitudes at or near the mutual orbit plane. The corrections provided by the human monitor are not specific to any region, instead spanning at altitudes both above and below the orbital plane, indicating that human intervention was required in order to stop the spacecraft from leaving the asteroid system by going too far out of plane. In the next 100 episodes (batches 3 and 4), we observe that the agent is primarily executing maneuvering actions around P1, both near the mutual orbital plane as well as out of plane. This orbital motion corresponds to the peak in subscore learning curves as observed in Figure 6.14 (recall that the learning curves are moving averages with a window size of 50 episodes). For this range of episodes, we observe that the majority of human corrections also remain in vicinity of P1, indicating that human intervention was required in order to avoid crashing into the asteroid. In the final 100 episodes (batches 5 and 6), the agent is not executing a clear policy that improves upon its gathered data. We observe that the agent's control actions have no primary focus and are instead scattered across the regions of the asteroid system. Human corrections are mainly centralized around P1, indicating that occasional intervention is required for crash avoidance. Furthermore, the human corrections are not near the mutual orbital plane, as observed in batched 3 and 4, and are similarly scattered across various out of plane altitudes. This observation indicates that the agent is frequenting states that are out of plane and require intervention in order to keep the spacecraft within the asteroid system. This is similar to what is observed in batches 1 and 2, and is consistent with the drop of the corrective coaching learning curve.

In comparison to the corrective coaching frequency maps, those corresponding to the

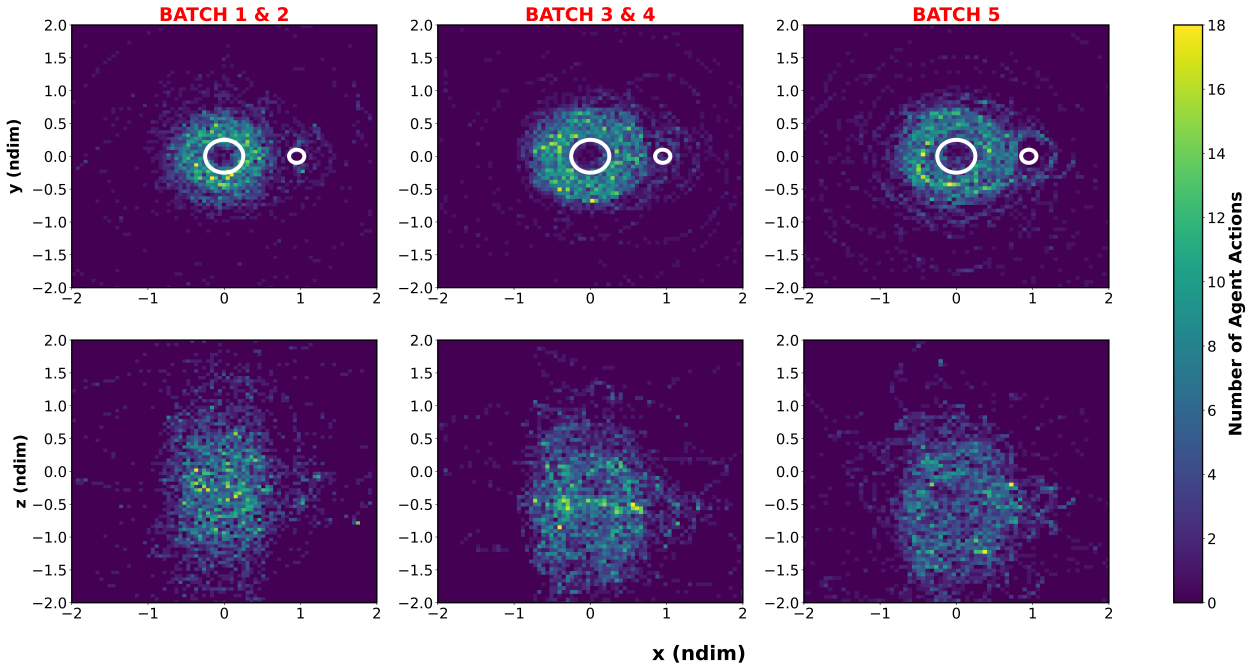


(a) Agent

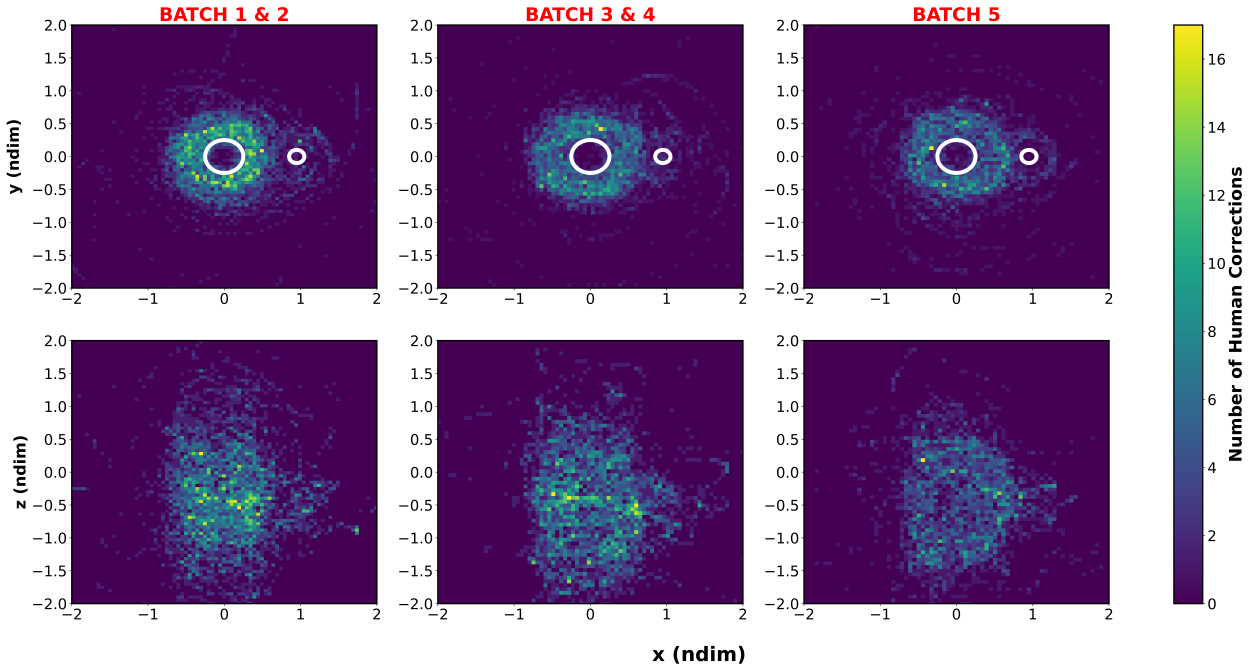


(b) Human Corrections

Figure 6.15: Frequency maps for the corrective coaching agent that represent the agent's actions and the human corrections. The general location of each of the asteroids is represented by the respective white circles, with the radii being that of the average diameter for each primary based on the asteroid systems that were encountered by the agent.



(a) Agent



(b) Human Corrections

Figure 6.16: Frequency maps for the evaluative coaching agent that represent the agent's actions and the human corrections. The general location of each of the asteroids is represented by the respective white circles, with the radii being that of the average diameter for each primary based on the asteroid systems that were encountered by the agent.

evaluative coaching provide a different insight. Recalling that the human monitor can arbitrarily intervene, we can visualize the evolution of the agent's learned maneuvering strategies in Figure 6.16, with Figure 6.16a representing the agent's actions and Figure 6.16b representing the human corrections. For the first 100 episodes (batches 1 and 2), we see that the agent is primarily executing maneuvers in the vicinity of P1. From the frequency map at the bottom left of Figures 6.16a and 6.16b, we observe an almost column like structure being formed by the spatial distribution of the actions, indicating that the human corrections are being used in order to get the agent back into a nominal orbital motion about P1. For the next 100 episodes (batches 3 and 4), we see that the agent is frequenting more states near P1, and the out of plane motion is more contained when compared to batches 1 and 2. Furthermore, we see that the amount of human corrections also decreased, and the out of plane corrections are more in the vicinity of the mutual orbital plane. We also begin to observe orbital motion about P2. For the final 50 episodes (batch 5), the agent is executing impulsive maneuvers around both asteroids, visiting states at lower altitudes of the mutual orbit plane when compared to batched 3 and 4. Human corrections rise compared to the previous two batches, particularly around P1. The rise in human corrections may be attributed to the fact that the agent is now experiencing unseen trajectories; in Figure 6.14, learning curves increasing for all subscores towards the end of the training period may point to new path planning behavior. Furthermore, the agent begins to orbit P2 more frequently. As orbiting P2 offers a much higher score than orbiting P1 for a given amount of time, the agent's current path planning strategy is resulting in novel trajectories, which may lead to incorrect action predictions and consequently require more corrections from the human monitor.

Comparing maneuvering frequency maps for both the coached agents and tracking the evolution in the path planning behavior may qualitatively indicate the level at which the agent is able to contextualize on the base of positional information. For example, from maneuvering frequency maps in Figure 6.15, we observe that the corrective coaching agent

initially recognizes the P1 vicinity as a location requiring more active control (i.e., higher maneuvering frequency). Quite interestingly, this learned contextual behavior becomes diluted as training progresses. In batches 5 and 6, the maneuvers become sporadic across the asteroid system, complementing the observed devolution in learning behavior. Conversely, the evaluative coaching agent display a more stable learning progression of contextual behavior based on position information. In Figure 6.16, the occurrence of actions across configuration space indicates that the agent first learn to maintain orbital motion nearby P1, for both out of plane and near mutual orbital plane trajectories, and then proceeds to learning how to maintain orbital motion about P2.

In conclusion, this analysis suggests that control policies reliant on contextual positional information may emerge from the early phases of learning and may be a heuristic acquired by the agent to operate across different binary asteroid systems. However, the occurrence of human monitor corrective actions with a spatial distribution similar to the agent actions may indicate that learned average behavior alone is insufficient to autonomously complete the task at hand. The investigation of contextual learning in terms of additional state features, such as ones that may be representative of the individual binary asteroid dynamics, is one of the possible next steps to improve agent performance. The development and inclusion of run-time safety assurance controllers based on the spatial distribution and frequency of human monitor actions is another direction that warrant further research. Furthermore, it is possible that run-time safety assurance controllers may ultimately replace the human monitor during training.

6.4.4 Agent Maneuvering Learning

From the previous section, we observed that the evaluative coaching agent may be more suited for contextualized learning, and from the respective frequency maps, we observed a decrease in human corrections at certain times. If the agent is truly learning maneuvering actions and requiring increasingly less intervention from the human, then the learning curves

of each control action should represent that.

Within the Asteroid Squadron simulator, there exist six control maneuvers that the agent can select from for stationkeeping. There also exists a seventh action, the null action. If maneuvering strategies are being learned then the agent should start executing more control actions autonomously, and executing them between periods of null actions in order to appropriately respond in a contextual manner. We can analyze the transfer of control authority from the human monitor to the agent by defining a specified quantity, the control authority ratio $r_{c.a.}$, as

$$r_{c.a.} = \frac{a_{maneuver}}{a_{agent} + a_{human}}$$

where $a_{maneuver}$ is the total number of specific control action we are looking at (either agent or human, but not both), a_{agent} is the total number of actions taken by the agent (including the null action), and a_{human} is the total number of corrections provided by the human (including intervals where the human agreed with the agent). Using this ratio, we can calculate the learning curves for each maneuvering action for a given agent. Again, for this analysis, we choose to focus on the coached and evaluative agents only, though the analysis can be extended to the original DAgger agent as well.

Figure 6.17 illustrates the learning curves for impulsive maneuvers for the corrective coaching agent (in black), as well as the human correction rates (in red). It should be noted that episodes with total scores less than 2 points were omitted as they represented extremely short duration episodes where the human corrections dominated the number of actions taken by the agent in order to avoid premature episode termination.

The first observation is that almost half of the episodes for this agent were omitted, which is seen by the total number of episodes in Figure 6.17 being significantly less than the total number of episodes from Figures 6.12 and 6.14. This indicates that the agent was not gathering good quality data. From Figure 6.17, we note that the control actions along the normal axis generally increased following a slight decrease. The other four control actions demonstrated oscillatory learning behavior for the agent. The -normal control action

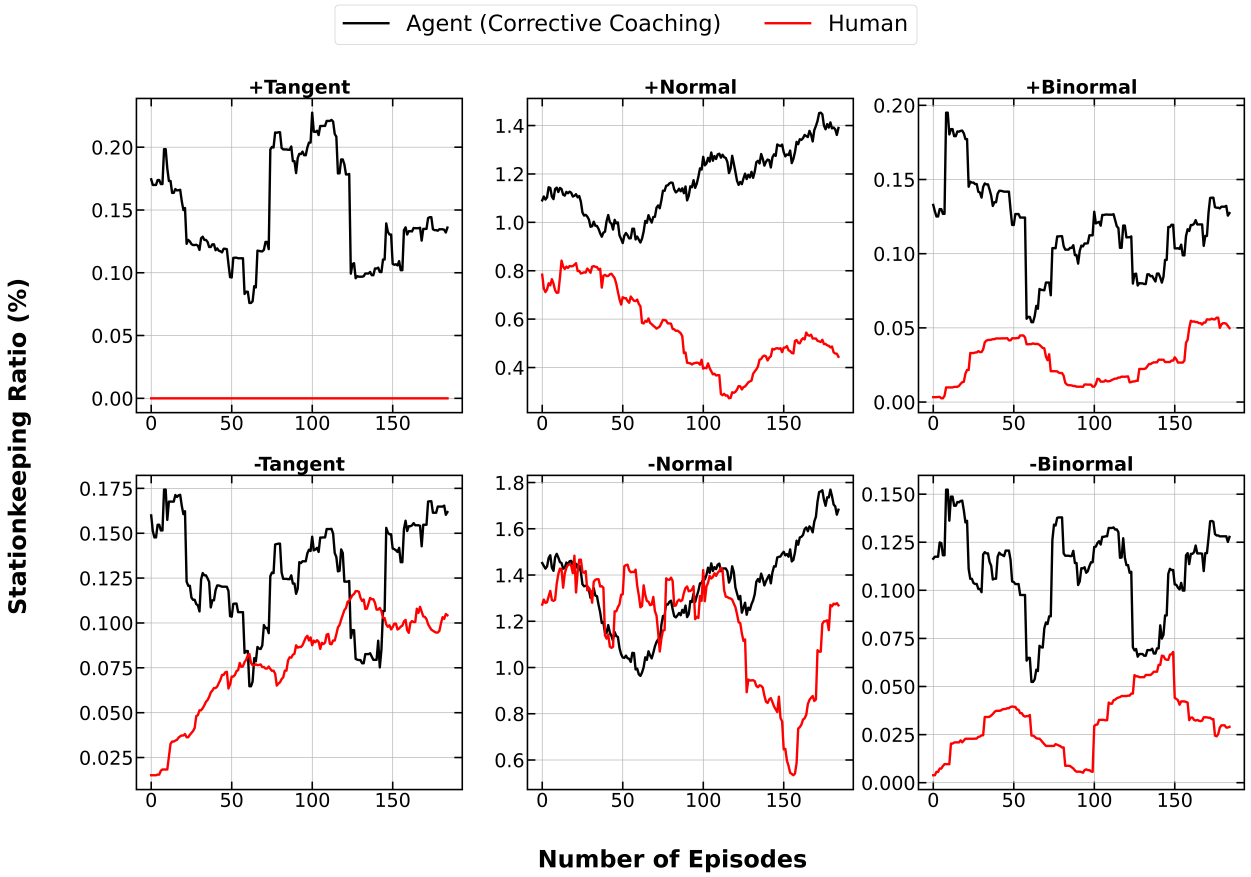


Figure 6.17: Learning curves for each of the six impulsive maneuver actions for the corrective coaching agent, as given by the control authority ratio $r_{c.a.}$ (moving average window size = 50 episodes)

remained constant, and near the same frequency as that for the agent, before showing any signs of decreasing. Additionally, the human corrections for the -tangent action, which reduces spacecraft velocity in order to facilitate temporary orbital capture, greatly increased over time even though the agent did not demonstrate learning behavior for this control maneuver. Note that the correction frequency of the human pilot for the +tangent action is flat as they did not designate this action for any of the corrections provided. The human corrections for +normal is the only one that generally dipped and the agent demonstrated a learning ability; though this is a promising result, the lack of this observation for the rest of the control actions further contributes to corrective coaching being a less suited training approach.

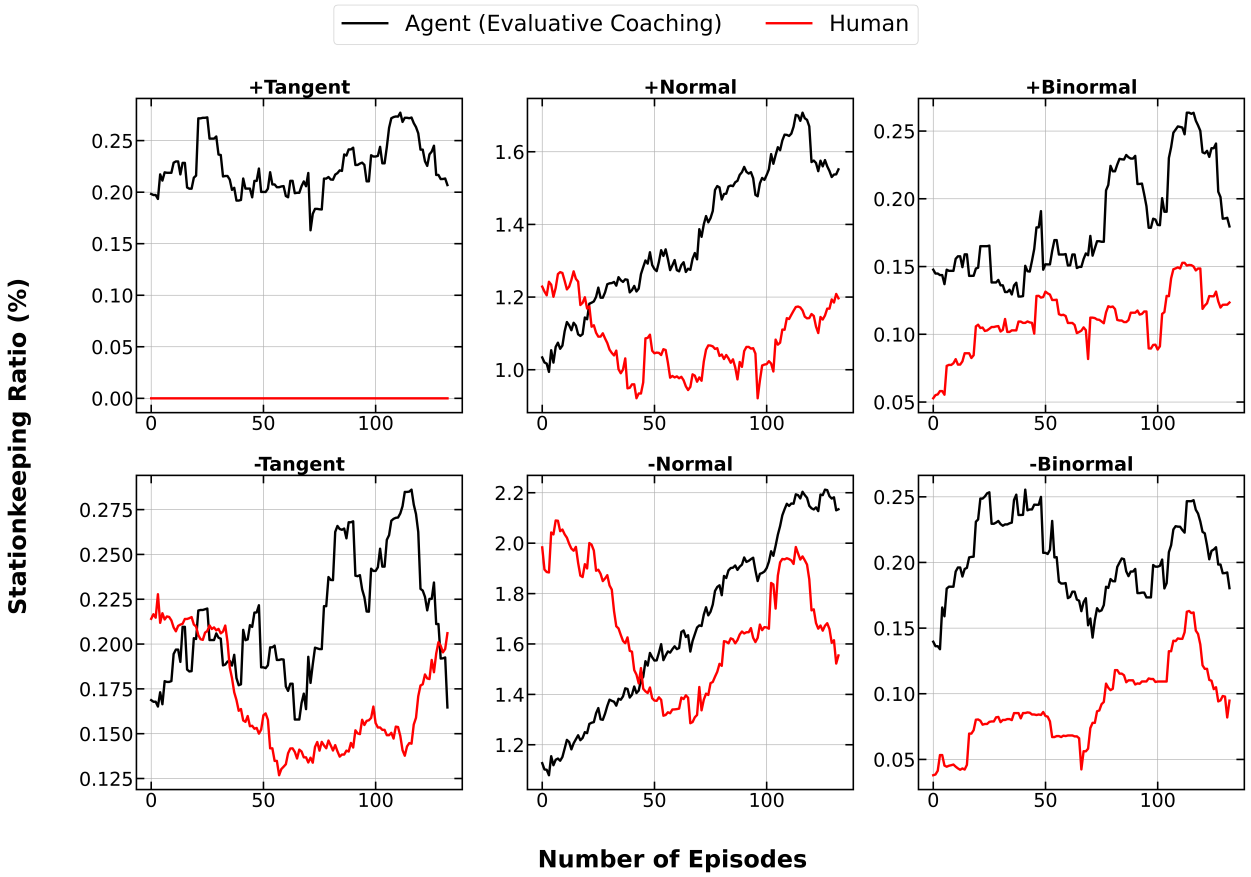


Figure 6.18: Learning curves for each of the six impulsive maneuver actions for the evaluative coaching agent, as given by the control authority ratio r_{sk} (moving average window size = 50 episodes)

We can compare these results to those of the evaluative coaching agent, as illustrated in Figure 6.18; again, episodes with a total score less than 2 were omitted. For this agent, the normal axis actions, as well as the negative tangent and positive binormal action learning curves are increasing. The rates of human corrections also initially decreased, before they begin to rise slightly again towards the end of the data set. This can be attributed to the prior observations in which the agent was found to be visiting new states, consequently resulting in unforeseen trajectories that prompt more frequent responses by the human monitor. Given the fact that the evaluative coaching agent displays increasing learning curves for a majority of the stationkeeping actions, this may indicate evaluative coaching as an appropriate training method.

6.4.5 Action Prediction Statistics

In this section we investigate how the human path planning logic is acquired by the behavioral clone. This analysis serves to further assess the quality and limitations of the current training framework. The action probability distribution by the agent at the end of each training batch, or benchmark, may provide a measure of similarity between the agent and the original human expert logic. Note that the action probability distribution is the final output of the neural network architecture in Figure 6.8, and comprises of 7 nodes. To investigate the similarity between the agent and the human expert, we can select a chosen trajectory and control strategy and provide the scenario as the input to each benchmark agent, we can observe the action probability distribution predicted by the agent. The similarity between this distribution and the action probability distribution of the original human expert may be an early indication of the similarity between the agent and general human logic.

Figure 6.19 illustrates the chosen control strategy for a trajectory generated by the human pilot, one that has not been experienced by the agent and was not included in the behavioral clone training dataset. The scenario details navigating the spacecraft through the narrow corridor region between both the asteroids. This region may prove difficult for a machine learning agent to navigate through, given the interacting dynamics of both asteroids, and human intervention may be needed. The strategy in Figure 6.19 has three control maneuvers, sequentially labeled in the manner they were executed in. For the agent benchmark analyses, the state at point 2 was chosen, the previous eleven states were derived in order to complete the required input matrix for the neural network, and the predicted actions of the agent were recorded. The probability distributions of each benchmark agent is given in Table 6.6 (recall that these probability distributions are the result of applying a Softmax activation function to the final layer in the feed forward neural network layers, as discussed in Section 5.1). Additionally, we note that the trained agents learns an average behavior, rather than a distinct strategy; therefore, in order to provide a more reasonable

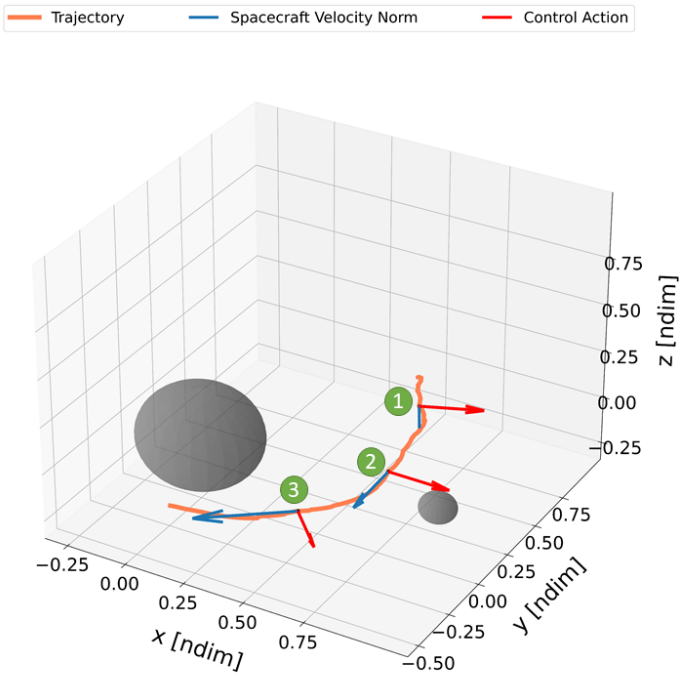


Figure 6.19: An example trajectory arc that visualizes a control strategy for navigating the corridor region between the two asteroids. The given data is from the original human pilot dataset.

basis for comparison, we examine the average human pilot action statistics in the region surrounding the chosen state at point 2. More specifically, the average human pilot actions within a spatial sphere, with origin at point 2, with a radius of 0.25 nondimensional length is considered. Furthermore, we only consider the human pilot actions in the last 125 episodes in the generated dataset, as this best represents the learned path planning strategy. These average action statistics are included in Table 6.6 as well.

The most probable action predicted by all benchmark agents is the null action. In fact, 90% of the human pilot's actions in the original dataset consist of the null action, and this bias was not removed prior to training. Additionally, we note that, on average in the region surrounding the chosen state, the null action constitutes over 93% of all actions. When considering the explicit control maneuvers, all benchmark agents correctly predict the negative normal action as the most probable. The specific probability distributions of the actions vary during training, and this may be attributed to the influx of new training

Table 6.6: The action probability distribution for scenario 2 in the reference trajectory illustrated in Figure 6.19.

	CONTROL MANEUVER PROBABILITY (%)						
AGENT	**Null**	**+ Tan**	**-Tan**	**+Norm**	**-Norm**	**+Binorm**	**-Binorm**
Human Pilot (state cell average)	93.46	0.02	0.13	2.66	3.69	0.02	0
Human Pilot (specified state)	0	0	0	0	100	0	0
Benchmark #1	85.71	0.40	0.55	5.47	6.93	0.53	0.41
Benchmark #2	86.10	0.61	0.92	4.93	6.51	0.65	0.57
Benchmark #3	83.40	0.67	0.83	5.56	7.94	0.74	0.87
Benchmark #4	84.22	0.69	0.98	5.65	6.98	0.76	0.72
Benchmark #5	85.02	0.67	0.79	5.13	7.06	0.69	0.65

data at the end of each batch. Though these results provide an early indication of the evaluative coaching agent's ability at predicting the control strategies of the human pilot, substantially more training time is needed for the agent to achieve performance that is on par with the human pilot. Furthermore, considering the strong effects of the null action bias in the original training dataset, a way to normalize these results may prove beneficial in future training experiments. Results in Table 6.6 may indicate an early ability of the agent to mimic the underlying logic of the human expert. However, the learned clone logic tends to average human behavior, introducing biases in the trained action probability distribution that may become critical when very contextual actions are required. The reduction of action probability biases that are induced by average behavior is an element that warrant further investigation to improve agent performance.

6.5 Preliminary Conclusions from this Investigation

The primary objective of this numerical experiment is to assess the feasibility, and understand the challenges, of implementing an imitation learning framework that leverages the cooperation of human and artificial intelligence to facilitate training of autonomous agents for complex and high dimensional path planning problems. The problem posed to the imitation learning agent incorporated multiple objectives within largely varying dynamical

environments. In this context, we analyze the performance of the trained autonomous agents via both quantitative and qualitative metrics, while identifying challenges across the learning process that derive from considering a complex path-planning task. Our work points to a few important conclusions and directions for future works that are elaborated in following subsections.

6.5.1 Bootstrapping autonomy with imitation for spacecraft path-planning

Our preliminary analysis offers promising early evidence that training of autonomous path-planning agents within chaotic dynamics with highly randomized parameters may be successfully bootstrapped with imitation learning techniques. While DAgger offers performance that is only minimally superior to a random policy, the introduction of a human monitor significantly aids the learning process, with a tenfold increase of the average total score as noted from the results in Figure 6.12. While this is not surprising due to the inclusion of a human monitor, it is notable that the frequency of intervention of the human monitor is significantly lower than an equivalent human pilot. That may imply that partial control authority is successfully transferring from the human to the machine. For the evaluative coaching agent, this conclusion may also be supported by observing that the number of agent actions relative to the number of human actions increased and that the spatial distribution of action frequency seems to converge toward a meaningful pattern.

Analyzing the explicit level of intervention provided by the human expert offers additional insight into the true ability of an agent to fly autonomously. In order to verify whether there is reduction of workload on the pilot, let us consider the highest scoring episode both of the coaching agents achieved: 1,593.83 for the evaluative coaching agent, and 1,588.03 for the corrective coaching agent. Of all actions taken for the evaluative coaching scenario, the agent chose the null action 97.7% of the time, and impulsive control maneuvers 2.3% of the time. In response, the human monitor agreed with 96.6% of the agent's actions. Comparatively, for the corrective coaching scenario, the agent chose the null action 99.2% of

the time, and impulsive control maneuvers for the remaining 0.8%. In response, the human monitor agreed with 98.86% of the agent's actions. These parametrics are further quantified in Table 6.7. In comparison, the action statistics of a sole human pilot, from the original human pilot dataset, on a similar episode with a total score of 1,585.03 are also provided.

Based on these metrics, the evaluative coaching agent may have learned a policy that has more frequent stationkeeping maneuvers as compared to the corrective coaching agent. Consequently, some of the control maneuvers chosen by the former may be incorrect, warranting a more intense corrective response from the human monitor, though the level of intervention required was just slightly higher as compared to the human pilot alone. Conversely, there was higher agreement between the agent and the human monitor for the corrective coaching agent, and there is a considerably lower level of stationkeeping intervention as compared to the human pilot alone. Although both agents achieved similar high scores, the variance in the orbital dynamics between these two episodes must be noted. It may be possible that, in a less chaotic dynamical regime, the corrective coaching agent provide better performance as less frequent stationkeeping may be required. Nevertheless, for both coached agents, the still evident performance instability remains an outstanding issue.

6.5.2 Average versus episodic performance: need for contextual learning

In our application, we observe large performance variations as a function of the underlying orbit dynamics. Moving average trends do not fully capture the complexity of such variations, such as the alternation of very high and very low scoring episodes, driven by the binary asteroid system that is encountered. For example, agents episodically achieve performance on par to the human pilot during coached training (e.g., episode score >1000, compared to a corresponding moving average of ~ 80 for the agent, and ~ 400 for the original human pilot).

These extreme variations were also observed during human pilot training. One key difference however is that the human pilot appears to better adapt to the more challenging

	NUMBER OF ACTIONS			
AGENT TYPE	**Null Action**	**Control Maneuver**	**Human - Agent Agreement**	**Total**
Human Pilot (episode score: 1,585.03)	46,967 (98.04%)	934 (1.97%)	-	47,901
Evaluative Coaching (episode score: 1,593.83)				
Agent	14,781 (97.7%)	345 (2.3%)	-	15,126
Human Expert	-	515 (3.4%)	14,611 (96.6%)	
Corrective Coaching (episode score: 1,588.03)				
Agent	28,141 (99.17%)	235 (0.83%)	-	28,376
Human Expert	-	321 (1.14%)	28,055 (98.86%)	

Table 6.7: Explicit level of intervention for the highest scoring episode for both coached agents. The average statistics for the human pilot from a similar episode are also provided as "Human Pilot Average".

dynamical environments. This may be an example of contextual behavior that is not being acquired by the current agent. For our purposes, we define contextual behavior as the adaptation of a general, learned policy in appropriate response to the observed spacecraft motion as a function of the orbital dynamics. Nonetheless the agent may be learning a general path planning policy, it may still be unable to independently achieve the adaptability and contextual maneuvering that is characteristic of a human pilot for this problem. This may further underscores the need for a human interactor during the formal training process, in order to impart the knowledge of contextual path planning when required. In addition, an agent architecture that is capable of learning maneuvering policy more contextual to the underlying orbit dynamics is also warranted.

6.5.3 Training time

From the comparison of the agent's learning curves and the human pilots learning curve, illustrated in Figures 6.12 and 6.13 respectively, we immediately note that currently, the agent is not on par with a human (currently our benchmark for best performance).

Rather, the performance of the agent is considerably lower. A factor contributing to the observed low performance may be the low number of updates for the coached learner agents during training. In particular, taking the evaluative coaching agent as a reference, the learner policy was only retrained 4 times, as the size of each batch collection was 50 episodes. The DAgger algorithm generally employs the use of smaller batch sizes to train a given agent. However, as our problem is composed of varying dynamical parameters for each episode, we opted for a larger batch size in order to provide the agent enough variance in its interactions with the game environment. This choice results in slower learning, and consequently warrants substantially more interactive training time. However, we must also recognize that the level of interaction required may be impractical for scenarios where the access to a human expert is limited. The use of smaller batch sizes may mitigate this problem, combined with more direct methods of domain knowledge transfer between the human expert and the agent. Transfer learning methods, such as domain adaptation and/or feature augmentation [98] , may allow us to map logical patterns from the human expert's source domain to the agent's target domain, and significantly decrease the amount of interactive training time required.

6.5.4 Influence of bias in the human expert's path planning logic

The original human pilot's datatset, which was utilized in order to train the behavioral clone, consists of a large bias in the dataset towards the null action. In other words, the human pilot's path planning strategy was predominantly coasting, with intermittent stationkeeping maneuvers when deemed necessary. As this bias was not removed prior to training the behavioral clone, it is possible that the large preference towards the null action dominated the learned policy; by extension, due to the stochastic policy mixing that is characteristic of the DAgger algorithm, the learner agent may also learn a policy that echoes this unmeditated bias. A more robust implementation of the current framework may warrant a mitigation of this bias in the human pilot's policy.

6.6 Initial Implementation of Recurrent Neural Networks (RNN)

Additionally, the underlying neural network architecture may also be a driving factor behind the agent's performance. In Section 5.1, we briefly discussed the problem of exploding and vanishing gradients in RNN's, and how a CNN with ReLU activation functions may be able to mitigate this phenomenon in an early agent implementation. However, we recognize that, regardless of vanishing gradients, RNN are specifically designed to solve sequence prediction problems, such may be a spacecraft path-planning task. Therefore, RNN are the next natural network architecture to consider in order of complexity. More specifically, let us consider how a long short-term memory (LSTM) network would perform in comparison to the implemented CNN architecture illustrated in Figure 6.8 at solving our imitation learning problem.

6.6.1 The Stateless LSTM

LSTM networks are specifically designed to provide capability of learning, and remembering, long term dependencies in time series data. This is done via use of hidden states and cell states. Hidden states are analogous to a working memory, or short term memory, and carry a representation of the immediately previous input state. Cell states are analogous to a long term memory that is capable of remembering beyond the immediately previous input state, and is updated via an algebraic calculation that is a function of the network's weights, biases and the hidden state.

Consider a single layer LSTM, with 120 features in the hidden state, followed by two fully connected, feed forward layers that have 100 and 7 nodes, respectively. Additionally, the ReLU activation function is used after the LSTM layer and the first feed forward hidden layer, whereas the Softmax activation function is used after the second hidden layer. The Softmax activation function in the second layer generates the probability distribution of each of the possible actions in the discrete action space for our problem.

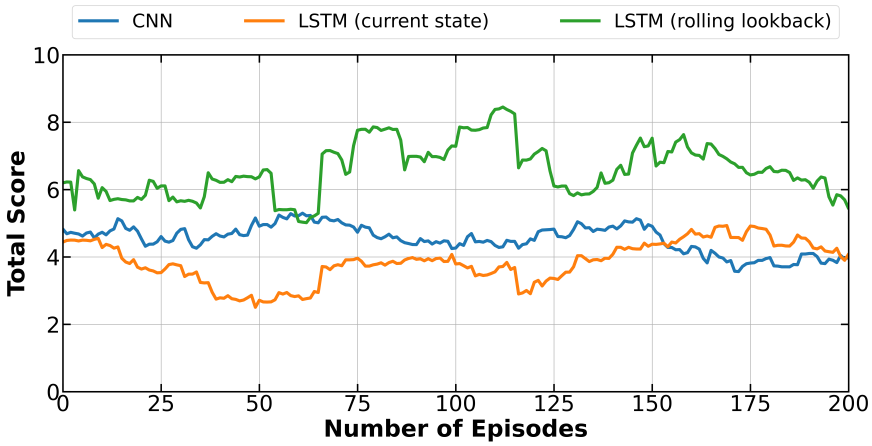
We must now make an important modification to our LSTM. In our scenario, we have

multiple, independent time series that the singular LSTM network must learn from. If we stack independent time series into a single data stream input, then a stateless LSTM implementation is required. As previously discussed in Chapter 3, the stateless LSTM implementation requires that we re-initialize the hidden and cell states to zero at the end of each individual time series. That prevents the learned temporal relationships in one dataset from influencing the learning from the next, individual dataset [99].

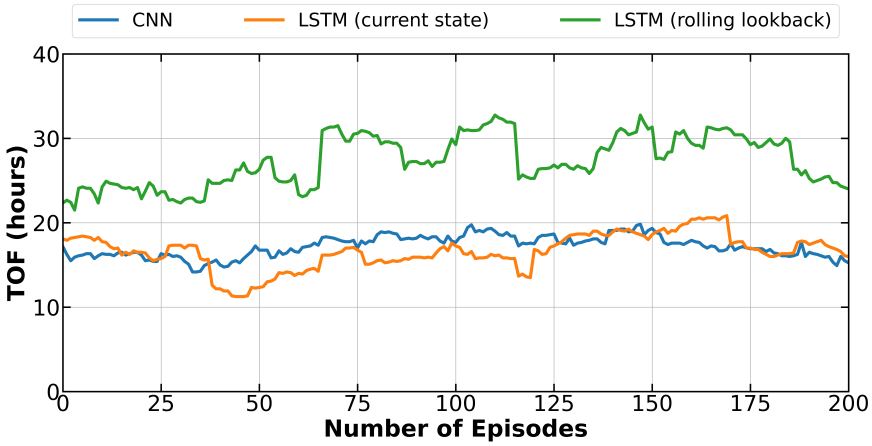
6.6.2 Stateless LSTM Implementation and Preliminary Results

Using the LSTM stateless network, two versions of behavioral clone were trained using the same training episodes that were utilized for the CNN behavioral clone. In the first version, we only provide the current state as the input to the LSTM network, whereas in the second version, we implement the same rolling lookback window that was used for the CNN architecture. Furthermore, the input to the LSTM network is the same 12 states that were utilized by the CNN architecture (i.e. the spacecraft position, velocity, total score, fuel remaining, the x positions of both asteroids, and the spacecraft's distance from each asteroid). The performance of each of these behavioral clones, in regards to the total score and the time of flight (TOF), are provided in Figure 6.20 as moving averages. We see that the performance of the CNN behavioral clone results in slightly higher total scores, but generally comparable times of flight, to the LSTM where we only provide the current state as the network input. In comparison, the LSTM with the rolling lookback as the network input results in the highest total score and time of flight, the latter in most cases exceeding one day of survival time. These results are indicative of an LSTM where we provide a rolling lookback window on the state histories as a more appropriate neural network architecture for this problem.

An initial implementation of the LSTM with rolling window lookback with the DAgger algorithm was trained for approximately 10 hours. This short investigation resulted in 104 completed training episodes, with each batch collection comprising of 50 episodes.



(a)



(b)

Figure 6.20: Performance metrics, provided by the moving average (window size = 50 episodes, end effects removed) for the CNN, and the two LSTM behavioral clones as quantified by (a) the total score and (b) the time of flight (TOF).

Preliminary learning parametrics of the total score, subscores, and time of flight (TOF) are illustrated in Figures 6.21, 6.22 and 6.23 respectively. We can note that after the first batch is complete, and the learner agent is trained, the LSTM network resulted in a considerably higher total score in the next batch. Also evident in the next batch is the corresponding increase in the subscores, indicating that the agent may be more receptive to the multi-objective behavior simultaneously, rather than focusing on singular subtasks as was noted for the evaluative coaching agent. Finally, we also note a rise in the time of flight during the second batch of episodes. Recalling that each of the episodes represents varying orbital

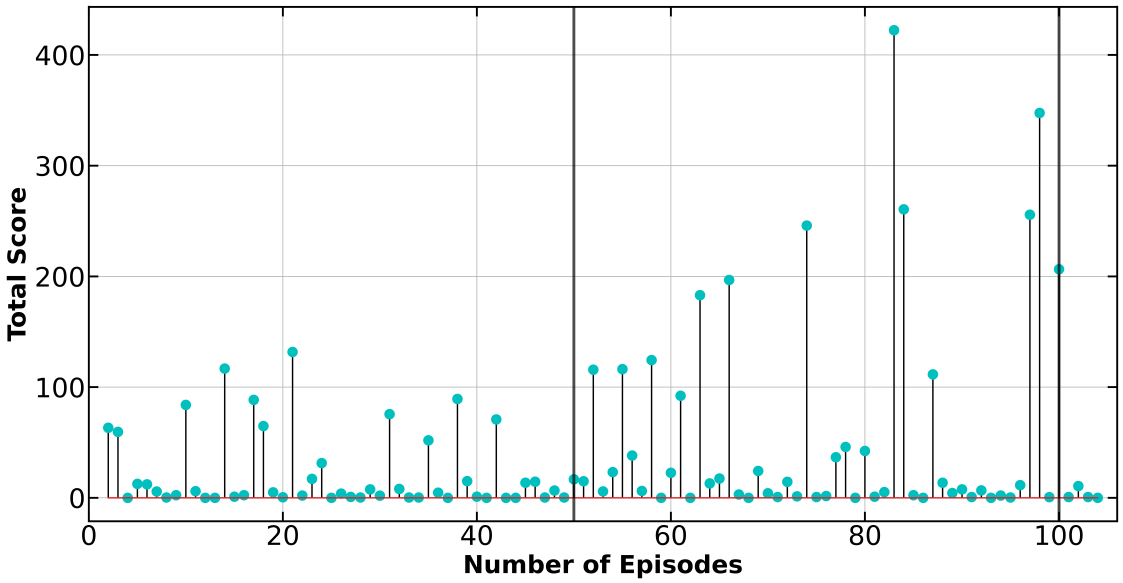


Figure 6.21: Learning curve, via total score, for the LSTM with rolling lookback neural network architecture. The end of each batch is visualized by vertical black lines.

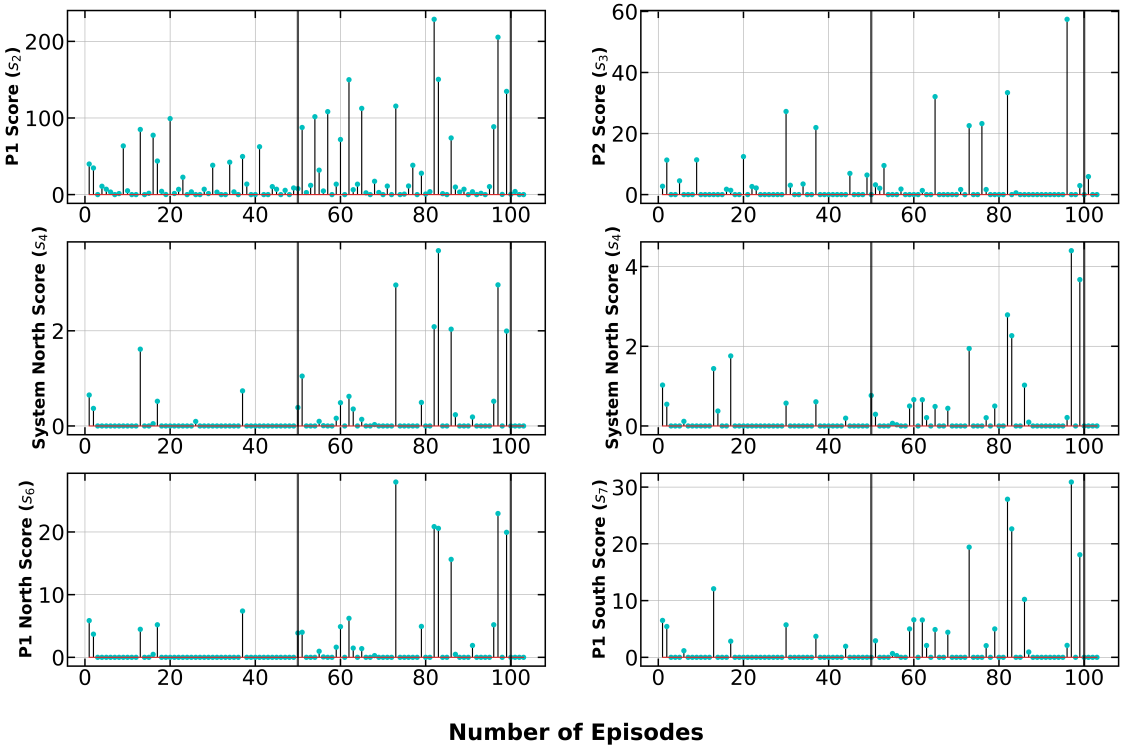


Figure 6.22: Learning curves for each individual subscore for the LSTM with rolling lookback neural network architecture. The end of each batch is visualized by vertical black lines.

dynamics, the general increase in performance across both scoring metrics and the sustained

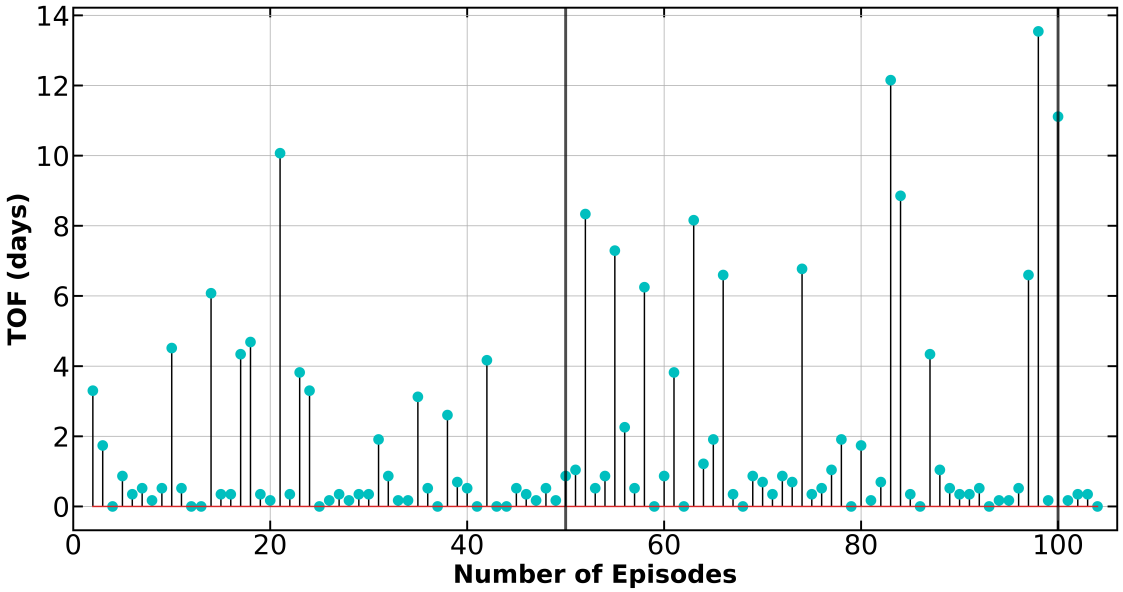


Figure 6.23: Time of flight (TOF) in hours for the LSTM with rolling lookback neural network architecture. The end of each batch is visualized by vertical black lines.

time of flight, this version of the LSTM network may be more appropriate than the CNN used in this investigation.

6.7 Identification of further challenges in ML path planning applications

The analyses conducted in this investigation resulted in preliminary identification of challenges in implementing an interactive imitation learning framework that combines human and artificial intelligence for path planning in complex environments. Specifically, future development directions that may be prioritized include:

1) Integration with transfer learning and interactive reinforcement learning: The training architecture may become more autonomous by merging the current imitation learning framework with reinforcement learning (RL) agents. The merging between the imitation and reinforcement learning frameworks is an instance of transfer learning. When transferring knowledge from expert demonstrations to a reinforcement learning agent, there may exist underlying differences in perception and task representation that inherently influence the learned path planning logic. Transfer learning methods are an active field of research

devoted to understanding the method of mapping policy knowledge from a source domain to the agent's target domain to more effectively combine different sources of intelligence into a singular policy. Furthermore, reinforcement learning agents may be trained interactively, similarly to the behavioral clones, in a technique known as interactive reinforcement learning.

2) Hierarchical learning to enable context-based actions and reduce training biases: The need for contextual path planning is required when the orbital dynamics are varying. This need, along with an additional reductions of bias in the source domain, may be addressed with a hierarchical learning architecture. For example, higher levels controllers may be responsible for the selection of a lower level path-planning policy based of the perceived orbit dynamics. Similarly, coast/thrust decisions may be delegated to a higher level controller, whereas the type of maneuver is determined by lower level of control.

3) Replacement of the human monitor with real-time flight assurance mechanisms: it may be possible to replace the human monitor in the corrective coaching framework with hard-coded logic triggered by safety thresholds. Furthermore, safety and real time assurance must be considered from a practical standpoint, especially when attempting to increase autonomy for a complex task.

4) Better policy approximation architecture and imperfect human demonstrations: Lastly, better policy approximation may be achieved via the implementation of a more advanced neural network architecture for sequential prediction problems. This may include the expansion of the LSTM formulation implemented in Section 7.5, with an investigation into stacked LSTM and optimized architectures in order to converge on a more robust training framework. Additionally, an expanded framework should address imperfect and sub-optimal human demonstrations to account for human errors. Imperfect demonstrations within a reinforcement learning setting may be accounted for via the use of a unified loss function that combined processes both offline and online demonstrations [100], and/or soft constraints during policy updates [101]. Together with this feature, future work should deliver insights into the minimal level and quality of demonstrations that an agent would need

before it is able to autonomously execute a policy within predefined safety and assurance thresholds.

Chapter 7

Conclusions and Future Work

This analyses presented in this work constituted an exploratory investigation of spacecraft path-planning methods that may be applicable to realize generalized and autonomous solutions. Though the standard approach at designing spacecraft path-planning solutions is driven by rigorous optimal control techniques, recent trends in published literature indicate an increasing interest in machine learning based methods as they may be able to provide more generalized solutions. However, the performance of these methods in relation to the standard approach has not been extensively explored, as most machine learning based applications for astrodynamics and spacecraft path-planning have modified the underlying algorithms and neural network architectures to be better applicable for specific scenarios. Removing these modifications, as done for the benchmarking investigation conducted in this work, allowed for the emergence of limits of applicability. It was observed that certain neural network architectures may only be applicable for a small range of dynamical scenarios, and some methods such as a simple PPO algorithm, explicitly fail to learn generalized solutions. These observations can be then utilized in order to construct a synergistic framework that is capable of addressing the inherent drawbacks of each distinct method, and is able to better achieve autonomous path-planning solutions. Furthermore, as discussed in Chapter 6, the extensive exploration of the novel method of human-AI cooperative learning highlighted the necessary modifications, and possible advances in state of art, that are required in order to leverage innate human logic as part of the machine learning training process in order to generate generalized and autonomous path-planning solutions.

Given the observations from both the benchmarking investigation as well as the exploratory work regarding cooperative human-AI learning method, there are multiple remaining points of investigation that can be further analysed in future work, some of which are summarized below

1. **Development of a synergistic path-planning framework**: as alluded to at the end of Chapter 5, the appropriate combination of the various path planning methods explored in this work may not only address the inherent drawbacks of distinctly utilized methods, but may also result in more autonomous solutions. However, the best structure of this synergistic framework is currently not known and would require additional investigations

2. **Continued investigation on the best implementation of human-in-the-loop training schemes**: the implementation of interactive DAggger evidenced an early learning ability of an artificial agent to learn path-planning logic in complex dynamics via human generated solutions. However, it was concluded that the current formulation of the learning framework would require a considerable amount of interactive training in order to achieve a converged solution. In this case, an alternate approach, such as one based on the cycle of autonomy as presented in [53] and previously discussed in Chapter 2, may be better suited.

3. **Real-time assurance of automated solutions**: lastly, as alluded to in the discussion of results in Chapter 6, the autonomous solutions must provide some level of real-time flight assurance, that ensures the generated path-planning strategy is within pre-defined safety thresholds. The exact manner in which this can be achieved is currently not definitively known, and warrants further investigations.

In conclusion, given the results of the work presented in this manuscript, there are numerous lines of investigations that can be conducted in order to address not only the noted challenges, but also the gaps in the existing state of art.

Bibliography

[1] Mazur, M., "A Step by Step Backpropagation Example," https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/, Accessed: 2021-10-17.

[2] Donges, N., "A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks," https://builtin.com/data-science/recurrent-neural-networks-and-lstm, Accessed: 2021-10-17.

[3] "Understanding LSTM Networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[4] Guzzetti, D. and Parmar, K., "Human Agent Path-Planning for Spacecraft Motion with Deterministic Chaos, Small Random Perturbations and Random Parameters," *International Astronautical Congress*, Washington, D.C., October 2019.

[5] Scheeres, D. J., "Close Proximity Operations for Implementing Mitigation Strategies," *2004 Planetary Defense Conference: Protecting Earth from Asteroids*, Orange County, California 2004.

[6] Swindle, T. and the SBAG Steering Committee, "Small Bodies Exploration in the Next 35 Years," *Planetary Science Vision 2050 Workshop*, Washington, D.C., February-March 2017.

[7] Ogawa, N., Terui, F., Yasuda, S., Matsushima, K., Masuda, T., Sano, J., Hihara, H., Matsuhisa, T., Danno, S., Yamada, M., Mimasu, Y., Yoshikawa, K., Ono, G., Yokota, Y., Saiki, T., and Tsuda, Y., 2020.

[8] Lu, P. and Liu, X., "Autonomous Trajectory Planning for Rendezvous and Proximity Operations by Conic Optimization," *Journal of Guidance, Control and Dynamics*, Vol. 36, No. 2, March 2013, pp. 375–389.

[9] Guzzetti, D., "Reinforcement Learning and Topology of Orbit Manifolds for Station-keeping of Unstable Symmetric Periodic Orbits," *AAS/AIAA Astrodynamics Specialist Conference*, Portland, Maine, August 2019.

[10] LaFarge, N. B., Miller, D., Howell, K. C., and Linares, R., "Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment," *Acta Astronautica*, Vol. 186, 2021, pp. 1–23.

[11] Miller, D., Englander, J. A., and Linares, R., "Interplanetary Low-Thrust Design Using Proximal Policy Optimization," 2019.

[12] Zavoli, A. and Federici, L., "Reinforcement Learning for Low-Thrust Trajectory Design of Interplanetary Missions," 08 2020.

[13] Sullivan, C. J. and Bosanac, N., "Using Reinforcement Learning to Design a Low-Thrust Approach into a Periodic Orbit in a Multi-Body System," 2020.

[14] Yanagida, K., Ozaki, N., and Funase, R., "Exploration of Long Time-of-Flight Three-Body Transfers Using Deep Reinforcement Learning," 01 2020.

[15] Scorsoglio, A., Furfaro, R., Linares, R., and Massari, M., "Actor-Critic Reinforcement Learning Approach to Relative Motion Guidance in Near-Rectilinear Orbit," 02 2019.

[16] Gaudet, B., Linares, R., and Furfaro, R., "SIX DEGREE-OF-FREEDOM HOVERING USING LIDAR ALTIMETRY VIA REINFORCEMENT LEARNING," 11 2019.

[17] LaFarge, N., Miller, D., Howell, K., and Linares, R., "Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits," 01 2020.

[18] Gaudet, B. and Furfaro, R., "Robust Spacecraft Hovering Near Small Bodies in Environments with Unknown Dynamics Using Reinforcement Learning," 08 2012.

[19] Yamaguchi, T., Saiki, T., Tanaka, S., Takei, Y., Okada, T., Takahashi, T., and Tsuda, Y., "Hayabusa2-Ryugu proximity operation planning and landing site selection," *Acta Astronautica*, Vol. 151, October 2018, pp. 217–227.

[20] Wibben, D. R., Williams, K. E., McAdams, J. V., Antreasian, P. G., Leonard, J. M., and Moreau, M. C., "Maneuver strategy for OSIRIS-REx proximity operations," *GNC 2017: 10th International ESA Conference on Guidance, Navigation and Control Systems*, Salzburg, Austria 2017.

[21] Jean, I., Mishra, A., and Ng, A., "Controlled Spacecraft Trajectories in the Context of a Mission to a Binary Asteroid System," *The Journal of Astronautical Sciences*, Vol. 68, 2021, pp. 38–70.

[22] Sánchez, J. P. and García Yárnoz, D., "Asteroid retrieval missions enabled by invariant manifold dynamics," *Acta Astronautica*, Vol. 127, 2016, pp. 667–677.

[23] Liu, X., McInnes, C., and Ceriotti, M., "Strategies to engineer the capture of a member of a binary asteroid pair using the planar parabolic restricted three-body problem," *Planetary and Space Science*, Vol. 161, 2018, pp. 5–25.

[24] Yang, H., Zeng, X., and Baoyin, H., "Feasible Region and Stability Analysis for Hovering around Elongated Asteroids with Low Thrust," *Research in Astronomy and Astrophysics*, Vol. 15, 09 2015.

[25] Yang, H. and Li, S., "Fast Homotopy Method for Binary Asteroid Landing Trajectory Optimization," 07 2019.

[26] Bellerose, J., Furfaro, R., and Cersosimo, D., "Sliding Guidance Techniques for Close Proximity Operations at Multiple Asteroid Systems," 08 2013.

[27] Bu, S., Li, S., and Yang, H., "Artificial Equilibrium Points in Binary Asteroid Systems with Continuous Low-Thrust," *Astrophysics and Space Science*, Vol. 362, 07 2017.

[28] Scheeres, D. J., "Close proximity and landing operations at small bodies," San Diego, California, July 1996.

[29] Izzo, D. and Ozturk, E., "Real-Time Optimal Guidance and Control for Interplanetary Transfers Using Deep Networks," 02 2020.

[30] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J., "Building machines that learn and think like people," *Behavioral and Brain Sciences*, Vol. 40, 2016.

[31] Miller, T., "Explanation in Artificial Intelligence: Insights from the Social Sciences," *Artif. Intell.*, Vol. 267, 2019, pp. 1–38.

[32] Singh, A., Thakur, N., and Sharma, A., "A review of supervised machine learning algorithms," *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 1310–1315.

[33] Ross, S., Gordon, G. J., and Bagnell, J. A., "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," *14th International Conference on Artificial Intelligence and Statistics*, Ft. Lauderdale, Florida, April 2011.

[34] Castelvecchi, D., "Can we open the black box of AI?" *Nature*, Vol. 538, October 2016, pp. 20–23.

[35] Peng, H. and Bai, X., "Artificial Neural Network–Based Machine Learning Approach to Improve Orbit Prediction Accuracy," *Journal of Spacecraft and Rockets*, Vol. 55, 06 2018, pp. 1–13.

[36] Sánchez-Sánchez, C. and Izzo, D., "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, 10 2016.

[37] Izzo, D., Martens, M., and Pan, B., "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, Vol. 3, No. 4, 2019, pp. 287–299.

[38] Rajeswaran, A., Kumar, V., Gupta, A., Schulman, J., Todorov, E., and Levine, S., "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," *ArXiv*, Vol. abs/1709.10087, 2018.

[39] Guzzetti, D. and Baoyin, H., "Human Path-Planning for Autonomous Spacecraft Guidance at Binary Asteroids," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 55, No. 6, February 2019, pp. 3126–3138.

[40] "Interactive imitation learning for spacecraft path-planning in binary asteroid systems," *Advances in Space Research*, Vol. 68, No. 4, 2021, pp. 1928–1951.

[41] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J., "Building machines that learn and think like people," *Behavioral and Brain Sciences*, Vol. 40, 2016.

[42] Pomerol, J.-C., "Artificial intelligence and human decision making," *European Journal of Operational Research*, Vol. 99, 03 1997, pp. 3–25.

[43] Sutton, R. S. and Barto, A. G., *Reinforcement learning: An Introduction*, The MIT Press, Cambridge, MA, 2018.

[44] Hovell, K. and Ulrich, S., "On Deep Reinforcement Learning for Spacecraft Guidance," 01 2020.

[45] Pérez-Dattari, R., Celemin, C., del Solar, J. R., and Kober, J., "Interactive Learning with Corrective Feedback for Policies based on Deep Neural Networks," *2018 International Symposium on Experimental Robotics*, Buenos Aires, Argentina, November 2018.

[46] Celemin, C. and del Solar, J. R., "An Interactive Framework for Learning Continuous Actions Policies Based on Corrective Feedback," *Journal of Intelligent Robotic Systems*, 2019, pp. 1–21.

[47] Argall, B. D., Browning, B., and Veloso, M., "Learning robot motion control with demonstration and advice-operators," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 399–404.

[48] Akrour, R., Schoenauer, M., and Sebag, M., "Preference-Based Policy Learning," *Machine Learning and Knowledge Discovery in Databases*, edited by D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 12–27.

[49] Knox, W. B. and Stone, P., "Interactively Shaping Agents via Human Reinforcement: The TAMER Framework," *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 9–16.

[50] Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D., "Deep Reinforcement Learning from Human Preferences," *ArXiv*, Vol. abs/1706.03741, 2017.

[51] Warnell, G., Waytowich, N. R., Lawhern, V., and Stone, P., "Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces," *ArXiv*, Vol. abs/1709.10163, 2018.

[52] Kelly, M., Sidrane, C., Driggs-Campbell, K., and Kochenderfer, M. J., "HG-DAgger: Interactive Imitation Learning with Human Experts," *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019, pp. 8077–8083.

[53] Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J., and Waytowich, N. R., "Efficiently Combining Human Demonstrations and Interventions for Safe Training of Autonomous Systems in Real-Time," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. abs/1810.11545, 2019.

[54] Curtis, H. D., *Orbital Mechanics for Engineering Students, 4th Edition*, Butterworth-Heinemann, 2004.

[55] Kirk, D. E., *Optimal Control Theory: An Introduction*, Prentice-Hall, 1970.

[56] Kelly, M., "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, Vol. 59, No. 4, 2017, pp. 849–904.

[57] Kingma, D. and Ba, J., "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 12 2014.

[58] "Exploding Gradients in Neural Networks," `https://machinelearningmastery.com/exploding-gradients-in-neural-networks/`, Accessed: 2021-01-08.

[59] Hochreiter, S. and Schmidhuber, J., "Long Short-term Memory," *Neural computation*, Vol. 9, 12 1997, pp. 1735–80.

[60] Rastogi, M., "Tutorial on LSTMs: A Computational Perspective," `https://towardsdatascience.com/tutorial-on-lstm-a-computational-perspective-f3417442c2cd#b10c`.

[61] "How to Choose an Activation Function for Deep Learning," `https://www.machinelearningmastery.com/choose-an-activation-function-for-deep-learning/`.

[62] Brownlee, J., "Introduction to Dimensionality Reduction for Machine Learning," `https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/`, Accessed: 2021-11-17.

[63] Hyvärinen, A. and Oja, E., "Independent Component Analysis: A Tutorial," `https://machinelearningmastery.com/exploding-gradients-in-neural-networks/`, Accessed: 2021-09-15.

[64] Hyvärinen, A. and Oja, E., "Independent component analysis: algorithms and applications," *Neural Networks*, Vol. 13, No. 4, 2000, pp. 411–430.

[65] Jaadi, Z., "A Step-by-Step Explanation of Principal Component Analysis (PCA)," `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`, Accessed: 2021-09-15.

[66] Brems, M., "A One-Stop Shop for Principal Component Analysis," `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`, Accessed: 2020-06-20.

[67] Mishra, S., Sarkar, U., Taraphder, S., Datta, S., Swain, D., Saikhom, R., Panda, S., and Laishram, M., "Principal Component Analysis," *International Journal of Livestock Research*, 01 2017, pp. 1.

[68] Bagheri, R., "Understanding Singular Value Decomposition and its Application in Data Science," https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388
Accessed: 2020-12-05.

[69] "Singular Value Decomposition (SVD) Tutorial," https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm, Accessed: 2020-12-05.

[70] "Singular Value Decomposition (SVD)," https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf, Accessed: 2021-01-11.

[71] Hui, J., "Machine Learning - Singular Value Decomposition (SVD) and Principal Component Analysis (PCA)," https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca
Accessed: 2021-11-17.

[72] Sharma, P., "A Beginner's Guide to Heirarchical Clustering and how to perform it in Python," https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/, Accessed: 2021-03-22.

[73] Garbade, M. J., "Understanding k-means Clustering in Machine Learning," https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1, Accessed: 2021-03-22.

[74] Singh, A., "Build Better and Accurate Clusters with Gaussian Mixture Models," https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/, Accessed: 2021-03-20.

[75] Chakure, A., "Decision Tree Classification: An Introduction to Decision Tree Classifier," https://medium.com/swlh/decision-tree-classification-de64fc4d5aac, Accessed: 2021-03-20.

[76] Gandhi, R., "Support Vector Machine - Introduction to Machine Learning Algorithms," https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47, Accessed: 2021-03-21.

[77] Srivastava, T., "Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python R)," https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, Accessed: 2021-03-21.

[78] Yiu, T., "Understanding Random Forest: How the Algorithm Works and Why is it so Effective," https://towardsdatascience.com/understanding-random-forest-58381e0602d2, Accessed: 2021-03-21.

[79] Gandhi, R., "Naive Bayes Classifier," https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c, Accessed: 2021-03-21.

[80] Szebehely, V. G., *Theory of Orbits: The Restricted Problem of Three Bodies*, Academic Press, New York and London, 1967.

[81] Lawden, D. F., *Optimal trajectories for space navigation*, Vol. 3, Butterworths, 1963.

[82] Yue, X., Yang, Y., and Geng, Z., "Continuous low-thrust time-optimal orbital maneuver," *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, IEEE, 2009, pp. 1457–1462.

[83] Taheri, E. and Junkins, J., "Generic Smoothing for Optimal Bang-Off-Bang Spacecraft Maneuvers," *Journal of Guidance, Control, and Dynamics*, Vol. 41, 09 2018, pp. 1–6.

[84] Nikoobin, A. and Moradi, M., "Indirect solution of optimal control problems with state variable inequality constraints: finite difference approximation," *Robotica*, 07 2015, pp. 1–23.

[85] Taheri, E. and Abdelkhalik, O., "Fast initial trajectory design for low-thrust restricted-three-body problems," *Journal of guidance, control, and dynamics*, Vol. 38, No. 11, 2015, pp. 2146–2160.

[86] Betts, J. T., *Practical methods for optimal control and estimation using nonlinear programming*, SIAM, 2010.

[87] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," 2017.

[88] Hiday, L. A., *Optimal Transfers Between Libration-Point Orbits in the Elliptic Restricted Three-Body Problem*, PhD dissertation, School of Aeronautics and Astronautics, Purdue University, August 1992.

[89] Short, C. R., Blazevski, D., Howell, K. C., and Haller, G., "Stretching in phase space and applications in general nonautonomous multi-body problems," *Celestial Mechanics and Dynamical Astronomy*, Vol. 122, July 2015, pp. 213–238.

[90] Santos, W. G., Prado, A. F. B. A., Oliveira, G. M. C., and Santos, L. B. T., "Analysis of impulsive maneuvers to keep orbits around the asteroid 2001SN263," *Astrophysics and Space Science*, Vol. 363, No. 1, 2017.

[91] Surovik, D. A. and Scheeres, D. J., "Adaptive Reachability Analysis to Achieve Mission Objectives in Strongly Non-Keplerian Systems," *Journal of Guidance, Control, and Dynamics*, Vol. 38, No. 3, 2015, pp. 468 — 477.

[92] Guzzetti, D. and Baoyin, H., "Human Path-Planning for Autonomous Spacecraft Guidance at Binary Asteroids," *IEEE Transactions on Aerospace and Electronic Systems*, February 2019.

[93] de Haan, P., Jayaraman, D., and Levine, S., "Causal Confusion in Imitation Learning," *Neural Information Processing Systems Workshop*, Vancouver, Canada, December 2019.

[94] He, H., III, H., and Eisner, J., "Imitation learning by coaching," *Advances in Neural Information Processing Systems*, Vol. 4, January 2012, pp. 3149–3157.

[95] Pascanu, R., Mikolov, T., and Bengio, Y., "On the difficulty of training recurrent neural networks," *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28, PMLR, Atlanta, Georgia, USA, 17–19 Jun 2013, pp. 1310–1318.

[96] Bai, S., Kolter, J., and Koltun, V., "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," March 2018.

[97] Mishkin, D., Sergievskiy, N., and Matas, J., "Systematic Evaluation of Convolution Neural Network Advances on the ImageNet," *Computer Vision and Image Understanding*, Vol. 161, May 2017.

[98] Kouw, W. and Loog, M., "An introduction to domain adaptation and transfer learning," 12 2018.

[99] Vlachas, P., Byeon, W., Wan, Z., Sapsis, T., and Koumoutsakos, P., "Data-Driven Forecasting of High-Dimensional Chaotic Systems with Long-Short Term Memory Networks," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, Vol. 474, February 2018.

[100] Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T., "Reinforcement Learning from Imperfect Demonstrations," 2019.

[101] Jing, M., Ma, X., Huang, W., Sun, F., Yang, C., Fang, B., and Liu, H., "Reinforcement Learning from Imperfect Demonstrations under Soft Expert Guidance," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 04 2020, pp. 5109–5116.