PILOT DEMONSTRATION BASED REINFORCEMENT LEARNING WITH APPLICATION TO LOW SPEED AIRSHIP CONTROL

by

ONUR DAŞKIRAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2016

To Rogeye, who gave up her own PhD to raise me.

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge Dr. Atilla Dogan for his guidance and support throughout this dissertation. His supervision enabled me to acquire numerous skills and gain different points of view to attack the challenges experienced during research. Dr. Brian Huff has also been an excellent mentor. His practical experience and smart solutions have helped and supported me greatly in many activities I participated in UTA. My thanks extend to respected committee members for their feedback and constructive criticism.

I wish to extend my gratitude to Dr. Frank Lewis and Dr. Luca Maddalena for providing significant resources for the experiments. Raghavendra Sriram and Raymond Correra provided great help for the tests. Sampath Reddy Vengate contributed to this work remarkably not only as an expert RC pilot but also as a sincere friend.

I also sincerely appreciate the MAE administrative staff, many friends in AVL, CACSD labs and UTA Turkish community for always reaching out in the time of need. I am grateful to all my past teachers and professors for all of the effort they have put into me.

My loving parents, brothers and fiancee have been intensely supportive and understanding even though I was not with them at times when they needed me around the most. I'm forever deeply indebted to them.

December 15, 2016

ABSTRACT

PILOT DEMONSTRATION BASED REINFORCEMENT LEARNING WITH APPLICATION TO LOW SPEED AIRSHIP CONTROL

ONUR DAŞKIRAN, Ph.D.

The University of Texas at Arlington, 2016

Supervising Professors: Atilla Dogan, Brian Huff

Designing control systems for airship has unique challenges as compared to conventional aircraft. Highly nonlinear dynamics, different mass/inertia relations, vast uncertainties in the model parameters and underactuation are the main reasons behind this. Airship dynamics is greatly influenced by the variations in the environmental (e.g., *room temperature*) and internal (e.g.,*helium distribution in envelope*) factors that can completely change the response characteristics of the blimp and make it infeasible for a model-based controller to perform. On the other hand, a skilled RC pilot can operate the manual flight easily under these conditions. This makes LfD (learning from demonstration) and RL (reinforcement learning) techniques suitable candidates to address the issues that model-based control design fails to do. In general, LfD covers the methods that aim to learn a control policy directly from the previously provided expert demonstrations. In reinforcement learning, it is aimed to reach an optimal policy through trial and error while a reward function continuously describes whether the action taken in a specific state creates good or bad outcome.

This dissertation research develops a three stage LfD/RL method which uses continuous multi-dimensional states and actions. Stages and subroutines used in the method is first explained in detail, then implemented on three simple example cases to show the performance and the convergence characteristics of exploration using discrete and continuous state-action spaces. The method is used for learning and executing 1D and 2D waypoint navigation tasks of a ground vehicle (UGV) for both simulation and hardware implementation. In order to apply the method to the motion of a low speed airship, a realistic airship flight simulator is designed by performing measurements and tests and pilot demonstrations are recorded with this simulator. Finally, the method used to learn and execute commanded position and orientation tasks demonstrated by the pilot, similar undemonstrated tasks and a case when these tasks are combined to represent a full mission. It is shown that selection of correct function approximator parameters are crucial in order to obtain satisfactory response when LfD/RL method is used.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

This chapter aims to declare the problem of low speed airship control and propose a solution using Learning from Demonstration (LfD) and Reinforcement Learning (RL) techniques. Previous studies conducted in this subject are addressed under several titles in the literature survey section. Finally, the original contribution of this research effort is presented in the last section.

1.1 Problem Statement

Airship are deprived of the control authority of the aerodynamic control surfaces when operated at low speeds. This causes number of the available control effectors to become much less than the number of the degrees of freedom, which puts the vehicle in the category of *underactuated* systems. Unlike fully-actuated systems, underactuated systems cannot follow commanded arbitrary trajectories [1]. Control problems of underactuated mechanical systems are usually addressed with fundamental nonlinear approaches, as the linear approximations around the equilibria are usually not controllable [2]. In order to propose control solutions, these methods require extensive analysis of the system equations and the constraints they are subjected to. On the other hand, skilled human pilots can easily operate these systems to fulfill desired tasks without the need of a detailed study of the system, using solely their previous experiences, even though they have not operated that particular vehicle before. Such facts bring on the idea of using expert demonstrations for the purpose of controlling this class of vehicles.

Apart from the difficulties due to underactuation, airships possess many distinctive features than the conventional air vehicles, which present additional challenges for control design. They are highly prone to changes in the environment such as ambient temperature and even very slow wind. Since buoyancy of the gas is the main source of the lift, small variations in the buoyancy-weight balance might completely change the optimal trajectories of the tasks desired to be performed. Some of the control effectors on airship such as tail thruster introduce more challenges to the control design. In the usual configuration, tail thruster is driven by a single DC motor which can rotate both directions. However, due to the motor and ESC (Electronic Speed Controller) dynamics, when the thruster sign changes, the DC motor momentarily goes to a full stop. While RC pilots can easily deal with this deadband issue, it would be very difficult for control design. In addition, the slow response time characteristic of a blimp requires to adopt a predictive control philosophy rather than a reactive one, because of the delay between the time control action taken and the time the effect of that action is seen on the response of the system. The inherent ability of pilots with experience to predict the motion of the blimp and take action beforehand is the type of skills which are wanted to be transferred to automatic control system using LfD/RL approaches.

The overall objective of this research is to deploy the LfD/RL methods on an underactuated indoor blimp with multiple effectors while the expert is a skilled human RC pilot. Reinforcement Learning has been applied to airship control tasks by constraining the motion into single degree of freedom and using a single control effector as in the yaw control case of [3] or altitude control case of [4]. However, usual tasks expected from unmanned vehicles would require moving within the all degrees of freedom available. For this reason, optimal policy, the mapping between states and actions searched should be a vector of multiple control actions. Continuous state

and action spaces are used to store the expert demonstrations and locally weighted regression technique is used as the function approximator.

For robotic tasks, the state and the action spaces are usually large. This creates several problems in employing RL techniques alone to learn optimal control policies. One of these problems is the duration of the learning. Time spent to try out all state-action pairs would become very long. Another problem is the safety of the learning mission. Random actions taken during the learning phase might cause irreversible consequences for a robotic platform. In this study, a combination of LfD and RL is proposed to overcome these challenges using a three stage framework.

1.2 Literature Survey

Both LfD and RL methods have been widely studied to solve robotics challenges [5, 6]. This section presents the literature review effort in various aspects including whether the state/action space is continuous or discrete, methods of dataset sparsification, application areas, action selection methods, discussion on reward function and policy improvement.

1.2.1 Continuity of State and Action Spaces

Reinforcement Learning (RL) problems are posed in the MDP (Markov Decision Process) setting. For this reason, the learning algorithms make use of discrete, and mostly finite, set of states and actions. However, most robotic challenges inherently possess continuous states and actions. In order to employ RL methods, either the state-action space is discretized or continuous space is used by introducing function approximation methods.

(i) Discretized State-Action Spaces

Many learning algorithms assumes a representation of the system in terms of

the finite number of discrete states and actions where the expected sum of future rewards, called "utility values" of state-action pairs are kept in a table and can easily be queried. On the other hand, real world robotic applications inherently possess continuous states and actions which require to be discretized according to the problem definition to be used with existing algorithms. Mobile robot navigation, path planning and obstacle avoidance tasks which use learning for guidance purposes have employed discrete/discretized states and actions [7, 8, 9, 10, 11]. Reducing the problem by limiting the number of states and actions such as blimp yaw control case in [3] has shown to yield good results by discretized state and actions. One technique to avoid large number of grid points is to use variable adaptive grid size in the state-action space [12, 13]. A comprehensive multi-resolution state-space discretization study [14] offers first determining the regions of interest in the state-action space, then applying a fine discretization for these areas only while keeping the rest coarsely discretized. A similar approach is to use regression trees that can discretize continuous state space while leaving equivalent areas as single states [15]. Main drawback of discretized state-action spaces comes from the difficulty in determining of an appropriate resolution for discretization. Using a coarse discretization might cause an unrealistic representation of the actual system and introduce hidden states. In order to have a realistic discretization, the state-action space must be partitioned into very large number of points which will in return increase the memory and computational power requirement to process such large datasets. For this reason, nature of the robotic challenge must be suitable to be addressed by discretized state and actions. For the remaining cases, either elaborate techniques for discretization should be devised or function approximation methods should be employed.

(ii) Continuous State-Action Spaces

When continuous state-action space is used, utility values can no longer be easily queried from a database. Instead, a reasonable utility value must be estimated using the previously recorded state-action pairs and their respective utility values. An earlier function approximation method which has been widely used in the literature is Cerebella Model Articulation Controllers (CMAC) which groups the features, inputs of the function to be approximated, into exhaustive partitions of input space and represent the output as the weighted sum of the features activated by the inputs [16, 17, 18]. In basic terms, all function approximators map the features of the state-action space into utility values. These mappings can be evaluated by several methods such as linear basis functions [19, 20, 21, 22], neural networks [23, 24, 25, 26, 27], Gaussian process regression (GPR) [28, 4, 29, 30, 31] or local regression along the nearest neighbors [32, 33, 34, 35, 36]. While using continuous state and action spaces eliminates the problems due to discretization, they introduce additional setbacks. Methods such as neural networks require many training samples before making plausible generalizations and they suffer from destructive interference which means the newly acquired knowledge destroys the generalization about the previous knowledge. Instance based function approximation methods store all the data samples to avoid destructive interference. As a consequence, problems such as limited memory space and computational power arise, as the dataset size continuously grows. In addition, when function approximation methods are used, action selection becomes an optimization problem which gets more difficult to solve as the dimensions of the action space increases. Thus, selection of high dimensional actions in a fast and accurate way is an important area for improvement.

1.2.2 Dataset Sparsification

Instance based function approximation methods such as Gaussian process regression or Locally weighted regression keep the learning instances in a dataset and use all of its entries while estimating the value functions. As the dataset gets larger, memory space and computation power become important issues. These issues can be alleviated by using smarter data structures such as kd-trees [37, 4]. However, considering the fact that tree structures also have a limited memory, sparsifying the dataset by deleting redundant entries becomes an important task.

(i) Online sparsification

In this class of methods, the main approach is to decide on either accepting or rejecting the new data sample according to a threshold, as soon as the data arrives. This threshold can be evaluated from the predictive variance between the samples [28], linear independence of the incoming sample [38] or local validity [39]. Another method assigns dynamic weights to both old and new dataset entries based on their usefulness [40]. Weights are updated at every step and the data entries with weights below a threshold are deleted.

(ii) Batch sparsification

These methods aims to create a smaller size dataset while keeping the prediction quality still sufficiently well enough. The new dataset can be a direct subset of the main dataset [41] or a completely different set which has the same statistical properties with the main dataset [42]. However, these methods are computationally expensive to perform and not preferable for real time robotic missions.

1.2.3 Application Areas

There are various applications areas where LfD and/or RL are utilized. This section groups the prior work based on these application areas.

(i) Robotic Manipulators

Some of the earliest studies on LfD/RL have been performed on robotic manipulators. Expert demonstrations are used to train neural nets for peg-in-hole task [43]. Hidden Markov Models combined by k-nearest neighbors method are applied to object grasping and egg flipping tasks [44]. Using stereo vision, expert trajectory is captured and pole balancing task is taught [32]. A regression based solution is proposed for part assembly task under suboptimal teaching [45]. Expert grasping object demonstrations are used for trajectory learning [46]. Natural actor critic method is applied to baseball bat swing task [47].

(ii) Humanoid, Legged Robot

As the main purpose of humanoids is to think and act like humans, LfD finds one of the most appropriate applications in this area. Walking [48, 49, 50], playing sports [35, 34], playing instruments [51], performing torso movements and dexterous skills such as ball sorting and pouring [52, 53, 54, 55] are some of the successful implementations with humanoids.

(iii) Unmanned Vehicles

(a) *Ground Vehicles:* Overall, the most preferred vehicle platform type for the Reinforcement Learning applications is ground vehicles due to their easy operability and less complex dynamics. On ground vehicles, indoor navigation and obstacle avoidance are implemented in numerous cases [33, 7, 8, 17, 9, 10, 11, 56], whereas, learning driving skills and lane following is implemented on real size cars [57, 26, 58, 59].

(b) *Surface/Underwater vehicles:* Learning based algorithms are attempted to generate solutions for the challenges of unmanned surface/underwater vehicles. One challenge is to take care of underactuated and noisy dynamics. A direct policy search method using partially observable states is suggested for a high level control action selection task [60]. Motion planning under tough operating conditions is another area to employ reinforcement learning. A multilayered Q-learning, a model free reinforcement learning method, based algorithm for motion planning under strong sea currents [61] as well as an artificial neuron based model for navigating in the unknown non-uniform sea flow [62] are proposed. An adaptive obstacle avoidance algorithm using on-policy RL technique, in which the learned policy is same with the current policy-in-use, is proposed for unmanned surface vehicle [63].

(c) *Helicopters:* Unmanned helicopters by far are the most skill requiring vehicles to operate due to their high speed motion and high sensitivity to the accuracy and timing of the control inputs. Such features make them challenging yet encouraging candidates to be employed as testbeds for LfD/RL algorithms. One of the earliest applications [64] uses neural networks to approximate the human controller's ability within specified bounds. A partially observable policy search method is introduced in [65]. Most fulfilling results are obtained for the control of helicopters in apprenticeship learning setting, either learning the reward function [66] or the intended trajectory [67]. A recent model based reinforcement learning algorithm using GPR (Gaussian Process Regression) is shown to be learning control trajectories in a data efficient way [68].

(d) *Airship:* Indoor airship are employed in several reinforcement learning studies. Some applications are implemented in simulation environment such as tracking control of yaw angle by Q-learning on discretized state space [3] or by combining Q-learning with genetic algorithms to acquire an adaptive control policy on continuous state space [69]. In a recent study, altitude control of an actual miniature blimp is implemented [4]. The model free RL method uses GPR to approximate the utility values without using any prior information or demonstration. GPR method has also been used for learning the system dynamics of an indoor blimp [29]. A previous similar study [30] uses a nonlinear airship model for the control and online parameter identification of the blimp using GPR in reinforcement learning setting. Bayesian nets are deployed in partially observable learning setting with GPR for the purpose of altitude control of a blimp under rewards corrupted by noise [70]. These studies on controlling the airship approach the problem by reducing the number of degrees of freedom and use single control effector. However, for real time challenges, reducing the motion into single state is not possible and the airship is required to use multiple effectors at the same time to perform the tasks expected from it.

1.2.4 Action selection

Retrieving the optimal action in a LfD/RL system is one of the most important stages as it is needed not only while operating the system under the learned optimal policy, but also during the learning, to evaluate the value function. For robotic applications, employing a fast and accurate action selection method is a necessity, as most of the robotic systems require time sensitive inputs.

(i) Direct Lookup

In discretized RL framework, action utility values are kept in a table which the maximum utility yielding action can easily be looked up from [7, 8, 9, 10, 11]. In actor-critic RL framework, the value function is updated by the critic according to the feedback received from the environment and the policy function is updated by the actor according to the reward from the critic [47]. Therefore, the action can directly be evaluated from the policy function.

(ii) Optimization

When the continuous state-action space is used, utility values are fitted to a Q-function by function approximation methods. Extracting the best action from this function requires solving the optimization problem that aims to find the action that maximizes the utility for a given state. Nearest neighbor actions are used to form a quadratic function and Brent's method, a derivative free root finding algorithm, is used to locate the maximum [33]. Resilient propagation algorithm which is originally used to train neural networks is adapted to Gaussian process regression to act as an unconstrained optimization tool [31]. Depending on the number of states and actions, optimization problem might get very hard to solve. In addition, optimization methods require re-evaluation of the cost function repeatedly which requires long durations.

(iii) Interpolators

This approach makes use of several action values and maximum is determined by using a weighted sum. Wire fitting method is proposed as an interpolator that can work with any function approximator [71]. Control wires are multidimensional action vectors. This method yields very fast action selection when used with neural networks where interpolator weights are updated by backprop-

agation of the error at every step [23, 72]. However, when used with instance based methods, problems due to large size datasets might affect its performance.

1.2.5 Reward Function

One of the most important characteristics of the RL framework is the reward function which determines whether an action taken in a state either results in positive or negative outcome. For this reason, the choice of the reward function must be compatible to the needs of the task. When incorporating the expert demonstrations into RL framework, two main approaches exist.

(i) Known Reward function

In the first approach, demonstrations are taken as an initial policy and an engineered reward function is employed. While the expert operates the system, states and actions are recorded and either rewarded or penalized according to this existing reward function. Later, if the learner explores new regions of the state-action space, the same reward function is used to add new experiences to the dataset [33, 73, 74]. This approach takes the expert demonstrations as starting policies that are open to further improvement.

(ii) Learning the Reward function

Second approach in this context suggests that for complex robotic challenges, designing a reward function that covers the whole state-action space is also a hard task and interprets the learning problem as the learning of the actual reward function out of the demonstrations provided by the expert. Usually several demonstrations of the same task are recorded and combined to create the ideal demonstration that the expert intended to perform. This and similar approaches branched from this idea are generally called *Inverse Reinforcement Learning* or *Apprenticeship Learning* [75, 57, 76, 66, 58, 59]. In this setting, the

expert demonstrations are thought to be the most optimal trajectories although research showed that the learner can outperform the expert in many tasks.

1.2.6 Improving Policy

Learning systems perform exploration in order to improve the current policy which is followed. Especially, if no prior knowledge is attached to learning framework, exploration becomes more important. In basic terms, exploration is to select an action that is not the best action according to the current policy. There are several methods suggested to control the amount and shape of exploration.

(i) Undirected Exploration

In undirected exploration, explorative actions are selected randomly. The most uninformed way of exploration of an unknown environment is to pick actions from a uniform distribution. Exploration can be balanced with an ϵ-*greedy* scheme that uses a random walk experiment to take either the optimal or the exploratory action [27, 63]. By selecting actions from a random distribution such as Boltzmann distribution, instead of the uniform distribution, exploration balance can be automatically handled as the parameters of the distribution determines the amount of randomness [24]. Although these methods are well applicable for simulations, for actual implementations of the robotic systems, undirected explorations are usually expensive and carry safety risks.

(ii) Directed Exploration

Directed exploration uses some knowledge about the system to decide on explorative actions. Consecutive rewards are used to determine exploration rate [68, 56]. Safe exploration methods perform a controlled exploration by choosing the action from a Gaussian process that has a mean at the current best action [77, 78, 18]. Statistical methods are used to find the explorative action that

will provide the maximum improvement in order to find the global maximum of the Q-function [4]. While these methods provide a safe and controlled exploration, learning becomes restricted and some portions of state-action spaces might never be learned.

(iii) Reengaging the Expert

Effectiveness of LfD systems depend on the dataset provided by the expert. Therefore, during the operation if the learner encounters an unknown state which is not demonstrated by the expert, learner should perform a generalization from the existing states which might not always produce reliable results. A certainty function can be used to decide how reliable of an approximation of the unknown state can be obtained from the existing demonstrations. According to this confidence value, expert may be asked to create new demonstrations [73]. Learner submits this confidence value to the expert, who decides to either take over the system or let the learner perform [79]. These methods require the expert to be in the loop at all times, therefore not applicable to robotic systems expected to perform fully autonomous tasks.

1.3 Original Contributions

The contributions of this research can be listed as follows:

(i) An LfD/RL technique is employed for the task of controlling an underactuated blimp system. Previous studies conducted for airship control used RL in a conventional setting and thus attempted to learn without any prior information. However, these studies limited the problem to single state, either by learning only yaw control or altitude control without pitching. Airship exhibits several distinctive features that makes predictive control to perform better than the

reactive control methodologies. Using expert knowledge in RL setting helps designing a control system that is shaped by experiences of the expert.

(ii) A Locally Weighted Regression based continuous state-action Q-Learning method is implemented for the control of airship. Previous studies on airship either used discretized state-action spaces, or used Gaussian process regression based approaches. Discretized state-action spaces requires a fine discretization to get a good representation of the airship dynamics. However, this increases the memory requirement. Gaussian process regression creates accurate global models. However, it introduces computationally challenging optimization tasks which are needed to be performed at every step. Locally weighted regression provides a new computationally efficient framework that makes it easier to run in real time applications.

(iii) Control of airship in very low speed phase of motion is focused, particularly slowing down to hovering and speeding up from hover. This flight phase is especially challenging to control because control authority from the aerodynamic control surfaces is lost which causes the number of the available control effectors to become much less than the number of the degrees of freedom. This puts the vehicle in the category of *underactuated* systems. Designing traditional control systems for underactuated requires extensive analysis and design efforts. Using expert knowledge alleviates these efforts.

(iv) Optimal action selection algorithm allows more than one dimensional action spaces for the learning control of airship, as opposed to the existing literature examples on airship which can work with only single dimensional action. Optimal selection of multiple actions have been implemented in other kind of robotic systems. However, unmanned airship use a variation of vectored thrust which thrust force and the thrust angle are two control effectors. There are regions in

the action space, such as changing thrust angle when thrust magnitude is zero, that make the optimization problem become harder.

CHAPTER 2

Overview of Reinforcement Learning

This chapter aims to give a brief description of widely used terms and definitions in reinforcement learning. First section presents key concepts that are fundamental to the subject by focusing on the discrete state and discrete action spaces while the second section discusses the encountered problems and suggested solution methods when the subject is extended to continuous state action space. As the subject borrows many ideas and tools from different disciplines and covers a wide span, discussion is limited only to the methods and definitions which are used in this research.

2.1 Introduction in Discrete Systems

Reinforcement learning is the collection of methods in artificial intelligence, machine learning and optimal control that learn appropriate behaviors through interaction. In RL, unlike supervised learning, the learner or the *agent* is not provided with a predefined set of correct input/output pairs. Instead, the agent operates in the *environment* and implements "trial and error" by exploring the outcome of possible choices to learn to achieve a *task*. An instantaneous representation of the configuration of the system in the environment is called a *state* [80]. At time t, the agent observes state s_t and performs an *action* a_t according to a mapping from state to action. This mapping is called the agent's *policy* and denoted as π, where $\pi(s, a)$ is the probability of taking the action $a_t = a$ in the state $s_t = s$ [81]. As a consequence of this action, the agent transitions into another state s_{t+1} and receives a *reward* r_{t+1} from the environment as a scalar performance feedback. A simple example in

Figure 2.1. A simple example to illustrate state, action and reward concepts in RL.

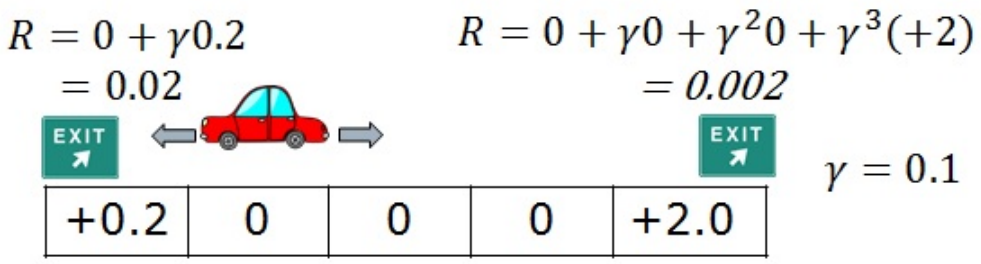


Figure 2.2. Comparison of immediate and long term rewards in discounted setting.

Fig. 2.1 illustrates concepts of state, action and reward. In this example, the agent "car" moves in the environment which has been partitioned into states in the form of "blocks". Car has two available actions as to move forward and backward that will cause a state transition and result in a reward according to the new state's proximity to the target "house".

Main objective of the agent is to maximize the *return*

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \tag{2.1}$$

which is defined as the sum of individual rewards over the course of a task. This representation of the return uses a final time T, which makes sense for *episodic tasks* when the interaction of agent and environment can be broken into identifiable parts [81]. However, for the *continual tasks*, a terminal state cannot be formulated and maximizing the return becomes problematic. In such a case, *discounting* is introduced to

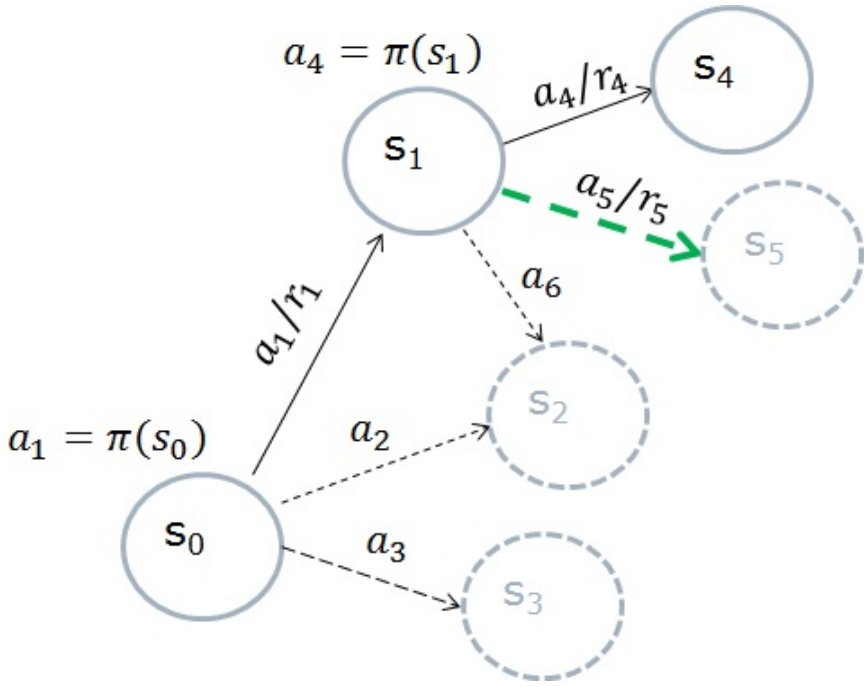


Figure 2.3. Representation of states, actions and policy.

balance immediate rewards over the long term rewards. In such a case, the discounted return is defined as

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate [81]. When $\gamma = 0$, the agent is only concerned with the immediate reward and as the γ increases, future rewards gain importance. Figure 2.2 depicts an example of the effect of discount rate on the long term rewards over immediate ones. In this example, an episodic task is presented and the episode ends when agent arrives at either one of the exit states. Agent either can take left action and receive the immediate low reward, or it can take several right actions in order to achieve long term high reward. However, as the discount rate is close to zero, after several steps to get to the long term reward, return of the high reward will become less than the return of the low reward.

The goal of RL is to learn the optimal policy $\pi^*(s,a)$ that will maximize the expected long term reward over the states. *Value* of a state s under a policy π,

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left[R_t \mid s_t = s\right] \\ &= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^i r_{t+k+1} \mid s_t = s\right] \end{aligned} \tag{2.3}$$

is the corresponding long term reward when starting in s and following π thereafter [81]. E_{π} is the expected value operator given that agent follows policy π and t is any time step. $V^{\pi}(s)$ is called as the *state-value function* for policy π. Similarly, the *action-value* function for policy π, $Q^{\pi}(s,a)$ gives the expected return starting from state s, taking the action a and thereafter following policy π.

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left[R_t \mid s_t = s, a_t = a\right] \\ &= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \end{aligned} \tag{2.4}$$

Further, using the recursive relationship, properties of conditional expected value operator and deterministic system assumption, Eq. (2.3) can be rewritten as:

$$V^{\pi}(s_t) = r_{t+1} + \gamma V^{\pi}(s_{t+1}) \tag{2.5}$$

A policy can be said to be better than another policy by comparing their value functions for all state-action pairs. In Fig. 2.3, agent starts in the state s_0 with available actions a_1, a_2, a_3, selects action a_1 as dictated by the policy π and as a result arrives at state s_1 with received reward r_1. At s_1, among the available actions a_4, a_5, a_6, the agent takes a_4 once again according to the policy-in-use π, even though the optimal action is a_5 which is shown with a green dashed arrow. By always deciding according to the policy π, value function $V^{\pi}$of the policy π can be calculated using the recursive relation in Eq. (2.5). This procedure is known as the "policy evaluation". After the policy is evaluated, "policy improvement" can be performed by taking

actions $a \neq \pi(s)$ in the states and modifying the policy to yield the higher value. As finite number of policies can be defined for finite state-action space, "policy iteration" obtains the optimal policy by repeating policy improvement until convergence.

In reinforcement learning, there are several methods to solve the problem of finding optimal policy. Temporal Difference (TD) methods are one of the approaches that attempt to find the optimal value function rather than directly searching for the policy. In TD methods, the *Value* estimates are updated based on the previously learned estimates and full knowledge of the system is not required. TD methods start from an initial value, $V(s_t)$ and update it every time step, immediately after observing a reward r_{t+1}, as a result of the transition from state, s_t to s_{t+1}. The simplest TD algorithm known as TD(0) has the update equation of:

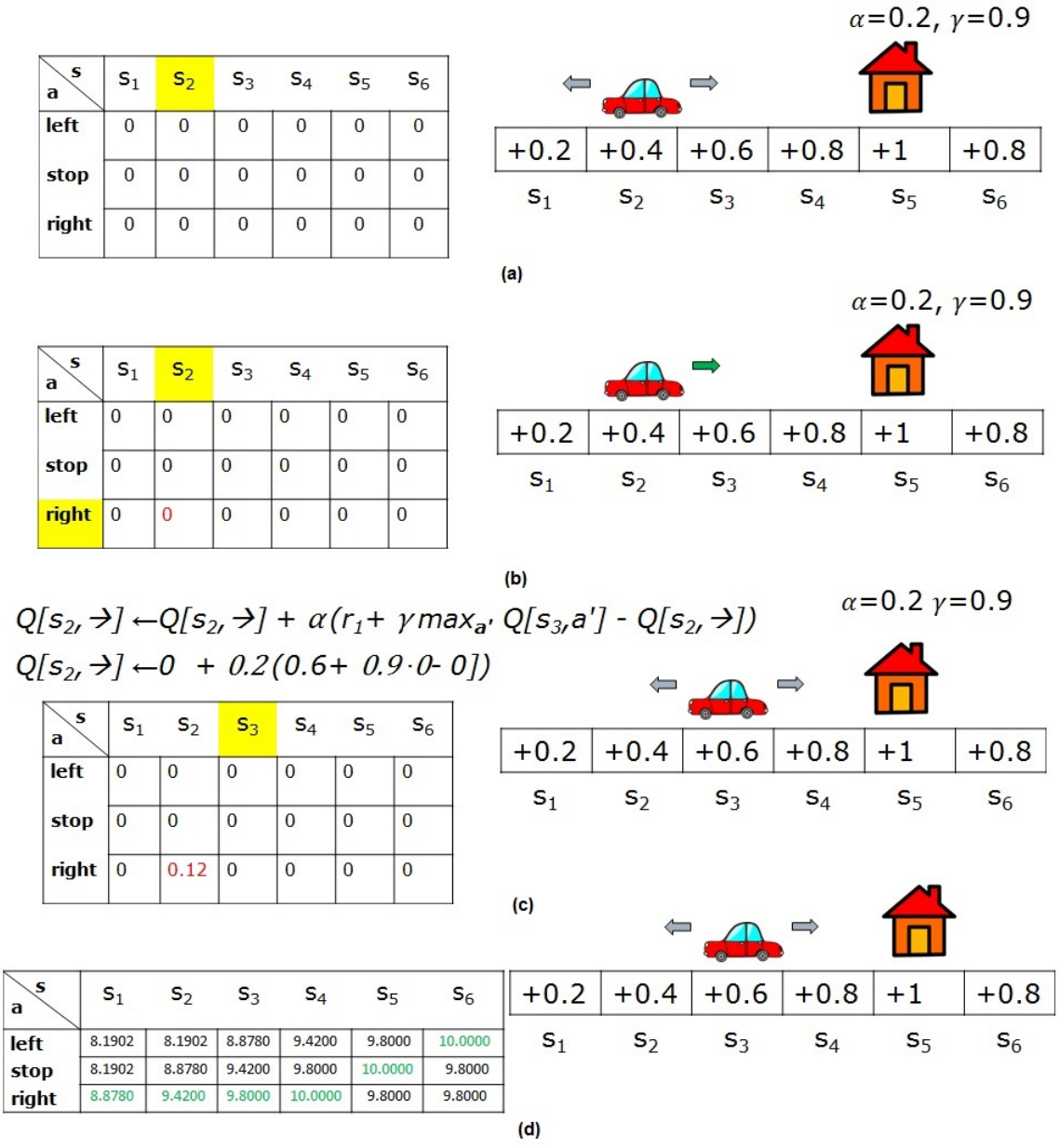
$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{2.6}$$

where α is the learning rate, $0 \leq \alpha \leq 1$, that controls convergence. TD(0) is called an "on-policy" method as the agent only learns the value of the policy being followed. A widely used TD method Q-Learning has the update equation

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{2.7}$$

which operates on the action-value function. In Eq. (2.7), the term $\max_a Q(s_{t+1}, a)$ represents the optimal action-value that can be achieved in state s_{t+1}. This means that agent learns the value of the optimal policy independently of the actions taken which makes Q-Learning an "off-policy" method. Q-Learning, by default, assumes a discrete set of states and actions where the action-values are represented in a tabular form. This way, optimal action and action value can easily be retrieved through a simple look-up or search procedure.

Figure 2.4(a)-(d) illustrates an example of Q-Learning implementation for the simple



a \ s	s_1	s_2	s_3	s_4	s_5	s_6
left	0	0	0	0	0	0
stop	0	0	0	0	0	0
right	0	0	0	0	0	0

a \ s	s_1	s_2	s_3	s_4	s_5	s_6
left	0	0	0	0	0	0
stop	0	0	0	0	0	0
right	0	0	0	0	0	0

a \ s	s_1	s_2	s_3	s_4	s_5	s_6
left	0	0	0	0	0	0
stop	0	0	0	0	0	0
right	0	0.12	0	0	0	0

a \ s	s_1	s_2	s_3	s_4	s_5	s_6
left	8.1902	8.1902	8.8780	9.4200	9.8000	10.0000
stop	8.1902	8.8780	9.4200	9.8000	10.0000	9.8000
right	8.8780	9.4200	9.8000	10.0000	9.8000	9.8000

Figure 2.4. Illustration of Q-Learning implementation for the car-house example.

car-house example. At the beginning, no states has been visited, thus the table that holds the action-values with respect to states and actions is empty. Car is located at the s_2 state and two available actions exist. On (b), car selects to take "right, ($\rightarrow$)" action. As a result of this action as shown in (c), car moves to state s_3 and receives immediate reward of +0.6. In this step, action-value of the state action pair $(s_2, \rightarrow)$ is updated according to Eq.(2.7), using the immediate reward obtained and the current estimate of the largest action-value that can be achieved in state s_3. As all available action values for s_3 on the table are zero, only the immediate reward has influence on the update. When this procedure is repeated for longer time, at some point, action-values converge and optimal policy can easily be viewed in the table as can be seen on (d).

Q-Learning is the preferred learning method since it is model free, learns off-policy and has the ability to incorporate prior knowledge [23]. However, it is defined for discrete state-action spaces, while the most of the robotic challenges are continuous in nature. For implementation of the method in continuous state-action spaces a function approximator is needed to represent the action-value function and a maximization procedure is required to find the optimal action-value. In the following section, main issues in the implementation in the continuous systems are presented.

2.2 Issues in Continuous System Implementation

The methods described in the previous section can be made applicable to the continuous state-action spaces by discretizing the state and action spaces. However, discretizing the state and the action space results in problems such as hidden states or high demand for memory and computing power especially when states and actions are multi-dimensional. Another solution to work directly with continuous states and action is to utilize function approximation techniques. A function approximator finds

a relation between a given set of inputs and outputs using mathematical methods. In machine learning, the given set of inputs and outputs are called *features* and *labels*, respectively. Features and labels constitute the *training* set. In the context of Q-Learning, features are state-action pairs and the labels are action-values. The approximation function is formulated based on the training dataset and is used to compute the action-values for a given state in the continuous state space.

One of the popular value function approximation methods is Cerebella Model Articulation Controllers (CMAC) which can be viewed as a mixture of look up tables and neural networks [82]. In this method, several feature fields are defined in the state-action space, in the form of overlapping rectangular tiles where every tile has a mapped value. A given state activates several of these tiles and the output is a weighted sum of the corresponding mapped values. Although the inputs are continuous, outputs are still discrete and the method cannot be classified as a fully-continuous method. Neural networks are widely used in continuous RL applications to approximate the value function.

Neural networks are formed of multiple neurons. Each neuron has multiple inputs and an output. Weighted sum of neuron inputs pass through an activation function and if it is larger than a threshold, the neuron fires and output is created. When a neural net is trained, the weights are adjusted to satisfy the desired input output relation, however the training samples are not kept. Using multiple layers of neurons, most classes of the functions can be approximated. However, they are not aggressive learners, which means they require many training samples before they can generate plausible predictions [37]. Another problem called *destructive interference* is likely to occur in neural networks, as the approximated value function can have different shapes on the different parts of the state-action space. When training starts, neural network learns a model which is locally valid in the state-action space. Later,

when it gets samples from a different portion of the state space, neural net starts to modify the learned model to fit the new data as well. This destroys the previously gained knowledge in an irreversible way [37].

On the contrary, *instance* or *memory* based algorithms store all the training samples and create models based on linear or nonlinear kernel functions to approximate the values of the queries. Gaussian process regression is a subset of instance based algorithms in which a global model is attempted to be learned by Gaussian kernels. A Gaussian kernel is specified by a mean and covariance function. Covariance functions can have several forms and they describe how the outputs are correlated as a function of the location of the inputs. When the value of a state-action pair is queried, covariances between the given state-action pair and each entry in the dataset are calculated to form the covariance matrix. The output value is predicted using the inverse of the covariance matrix and the outputs of the training dataset. Parameters of the covariance function are called *hyperparameters* and quality of the prediction is dependent on finding the best set of hyperparameters [42]. However, in most cases, these parameters are not known at the start. The process of finding the hyperparameters is called *model selection*. One method for model selection is to solve an optimization problem by finding the set of parameters which are maximizing the marginal data likelihood given the existing dataset [4]. Although Gaussian process regression can create accurate predictions, as the size of dataset increases, the computational complexity of the operations becomes very high since all entries of the dataset are used in the covariance calculations to fit a global model and hyperparameters need to be updated every time when new samples arrive.

Another subset of instance based methods, called *locally weighted learning* methods, create local models, only when a query is needed to be answered [83]. These methods are also referred as *lazy learning* methods. Simplest method in lo-

cally weighted learning is *nearest neighbor* which finds the closest point to the given state-action pair in the training dataset and uses its value. *Weighted average* method retrieves closest points, based on some measure of "closeness" or a limit on the number of close points, and creates an average by using the distances as inversely proportional weights. *Locally weighted regression* (LWR) fits a surface where the nearby dataset entries of a given state-action pair have more influence on the shape of the regression. LWR employs a more complex procedure than the methods such as nearest neighbor. However, the prediction quality is better. All locally weighted learning methods require a measure of relevance in order to find the closest data points [37]. For this purpose, several distance metrics can be used in order to define the closeness of the points. Distance function can be a global measure or query based local functions can be used. Weighted distance functions can be used for feature selection. Most widely used distance metric is ℓ_2-norm, which is also known as the Euclidean distance. Euclidean distance defines closeness according to the geometrical relation of the features [84]. However, as the number of input dimensions increases, Euclidean distance loses it geometrical meaning and no longer provides an accurate relation of closeness. Designing a reasonable distance metric is required to employ locally weighted learning in higher dimensional spaces.

Another requirement for locally weighted learning is the training data. There should be enough number of data points and they should be labeled, which means that every input must have an associated output. Although locally weighted learning saves computational effort from the training, lookup procedure requires an expensive computational cost especially when the dimension and the number of the samples increase [83]. However, with small number of training samples, prediction can be performed and the methods are not prone to destructive interference.

Another issue encountered during the implementation in the continuous state-action spaces is the action selection. In discrete state-action space, states, actions and the corresponding action-values are stored in a tabular form and the optimal action at a given state can be found by searching the maximum action-value generating matrix element. However, in continuous state-action space an optimization procedure is needed to compute the optimal action which generates the highest action-value in a given state. Computation speed of the optimization is an important factor as the robotic systems require calculation of fast and accurate control actions. This becomes a challenge especially when there are many states and actions, as the optimization function becomes very hard to solve. In addition, continuity of action-value approximation function is an important factor determining the accuracy of the calculated action. Especially, during the early stages of learning, approximated action-value function has discontinuities as the state-action space is not thoroughly visited. This causes most optimization methods to fail as they require the derivative/gradient of the function to be optimized. Thus, derivative free root finding methods such as Brent's method [33] or gradient free optimization algorithms such as Nelder-Mead [85] performs better in this circumstances.

CHAPTER 3

Formal Description of LfD/RL Framework

This chapter describes the details of the proposed LfD/RL framework. In the first four section, three stages and main three subroutines used in the method is first explained in detail. On the last section, method is implemented on three simple example cases to show the performance and the convergence characteristics of exploration in discrete state discrete action, continuous state discrete action, and continuous state continuous action spaces.

3.1 Stages of the LfD/RL Framework

Reinforcement learning techniques can be employed to learn how to perform a task without knowing any prior information. This is done by exploring the environment by applying random actions and collecting rewards. In a small state-action space, optimal policy can be obtained in a short time. However, as the number of states and actions increase, the time to visit all state-action pairs and learn action-values would take a considerably long time. In addition, for robotic tasks, applying random actions can create safety risks. Such reasons create the demand for incorporating prior knowledge into the learning framework to speed up the learning and provide safety by setting guidelines. In LfD techniques, the main objective is to mimic a demonstrated task exactly as it is performed by the expert. If the expert teaches a suboptimal trajectory, learning agent cannot improve this. By combining the two approaches, a fast and accurate learning can be achieved. In the approach of Joystick

Aided Q-Learning (JAQL) algorithm [37], an expert creates an initial policy and the agent starts learning from this initial policy.

In this study, a learning framework which implements Q-Learning algorithm is followed in a similar approach. Q-Learning algorithm implementation that is depicted in the flowchart in Fig. 3.1 has three subroutines which are repeated at every execution/update time in the order of *Action Selection*, *Action-Value approximation*, and *Action-Value update*. Action selection is the procedure of determining the action to take for a given state according to the policy in use. This subroutine is also used to compute the optimal action for a given state and its corresponding maximum action-value. Action-value approximation evaluates the predicted action-value of a given state-action pair according to the current knowledge extracted from previous experiences over the state-action space. Action-value update recalculates the action value with the received reward after a state transition. These three functions need to be implemented differently for continuous and discrete state/action spaces as described in following sections.

Flowchart given in Fig. 3.1 summarizes the stages of Q-Learning algorithm. The algorithm starts from a given state and the training dataset which can be initially empty. First, an action is selected according to the policy-in-use. Action selection subroutine is used to compute the current best action, which the policy may choose to use, or take a random action instead. In the next stage, it is checked to see if the state-action pair has been visited before by looking for an entry in the dataset which is closer than a certain small threshold. If the pair has been visited before, when the action-value prediction subroutine is run, it will automatically yield existing action-value on the dataset, moreover, in the action value update subroutine, a new entry on the dataset will not be created and the current value will be updated.

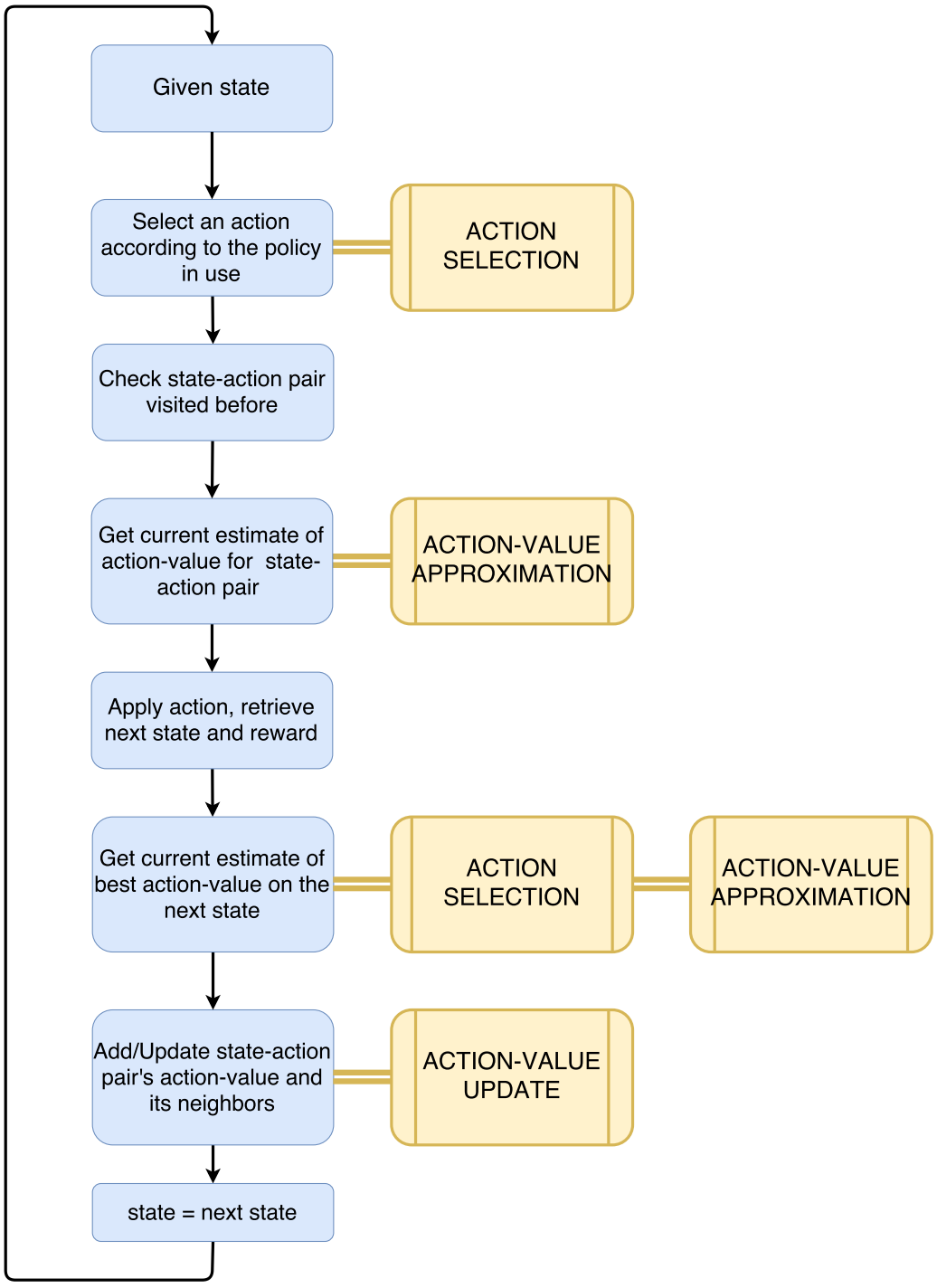


Figure 3.1. Flowchart of the continuous state-action Q-Learning.

After current action-value estimation for the state-action pair is completed, selected action will be applied and as result the agent will experience a state transition and achieve a reward. At this point, maximum action-value of the new state will be estimated using the same action selection and action-value approximation subroutines and all data will be passed into action-value update subroutine. Finally, algorithm will go back to the start by setting the new state as the given state.

Proposed LfD/RL framework consists of three consecutive stages which are *pilot demonstration*, *exploration* and, *mission execution*. In each stage, the same subroutines of the Q-Learning algorithm is performed.

In the *pilot demonstration* stage, initially there is no prior information which means the training dataset for continuous state action space or table for discrete action space is empty. Expert operating the vehicle acts as the policy which is used to select actions. When the expert applies an action a_t at a state s_t, agent moves to state s_{t+1} and the reward function delivers the associated reward r_{t+1}. The corresponding action-value is calculated for the state-action pair using Eq. (2.7) and is stored in the training dataset.

Expert performs several demonstrations that should cover different, preferably frequently visited, areas of the state-action space. Demonstrations performed by the expert may have similar state-action pairs. In such a case, it is not desired to register identical or very close training samples into the dataset due to limited memory capacity and also because redundant entries will increase the need for higher computational power. In order to avoid these issues, an online sparsification procedure is employed to discard the identical or very close training samples. In the literature, predictive variance threshold based online sparsification applications has been suggested [4]. Since the covariance in LWR is a function of the distance between samples, this can be simplified to using a distance threshold on the state-action pairs.

However, no matter how well sparsification is handled, the memory is limited and when the memory allocated for training data is full, there should be other methods required to deal with the issue.

In the *exploration* stage, the expert is no longer in charge. The main objective of this stage is to populate the training dataset by operating the vehicle in different parts of the state-space and/or taking actions that are distributed randomly. At every execution/update time, either a random action or the optimal action based on the current training dataset is selected. This selection is made by comparing a randomly picked number to the exploration rate, ϵ. If explorative actions are taken, a new training sample will be added to the dataset. However, if the optimal action is used, best action based on the training dataset will be selected and the response will be almost same as if the vehicle is executing the same task the expert performed. Also, since the system is deterministic, the state-action pairs in the same neighborhood are expected to produce similar action-values.

Finally, in the *Mission execution* stage, control actions are generated to perform an assigned task using the optimal actions based on dataset obtained at the end of exploration stage. In this stage, learning continues along with the execution of the mission. However, since the dataset already covers most of the states and system is deterministic, new entries on the dataset are rarely generated and, mostly, action-values of existing points are updated. Performance of the learning framework can be evaluated by the demonstration results obtained in this stage.

Subroutines in the LfD/RL framework intensively perform distance calculations among the entries of the training dataset. For this purpose, the simplest distance metric choice is the Euclidean distance. However, when the components of the states and actions are not in the same scale, calculated distance becomes biased towards the largest valued component. Furthermore, in some cases, specific components might

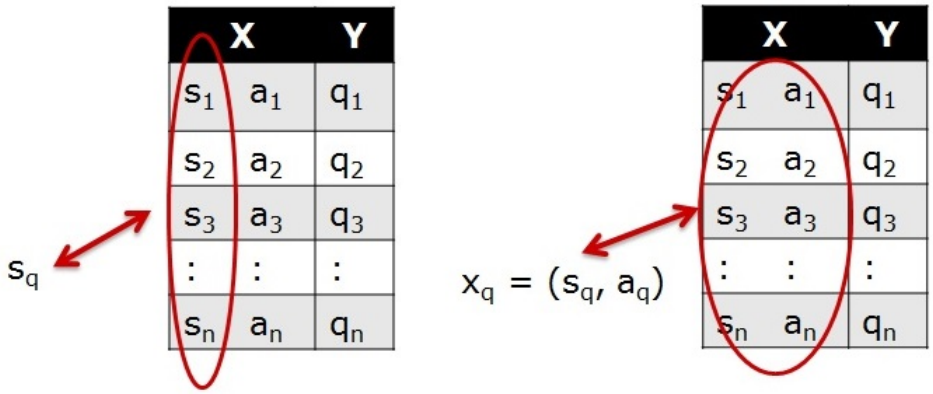


Figure 3.2. State based and state-action based distance calculations.

be desired to have more influence over the distance than the others. These can be addressed by weighting the components of states and actions in the distance calculation with a diagonal feature weighting matrix. In this implementation, features are weighted inversely proportional to their anticipated maximum values, such that the scale of state and action components and their contributions to the calculated distance are normalized. Depending on the task in the subroutine, state based or state-action based distances are used. For state based distance,

$$d_s = \sqrt{(s_q - s_i)\mathbf{M_s}(s_q - s_i)^\mathsf{T}} \tag{3.1}$$

only state s_q is provided, therefore, only the states in the training dataset s_i, as can be seen in Fig. 3.2, are used in the distance calculation with the weight matrix M_s which only contains weightings for components of the states. In the state-action based distance,

$$d_{sa} = \sqrt{(x_q - x_i)\mathbf{M_{sa}}(x_q - x_i)^\mathsf{T}} \tag{3.2}$$

state-action pair x_q is given and the distance calculation is performed using the state-action pairs x_i in the dataset with the weighting matrix M_{sa} which includes weightings for both state and action components.

3.2 Action Selection

Selecting the action with the best outcome at a given state is a necessary step for both the learning and actual control phases. Action selection algorithm must be fast and accurate as most of the robotic systems require (near) real-time execution. In discrete action spaces, this can be handled easily as the action associated with the maximum action-value at a given state is the optimal action. However, in continuous state/action spaces, selecting the best action turns into another optimization challenge. In the following discussion, a fast quadratic approximation method for single action case and its extension to multi-action case are presented. Subsequently, an alternative optimization method to find multi-dimensional optimal actions is introduced.

3.2.1 Quadratic Approximation

In this approach, to get the best action at a given state, the first step is to find the closest k neighbors among the current state samples. This is done by an exhaustive search based on the weighted Euclidean distance. Then k actions of these neighbor states are taken as the local action set. These local actions are paired with the given state and their respective action-values are estimated using the action-value approximation procedure detailed in Section 3.3.2. A quadratic function is fitted using least square error method between the local action set and the corresponding action-values, and this quadratic function provides direct analytic solution for the maximum. Depending on the dimension of the action space, different quadratic functions are utilized and the process to obtain the local maximum slightly varies.

3.2.1.1 Single Action Case

In the single action case, the quadratic function is a parabola. Consider a given state s_0 with three closest neighbor state-action pairs are

$$\begin{aligned} p_1 &= (s_1, a_1) \\ p_2 &= (s_2, a_2) \\ p_3 &= (s_3, a_3) \end{aligned} \tag{3.3}$$

Considering these three actions taken at state s_0, action-values are estimated as detailed in Section 3.3.2

$$\begin{aligned} q_1 &= \hat{Q}(s_0, a_1) \\ q_2 &= \hat{Q}(s_0, a_2) \\ q_3 &= \hat{Q}(s_0, a_3) \end{aligned} \tag{3.4}$$

which are used to compute the coefficients of the parabolic fit for action-values

$$Q(a) = c_0 + c_1 a + c_2 a^2 \tag{3.5}$$

by solving the matrix equation

$$\begin{bmatrix} 1 & a_1 & a_1^2 \\ 1 & a_2 & a_2^2 \\ 1 & a_3 & a_3^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{3.6}$$

The local optimum is known to be at

$$a_{opt} = \frac{-c_1}{2c_2} \tag{3.7}$$

This point can be either a maximum or minimum depending on the sign of c_2. However, even if the point is a maximum, it cannot be directly used since the parabola

fit may have its maximum at a very far away point than the three neighbor action range. For this reason, two extended boundary values a_{high} and a_{low} are defined as:

$$\begin{aligned} a_{high} &= a_{max} + 0.2(a_{max} - a_{min}) \\ a_{low} &= a_{min} - 0.2(a_{max} - a_{min}) \end{aligned} \tag{3.8}$$

where

$$\begin{aligned} a_{max} &= max(a_1, a_2, a_3) \\ a_{min} &= min(a_1, a_2, a_3) \end{aligned} \tag{3.9}$$

Action-values of a_{opt}, a_{high}, a_{low} are calculated as

$$\begin{aligned} q_{opt} &= \hat{Q}(s, a_{opt}) \\ q_{high} &= \hat{Q}(s, a_{high}) \\ q_{low} &= \hat{Q}(s, a_{low}) \end{aligned} \tag{3.10}$$

Finally, the best action is chosen as the maximum action-value yielding of these three actions.

$$a^* = \arg\max_a(q_{low}, q_{opt}, q_{high}) \tag{3.11}$$

if a_{opt} is not within (a_{min},a_{max}), then

$$a^* = \arg\max_a(q_{low}, q_{high}) \tag{3.12}$$

when the optimum action is outside the neighborhood action range as can be seen in Fig. 3.3.

3.2.1.2 Two Action Case

Many of the robotic systems require application of more than one action simultaneously in order to achieve desired control characteristics. This requires to have

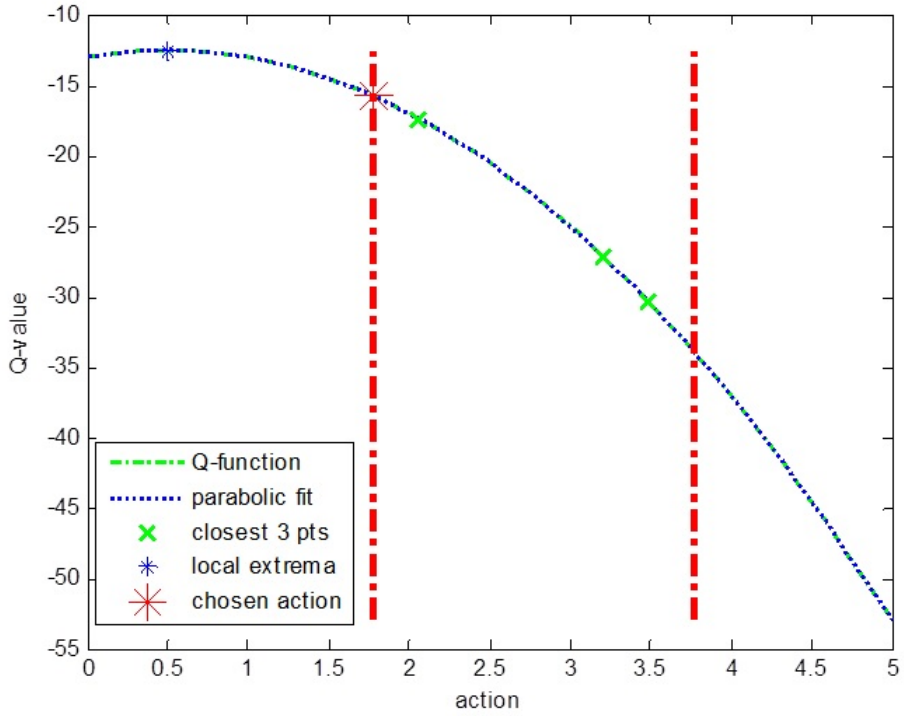


Figure 3.3. One Action Quadratic Approximation.

a multi-dimensional action space where the optimal action that will yield the maximum action-value becomes a set of values rather than a single value. Similar to the single action case, for a given state s_0, a set of closest neighbor state-action pairs are selected and respective action-values are calculated by considering as if these actions were taken at the given state. Coefficients of the parabolic expression:

$$Q(a_1, a_2) = c_1 a_1^2 + c_2 a_1 + c_3 a_1 a_2 + c_4 a_2 + c_5 a_2^2 + c_6 \tag{3.13}$$

are computed by solving a similar matrix equation in Eq. (3.6), where a_1, a_2 are the two actions of the multi-dimensional action space. Partial derivatives of Eq. (3.13)

$$\begin{aligned} \frac{\partial Q}{\partial a_1} &= 2c_1 a_1 + c_2 + c_3 a_2 \\ \frac{\partial Q}{\partial a_2} &= c_3 a_1 + c_4 + 2c_5 a_2 \end{aligned} \tag{3.14}$$

equal to zero at the extremum points,

$$a_1^* = -(2c_2c_5 - c_3c_4)/(-c_3^2 + 4c_1c_5)$$
$$a_2^* = -(2c_1c_4 - c_2c_3)/(-c_3^2 + 4c_1c_5) \tag{3.15}$$

Using second derivative test, if the conditions:

$$c_1 > 0$$
$$4c_1c_5 > c_3^2 \tag{3.16}$$

are satisfied, then (a_1^*,a_2^*) pair is said to be a local maximum. In the single action case, after the local maximum was found, it was tested if it lies within the range of minimum and maximum values of the closest neighbor actions. In the multi action case, same test is applied by defining this range as a convex hull as the action space is two dimensional. Convex hull of the closest neighbors are computed by the *Graham scan* algorithm [86], then scaled up to create an extended convex hull as shown in Fig. 3.4. If the local maximum point is located inside this extended hull, it is accepted and used as the optimal action. In the cases when local maximum is not located inside the convex hull or a local maximum was not found from the second derivative test, the optimal action is sought on the boundaries of the convex hull. For the ith edge of the convex hull, first a linear relation:

$$a_2 = m_i a_1 + n_i \tag{3.17}$$

is derived using two consecutive corners of the convex hull and later inserted into Eq. (3.13) to obtain a single variable quadratic expression:

$$Q(a_1) = c_1a_1^2 + c_2a_1 + c_3a_1(m_ia_1 + n) + c_4(m_ia_1 + n_i) + c_5(m_ia_1 + n_i)^2 + c_6 \tag{3.18}$$

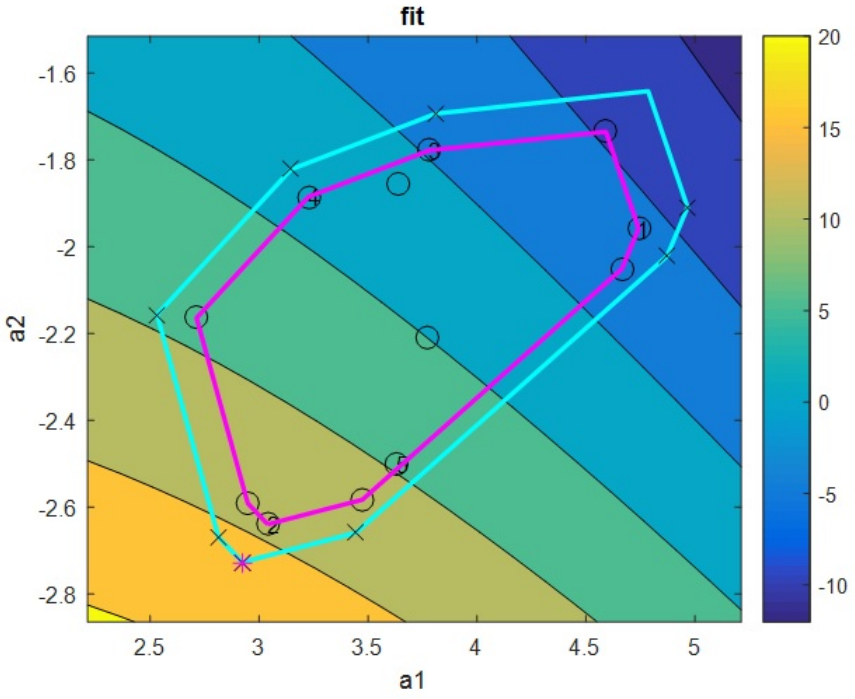


Figure 3.4. Two Action Quadratic Approximation.

Then local maximum candidates for the edge

$$a_1^* = \arg\max Q(a_1)$$
$$a_2^* = m_i a_1^* + n_i \tag{3.19}$$

are found with the same technique with the single action case. This procedure is repeated for all edges of the boundary of the convex hull. Among the optimal actions of all the edges, the largest action-value yielding one is selected as the optimal action. Although this multi-action selection logic can be applied to more than 2-dimension case by setting an appropriate form for the action-value function and implementing the convex hull in n dimensional, this would highly increase the computational complexity. Therefore, for higher dimension action spaces, an optimization method should be used.

3.2.2 Direct Search Optimization

Direct search algorithms are gradient free optimization algorithms that are suitable for non-differentiable functions or functions with discontinuities. Algorithms in this family search for the optimal value around a provided initial guess. Nelder-Mead or Downhill Simplex method is one of the most known methods in this family and has been used in several RL applications for the purpose of continuous action selection [85].

The *fminsearch* function in Matlab Optimization Toolbox implements Nelder-Mead algorithm and is utilized in this study for multi-action selection as depicted in the flowchart in Fig 3.5. As can be seen, the subroutine initially starts from a given state and the dataset. Dataset initially can be empty and in this case, best action is set to zero. However, any value, not necessarily zero, could be used, just as a placeholder, as this action is never used since the presented LfD/RL method only have an empty dataset before the pilot demonstration stage and in this stage only pilot actions are applied. If there are entries on the dataset, distances based on the state are calculated. If the number of entries in the dataset is not enough (less than predetermined value k), best action is selected as the closest neighbor's action. For the example cases in this work k is taken as 10 through trial and error. If there are more than k entries in the dataset, among closest k neighbors, the action with the highest action-value is selected as the initial value that the optimization routine will start its search from.

The optimization routine *fminsearch*, by default attempts to minimize a given optimization function. In this case, the provided optimization function is

$$f(a) = -f_Q(s, a, X, Y)^2 \tag{3.20}$$

where X and Y are the state-action pair entries and their corresponding action-values in the training dataset, respectively and f_Q represents the action-value approximation subroutine in Section 3.3. Negative sign is added since the goal is to find the action that will maximize the action-value. Finally, returned action from the *fminsearch* is taken as the best action and subroutine terminates.

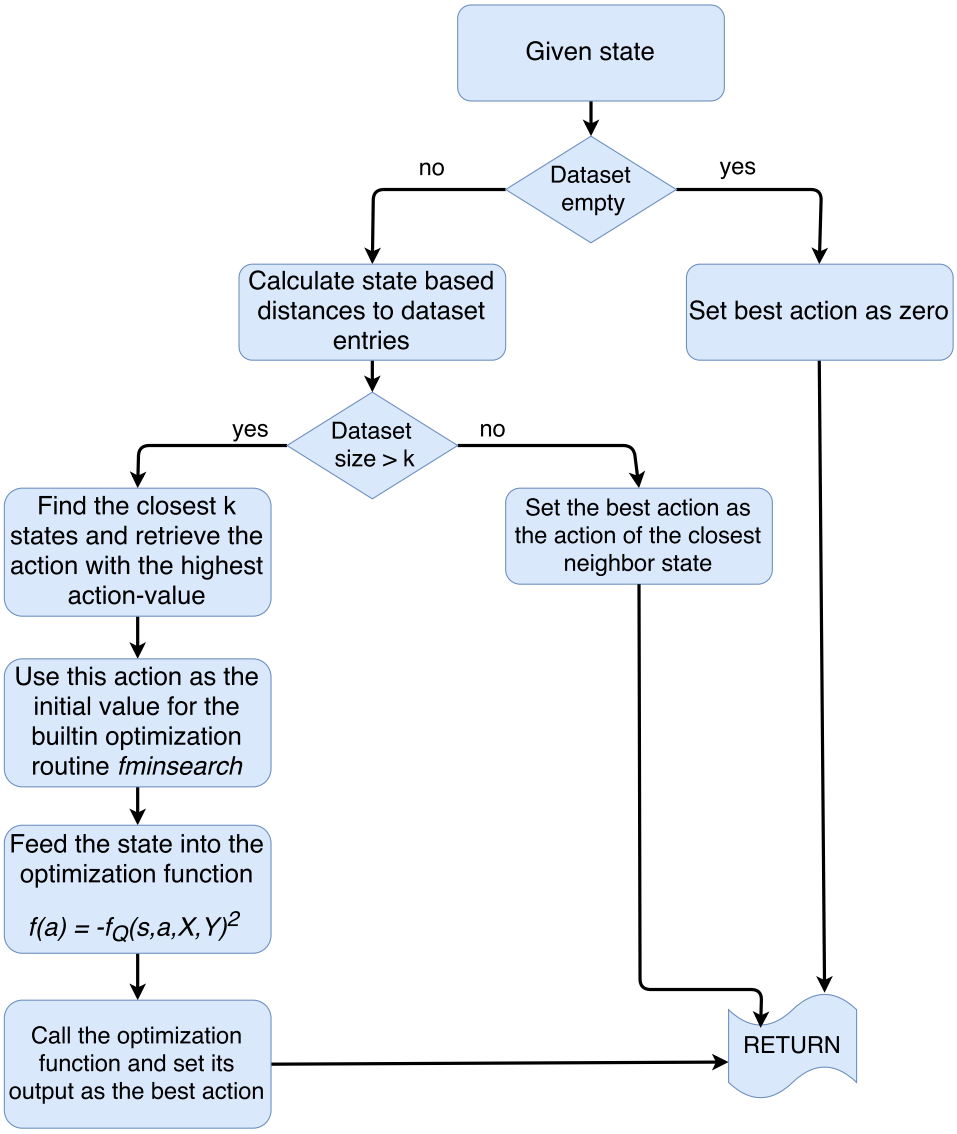


Figure 3.5. Flowchart of the multi-dimensional action selection subroutine.

3.3 Action-Value Approximation

In discrete state-action applications, action-values of state action pairs are stored in a tabular form and retrieved by a simple lookup procedure. Main role of this subroutine is to create an equivalent procedure for continuous state-action cases as depicted in Fig. 3.6. Subroutine starts with the state-action pair. If the dataset is initially empty, subroutine returns zero so that learning update of Eq. (2.7) is performed only based on the immediate reward. In addition, if the state-action pair is previously visited and there is already an entry in the dataset, this value is returned as the output. Next, the state-action based distances to the entries in the dataset are calculated using Eq. (3.2). If there is not enough number of neighbors with respect to a previously set threshold k, closest state-action pair's action-value is returned. If there are more entries than the threshold, the elliptic hull which is detailed in Section 3.3.2 is created and it is checked to see if the given state-action pair lies inside the hull. If the state-action pair is not surrounded by the elliptic hull, then once again the closest neighbor's action-value is returned. If the state-action pair is surrounded by the hull, locally weighted regression is performed to calculate approximated action-value as detailed in Section 3.3.1. Numerical errors happen during the calculation of Eq. (3.28) when the matrix inversion cannot be carried out due to singularity. If this numerical error occurs, the closest neighbor's action-value is returned. At this point subroutine is terminated.

Locally weighted regression method is used to predict the action-value of a state-action pair in order to implement the Q-Learning with continuous state and action spaces. However, prediction procedure becomes computationally challenging as the number of training samples increases. Furthermore, regression results may diverge, if the given state-action pair is not surrounded by neighbor datapoints and as a result, an extrapolation, instead of interpolation, is performed. Safe action-

value approximation methods [87] avoid this situation by taking precautions. Value approximation procedure used in this study follows some improvements suggested in [37] to make the prediction procedure more efficient and reliable such as limiting the prediction set size and creating a convex hull around the training data and perform regression only if the given state-action pair lies within the hull as described in the next two sections.

3.3.1 Locally Weighted Regression

Consider a training dataset

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ \ldots (x_n, y_n)\} \tag{3.21}$$

where x_i are the vectors of features and y_i are the corresponding scalar labels. Standard linear regression technique aims to solve

$$\mathbf{y} = \mathbf{X}\beta \tag{3.22}$$

where $\mathbf{X}$ is a matrix whose ith row is x_i, $\mathbf{y}$ is a vector whose ith element is y_i and β is the parameter set that minimizes the sum of residual squares criterion

$$\mathcal{C} = \sum_i \left(x_i\beta - y_i\right)^2 \tag{3.23}$$

Then, the ordinary least squares solution for β is

$$\beta = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} \tag{3.24}$$

In this case, for a given feature vector x_p, output of standard linear regression is

$$y_p = x_p\beta \tag{3.25}$$

LWR is a variation of standard linear regression. While linear regression attempts to fit a global approximation to the dataset, LWR fits a local model using weightings

$$w_i = \kappa\left(d\left(x_p, x_i\right)\right) \tag{3.26}$$

where $\kappa(.)$ is the weighting function or the kernel, $d(.)$ is the state-action based distance function in Eq. (3.2), x_p is the given feature vector. Dataset entries with higher weightings have more influence over the regression. With the addition of weightings, the error criterion in Eq. (3.23) becomes

$$\mathcal{C} = \sum_i w_i \left(x_i \beta - y_i\right)^2 \tag{3.27}$$

and Eq. (3.24) can be rewritten for weighted training set as

$$\beta = \left(\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{y} \tag{3.28}$$

where $\mathbf{W}$ is a diagonal matrix whose ith diagonal element is w_i. Once β is found, the value of given feature vector x_p is calculated as in Eq. (3.25)

Datapoints that are further away from the given feature vector are assigned very low weights by the kernel and their contributions are almost zero. Efficiency of LWR can be increased by using only the close neighbors instead of all the datapoints, as the matrix inverse operation in Eq. (3.28) will become simpler. This procedure needs an efficient and fast way of searching for neighbors. Tree structures such as kd-tree store the datapoints in an efficient way and greatly reduce the search time needed compared to exhaustive search. Another problem is the reliability of the prediction. If the close neighbors do not surround the given feature vector, the regression performs extrapolation rather than interpolation. This is not desired as the prediction of output quickly diverges when the inputs go beyond the range of training dataset. In order to avoid this, a convex hull which draws an envelope to the boundaries of the training dataset can be constructed and locally weighted regression can be performed only when the given feature vector is within the hull. However, constructing the exact convex hull for high dimensional training data is computationally expensive. As an alternative, an approximate elliptic hull can be defined. Considering $\mathbf{K}$ and $\mathbf{b}$ are the

subsets of $\mathbf{X}$ and $\mathbf{Y}$, representing the close neighbors which are determined by the distance metric, the *hat matrix*

$$\mathbf{H} = \mathbf{K}\left(\mathbf{K}^{\mathsf{T}}\mathbf{K}\right)^{-1}\mathbf{K}^{\mathsf{T}} \tag{3.29}$$

represents how much influence each data has on each fitted value [88]. Given feature vector x_p lies within the elliptic hull if

$$x_p{}^{\mathsf{T}}\left(\mathbf{K}^{\mathsf{T}}\mathbf{K}\right)^{-1}x_p \leq \max_i h_{ii} \tag{3.30}$$

where h_{ii} are the diagonal elements of the hat matrix $\mathbf{H}$. Then, the predicted value of the given feature vector x_p

$$y_p = x_p\left(\mathbf{K}^{\mathsf{T}}\mathbf{W}\mathbf{K}\right)^{-1}\mathbf{K}^{\mathsf{T}}\mathbf{W}\mathbf{b} \tag{3.31}$$

If the given feature vector is not located in the hull, then a default value is returned. This default value must be selected according to the structure of the reward function. In this implementation, the closest neighbors action-value is returned instead of a default constant.

3.3.2 Predicting the action-value

Previous section discussed prediction of locally weighted regression output y_p for a given feature vector x_p. Predicted action-value for a given state-action pair can be calculated in the same manner. In this study, the training dataset is not stored in a tree structure. In order to find the close neighbors, first the state-action based distances of the given state-action pair to the state-action entries in the training dataset, $\mathbf{X}$ must be calculated using Eq. (3.2). Calculated distances are sorted by a MATLAB command, which implements the quicksort algorithm. Quicksort sorts the data by partitioning it into smaller parts and calling itself recursively and is quite

efficient for large sized datasets [86]. In the sorted list, a fixed number of entries with the smallest distance are taken to form the close neighbor inputs $\mathbf{K}$ and their corresponding outputs $\mathbf{b}$. For the simulation cases considered in this work, this fixed number of entries is taken as 20 from trial and error. For each of $k_i \in \mathbf{K}$, kernel weightings are calculated with a Gaussian kernel

$$w_i = \kappa(x_p, k_i) = e^{(x_p - k_i)^2/h^2} \tag{3.32}$$

where h is the bandwidth parameter, which determines the smoothness of the fitted function. This parameter can be recalculated online using the dataset or kept constant. For each feature, a different parameter can be selected. In this study, smoothness parameter is assumed to be constant and same for all features of the state-action pairs. After the weights are calculated, action-value of the given state-action pair is predicted by the Eq. (3.31)

3.4 Action-Value Update in Dataset

The value approximation method described in the previous section can provide the predicted action-value of a continuous state-action pair. However, there is still a need to update the action-values according to the rewards received by the agent through the steps described in Fig. 3.7. Consider starting from a state s_t and taking an action a_t according to some policy-in-use, π. In this case, using the procedure presented in Section 3.3.2, we can predict the expected action-value $\hat{Q}(s_t, a_t)$ of taking action a_t in state s_t. As a result of this action, agent moves to state s_{t+1} and receives a reward r_{t+1}. At this point, it is desired to update the previously predicted action-value, according to the temporal difference which is due to the immediate reward of the state transition and the state-value of the new state reached. In order to employ Q-Learning, the same value approximation method in Section 3.3.2 is used to predict

the action-value of the optimal action at the new state. After the optimal action-value $\max_a \hat{Q}(s_{t+1}, a)$ is predicted, action-value of previous state is updated as

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a) - \hat{Q}(s_t, a_t) \right] \tag{3.33}$$

and the state-action pair (s_t, a_t) and its corresponding action-value $\hat{Q}(s_t, a_t)$ are inserted into the dataset as a new training sample.

In order to make the value approximation smoother, a secondary update is performed on the neighbors which were used in predicting the action-value of the state-action pair (s_t, a_t) by bringing their values closer to the newly calculated action-value, $\hat{Q}(s_t, a_t)$. For all the action-values, $b_i \in \mathbf{b}$, are updated as

$$b_i \leftarrow b_i + \alpha w_i \left(\hat{Q}(s_t, a_t) - b_i \right) \tag{3.34}$$

where w_i is the weights in Eq. (3.32), and α is the learning rate. At this point, learning for this iteration ends. The algorithm continues by selecting another action according to the policy-in-use, π and agent moves to another state and the same procedure is repeated until termination.

3.5 Simple Case Studies

In order to provide a good understanding of LfD/RL method and investigate effect of exploration especially when extended to the continuous state-action space, three simple case studies are performed.

In the first example both states and actions are discrete. In the second example, states are continuous and actions are discrete. However, states are discretized to create discrete states that make it possible to utilize Q-Learning method. In the final case, both states and action are continuous and effects of different sampling periods and function approximator parameters are considered in the analysis.

For all the examples, the UGV which will be described in detail in Chapter 4 is used. As the dynamics of the UGV is continuous, it is required to derive a discrete equivalent of the model to be able to use with Q-Learning. UGV has two actions which are left and right motor duty cycles and four states which are the x-axis position, x, y-axis position, y, orientation, θ, and the speed, V. For the simplified example, only forward backward motion is considered, and state is reduced to only x-axis position. In addition, for left and right motors same duty cycle is applied which results in a single action. Action can easily be discretized by selecting, for example, 10% increments over the range of [-100%,100%] to create 21 actions. However, it requires some consideration to decide on the state discretization margins based on the discussion in Section 1.2.1.

In order to create a discretized version of the UGV dynamics, discretized actions are applied to the system in 0.1 second long steps and the resulting x-axis positions are recorded as can be seen in Table 3.1. Here, δ represents the smallest position partition, which occurs with respect to the lowest motion generating action. The multiples of the lowest motion generating actions results in the same multiples of the smallest position partition.

Table 3.1. Steady state position changes when discretized actions applied to UGV

action	position	partitioned position	Δs
[-30:30]	0	0	0
± 40	± 0.00613	$\pm\delta$	± 1
± 50	± 0.01226	$\pm 2\delta$	± 2
± 60	± 0.01839	$\pm 3\delta$	± 3
± 70	± 0.02452	$\pm 4\delta$	± 4
± 80	± 0.03065	$\pm 5\delta$	± 5
± 90	± 0.03678	$\pm 6\delta$	± 6
± 100	± 0.04291	$\pm 7\delta$	± 7

Since only a single state is used to represent the multi-state dynamics, it is very important to make sure that selected single state can account for the UGV motion completely. For this reason, effect of lowest and largest motion generating actions, 40% and 100% are analyzed in detail. As can be seen in Figs. 3.8 and 3.9, even though the action is only applied for 0.1 seconds, the speed takes 0.3 seconds to settle down to zero. This suggests that if the learning updates are taken at the same rate with actions, existing nonzero speed as a hidden state will interfere with the state representation and will lead incorrect learning.

3.5.1 Discrete State - Discrete Action case

After a discrete state discrete action representation for the UGV is obtained, three stage LfD/RL method can be implemented. It can be seen from the "action" and "Δs" columns on Table 3.1 that action is in the range of [-100,100] and Δs is in the range of [-7,7]. By defining a mapping T from action to Δs, the equations of motion for the discrete state-discrete action system is written as

$$\Delta s = T(a_t) \tag{3.35}$$

$$p_{t+1} = p_t + \Delta s$$

where p represents the position state of the system. While this setup is used for the physical position of the system, for Reinforcement Learning, another state definition is introduced to reflect the fact that the main task to be learned is how to act to go to an assigned position, called waypoint. Thus, the state is defined as

$$s = p_{target} - p_{current} \tag{3.36}$$

This means, for example, that, if the current position of the vehicle is at the target position, state is zero and if the target is in front of the vehicle by $\pm 7\delta$, state is ± 7.

Two additional states must be considered for the cases when the difference between target and the vehicle is more than ± 7. By including these two states, state-action space can be represented by total 17 states and 21 actions.

After state-action space setup is finished, first stage of LfD/RL framework is done by two pilot demonstrations as shown in Fig. 3.10: (1) target is behind the vehicle by -3δ, ($s_6 \rightarrow s_9$), and (2) target in front of the vehicle by 4δ ($s_{13} \rightarrow s_9$). Demonstrations are performed with a suboptimal control policy

$$u = \begin{cases} 40\% & s < 0, \\ 0\% & s = 0, \\ -40\% & s > 0 \end{cases}$$

while learning updates are performed with learning rate, $\alpha = 0.2$, and discount rate, $\gamma = 0.9$. Reward function,

$$r = - \mid s \mid \tag{3.37}$$

is always negative except when the distance between target and the vehicle becomes zero which causes the highest attainable reward. At the end of the first stage, using the populated training dataset, the action-values with respect to states and actions and the policy is visualized in Fig. 3.11. As can be seen, stored information only covers the scope of training demonstrations.

In the second stage, several exploration cases are performed to gain knowledge about the state-action pairs that are not covered by the pilot. For this purpose, $\epsilon - greedy$ exploration is implemented. Explorations are performed by selecting random actions at 40% of the time while taking the current known best action at the remaining times. Figs. 3.12, 3.13, and 3.14, show the action-value function and the policy with increasing amounts of exploration. Beyond 5000 exploration moves, learning converges and the shape of policy no longer changes with more explorative actions. It

is obvious in the converged policy shown in Fig. 3.14 that linear state-action relation, which is seen in Table 3.1, can be achieved.

Mission execution is the third stage of LfD/RL method and can be used to judge the performance of the explorations. In order to compare the performances, undiscounted return expression in Eq. (2.1) is used. Figure 3.15 displays the return and state history for the different simulation cases which uses of the policies obtained after exploration runs. As can be seen, all cases performs better than the initially provided suboptimal demonstration.

3.5.2 Continuous State - Discrete Action case

Discrete state- discrete action case example shows the capability of Q-Learning and the LfD/RL framework. However, in the previous example, discrete states were used not only for the learning but also for the system states and state transitions as well. In this example, system states are kept continuous while only the learning uses discrete states by sampling and discretizing the continuous state at each learning update. Learning updates are performed at every 0.3 seconds due to discussion in Section 3.5 while the action is only applied for the first 0.1 second and then returned to zero until the next discrete update time. This is done in order to allow the velocity to settle to zero so that the position can represent the system state by itself as discussed above.

As the state and action space is completely identical to the discrete state - discrete action case, one question arises is whether the policies generated in the first example also works in the continuous state - discrete action setting as well. Figure 3.16 shows the answer of this question. As can be seen, it is possible to tracks several commanded target positions using the policies generated in the discrete state - discrete action case.

Second experiment in this case is to test if the same policy can be generated using the new state transitions with continuous dynamics are involved. The answer to this question is also seen to be true as shown in Fig. 3.17, for the final converged action-value and the policy plot. Similarly, Fig. 3.18 shows the mission executions of the increasing amounts of exploration runs and the performance results are exactly same as the discrete state-discrete action case.

3.5.3 Continuous State - Continuous Action case

In the third case, the same example case is performed when both states and actions are continuous. Main goal of this example is to investigate whether the satisfactory results of discrete/discretized states - discrete actions case can be obtained when no discretization is performed either on the states or the actions. As described in the previous section, for the continuous implementation, a function approximator is needed to replace the tabular form used for action-values. Simulation cases also aims to determine the effects of the parameters of the function approximator on the performance of the learning.

3.5.3.1 Action and learning updated every 0.1 s

In the first case, continuous state and actions are implemented by executing the learning at every 0.1 second. Actions are also kept constant until the next update time. Similar to previous cases, first the pilot demonstrations are recorded. Learning rate and discount rate are the same as the discrete state case while, for the function approximator, the kernel bandwidth is chosen to be $h = 5 \times 10^{-6}$. Demonstration target positions are the continuous valued equivalents of the discrete states as can be seen in Fig. 3.19. It can be seen that demonstrations are not able to reach the target perfectly due to the update rate of the action, thus there is a steady state error in

both demonstrations. At the end of the demonstrations, the action value and the policy generated can be seen in Fig. 3.20 with the discrete policy overlaid on top.

Exploration stage of the LfD/RL method is also applied in the same manner for the continuous case as can be seen in Figs. 3.21, 3.22 and 3.23. Unlike the discrete case, it can be seen that as the number of exploration moves increase, the policy and the action-value function do not take a converged form. This also can be observed from the simulation results in Figs. 3.24 and 3.25. In Fig. 3.24, previously demonstrated task is performed with the each policy obtained at the end of exploration runs. It can be seen that the initial policy which is obtained by combining two different demonstrations without any exploration performs better than the expert. However, unlike the discrete case, as the explorations increase, no improvement on the policy is seen. On the contrary, oscillations and steady state error are visible.

Fig. 3.25 shows the responses obtained by employing the policies for a previously undemonstrated target position. In this case, it can be seen that under the initial policy, vehicle starts moving towards the undemonstrated target, however becomes stuck before the demonstrated target position. As the number of explorations increase, resulted policies make it possible to reach the target position. However, there is no convergence observed with the increased number of exploration moves as the 500 exploration moves delivers a better performing policy than the 5000 exploration moves.

3.5.3.2 Action and learning updated every 0.3 s

In this case, learning update is performed at every 0.3 seconds and action is kept constant during this period. Pilot demonstrations are performed in the same way, as can be seen in Fig. 3.26. Initial policy that is obtained after the first stage of learning is shown in Fig. 3.27, and is the same as the previous case.

Second stage of the learning is also performed similarly and different policies are generated by applying different amounts of exploration which are shown in Figs. 3.28, 3.29 and 3.30.

After the explorations, generated policies are used to repeat the demonstration which was performed by the expert. Figure 3.31 shows the responses to the demonstrated target position. Initial policy is able to get rid of the steady state error of the expert response, however introduces too much overshoot. Once again policy obtained after 500 exploration moves exhibits the best response by getting rid of both the overshoot and steady state error. However, policy obtained after 5000 exploration moves does not improve the policy but worsens it by reintroducing steady state error.

Response to the undemonstrated target position which is shown in Fig. 3.32 has different outcome than the previous cases. In this case, initial policy is able to reach to the target position. Exploration policies starts faster than the expert, however they either settle to large steady state errors or goes in undamped oscillation around the target point. These behaviors decreases the discounted returns, therefore the initial policy is said to have the best performance overall.

3.5.3.3 Action applied for only 0.1 s. while learning updated every 0.3 s.

In this case, the learning update is performed every 0.3 seconds while the action is only applied for 0.1 seconds and taken back to zero for the remaining time in order to allow the speed to settle to zero. In this setting, when the expert demonstrations are performed in the same way, vehicle is able to reach the target perfectly as can be seen in Fig. 3.33. Initial policy displayed in Fig. 3.34 is exactly the same, moreover, the exploration cases shown in Figs. 3.35, 3.36 and 3.37 have similar behaviors with the previous continuous state continuous action cases.

Demonstrated target task results show that initial policy and the increasing number of explorations creates better performance as can be seen in Fig. 3.38. This suggests that if the correct state representation is, continuous state continuous action can perform similar to discrete state- discrete action case.

Similarly, undemonstrated target task also shows similar performance as can be seen in Fig. 3.39. However, this performance can not be fully attributed to the selection of the accurate state representation, as the parameters of the function approximator have a huge impact on the performance. As can be seen in Figs. 3.40 and 3.41, demonstrated and undemonstrated target tasks have completely different and unsatisfactory results when the kernel parameter h is chosen as a larger value.

Through this simple system example, it is demonstrated that LfD/RL does have nice performance and convergence characteristics if the system is represented by discrete state action pairs. However, same methods do not have these nice performance and convergence characteristics when applied to continuous state-action model.

This is attributed to several reasons. First of all, choice of the states should be appropriate for dynamics of the system. Discretization of the continuous states and actions through sampling should be appropriate for the choice of state. In addition, performance of LfD/RL depends heavily on the trajectories of the pilot demonstrations and the values of function approximator parameters.

Therefore, for the scope of this project, the aim is to find a constant kernel bandwidth parameter h, that will produce a policy to mimic expert response and the main objective is to use this obtained policy to do different maneuvers that are not directly shown by the expert.

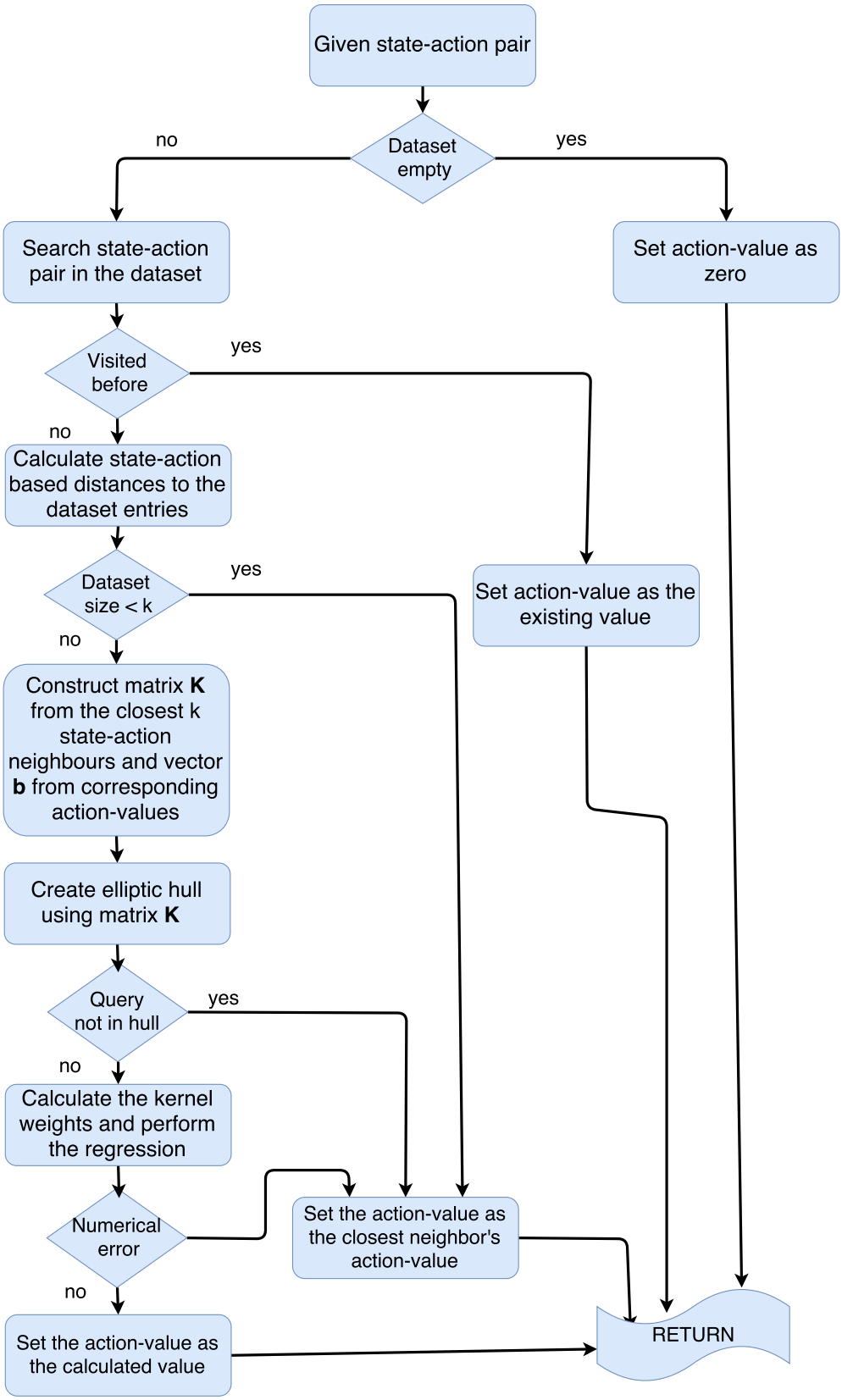


Figure 3.6. Flowchart of the action-value approximation subroutine.

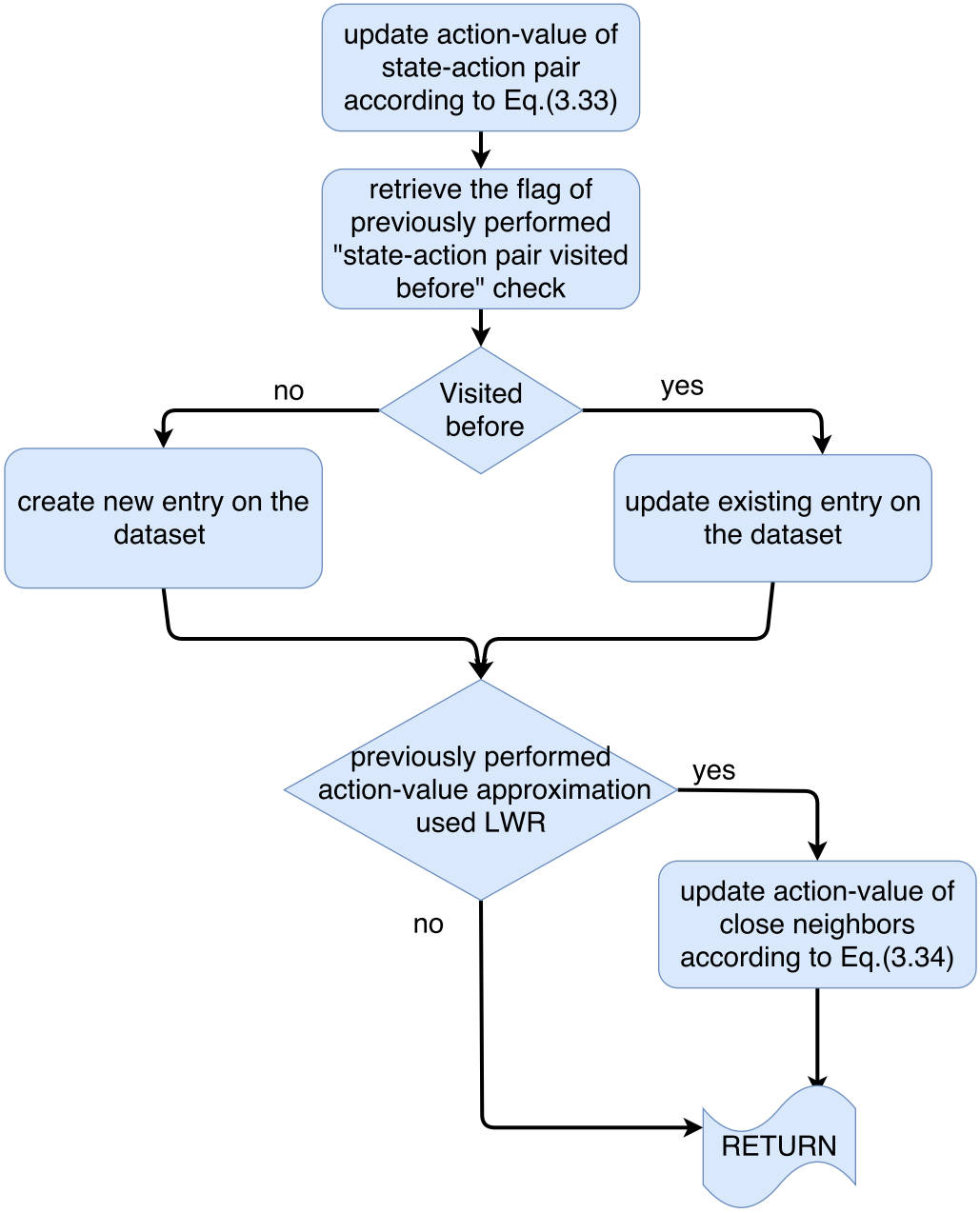


Figure 3.7. Flowchart for action value update subroutine.

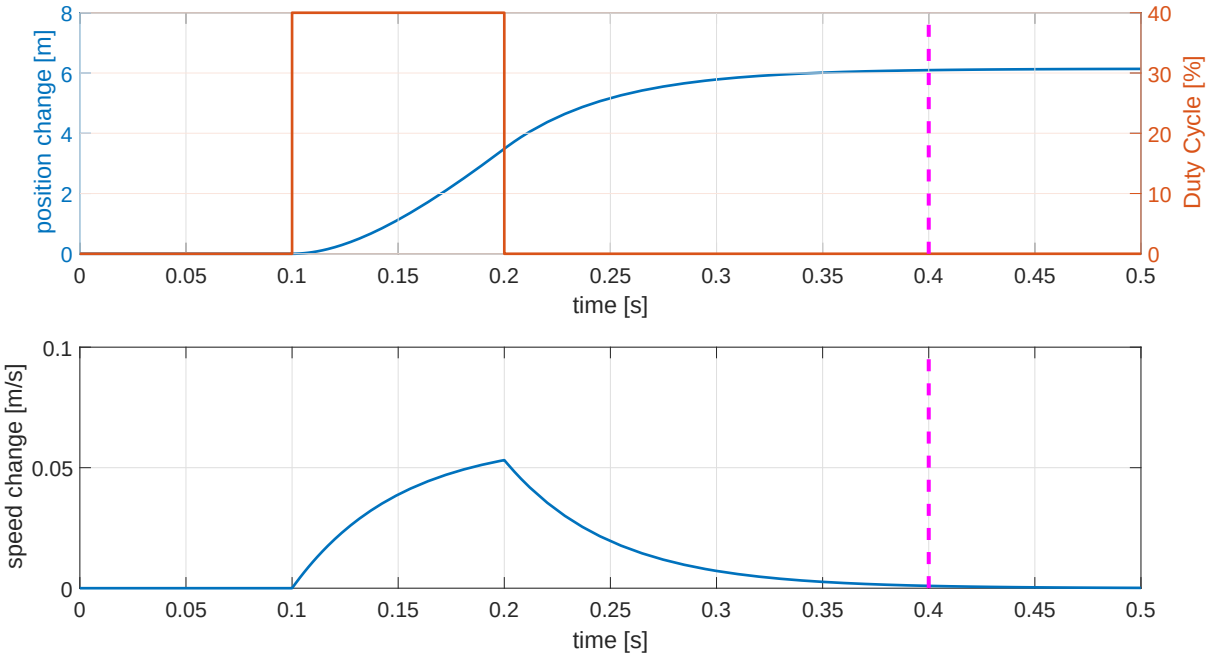


Figure 3.8. Change in position and speed when 40% duty cycle is applied.

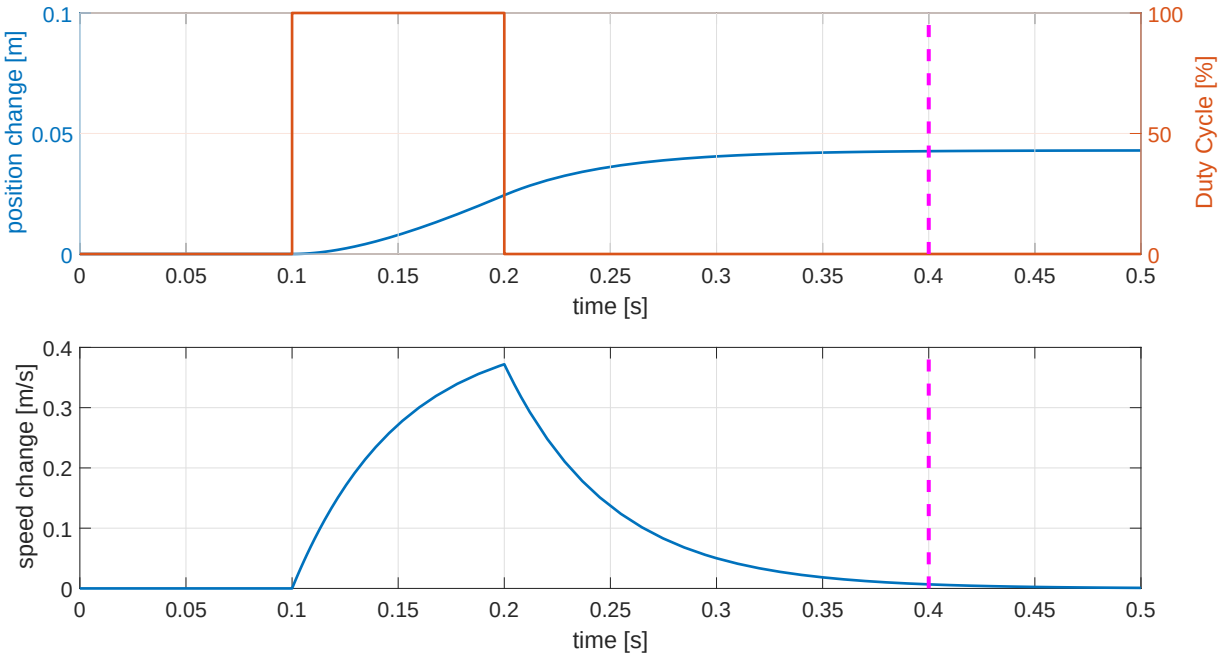


Figure 3.9. Change in position and speed when 100% duty cycle is applied.

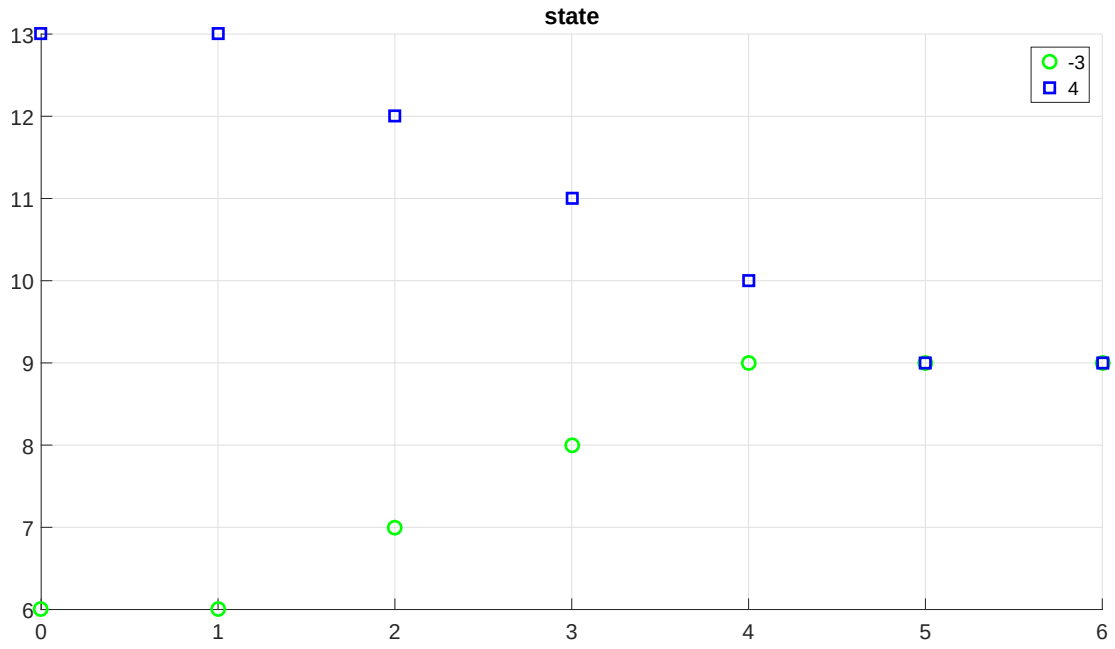


Figure 3.10. Demonstrations performed by the pilot.

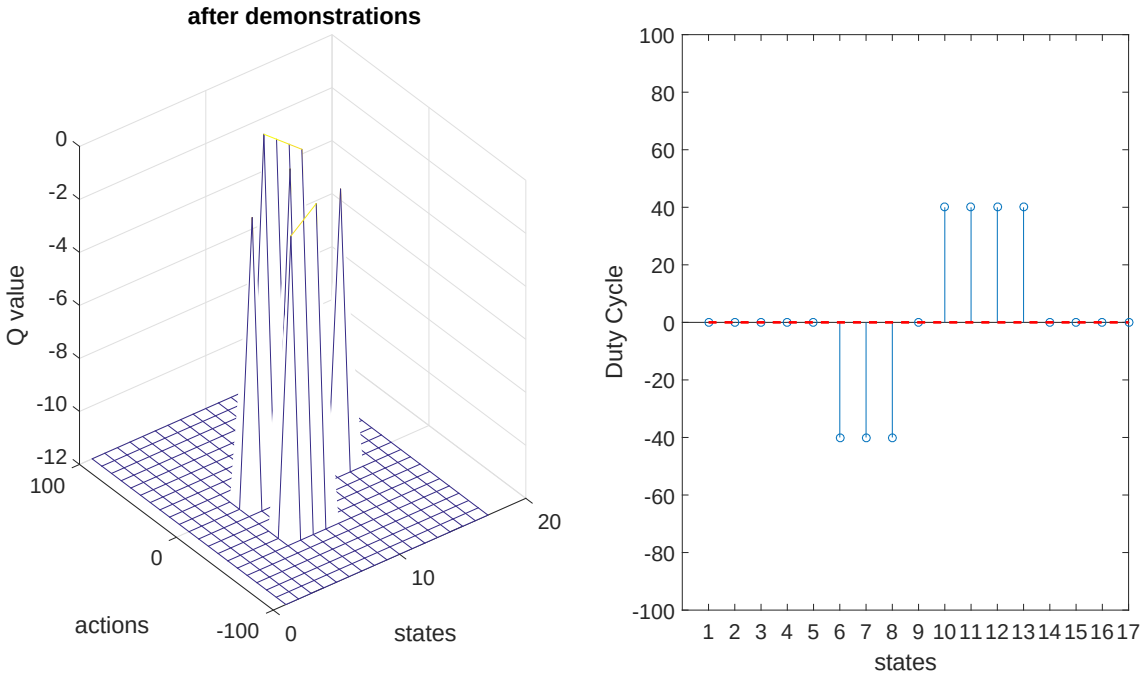


Figure 3.11. Action value and policy after the pilot demonstrations.

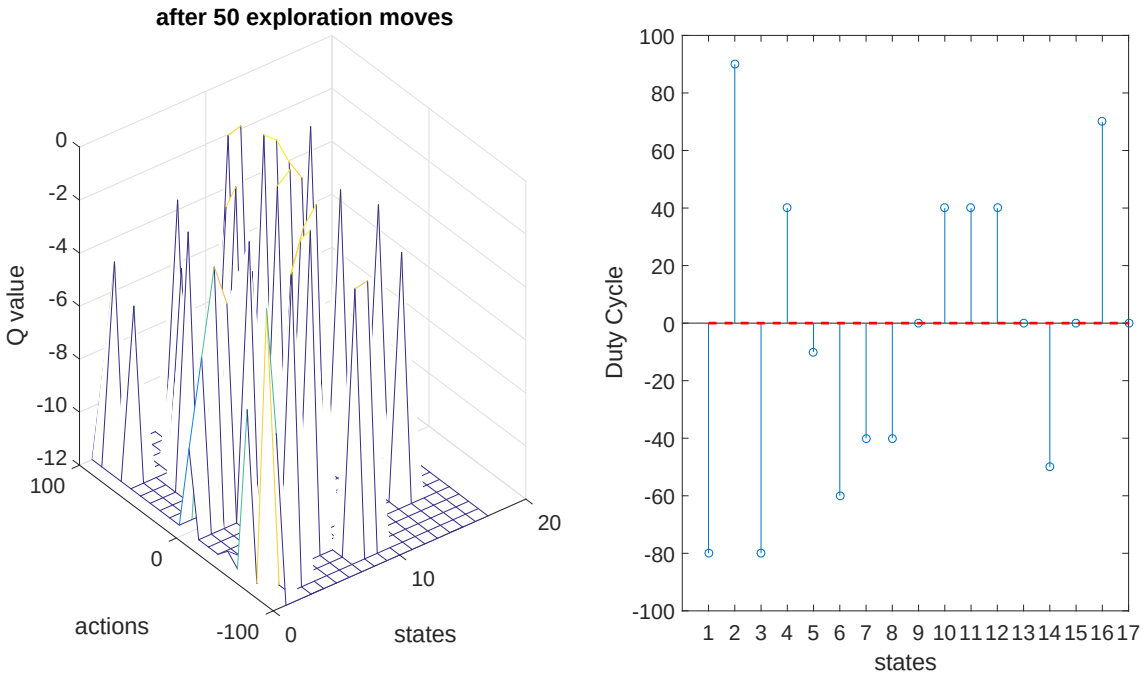


Figure 3.12. Action value and policy after 50 exploration moves.

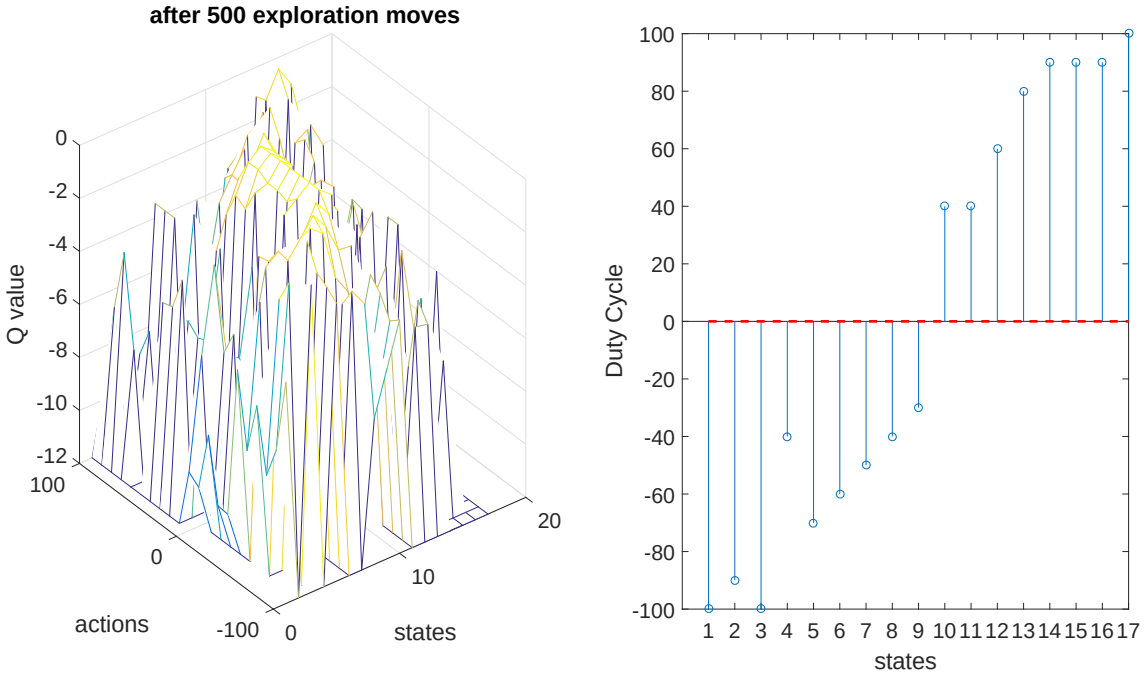


Figure 3.13. Action value and policy after 500 exploration moves.

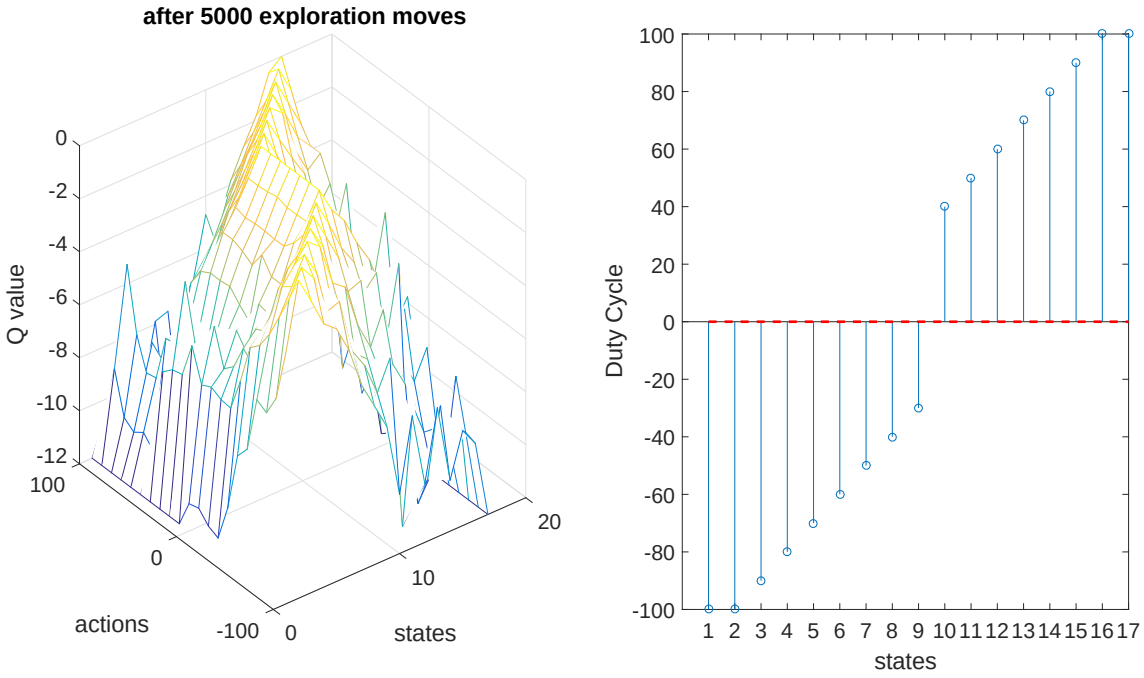


Figure 3.14. Action value and policy after 5000 exploration moves.

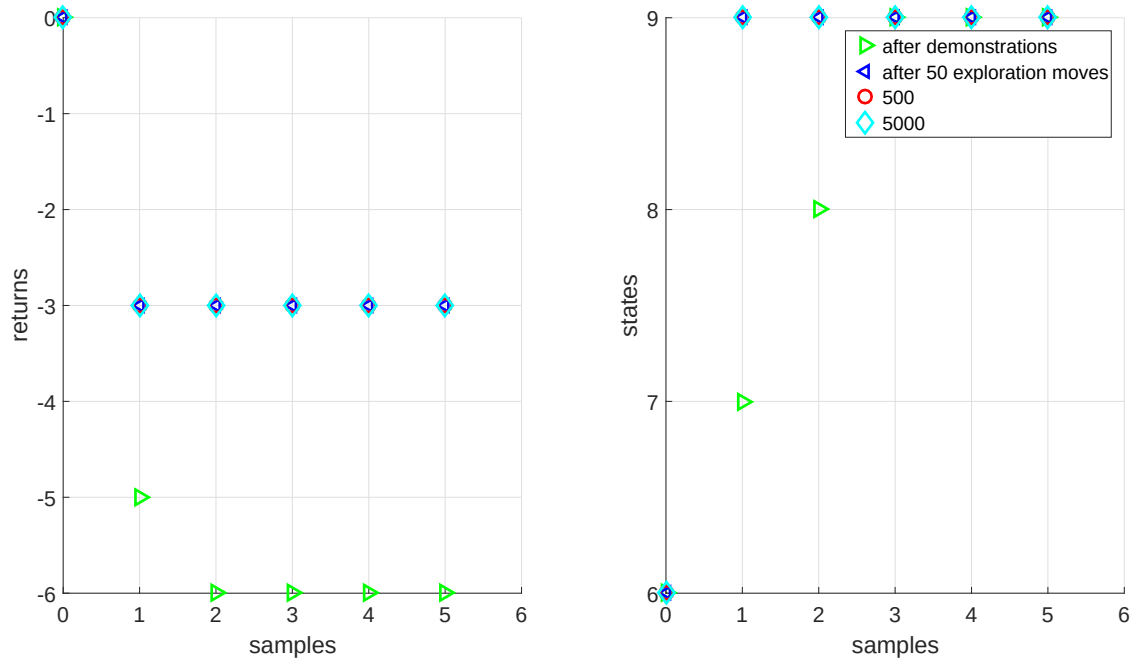


Figure 3.15. Comparison of exploration results for demonstrated target for the discrete state discrete action case.

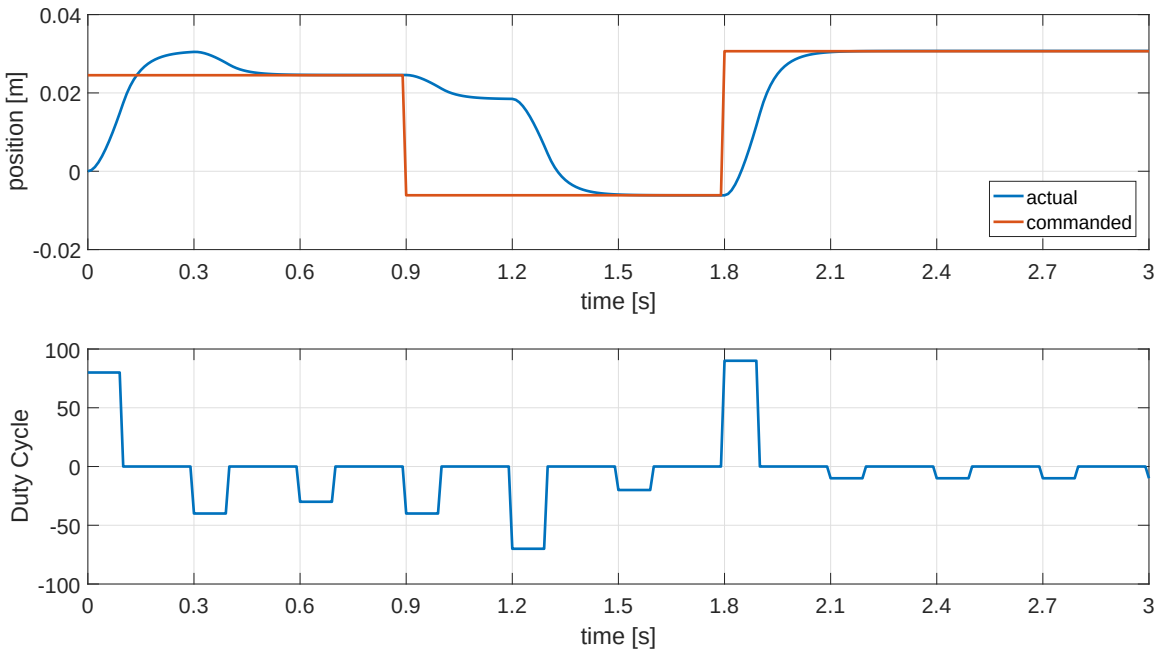


Figure 3.16. Result of using Q-values of discrete state discreet action case in continuous state discrete action case .

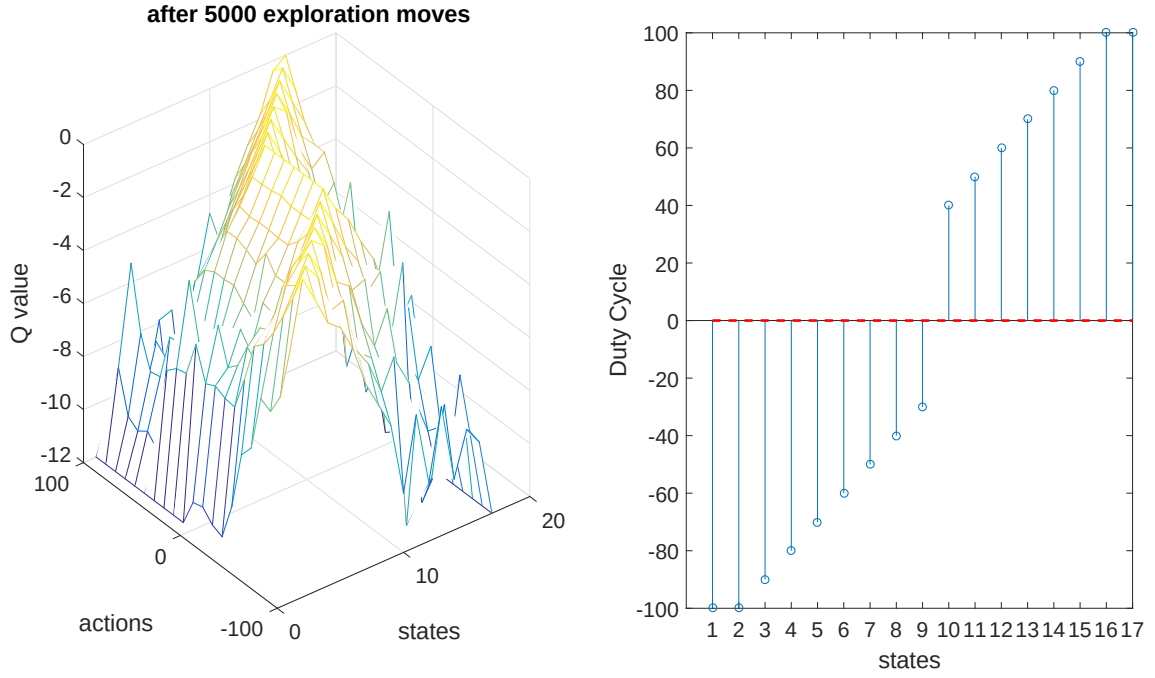


Figure 3.17. Action value and policy after 5000 exploration moves for continuous state discrete action case.

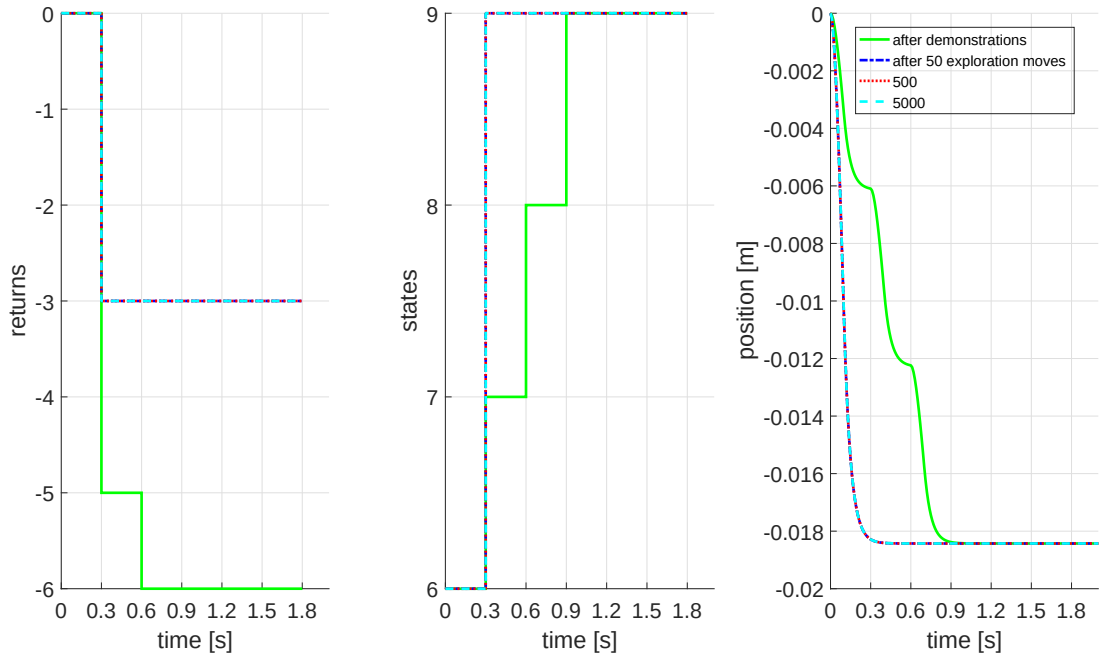


Figure 3.18. Comparison of exploration results for demonstrated target for the continuous state discrete action case.

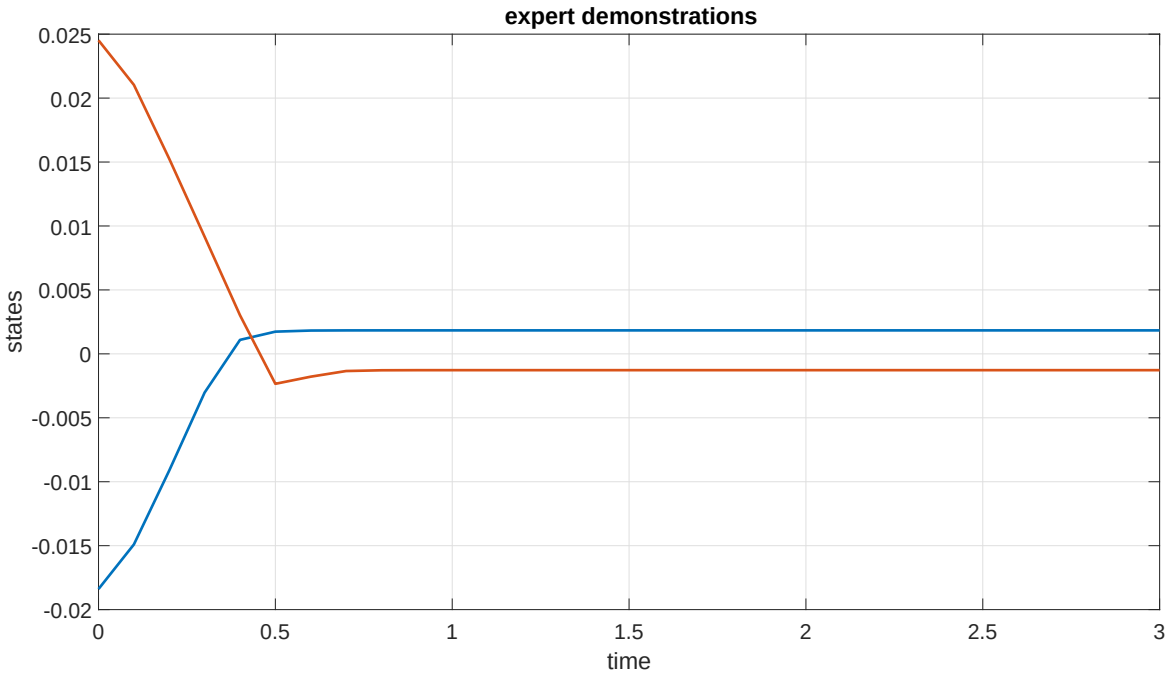


Figure 3.19. Demonstrations performed by the pilot in the continuous state continuous action case with action held constant for 0.1 seconds.

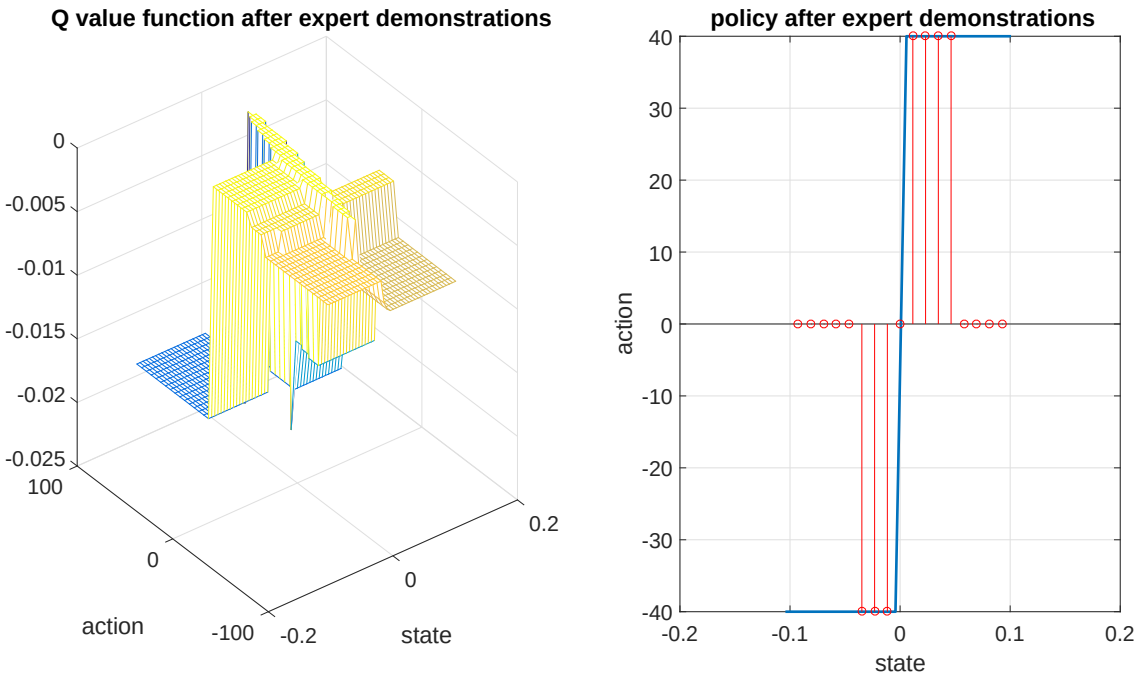


Figure 3.20. Action value and policy after the pilot demonstrations in the continuous state continuous action case with action held constant for 0.1 seconds.

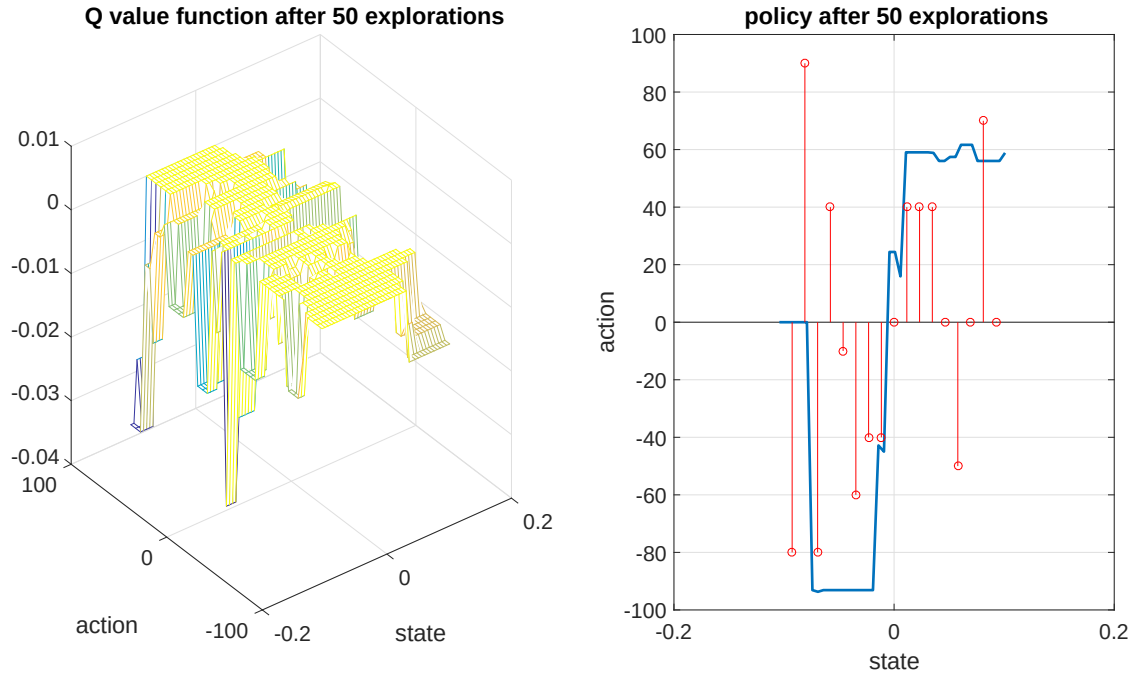


Figure 3.21. Action value and policy after 50 exploration moves in the continuous state continuous action case with action held constant for 0.1 seconds.

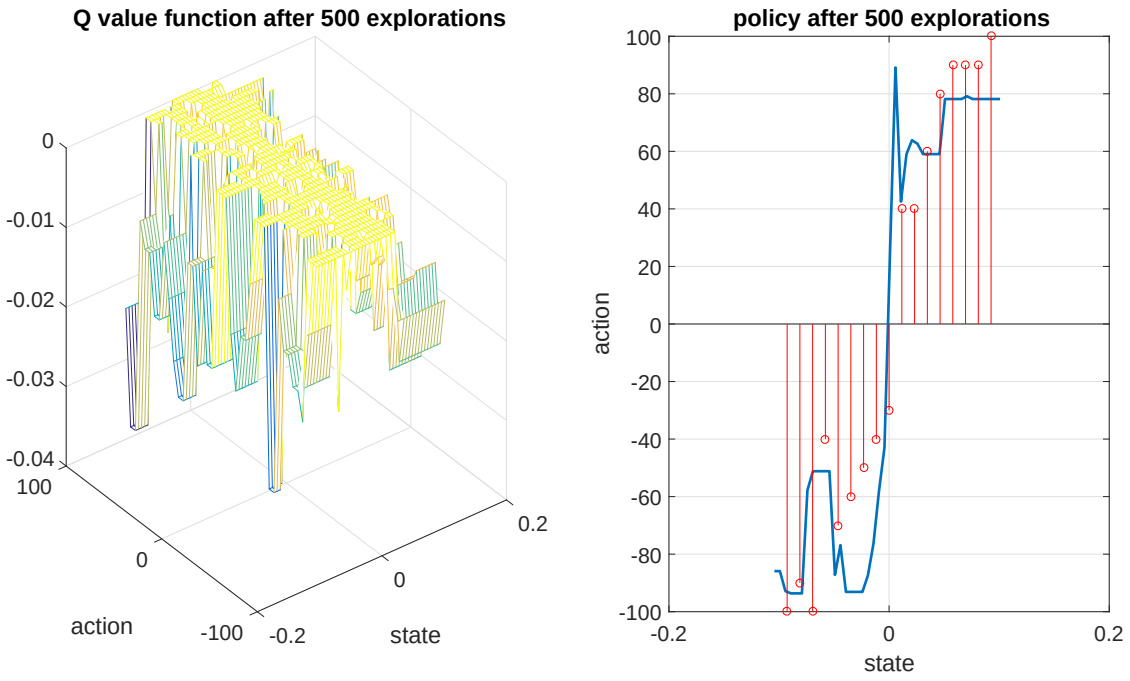


Figure 3.22. Action value and policy after 500 exploration moves in the continuous state continuous action case with action held constant for 0.1 seconds.

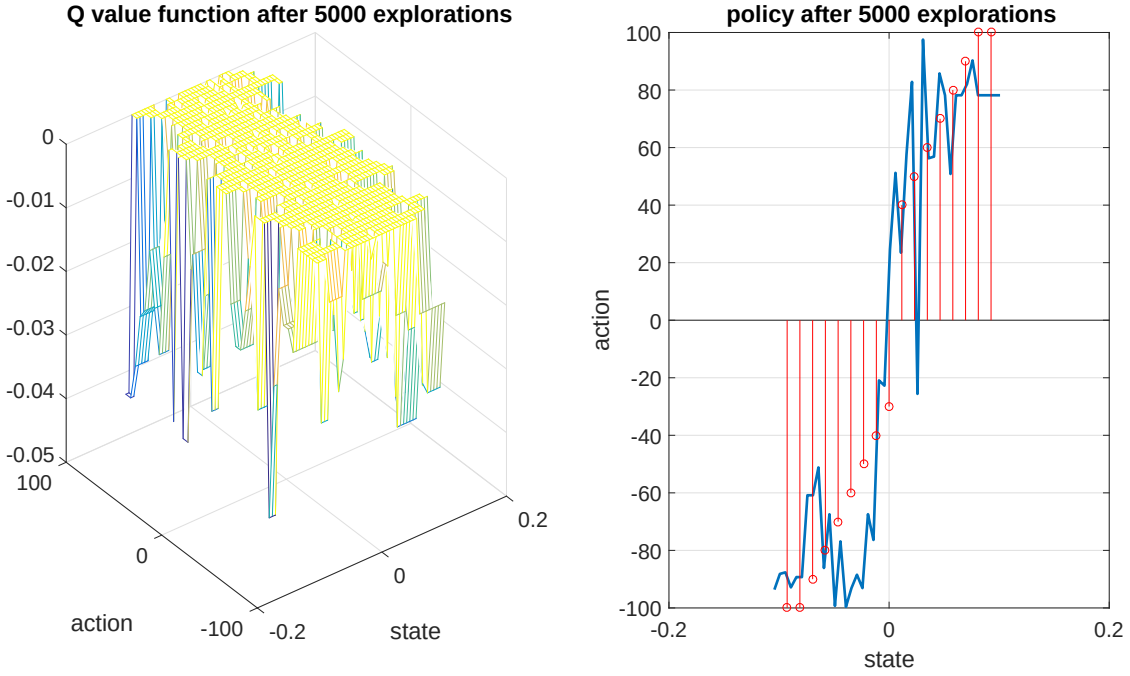


Figure 3.23. Action value and policy after 5000 exploration moves in the continuous state continuous action case with action held constant for 0.1 seconds.

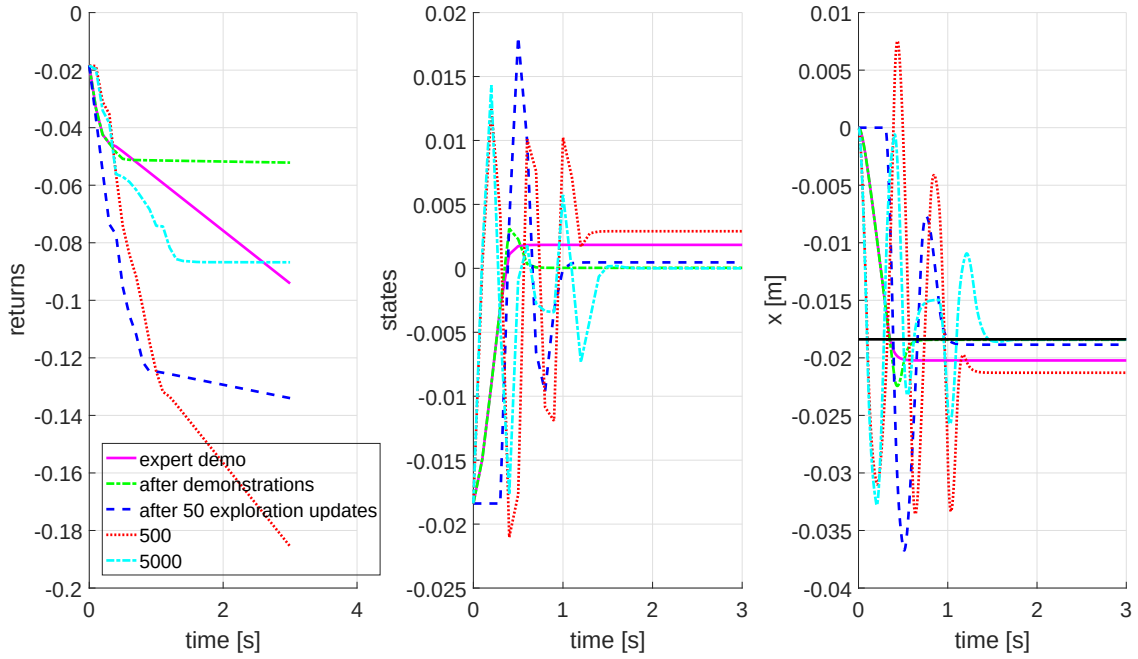


Figure 3.24. Comparison of exploration results for demonstrated target in the continuous state continuous action case with action held constant for 0.1 seconds.

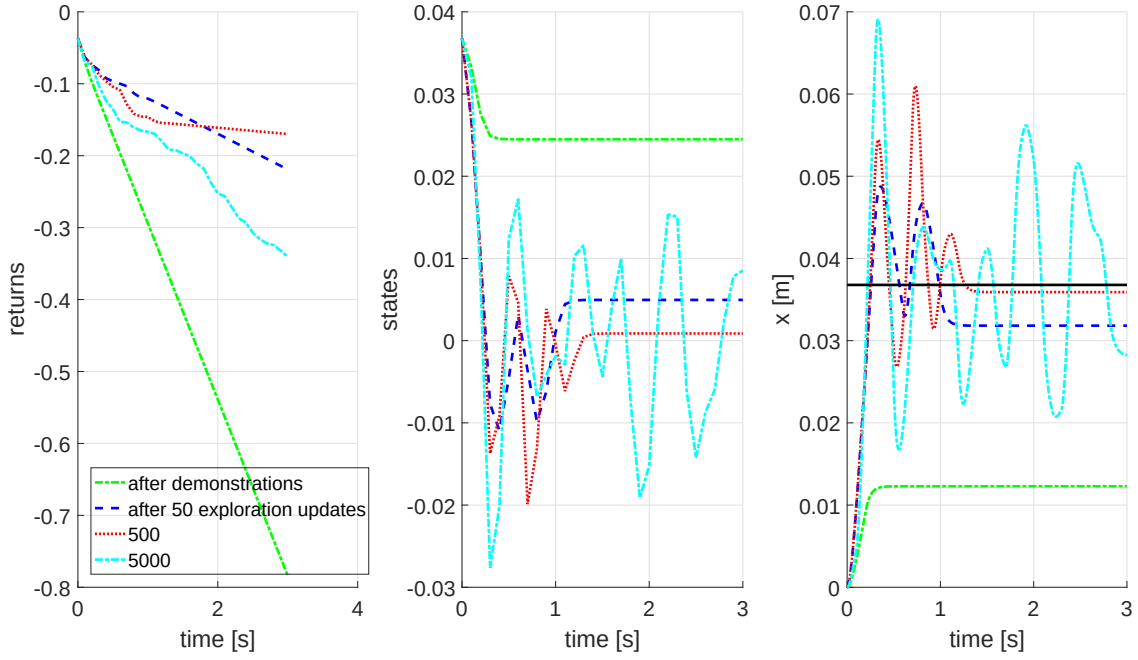


Figure 3.25. Comparison of exploration results for undemonstrated target in the continuous state continuous action case with action held constant for 0.1 seconds.

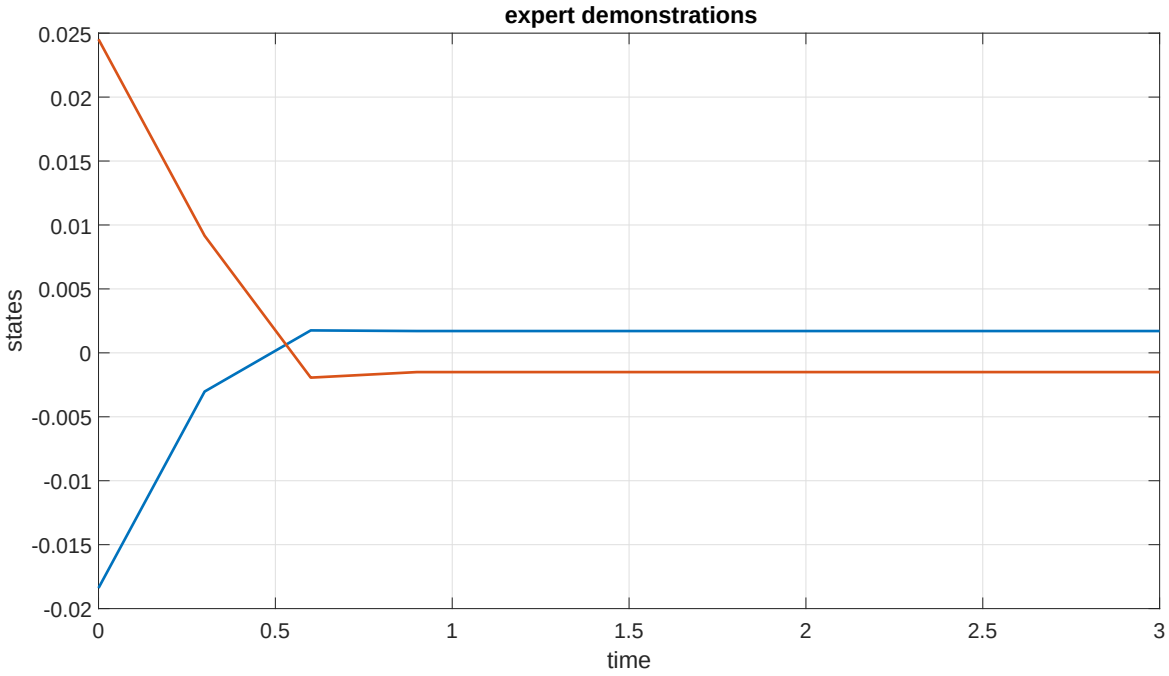


Figure 3.26. Demonstrations performed by the pilot in the continuous state continuous action case with action updated every for 0.3 seconds.

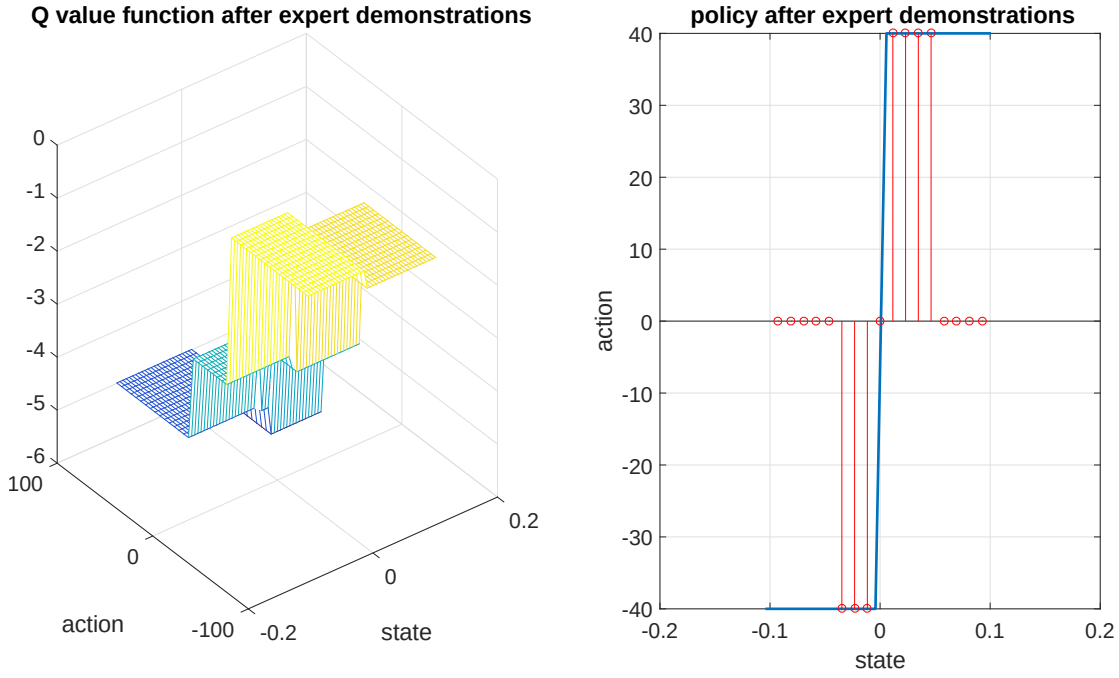


Figure 3.27. Action value and policy after the pilot demonstrations in the continuous state continuous action case with action updated every for 0.3 seconds.

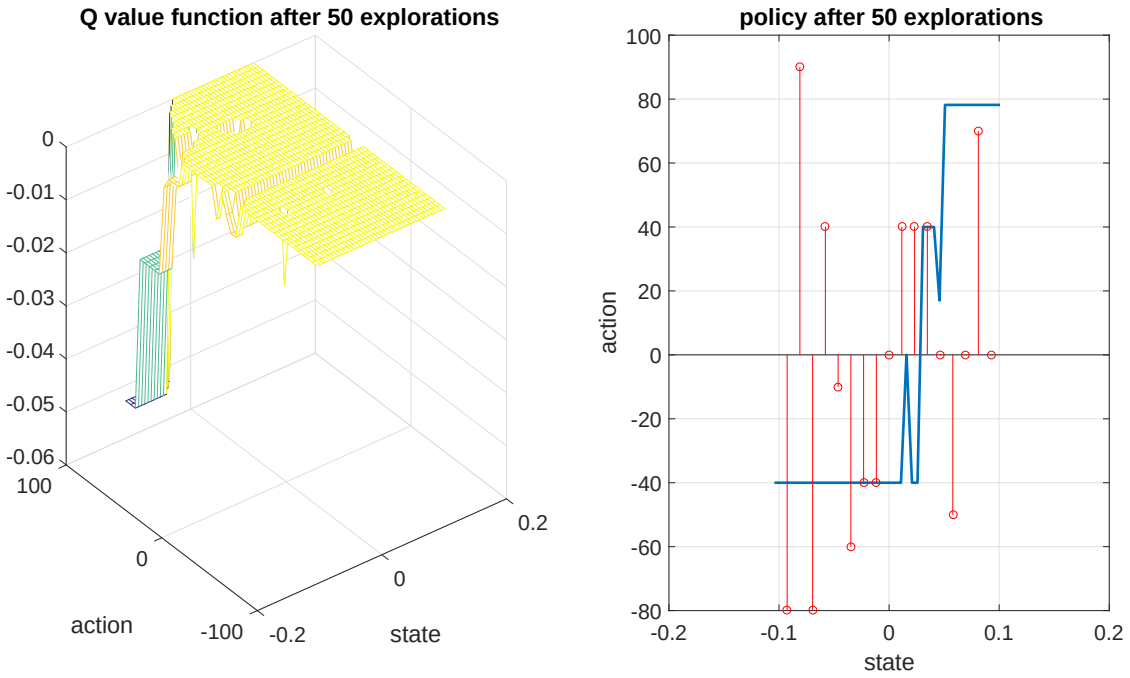


Figure 3.28. Action value and policy after 50 exploration moves in the continuous state continuous action case with action updated every for 0.3 seconds.

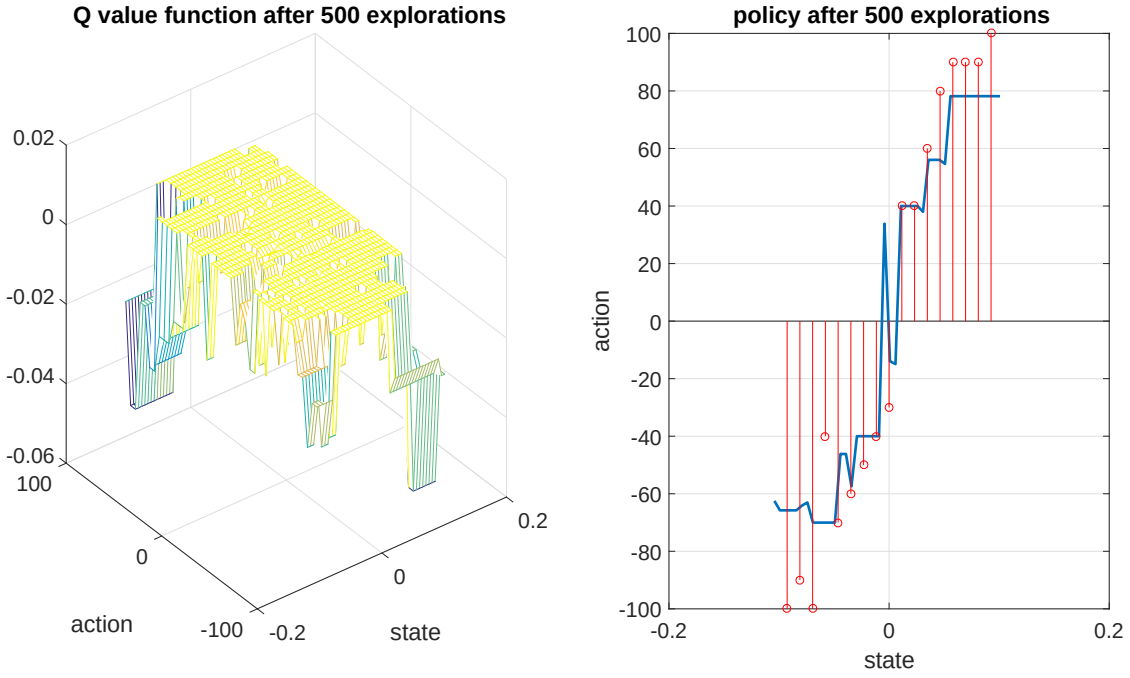


Figure 3.29. Action value and policy after 500 exploration moves in the continuous state continuous action case with action updated every for 0.3 seconds.

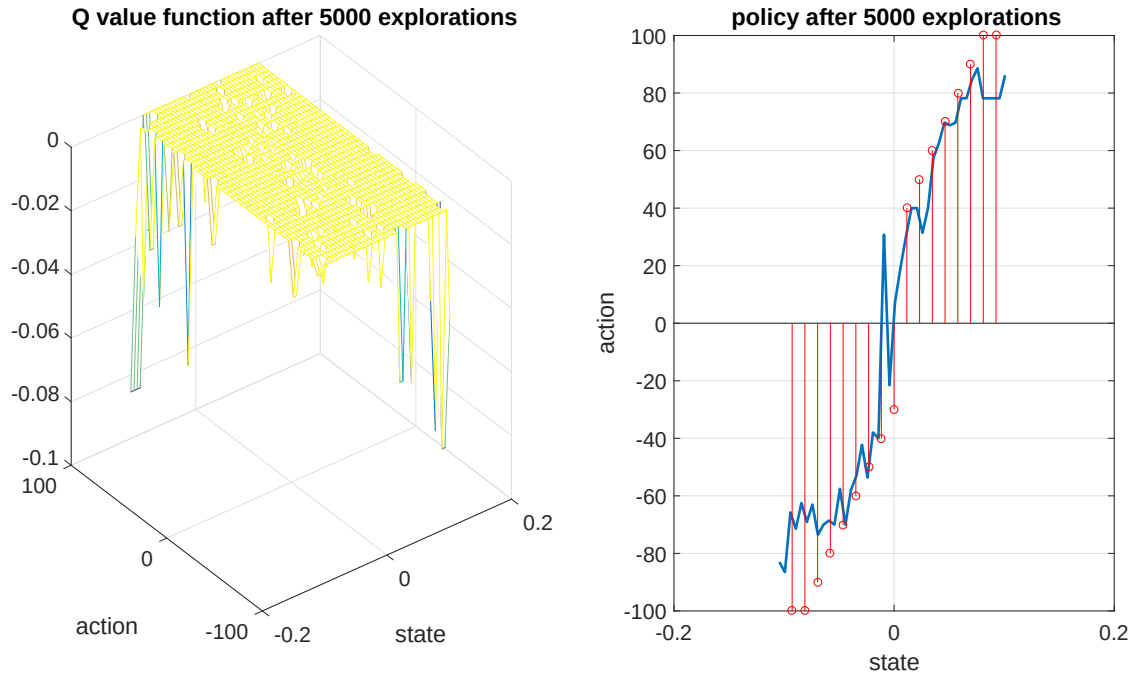


Figure 3.30. Action value and policy after 5000 exploration moves in the continuous state continuous action case with action updated every for 0.3 seconds.

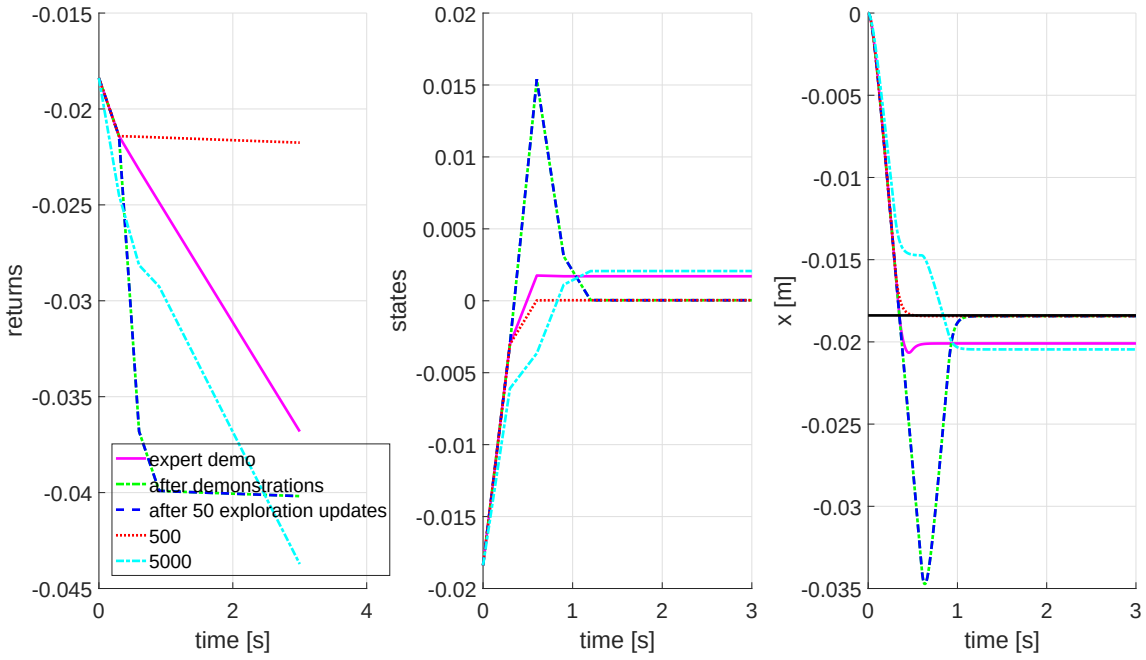


Figure 3.31. Comparison of exploration results for demonstrated target in the continuous state continuous action case with action updated every 0.3 seconds.

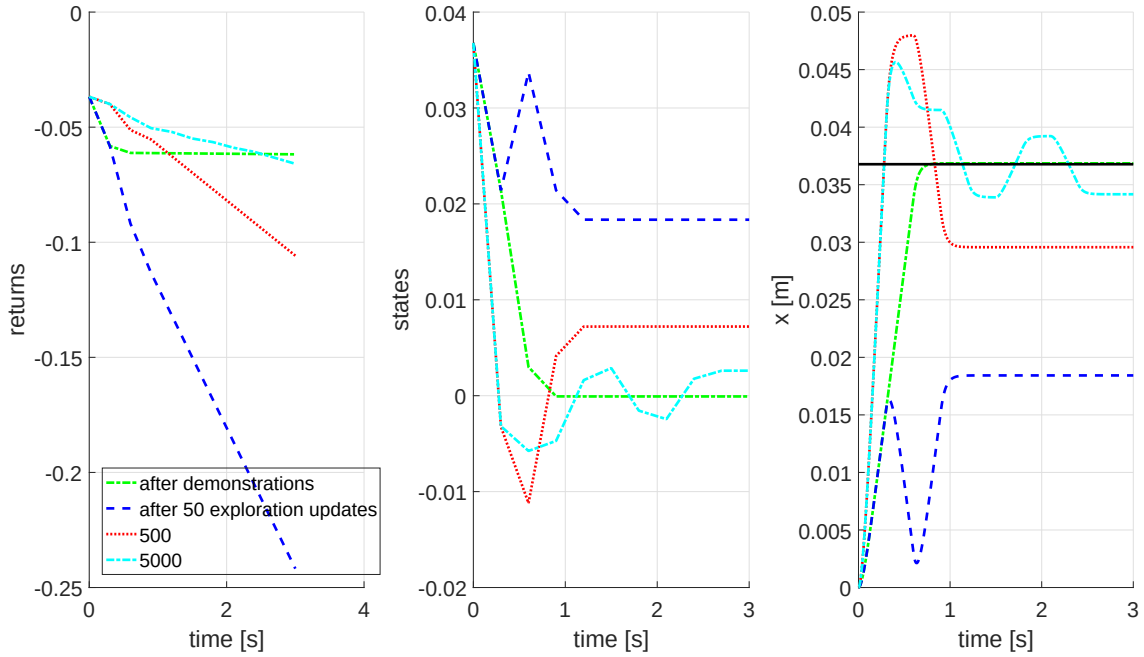


Figure 3.32. Comparison of exploration results for undemonstrated target in the continuous state continuous action case with action updated every for 0.3 seconds.

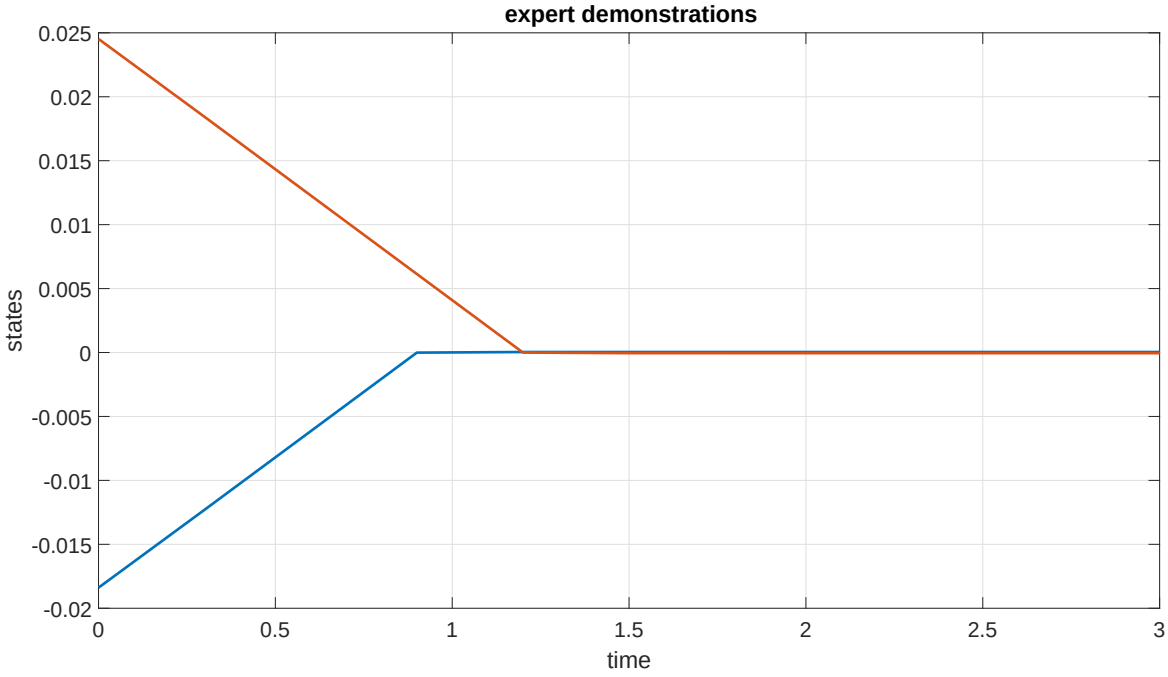


Figure 3.33. Demonstrations performed by the pilot in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

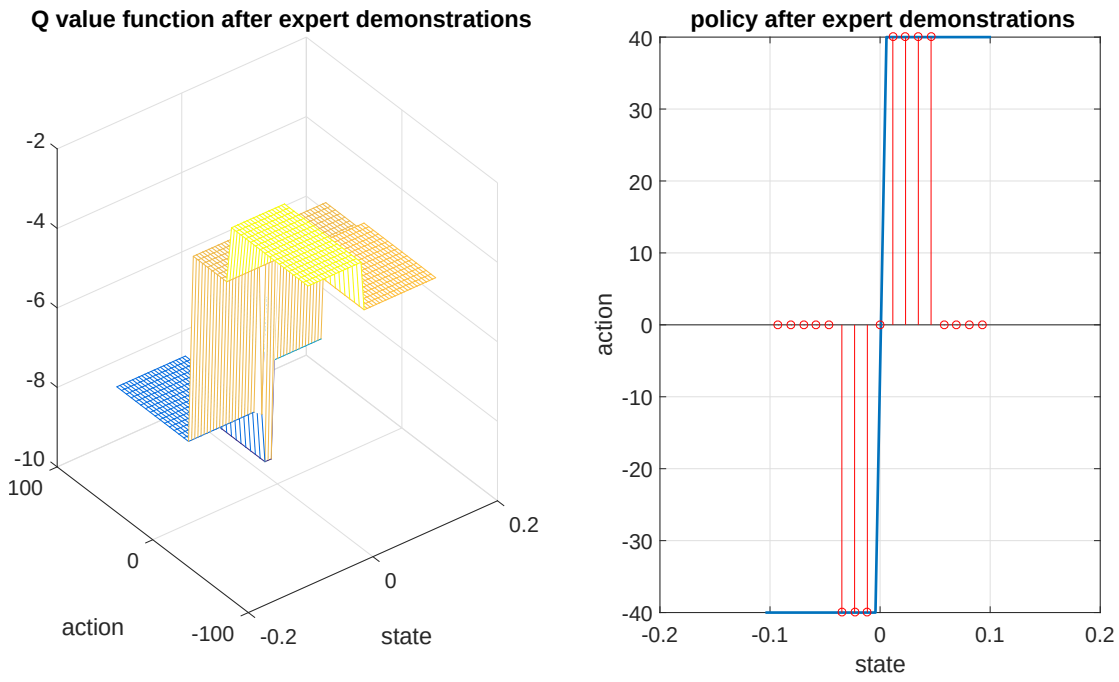


Figure 3.34. Action value and policy after the pilot demonstrations in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

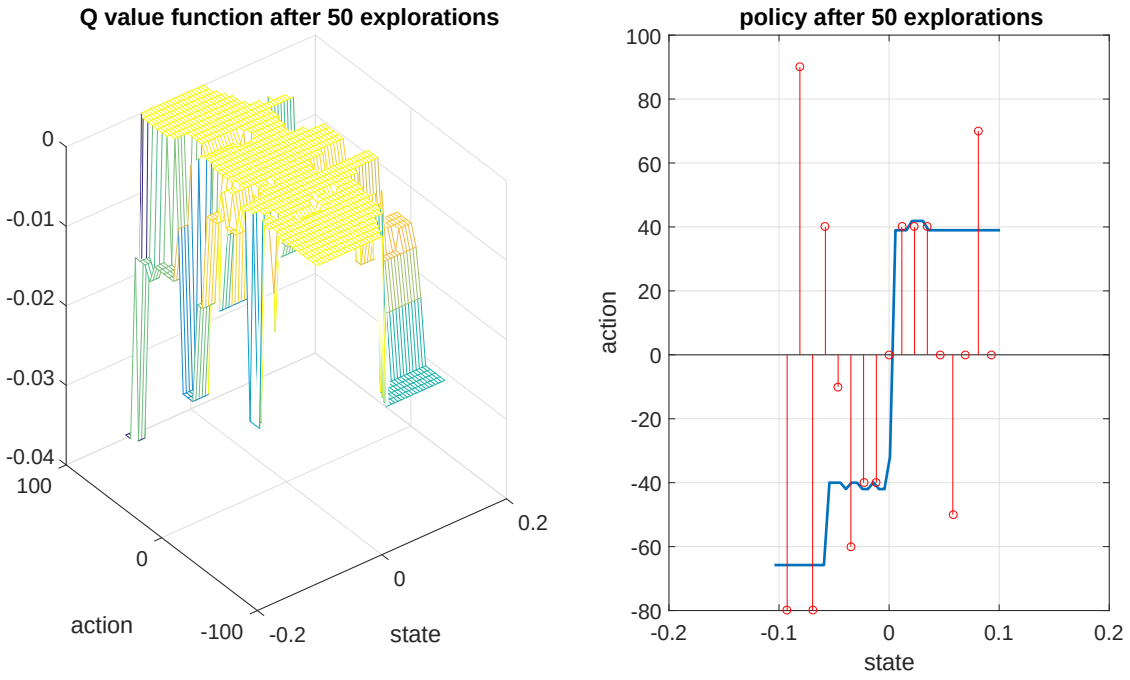


Figure 3.35. Action value and policy after 50 exploration moves in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

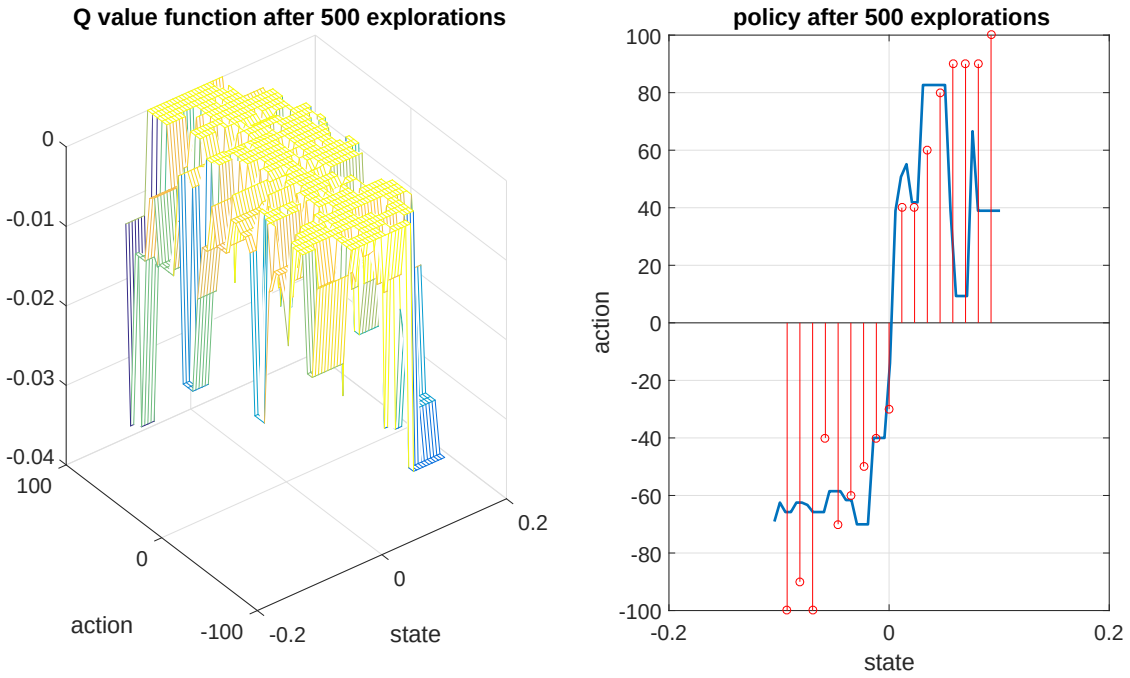


Figure 3.36. Action value and policy after 500 exploration moves in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

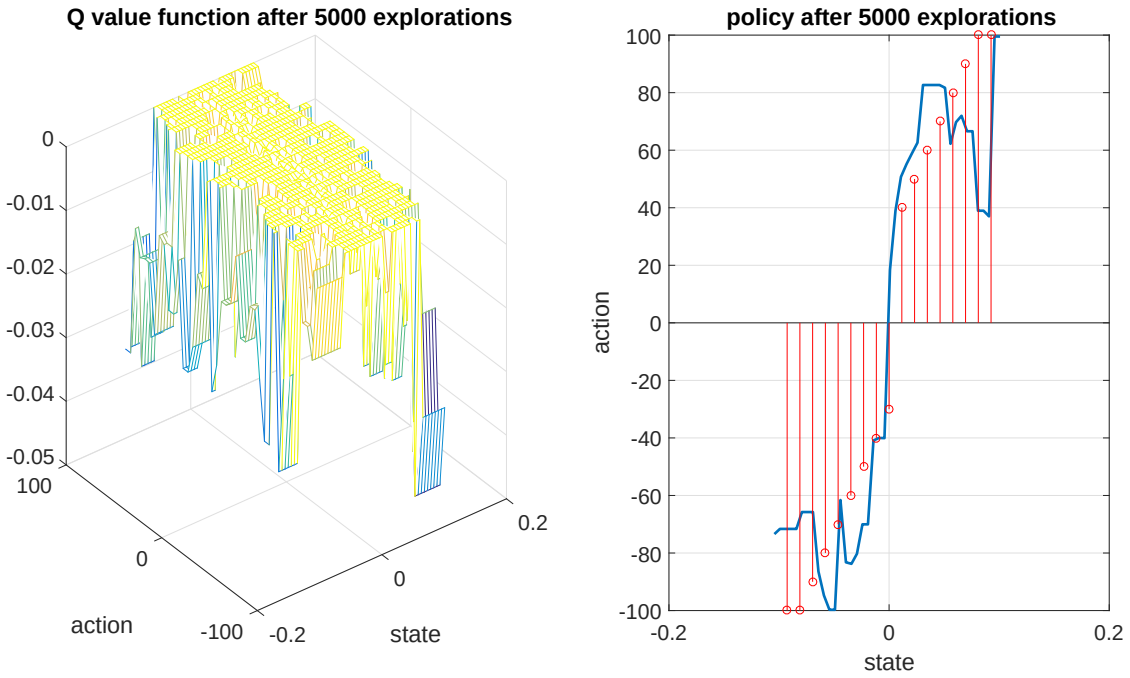


Figure 3.37. Action value and policy after 5000 exploration moves in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

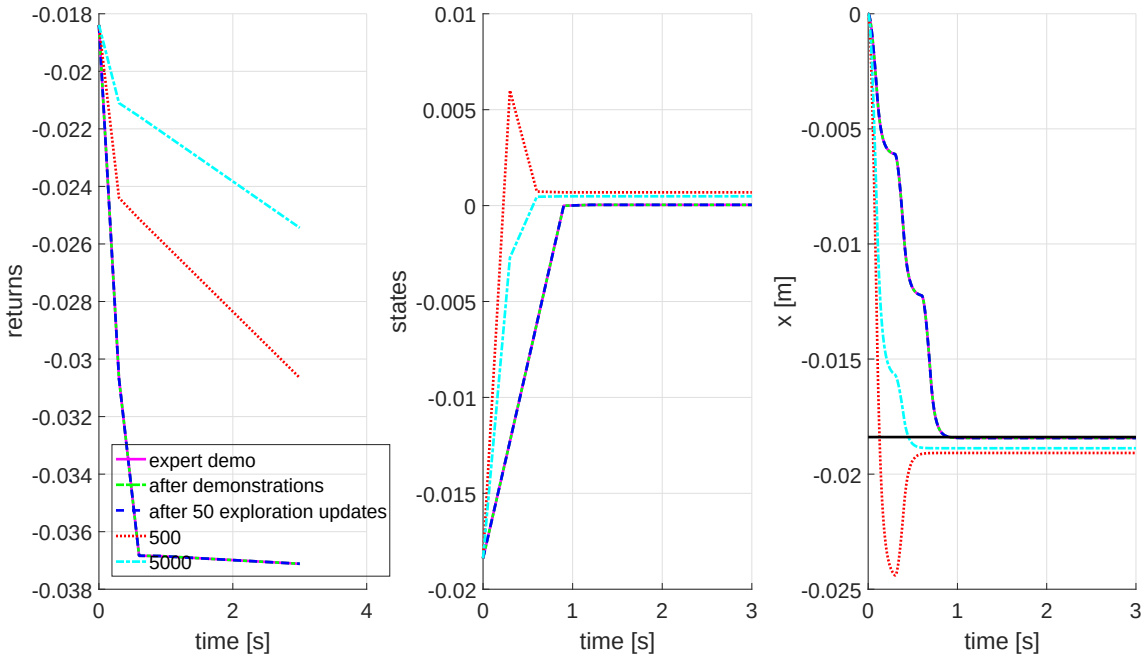


Figure 3.38. Comparison of exploration results for demonstrated target in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

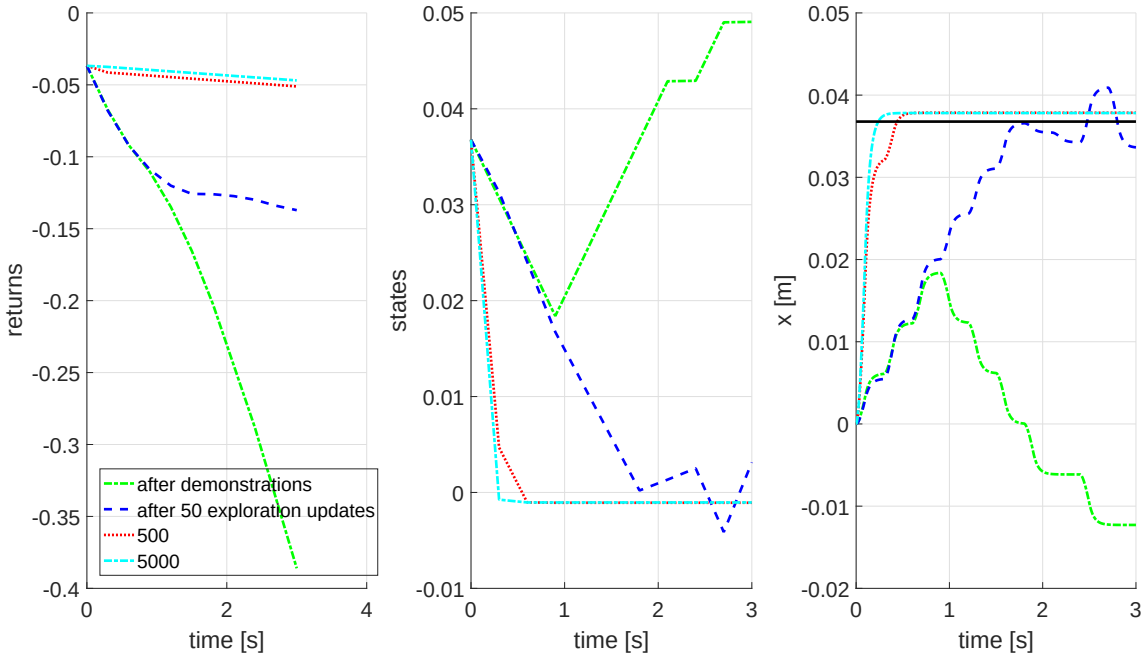


Figure 3.39. Comparison of exploration results for undemonstrated target in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds.

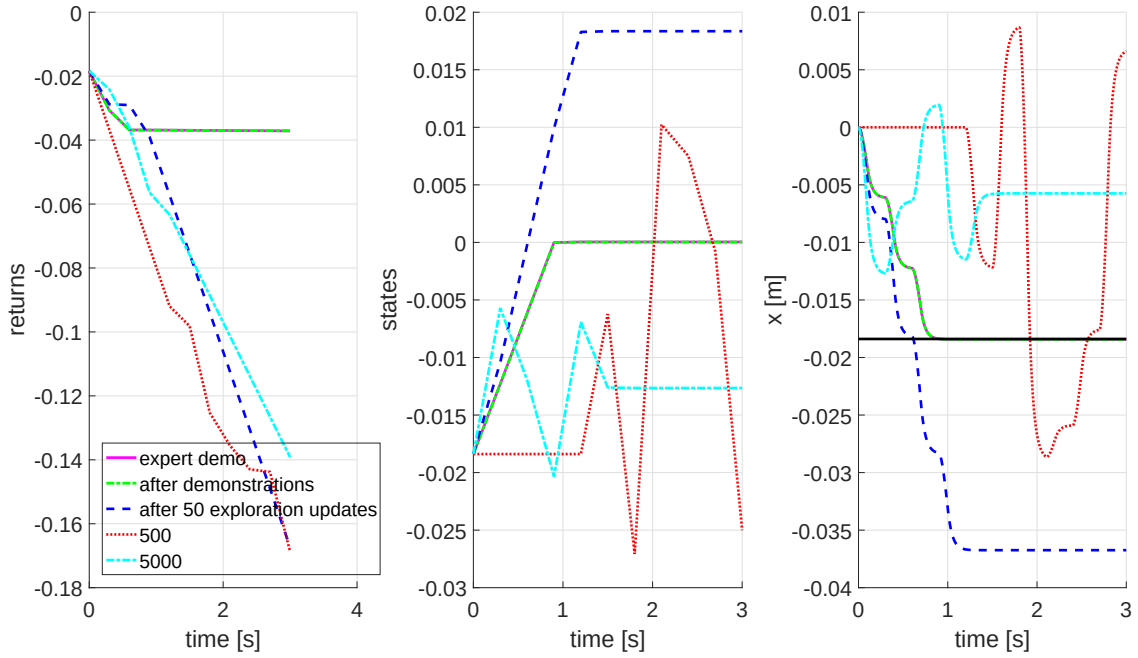


Figure 3.40. Comparison of exploration results for demonstrated target in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds with different parameters.

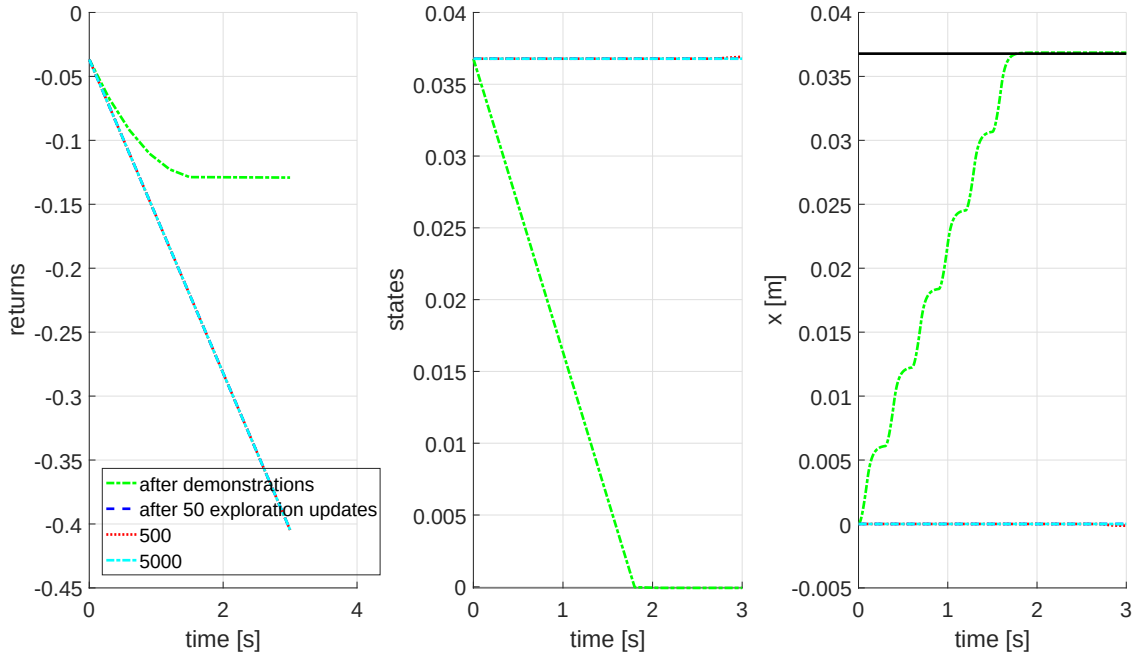


Figure 3.41. Comparison of exploration results for undemonstrated target in the continuous state continuous action case with action updated every 0.3 seconds but applied for the first 0.1 seconds with different parameters.

CHAPTER 4

Experiments on the UGV platform

In the previous sections, an LfD/RL robotic learning module that uses expert demonstrations as initial knowledge was introduced. It is aimed to test the capabilities of the algorithm on unmanned vehicles. For this reason, dynamics of the vehicles are simplified to allow to be operated by a single action. Two unmanned vehicle platforms are used as testbeds. Although the overall objective is to control an airship, an unmanned ground vehicle is used as the first test platform since less complex dynamics allows a better observation of the performance of the learning module. In addition, it is easier to operate and create demonstrations and implement the learning algorithm on the actual vehicle.

4.1 Experiments using single action

The UGV platform is a small, tracked skid-steer differential drive platform, that is driven by two conventional brushed DC motors [89, 90] (see Fig. 4.1). As the vehicle has 3-degrees of freedom but only two control inputs, it is an underactuated non-holonomic vehicle. UGV is equipped with encoders that measure the angular

Figure 4.1. Unmanned Ground Vehicle (UGV) Platform.

displacement of the gearbox output shaft, that is used to obtain the angular speed of the wheels. Due to the mechanical imperfections in the drive train, when both motors are given identical inputs, the platform does not move on a straight line which brings the need for actively controlling the platform if consistent autonomous motion behaviors are desired. The platform is controlled by a small form factor PC with 16 GB RAM and Intel i5 processor that can run Matlab/SIMULINK. The platform is also modeled in the software environment based on kinematic equations [90]. Since the actual inputs of the system are the duty cycles of the brushed motors, a look up table is used to map the angular velocity to the duty cycle input. Since the vehicle has tracks, the slip of the tracks is also modelled in the equations. The equations that govern the kinematics of the vehicle calculate the translational and angular velocity based on the angular velocities (ω_R, ω_L) of each wheel, orientation (θ) of the platform, and slippage coefficients (s_R, s_L) on each side.

$$\dot{x} = \left((1 - s_R) r_R \omega_R + (1 - s_L) r_L \omega_L\right) \frac{\cos(\theta)}{2} \tag{4.1}$$

$$\dot{y} = \left((1 - s_R) r_R \omega_R + (1 - s_L) r_L \omega_L\right) \frac{\sin(\theta)}{2} \tag{4.2}$$

$$\dot{\theta} = \left(\left((1 - s_L) r_L \omega_L - (1 - s_R) r_R \omega_R\right)\right) \frac{1}{b} \tag{4.3}$$

where b is the reference length of the platform, which can be estimated by experiments. By integrating $\dot{x}$, $\dot{y}$, and $\dot{\theta}$, we get x, y and θ, as the current position and orientation of the vehicle. The delays between changes in duty cycle inputs and changes in translational and angular speeds of the platform are due to the dynamics of the UGV components such as the electric motors and gear backlash. These components are represented simply by two first order transfer functions on ω_L and ω_R before they are fed into the above equations.

In the preliminary case study, the objective is to test the capability and applicability of the algorithm. For this reason, only the forward/backward motion of the

vehicle when both motors are applied a single control input is considered. Sign of the input determines the rotation direction of the motors and magnitude determines the duty cycle that will be applied. For the simulations, slippage is assumed to be zero. Demonstrations are generated by using the GNC system in [90]. First, the vehicle is commanded to go to several single waypoints and the training dataset is constructed by these expert demonstrations. Then, the training datasets are combined and sparsified to create the initial dataset that will be used in the second part of the learning.

4.1.1 Simulation Results

Expert demonstrations are created with a full GNC solution for the desired waypoints on xy-plane passed as parameters [90]. Expert is considered as the Guidance and the Controller blocks that are located inside the black dashed rectangle in Fig. 4.2. During the first stage of learning, learning module observes the inputs from the expert and the outputs of the vehicle, and creates the training dataset. In the second stage, the learning module replaces the expert and performs the desired tasks by itself. In order to create demonstrations of only forward and backward motion, single waypoints on the x-axis are considered. For learning, forward x-axis displacement Δx, speed, V are the states and the mean of duty cycles applied left and right motors, a, is the action that were used as the state-action variables. Throughout the demonstration, action-values of state action pairs are calculated with the methods described in previous sections with a learning rate of $\alpha = 0.45$ and quadratic reward function

$$r(\Delta x, V) = -\left(\Delta x^2 + 0.2V^2\right) \tag{4.4}$$

that gives negative rewards when vehicle is away from the desired point and the highest reward can be achieved is zero when it comes to a full stop at the desired

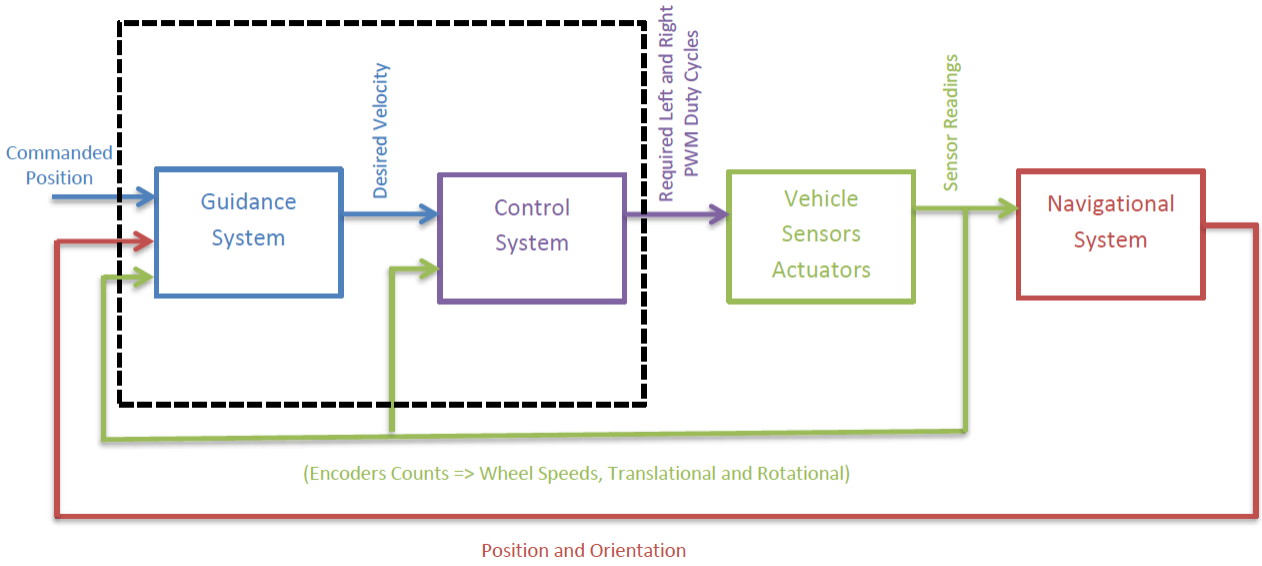


Figure 4.2. Description of the GNC system.

point. The GNC algorithm, by design, computes the heading angle command to turn the vehicle towards the assigned waypoint and then moves the vehicle straight towards it. This means that when a waypoint is right behind the vehicle, the GNC first turns the vehicle towards the waypoint instead of moving it backward. Since in this preliminary case only a single action is considered, demonstration of turning of a differential drive vehicle cannot be captured. Due to this reason, demonstrations of only forward motion is generated. GNC system requires at least two waypoints to be provided. Two demonstrations shown in Fig. 4.3 were performed. In the first case, the UGV is commanded to go to waypoint (1,0) and then (5,0) in meters in xy-frame. In the second case, waypoints are (1,0) and (2.5,0) in order. Two additional demonstration runs are intended for the symmetric waypoints in the negative x-axis. Since this preliminary learning module uses only one action, the cases where the UGV moves backward to go to the waypoints behind instead of turning are generated by reversing the signs of Δx and a while keeping the same action-values. Forward and

(a) (b)

Figure 4.3. Expert demonstrations for UGV in the simulation environment (a) 5m forward (b) 2.5m forward.

backward demonstrations are combined to create the whole demonstration set.

Dataset is sparsified to remove the redundant entries based on the closeness of the samples. Samples are considered to be close if the weighted Euclidean distance between them is less than 0.001 where the weightings are selected same as the reward function in Eq. (4.4). Elements of the raw and sparsified dataset are depicted as red and green, respectively in Fig. 4.4. As can be seen, the red entries that are not overlapped by green entries are considered as redundant, as the same value approximation can be performed without them.

In the second part of the study, G-C subsystems of the GNC module are replaced by the "learning module", employing the training dataset gathered and sparsified during the previous run with the GNC. Different cases are simulated to evaluate the performance of learning module. In the simulation cases, no exploration was considered and vehicle employs action selection algorithm based on the data from the demonstration dataset.

In the first simulation case, vehicle is given the waypoints of the first demonstration case. In Fig. 4.5, trajectories of expert and the learner while performing

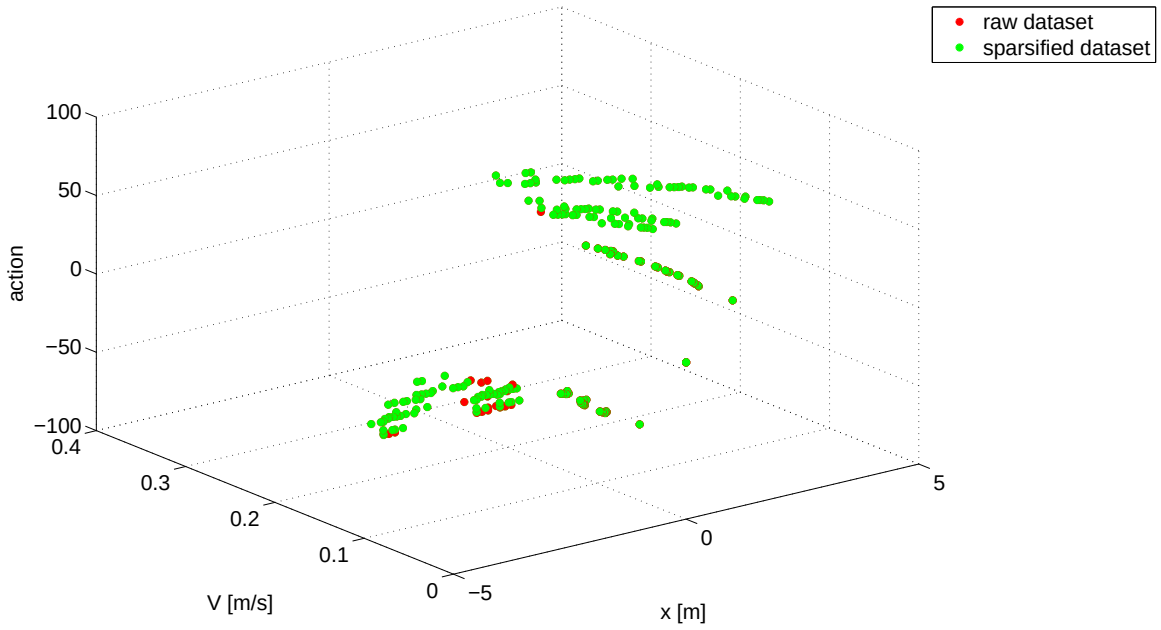


Figure 4.4. Sparsified expert demonstrations for UGV in the state-action space.

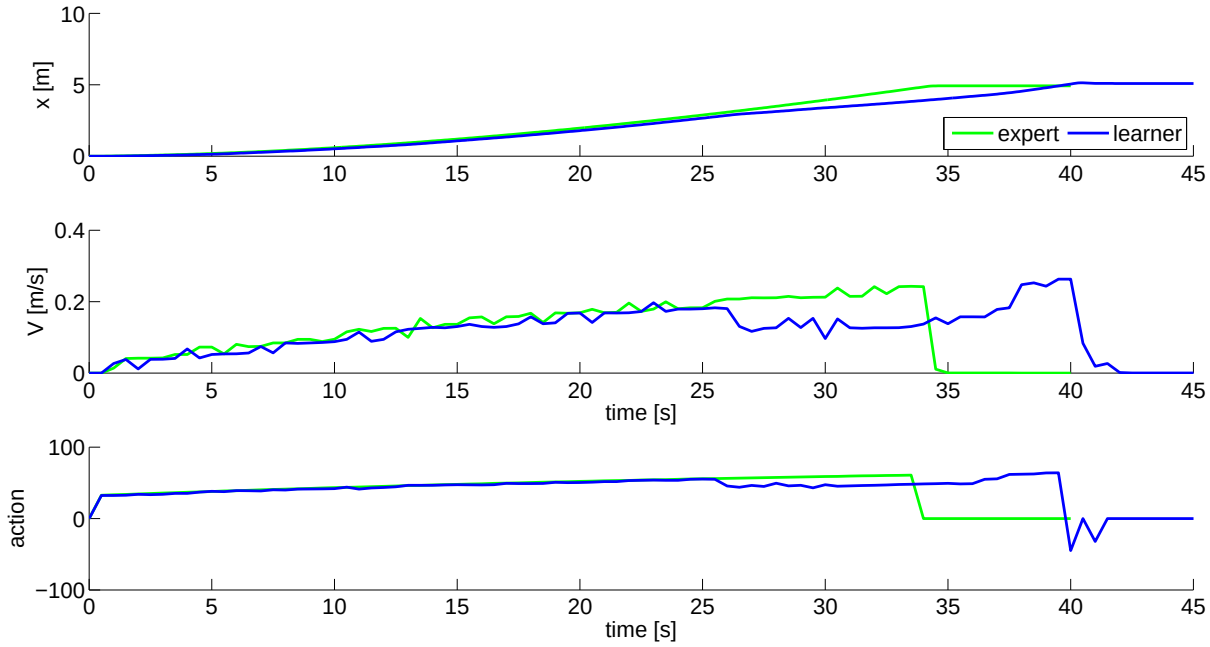
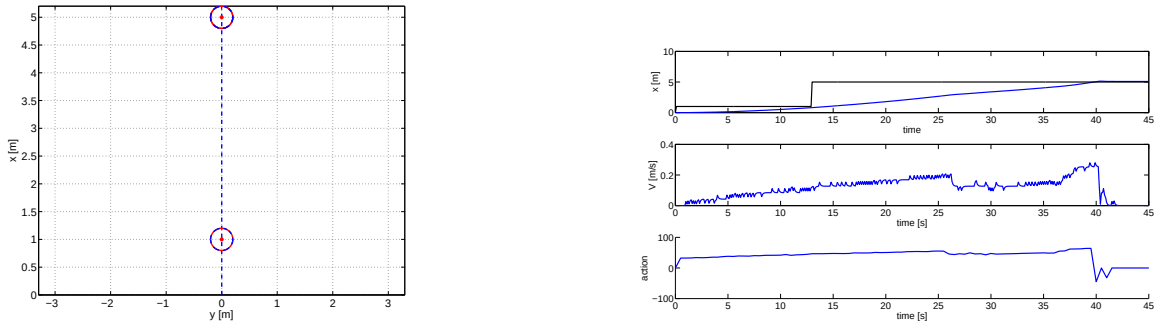
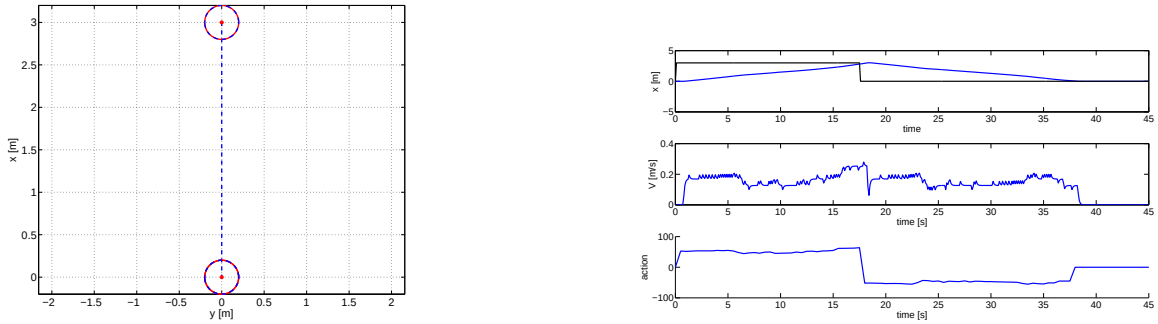


Figure 4.5. Comparison of expert and the learner trajectories for moving 5m forward task.



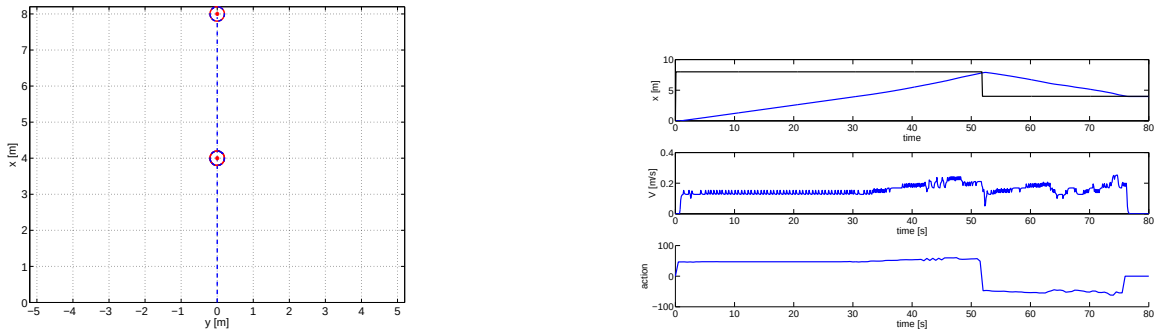
(a) (b)

Figure 4.6. Simulation results of moving 5m forward task.



(a) (b)

Figure 4.7. Simulation results of moving 3m forward then back to zero task.



(a) (b)

Figure 4.8. Simulation result of moving 8m forward then back to 4m task .

the same task are shown. Initially, the learner acts and behaves very similar to the expert. However, since the learner decides its actions using the combined dataset, the learner's actions starts to differ from the expert's and it arrives at the waypoint later than the expert. The learner's trajectory in the xy-plane can be seen in Fig. 4.6.

In the second simulation case, the learner is commanded to go to (3,0) and then come back to where it started at (0,0). That is, $\Delta x = 3$ for the first waypoint, and $\Delta x = -3$ for the second waypoint. Note that in the demonstration cases, Δx values were 1, 4, and 1.5. Thus, in this case the UGV is commanded to do a task that was not exactly shown before, but within the range of the demonstrated actions. As a result, it is expected that there are sufficient neighbor state-action pairs to create local models. As can be seen in Fig. 4.7, vehicle is able to move forward to 3m then come back to zero by moving backwards. In the third case, waypoints that are beyond the range of demonstrations are commanded. Although the vehicle was able to perform this task as seen in Fig. 4.8, this is mainly due to the simplicity of the model and the definition of the "closeness" threshold which was used in the simulation. However, for different more complex models, such performance should not be expected.

4.1.2 Implementation on the Real UGV

One of the future goals of this study is to demonstrate the learning algorithm on a real computing and hardware platform with multiple actions. This initial implementation study was conducted to see the capabilities of the vehicle and assess the performance of processing unit and the algorithm. For this purpose, the same tasks that were considered on the simulation cases were tested on the real platform. However, simulations assumed no slippage and did not model the mechanical imperfections. In the actual platform, because of these problems, single action cannot be used to operate the vehicle as the vehicle does not follow a straight line. Expert

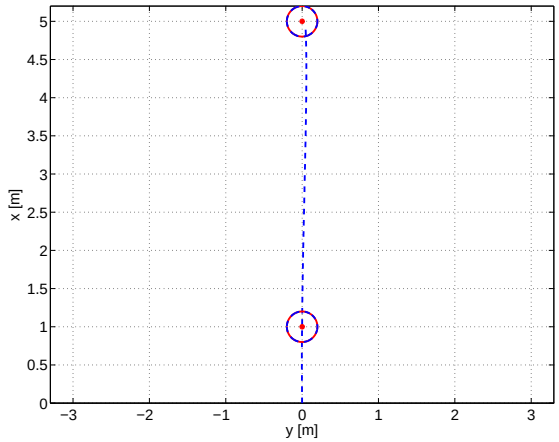


Figure 4.9. Expert demonstration for moving 5m forward task on UGV platform.

demonstrations are created by operating the vehicle under the GNC system. This way, by applying different duty cycles to the left and right motors, vehicle can track a straight line as can be seen in Fig. 4.9. However, the learning module continues taking the mean of left and right motor duty cycles to log the action into the dataset. This greatly effects the performance of the learner. For this preliminary test, focus is given to observing the behavior of the vehicle in the current configuration.

In the first test, vehicle is commanded to perform the demonstration case. The expert and the learner trajectories are compared at Fig. 4.10. Unlike the simulation case, the learner arrives at the waypoint before the expert. However, due to problems discussed, the learner cannot follow straight path and as can be seen in Fig. 4.11 diverges to right as it moves. Second simulation case is repeated on the real platform by commanding to move forward 3m and come back to zero by backwards motion. As shown in Fig. 4.12, vehicle can perform this task. However, similar to the first case, it diverges during both forward and backward motion. In the third case, vehicle is commanded to go to an undemonstrated state. Vehicle managed to perform this similar to the simulation case. Once again the vehicle slipped right as it moved forward as depicted in Fig. 4.13.

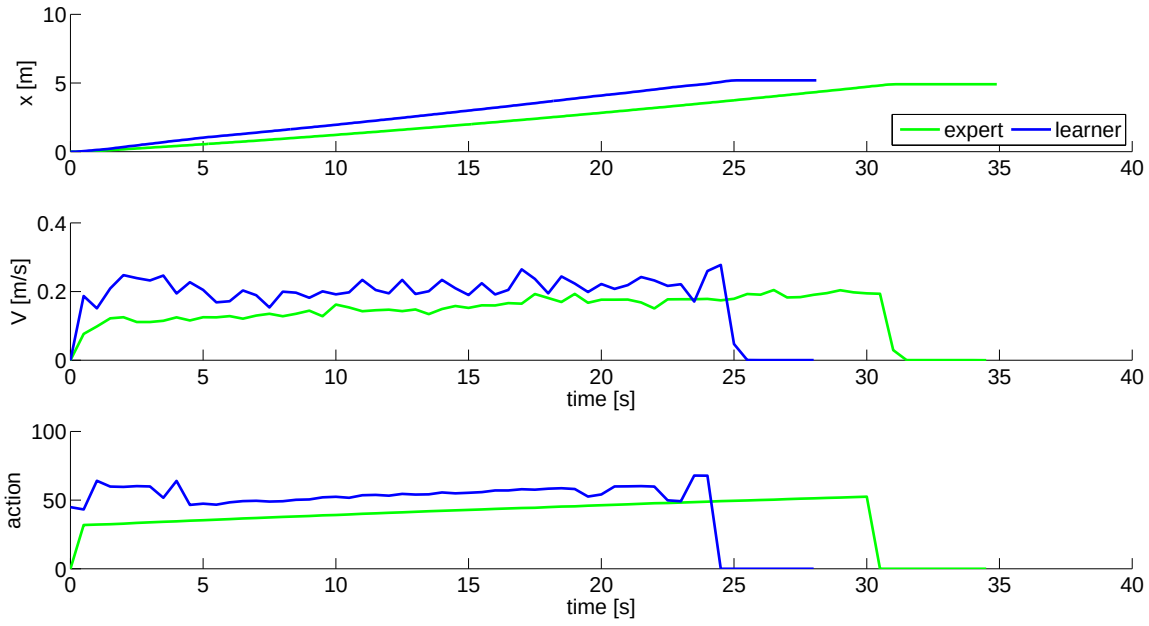


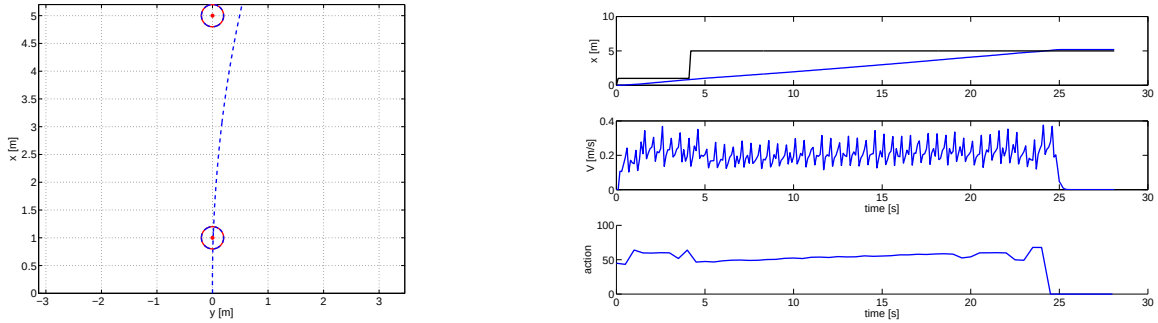
Figure 4.10. Comparison of expert and the learner trajectories for moving 5m forward task on UGV platform.

4.2 Experiments using multiple actions

In the previous part, experiments were performed using single actions to demonstrate the capabilities of the learning algorithm on simplified dynamics. However, actual robotic challenges are more complex and require handling multiple actions simultaneously. In this part, the simplifications on the UGV equations of motion are removed and learning algorithm is extended to fulfill complete 2-D waypoint navigation task. For this purpose new states are defined. Body frame of the UGV is defined at the center of the UGV as shown in Fig. 4.14. First two states are the position relative to the next waypoint in body axis x and y-axes, (Δx_B, Δy_B), which is

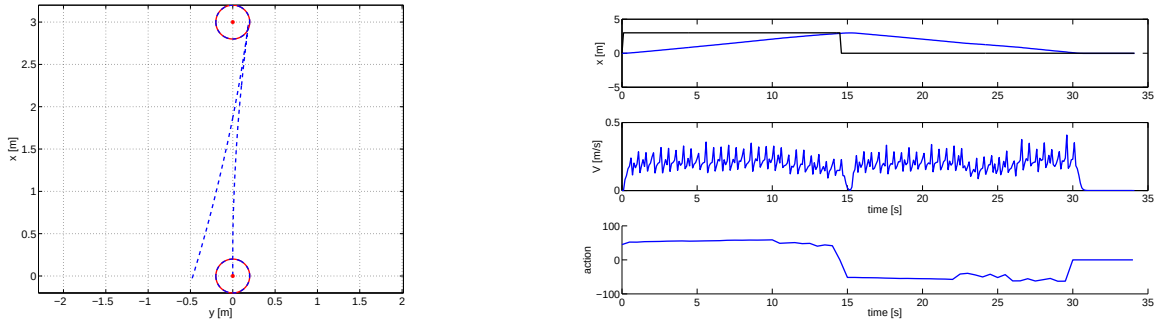
$$\begin{bmatrix} \Delta x_B \\ \Delta y_B \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} \Delta x_N \\ \Delta y_N \end{bmatrix} \tag{4.5}$$

where (Δx_N, Δy_N) are the position relative to the next waypoint in the navigational frame. Error in speed relative to the commanded speed, ΔV is the third state while



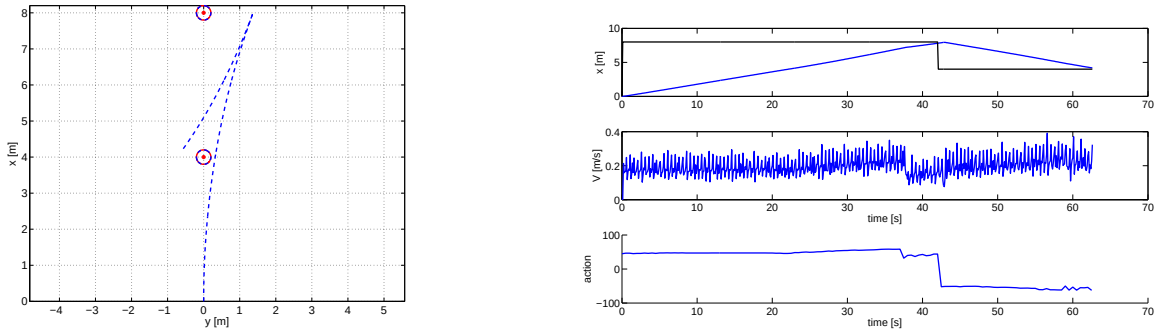
(a) (b)

Figure 4.11. Results of moving 5m forward task on UGV platform.



(a) (b)

Figure 4.12. Results of moving 3m forward then back to zero task on UGV platform.



(a) (b)

Figure 4.13. Results of moving 8m forward task then back to 4m on UGV platform.

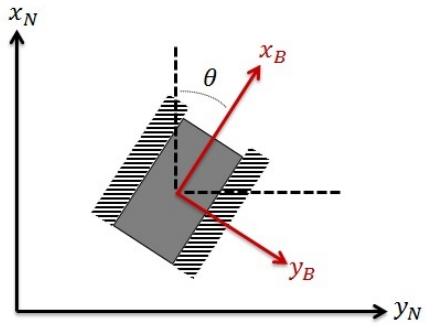


Figure 4.14. Navigational and body frame of the UGV.

right and left motor duty cycles, (a_L, a_R) are the actions. Learning rate of $\alpha = 0.45$ and quadratic reward function

$$r = -\left(\Delta x_B^2 + \Delta y_B^2 + \Delta V^2\right) \tag{4.6}$$

used during the learning. Created demonstration dataset is slightly modified with the introduction of action replays. After a demonstration is obtained, it is repeated several times until the action-values of the state action pairs settles at constant values. For selected 10 state-action pairs, change of the action-values can be seen in Fig. 4.15, as the demonstration is repeated for 25 times. Both simulation and implementation on the actual vehicle are performed to show the capability of the learning module with multiple actions.

4.2.1 Simulation results

Demonstrations are performed by locating 12 single waypoints around a circle with equal distances. Three different radius values, *0.4m., 1m., 2m.* are used for the waypoint circles to obtain the expert trajectories depicted in Fig. 4.16. After expert trajectories are recorded, learning dataset is obtained by combining and sparsifying the demonstrations as shown in Fig. 4.17. Two simulation cases are performed to test the capability of multiple action selection process. In the first case, the UGV is

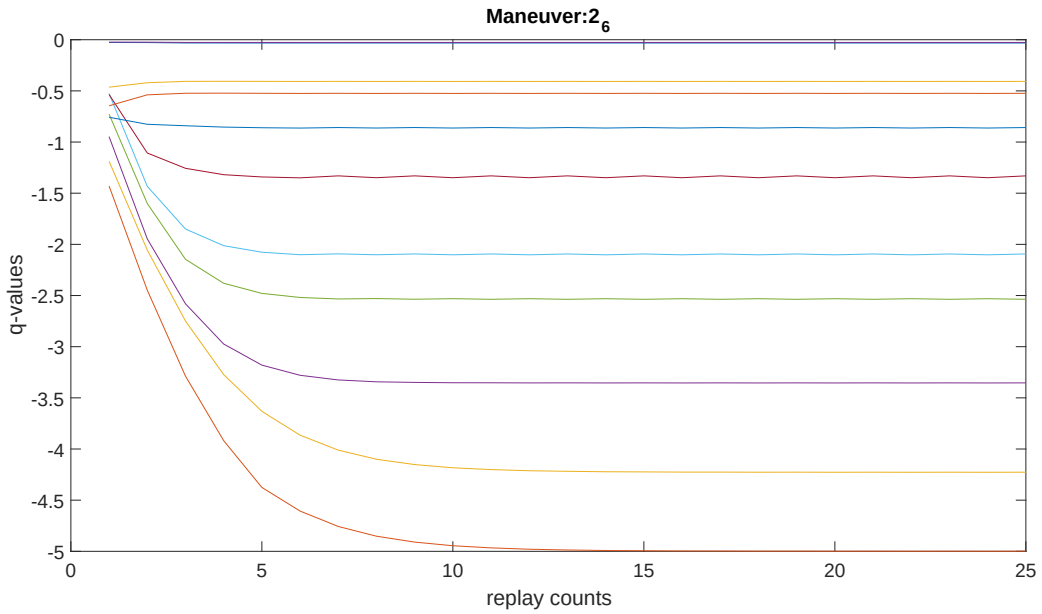


Figure 4.15. Convergence of the action-values with the action replay process.

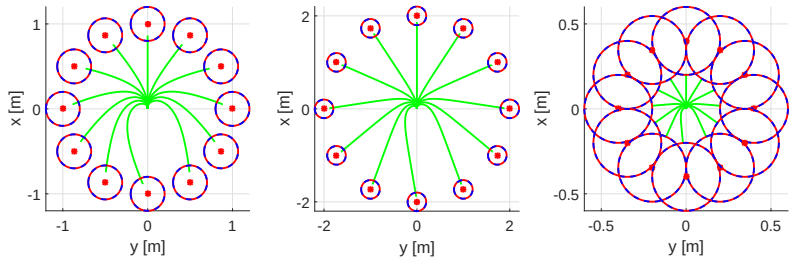


Figure 4.16. Expert demonstration trajectories of the UGV in the simulation environment.

commanded to drive a triangle by going to waypoint (1,0) and then (1,1) and return to the (0,0) meters in xy-frame. In the second case, waypoints are on the corners of a square (1,0), (1,1), (0,1) and (0,0) in order. Trajectory and time history plots in Figs. 4.18 - 4.19 show that both waypoint navigation tasks are completed succesfully.

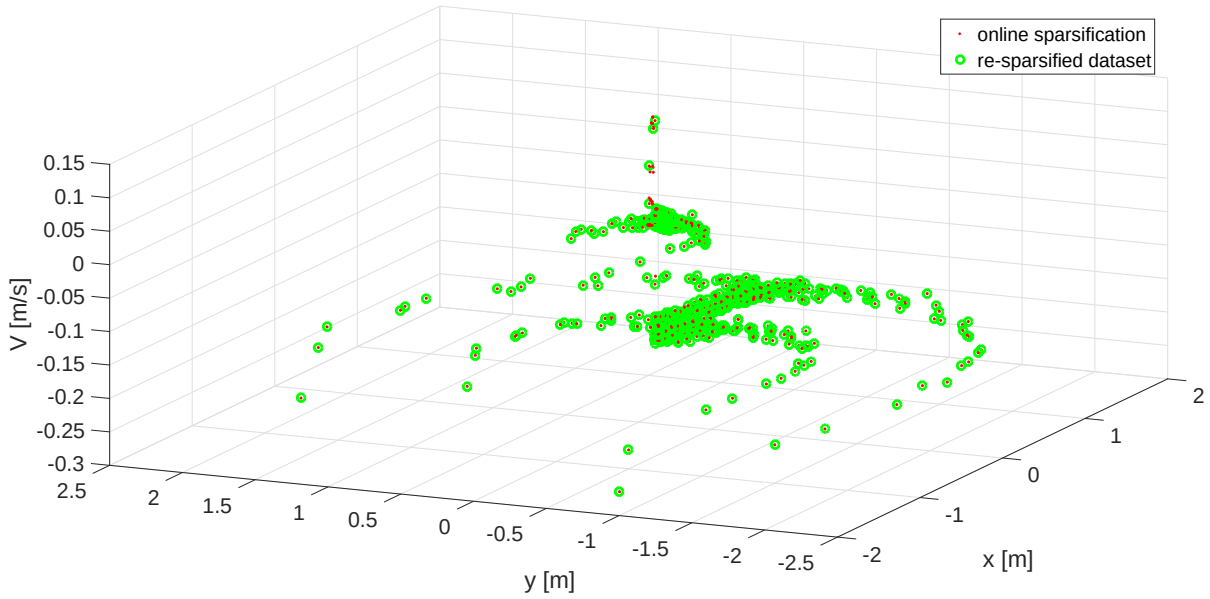
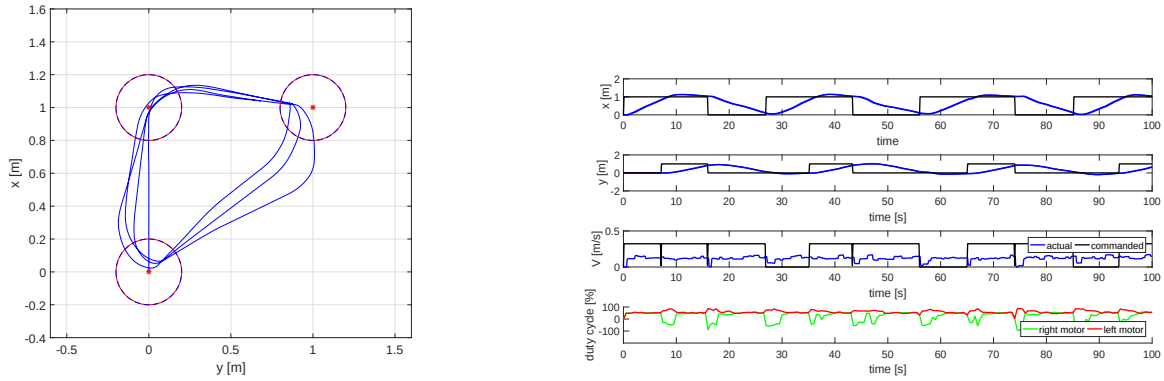


Figure 4.17. Sparsified expert demonstration trajectories of the UGV in the simulation environment.

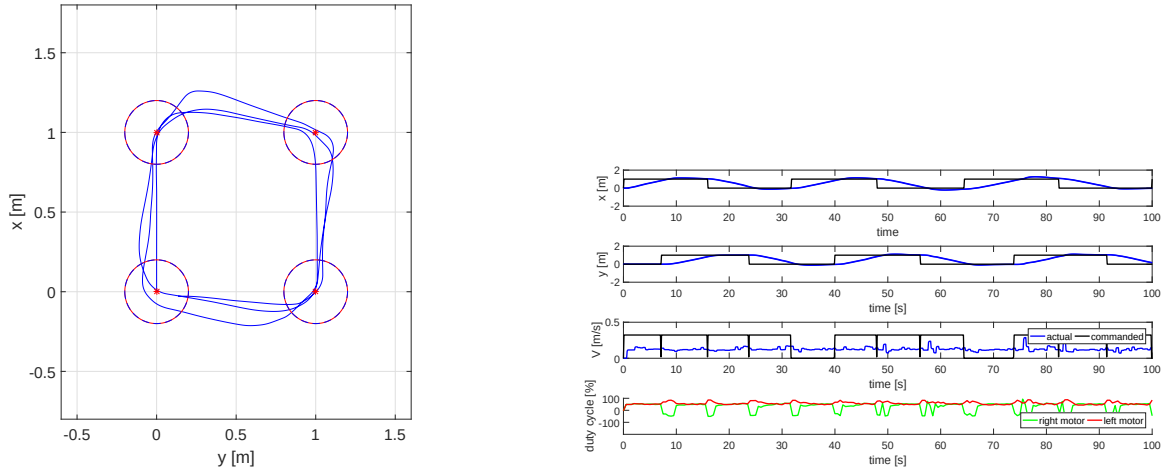
4.2.2 Hardware Implementation

In order to show the capabilities of the learning module on actual hardware, the same expert demonstrations and the waypoint navigation tasks are implemented on the real vehicle. After expert trajectories are recorded, learning dataset is obtained by combining and sparsifying the demonstrations as shown in Fig. 4.21. The same waypoint navigation tasks are repeated to test the real time operation capability of multiple action selection process. Both triangle and the square shape waypoints were successfully tracked as can be seen in Figs. 4.22 - 4.23.



(a) (b)

Figure 4.18. Simulation results of triangle shaped waypoints.



(a) (b)

Figure 4.19. Simulation results of square shaped waypoints.

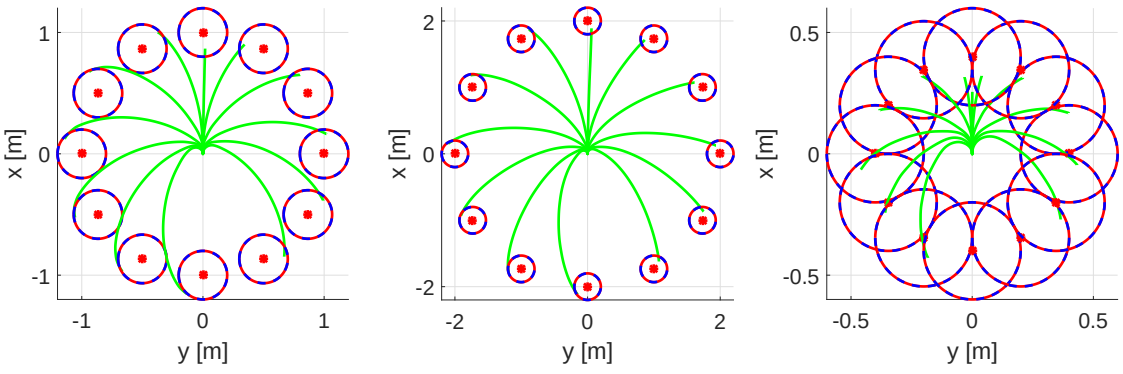


Figure 4.20. Expert demonstration trajectories on the UGV platform.

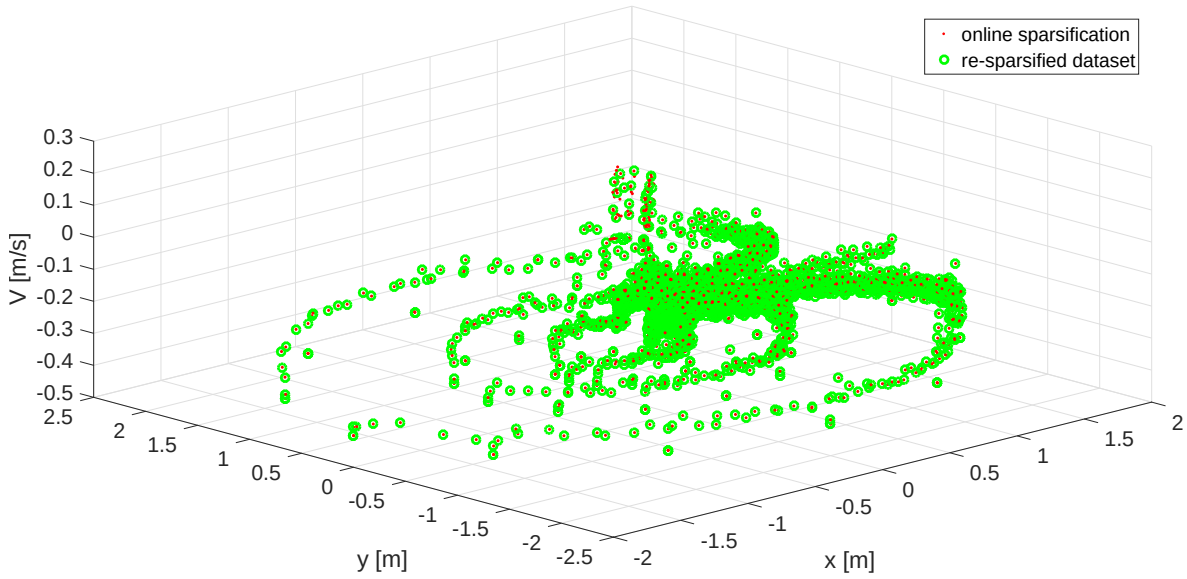
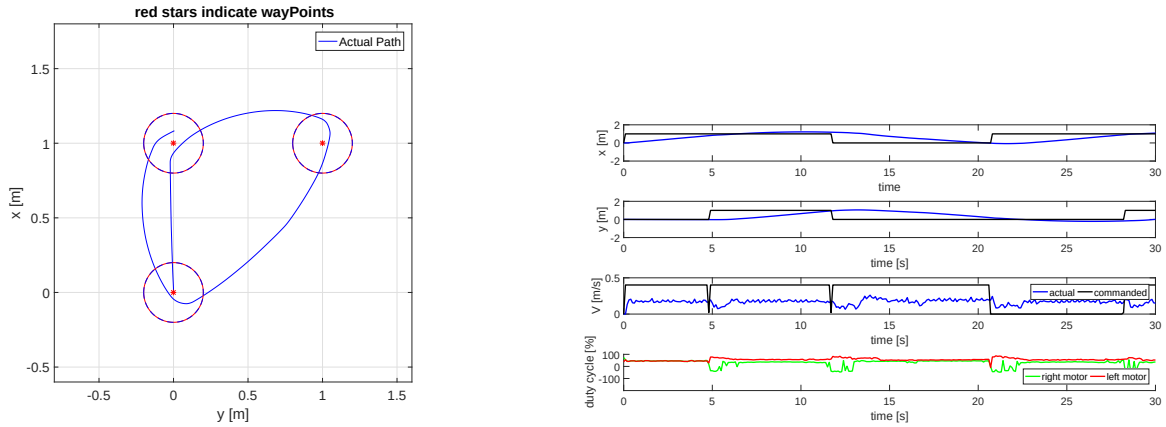
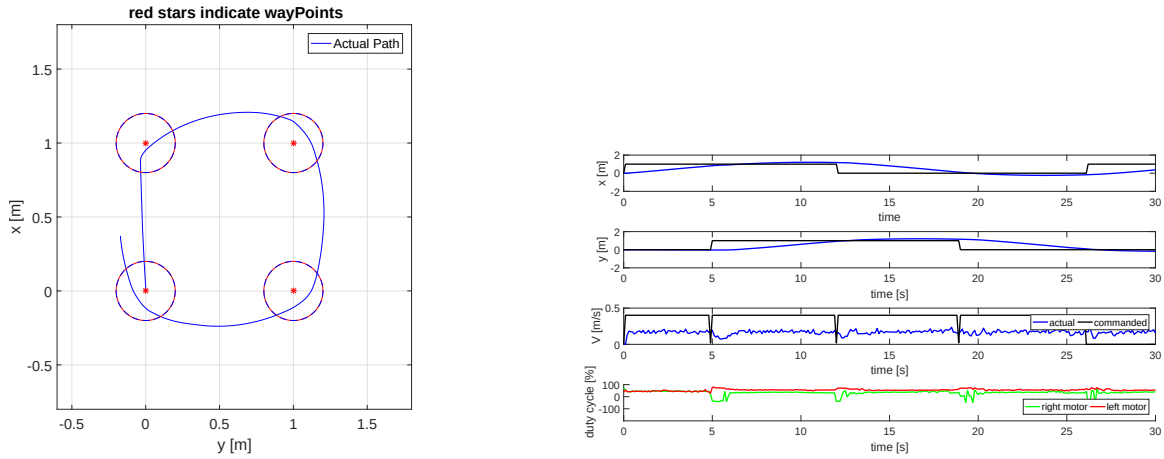


Figure 4.21. Sparsified expert demonstration trajectories on the UGV platform.



(a) (b)

Figure 4.22. Hardware implemetation results of triangle shaped waypoints.



(a) (b)

Figure 4.23. Hardware implemetation results of square shaped waypoints.

CHAPTER 5

Airship Flight Simulator Development

In LfD/RL, reward acquisition through interaction with the environment is the essential part of achieving learning. In robotic applications, this reward is almost always a function of system states and actions. Conditions which adversely effect the quality of reward such as unobservable states, noise or sensor bias are often encountered in robotic control applications. These problems are addressed in the control system design by using frequency filters and state observers or fusing several sensor data to lower the uncertainty in measurements. In machine learning, partially observable Markov models deal with the problems of similar nature. However, these methods always assume that sensor noise and bias can be modelled or by combining different agents' measurements, quality can be enhanced.

Conventional unmanned aerial systems which are operated outdoors usually depend on combined sensor suites to obtain feedback of the system states. In this study, the main goal is to demonstrate the capability of LfD/RL framework on the control of low speed airship motion. For this purpose, selected testbed is an indoor RC airship. However, in the low speed control of indoor airship, several navigation challenges are encountered. For instance, reliable ground speed readings can be retrieved from GPS only outdoors just as pitot tube or IMU can only produce healthy data while maintaining high speeds. This makes it necessary to use indoor navigation solutions to be able to get ground-truth measurements. Motion capture systems are advanced indoor navigation solutions which use multiple cameras to track and calculate the 3D position of objects in a certain coverage space. However, this coverage is dependent

Figure 5.1. MAE Blimp.

on the number and the specifications of the cameras and as a result cost becomes an important factor. For large objects such as airship, meeting the space requirements for guidance and control maneuvers becomes infeasible.

This brings the necessity for a validated flight simulator that can be used to collect expert demonstrations and test the capabilities of the proposed LfD/RL learning framework. For this purpose, first a simulation model is generated using the airship equation of motion, and the geometry and mass data of the actual indoor airship, referred to as "MAE Blimp" and shown in Fig. 5.1. Then, based on the response of the airship to various stimuli, measured in various experiments, the parameters of the simulation model is tuned and its fidelity is validated.

5.1 Airship Model

The airship equations of motion were derived [91, 92] by defining the origin of the airship's body frame (B_A-frame) at a geometrically fixed point, with a vector denoted ρ_{CM} from the origin of the body frame to the location of center of mass of

the rigid body. This derivation is adapted here for a body frame with its origin at most-forward tip of the envelope, centered in the ellipsoid YZ plane, as depicted in Fig. 5.2.

5.1.1 Modeling of Aerodynamics with Added Mass and Inertia Effect

Added mass and inertia, also called "virtual" mass and inertia, are the contribution of the displacement of the fluid medium to the overall momentum change of a system moving through that fluid. For LTA (Lighter-Than-Air) vehicles such as airship, this phenomenon cannot be neglected as the displaced mass is closed to that of the vehicle mass. Added mass and inertia coefficients are calculated using slender body theory and a detailed derivation is given in [93].

Following the practice of Ref. [92], the aerodynamic force representation in the B_A-frame, A_A is expanded as the summation of three terms: (i) aerodynamic force depending on the translational and rotational velocity components as well as aerodynamic control effectors, (ii) aerodynamic force due to the translational acceleration and (iii) aerodynamic force due to the rotational acceleration. Thus,

$$A_A = A_0 - A_1\dot{U}_A - A_2\dot{\omega}_{B_A} \tag{5.1}$$

where A_0 is the standard aerodynamic force expression while A_1 is the "added" mass and A_2 represents the aerodynamic effect of angular acceleration on the translational acceleration. The standard aerodynamic force representation in the B_A-frame, A_0, is expanded as

$$A_0 = \begin{bmatrix} A_X \\ A_Y \\ A_Z \end{bmatrix} = - \begin{bmatrix} \frac{1}{2}\rho V_a^2 S_{ref} C_T \\ \frac{1}{2}\rho V_a^2 S_{ref} C_L \\ \frac{1}{2}\rho V_a^2 S_{ref} C_N \end{bmatrix} - D_U \begin{bmatrix} U_A \\ \omega_{B_A} \end{bmatrix} \tag{5.2}$$

where C_T, C_L, and C_N are the body frame X-direction, Y-direction, and Z-direction aerodynamic force coefficients, and D_U is the translational portion of the Coriolis-centrifugal coupling matrix. Additionally, ρ is the air density and V_a is the airspeed of the airship, which can be calculated as

$$V_a = \sqrt{u^2 + v^2 + w^2} \tag{5.3}$$

The angle of attack and sideslip angles are calculated by

$$\begin{aligned} \alpha &= \tan^{-1} \frac{w}{u} \\ \beta &= \tan^{-1} \frac{v \cos(\alpha)}{u} \end{aligned} \tag{5.4}$$

Similar to the aerodynamic force, the aerodynamic moment representation can be expanded as the summation of the body frame aerodynamic moment and the additional moments that are a result of the added mass and inertia characteristic of an aircraft which displaces air of significant mass as

$$M_A = M_0 - M_1 \dot{\omega}_{B_A} - M_2 \dot{U}_A \tag{5.5}$$

The representation of the aerodynamic moment vector in the B_A-frame, M_0, is expanded as

$$M_0 = \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix} = - \begin{bmatrix} \frac{1}{2}\rho V_a^2 S_{ref} L_{ref} C_l \\ \frac{1}{2}\rho V_a^2 S_{ref} L_{ref} C_m \\ \frac{1}{2}\rho V_a^2 S_{ref} L_{ref} C_n \end{bmatrix} - D_\omega \begin{bmatrix} U_A \\ \omega_{B_A} \end{bmatrix} \tag{5.6}$$

where C_l, C_m, and C_n are aerodynamic moment coefficients about the body frame X, Y, and Z axes, respectively, and D_ω is the rotational portion of the Coriolis-centrifugal coupling matrix.

As the low speed motion of the airship is considered for the scope of this work, several simplifications are made on the aerodynamic model of the airship. First of

all, since the effects of added mass and inertia manifest themselves only during the acceleration of the airship, added mass A_1, A_2 and added inertia M_1, M_2 terms as well as effect of wind are omitted from the equations of motion.

Airship aerodynamic force and moment expressions commonly in the literature mainly come from Munk's study [94] which uses potential flow theory around an ellipsoid to derive aerodynamic coefficients as functions of angle of attack (α) and angle of sideslip (β). When applied for high Reynolds number cases such as cruise, these coefficients produce accurate forces and moments. However, in the very low speed range around zero, angle of attack and angle of sideslip calculations lose their validity and yield erroneous forces and moments. To avoid such issues in low speed simulation of airship, A_0, M_0 terms in Eqs. (5.2) and (5.6) are replaced with simple expressions similar to [30], to model the energy dissipation of the system proportional to the translational and rotational speeds.

With these assumptions, aerodynamic force and moment contributions, (A_A and M_A) are written as

$$A_A = \begin{bmatrix} A_X \\ A_Y \\ A_Z \end{bmatrix} = - \begin{bmatrix} C_u u \\ C_v v \\ C_w w \end{bmatrix} |U_A| \tag{5.7}$$

$$M_A = \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix} = - \begin{bmatrix} C_p p \\ C_q q \\ C_r r \end{bmatrix} \tag{5.8}$$

where the coefficients are left for identification through experiments.

5.1.2 Equations of Motion

The translational kinematics equation is written in matrix form in terms of the position vector of the airship with respect to an inertial frame. To consider the wind effect, the translational kinematics of the airship is written as

$$\dot{r}_{B_A} = \mathbf{R}^{\mathsf{T}}_{\mathbf{B_A I}} U_A + W \tag{5.9}$$

where $\dot{r}_{B_A}$ is the representation of the velocity vector in the inertial frame, and U_A is the representation of the velocity vector of the airship relative to the surrounding air in the body frame, which is expanded as

$$U_A = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.10}$$

where u, v, and w are the airship velocity components in its body x-, y-, and z-axes. Additionally, the vector W is the representation of the local wind velocity in the inertial frame, written as

$$W = \begin{bmatrix} Wx \\ Wy \\ Wz \end{bmatrix} \tag{5.11}$$

The translational dynamics is written in the matrix form using Newton's second law and rotation matrix as

$$\begin{aligned} [m\mathbf{I}_{3\times 3} + A_1]\, \dot{U}_A = m\, &\left[\mathbf{S}(\omega_{\mathbf{B_A}}) U_A - \mathbf{R}_{\mathbf{B_A I}} \dot{W}\right] \\ &+ \mathbf{R}_{\mathbf{B_A I}}\, F_G + A_0 + P_A - m\mathbf{S^2}(\omega_{\mathbf{B_A}})\rho_{CM} \\ &- \left[m\mathbf{S}(\rho_{\mathbf{CM}}) + A_2\right] \dot{\omega}_{B_A} \end{aligned} \tag{5.12}$$

where ω_{B_A} is the angular velocity vector of the airship relative to the inertial frame, $\mathbf{S}(\omega_{\mathbf{B_A}})$ is the skew-symmetric form of ω_{B_A} and $\mathbf{I}_{3\times 3}$ is the 3×3 identity matrix. F_G

is the representation of the gravity and buoyancy force vector in the inertial frame and P_A is the representation of the propulsive force vector in the B_A-frame.

Rotational kinematics is written in terms of the Euler angles of the airship body frame with respect to the inertial frame as

$$\begin{aligned}
\dot{\psi} &= (q\sin\phi + r\cos\phi)\sec\theta \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\phi} &= p + q\sin\phi\tan\theta + r\cos\phi\tan\theta
\end{aligned} \tag{5.13}$$

where (ψ, θ, ϕ) are the 3-2-1 Euler angles and (p, q, r) are the components of the airship angular velocity vector expressed in B_A-frame.

Finally, the rotational dynamics equations are written as

$$\begin{aligned}
(\underline{\underline{\mathbf{I}}}_{\mathbf{B}} + M_1)\dot{\omega}_{B_A} = {} & M_0 + M_G + M_P + \mathbf{S}(\omega_{\mathbf{B_A}})\underline{\underline{\mathbf{I}}}_{\mathbf{B}}(\omega_{B_A}) \\
& + m\mathbf{S}(\rho_{\mathbf{CM}})[-\mathbf{S}(\omega_{\mathbf{B_A}})U_A + \mathbf{R}_{\mathbf{B_A I}}\dot{W}] \\
& + [m\mathbf{S}(\rho_{\mathbf{CM}}) - M_2]\,\dot{U}_A
\end{aligned} \tag{5.14}$$

where $\underline{\underline{\mathbf{I}}}_{\mathbf{B}}$ is the inertia matrix with respect to the B_A-frame, M_G is the representation of the moment vectors due to gravity and buoyancy in the B_A-frame and M_P is the representation of the moment due to propulsion in the B_A-frame.

5.1.3 Propulsion System

Airship platform considered for this study is equipped with twin ducted main propellers mounted under the envelope as well as a propeller in the lower fin (see Fig. 5.2). Although it has vertical and horizontal tail fins, no control surfaces are attached. If the thrust generated by the main envelope propellers in the body frame is denoted as T_M with a tilt angle of μ, and the thrust generated by the tail propeller

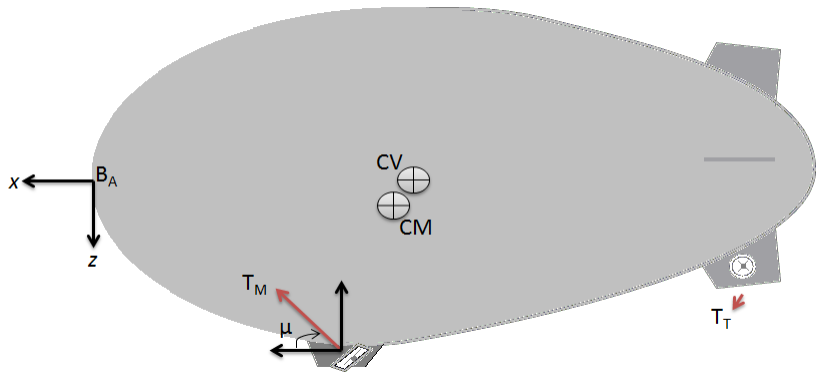


Figure 5.2. A depiction of the airship.

in the body frame is denoted as T_T, then the representation of the propulsion force vector P_A as it appears in Eq. (5.12) is

$$P_A = P_{A_M} + P_{A_T} \tag{5.15}$$

where P_{A_M} is the propulsion force associated with the main propellers and P_{A_T} is that associated with the tail propeller, which are written as

$$P_{A_M} = \begin{bmatrix} T_M \cos\mu \\ 0 \\ -T_M \sin\mu \end{bmatrix} \qquad P_{A_T} = \begin{bmatrix} 0 \\ T_T \\ 0 \end{bmatrix} \tag{5.16}$$

The moment due to propulsion as it appears in Eq. (5.14) is

$$M_P = -S(\rho_M)P_{A_M} - S(\rho_T)P_{A_T} \tag{5.17}$$

where ρ_M and ρ_T are the main propeller and tail propeller position vectors with respect to the origin of the airship's body frame, respectively.

5.2 Experiments

Generic equations of motion of the airship can be specialized for a vehicle by plugging in the vehicle specific parameter values. Mass, inertia, dimension and sizes

can be determined or estimated by measurements and engineering calculations within certain accuracy. Parameters such as aerodynamic coefficients or the thrust characteristics can be approximated using similar vehicle's data or manufacturer datasheets. The simulation model built with the specific values of the airship system and component parameters should still be validated against the response of the actual airship, if available. In our study, the indoor blimp, "MAE Blimp", represented by the simulation model and some of its subsystems are available for data collection to be used in simulation validation, albeit in limited capacity.

For this purpose, MAE blimp mass and geometry are estimated/measured, and used in the first version of simulation. For the thrust generation characteristics of the main and tail thrusters, thrust magnitude measurements in a force-balance sensor are used to develop first-order transfer function representations. Aerodynamic coefficients are identified through the experiments performed on the airship under a motion capture system. As the final phase of the validation, while a human pilot operated the vehicle through an RC transmitter, the flight test data are recorded and used to further refine the model parameters.

5.2.1 Thrust measurement tests

Main objective of the thrust measurement tests is to design a realistic thruster model that represents the relation between the pilot commands and thrust generated by the thrusters. The pilot operates the MAE blimp by moving the sticks on a standard 2.4 GHz RC radio transmitter. These stick commands are converted into PWM signals such that, in the ideal case, the lowest and highest stick positions are represented by 1000 and 2000, respectively. Electronic speed controller (ESC) unit receives the PWM signal from the receiver and controls the motors according to the built-in programming modes. A rough estimation of the generated thrust can be made

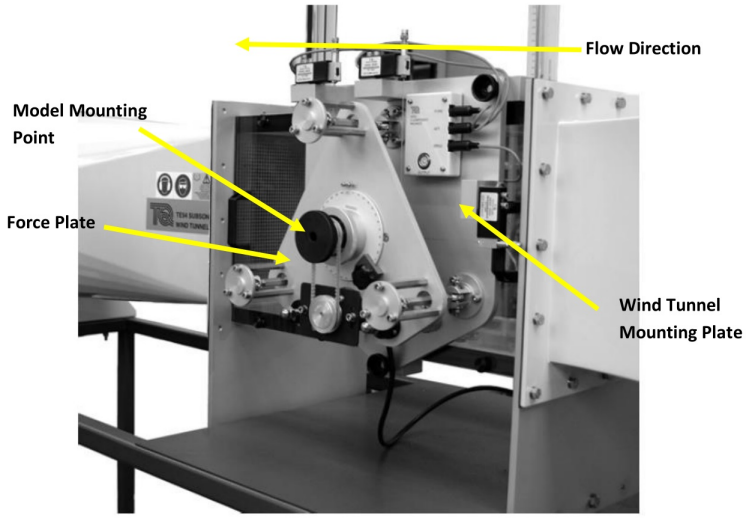


Figure 5.3. TecQuipment AFA-3 Three Component Balance.

if some characteristics of the RC radio/receiver, ESC, motors and the propellers are known. However, this approximation would not be accurate or reliable enough to be used in control or learning applications. This brings the necessity of experimental methods to identify the thruster characteristics in a suitable test environment.

For the experiments, AFA-3 Three Component Balance equipment [95], shown in Fig. 5.3, is used. Thrust measurements are transferred to a computer through SCB-68 data acquisition box [96] and LabVIEW software is used to record the test data. Main and tail thrusters are removed from the airship and attached to the AFA-3 using an aluminum alloy connection rod as can be seen in Figs. 5.4 and 5.5, respectively.

Thrust tests are carried out by sending PWM commands from the transmitter to the ESC through the RC receiver. Signal on the receiver is also wired into a datalogger through a Y-cable in order to record the PWM commands. Stick is held constant at several positions for certain durations while thrust values are being recorded as can be seen in Fig. 5.6. The tests are repeated several times in order to make sure the results

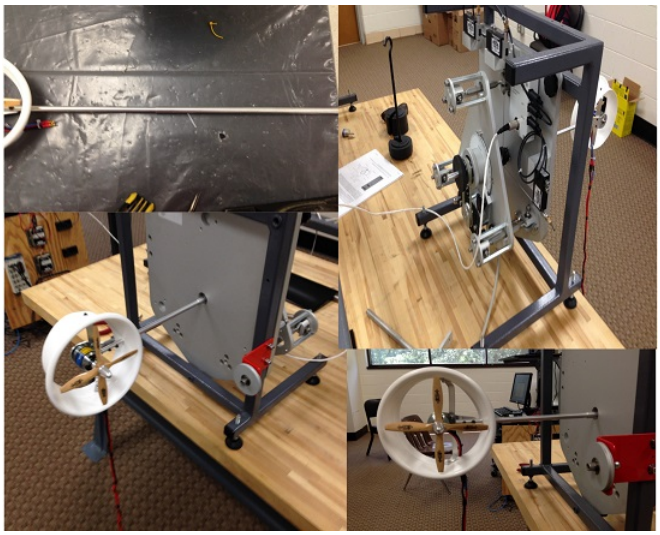


Figure 5.4. Main thruster test setup.

are consistent and reliable. Applied PWM commands and the measured steady-state thrust values after each change in PWM commands are used in constructing a static look up table in order to create an interpolation based relation between PWM input and the thrust output. However, result of interpolation makes sudden changes in the value of the thrust as the motor dynamics and the propeller aerodynamics are neglected as shown in the Fig. 5.7. In order to compensate for this, a first order transfer function is inserted to the thruster model. At this point, only parameter required to be tuned is the time constant of the transfer function and by trial and error a satisfactory output response is obtained as shown in Fig. 5.8.

Same steps of test and validation are performed for the tail thruster and similar thruster model have been created. Tail thruster on the airship is driven by a single ESC which creates opposite thrust forces as the PWM input goes above and below center level. Due to several reasons, including installation of the thruster and ESC programming mode, PWM-thrust relation is not symmetric between the left and right thrust commands. For this reason, left and right thrusters are modeled and tested

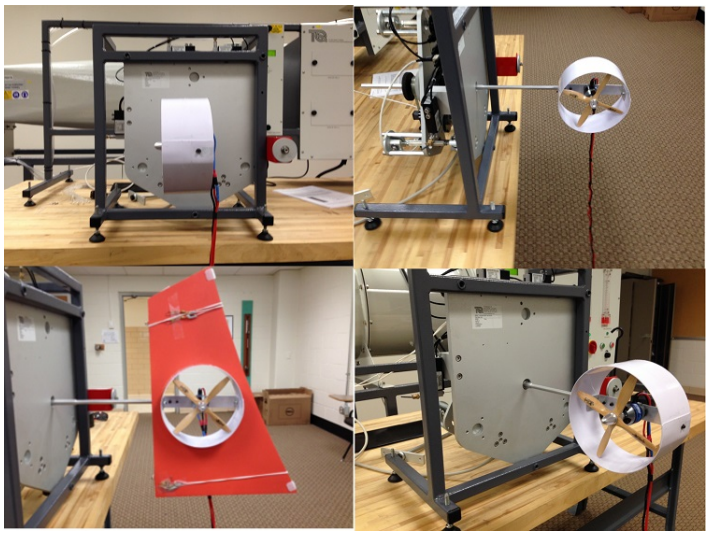


Figure 5.5. Tail thruster test setup.

separately. A comparison of tail thrust forces between the test data and constructed thruster model for right and left sides is presented in Fig. 5.9.

5.2.2 Motion Capture tests

Several parameters in the airship equations of motion as well as the calculations based on the size and weight measurements of the actual blimp are subject to uncertainty. In order to have a realistic flight simulator, these parameters should match the actual airship's. However, most of these parameters can not directly be measured, and thus requires to be constructed from the measurable airship states. Accuracy of the obtained parameters are highly dependent on the quality of the sensor measurements. Since the airship is operated in the indoor environment and under low speeds, many widely used sensor solutions fail to provide reliable measurement data. As mentioned before, motion capture is one of the most accurate indoor navigation solutions for small unmanned systems and can be used for identifying the parameters of the airship and validating the flight simulator.

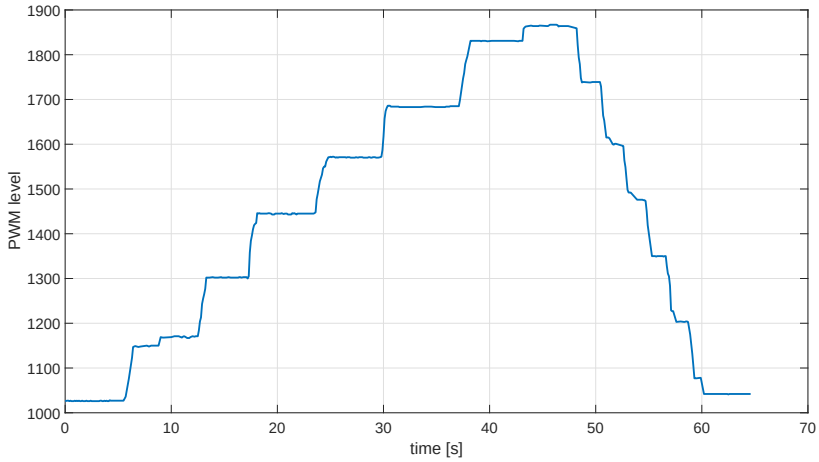


Figure 5.6. RC input signal applied to the main thruster test.

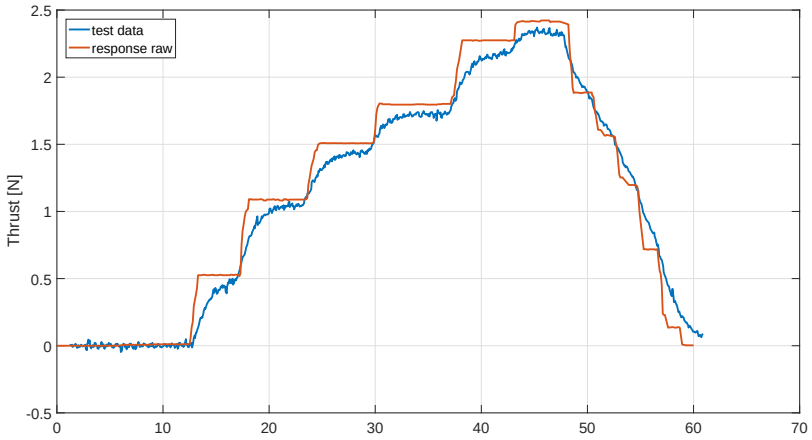


Figure 5.7. Comparison of thrust forces for the test data and constructed static look up table.

For this purpose, MAE blimp is placed inside a $16ft \times 16ft \times 14ft$ area surrounded by 16 VICON Bonita cameras as can be seen in Fig. 5.10. Each Bonita camera contains 68 LEDs which illuminate the reflective markers and capture data at 240 fps. Markers are combined to form separate rigid body objects in VICON Tracker software as can be seen in Fig. 5.13 in order to predict the 3D position in millimeter accuracy. Airship motion is detected by the several reflective markers which are placed on the blimp envelope and gondola as can be seen in Figs. 5.11-5.12.

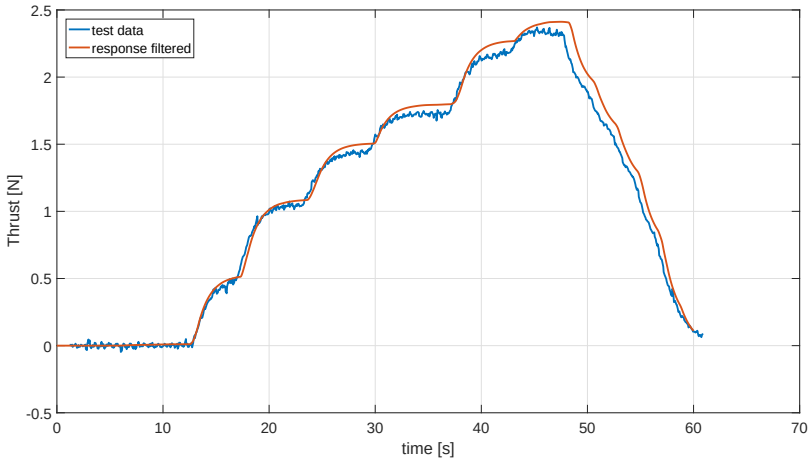
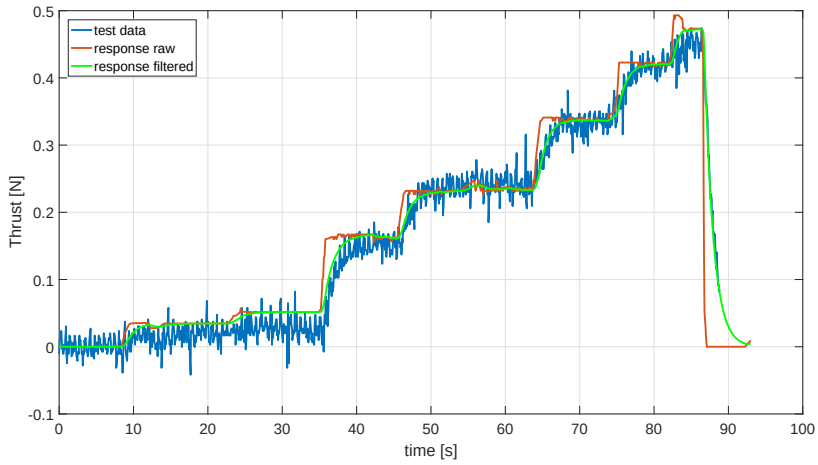


Figure 5.8. Comparison of thrust forces for the test data and constructed look up table with first order dynamics.

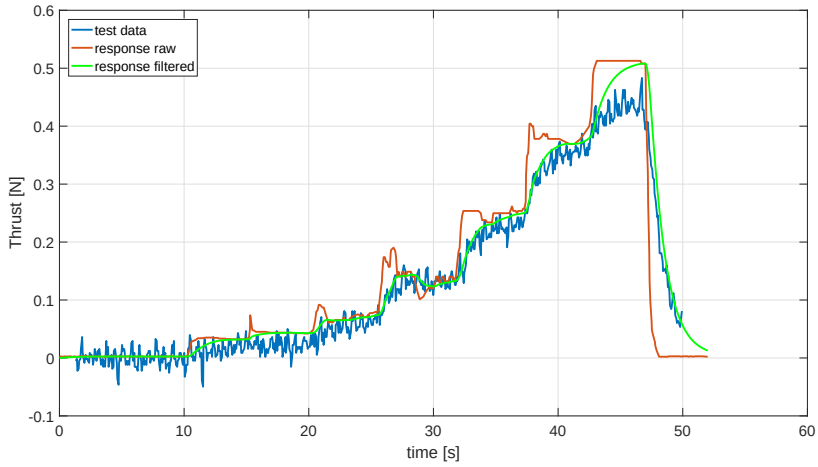
Experiments under motion capture are conducted in two parts. In the first part, no thrust is applied and only natural response of the airship is investigated. In the second part of the experiments, airship is flown by a human RC pilot in low speed maneuvers.

In the experiments to record the natural response of the airship, MAE blimp is brought into a state in which the weight and buoyancy are equal to each other. For the pitch and roll motions, natural response of the airship to the non-equilibrium initial values is investigated. When released from initial conditions, the airship tends to return to the equilibrium point while exhibiting damped oscillations under the influence of aerodynamic and gravitational moments. This can easily be reproduced in the simulation environment and by comparing the two responses, aerodynamic pitch and roll moment coefficients in the Eq. (5.8) are determined. At the equilibrium point, airship holds a pitch angle, θ_e, where the gravity and buoyancy forces/moments are balanced and this value can also be measured during these tests.

In addition, the moments of inertia of the airship can be identified from the same demonstrations as the inertia plays a prominent role in the free oscillatory motion of



(a)



(b)

Figure 5.9. Comparison of tail thrust forces for the test data and constructed thruster model for right (a) and left (b) sides.

the airship. Since the initial estimate of the inertia is calculated by empirical formulas and geometrical approximations, it is expected to have discrepancies with the actual figures.

Figures 5.14 and 5.15 show the pitch and roll responses, respectively, of the airship model in the simulation as the aerodynamic and inertial parameters are tuned as compared to the responses of MAE blimp recorded in the experiments. For both figures, the top plot depicts the comparison between the data collected under mo-

Figure 5.10. Airship inside the test area under motion capture system.

Figure 5.11. Marker placement on the airship hull.

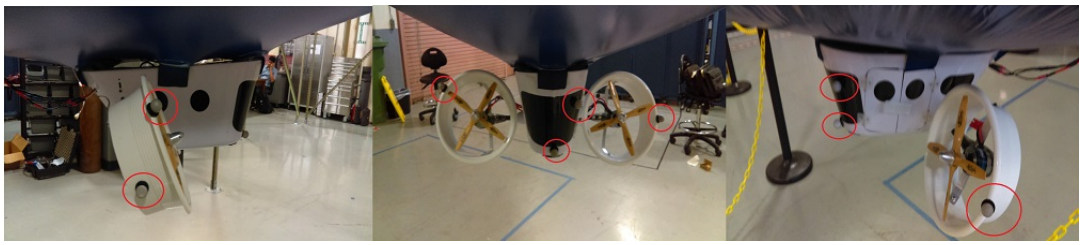


Figure 5.12. Marker placement on the airship gondola.

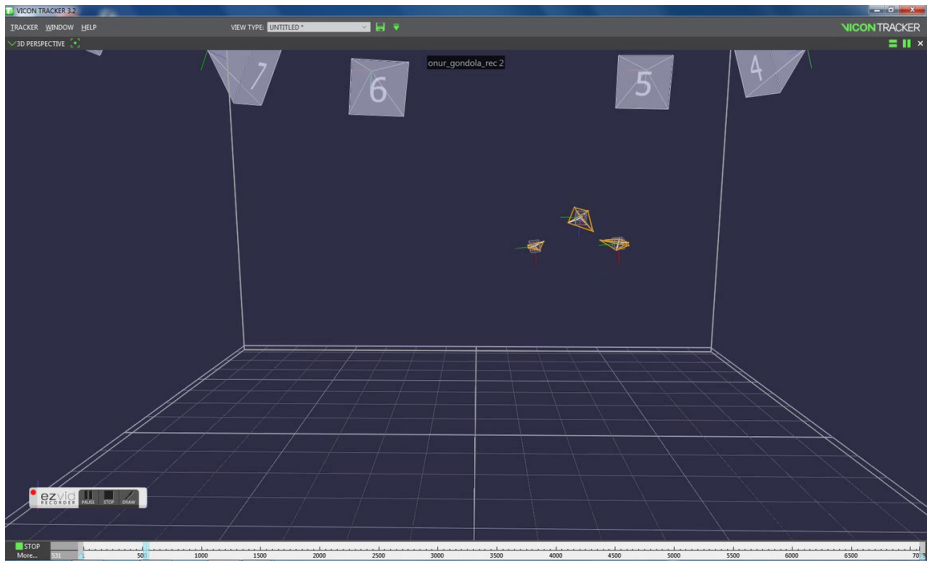


Figure 5.13. VICON Tracker preview of the airship gondola and two ducts.

tion capture and the data recreated in the simulation environment using the initial estimates of the parameters whereas middle and bottom figures illustrates the simulation with modified aerodynamic coefficients and inertia coefficients, respectively. It is observed that aerodynamic moment coefficients controls the exponential decay rate while the corresponding moment of inertia weighs on the frequency of the oscillations.

In the second part of the experiments, remaining airship parameters are estimated by performing low speed flight maneuvers such as turning left and right on the spot, altitude increase and decrease and moving forward and backward. Similar to natural response experiments, demonstrations are recreated in the simulation environment by applying the same pilot commands to the airship model. Aerodynamic force and moment coefficients and inertia are calibrated by comparing the simulated results to the test data until satisfactory results are obtained. The comparisons of the simulation results with the final values of the parameters and the experiment measurements are given in Figs. 5.16-5.17.

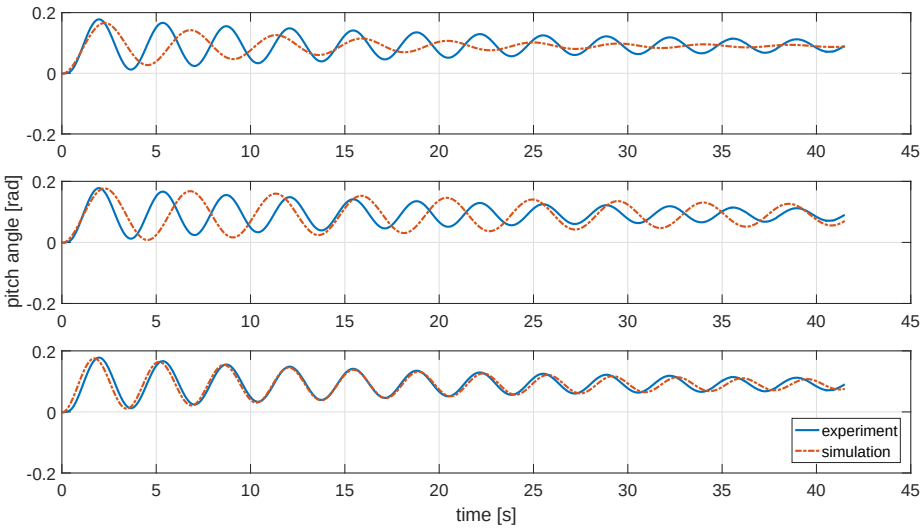


Figure 5.14. Natural pitch response of the blimp with respect to initial condition.

Final parameter values obtained through this procedure, which are shown in Table 5.1, provide satisfactory performance in order to accomplish the intended realistic flight simulator. It should be noted that what is meant by realistic is not to create an exact model of the actual indoor airship. Rather, the aim is to validate that the airship flight simulator behaves similar to an airship and it can be trusted to create more pilot demonstrations within the envelope of the experiments performed.

Table 5.1. MAEBlimp parameter values obtained through tests

c_u	2.00	I_{xx}	1.41
c_v	11.28	I_{yy}	10.33
c_w	10.75	I_{zz}	9.08
c_p	0.19	I_{xz}	1.64
c_q	0.29	θ_e	1.15°
c_r	1.41		

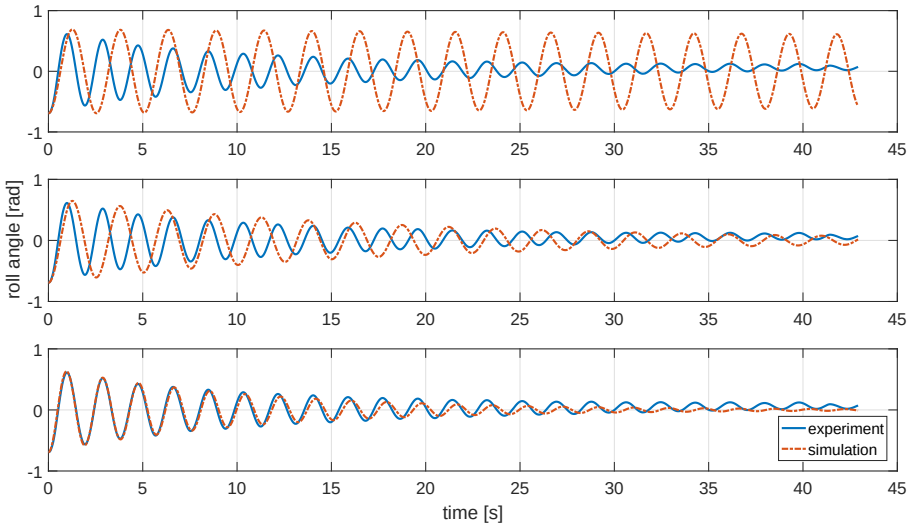


Figure 5.15. Natural roll response of the blimp with respect to initial condition.

5.3 Visualization and Communication

One of the main reasons for an airship flight simulator is to create a tool which enables performing human pilot demonstrations that cannot be performed under motion capture due to space restrictions. Main source of feedback for human pilot to decide on control actions is the visual cues they obtain by watching as the airship fly in the environment. For this reason, a visualization of the 3D position and orientation of the airship must be made available to the pilot in the simulator environment. In addition, the PWM inputs applied by the expert must be transferred to the simulation environment real time to provide the correct execution of the control actions.

5.3.1 Virtual Reality Tool

Simulink 3D Animation toolbox allows creation of virtual world blocks that contain 3D objects, scenery, sounds and lighting using VRML (Virtual Reality Modelling Language). Virtual world blocks can directly communicate with Simulink and the motion of the airship within the environment can be displayed on a screen in real

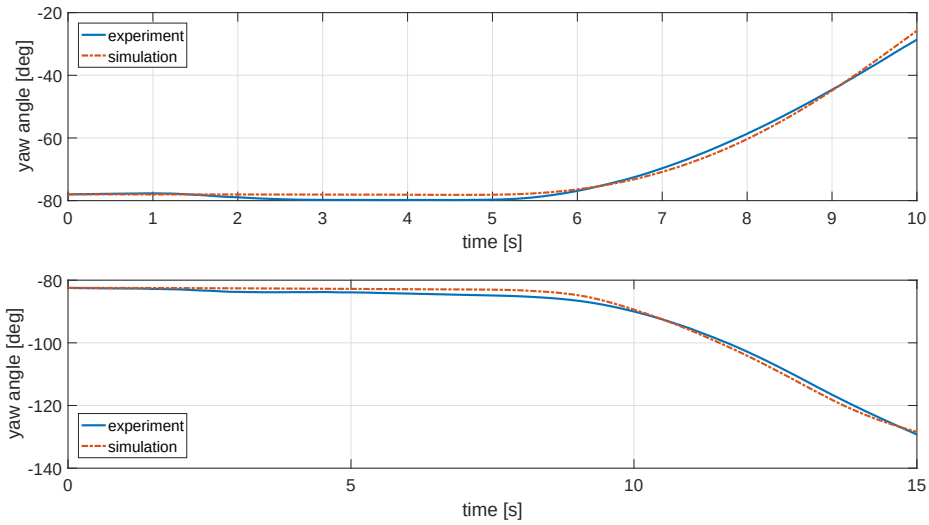


Figure 5.16. Comparison of test data and the simulation result of yaw angle for right and left turn.

time. Using the size and the geometry data of MAE blimp, a real size 3D model is created and placed in a virtual world as can be seen in Fig. 5.18. In addition, to help the expert perform desired demonstrations, a virtual target is created in the simulation. Virtual target is in the shape of a rectangular box that is large enough to surround the airship. Virtual target moves and rotates based on the specified desired/commanded trajectory of the airship. This provides a visual cue for the pilot to execute the commanded maneuvers in order to either keep the airship image within the moving and/or rotating virtual target box or fly into the target box stationary at a certain location in the virtual environment. This way the necessary control actions to follow the desired demonstrations are performed by the pilot in a more straightforward way.

5.3.2 Communication Interface

In order to perform the necessary flight maneuvers, pilot must be provided with a physical interface that conveys applied control commands into the airship

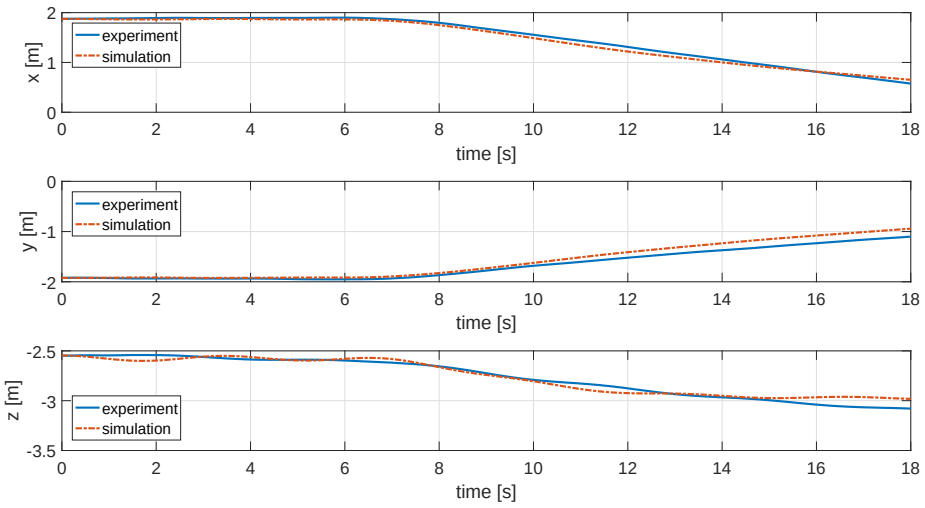


Figure 5.17. Comparison of test data and the simulation result of translational kinematics.

flight simulator. Devices such as keyboard or joysticks can easily be used for this purpose as many of them are already supported in Simulink. However, it is desired to facilitate the usage of the actual RC transmitter which is used by the pilot in the normal operation of the airship as well as in the thrust measurement experiments.

Each switch or stick on the RC transmitter creates PWM signals on individual channels and this signal can be reached on the corresponding channel of the RC receiver. Pulse Width Modulation (PWM) is a modulation method that is used to encode data into a pulsing signal which has two states as on or off. For one period of the pulse, duty cycle represents the proportion of on time to the off time. RC appliances such as servoactuators and ESCs employ a special PWM configuration where the period of the pulse is fixed at 20 ms (50 Hz) and the minimum and maximum duty cycle only varies between 5% (1 ms) and 10% (2 ms), corresponding to the allowable minimum and maximum limits of the RC appliance. This duty cycle limits are usually represented in microseconds in practice, therefore level of the PWM varies between 1000 and 2000 ideally. Signals on the receiver are processed by an Arduino

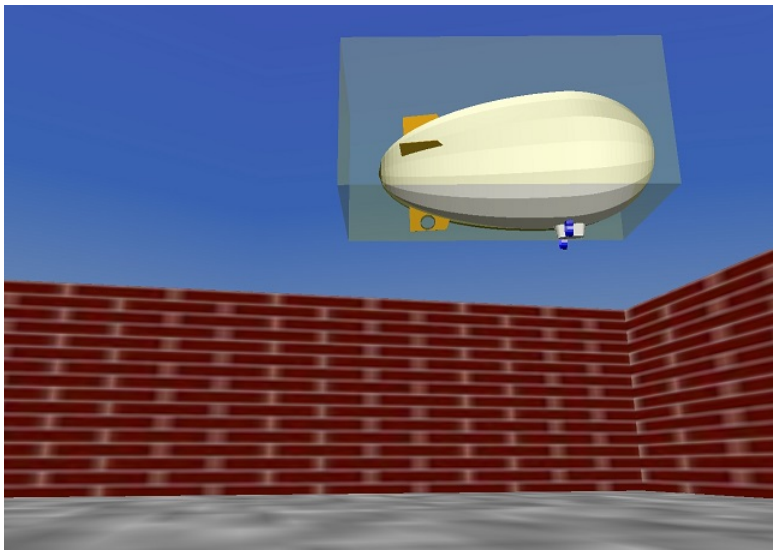


Figure 5.18. Visualization of indoor blimp and virtual target in the 3D virtual world.

Pro Mini [97] and the PWM values are transferred to the computer using USB connection. Using a custom Simulink block, this serial data is captured and forwarded to the Airship Flight Simulator to give full control authority of the simulated airship to the pilot using the RC transmitter. The hardware setup for the communication interface between the pilot and the airship flight simulator can be seen in Fig. 5.19.

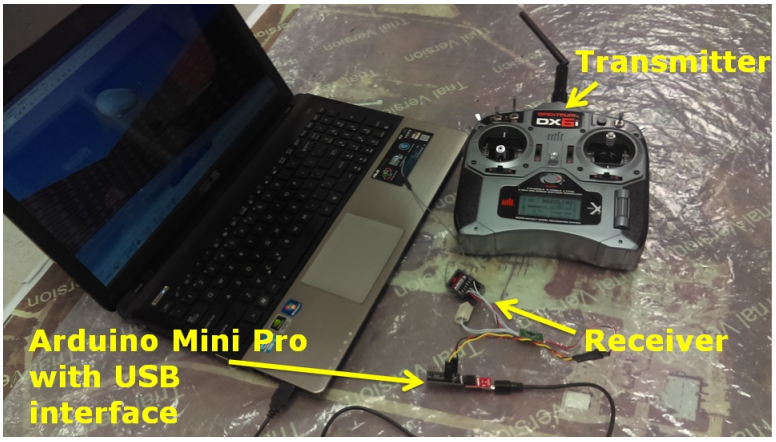


Figure 5.19. Hardware setup for the communication interface between the pilot and the airship flight simulator.

CHAPTER 6

Airship Control by LfD/RL

In this chapter, the LfD/RL method described in previous chapters are used to learn and execute low speed airship motion tasks. Airship flight simulator developed in the previous chapter is used in all stages of the LfD/RL method. In the first section, collected pilot demonstrations are presented and followed by the second section, which describes the exploration stage. In the third section, mission execution of waypoint commands demonstrated by the pilot, undemonstrated waypoint commands and a case when these waypoint commands are combined to represent a full mission are shown and results are discussed.

6.1 Pilot Data Collection

After the fidelity of the airship flight simulator is validated and the necessary visualization and communication interface is added, pilot can operate the airship in the simulator environment as can be seen in Fig. 6.1, in order to collect flight data that will be used in the *Pilot Demonstration* stage of the proposed three stage LfD/RL framework. Tasks for the pilot to execute are specified through the virtual target trajectory, or position and orientation, then the one of the two types of assignment is given to the pilot: (1) starting with the airship placed within the virtual target box, the pilot is required to keep the airship within the virtual target as the virtual target box moves through the specified trajectory, or (2) starting with the virtual target box and the airship placed at different positions and/or orientations, the pilot

is required to fly the airship into the virtual target box while the virtual target stays at the initial position and orientation.

In *Pilot Demonstration* stage, learning is performed while the pilot operates the airship and the training dataset is populated with the state-action pairs and corresponding action-value approximations. This process is completed in two consecutive stages. In the first part, pilot operates the airship and all flight data including the actions and the states are recorded. Q-Learning is performed in the second part by replaying the recorded flight data in repeated cases.

Airship longitudinal and lateral modes are separated for the learning of the demonstrated tasks and the mission executions. Two separate training datasets are created after the replay of the recorded pilot demonstrations. The dataset concerned with the longitudinal mode are set for 6 states, which are forward speed, u, vertical speed, w, pitch rate, q, pitch angle, θ, and x and z-axis position errors Δx and Δz with respect to the commanded target position. Actions in the longitudinal mode dataset are main thrust, T_M and the tilt angle, μ. Dataset concerning the lateral motion are set for 2 lateral states, which are yaw angle error, $\Delta\psi$ and yaw rate r, one control action, which is tail thrust T_T. LfD/RL subroutines in Chapter 3 are performed in the same manner for both datasets, however, kernel bandwidth parameter, h and weighting matrix used in the distance function in Eq. (3.2) differs. This phase is also used for tuning Q-learning parameters such as the feature weight matrix, $\mathbf{M}$ in Eq. (3.2) and bandwidth parameter, h in the kernel weightings in Eq. (3.32). In this section, the results of the first part are presented in terms of the time history plots of the recorded states and actions, and phase-portraits of the states with the action vectors superimposed as quiver plots. This is to visualize how the pilot take actions at different states during his "demonstrations" of completing a specific task.

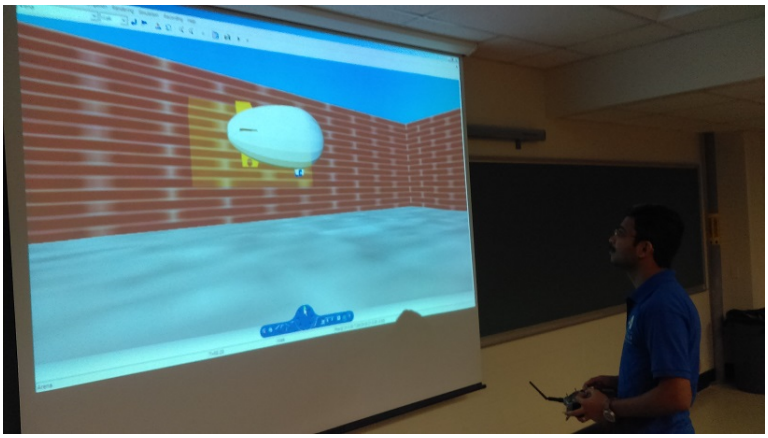


Figure 6.1. Pilot data collection using the airship flight simulator.

6.1.1 Translational and Rotational Speed commands

The first set of the pilot demonstrations are performed by commanding the virtual target to move with different values of forward speed, (u), vertical speed, (w) and yaw rate, (r). In each demonstration, only a single speed command is passed to the virtual target while rest of the desired speed values are kept at zero. Speed commands consists of two step functions applied with a phase lag which have same magnitude but opposite signs. This makes the commanded speed to stay constant for some time and return to zero after the time interval. Pilot is asked to achieve and hold this speed and go to a full stop at the end.

As the state-action space is high dimensional, it is not possible to visually present the dataset using conventional graphical methods. For this reason, only the states/actions which are the most relevant to the motion are selected to be the ones used in the visualization of the demonstrations, as shown in Figs. 6.2- 6.13. In these plots, first a time history of airship response with respect to the commanded speed is given. Remaining two plots present the phase-portrait and action quivers of the demonstration with the selected relevant features. In order to represent a cleaner visualization, the phase-portrait plots are separated into two parts. The middle figure

shows the phase-portrait of the first part of the demonstration when the pilot attempts to move the airship from the hover condition to track the constant commanded speed while the bottom/rightmost figure shows the last part of the demonstration when the pilot brings the airship from steady-state motion back to hover condition. In the two-dimensional plots, green line describes the states visited during the demonstrations, green dot indicates the starting state, and the red dot indicates the target state. Blue arrows are the representation of the pilot action in terms of thrust or moment vector computed from the main thrust magnitude and angle, or tail thrust magnitude, which are controlled by the pilot.

Figure 6.2 shows the demonstration results when the pilot is tasked to move the airship forward with a constant speed starting from still, and then to bring it back to full stop. The time history plot shows that the pilot has a reaction lag as he starts taking action a few seconds after the virtual target starts moving. This lag was likely contributed by the difficulty of perceiving the virtual target box motion as the visual cue. A similar reaction lag can be seen in the second phase when he is to bring the airship back to hover position. Another observation is the overshoot once the pilot starts his action, and the steady-state error in tracking the speed command. The first phase-portrait plot visualizing the first part of the demonstration can also clearly show the overshoot and the reason for the overshoot. The pilot initially applies too much thrust, which causes the overshoot, and continues applying the thrust in the same direction even after the speed is above the commanded speed. Later, the pilot cuts down on thrust and, due to the drag, the speed comes close to the command. This plot also show the induced motion in the vertical direction, w while the pilot tries to control the speed in the forward direction, u. The second phase-portrait plot shows the phase of the demonstration when the pilot is tasked to stop the airship. In this phase, the pilot rotates the main thruster and applies thrust in the opposite

direction to slow down and stop the airship. Once the forward speed is reduced to zero, the pilot seems to be rotating the thruster up to stop the induced motion in the vertical direction.

Figure 6.3 presents the pilot demonstration of moving the airship backward. The virtual target starts from still, and moves backward with constant speed of $u = -0.1$ m/s, and later stops. In general, the pilot actions and the airship responses seem similar to the forward motion case. For example, there are pilot reaction lag and overshoot as in the forward motion case. The pilot in this case seem to track the speed command better, with almost zero steady state error. However, he was not able to bring the airship to full stop at the end of 60 second demonstration episode. The vertical motion excited by the pilot action while trying to control the forward motion is also apparent in this case. Figs. 6.4 and 6.5 show similar forward and backward motion demonstrations with higher commanded speeds.

Figures. 6.6- 6.9 show the demonstrations of pilot, stating from hover condition, moving the airship up or down vertically with different speeds, and stopping eventually. The most distinct feature of vertical motion demonstrations as compared to the forward motion is the small amplitude oscillation at the steady state. Similar to the forward motion, vertical motion demonstrations also show overshoots, pilot response lag, and difficulty in bringing the airship to full stop.

Figures 6.10- 6.13 shows the pilot demonstrations of yawing the airship (turning around the vertical axis) to the left and right, and at different speeds. The pilot actions and airship response show similar general characteristics as the other demonstration cases. Overall, the pilot follows the turn and stop commands satisfactorily, and seems to be more successful bringing the airship to full stop as compared to the translational motion demonstrations.

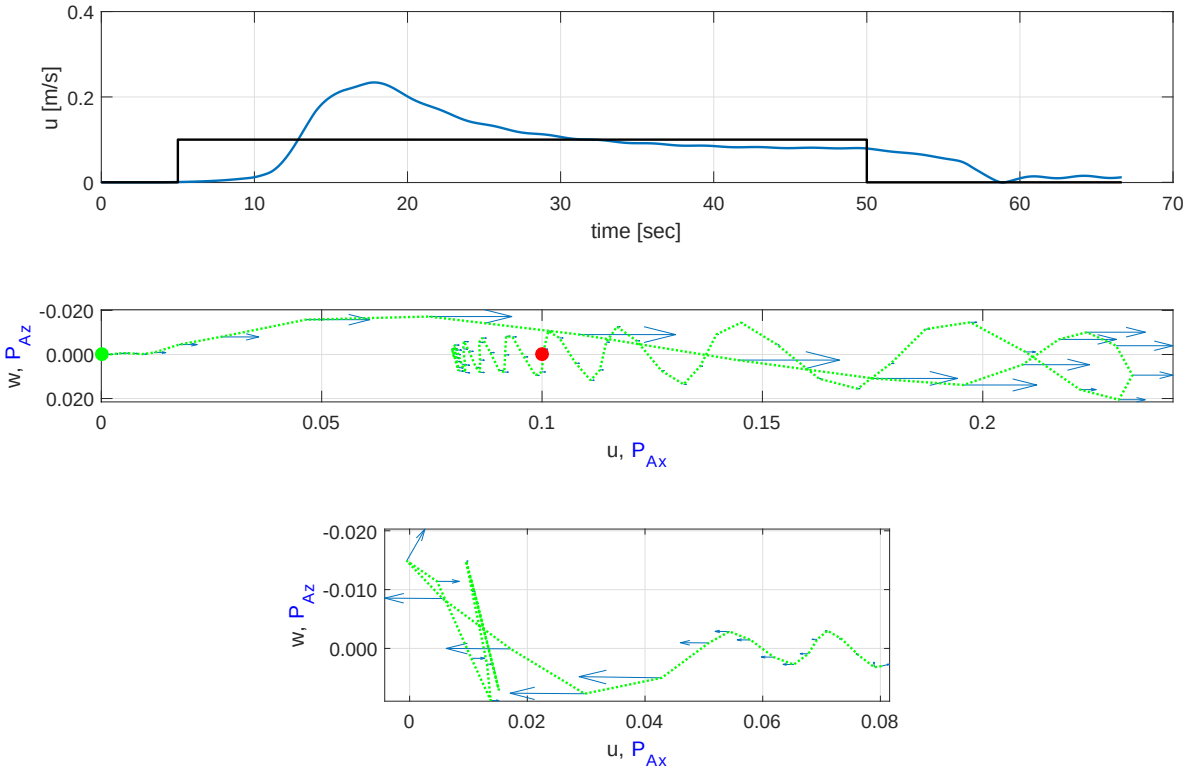


Figure 6.2. Pilot demonstration of forward speed command, $u = 0.1m/s$.

6.1.2 Position and orientation commands

In the normal flight operation, pilot uses a proactive control approach in which the response of the airship with respect to certain commands are predicted and the motion is pre-planned using prior experiences. Demonstrations applied in the form of speed commands prohibits the pilot to act in this manner. As a result, pilot performs reactive control actions which does not completely reflect the expertise of the pilot. For this reason, a second set of pilot demonstrations are performed by placing virtual target box at the desired position and orientations and asking the pilot to fly the airship, from its initial position and orientation, into the virtual target box.

Figures 6.14 and 6.15 show the demonstrations when the airship was moved forward to reach a target located at 7 m and 3.5 m ahead, respectively. As can be

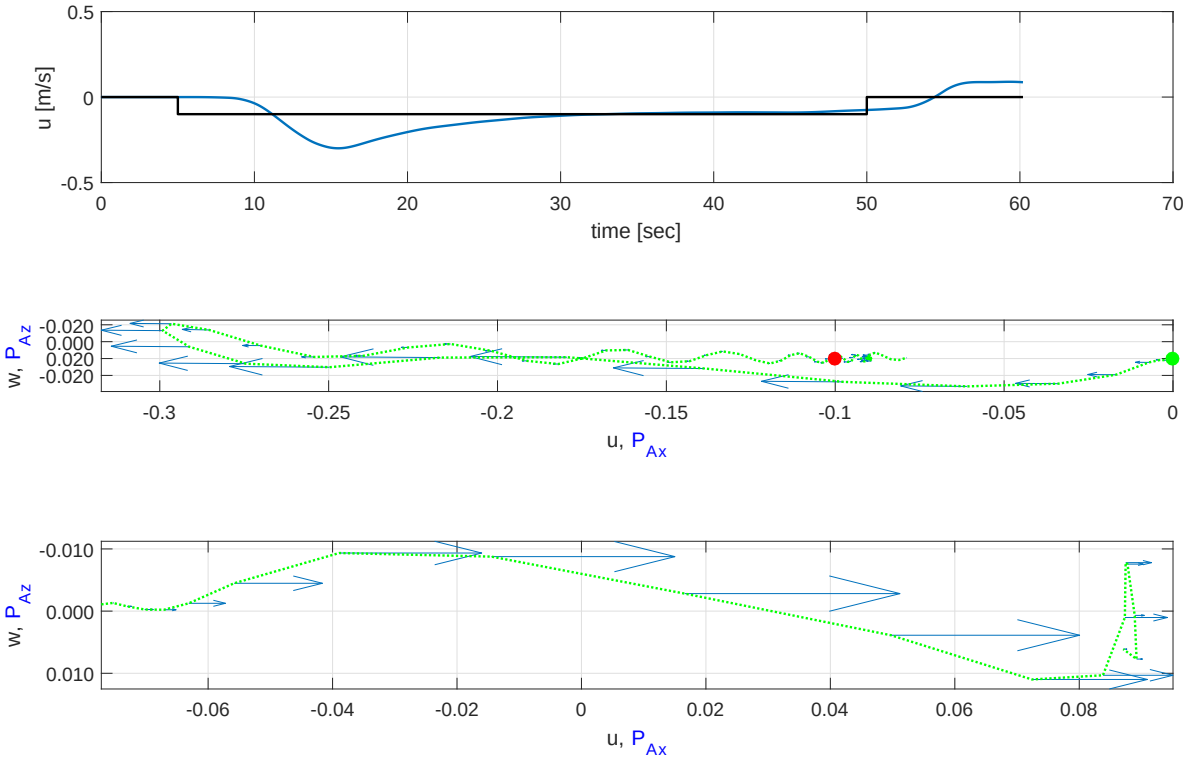


Figure 6.3. Pilot demonstration of backward speed command, $u = -0.1m/s$.

seen, the pilot response is slow and exhibits overshoots, oscillations and large steady state error. In the backward motion demonstrations shown in Figs. 6.16 and 6.17, targets are located -7 m and -3.5 m (7 m and 3.5 behind the airship), respectively. As the main thrusters cannot be tilted fully 180 degrees, the pilot needs to be slower and more careful not to diverge in the altitude. As a result, the backward demonstrations have low steady state error but slower response, overall.

Altitude demonstrations are performed when the airship starts initially at altitude of 5 m. In the altitude decrease demonstrations shown in Figs. 6.18 and 6.19, target positions are at 1 m and 3.5 m, respectively. In the learning algorithm, not the absolute altitude but the relative distance with respect to the target position is considered among the states. Thus, the pilot in these cases demonstrates decreasing altitude by 4 m and 1.5 m. Pilot performs these demonstration cases more accurately

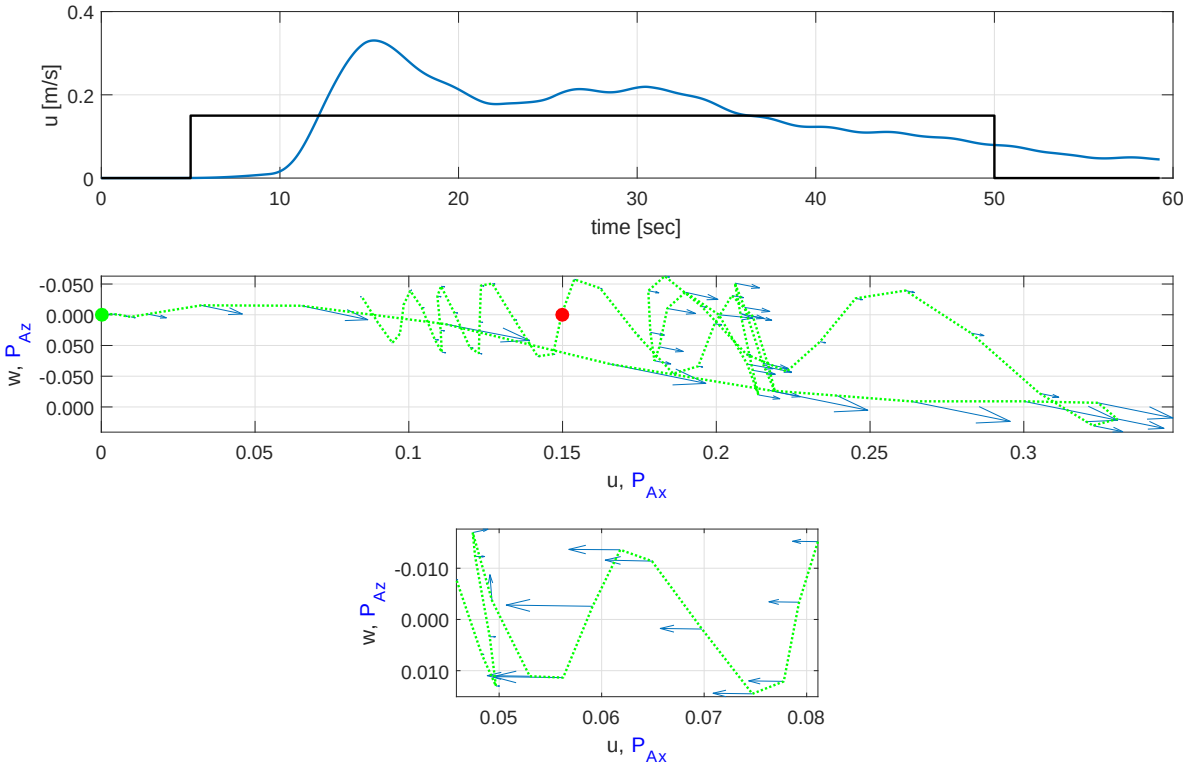


Figure 6.4. Pilot demonstration of forward speed command, $u = 0.15m/s$.

with no significant steady state error or oscillations. Figures 6.20 and 6.21 show the altitude increase demonstrations when the target is located at 6.5 m and 9 m, which correspond to 1.5 m and 4 m altitude increases. In this case, pilot once more performs well, although in the 1.5 m altitude increase case, the airship stops slightly above the target position.

Airship turn commands are shown in Figs. 6.22 and 6.23 for 45 and 90 degrees right turn, respectively and Figs. 6.24 and 6.25 for 45 and 90 left turn, respectively. Pilot demonstrations for turn cases are accomplished with almost no steady state error and relatively faster than the other demonstrations.

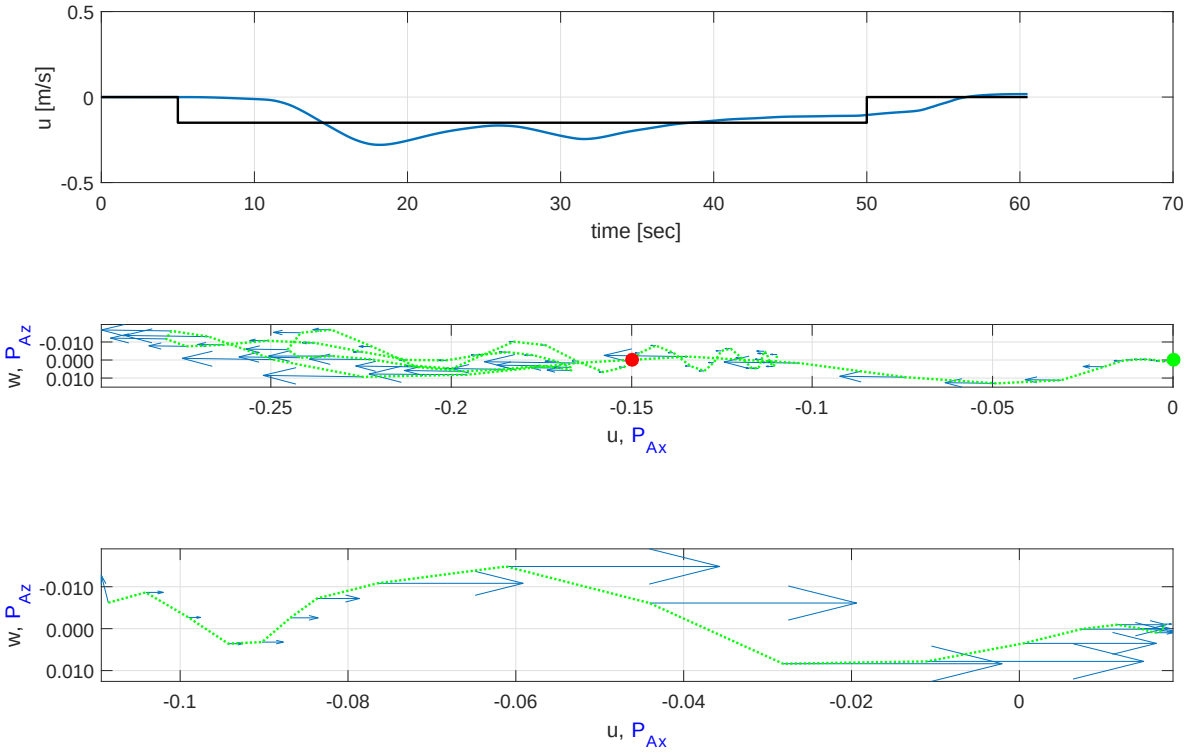


Figure 6.5. Pilot demonstration of backward speed command, $u = -0.15m/s$.

6.2 Exploration

In Section 3.5.3, a simple continuous state continuous action example was performed with three different sample cases and two different bandwidth parameters. In the end of these test cases, the conclusion drawn was that selection of correct kernel parameters had crucial impact on the performance of the learning. Random explorations are the main source to gain knowledge about the environment in the conventional RL setting. However, when the number of state-action pairs increase, the statistical properties of the dataset also changes and selected kernel parameters becomes inadequate. Thus, in the airship control case, random explorations are not performed, however the policies are attempted to be improved by incorporating more demonstrations to the training dataset.

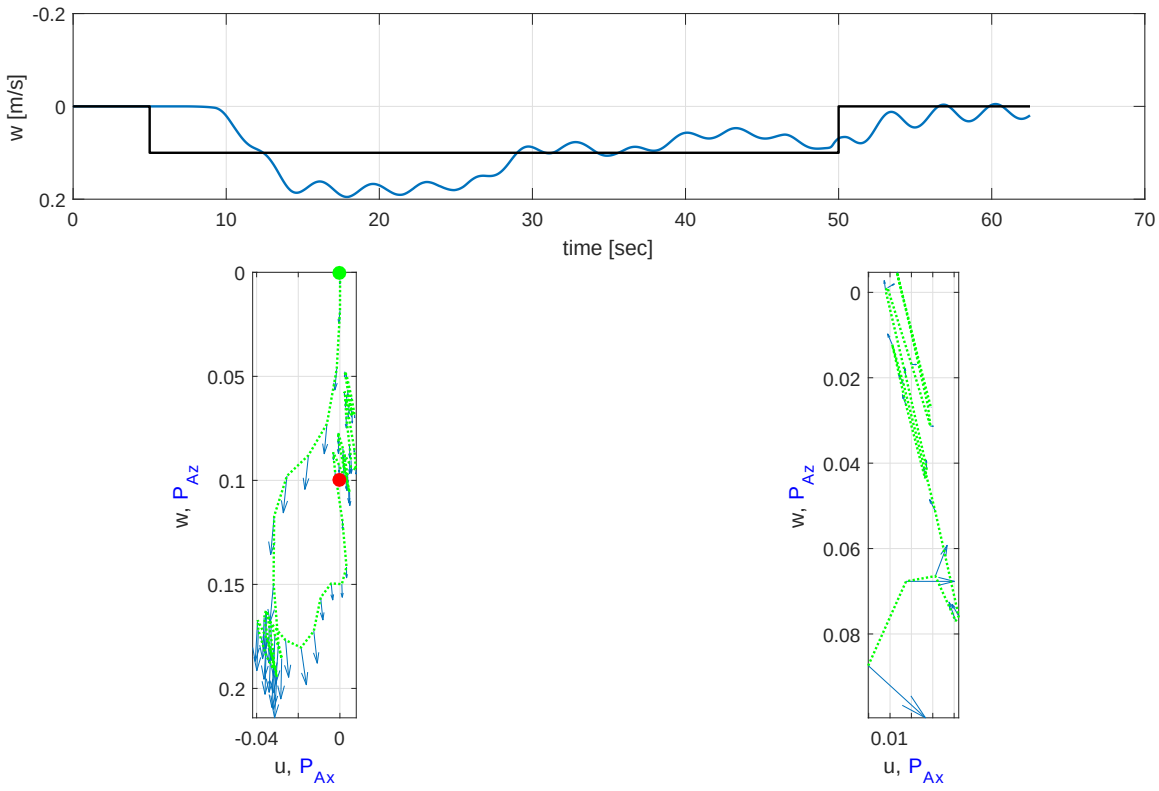


Figure 6.6. Pilot demonstration of downward speed command, $w = 0.1m/s$.

6.3 Mission Executions

In the *Mission execution* stage, control actions are generated to perform an assigned task using the optimal actions based on dataset obtained at the end of exploration stage. In this stage, learning continues parallel to the execution of the mission. However, since the dataset already covers most of the states and system is deterministic, new entries on the dataset are rarely generated and, mostly, action-values of existing points are updated. Performance of the learning framework can be judged by the demonstration results obtained in this stage by comparing the total values of undiscounted return of the mission executions, which is formulated in Eq. (2.1).

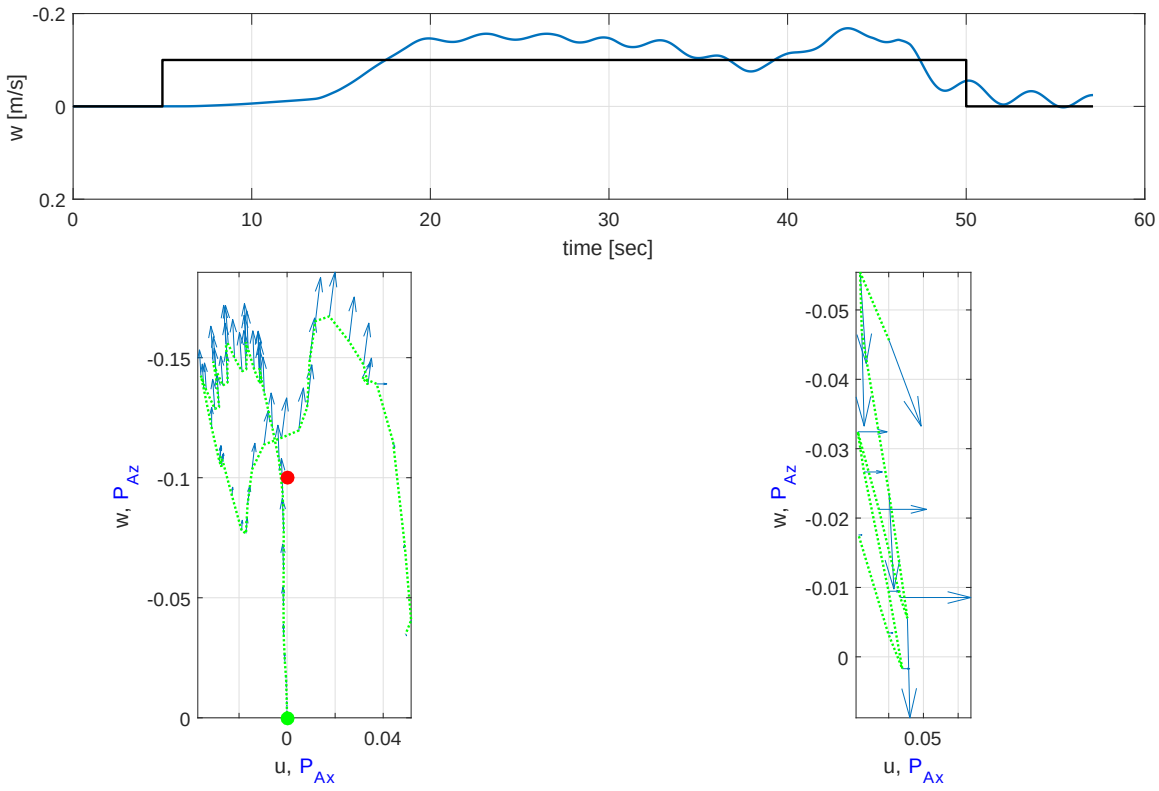


Figure 6.7. Pilot demonstration of upward speed command, $w = -0.1m/s$.

In order to test the capability of the learning framework on the airship simulator, three simulation cases are performed. In the first case, the airship, while initially at rest at 5m altitude is commanded to move forward on the x-axis by going from (0,0,-5) to waypoint (7,0,-5), which corresponds to one of the cases demonstrated by the pilot. In the first execution of the mission, only a single demonstration is used in order to create the dataset and action-values of the learning module. Fig. 6.26 shows the x-position response along with the commanded x-position in the two demonstration cases and two execution cases. In the execution cases, airship arrives the target position in about the same time. The first execution case has similar overshoot and slightly higher final error. In the second execution case, the airship shows slightly higher overshoot, but the final error is smaller, almost zero.

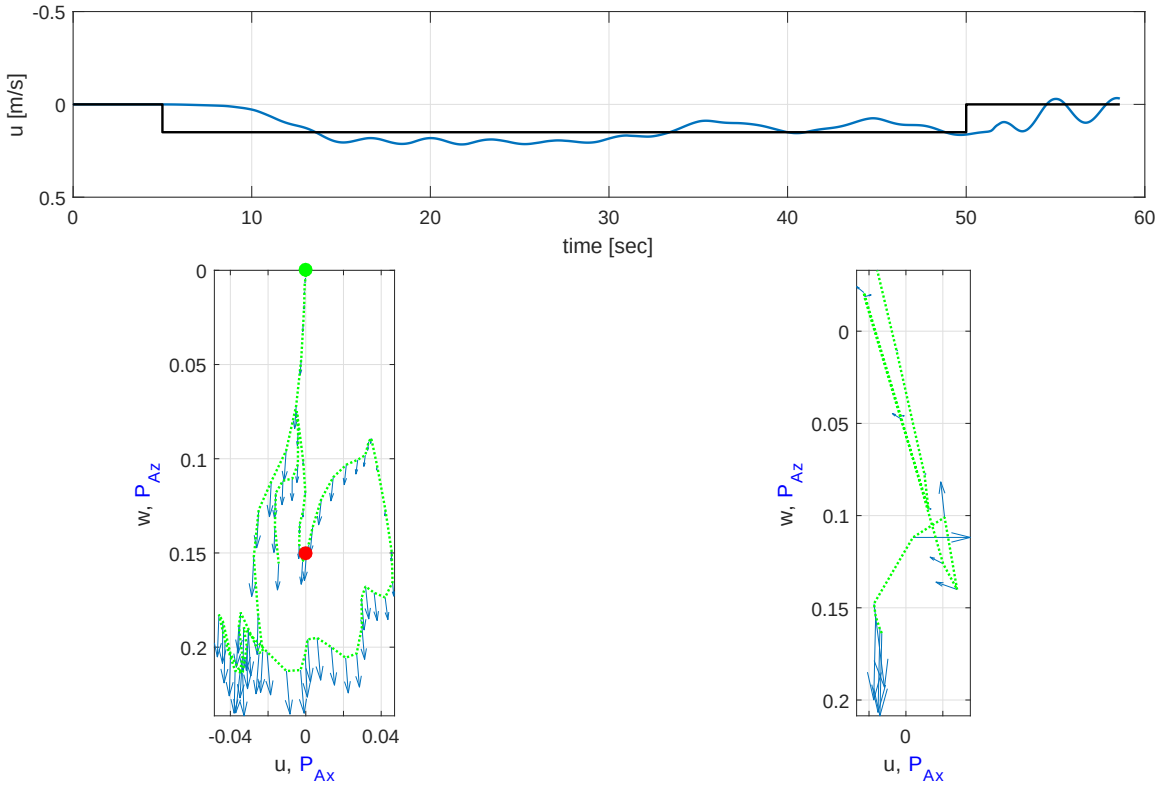


Figure 6.8. Pilot demonstration of downward speed command, $w = 0.15m/s$.

At this point, another learning demonstration for the waypoint (-3.5,0,-5) is added to the dataset and second mission execution is performed. In this case, response of the airship to the waypoint command becomes faster as can be seen by comparing trajectory and time history plots of previous execution and the pilot demonstration. Similar observation can be made by examining the undiscounted return shown in the Fig. 6.27. As can be seen, both execution demonstrations yield higher return throughout the whole mission execution and at the end of the mission, which indicates consistently better performance than that of the demonstration case.

Another observation is that although airship is able to reach the waypoint, it does not go into a full stop. This is due to several reasons. First reason is that even in the provided pilot demonstrations, airship speed is brought to a value close to zero

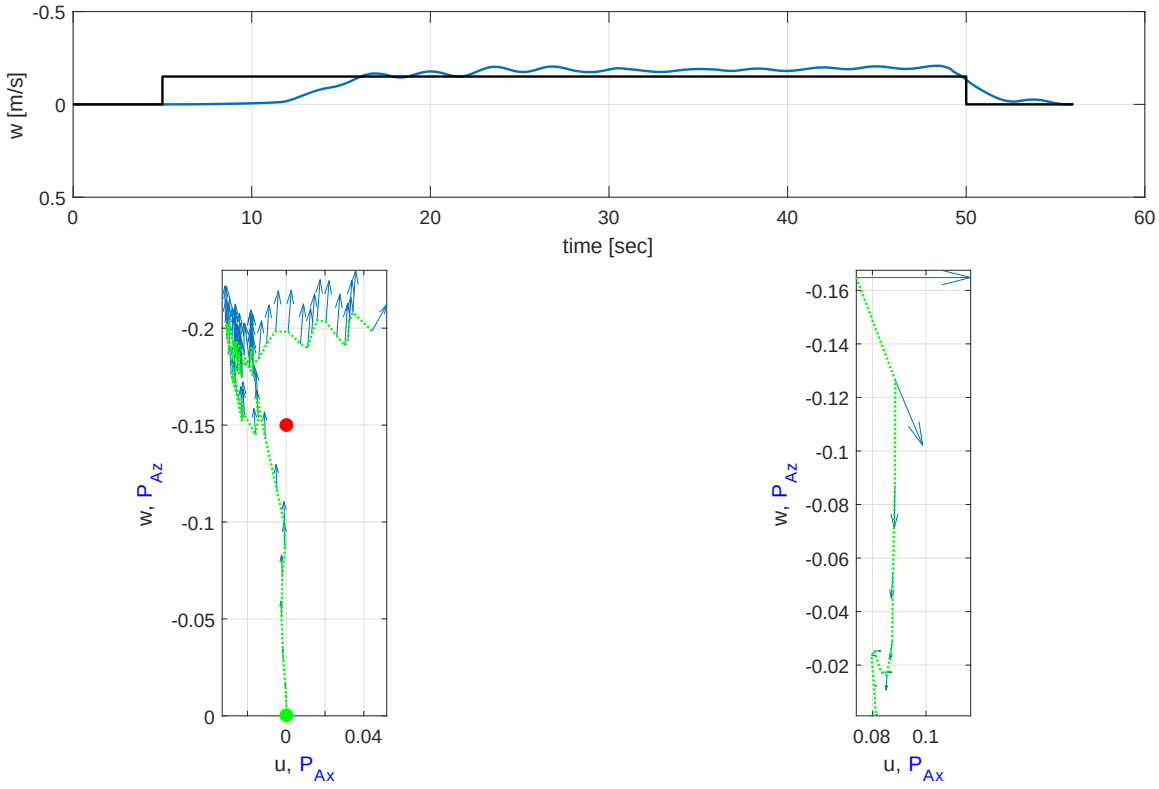


Figure 6.9. Pilot demonstration of upward speed command, $w = -0.15m/s$.

but fails to stop completely. Second reason is due to the geometric constraints of the airship thrusters. As the main thrusters can not travel back full 180 degrees, it is impossible to apply thrust in the negative direction without causing motion in the vertical direction.

In the second simulation case, airship motion in the vertical direction is considered. Similar to the previous case, airship is commanded to move to a waypoint which was demonstrated by the pilot. In this case, airship is commanded to go downward to (0,0,-1). As can be seen in Fig. 6.28, the first mission execution, using the dataset coming from the single demonstration exhibits an overall comparable but slightly slower response. After adding the second pilot demonstration to waypoint

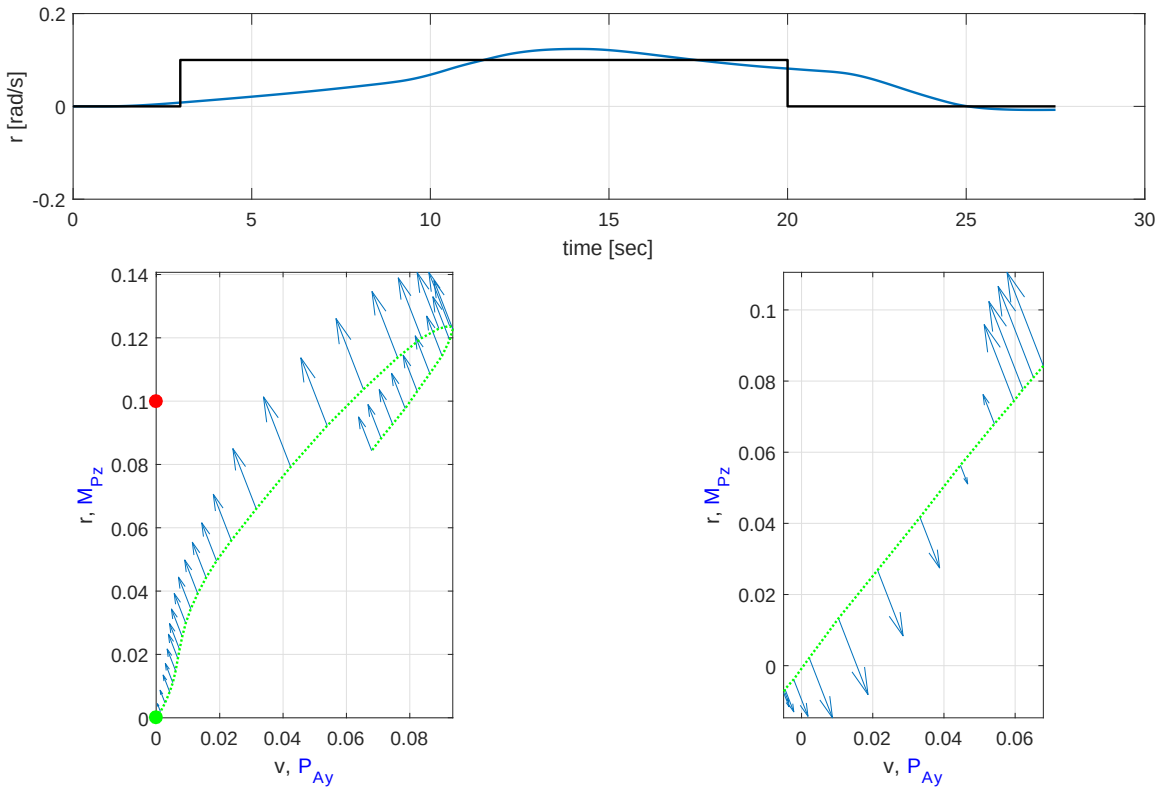


Figure 6.10. Pilot demonstration of right turn speed command, $r = 0.1 rad/s$.

(0,0,-6.5), the second mission execution performs faster than the first demonstration case although still slightly slower as compared to the demonstration case.

As can be seen in Fig. 6.29, in terms of the overall return value, the first execution case successfully completes the tasks although slightly worse at the end of the mission than the demonstration. Fig. 6.29 also shows that the second demonstration case results in better performance than the demonstration case. In addition, it is possible to command a waypoint which was not demonstrated directly but within the pilot demonstration range. In Fig. 6.30, performance of the learning algorithm in executing a task of moving to an undemonstrated waypoint at (0,0,-2) can be seen.

In the third simulation case, lateral motion of the airship is considered in a similar manner with the forward and vertical motions of the airship. In Fig. 6.31,

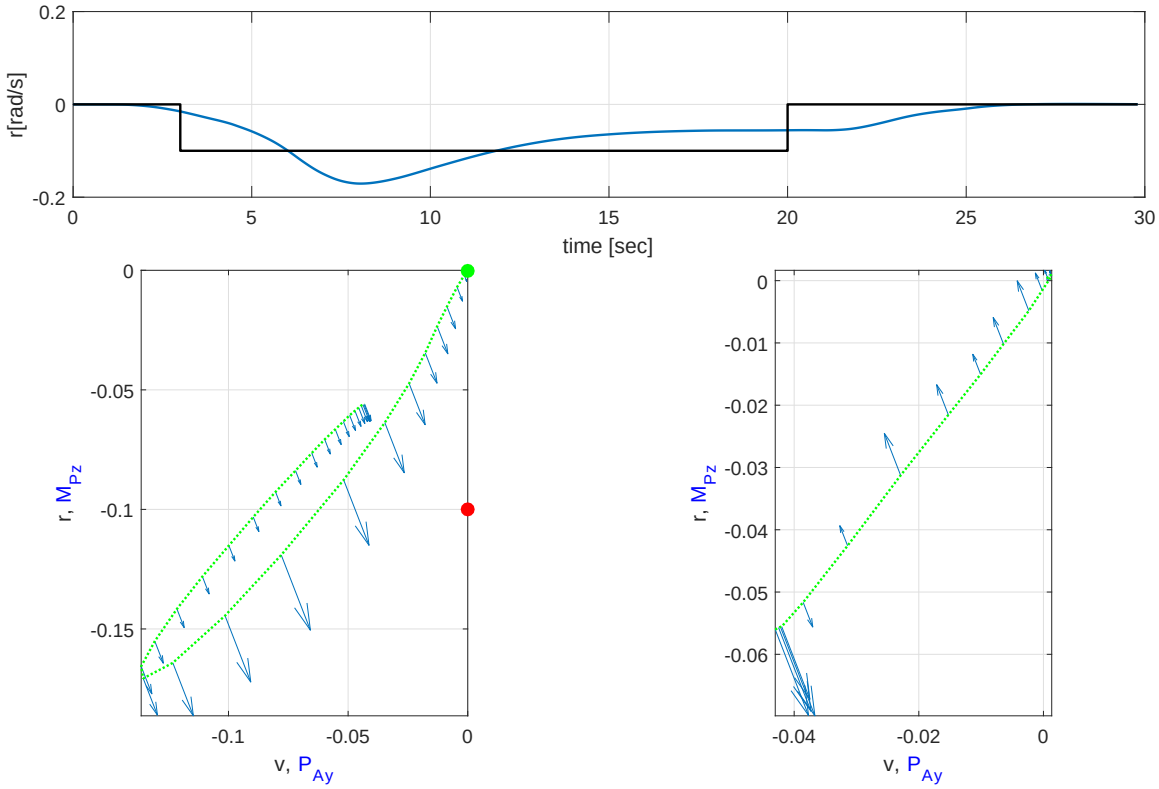


Figure 6.11. Pilot demonstration of left turn speed command, $r = -0.1 rad/s$.

comparison of first and second mission executions of a directly demonstrated waypoint can be seen. Once again, adding the additional demonstration results in slightly better performance as can also be seen in Fig. 6.32. Fig. 6.33 shows the successful performance of the learning algorithm executing an un-demonstrated task of turning left by 30 degrees.

In the final case, presented in Figs. 6.34, 6.35 and 6.36, a full mission that makes use of the both longitudinal and lateral mission execution is carried out. The mission starts with the airship at rest at 5 m altitude. The airship is first commanded to turn left by 30 degrees, followed by a longitudinal command to move forward by 7 m, and finally a vertical command of descending 2 m. The three commands are not

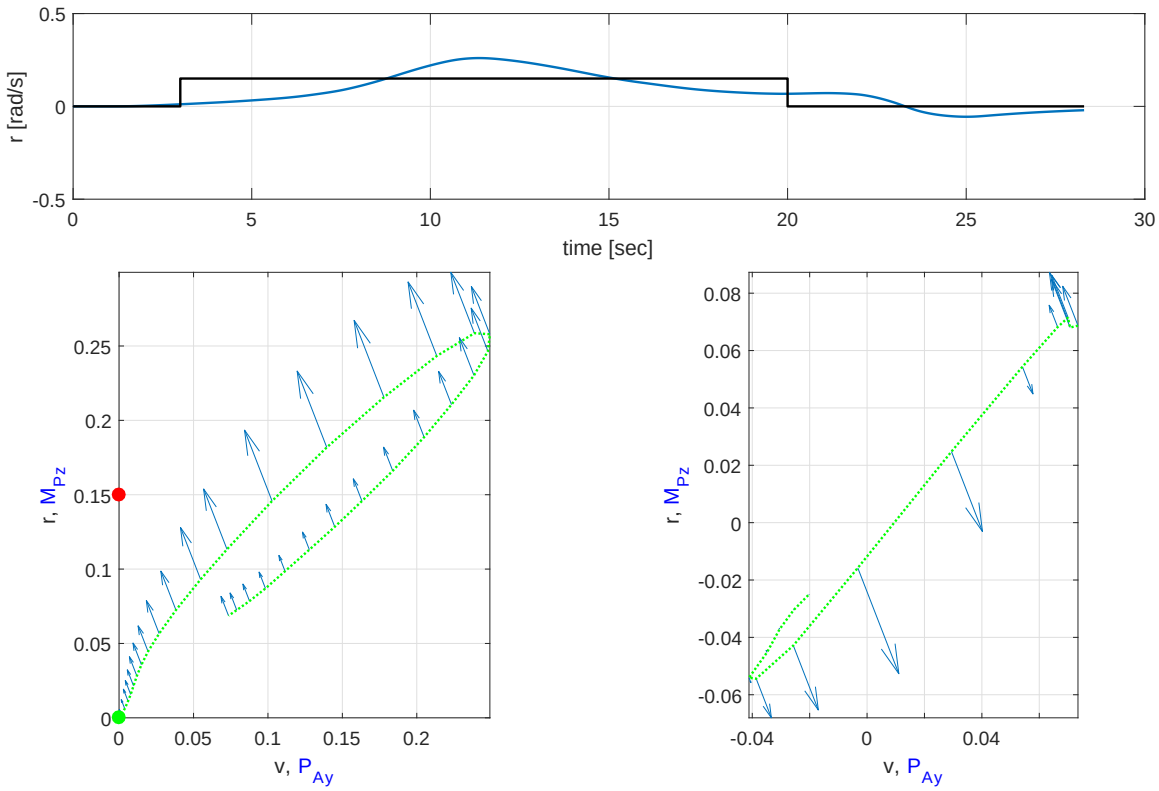


Figure 6.12. Pilot demonstration of right turn speed command, $r = 0.15 rad/s$.

given simultaneously; instead, the airship is commanded to execute each command after the previous one is executed.

Simulation results are summarized in orientation and position time histories in Figs. 6.34, and 6.35, respectively, and the applied actions in 6.36. As can be seen, the airship successfully executes the 30 deg. left turn command and holds the target heading through the rest of the mission. Afterwards, the airship starts moving forward in x-axis in order to reach the target position, which is 7 m ahead. However, as the airship moves forward, applied actions also cause a change in the altitude and by the time airship reaches 7 m, airship altitude has decreased by 0.7 m. In the final segment of the mission, the airship successfully descends to the commanded altitude of 2 m while the altitude maneuver causing deviation from the commanded x-position.

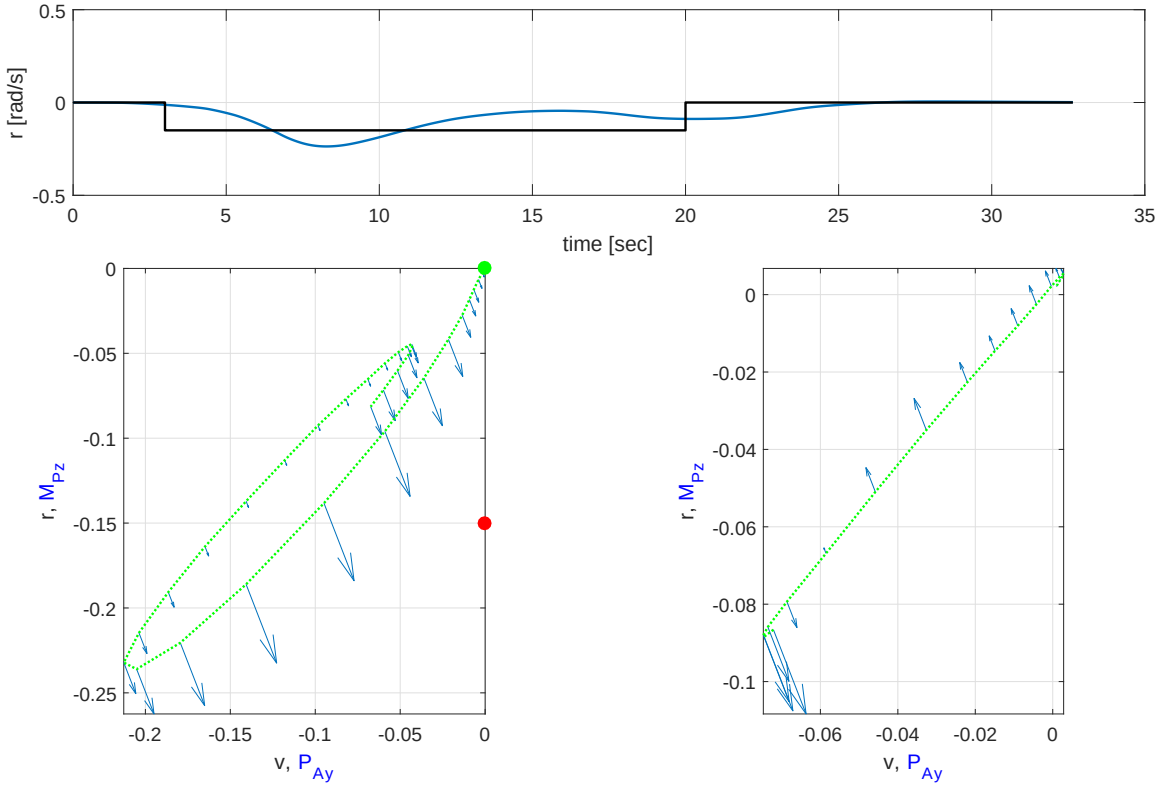


Figure 6.13. Pilot demonstration of left turn speed command, $r = -0.15 rad/s$.

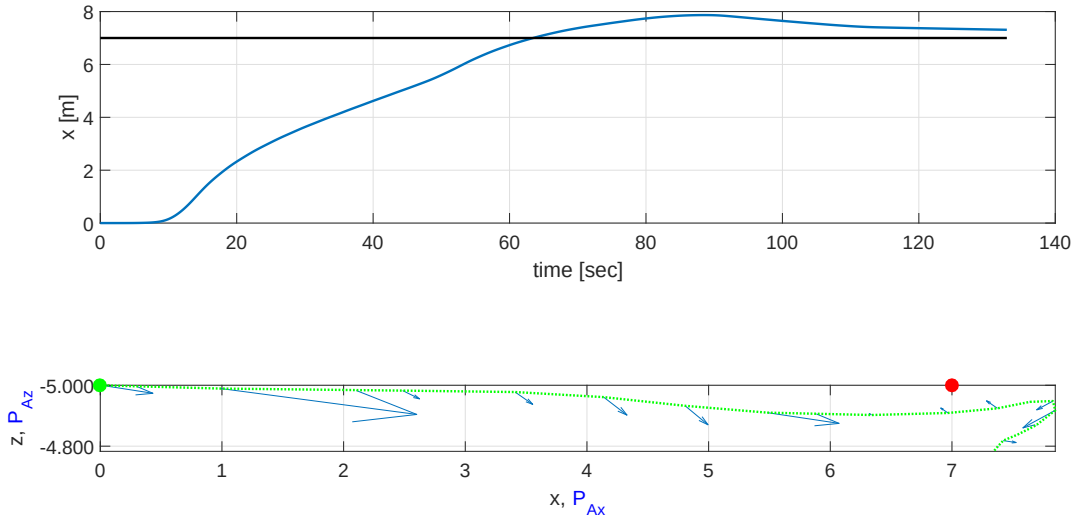


Figure 6.14. Pilot demonstration of forward waypoint, $x = 7m$.

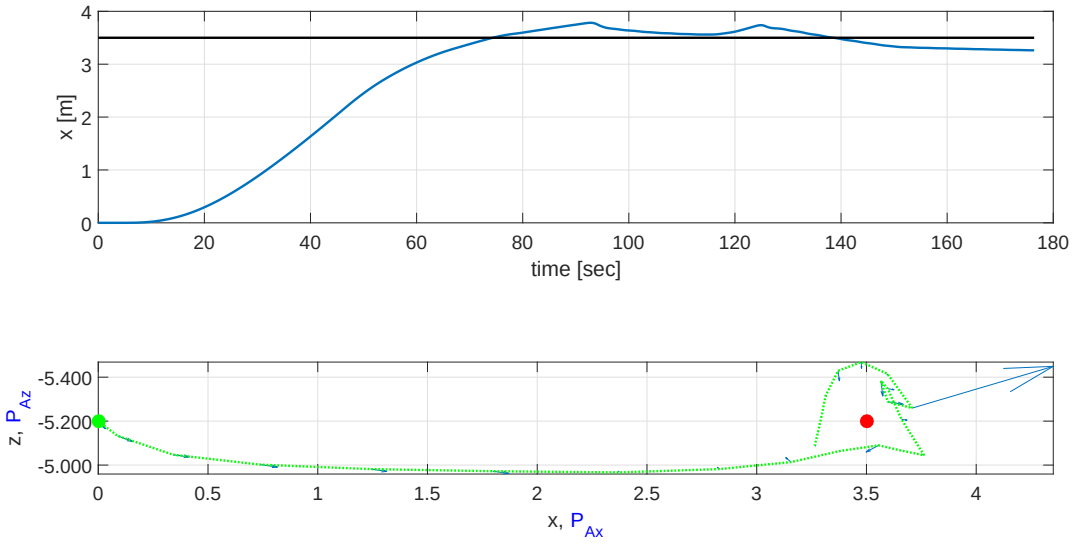


Figure 6.15. Pilot demonstration of forward waypoint, $x = 3.5m$.

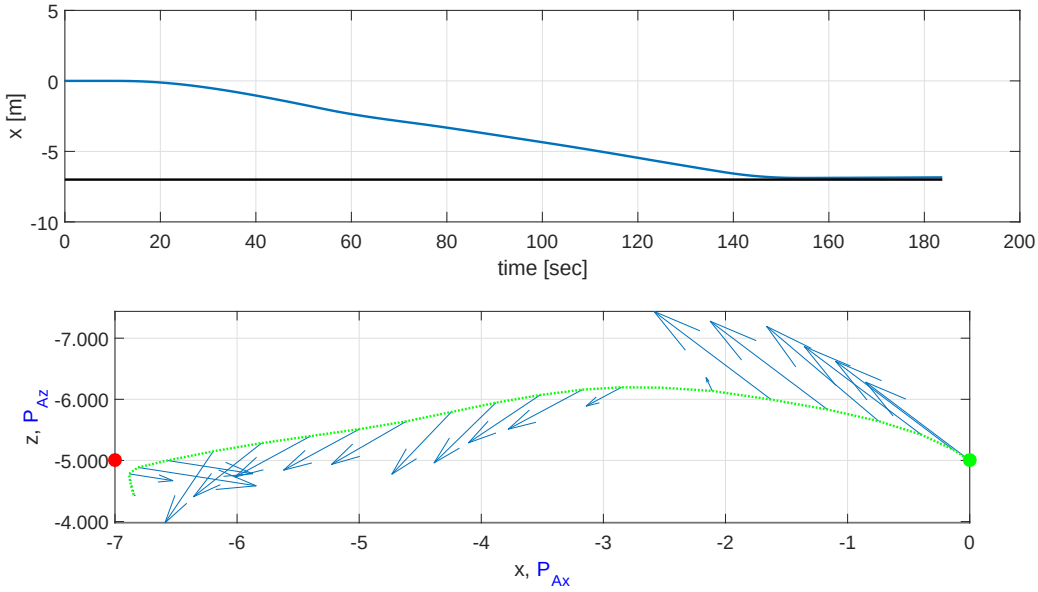


Figure 6.16. Pilot demonstration of backward waypoint, $x = -7m$.

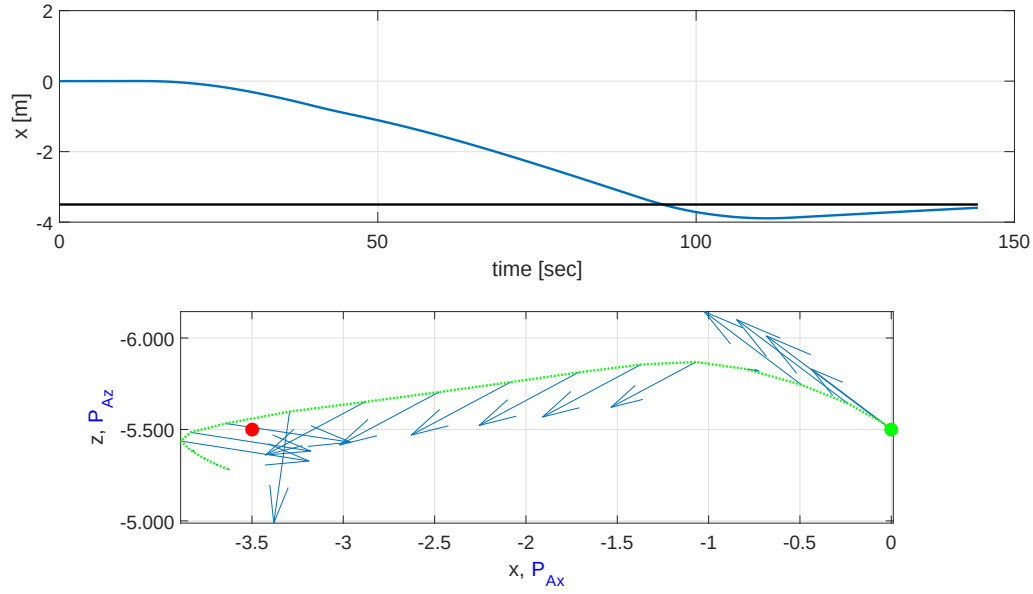


Figure 6.17. Pilot demonstration of backward waypoint, $x = -3.5m$.

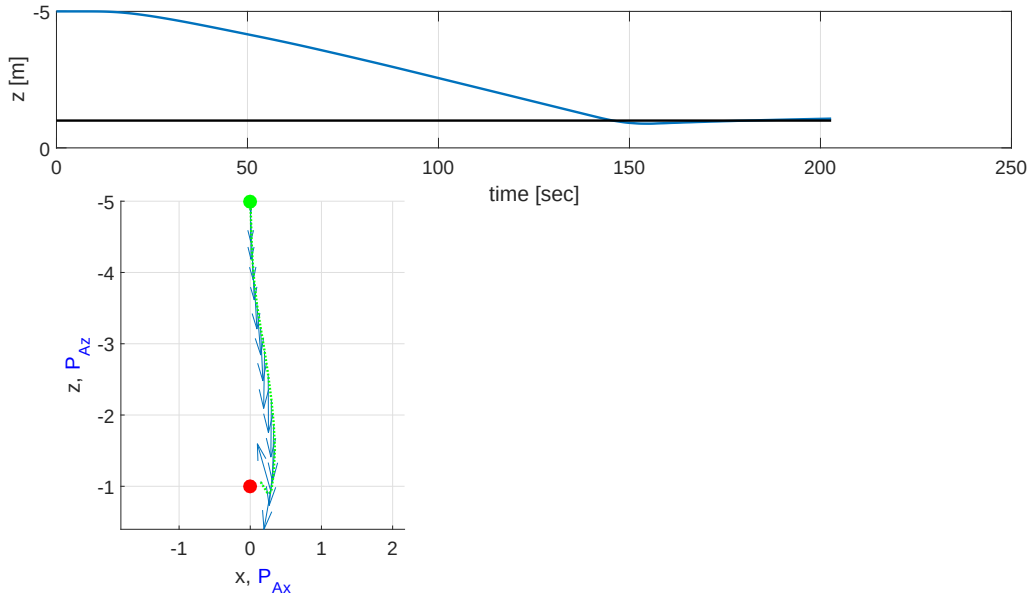


Figure 6.18. Pilot demonstration of downward waypoint, $z = -1m$.

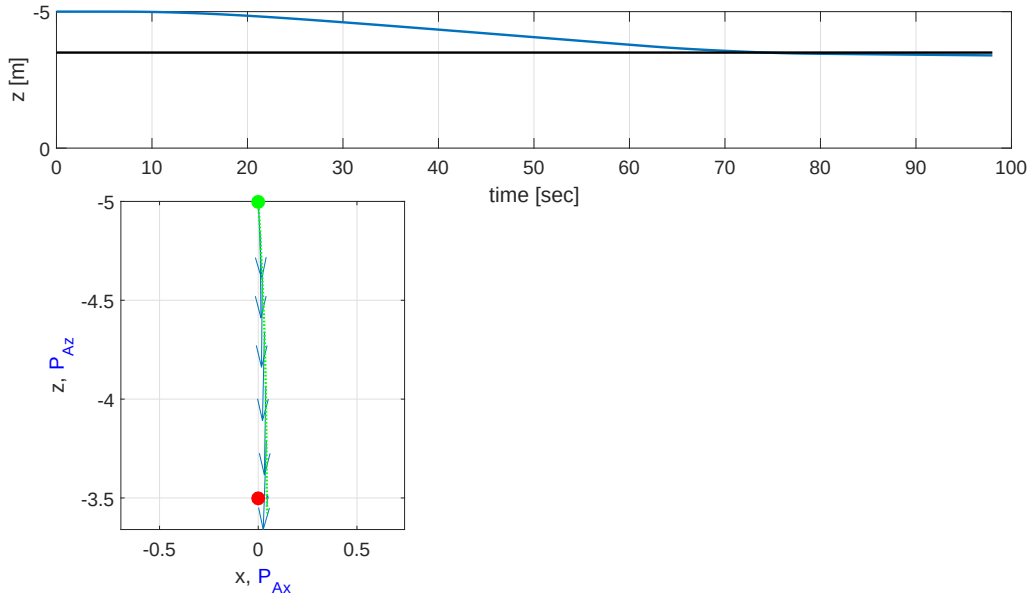


Figure 6.19. Pilot demonstration of downward waypoint, $z = -3.5m$.

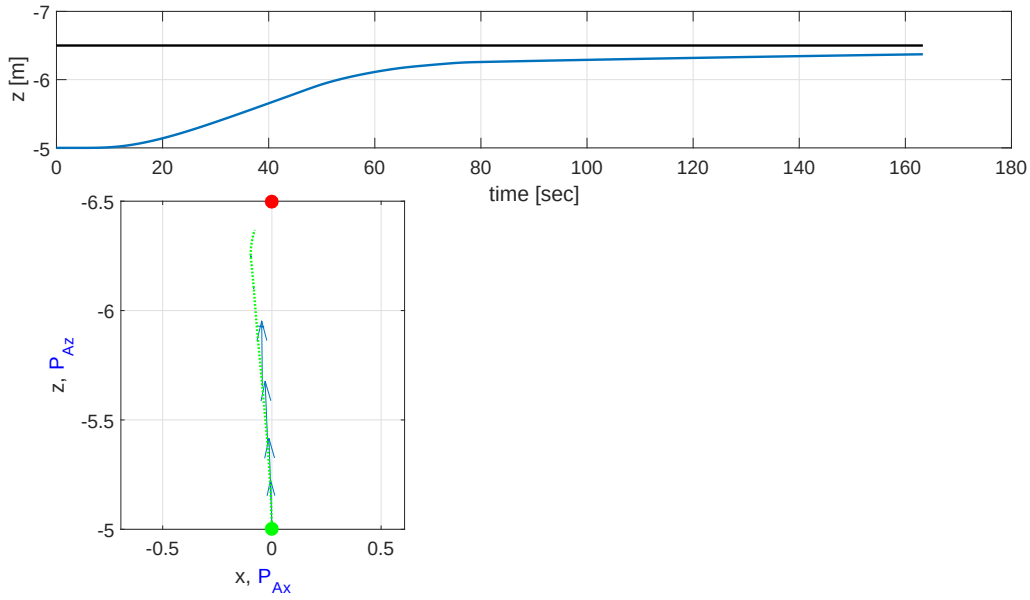


Figure 6.20. Pilot demonstration of upward waypoint, $z = -6.5m$.

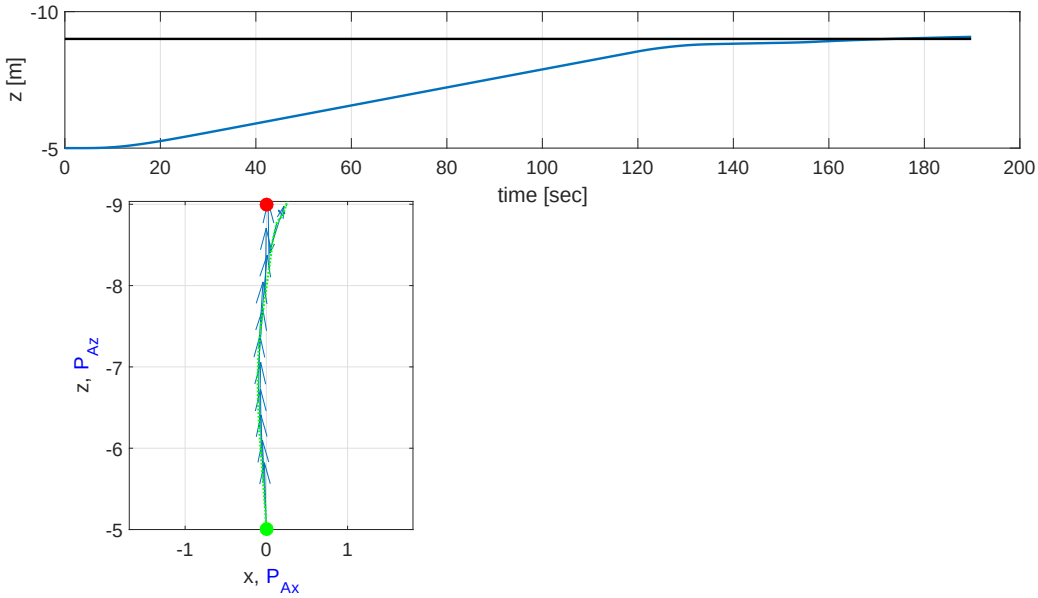


Figure 6.21. Pilot demonstration of upward waypoint, $z = -9m$.

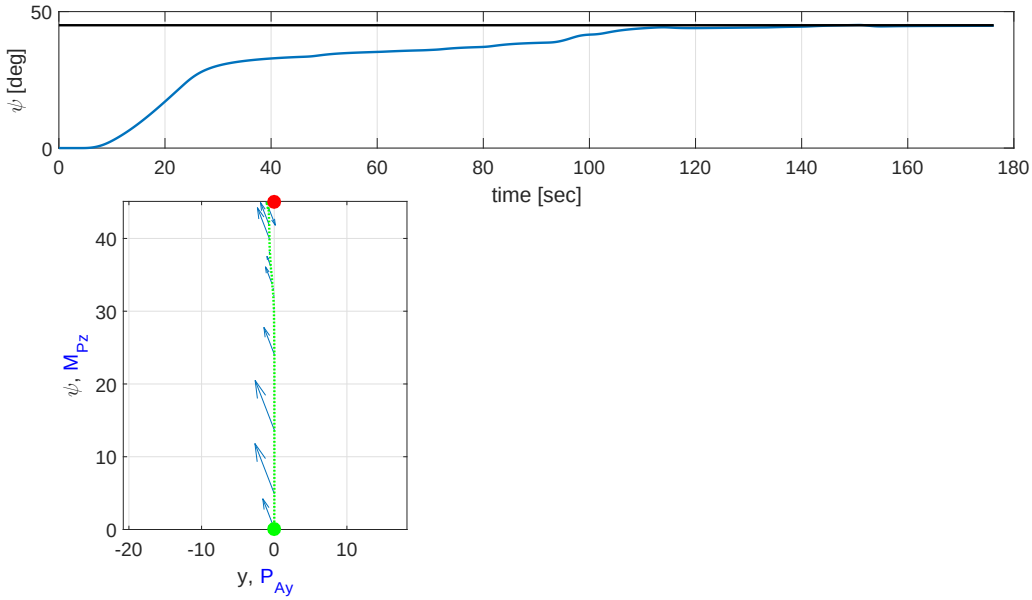


Figure 6.22. Pilot demonstration of right turn waypoint, $\psi = 45deg$.

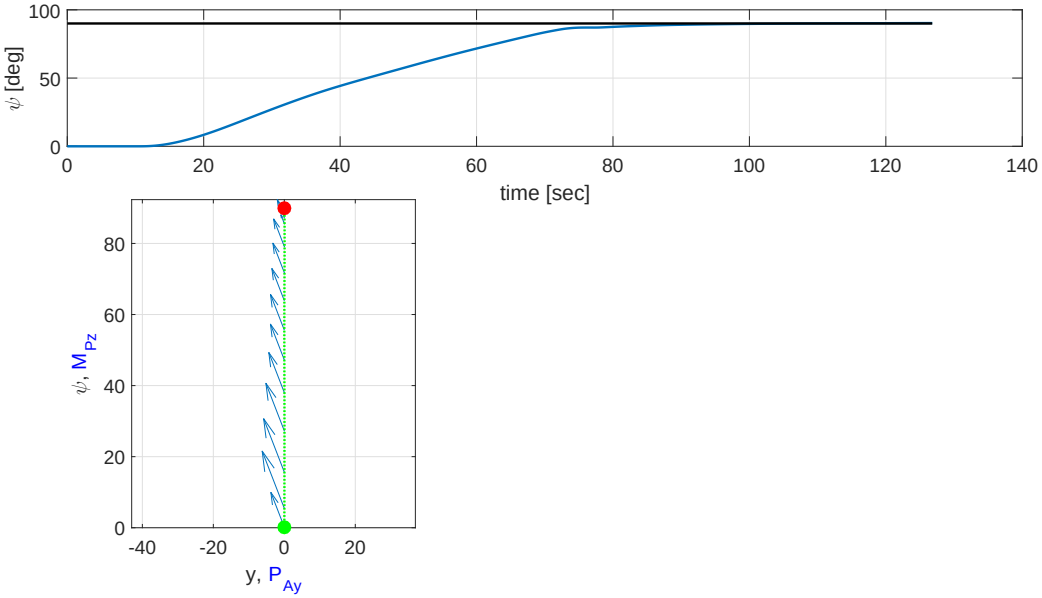


Figure 6.23. Pilot demonstration of right turn waypoint, $\psi = 90deg$.

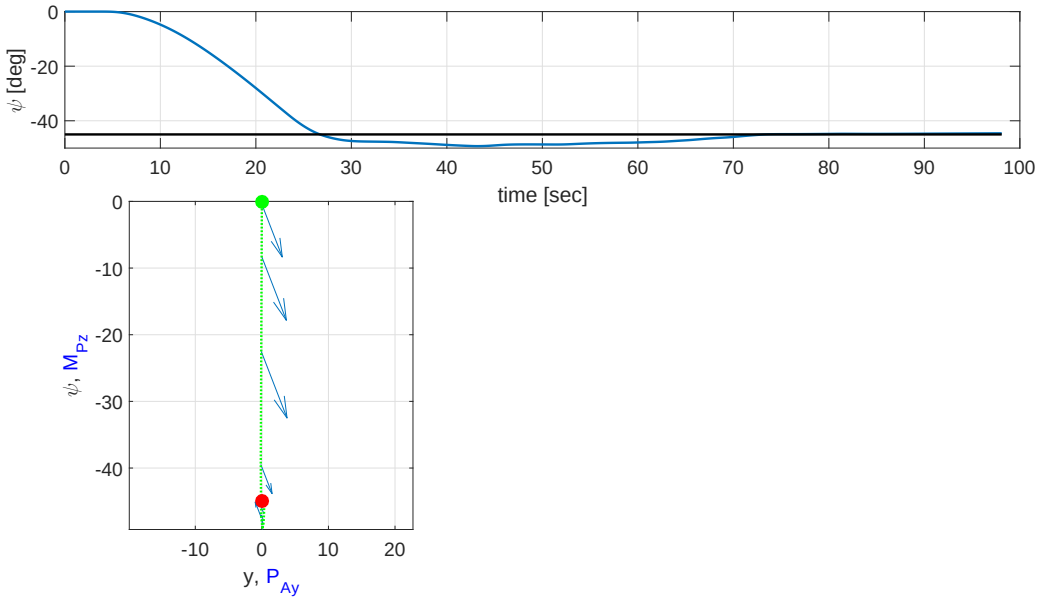


Figure 6.24. Pilot demonstration of left turn waypoint, $\psi = -45deg$.

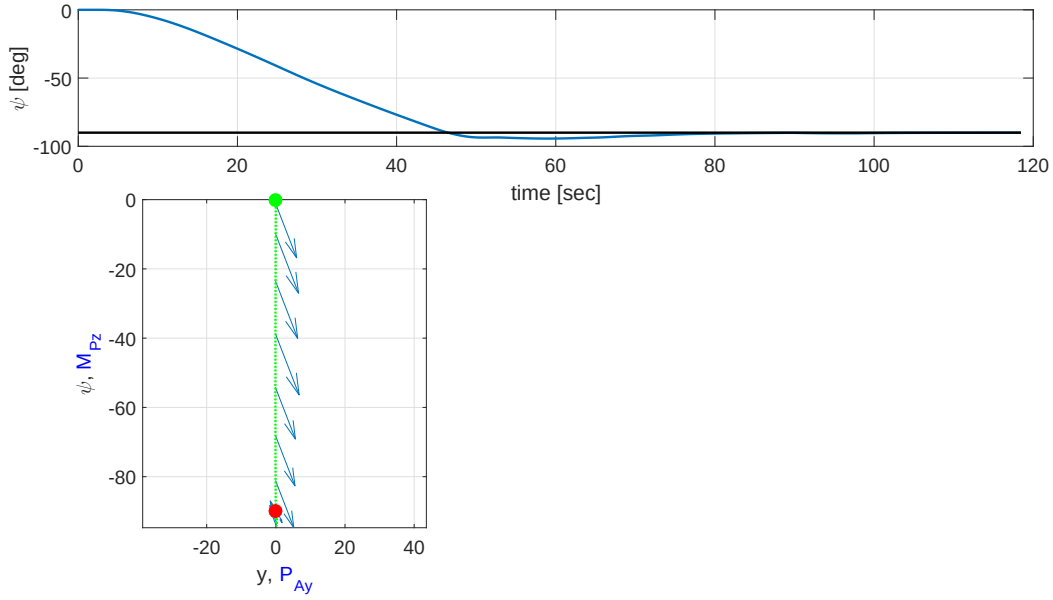


Figure 6.25. Pilot demonstration of left turn waypoint, $\psi = -90deg$.

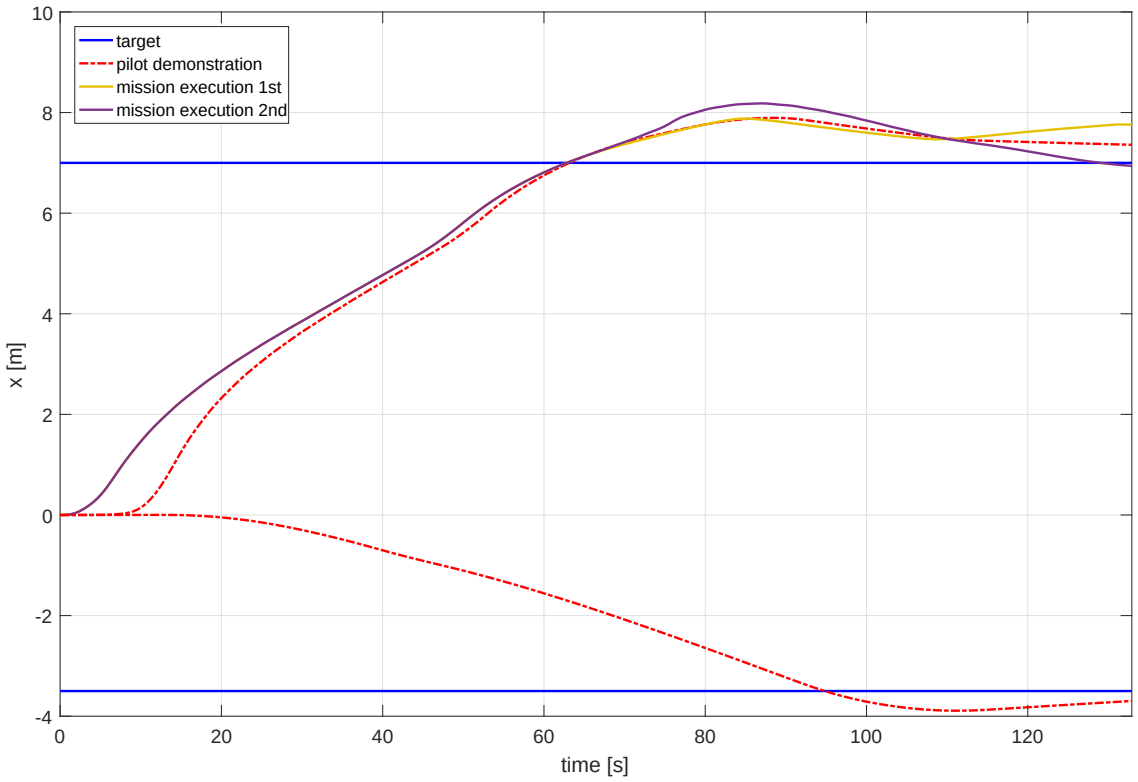


Figure 6.26. Mission execution stage for the forward motion of the airship.

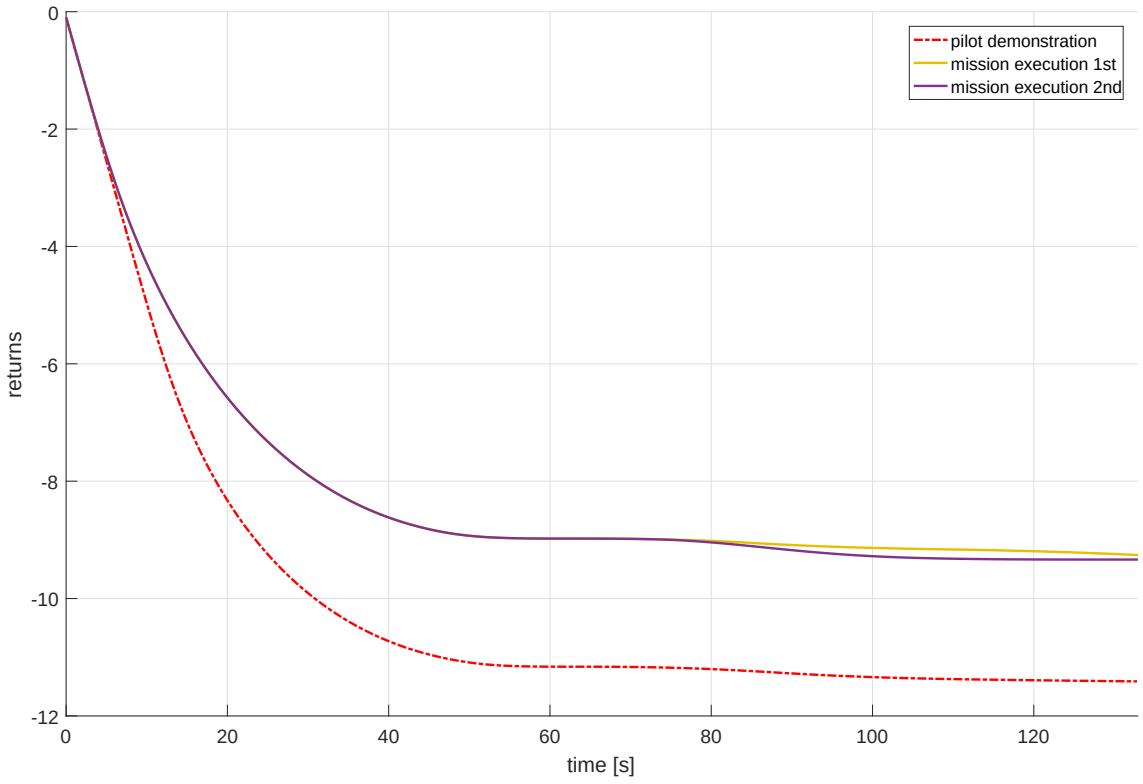


Figure 6.27. Returns achieved during the simulation cases in the mission execution stage for the forward motion of the airship.

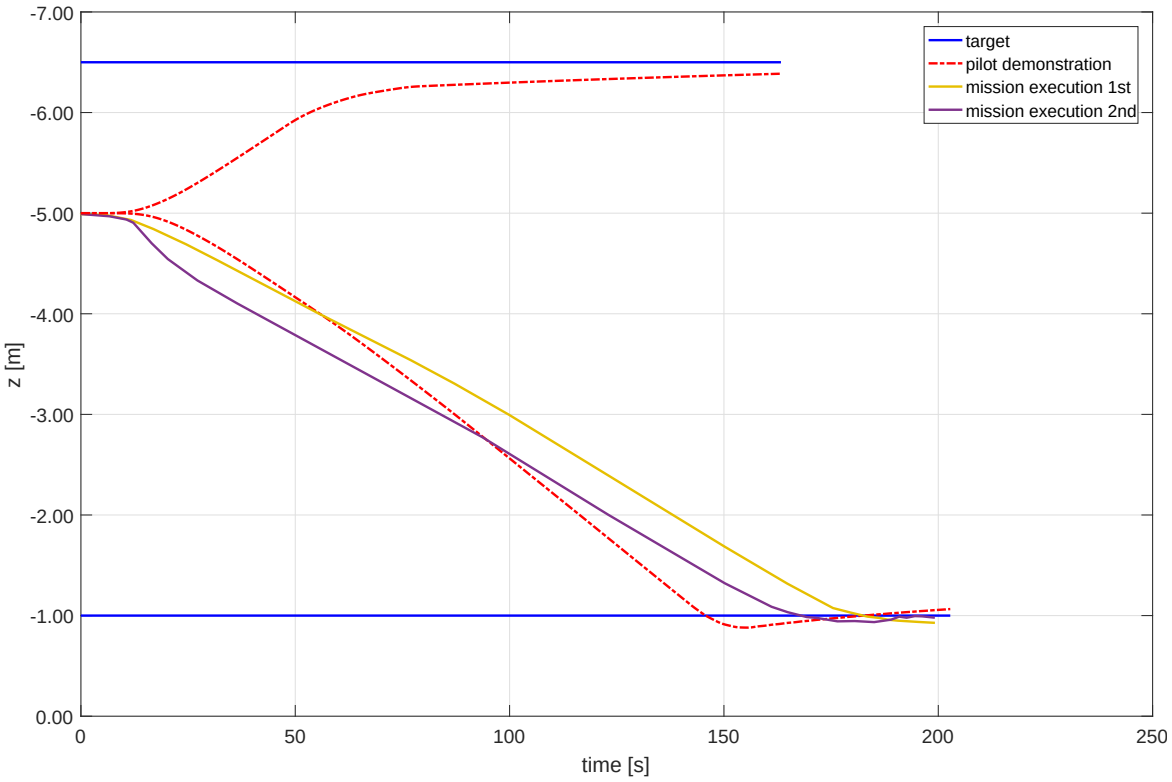


Figure 6.28. Mission execution stage for the vertical motion of the airship.

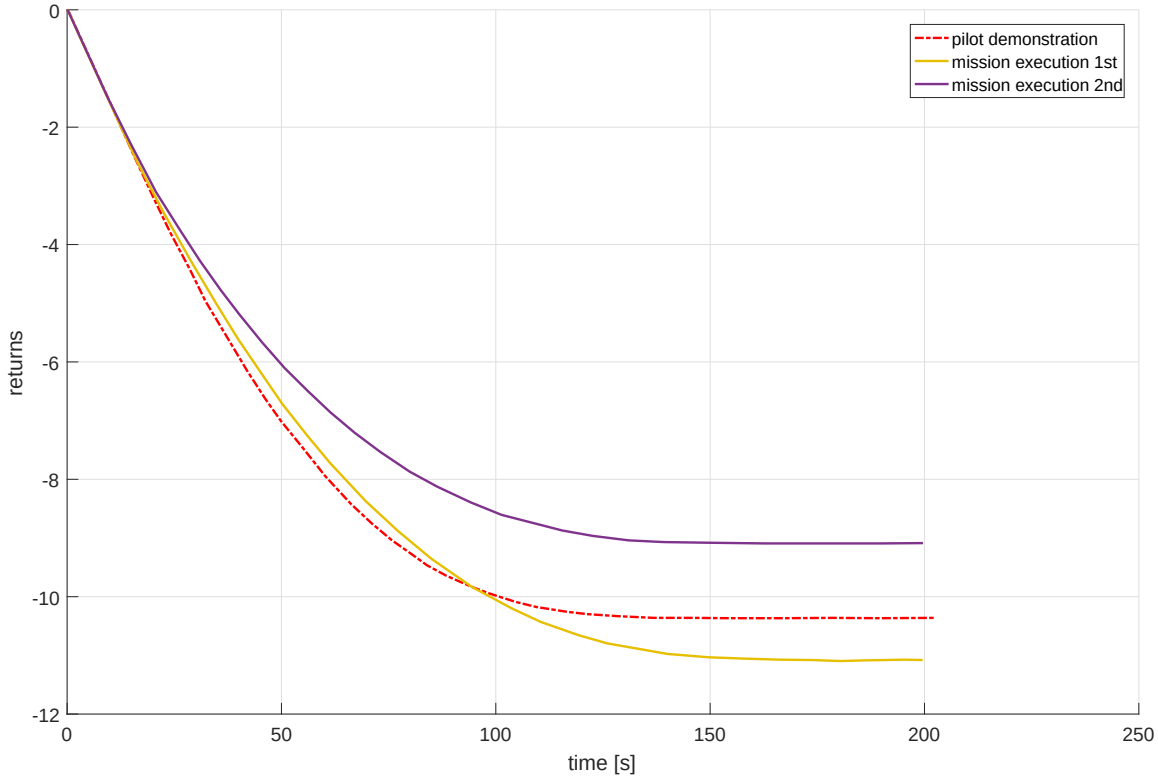


Figure 6.29. Returns achieved during the simulation cases in the mission execution stage for the vertical motion of the airship.

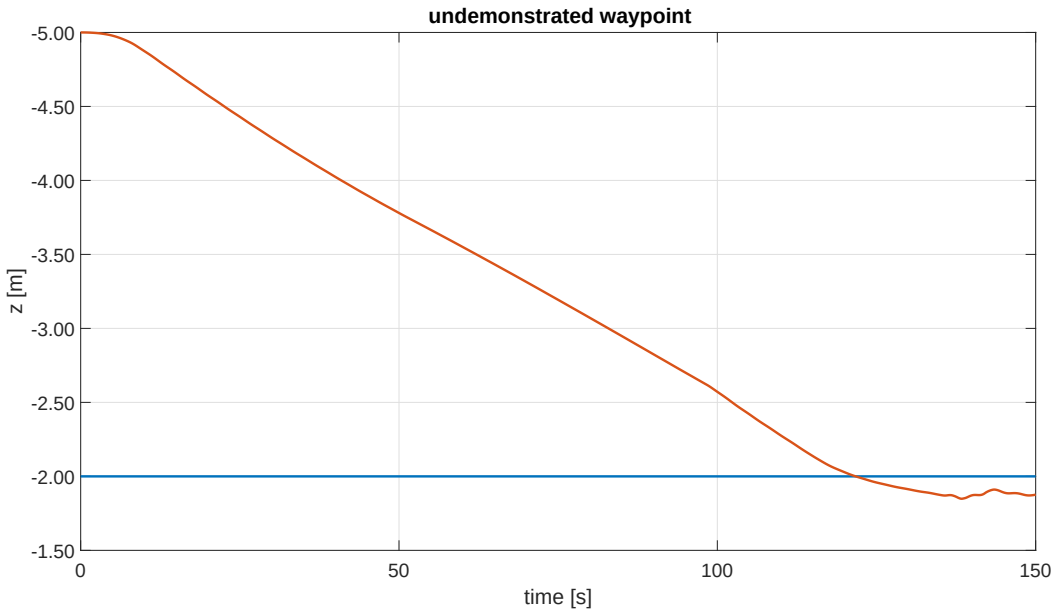


Figure 6.30. Mission execution of an undemonstrated altitude waypoint.

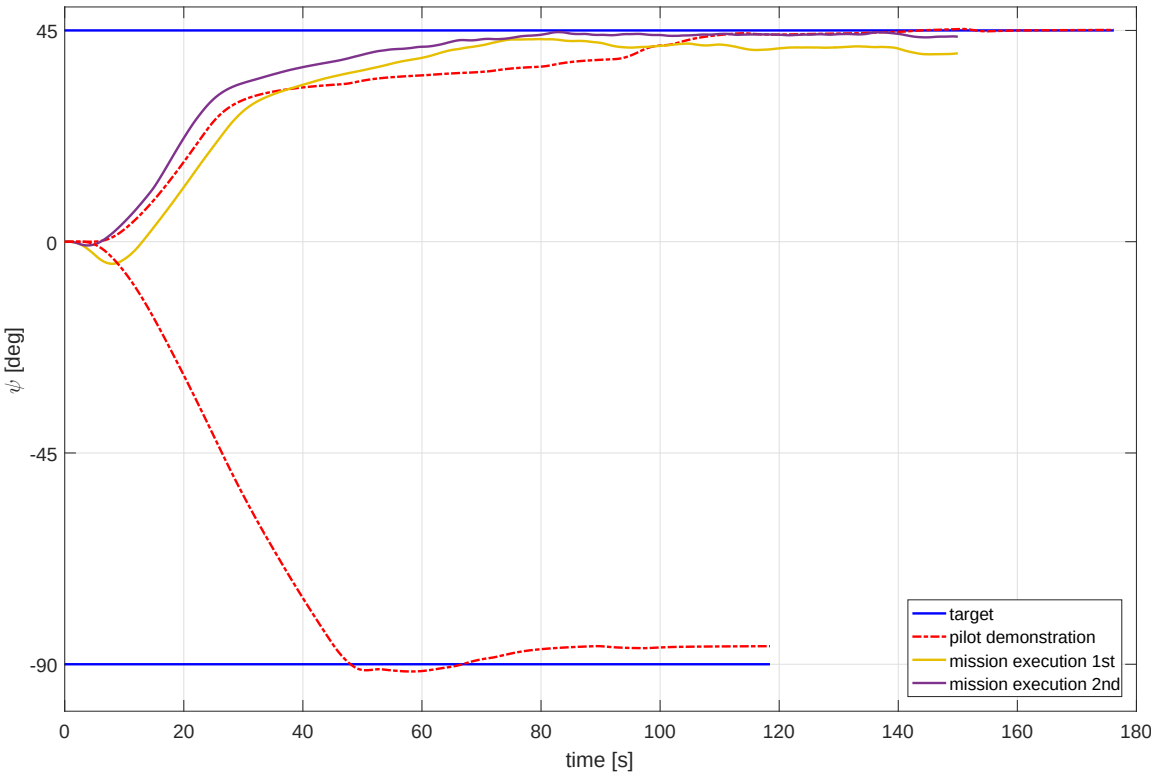


Figure 6.31. Mission execution stage for the turning motion of the airship.

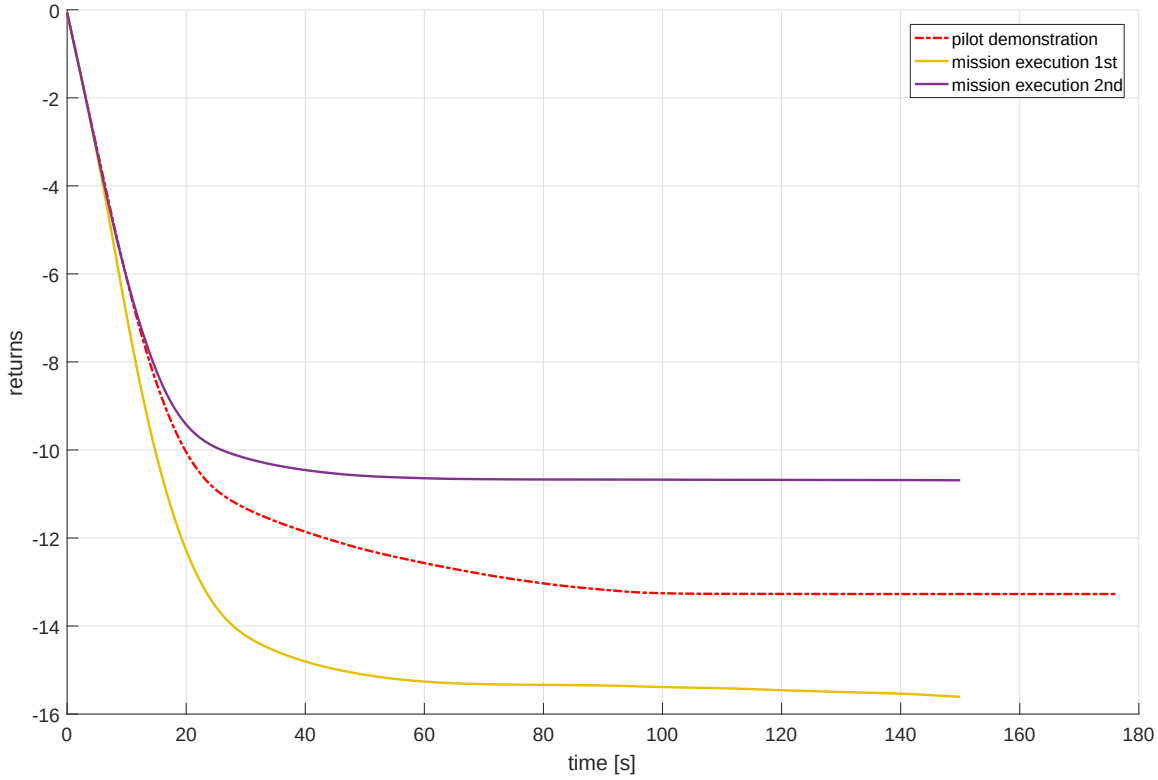


Figure 6.32. Returns achieved during the simulation cases in the mission execution stage for the turning motion of the airship.

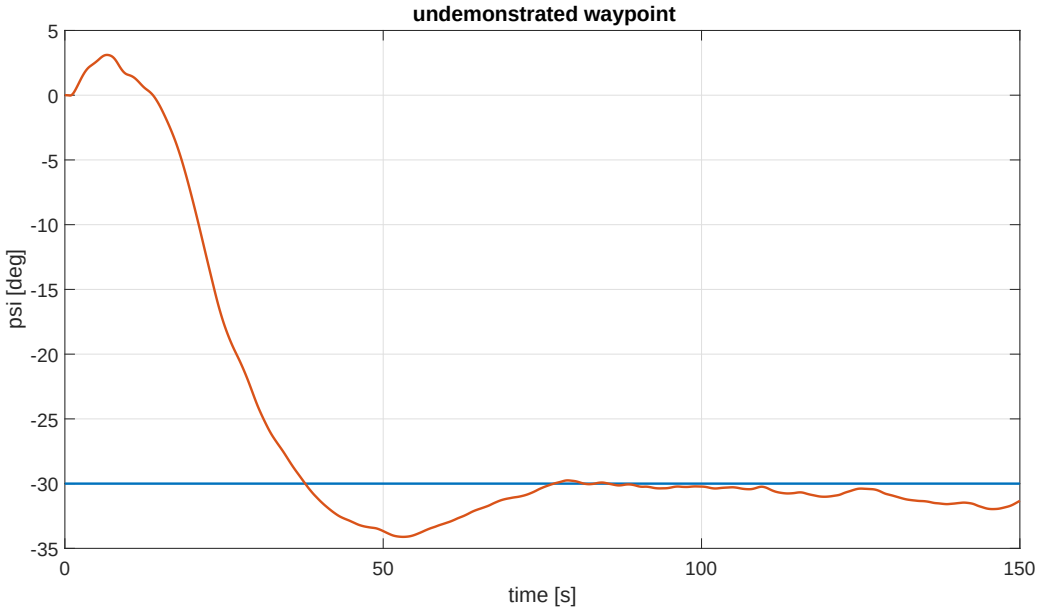


Figure 6.33. Mission execution of an undemonstrated turn waypoint.

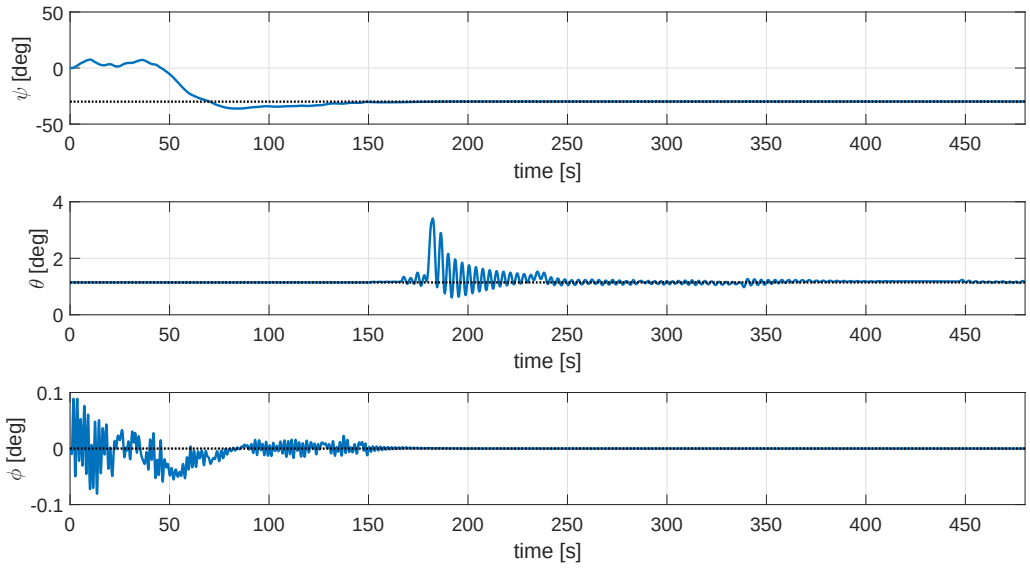


Figure 6.34. Orientation changes in mission execution of 3D mission.

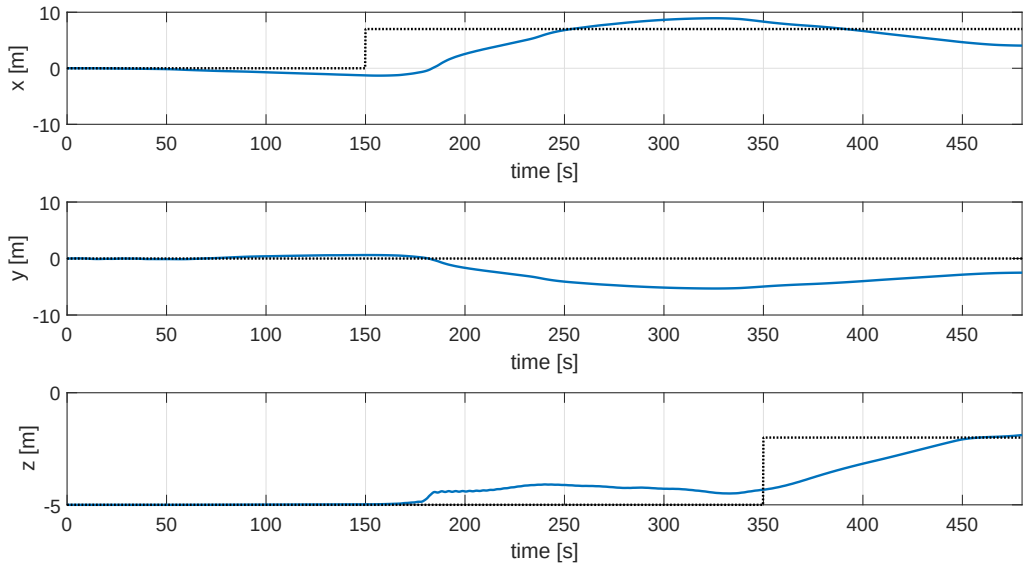


Figure 6.35. Position changes in mission execution of 3D mission.

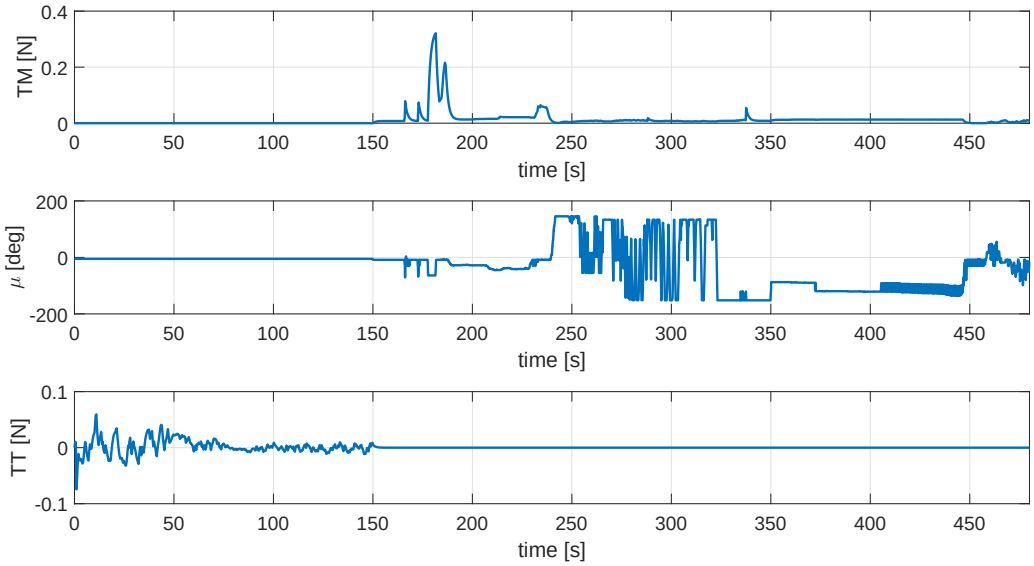


Figure 6.36. Control action changes in mission execution of 3D mission.

CHAPTER 7

Conclusion

Control system design for airship has its own unique challenges such as different mass-inertia characteristics than the conventional aircraft, nonlinear and underactuated dynamics, challenges in control effectors and high sensitivity to environmental conditions. This challenges can affect the way vehicle is controlled, thus make application of model-based control techniques infeasible. On the other hand, a skilled RC pilot can operate the vehicle without any difficulty. This makes Learning from demonstration (LfD) and Reinforcement learning (RL) methods suitable candidates for the control of airship.

In learning from demonstration, an expert performs demonstrations to show the correct set of actions to accomplish a task and a learner attempts to mimic the expert. However, in reinforcement learning correct behavior is not directly provided. The learner, also called agent, attempts to learn how to act by applying different actions in divided partitions of the environment, which are called states, and receiving rewards describing how well it has performed. Sum of the rewards are used to form state-value and action-value functions that can express which action is the most feasible to take given a state. A well known reinforcement learning method is called Q-Learning and attempts to learn action-values from the experiences of the learner. Q-learning uses discrete states and actions by default and stores state-action pairs and the corresponding action-values in a tabular form called Q-tables, which are used to retrieve both the action-values for a given state-action pair and the optimal action to take in a given state.

As the control of robotic vehicles is continuous in nature, implementing Q-Learning in continuous state-action space is an important task. However, this brings new challenges as the discrete Q-table structures no longer can be used. Thus, using function approximation techniques to calculate the action-value of a given state-action pair and optimization techniques to find the optimal action in a given state becomes necessary. In this study, locally weighted regression, which is a weighted version of the ordinary least squares algorithm, is used as the function approximator and weighted Euclidean distance is used as the distance function to perform the nearest neighbor calculations required for the regression algorithm.

In this study, basic theoretical background in LfD/RL (Learning from Demonstration and Reinforcement Learning) methods have been provided and a multi-state multi-action learning module is introduced. Learning module employs three stages, pilot demonstration, exploration and mission execution, consecutively. In the pilot demonstration stage, pilot is asked to perform several demonstrations. During these pilot demonstrations, the pilot actions and airship response are recorded and later used to create the dataset, i.e. Q-tables for Q-Learning. Exploration stage aims to enhance the created dataset by visiting unexplored regions of the state-action space through application of randomly selected actions. In mission execution stage, created training dataset is used to calculate the optimal actions and desired tasks are performed.

In order to show the capabilities of the framework, results of three simple example cases utilizing discrete states - discrete actions, continuous states - discrete actions, and continuous states - continuous actions are shown. For these cases, control of a UGV (Unmanned Ground Vehicle) in x-axis with a single action is selected as the task to be learned. Available action range of the UGV is discretized to create a set of discrete actions. A discrete equivalent of the continuous system is derived

by applying discrete actions to the continuous dynamics and recording the changes in the states. It has been shown that in discrete state - discrete state action case all three stages can successfully be applied and by increasing the amount of explorations, action-values can achieve convergence.

In the continuous state - discrete action case, continuous UGV states are quantized to create a set of discrete states. Using these states and the discrete actions two parts of experiments were performed. In the first part, the converged Q-table solution obtained in discrete state - discrete action example is directly used and it was shown that UGV can track set of position commands successfully. In the second part, three stages of the framework are applied and similar convergence results were obtained.

In the continuous state - continuous action case, three different update/execution times and two different function approximation parameters were tested. Results of the experiments showed that same convergence characteristics of discrete state cases can only be obtained when continuous states are used if the function approximator parameters are selected correctly.

Proposed learning framework is tested on both simulation and actual implementation of a UGV (Unmanned Ground Vehicle) for single state single action and then for multi state and multi action cases. It has been shown that UGV can perform mission executions of 1-D and 2-D navigation tasks successfully.

Low speed and indoor operation of the airship suggests several physical limitations of data collection by flight tests due to the lack of capable sensor systems. lt has been shown that these limitations can be avoided by a realistic indoor airship flight simulator. An existing airship model is improved by flight test and experimental data obtained from an RC indoor blimp using motion capture system and force-balance equipments.

Performance of LfD/RL methods are directly affected by the amount and the quality of the demonstrations. A skilled human pilot who operates the actual vehicle has been connected into a 3D animated visualization tool and pilot RC commands are transferred into the computer. With this approach, a skilled RC pilot is asked to fly the airship to accomplish specific tasks such as forward/backward, employs three up and down, and yawing motion maneuvers.

In order to implement the stages of the LfD/RL method using the airship, states and relevant control actions are divided into longitudinal and lateral modes and two learning units are used. Selecting accurate parameters for function approximator and the distance function for high dimensional state-action spaces is a hard task. By defining separate modes size of the state-action space can be reduced and distance function weightings can be assigned relatively easily.

In the mission execution phase, learned pilot demonstrations are used to perform commanded tasks. The basic capability is tested by demonstrating single orientation and position change commands and performing the same maneuver through the learning algorithm. These tests shows that depending on the quality of given demonstrations and fine tuning of learning parameters, the LfD/RL algorithm can successfully mimic the skills of human expert as well as perform tasks that are not directly demonstrated.

In addition, longitudinal and lateral units used in the waypoint cases are combined into a single system to perform a 3D mission. Results of this case was partially successful as the airship was successful in turning and keeping the constant heading during the rest of the mission. However, actions applied in altitude and forward motion maneuvers adversely affect each other and thus executing a forward motion command induces deviation in altitude, and the altitude change maneuver causing deviation in the x-position.

Several improvements can be performed to enhance the findings of this study. One example is combining the separated lateral and longitudinal modes and enhancing the LfD/RL method to deal with full states and actions. This requires use of different function approximation parameters in different local regions of the state-action space or adaptively changing the value of these parameters. A better distance function, which is more suitable for high dimensional state-actions, is crucial for the success of such implementation. Although weighted Euclidean distance has given satisfactory results with the lateral and longitudinal units, selection of the weights were mostly performed by trial and error. An analytical method that considers the statistical properties of the state-action dataset can be devised for the selection of these weightings.

Function approximation subroutine used in this study implements the locally weighted regression to calculate the output, action-value, of a given input, state-action pair, after several conditional checks. In these checks input is checked whether it is a repeating entry or it is bounded by the neighbors in the dataset. This calculations have long execution time and are run repetitively at every execution/update time. In addition, subroutine relies on too many constants such as weightings, distance margins, and default return values. A faster and more robust implementation of the subroutine and a more formulated way for selection of these constants would yield a better performance.

Locally weighted regression is an instance based method that keeps input pairs presented to the function approximator in a dataset and reuses them. Long runs of the algorithm creates too many entries in the dataset, which requires a memory management system to cope with the limited memory available. In this study, a distance based calculation is performed to decide whether a point is a new or a repeating entry in the dataset using constant distance margins. This method is efficient

for low dimensional state- action pairs, however, suffers the same problems that distance functions face in the high dimensional state-action spaces. Incorrect selection of these distance margins highly reduce the performance. In addition, the method implements repetitive sort operations, which become tedious when the dataset size becomes larger. Some of these problems can be addressed parallel to the improvements on the distance function. However, different criterions than the distance can be used for deciding whether an entry should be kept or discarded in the dataset, such as its contribution to the learning performance. For this purpose, statistical methods can be utilized.

Hardware implementation of this method on the actual airship is also an important improvement. Low speed and indoor operation of the airship results in physical limitations of the sensor systems and state measurements becomes inaccurate or unreliable. Indoor navigation aids such as motion capture systems, which were used in the flight tests in this study, provides accurate position and orientation feedback. However, covering a region large enough to suit whole envelope of airship maneuvers, brings extremely high costs and not feasible in engineering perspective. Simultaneous Localization and Mapping (SLAM) and visual odometry methods are some of the navigation solutions used for indoor operations of the robotic vehicles. These methods can be possible candidates to overcome some of the issues encountered in hardware implementation.

REFERENCES

[1] R. Tedrake, "Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for mit 6.832," *Working draft edition*, 2009.

[2] N. Aneke, "Control of underactuated mechanical systems," Ph.D. dissertation, Technische Universiteit Eindhoven, 2003.

[3] Y. Liu, "Autonomous blimp control with reinforcement learning," Master's thesis, University of Wollongong, 2009.

[4] A. Rottman, "Approaches to online reinforcement learning for miniature airships," Ph.D. dissertation, Albert Ludwigs University of Freiburg, 2012.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, May 2009. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2008.10.024

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, p. 0278364913495721, 2013.

[7] J. Blynel, "Reinforcement learning on real robots," Master's thesis, Aarhus Universitet, Datalogisk Afdeling, 2000.

[8] C. Strauss and F. Sahin, "Autonomous navigation based on a q-learning algorithm for a robot in a real environment," in *System of Systems Engineering, 2008. SoSE'08. IEEE International Conference on.* IEEE, 2008, pp. 1–5.

[9] L. M. Zamstein, "Koolio: Path-planning using reinforcement learning on a real robot in a real environment," Ph.D. dissertation, University of Florida, 2009.

[10] L. Khriji, F. Touati, K. Benhmed, and A. Al-Yahmedi, "Mobile robot navigation based on q-learning technique," *International Journal of Advanced Robotic Systems*, vol. 8, no. 1, pp. 45–51, 2011.

[11] D. Tamilselvi, S. M. Shalinie, and G. Nirmala, "Q learning for mobile robot navigation in indoor environment," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on.* IEEE, 2011, pp. 324–329.

[12] S. Pareigis, "Adaptive choice of grid and time in reinforcement learning," in *NIPS.* Citeseer, 1997.

[13] U. Krothapalli, T. Wagner, and M. Kumar, "Mobile robot navigation using variable grid size based reinforcement learning," *Infotech at Aerospace*, pp. 1–11, 2011.

[14] A. K. Lampton, "Discretization and approximation methods for reinforcement learning of highly reconfigurable systems," Ph.D. dissertation, Texas A&M University, 2009.

[15] W. T. Uther and M. M. Veloso, "Tree based discretization for continuous state space reinforcement learning," in *Aaai/iaai*, 1998, pp. 769–774.

[16] J. C. Santamaría, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive behavior*, vol. 6, no. 2, pp. 163–217, 1997.

[17] Y.-P. Hsu, W.-C. Jiang, and H.-Y. Lin, "A cmac-q-learning based dyna agent," in *SICE Annual Conference, 2008.* IEEE, 2008, pp. 2946–2950.

[18] J. Garcıa, I. López-Bueno, F. Fernández, and D. Borrajo, "A comparative study of discretization approaches for state space generalization in the keepaway soccer task," *Reinforcement Learning: Algorithms, Implementations and Aplications. Nova Science Publishers*, 2010.

[19] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *Automatic Control, IEEE Transactions on*, vol. 42, no. 5, pp. 674–690, 1997.

[20] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *ICML*. Citeseer, 2001, pp. 417–424.

[21] J. Peng, "Efficient dynamic programming-based learning for control," Ph.D. dissertation, Northeastern University, 1993.

[22] M. Irodova and R. H. Sloan, "Reinforcement learning and function approximation." in *FLAIRS Conference*, 2005, pp. 455–460.

[23] C. Gaskett, "Q-learning for robot control," Ph.D. dissertation, The Australian National University, 2002.

[24] S. Buck, M. Beetz, and T. Schmitt, "Approximating the value function for continuous space reinforcement learning in robot control," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1. IEEE, 2002, pp. 1062–1067.

[25] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*. Springer, 2005, pp. 317–328.

[26] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007*. IEEE, 2007, pp. 645–650.

[27] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, 2009.

[28] Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: The gaussian process approach to temporal difference learning," in *ICML*, vol. 20, no. 1, 2003, p. 154.

[29] J. Zubizarreta-Rodriguez and F. Ramos, "Multi-task learning of system dynamics with maximum information gain," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 5709–5715.

[30] J. Ko, D. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *Robotics and Automation, 2007 IEEE International Conference on*, April 2007, pp. 742–747.

[31] K. Gräve, J. Stückler, and S. Behnke, "Learning motion skills from expert demonstrations and own experience using gaussian process regression," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[32] S. Schaal *et al.*, "Learning from demonstration," *Advances in neural information processing systems*, pp. 1040–1046, 1997.

[33] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces." Morgan Kaufmann, 2000, pp. 903–910.

[34] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 2369–2374.

[35] D. C. Bentivegna, "Learning from observation using primitives," Ph.D. dissertation, Citeseer, 2004.

[36] C. Szepesvári and W. D. Smart, "Interpolation-based q-learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 100.

[37] W. D. Smart, "Making reinforcement learning work on real robots," Ph.D. dissertation, Brown University, 2002.

[38] D. Nguyen-Tuong and J. Peters, "Incremental online sparsification for model learning in real-time robot control," *Neurocomputing*, vol. 74, no. 11, pp. 1859–1867, 2011.

[39] X. Chen, Y. Gao, and R. Wang, "Online selective kernel-based temporal difference learning," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 24, no. 12, pp. 1944–1956, 2013.

[40] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," in *Advances in neural information processing systems*, 2009, pp. 905–912.

[41] M. Seeger, C. K. I. Williams, and N. D. Lawrence, "Fast forward selection to speed up sparse gaussian process regression," in *IN WORKSHOP ON AI AND STATISTICS 9*, 2003.

[42] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, 2005, pp. 1257–1264.

[43] R. Dillmann, M. Kaiser, and A. Ude, "Acquisition of elementary robot skills from human demonstration," in *International Symposium on Intelligent Robotics Systems*. Citeseer, 1995, pp. 185–192.

[44] P. K. Pook and D. H. Ballard, "Recognizing teleoperated manipulations," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 578–585.

[45] J. Chen and A. Zelinsky, "Programing by demonstration: Coping with suboptimal teaching actions," *The International Journal of Robotics Research*, vol. 22, no. 5, pp. 299–319, 2003.

[46] J. Aleotti and S. Caselli, "Robust trajectory learning and approximation for robot programming by demonstration," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 409–413, 2006.

[47] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[48] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, 2003, pp. 1–20.

[49] M. Ogino, Y. Katoh, M. Aono, M. Asada, and K. Hosoda, "Reinforcement learning of humanoid rhythmic walking parameters based on visual information," *Advanced Robotics*, vol. 18, no. 7, pp. 677–697, 2004.

[50] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 3030–3035.

[51] S. Degallier, C. P. Santos, L. Righetti, and A. Ijspeert, "Movement generation using dynamical systems: a humanoid robot performing a drumming task," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, 2006, pp. 512–517.

[52] J. Lieberman and C. Breazeal, "Improvements on action parsing and action interpolation for learning through demonstration," in *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, vol. 1. IEEE, 2004, pp. 342–365.

[53] M. J. Mataric, "Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics," in *Imitation in animals and artifacts*. Citeseer, 2000.

[54] B. Nemec, M. Tamošiūnaitė, F. Wörgötter, and A. Ude, "Task adaptation through exploration and action sequencing," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 610–616.

[55] S. Chernova and M. Veloso, "Learning equivalent action choices from demonstration," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1216–1221.

[56] I. Vincent and Q. Sun, "A combined reactive and reinforcement learning controller for an autonomous tracked vehicle," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 599–608, 2012.

[57] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[58] D. Silver, J. A. Bagnell, and A. Stentz, "Learning autonomous driving styles and maneuvers from expert demonstration," in *Experimental Robotics*. Springer, 2013, pp. 371–386.

[59] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seattle, USA*, vol. 134, 2015.

[60] A. El-Fakdi, M. Carreras, N. Palomeras, and P. Ridao, "Autonomous underwater vehicle control using reinforcement learning policy search methods," vol. 2. IEEE, 2005, pp. 793–798.

[61] H. Kawano and T. Ura, "Motion planning algorithm for nonholonomic autonomous underwater vehicle in disturbance using reinforcement learning and teaching method," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 4. IEEE, 2002, pp. 4032–4038.

[62] H. Kawano, "Method for applying reinforcement learning to motion planning and control of under-actuated underwater vehicle in unknown non-uniform sea flow." in *IROS*, 2005, pp. 996–1002.

[63] R. Zhang, P. Tang, Y. Su, X. Li, G. Yang, and C. Shi, "An adaptive obstacle avoidance algorithm for unmanned surface vehicle in complicated marine environments," *Automatica Sinica, IEEE/CAA Journal of*, vol. 1, no. 4, pp. 385–396, 2014.

[64] J. F. Montgomery and G. A. Bekey, "Learning helicopter control through teaching by showing," in *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, vol. 4. IEEE, 1998, pp. 3647–3652.

[65] J. A. Bagnell and J. G. S. Hneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 1615–1620.

[66] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, 2010.

[67] A. Coates, P. Abbeel, and A. Y. Ng, "Autonomous helicopter flight using reinforcement learning," in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 53–61.

[68] P. A. M. Mediano, "Data-efficient reinforcement learning for autonomous helicopters," 2014.

[69] K. Motoyama, H. Kawamura, M. Yamamoto, and A. Ohuchi, "Development of autonomous blimp robot with intelligent control," in *Entertainment Computing*, ser. IFIP The International Federation for Information Processing, R. Nakatsu and J. Hoshino, Eds. Springer US, 2003, vol. 112, pp. 191–198. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-35660-0_23

[70] P. Dallaire, C. Besse, S. Ross, and B. Chaib-Draa, "Bayesian reinforcement learning in continuous pomdps with gaussian processes," in *Intelligent Robots and*

Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, 2009, pp. 2604–2609.

[71] A. Klopf *et al.*, "Reinforcement learning with high-dimensional, continuous actions," DTIC Document, Tech. Rep., 1993.

[72] H. van Hasselt, "Reinforcement learning using continuous actions," 2005.

[73] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[74] T. J. Walsh, D. K. Hewlett, and C. T. Morrison, "Blending autonomous exploration and apprenticeship learning," in *Advances in Neural Information Processing Systems*, 2011, pp. 2258–2266.

[75] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97, 1997, pp. 12–20.

[76] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Advances in Neural Information Processing Systems*, 2007, pp. 769–776.

[77] F. J. G. Polo and F. F. Rebollo, "Safe reinforcement learning in high-risk tasks through policy improvement," in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on.* IEEE, 2011, pp. 76–83.

[78] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning." in *ESANN*, 2008, pp. 143–148.

[79] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *Robotics and Automation, 2007 IEEE International Conference on.* IEEE, 2007, pp. 2483–2488.

[80] J. R. N. Forbes, "Reinforcement learning for autonomous vehicles," Ph.D. dissertation, UNIVERSITY of CALIFORNIA at BERKELEY, 2002.

[81] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[82] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge England, 1989.

[83] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1-5, pp. 11–73, Feb. 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1006559212014

[84] C. C. Aggarwal, "Re-designing distance functions and distance-based applications for high dimensional data," *ACM SIGMOD Record*, vol. 30, no. 1, pp. 13–18, 2001.

[85] B. D. Nichols, "Continuous action-space reinforcement learning methods applied to the minimum-time swing-up of the acrobot," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2015, pp. 2084–2089.

[86] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[87] J. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," *Advances in neural information processing systems*, pp. 369–376, 1995.

[88] D. C. Hoaglin and R. E. Welsch, "The hat matrix in regression and anova," *The American Statistician*, vol. 32, no. 1, pp. 17–22, 1978.

[89] P. Desai, "Implementation of autonomous navigation and obstacle avoidance on an unmanned ground vehicle," Master's thesis, The University of Texas at Arlington, Arlington, TX, December 2009.

[90] H. E. Sevil, P. Desai, A. Dogan, and B. Huff, "Modeling of an unmanned ground vehicle for autonomous navigation and obstacle avoidance simulations," in *Proc.*

of DSCC 2012, ASME Dynamic Systems and Control Conference, Ft. Lauderdale, USA, 17-19 October 2012, pp. 972–977, dSCC2012-8867.

[91] J. Waishek, A. Dogan, and Y. Bestaoui, "Investigation into the time varying mass effect on airship dynamics response," in *Proceedings of Aerospace Sciences Meeting Including th New Horizons Forum and Aerospace Exposition, AIAA*, 2009.

[92] ——, "Comprehensive characterization of airship response to wind and time varying mass," *AIAA Atmospheric Flight Mechanics Conference*, pp. 1–22, August 2010.

[93] A. Bonnet, "Identification des coefficients aerodynamiques du dirigeable as500 du laas," SUPAERO, Tech. Rep., 2003, etude Hydro-Aerodynamique.

[94] M. M. Munk, "The aerodynamic forces on airship hulls," National Advisory Committee for Aeronautics, Tech. Rep., 1924.

[95] *AFA3 Three-Component Balance*, TecQuipment Ltd., http://www.tecquipment.com/Datasheets/AFA3_1015.pdf.

[96] *SCB-68 User Manual for Advanced Functions*, National Instruments, http://www.ni.com/pdf/manuals/372551a.pdf.

[97] "Arduino pro mini," https://www.arduino.cc/en/Main/ArduinoBoardProMini.

BIOGRAPHICAL STATEMENT

Onur Daskiran was born in Istanbul, Turkey in 1987. He received his B.Sc. degrees in Avionics and Electrical Engineering both from Anadolu University, Turkey, in 2008 and M.Sc degree in Aeronautical Engineering from Istanbul Technical University in 2011. Following that, he started the Ph.D. program in Aerospace Engineering at the University of Texas at Arlington (UTA) in Autonomous Vehicles Laboratory. He was also appointed as a Graduate Teaching Assistant (UTA) from 2011 to 2016 at UTA during this period. His research interests are navigation, control, and path planning of mobile robots and unmanned vehicles, machine learning, and embedded system applications. He is an AIAA, AUVSI and a past IEEE member.