Sim-to-Real Reinforcement Learning Framework for Autonomous Aerial Leaf Sampling

Ashraful Islam
University of Nebraska - Lincoln, mislam@huskers.unl.edu

SIM-TO-REAL REINFORCEMENT LEARNING FRAMEWORK FOR AUTONOMOUS AERIAL LEAF SAMPLING

by

Ashraful Islam

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Carrick Detweiler

Lincoln, Nebraska

May, 2023

SIM-TO-REAL REINFORCEMENT LEARNING FRAMEWORK FOR AUTONOMOUS AERIAL LEAF SAMPLING

Ashraful Islam, M.S.

University of Nebraska, 2023

Adviser: Carrick Detweiler

Using unmanned aerial systems (UAS) for leaf sampling is contributing to a better understanding of the influence of climate change on plant species, and the dynamics of forest ecology by studying hard-to-reach tree canopies. Currently, multiple skilled operators are required for UAS maneuvering and using the leaf sampling tool. This often limits sampling to only the canopy top or periphery. Sim-to-real reinforcement learning (RL) can be leveraged to tackle challenges in the autonomous operation of aerial leaf sampling in the changing environment of a tree canopy. However, transferring an RL controller that is learned in simulation to real UAS applications is challenging due to the risk of crashes. UAS crashes pose safety risks to the operator and its surroundings which often leads to expensive UAS repairs.

In this thesis, we present a Sim-to-Real Transfer framework using a computer numerical control (CNC) platform as a safer, and more robust proxy, before using the controller on a UAS. In addition, our framework provides an end-to-end complete pipeline to learn, and test, any deep RL controller for UAS or any three-axis robot for various control tasks. Our framework facilitates bi-directional iterative improvements to the simulation environment and real robot, by allowing instant deployment of the simulation learned controller to the real robot for performance verification and issue identification.

Our results show that we can perform a zero-shot transfer of the RL agent, which is trained in simulation, to real CNC. The accuracy and precision do not meet the requirement for complex leaf sampling tasks yet. However, the RL agent trained for a static target following still follows or attempts to follow more dynamic and changing targets with predictable performance. This works lays the foundation by setting up the initial validation requirements for the leaf sampling tasks and identifies potential areas for improvement. Further tuning of the system and experimentation of the RL agent type would pave the way to autonomous aerial leaf sampling.

DEDICATION

I dedicate this thesis to my daughter, Camila Islam, who makes every day brighter than the last one.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Carrick Detweiler, for his continuous support, encouragement, and patience throughout my time in the NIMBUS lab. His encouragement gave me the courage to explore my interest in computer science courses and discover how much I enjoyed them. Which eventually led me to apply for graduate school in computer science right after finishing my graduate degree in mechanical engineering.

I would like to thank my wife, Rubi Quiñones, for her invaluable feedback on the thesis, and for always being there for me and supporting me throughout my graduate career.

I would also like to thank my fellow NIMBUSers who have always been really good friends. Thank you for maintaining such a happy and helpful culture in the lab. There was always someone to help with technical knowledge when I am stuck with something or provide a helping hand whenever I had outdoor experiments.

I would also like to thank Nathan and Jayden for their help with the electro-mechanical components of this project.

GRANT INFORMATION

The work was partially supported by the National science foundation (NSF) and the National Aeronautics and Space Administration (NASA). Grant numbers are: NSF IIS-1925368, NSF IIS-1925052, NASA ULI-80NSSC20M0162. The views or findings presented in this thesis are those of the authors and not of the funding agencies.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Reinforcement Learning (RL) can be utilized to control complex robotic systems [1]. RL agents are capable of learning an optimal controller by interacting with the environment and receiving rewards based on actions taken. This is similar to a trial-and-error approach, where the controller improves over time as more data is gathered. We can combine many types of sensors and image data as inputs for the RL agent which is not commonly used in traditional control algorithms.

Plant scientists study leaves to understand the genotype and environment interaction in individual plants and as a collective in a forest. This is important to better understand the negative effects global warming has caused in a range of environments. Traditional leaf sampling methods require human experts that collect samples with a pole pruner, ladder, slingshot, shotgun, rope launcher or they try to climb trees. Most of these methods either have a low reach, low access (only inner stronger branches), or have the potential risk of injury to the operator or the wildlife.

Aerial leaf sampling is a relatively new sampling method that uses a UAS with some leaf or branch cutting mechanism [2]. This requires collaborative effort in areas of mechanical, electrical, computer science, and plant science. Aerial leaf sampling methods can be divided into five broadly defined processes [3]: (1) the UAS flies from the base station to the desired tree canopy; (2) the UAS operator identifies the desired target branch according to sampling requirements; (3) the UAS operator lowers the leaf sampler to the target branch; (4) the leaf sampler collects the leaves; and (5) the UAS retrieves the sampler, and flies back to the base station.

The overall goal of this thesis is to automate the entire aerial leaf sampling process to reduce human intervention and error. We first identify some of the challenges that we must overcome for the future development of automated leaf sampling systems including the RL agent. The challenges are as follows:

1. Since our target environment is an unstructured tree canopy with lots of uncertainties that can cause the UAS to crash, learning directly from real experience is not feasible. However, we can train the RL agent in a simulation environment using synthetic data, and then transfer the learned agent to a real robot working in the real environment. This method is known as sim-to-real RL and shows great success in many different robotics applications [4]. There would still be a challenge to address the 'reality gap' that may occur between the simulation and the real-world system [5].

2. The lack of a precise UAS motion capture system outdoors makes it difficult to perform precise maneuvers. Performing experimental control methods for the UAS over a tree can result in setbacks due to catastrophic crashes when there is a system failure or wrong control inputs are given. Furthermore, outdoor UAS experimentations can not be performed on rainy, snowy, or windy days. This imposes seasonal limitations on developing the different aspects of the leaf sampling process.

To overcome these challenges we developed a sim-to-real framework using reinforcement learning. We limit our scope to indoor implementation of the leaf sampler with the help of the Vicon motion capture system. We also use a computer numeric control (CNC) system mounted on the ceiling as an intermediate platform for autonomous development instead of the UAS. Although we reduce the dimensionality of the problem from a six-degree-of-freedom of the UAS to a three-degree-of-freedom of the CNC, the framework offers interoperability between the two systems by using similar observations and control inputs. This mitigates challenges of flight time and crashes of risk when directly using UAS and offers non-stop power tethered operation. Additionally, it allows us progressively implement different algorithms and techniques to handle various leaf sampling tasks while adding sensors and electronics as required.

This thesis establishes the pipeline for autonomous aerial leaf sampling and presents the initial steps in the sim-to-real leaf sampling processes.

1.1 Contributions

In this thesis, we present an indoor sim-to-real platform that would facilitate fast learning and deployment of reinforcement learning algorithms in the simulation environment and real-world deployment. We were able to do a zero-shot transfer of the RL agent trained in the simulated CNC to the real CNC. The agent performed robustly for a more dynamic target than what was trained in the simulation. Our contributions are:

- Development of a sim-to-real framework for training and deploying an RL agent for UAS aerial leaf sampling;
- Design of a hardware and software interface that allows fast deployment of simulation-trained agents to the CNC setup (real world); and
- Evaluation of the sim-to-real framework of a learned RL agent in simulation and in CNC.

Our results show that the RL agent performance does not fully meet the expectation in terms of accuracy and precision yet. However, the predictability of the results and identifiable limitations of the system that can be addressed in future iterations show that we are on the right path.

The advancement in sim-to-real leaf sampling would benefit researchers who might also use similar payloads for wildlife monitoring, critter, and insect studies, canopy microclimate study, enhanced imaging, depth map of inner forest layers, collection of genetic samples for plant identification, etc. as the framework can be adapted to any other similar UAS application.

The rest of the thesis is organized as follows: We present a literature review on reinforcement learning and leaf sampling to provide context to design decisions made throughout the paper. We then discuss our sim-to-real framework methods, simulation and real-world setup, results and discussion, and finish with a conclusion and future work.

Chapter 2

Related Work

Reinforcement learning has been shown to be very successful in solving many complex tasks in recent years. Following the success of AlphaGo defeating the world champion at Go [6], recently, RL trained algorithm defeated a team of professional players in the competitive e-sports game of DOTA 2 [7]. This is particularly impressive since the game of DOTA2 presents a lot of challenges to traditional RL: observation and action space are large, with a long horizon, and a partially observable environment. In recent years, there has been significant development in using the RL controller learned in simulation for real-world robotic applications such as object manipulation [8, 9, 10, 11, 12], robotic locomotion [13, 14], and Unmanned aerial systems (UAS) flights [15, 16, 17, 18].

In this chapter, we discuss the relevant background of reinforcement learning in the context of this thesis. We then discuss the state-of-the-art in sim-to-real reinforcement learning and UAS aerial leaf sampling.

2.1 Reinforcement Learning Background

In this section, we discuss the reinforcement learning (RL) background and discussion relating to what RL agent we used in this thesis. Although our framework can be used to test various different RL agents available, we used a Twin-delayed deep deterministic policy gradient agent, also known as TD3 [19] as discussed next.

A reinforcement learning (RL) agent receives a set of observations (S) from the environment and executes actions (A). The environment transitions to a new state S' and the agent receives a reward (R) based on the actions taken. The RL agent's policy (π) is updated from this experience such that the expected cumulative long-term reward is maximized [20]. In traditional RL, this can be formalized as Markov Decision Process (MDP) where the transition probability from (S) to (S') based on action (A) follows the probability distribution $P(S'|S, A)$ [20].

Figure 2.1 shows a schematic of an RL agent and corresponding mapping to the classical control system. The environment, agent policy, action, observation, and RL algorithm in reinforcement learning are equivalent to the plant, controller, control output, error signal, and controller tuning in classical control respectively. Optimal control would then be equivalent to minimizing the loss function in RL.

One major difference is that while the classical control uses a linearized model of the plant to derive the controller, reinforcement learning can either be model-based or model-free. Using deep learning network structure in the agent policy, it can learn a model-free, non-linear representation of the environment or plant.

2.1.1 Value-based, Policy-based, and Actor-Critic Agents

In RL, the actor and critic represent the next best action given some observations and the merit of such action in maximizing the long-term cumulative reward. The critic can be represented by value function ($V(s)$) if only using observations as input, Q-value ($Q(S, A)$) function if taking both observation and the action as input. On the other hand, actors ($\pi(S)$) can be represented by a deterministic policy, or a stochastic policy where the action is deterministic or stochastic as the name implies.

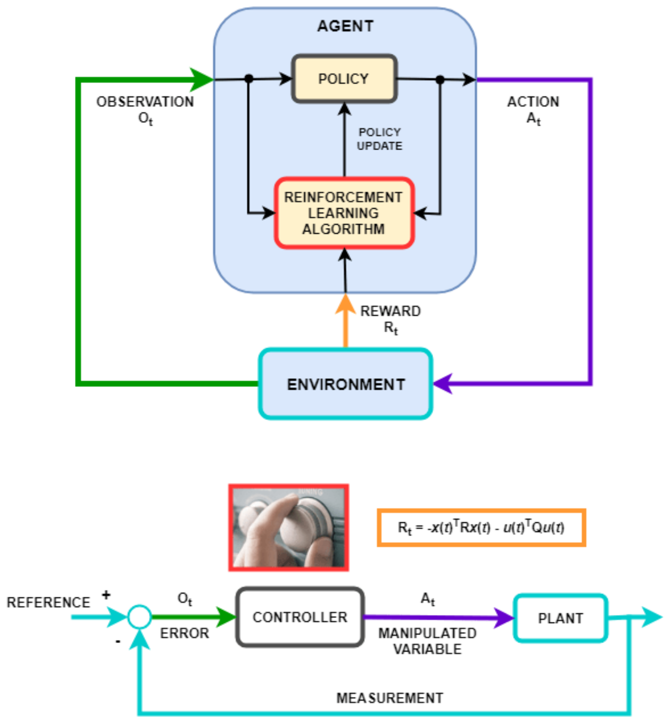


Figure 2.1: Reinforcement learning agent concept schematic and the mapping that represents equivalent control system. Image from [21]

Deterministic policy actor will add random noise to promote exploration.

When the actor and the critic are represented using deep neural networks, the algorithm would then be called deep RL. Using the deep neural network is especially useful because the discrete or continuous observation or action space is very large. Many RL algorithms have emerged that use different variations of the actor-critic network. Table 2.1 summarizes different RL algorithms, action space, and actor-critic representations used with each of them.

Agents that use critics only for selecting actions are called value-based or indirect policy representation agents. Agent of these type include: Q-Learning (Q) [22], Deep Q-Network (DQN) [23], State-Action-Reward-State-Action (SARSA) [24] agents.

Agents that only use actors are called policy-based or direct policy representation agents, e.g., Policy Gradient (PG) [25] agents.

Table 2.1: Summary of popular RL algorithms used in Literature. Observation space is omitted as it can be discrete or continuous in all listed algorithms

RL Algorithm	Agent Type	Action space	Critic Representation	Actor representation
Q-Learning (Q)	Value-Based	Discrete	Q-value function	×
Deep Q-Network (DQN)	Value-Based	Discrete	Q-value function	×
State-Action-Reward-State-Action (SARSA)	Value-Based	Discrete	Q-value function	×
Policy Gradient (PG)	Policy-Based	Either	value-function	stochastic
Actor-Critic (AC)	Actor-Critic	Either	value function	stochastic
Proximal Policy Optimization (PPO)	Actor-Critic	Either	value function	stochastic
Trust Region Policy Optimization (TRPO)	Actor-Critic	Either	value function	Stochastic
Deep Deterministic Policy Gradient (DDPG)	Actor-Critic	Continuous	Q-value function	Deterministic
Twin-Delayed DDPG (TD3)	Actor-Critic	Continuous	Q-value function	Deterministic
Soft Actor-Critic (SAC)	Actor-Critic	Continuous	Q-value function	stochastic

Agents that use both actor and critic representations are called actor-critic agents. The actor learns the best actions based on the feedback from the critic instead of the reward. And the critic learns from the reward to provide feedback to the actor. Examples of such agents are: Actor-Critic (AC) [26], Proximal Policy Optimization (PPO) [27], Trust Region Policy Optimization (TRPO) [28], Deep Deterministic Policy Gradient Agents (DDPG) [29], Twin-Delayed Deep Deterministic Policy Gradient (TD3) [19], Soft Actor-Critic agents (SAC) [30].

For continuous observation and action space, actor-critic agents are the most ideal choice since indirect and direct policy representation actors are computationally expensive, and noisy respectively in such scenarios.

2.1.2 Deep Deterministic Policy Gradient (DDPG)

DDPG algorithm is a model-free, online, off-policy RL learning algorithm [31]. A DDPG agent optimizes a policy that would return the most expected long-term reward when starting from any state. The network architecture of this algorithm is based on deterministic policy gradient (DPG) [29] algorithm's actor-critic method. The actor determines the best possible action at any given state using a policy that is learned based on the critic's feedback instead of directly using the reward. The critic is learned using the Bellman equation similar to Q-learning [31].

The most important feature of the DDPG algorithm is that it can be used in environments with continuous action space and continuous or discrete action space. This feature of the DDPG makes it suitable for applications using continuous control such as controlling a UAS.

Critic $Q(S, A|\phi)$ in DDPG uses a Q-value function for representing the total expected long-term reward. This function maps the observation-action pair to a scalar value that represents the total expected long-term rewards when the agent starts from a given observation (S) and executes the given action (A). Here ϕ represents the parameters of the network of the Q-value.

Actor $\pi(S|\theta)$ in DDPG uses a deterministic policy actor function approximator with a continuous action space. The actor takes observations (S) and outputs the action that maximizes the expected cumulative long-term reward as determined by the critic. Here θ represents the parameters of the network of the actor.

At each time step, for a given observation S, the actor executes an action $A = \pi(S) + N$, where N is a stochastic noise from a chosen noise model. The agent then observes the reward R and the next observation (S'). The experience (S, A, R, S') is stored in a circular buffer during training. The replay buffer enables updating the critic's Q-value and actor's policy in an off-policy method by randomly sampling a mini-batch of experiences from this buffer [32]. To improve the stability of optimization, a copy of the actor-critic network- known as the target actor-critic network is used which is updated less frequently from the original actor-critic with a smoothing factor τ.

The value function target (y_i) is the sum of the experience reward and the discounted future reward. The agent computes the cumulative reward recursively by first passing the next observation (S') to the target actor and then passing the next action and observation to the target critic. The critic parameters are updated by minimizing the following loss (L) equation across a mini-batch of M experiences,

$$y_i = R_i + \gamma Q_t(S'_i, \pi_t(S'_i|\theta_t)|\phi_t) \tag{2.1}$$

$$L = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(S_i, A_i|\phi_i)^2 \tag{2.2}$$

The actor parameters are updated by a gradient descent update that takes both the gradient output of the actor and the gradient output of the critic into account by the following formula.

$$\nabla_\theta J \approxeq \frac{1}{M}\sum_{i=1}^{M}\nabla_A Q(S_i, \pi(S_i|\theta)|\phi)\nabla_\theta \pi(S_i|\theta) \tag{2.3}$$

2.1.3 Twin-delayed Deep Deterministic Policy Gradient (TD3)

One of the limitations of the DDPG is that function approximation error can lead to overestimation of the Q-value function and result in a sub-optimal policy. To address this limitation, an extension of the DDPG algorithm is proposed as TD3 algorithm [19]. The main difference between TD3 with DDPG is: 1) it uses two Q-value functions instead of one, and updates the policy using the minimum value of the two; 2) updates the actor's policy and targets less frequently than the critic's Q functions; 3) during policy update, it adds noise to the target action to reduce the chances of the policy exploiting actions with high Q-value.

The value function target is slightly different from DDPG where the noise (ϵ) is added to the computed action and clipped based on the defined limit before passing the actions to the two target critics to find out the minimum value of the two.

$$y_i = R_i + \gamma * \min_{k=1,2} Q_{tk}(S_i', clip(\pi_t(S_i'|\theta_t) + \epsilon)|\phi_{tk}) \tag{2.4}$$

The loss function and policy update function is similar to DDPG with the difference in policy and target update frequency.

TD3 outperforms DDPG in similar tasks in learning time and average return. While an improvement from each TD3 modifications results are found to be different based on tasks, delayed policy update is found to improve performance and reduce training time in general [19].

2.2 Sim-to-Real Reinforcement Learning

RL agents trained in simulations are being transferred to real robots for control with success in many different fields, specially in robotic tasks involving object manipulation [8, 9, 10, 11, 12], robotic locomotion [13, 14], and UAS flights [15, 16, 17, 18]. Figure 2.2 shows some of the sim-to-real examples with a rendering of the simulation environment and corresponding real-world setup.

Some popular approaches for sim-to-real transfer used by researchers are system identification, domain randomization, domain adaptation, imitation learning, meta-learning, and knowledge distillation [5]. More often than not, the successful transfer comes from using a combination of these techniques instead of one singular approach. If the agent is successfully transferred to a real robot without further modification, it is termed as 'zero-shot transfer'. This is the most straightforward transfer method but requires careful design of the rest of the framework. We review the three main approaches in the following text: system identification, domain adaptation, and domain randomization. We also discuss the available software for the simulation environment

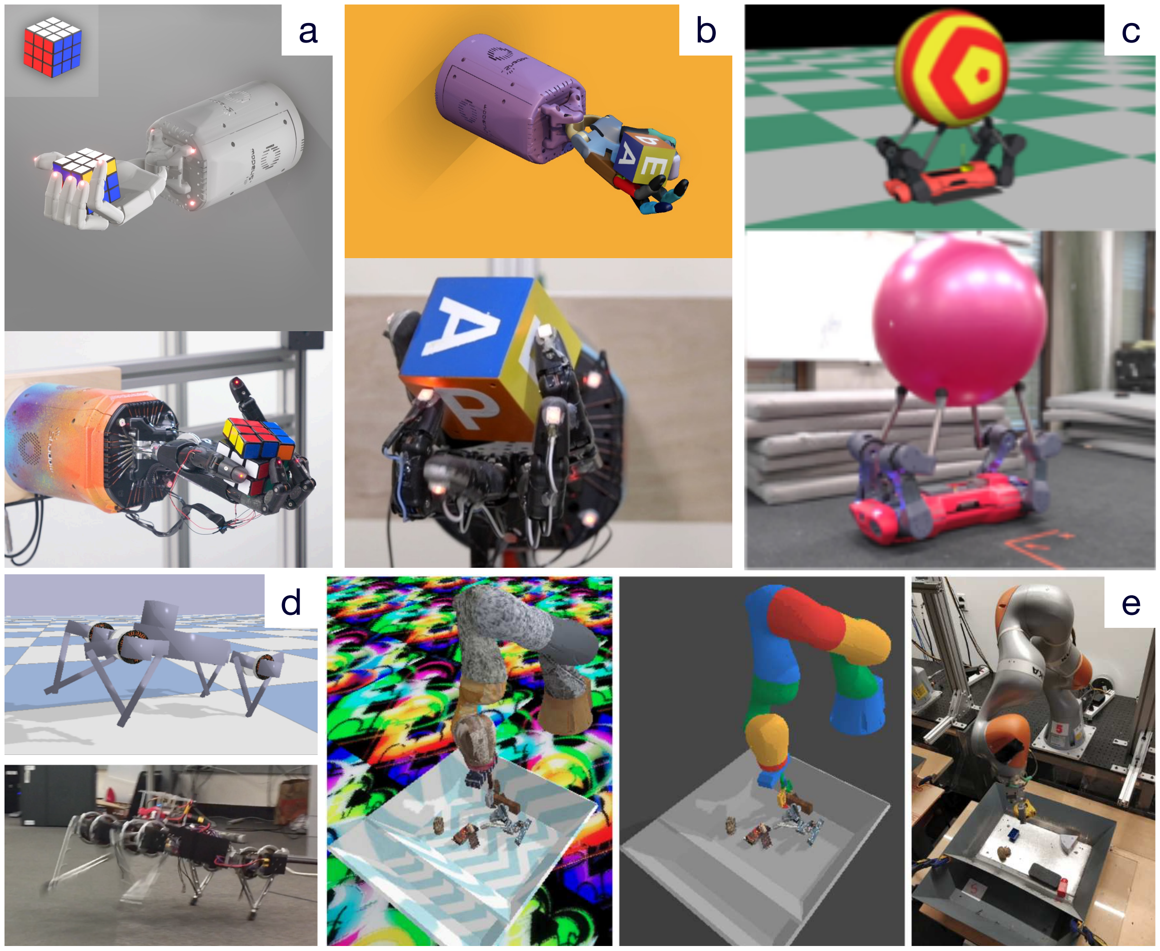


Figure 2.2: Sim-to-real examples from recent literature. a) Solving Rubik's cube with one robotic hand [10], b) dexterous manipulation with one robotic hand [9], c) manipulation using quadruped [12], d) quadruped locomotion [13], e) robotic manipulation with canonical simulation paired with real and simulated environment [11]

and the challenges of the 'reality gap' [33] between simulation and real-world when deploying the models in real-world complex environments.

2.2.1 System Identification

System identification is the method of 'calibrating' the simulator to improve the realistic representation of the physics and dynamics. Simulators are inherently prone to unreasonable outputs, especially for unreasonable input parameters. Modeling physical parameters like friction, inertia, and mass for each and every robot part is nontrivial and cumbersome. Most zero-shot transfer methods focus highly on system identification but this requires a lot of expertise and experience for successful implementation. Some level of system identification is performed by almost all successful sim-to-real transfer involving robots with dynamics and actuators involved [13, 34, 35].

Often used as an alternative to system identification, domain adaptation, and domain randomization techniques can be used to increase the robustness of the Agent, and to better capture the actual system dynamics.

2.2.2 Domain Adaptation

Domain adaptation is the idea of training an RL agent in a domain where data is abundant and then later transferring the learning to a domain where data is sparse. Domain adaption may include visual or dynamics adaption.

Visual domain adaption has been successfully demonstrated for 'zero-shot transfer' in robotic manipulation applications [36, 37]. The concept behind visual adaption proposed by [37], is that the agent 'learns to see' before 'learning to act'. The idea of 'learning to see' is implemented by training a variable auto-encoder (VAE) in an unsupervised manner that would learn 'a latent state representation' common to the source and target domain. Learning actions after the representation enhances the success rate of domain adaptation in both sim-to-sim and sim-to-real.

This disentanglement of perception approach is taken a step further by [11] where a cGAN (conditional generative adversarial network) [38] is trained to produce non-random canonical images from the randomized simulation images, and also from the real world images. The randomization of the simulation environment adds a domain randomization component (see Section 2.2.3) while canonical image generation improves domain adaptation. The agent learns from both simulation images and canonical images, allowing it to fine-tune joints on the real robot based on the real image and corresponding canonical image after deployment.

There's also a proposal to implement bi-directional domain adaption, real-to-sim for the visual domain and sim-to-real for the dynamics domain [39]. Real-to-sim is implemented using cGAN (cycle generative adversarial network) [40] with the argument that if the sensor characteristic changes from simulation to the real world but dynamics remain the same, the policy can make sense of the real-world observations through the lens of the cGAN. Sim-to-real is done using a perceptron regression network that residual error for state transition in simulation and reality. This network is then augmented in simulation for the agent to learn from the simulation while simultaneously being aware of the difference in dynamics in the real world.

2.2.3 Domain Randomization

Domain randomization improves the policy's robustness in case of dynamics or modeling errors. By adding variation to the physics parameter, the agent learns to handle varying real-world dynamics or disturbances better. Success with domain randomization can be seen in many vision-based sim-to-real transfers: learning quadrotor control without any real image [41], successful transferred object detector by adding variation to simulation visuals [42], or using locomotion of quadruped as manipulator [12].

An extension to this technique by [10] is automatic domain randomization (ADR). ADR is the process of gradually increasing the difficulty in the simulation while randomizing the dynamics. This method demonstrated robustness in learned policy as well as a reduction in training time while capturing the physics better than just the randomization technique alone.

Chebotar *et al.,* [43] proposed a method for dynamics randomization by periodically augmenting the training with real-world experience by deploying the agent in a real robot and updating the network parameters for the deviation in simulation dynamics observation from the real world.

Valassakis *et al.,*[44] argues that engineering domain randomization is nontrivial and requires a significant trial-and-error along with real-world data. On the other hand, random force injection requires very little setup and processing but performs consistently better than domain randomization in most of the reaching, pushing, and sliding tasks.

For robots that operate on cartesian space with a low-level controller can transfer to real robots without dynamics randomization while maintaining operational safety constraints [34]. In fact, dynamic randomization caused the training to be more unstable but system identification gave better results.

2.2.4 Simulation Environment

Simulation environments play a very important role in the success of sim-to-real transfer. The choice of the simulator is driven by many design choices, such as the task type (vision-based or control-based), required emulated/supported sensors, support for physical properties (such as contact, friction, deformation), collision detection, rendering speed, etc. Some simulators are standalone with a physics simulator and rendering engine, while others use physics engines that can interface with rendering engines of the user's choice for simulation. The ability to run headless and faster than in real-time is also a primary selection criterion for scalable RL learning.

Py-Bullet [45], and MuJoCo [46] are among two popular simulators among the RL community used by many to simulate robot environments [10, 47, 48]. Although both have native OpenGL rendering support, MuJoCo also offers the option to render using third-party rendering engines such as Unity [49]. Both of these integrate very well with Python-based deep learning libraries, but they are not as straightforward when connecting to a real robot. Gazebo [50] is another popular simulator choice, especially for robotic applications with inverse kinematic computations. Gazebo interfaces well with Robot Operating System [51] which interfaces well with any robots or network of robots.

For a UAS simulation, RotorS [52] is a popular UAS simulator based on Gazebo. In recent years, AirSim [53] has gained popularity for applications involving imaging or surveillance using UAS. AirSim interfaces with the Unreal engine to provide a photo-realistic environment. However, it uses a separate physics engine from the Unreal engine and has limited support for added external payloads.

Matlab Simulink is also an alternative to these popular simulators used for simulating UAS and rope/string connected payload [54, 55]. Simulink can simulate physics, render with Simulink 3D animation or Unreal engine, and connect to robots through ROS. Simulink offers a graphical way to design or model a dynamic system using blocks. Simulink blocks can represent a physical system, functions, and configurations in an input/output relationship. One benefit of this approach is that the user can make their own components or use numerous blocks available in the Simulink library as part of different toolboxes. Blocks can be reused in any compatible combinations, making the simulation environment highly configurable. This speeds up the process of building a simulation environment with the user having as simple or as advanced control over the design as desired. The block diagram approach has another important benefit when building physical systems, each block's input/output can be logged or analyzed; similar to doing unit tests to ensure they work as expected. Matlab and Simulink together provide a one-stop solution for designing the simulator environment, the model, and the RL agent, then training the agent, and transferring the agent to real robots. We chose to use Matlab for this reason to allow rapid prototyping and deployment of sim-to-real reinforcement learning.

2.2.5 Reality Gap

The reality gap refers to any difference between the mathematical model, simulation environment, and the real world. Collecting data from real robots can be very time-consuming and expensive [56], unrealistic or impossible in many cases where safety is concerned [57], or where the robots can be damaged in the real-world experiments [9]. Furthermore, these repairs can change the 'model' of the robot and physical parameters thus causing further gaps in reality [9]. Another problem of learning directly in real robots is often not collecting sufficient data with large enough randomness distribution to make the RL agent robust enough for unseen scenarios. Examples of collecting large-scale data using real-world robots also exist but the resource utilized to train the agent in such a manner is exceptionally high and not practical for most researchers [58]. Consequently, most researchers use a simulation environment to generate the data for training.

The simulation environment where the robot is modeled is a common source of reality gap [5]. The simulators that simulate accurate physics alongside collision, deformations, frictions, joints, actuators, etc., become too slow to be realistically useful for large-scale RL training. This slowdown stems from the complexity of solving the implicit dynamic equations that emerge in such models. For this reason, most simulation software in use by RL researchers (e.g., MuJoCo [46]) do not simulate actual physics but rather a 'close-enough' prediction of it that looks 'realistic enough'. This trade-off allows the simulations to run faster than real-time, and to be able to collect data for large-scale RL learning applications.

Lack of realism in the rendering of the environment can pose challenges to a successful transfer. One of the key failure points is the difference in visual observation in simulation and reality [59], especially if the task is highly dependent on vision.

The simulated sensors may not match the real ones in values, output dimensions, or frequency of output. These sensor implementations can be different depending on the simulation platform. Without a thorough understanding, matching or calibrating such sensors with real ones can be difficult.

In real robots disturbances from the environment, sudden changes of contact and friction parameters, and network dropouts are very common but these are nontrivial to implement in a simulator. Even unintentional rounding of numbers, often a trade-off for faster computation, can also contribute to the reality gap.

Consequently, researchers are focusing on bridging the 'gap' with many alternative techniques such as adding random perturbation force inputs during training [44], occasional real-world rollouts to update policy by augmenting real experience [43], collaborative learning by multi agent [60], or training predictors for sim-to-real transfer success [61].

2.3 Background on UAS Aerial Leaf Sampling

Researchers study leaves for understanding the physiology of plants (e.g., nutrient composition of the leaf, studying the effect of sunlight and shade on the tree's development, study genetics or species of the plant, etc.), ecology of the forest (e.g., nutrient cycling of a forest, the impact of the availability of water or drought on plants, etc.) or forest interaction with environments (e.g., studying the microclimate in the forest canopies and their impact on the local climate, and impact of global warming on the plants etc.).

Researchers have found many different ways to sample leaves from tree canopies over the last few decades. Some of these methods require expert humans for success: tree climbing, pole pruner, ladder, slingshot, shotgun, rope launcher, etc. Most of these methods either have a low reach, low access (only inner stronger branches), or have the potential risk of injury to the operator or the wildlife.

Some other methods that can give access to the upper canopy area require expensive massive structures such as: canopy tower, scaffolding, canopy crane [62, 63], canopy walkway, hydraulic lift, canopy rafts or helicopter. These traditional canopy sampling methods impart bias in site selection based on available access method, limits the number of samples collected, and is harder to replicate while being difficult to move between sites [64].

Based on a survey conducted in 2001 among canopy researchers: ground access, tower, and walkway are the most used access methods by plant physiology and ecology researchers [64].

Figure 2.3 shows some of the popular methods used by canopy researchers. Some of these access methods are also used for other canopy research such as wildlife monitoring, the study of insects and other critters, etc.

The majority of the tree canopy still remains hardly accessible. New studies are challenging the previous understanding that assumed properties of the top sunlit leaves are representative of the entire canopy [72, 73]. The lack of leaf sampling data in three dimensions is shifting the interest toward using the UAS in canopy sampling. UAS aerial leaf sampling has improved access to forest canopy for leaf sampling in recent years. Figure 2.4 shows currently available leaf sampling UASs with the sampling tool magnified in the inset.

UAS attached with a form of the manipulator that can cut or chop leaves, twigs, or branches [2, 69, 3]. Current aerial leaf sampling systems are mostly operated manually and are currently only suitable for accessing the side or top of the canopy in favorable wind conditions. Research in rainforest canopies, for example, would require a system that can access the more complex canopy layers of the forest as well

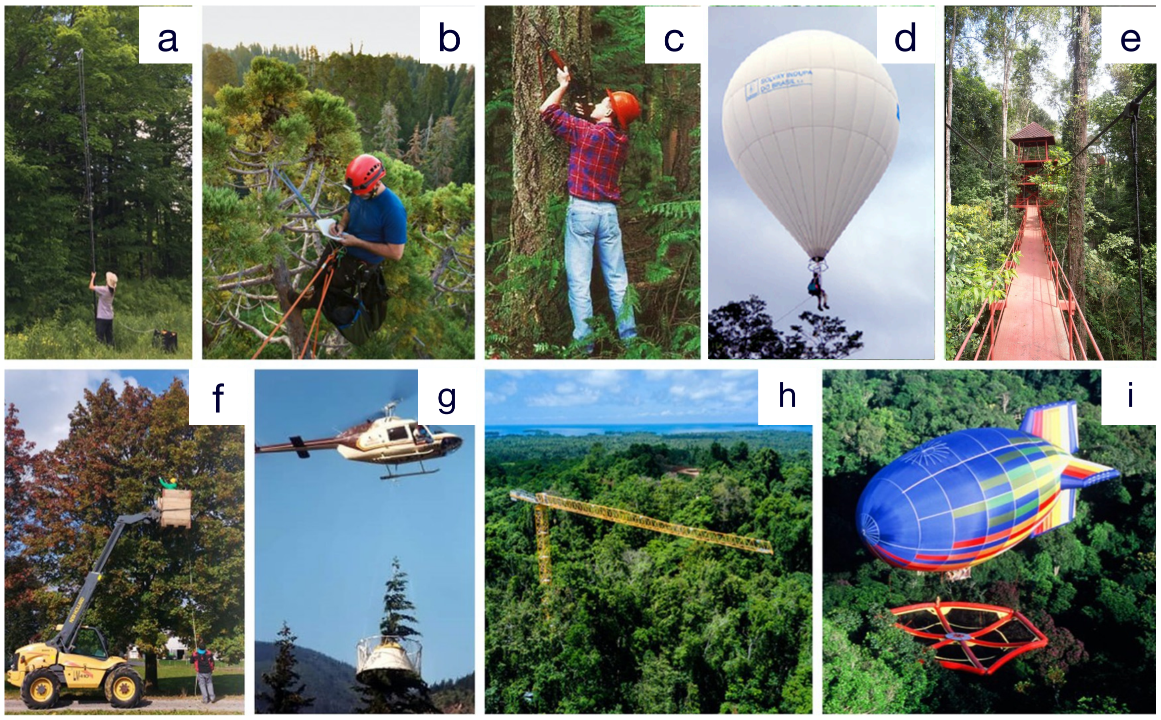


Figure 2.3: Forest canopy access techniques used by canopy researchers over the years: (a) pole pruner [2], (b) climber [65], (c) shot gun [66], (d) canopy balloon [67], (e) canopy walkway [68], (f) hydraulic lift [69], (g) helicopter [66], (h) canopy crane [70], (i) canopy raft [71]. Image collage modified from [2].

as epiphytes that grow on the trees.

While the use of UAS in leaf sampling has taken a major leap in mechanical design, the use of automation is still lacking. There has been some progress on adding vision to assist the pilot [74], or branch tracking algorithm by modeling tree branches to design assistive controllers [75].

Existing leaf samplers in this context can be classified using five design choices: mounting options, choice of cutting tools, the orientation of the target branch/leaf samples, target canopy position, and the UAS control method.

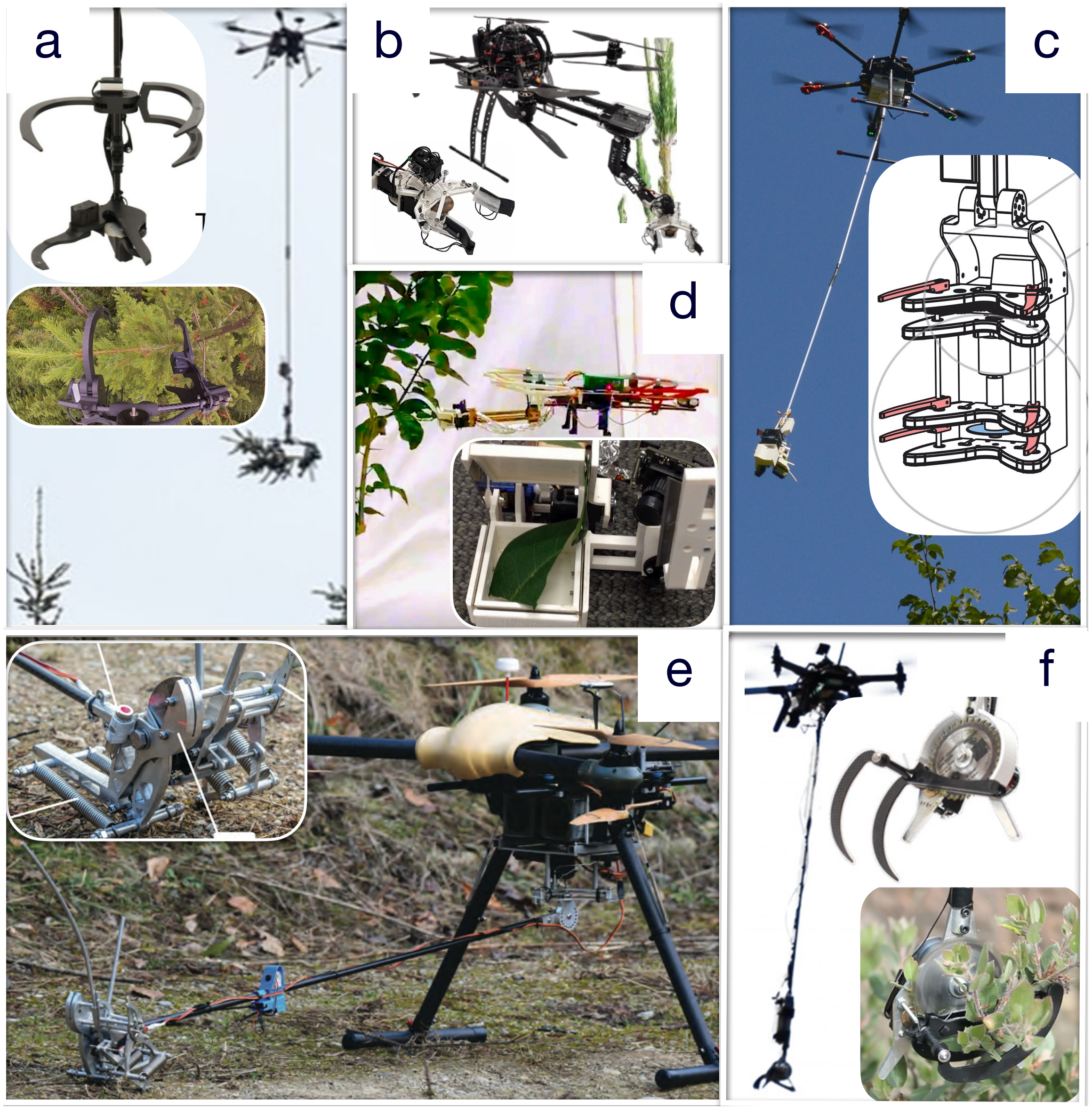


Figure 2.4: Currently available UAS leaf samplers in research publications. The close-ups of leaf sampling mechanisms are shown in the inset. Image source: (a)Charron *et al.,* [2], (b)Kutia [3], (c)Kaslin *et al.,* [69], (d)Orol *et al.,* [76],(e)Finzgar *et al.,* [77], and (f)Wu *et al.,* [78]

2.3.1 Mounting options

The primary design choice of UAS leaf sampling tools is whether the mechanism is extended horizontally outwards from the UAS [3, 76] or suspended below [77, 69, 78, 2].

Extending the mechanism horizontally reduces or eliminates the impact of the propeller downwash [79] on the branch; making it relatively easier to grab. This is also useful for reaching towards trees that have mostly upright branches such as *Pinus radiata* tree sampled by [80]. However, this design choice requires balancing the force exerted by the moment arm on the UAS by offsetting some mass, i.e., uas battery [76] or by controller compensation [80]. Another drawback of this design is the potential to transmit high-frequency vibration to the UAS controller resulting from the cutting action or wind-branch interaction when the sampler is coupled to the branch. This coupling requires careful design of the controller to avoid overshooting the integral controller [80]. The mechanism also potentially could impart horizontal moment on the UAS as it is rigidly connected to the UAS while also coupled to a branch that can sway with the wind [81].

The more popular design choice is suspending the sampler down from the UAS. Since mounting the sampler is at the center of gravity this eliminates the torque acting on the UAS. Hence, commercial UAS can be used without the need to design a custom controller [69, 2]. However, this design choice requires using a long extension to reduce the effect of propeller downwash on the target leaves. The long extension can be mounted on universal joints to further dampen the effect of the sampler coupling with a swaying branch. Long extension however can induce pendulum motion that may need to be accounted for.

Another approach explored for aerial pruning UAS is vertical mounting the mechanism on the top of the UAS [82]. This approach however is prone to the UAS being knocked over by the branch being cut. A similar mounting technique with a small drone equipped compliant mechanism mounted on the top or bottom to collect leaves by the movement of the UAS is shown by [83]. The method while being novel results presented only shows the indoor static condition of the leaves. Any wind disturbance outdoors will make it significantly harder for such mechanisms to be successful.

2.3.2 Cutting tools

There are three main types of cutting tools used by aerial sampling UASs - saw [69, 3, 78, 2], shear [77], blade [76].

Saw is the most popular choice for its simplicity and available cutting power. It is also much more controllable with repeated consistent results compared to a shear or blade. However, all the mechanism with one blade suffers from gyroscopic effect as noted by [2]. This is countered by adding a second blade rotating in the opposite direction in [81]. Adding two saws presents more control challenges as the chance of motor stalling increases.

One important characteristic of all saw design is that it has some form of gripper or claw mechanism to hold the branches as its performing the cutting action. Lack of this gripper results in failure to cut any branch [84]. Similar failure can occur in mechanisms using shears without using any gripper mechanism [85].

These mechanisms are by design restricted to collecting branches instead of just leaves. While this might be desirable in some cases where collecting samples of flowers, pollen, seeds, or fruits; the woody part is unused extra weight that is discarded and is not used for analysis [86]. Because of this additional woody part, 100 g to 400 g of sample collection was being targeted for 30 g to 40 g fresh leaves by [2]. Carrying this 3 times to 10 times extra weight far from the center of gravity of the UAS, and the wind drag profile from carrying such a sample also adds to the stability control burden of the UAS and reduces the flight time.

Some researchers used shear actuated by spring for cutting action [77]. The shear mechanism requires significant torque for cutting action. However, in an outdoor environment where the branch might be affected by wind movement- it would present a serious challenge to successful sampling as the orientation would affect the performance of cutting.

The blade is used in [76] to cut sections of leaves while [87] uses a boom-mounted blade to cut branches from the tree using UAS movement. Both of these methods are deemed to be unreliable in the outdoor environment with more atmospheric wind at play.

2.3.3 Sample orientation

All the leaf sampling or branch pruning UAS described above are very sensitive to the orientation of the incoming branch or leaf. Although [2] the claim to admit sample of any orientation, the adjustment must be made before the flight, and knowledge of the branch orientations is a pre-requisite for the said adjustments.

One way the researchers have worked around the sensitivity of the orientation is by using larger gripper claws to admit large errors from swaying branches. This comes at the cost of further reduction in flight time.

Also, this is critical for the operation and when manually piloted- it requires synchronized movement by two skilled operators one operating the UAS and one operating the sampling tool using camera feed from the UAS [2].

Another way to address the orientation issue is the use of computer vision techniques for branch tracking and designing a controller that uses this additional information [75] for the custom UAS controller design. Alternative use of the branch tracking is to design an assistive high-level controller in a semi-autonomous mode that moves the UAS towards the target branch [74]. These branch tracking algorithms are currently however leaf sampler design specific and might be only applicable to a specific plant or few specific types of plants; generalizing the use of such tracking method for all kinds of plants is challenging but need to be forthcoming for automation.

2.3.4 Target Canopy Position

The above-described mechanisms are only suitable for collecting branches at the side or top of the canopy due to the reach of the respective manipulators, and these systems are not designed for flying inside a tree canopy. One critical design consideration for this choice as argued by [2] is that collection of sunlit leaves is required for foliar experiments. However, other canopy researchers argue that sampling only the sunlit leaves does not represent the foliar or spectral traits of the entire canopy as previously thought. In fact, only chlorophyll and leaf area index (LAI) are more accurate when sampled from the sunlit upper canopy, but foliar nitrogen (N), carbon (C), and leaf mass per area (LMA) were more accurate when leaves were sampled from both sunlit upper and shaded middle canopy [88]. Most of these trait variations also persist throughout seasons [89] and affect the physical models used by canopy researchers. These findings are also supported by the previous authors from [2] in their follow-up work [73].

In this thesis, we are interested in leaf sampling in the forests where plants of interest or leaves of interest might be deeper in the canopy. As such, we design our leaf sampler as a cable-suspended payload that can be lowered by a winch mechanism when the UAS is at the sampling site. This allows the UAS to be at a safe distance from the canopy top, avoid the effect of propeller downwash, and reach deeper in the canopy.

2.3.5 UAS and Leaf Sampler Control Method

Controlling the UAS and the leaf sampler is a non-trivial task. The unstructured nature of the trees, random disturbances caused by winds, and leaf sampler tool are hard to mitigate but also safety critical.

Leaf sampling tools discussed above are partially automated in terms of triggering mechanisms but still require positioning the branch at a specific angle and position of the leaf sampler. This is usually done manually using wireless radio by a second operator who is watching a video feed [2] or from the operator's console [69].

Controlling the UAS is tricky with a sampler that extends horizontally as they require a custom controller [3, 76]. On the other hand, the vertical orientation allows using commercial UAS with a default controller with mechanical damping for vibration and load transfer to UAS [2, 69].

We propose using a model-free reinforcement learning (RL) algorithm policy due to the nature of the complexity of modeling the problem of leaf sampling, the unstructured nature of tree canopy and the potential UAS interaction with dynamic coupling, lack of availability of data to train the real robot, and safety-critical nature of the project.

2.4 Selection of Real Hardware: UAS vs CNC

While our final target real system is the leaf sampler mounted on the UAS as Figure 2.5, we selected a modified CNC platform as the real hardware for sim-to-real transfer. CNC works as an intermediate 'real' system step towards using sim-to-real agents on the real UAS.

Simulating the UAS dynamics is relatively easy in the simulation environment, however, it is not possible to perfectly simulate the real world and account for all the variability that might be present in the system. This is especially true for sensor noise, positioning error, effect on thrust from decreasing battery voltage, manufacturing defect, operating conditions, atmospheric disturbances, etc. that might occur during a UAS flight. Validating a RL agent or control algorithm directly in the real world to operate a leaf sampler mechanism through tree branches is nontrivial. Using CNC provides a safer alternative to validate the RL agent that is trained in simulation. This also allows validation in repeatable test scenarios with controllable complexity and disturbances that may occur during a UAS flight.

Using a CNC provides us with multiple benefits instead of using the UAS during development: 1) The CNC is plugged into a power outlet and can provide unlimited operation where a UAS needs to change its batteries very often, 2) We implemented hardware and software safety constraints on the CNC that reduces accidents and crashes that might otherwise occur when running experimental RL agents directly on a UAS. In essence, the CNC provides additional safety nets in finding potential problems during development and experimentations with the RL agent algorithms, 3) the CNC can be used to fine-tune the simulator's learned model before deploying to the UAS to reduce the risk of crashes, 4) the CNC can be controlled using the same high-level velocity controls that UAS can also be controlled with.

2.4.1 Similarities and Differences in dynamics

A UAS has six degrees of freedom (3 linear axes, and 3 rotational axes), and movement in all six axes is controllable. The leaf sampler we designed operates using a winch system that will be attached to a UAS and can linearly move up or down along the z-axis. In a three-axis CNC, where the z-axis uses a winch mechanism, there are 3-linear axes movements that are controllable. Additionally, due to the flexible material of the winch system in the z-axis of the CNC, we see limited movement in all three

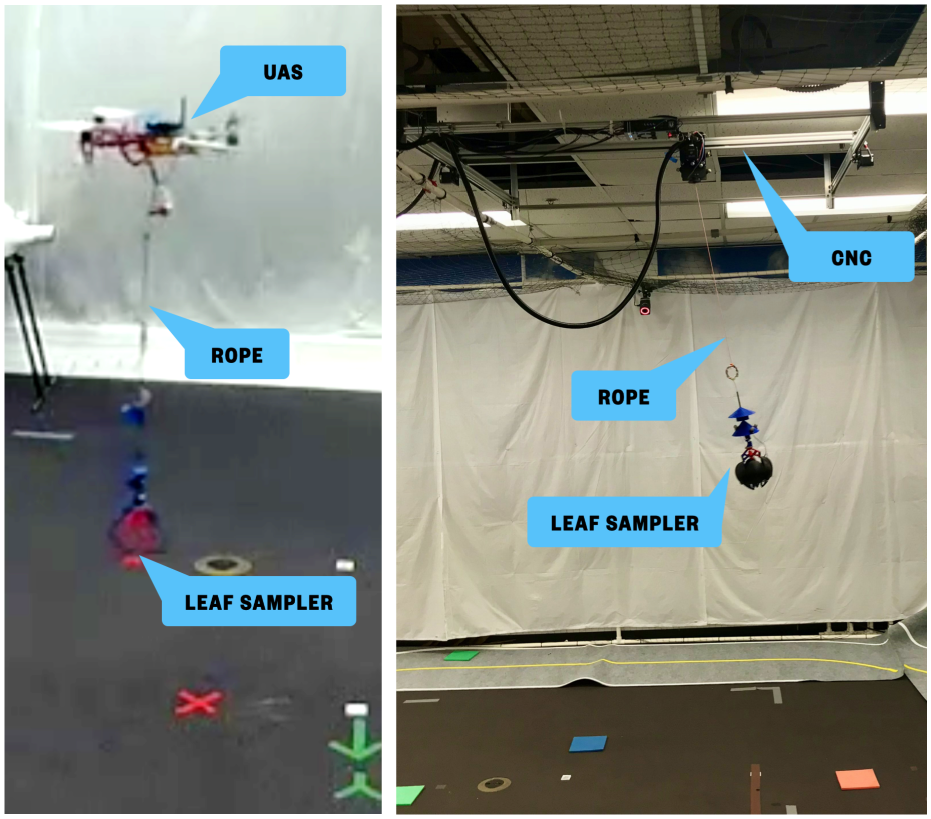


Figure 2.5: A leaf sampler mounted on a UAS (left) and CNC (right) using a braided rope.

of the rotational axes (pendulum motion in two axes and torsion/twist motion of the rope) that are observable, but not controllable. We selected the braided rope to reduce the torsion significantly but the other two rotational axes' movement can still occur when the CNC head moves. Figure 2.5 shows how the leaf sampler is mounted on UAS and CNC using a rope.

In a limited movement scenario where the UAS is hovering over the tree and we are operating the leaf sampler mechanism in the z-axis, or if we are moving the UAS slightly within a small bound, it can be approximated by the movement from the CNC.

Chapter 3

Sim-to-Real Reinforcement Learning Framework

In this chapter, we describe the reinforcement learning framework for training the RL agent in the simulation environment. The framework, shown in Figure 3.1, is divided into two parts: 1) RL agent, and, 2) Simulated Environment.

3.1 The RL Agent

In this thesis, we explore the performance of the twin-delayed deep deterministic policy gradient (TD3) [19] RL algorithm due to its robustness in continuous action space as described in Chapter 2. We validate our reinforcement learning framework by training our RL agent to move the leaf sampler to a target position from a random initial position.

We use the TD3 agent with two critics and one actor as it reduces the Q-value function overestimation bias while making the training more stable. This stability and improvement are achieved by using the minimum Q-value estimate of the two critics, updating the actor's policy and targets less frequently than the critic's Q functions, and by adding noise to the target action to reduce the chances of the policy exploiting actions with high Q-value [19]. The deep neural network architecture of the actor and critic uses fully connected deep neural networks with ReLu activation

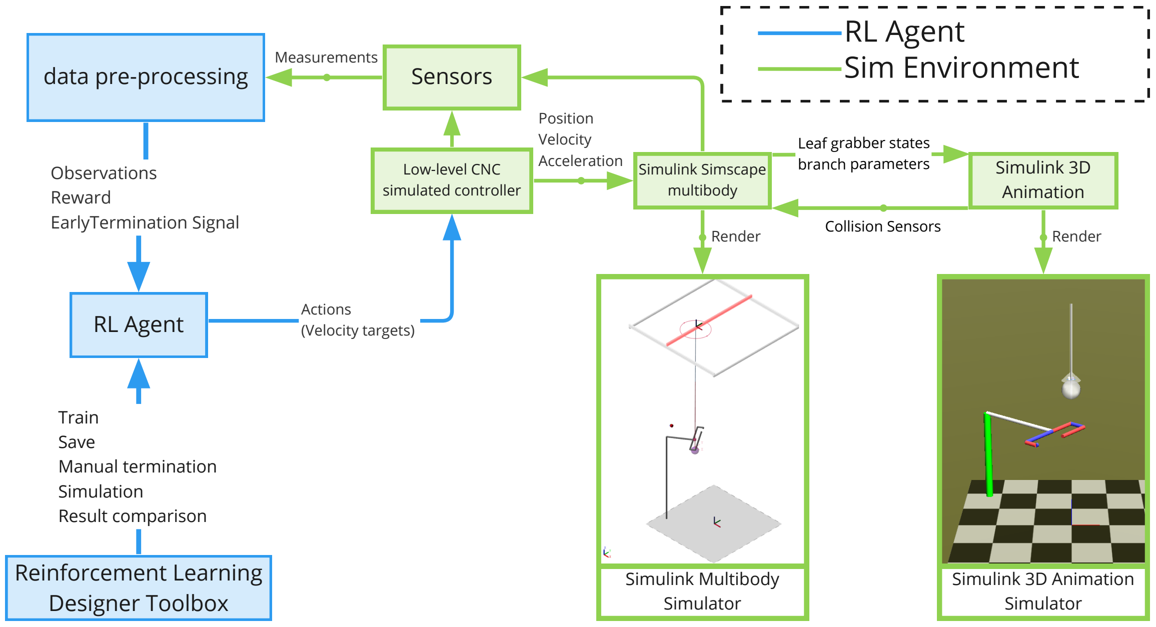


Figure 3.1: Reinforcement learning framework to train the RL agent in the Simulation environment (represented in the diagram with blue and green colors respectively). RL System agent interface with simulated CNC through a low-level controller that accepts velocity inputs to update the simulation environment. Creation, training, and validation of the agent are done in the Reinforcement Learning Designer toolbox.

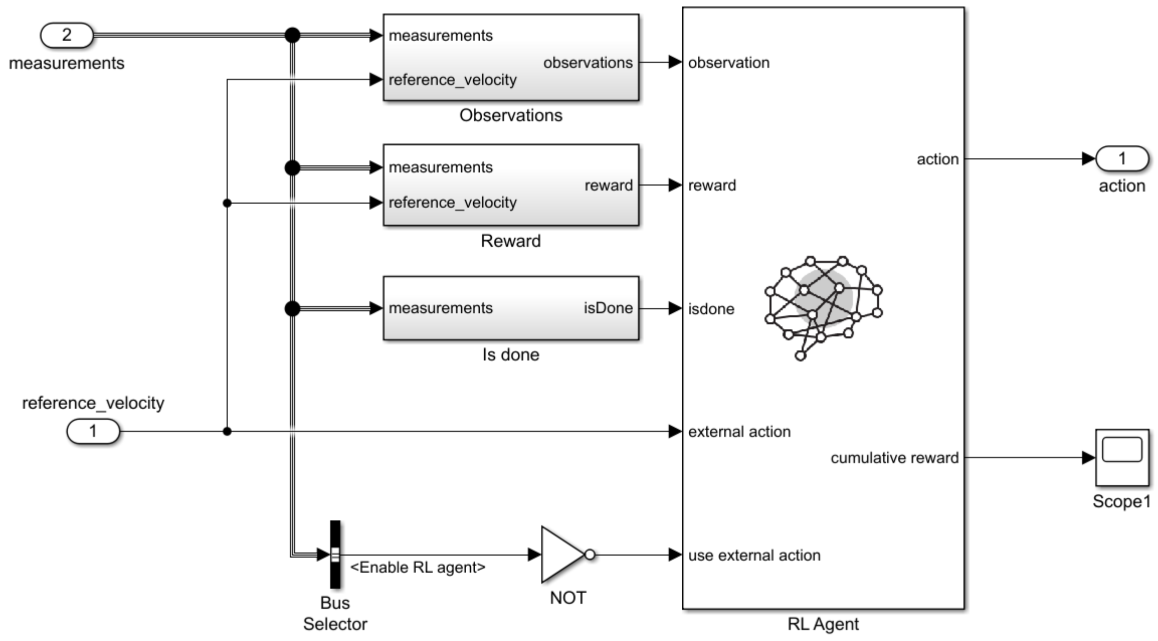


Figure 3.2: Simulink RL algorithm structure in block diagram format. External action is used to pass actions directly similar to imitation learning when the RL agent is not required to act.

layers between them. The actor has a Tanh layer as the last layer so the output can be limited between $[-1, 1]$. TD3 agent adds a random noise in the action to encourage exploration. The default noise model available to use with TD3 is the Gaussian noise model or Ornstein-Uhlenbeck (OU) noise model. We used the Gaussian noise model for the action noise and used action limits and scaling to pass to the low-level controller simulating the CNC. Although, Lillicrap [31] argues that OU noise would better represent continuous control application, where the inertia of the system would make the noise correlated similar to the 'Brownian motion'. However, Barth-Maron [90] argues that it is unnecessary to use OU block to replace Gaussian noise as it does not add to the performance. Figure 3.3 shows the actor-critic setup used in this thesis. We prepare observations, reward, and early termination conditions from the measurements and reference as shown in Figure 3.2 and described next.

3.1.1 Observations

We selected sensors and calculated measurements as observations for RL agents that would be common for both the real system and the simulated system. The observations selected for the real CNC are also observable from real UAS to reduce the adaptation required for deployment of the RL agent in real-world target systems.

We provide a total of 10 observations from the current simulation step and the same 10 observations from the previous step. This is to encourage the agent to learn its own smoothing and interpretation and estimation of velocity, acceleration, etc. [91, 12]

In particular, 1) we observe the position error and velocity of the CNC in the x-y axes since it drives the overall positioning of all the other components; 2) position error between the CNC origin and leaf sampler origin as we intend to move the leaf sampler in sync with the CNC; 3) position error of the leaf sampler in the z-axis,

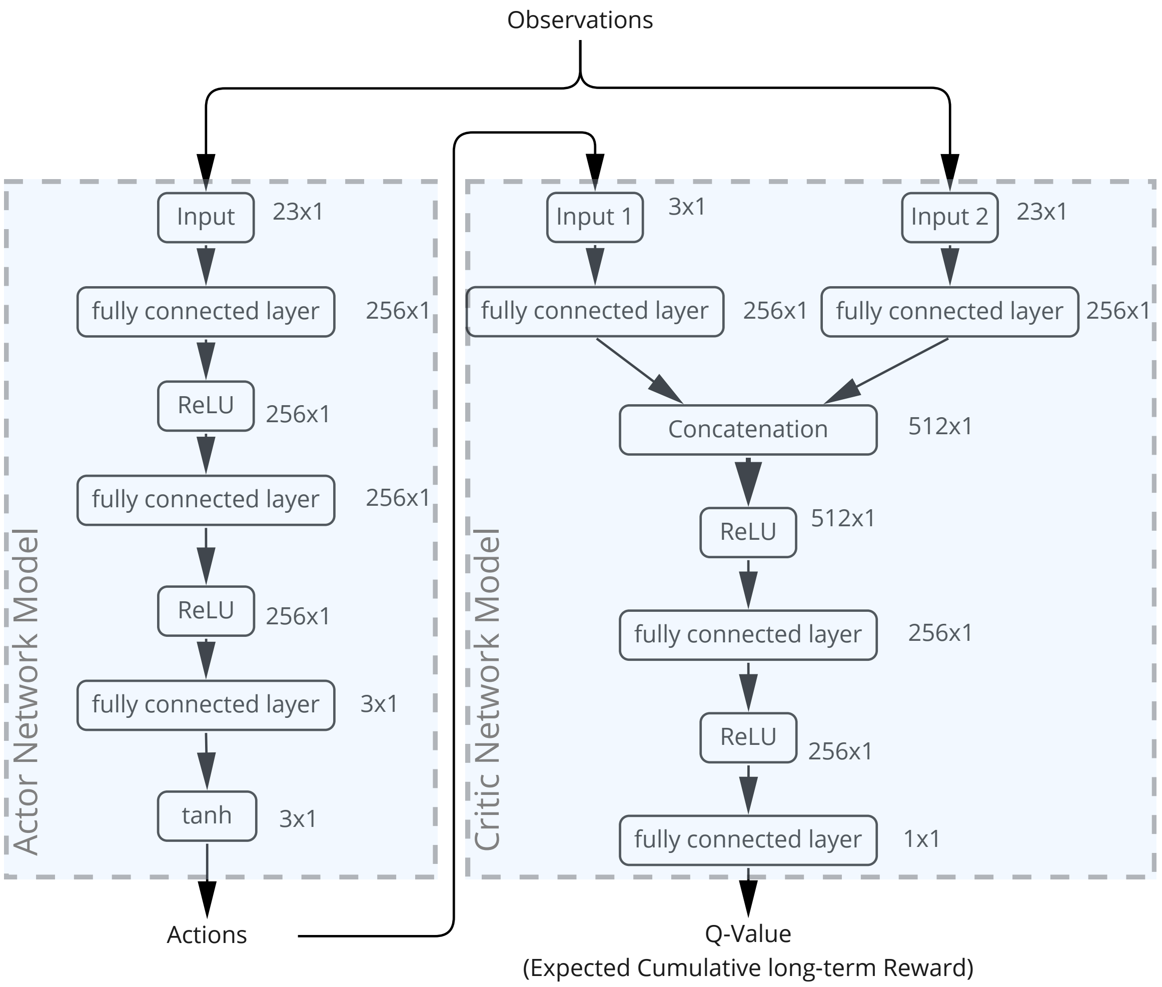


Figure 3.3: Network architecture model of the actor-critic used with the TD3 agent. TD3 agent uses one actor, and two identical critic models (one shown for brevity).

and leaf sampler velocity in x-y-z axes as this indicates where the target sampling position is relative to leaf sampler. The agent also receives actions taken by the RL agent in the last simulation step as additional observations. Thus total observation space is $\mathbb{R}^{23}$. The size of this observation space varies in literature but depends on the number of joints and actuation the robot may have. For example, we see the successful landing of UAS on a moving platform using observation size $\mathbb{R}^{7}$ [18], quadruped learning dexterous manipulation using $\mathbb{R}^{130}$ [12], solving Rubik's cube with robotic hand using observation space $\mathbb{R}^{478}$ for block orientation task [10].

3.1.2 Rewards

Crafting a good reward function can be tricky as it often comes with experience and a deeper understanding of the system as a whole. In the scope of this thesis, we are only concerned about moving the leaf sampler to a target position from a random position as fast as we can. We craft our reward function to achieve that goal:

$$\begin{aligned} reward = \quad & -\alpha \times a_{t-1}^2 && (3.1)\\ & -\beta \times (cnc_{x_{error}}^2 + cnc_{y_{error}}^2) && (3.2)\\ & -\gamma \times (sampler_{x_{error}}^2 + sampler_{y_{error}}^2) && (3.3)\\ & -\delta \times sampler_{z_{error}}^2 && (3.4) \end{aligned}$$

The first element of our reward (Equation 3.1) penalizes for action taken by the RL agent in the previous step a_{t-1} to prevent movements when the sampler is sufficiently close to the target location. The second element (Equation 3.2) penalizes for error in CNC x-y origin and target position. This encourages fast convergence to the target location as this component moves all the other components with it. The error in CNC origin is also more predictable compared to the leaf sampler position which may oscillate like a pendulum due to the flexible z-axis. The third element of the reward function (Equation 3.3) penalizes errors in leaf sampler origin and CNC origin. This penalty encourages the agent to minimize movements that induce oscillation of the leaf sampler and also move the CNC origin in sync with the leaf sampler. The fourth element (Equation 3.4) encourages the agent to minimize error in the z-axis and move the leaf sampler to the correct height. Separating it from horizontal errors gives the option to prioritize it to converge to the target value faster.

We use scalar multiples, (α, β, γ, δ), that are determined through experimentation and tuning. The final values used for experiments are presented in Table 4.3.

3.1.3 Early Termination

For this thesis, we ran each episode to a maximum number of steps. This number of steps is heuristically set for the experiments and the final value is listed in Table 4.3. We kept it simple since we had software constraints in place that prevents catastrophic CNC movements. Also, our validation experiments were simple enough that otherwise complex early termination criteria were not necessary.

However, more complex tasks such as navigating inside a tree canopy with the leaf sampler would benefit from early stopping to avoid catastrophic scenarios, and also to gather more relevant experience per episode for the experience buffer.

3.1.4 Domain Randomization

Domain randomization techniques improve the success of the transfer of the agent. We implement domain randomization techniques in the simulation environment. At the start of each episode, the position of the CNC and leaf sampler is randomized and the target location is also selected at random.

3.2 Simulation Environment

We created the simulation environment using the Matlab software suit as discussed in Chapter 2. The following sections describe the simulated CNC hardware, sensors, and the control interface employed in the simulation.

3.2.1 Simulated Hardware: CNC

We create the CNC in the simulation environment using a suit of Matlab software: Matlab, Simulink, and Simulink 3D animation. Matlab facilitates the design and training of RL agents, and communication of software and hardware using ROS nodes; Simulink is used to model the physics of CNC and also the environment where the RL agent would interact; Simulink 3D animation is used for modeling collision and other sensors that are not available in Simulink.

The CNC in the simulation environment is designed with the same dimensions and motion limits as the real CNC. Simscape multibody toolbox is used to model the axes and the winch mechanism. The rope is modeled as a series of discretized links connected with universal joints. The number of links affects the accuracy of rope behavior, with a higher number of links being more accurate at the cost of simulation speed. Figure 3.4 and 3.5 show the simulation environment setup and the block-based design in Simulink respectively.

3.2.2 Sensors

In Simulink, many sensors are available to attach to the physical frames of different objects to measure movement and rotational displacements, speeds, acceleration, etc. We can pre-process these measurements to generate appropriate observations as designed for the RL agent. Collision-based sensors are available through Simulink 3D animation visualization environment. Simulink 3D animation uses X3D language similar in structure to URDF that is often used with other simulators described in Chapter 2. The model in Simulink communicates with the 3D renderer at specified update rates to compute the new position and rotation of the elements and process any sensor updates.

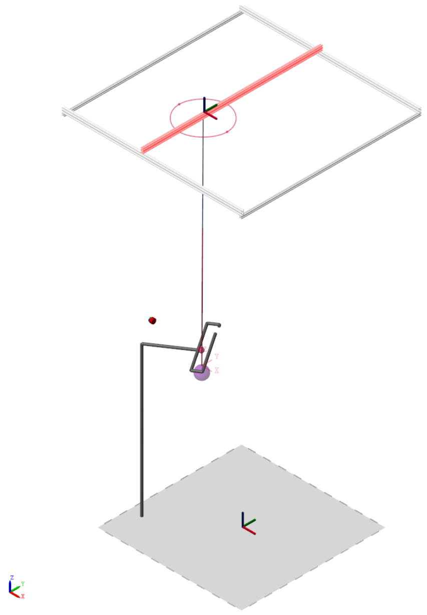
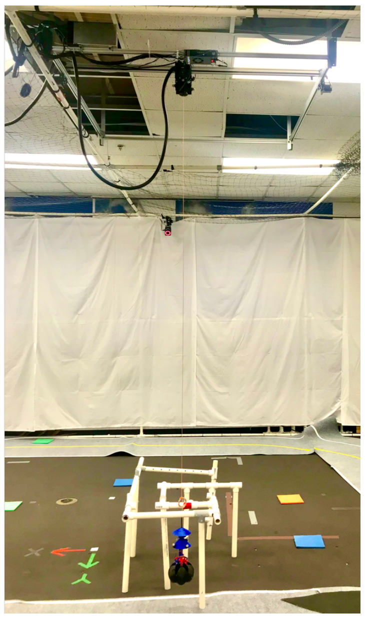



Figure 3.4: Rendering of the simulated robot environment (left), and the real-world robot setup (right).

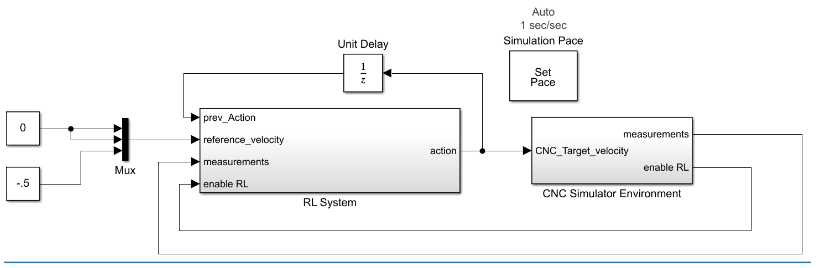


Figure 3.5: Simulation Environment Schematic in Simulink Block diagram

3.2.3 Control interface

A controller is implemented in Simulink that implements motion characteristics of the real CNC to the simulated one. This controller also takes target velocity as input and moves the CNC axes accordingly. The movements are implemented by calculating what the next position of the three axes should be given the current states and commanded states. Simulink multibody then calculates the forces necessary to achieve the desired state.

Chapter 4

Simulation and Experimental Setup

In this chapter, we discuss the RL agent hyperparameters and training workflow. We then describe the simulation and experimental setup we used for the sim-to-real transfer of the RL agent, we list the parameters and settings for each setup. We also discuss the validation experiments we designed to evaluate the success of the sim-to-real transfer.

4.1 RL Agent Setup

4.1.1 RL Agent architecture and hyper-parameters

We use the TD3 agent with two critics and one actor for our RL algorithm keeping the same structure as in Figure 3.3. Hyperparameters for the RL agent and actor-critic are tabulated in Table 4.1a, and 4.1b respectively. Hyperparameters for noise models to encourage exploration and for the target policy smooth model are listed in Table 4.2. These hyper-parameters are determined heuristically through training and experiments starting from recommended defaults set by the Matlab RL Designer toolbox. Since in this thesis, we are only validating our sim-to-real framework through trajectory following task, we keep the changes to a minimum from defaults. We experimented with the sample time and found 0.2 s provided faster simulation speed

Table 4.1: RL agent and actor-critic hyperparameters

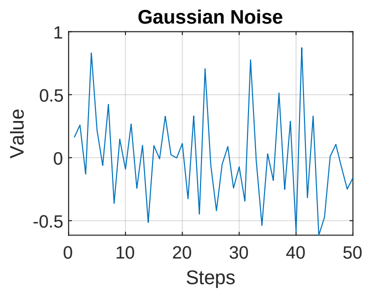
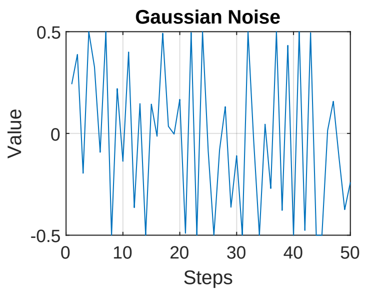
(a) Agent hyperparameters

Parameters	Values
Sample time	0.2
Discount factor	0.99
Batch size	256
Experience buffer length	1e4
Policy update frequency	2
Target smooth factor	0.005
Target update frequency	2
Reset experience buffer	Yes
Save experience buffer	No

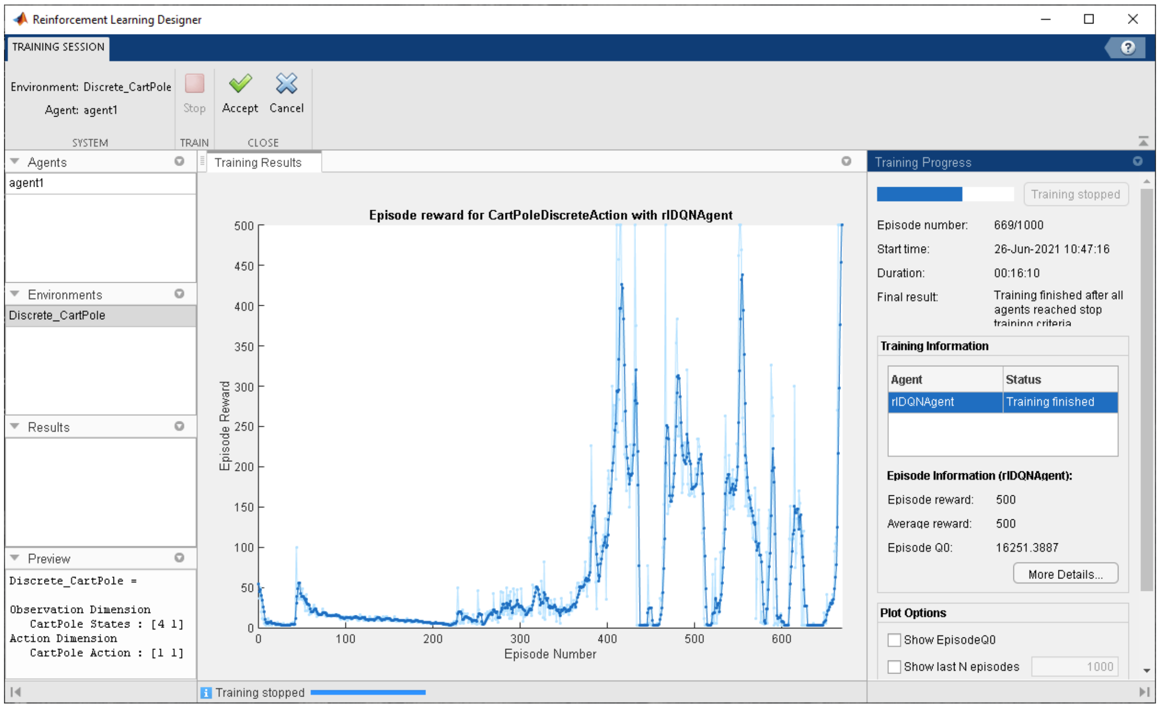
(b) Actor-critic hyperparameters

Parameter	Actor	Critic
Learn rate	1e-2	1e-2
Gradient threshold	Inf	Inf
Optimizer	adam	adam
Denominator offset	1e-8	1e-8
Gradient decay	0.9	0.9
Gradient threshold	l2norm	l2norm
L2 regularization	1e-4	1e-4

Table 4.2: Exploration and Target Policy smoothing method noise model hyperparameters

Parameter	Exploration Model	Target Policy Smoothing Model
Standard deviation (σ)	$\sqrt{0.1}$	$\sqrt{0.2}$
Mean (μ)	0	0
Lower limit	-Inf	-0.5
Upper limit	Inf	0.5
σ decay	0	0
σ min	0.1	0.1
Example plot		

compared to lower sample times. We also experimented with the noise model so that the standard deviation of the noise does not dominate the RL agent action output. We also added a decay in the noise which indicates the model would exploit the learned knowledge more and explore less as the training goes on.



Environment: Discrete_CartPole

Agent: agent1

agent1

Discrete_CartPole

```
Discrete_CartPole =

Observation Dimension
    CartPole States : [4 1]
Action Dimension
    CartPole Action : [1 1]
```

Episode number:	669/1000
Start time:	26-Jun-2021 10:47:16
Duration:	00:16:10
Final result:	Training finished after all agents reached stop training criteria

Agent	Status
rlDQNAgent	Training finished

Episode Information (rlDQNAgent):

Episode reward:	500
Average reward:	500
Episode Q0:	16251.3887

Figure 4.1: Matlab's Reinforcement learning designer app [92] showing training in progress. The left panel shows the RL agents, environments, and training results currently loaded in the Matlab workspace. The graph in the center shows rewards per episode. The right panel shows other training statistics such as the total number of steps, average reward, etc.

4.1.2 Training Workflow

We design and train the agent with a Reinforcement Learning Designer App [92] as shown in Figure 4.1. It provides a visual way of designing and training the agent and allows inspection and comparison of simulation data very easily between different runs of the simulation. The designer allows training a compatible agent for a selected environment that has the same number of observation and action spaces. Any previously trained agent can also resume training from their last saved point with or without a previous experience buffer. The training can be also initiated or performed directly from Matlab as well.

Deployment of the agent is done by loading the agent into the Matlab workspace and setting the appropriate agent name in the Simulink model environment. Also, the data preprocessing is done for the real and simulated CNC data and the appropriate one is connected to the agent's observation module depending on whether we are performing the experiment in simulation or real hardware. This makes the deployment of the trained agents much easier and helps fast iteration of the RL agent during developments. The setup is also ready to be used in a way where the agent is learning directly from the real CNC for the purpose of fine-tuning or training from scratch.

4.2 Simulation Setup

4.2.1 Simulation Environment and Parameters

We used Simulink in conjunction with Matlab to design the simulation environment as discussed in Chapter 3. Simulink models physical system dynamics using continuous input and output. Simulink uses continuous solvers for the physical systems to accurately model the physics and dynamic interaction between systems. The RL agent, however, outputs action at a fixed rate that we set. We used a variable step solver with support for continuous states that CNC physical system's continuous states are compatible with. We used Matlab's *daessc* solver (DAE solver for Simscape) with a variable step size.

We limit each episode to 20 seconds. Related simulation parameters as well as scalars used for reward scaling are listed in Table 4.3. We also measured the dimensions from the real CNC to match the simulated CNC for all the physical components.

4.2.2 Modeling the CNC in Simulation

The CNC is modeled using the block diagram design paradigm of Simulink with physics-enabled blocks from the Simscape Multibody toolbox. We divide the design

Table 4.3: Simulation parameters.

Parameter	Training	Evaluation
Episode length	20 s	30 s
RL Agent sample time ($\frac{1}{rate}$)	0.2 s	[0.025 0.05 0.1 0.2] s
Observation data type	continuous	
Action data type	continuous	
α	0.2	
β	5	
γ	1	
δ	4	

into two subsystems: 1) the x-y component of the CNC and 2) the z-axis of the CNC.

To keep the dynamics similar to the real CNC, we model the x-y axis using a rectangular joint. A rectangular joint in Simulink uses two prismatic joints primitive to allow linear motion in two axes. A joint in Simulink can be driven by position or force. We used position-based motion and calculate the positions using equations of motion for each step of the simulation. This allows us to closely match the movement of the real CNC since the real CNC is driven using equations of motion as well.

Ideally, the z-axis would be modeled using a winch and rope that closely matches the real CNC. However, modeling the z-axis with a flexible rope is a challenge for any simulation software. The rope is usually modeled using the finite element analysis method, where the rope is modeled as many smaller rigid segments connected by a suitable joint type. While this approach approximates the 'realistic' rope interaction, the implicit state-space becomes large enough that it slows down the simulation significantly. Physical interaction of this rope with other physical components not directly connected to it also slows it down significantly. To overcome this limitation and to maintain reasonable simulation speed, we approximate the rope with 20 rigid links connected with each other using universal joints. Each link is 50 mm long. To simulate the winch mechanism which actuates the rope up/down in the z-axis, we

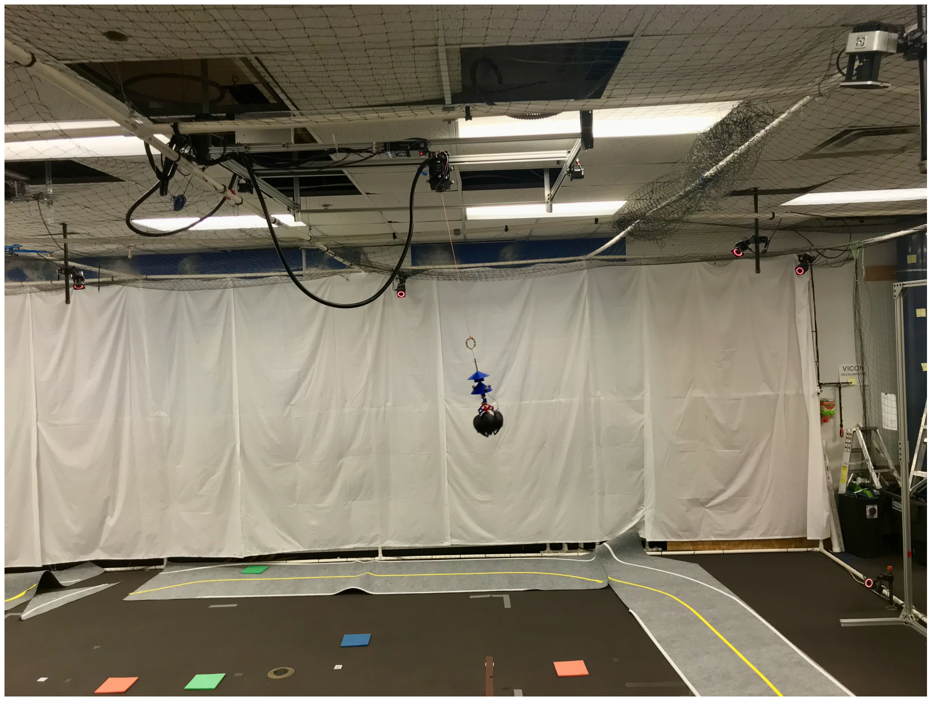


Figure 4.2: The real CNC is mounted on the ceiling. Vicon motion capture systems cameras (in the picture: cameras with red infrared glow) produce the position and orientation at 200 Hz for both CNC Head (x-y plate) and the Leaf sampler.

used a prismatic joint with a variable length cylinder connected to CNC x-y axes using another universal joint.

4.3 Real World Setup

4.3.1 Real Environment and Parameters

We use an open-source two-axis CNC hardware kit from OpenBuilds [93] and added a custom-built Z-axis that uses a winch mechanism to mount the leaf sampler mechanism. This allows the leaf sampler mechanism to move up and down from 0 m to 2 m. We then reinforced the CNC hardware using aluminum brackets and mounted it on the ceiling as shown in Figure 4.2. Table 4.4 shows the specifications of the CNC including motion limits. Specific motion limits used for the real experiments

Table 4.4: Specifications of the CNC from OpenBuilds [93] along with our modifications for the z-axis.

Specification	Value
Machine dimensions	1.5 m $\times$ 1.5 m
Work area	1.3 m $\times$ 1.3 m
Drive System	Belt driven GT2 timing belts
Driving Motor	NEMA 17 Stepper motor
Accuracy	0.10 mm–0.20 mm
Z-Axis	Stepper motor driven Winch-rope
Z-Axis travel	2.5 m
Limit switch	2 per axis
Connection interface	USB, XBee Wireless Module
CNC control system	OpenBuilds BlackBox motion control system
Processor	ATmega328p 8 bit μC

are shown in Table 4.5.

Braided climbing rope is used to reduce the torsional and bending disturbance that may be induced by the leaf sampler when actuating the CNC. This also dampens the pendulum motion that may occur on the leaf sampler mechanism from fast movement in the x-axis and y-axis. We also employ some hardware/software safety mechanisms to prevent the machine from executing commands that would lead to crossing the machine boundary. This is different from constraint enforcement in reinforcement learning [94] which uses a deep learning network to suggest safe alternative actions, whereas we just ignore the command that would violate safety.

Table 4.5: CNC parameters for Real World Experiments

Parameter	Value
Frame of reference	NED
Max Travel in X	1.2 m
Max Travel in Y	1.2 m
Max Travel in Z	2 m
Max velocity	0.5 m/s
Max acceleration	1 m/s^2
Command rate	25 Hz

4.3.2 Sensors

Sensors are crucial for observations that RL agents would use as one of the inputs to generate control signals. Since we are working indoors at this stage of the project, we take advantage of the Vicon Motion Capture system to track the position of the CNC head and the position/orientation of the leaf sampler. The Vicon system outputs data at a maximum frequency of 200 Hz. The motion capture system allows us to precisely track the position of the leaf sampler and the CNC head similar to the simulation environment, thus, reducing the need for domain adaptation at this stage. The Vicon's output rate can be modified or the noise can be added to simulate data available in outdoor environments when experimenting with domain adaptation techniques.

4.3.3 Control interface

The CNC is operated by a low-level controller that runs GRBL open-source CNC firmware. The CNC controller is connected to the ground station computer using wireless XBEE radio that is used for sending commands and getting status updates from the CNC. Traditionally, CNC is run using G-code that consists of a Cartesian position and a target velocity. The CNC would then take the shortest path between the initial and final positions.

Since we can use velocity-based inputs for an LQR controller for UAS indoor and outdoor flights [95], we wrote a ROS node that also takes the velocity input commands from the RL agent. Implementing velocity control reduces the complexity of domain adaptation and would allow seamless sim-to-real transfer from the simulation environment to both CNC and UAS as the agent's inputs and outputs are compatible with both systems.

The ROS node translates the target velocity commands into a series of small movements and speed into an equivalent G-code that is sent to the low-level controller of the CNC which is named the Blackbox motion control system. This allows simultaneous velocity control in all three axes, however, there's a caveat, movement appears jittery as GRBL uses constant acceleration/deceleration with variable speed which causes large changes in the direction of the moment of inertia. This is due to the fact that GRBL plans the path of motion in three segments: speed-up, constant-speed, and slow-down. During these three segments, it accelerates to max speed, cruises at max speed, and then decelerate to come to a stop by the time it reaches the target coordinate. GRBL firmware keeps the movements smooth by implementing a look-ahead of the planner buffer to see if the next movement will be in the same direction. If the next target coordinate is in the same direction, it would skip the slowdown segment for the current target coordinate. Using a high enough acceleration value, we can limit this speed-up and slow-down period. If the commands are sent to GRBL in a way where the next movement command is sent before the current one is finished, this would allow a smoother movement at the cost of 'reaction time' where the CNC output will lag the commanded input by the 'reaction time'.

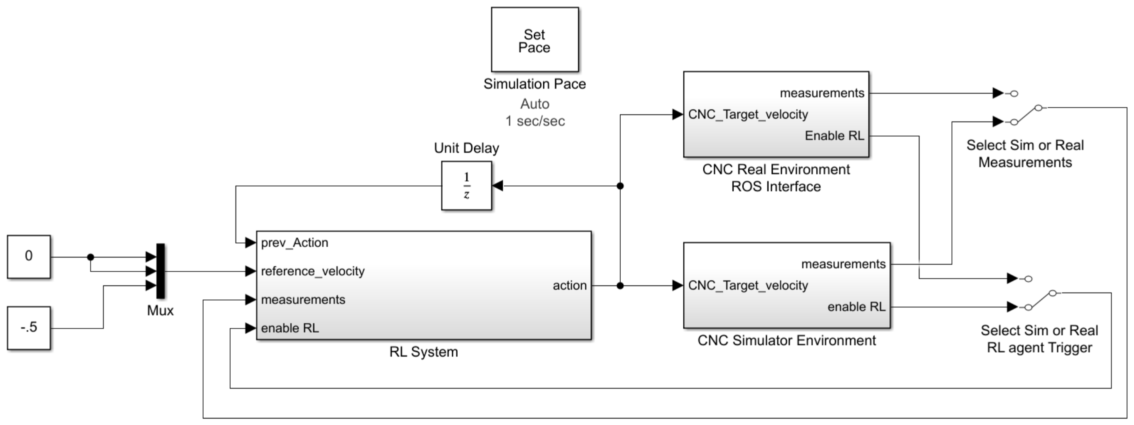


Figure 4.3: A schematic of the sim-to-real process in Simulink.

4.4 Sim-to-Real Experiments

The RL agent is trained in the simulation environment described above. The environment is replaced with the real hardware environment block in Simulink that has the measurement/observations and rewards. This new environment interfaces with the Vicon motion capture systems through ROS networks and enables the hardware-in-the-loop mode to test the RL agent. A simplified block diagram of this sim-to-real transfer process is shown in Figure 4.3

We perform two types of experiments: 1) sim-to-real transfer validation to compare the performance of the RL agent in simulated and real CNC, and 2) simulation calibration experiments to see how the physical parameters of the simulated and real CNC affect the RL agent performance.

4.4.1 Sim-to-Real Transfer Validation Experiments

For sim-to-real transfer validation, we train the RL agent in simulation to move the leaf sampler from a random starting position to a target position. During training, we keep the target position constant for the entire episode, similar to a 'step input'. We then use the trained model to evaluate the performance of the RL agent in simulation and the real world in achieving the target position using a step input.

We also extend the experiments to using a ‘trajectory input‘ of two shapes (circle and square) and compare the RL agent in simulation and the real world. Since we did not explicitly train the model for these trajectory inputs, these experiments show the robustness of the RL agent and the ability to transfer to tasks that are similar in nature.

RL agent was trained at 0.2 s sample rate. This low frequency works well for the CNC but our ultimate goal is to use UAS for leaf sampling, which would require the RL agent to perform well at a sample rate of 0.025 s to 0.03 s or lower as required by an LQR control framework such as Freyja [95]. For this reason, we also validate the RL agent's performance in simulation and real-world at three other sampling rates (0.025 s, 0.05 s, 0.1 s and 0.2 s) that are lower than what the RL agent was trained at. Results of these experiments are presented in Section 5.1.

4.4.2 Calibration Experiments

We do the calibration experiments to see what physical parameters affect the RL agent performance and to gain an understanding of how to match the performance of the RL agent in the simulated and the real CNC.

We first experiment on how changing the dynamics of the CNC affects performance. We change the acceleration parameter for X-Y-Z axes from [$1\,\mathrm{m/s^2}$, $1\,\mathrm{m/s^2}$ and $0.5\,\mathrm{m/s^2}$] to [$2\,\mathrm{m/s^2}$, $2\,\mathrm{m/s^2}$ and $2\,\mathrm{m/s^2}$] and compare the trajectory following performance using the same RL agent. The outcome of these experiments determines whether we can fine-tune the simulated or the real CNC to match the performance that would allow a zero-shot transfer. Failure to match the performance would mean that we would need to tune the RL agent for the real CNC before deployment.

We do a second experiment on calibration by following a dynamic target that we only move after the agent reaches the target position for 60 s. This experiment is aimed to identify the system response time for the specific sample rate, as well as how the RL agent performs over a longer period than the training episodes.

We then check if changing the simulation parameters such as sample rate have any effect on the simulation software performance. This is to identify limitations that we may need to consider when calibrating our simulated and real CNC environments.

Results of these experiments are presented in Section 5.2.

Chapter 5

Results and Discussion

In this chapter, we present the results from our experiments conducted in simulation and real-world that work as validation for the sim-to-real framework. Our work presented here sets up the initial work needed for autonomous aerial leaf sampling using UAS by defining the sim-to-real framework that uses the CNC as an intermediate step and validating the framework by training an RL agent in simulation and transferring it into the real world. As such, the experiments presented here are much simpler than what would be needed for leaf sampling applications where the goals and constraints would be different. Our results here do not meet the expectation in terms of accuracy and precision, however, they still provide valuable insights and information on the capability of our framework.

We first discuss how the framework can perform a zero-shot transfer of the RL agent that was only trained for following static targets to follow dynamic targets and trajectories. This demonstrates that our framework is on the path to reliable sim-to-real leaf sampling applications because the RL agent attempts to follow the trajectory, even when the trajectory type or sample rate changes, instead of taking unpredictable actions.

We then discuss what parameters affect the performance of our sim-to-real framework, so we can learn what works and what we need to change in the next stage of the project.

We present our sim-to-real transfer validation experiment results in Section 5.1 where we show the performance of our RL agent in two types of trajectories, and simulation calibration experiment results in Section 5.2 where we explore experiments designed for performance improvement to simulation and real CNC.

Whenever we refer to the target position in the following discussion, we mean the target position of the leaf sampler as we are working in the context of navigation of the leaf sampler through the canopy.

5.1 Sim-to-Real Transfer Validation

Our eventual goal is to navigate the leaf sampler through the tree canopy, much like [96] but using a model-free RL agent. We do not have any active propulsion on the leaf sampler, we rely on the movement of the UAS in the X-Y axes to position the leaf sampler horizontally and use the winch mechanism to position it vertically in Z-axis. This would allow the UAS to hover over the canopy at a specific Z-height and perform the movements safely to position the leaf sampler without needing to be too close to the canopy. We selected two trajectories for our validation experiments with CNC that are similar to UAS movements that would be performed during the sampling process, would help us characterize the RL agent-based control system responses, and would allow us to verify the robustness of the agent. In training, we only used a static target position that stayed the same for the entire episode. During the evaluation, described in the following sections, we test the RL agent performance with slightly higher difficulty by following step inputs that occurred in all three axes. We then evaluated the RL agent against sinusoidal input which changes at a much faster rate than static or step input does. RL agent was trained at a sampling time of 0.2 s but we evaluate the RL agent at 0.2 s, 0.1 s, 0.05 s and 0.025 s as the UAS requires sample time between 0.05 s to 0.025 s for its control. This experimentation with sample time validates the requirement for the RL agent to be used with UAS.

The required accuracy for trajectory following will vary depending on the task at hand. Several factors including the type of trajectory, the constraints on velocity and position, and the environment may affect this requirement. In the context of the leaf sampling tasks, such as guiding a leaf sampler through the tree branches out of the canopy, we need to be accurate enough so that the leaf sampler does not get stuck in a tree branch. Since our leaf sampler is around 150 mm in diameter, and if the branches are 160 mm apart, our accuracy needs to be better than 10 mm. On the other hand, if we are positioning our leaf sampler to sample the leaves, our system opens up to 200 mm and would only require accuracy better than 100 mm. In the context of the trajectories performed in the thesis, 'good enough' would be when the real trajectory matches the simulated trajectory within a few millimeters while being as accurate as possible compared to the commanded trajectory.

5.1.1 Square Step Input Trajectory

The first trajectory is the step input trajectory that looks similar to a square in the X-Y axis. This trajectory is similar to the waypoint navigation of the UAS. Step input trajectory allows us to characterize the response time, transient response, and steady-state response to our desired input (target position for the leaf sampler). Our results show that the real CNC follows the target position similar to the simulated CNC for training sample rates, but the leaf sampler struggles due to oscillation. Change in sample rate affects both the accuracy and precision as discussed below.

Figure 5.1 shows the output of the square step trajectory for a single episode of 30 s. Each column represents the same trajectory done at different sample rates

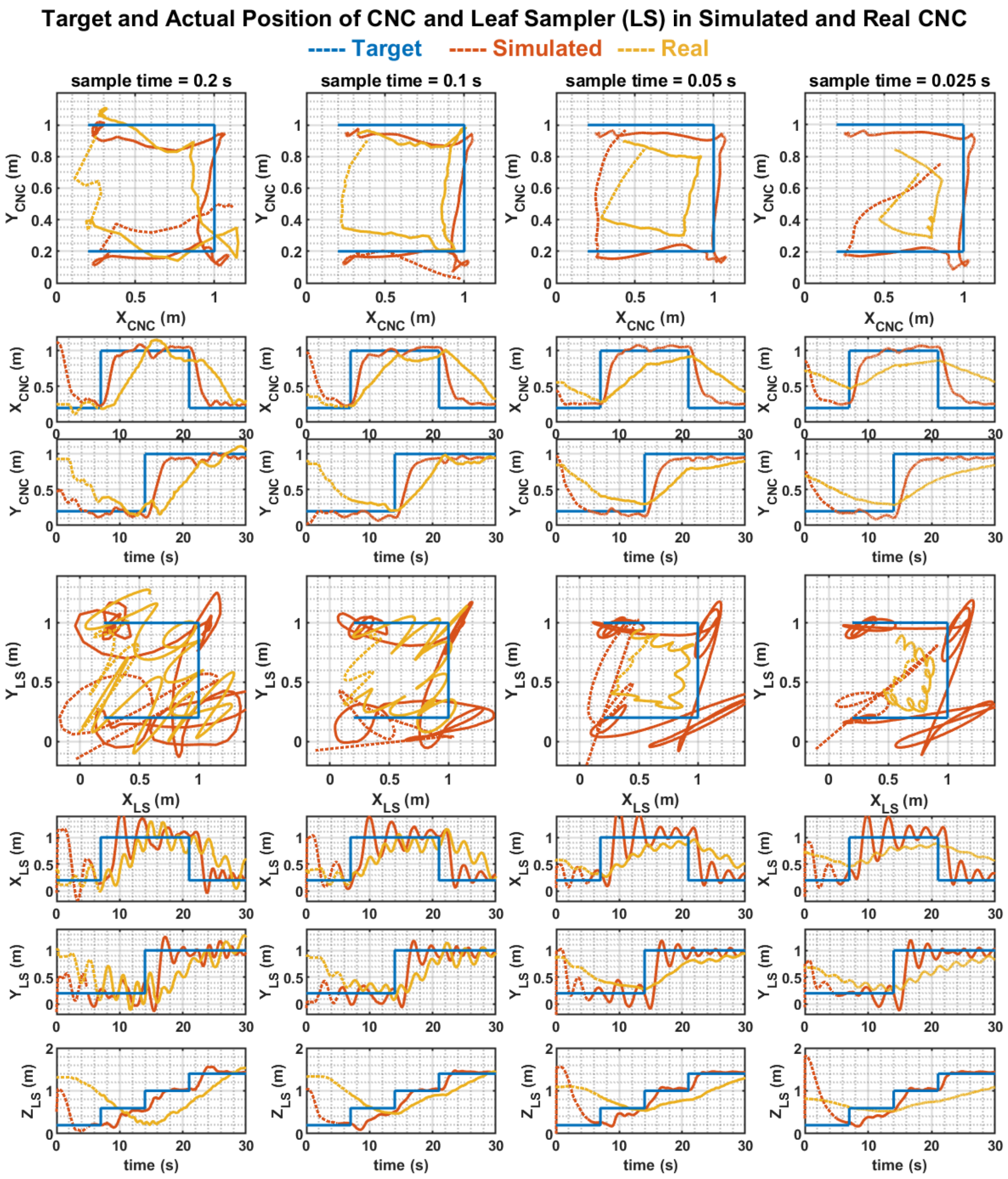


Figure 5.1: Graph of CNC and leaf sampler positions for a 30 s episode in response to a square step input trajectory. Each column represents experiments done at a specific sample rate as indicated at the top. The top three rows show CNC's X-Y position and timeseries, while the bottom four rows show the leaf sampler's X-Y position and X-Y-Z timeseries. The first 5 s is dotted as the CNC starts from random initial positions.

of 0.2 s, 0.1 s, 0.05 s and 0.025 s from left to right respectively. We test these lower sample rates to test the agent's robustness as well as predictability of performance when deployed in the UAS as the UAS control using an LQR controller, such as Freyja [95] that requires lower sample rates. Success or failure of using different sample rates would indicate if we need to fine-tune the RL agent when transferring to the UAS from the CNC. The first three rows represent the X-Y position of the CNC and the corresponding timeseries. The bottom four rows represent the X-Y position and X-Y-Z timeseries of the leaf sampler. First 5 s is plotted with dotted lines as the preparation time for the CNC as it starts from a random initial start position.

We break our result discussion into the CNC X-Y, and the Leaf sampler X-Y-Z in the following text.

5.1.1.1 CNC X-Y

We investigate how close the RL agent performance match in simulated and real CNC. If we look at the first row of X-Y plot in Figure 5.1, we can see that the trajectory is followed closely by CNC X-Y axes for all the sample rates in the simulation. In real CNC however, CNC axes match the target X-Y closely only for the 0.2 s and 0.1 s sample rate, but gets progressively worse for lower sample rates of 0.05 s and 0.025 s. Looking at the second and third rows for X-Y timeseries, we get a slightly different picture of the movement characteristic in each individual axis that explains why the performance is worse at lower sample times. The timeseries show that the response time increases as we move from left to right. At 0.2 s and 0.1 s sample time, the CNC position reaches to steady state in about 7 s compared to about 3 s in simulation, and this time doubles to about 16 s for the 0.05 s sample rate. The CNC X-Y axes fail to reach a steady state before the end of the episode at 0.025 s sample rate. This increase in response time is decreasing accuracy as we see the real squares get progressively smaller. However, the RL agent performs the task with high precision and follows the trajectory faithfully even with the change in dynamics of the CNC at all sampling rates. For leaf sampling tasks using CNC, this increase in response time for the real CNC means that at the current configuration of the real CNC, the target position in the trajectory needs to change slower if we want to operate at a higher sampling frequency. But the UAS with more agile dynamics and lower response time would perform much closer to what we see in the simulation.

One explanation for the increase in response time at a lower sample rate in real CNC is that, at smaller sample times, the movement commands being sent to the CNC are also shorter. As the CNC has a speed-up and slow-down period, this type of short movement reduces the overall speed the CNC is moving at between the target positions. This is artificially increasing the response time of the CNC by making it mostly stay at the less than optimum speed. The software that translates the velocity commands from the RL agent to G-code movement commands would likely need to be updated to account for this. We investigate this issue further in Section 5.2.1 to confirm whether more acceleration would increase accuracy by decreasing the speed-up/slow-down period.

5.1.1.2 Leaf sampler X-Y-Z

For the leaf sampler positions in Figure 5.1, which is our main target for X-Y-Z axes, the X-Y plot looks very noisy due to the leaf sampler overshooting its target position. Time series plots of X-Y axes in rows five and six show that the real sampler follows the trajectory of the simulated one very closely. The response time for the real CNC is half of that in the X-Y axes. The low response time and overshooting response are likely due to how CNC operates. CNC movement in X-Y axes is controlled by a stepper motor with a very precise starting and stopping position. Leaf sampler on

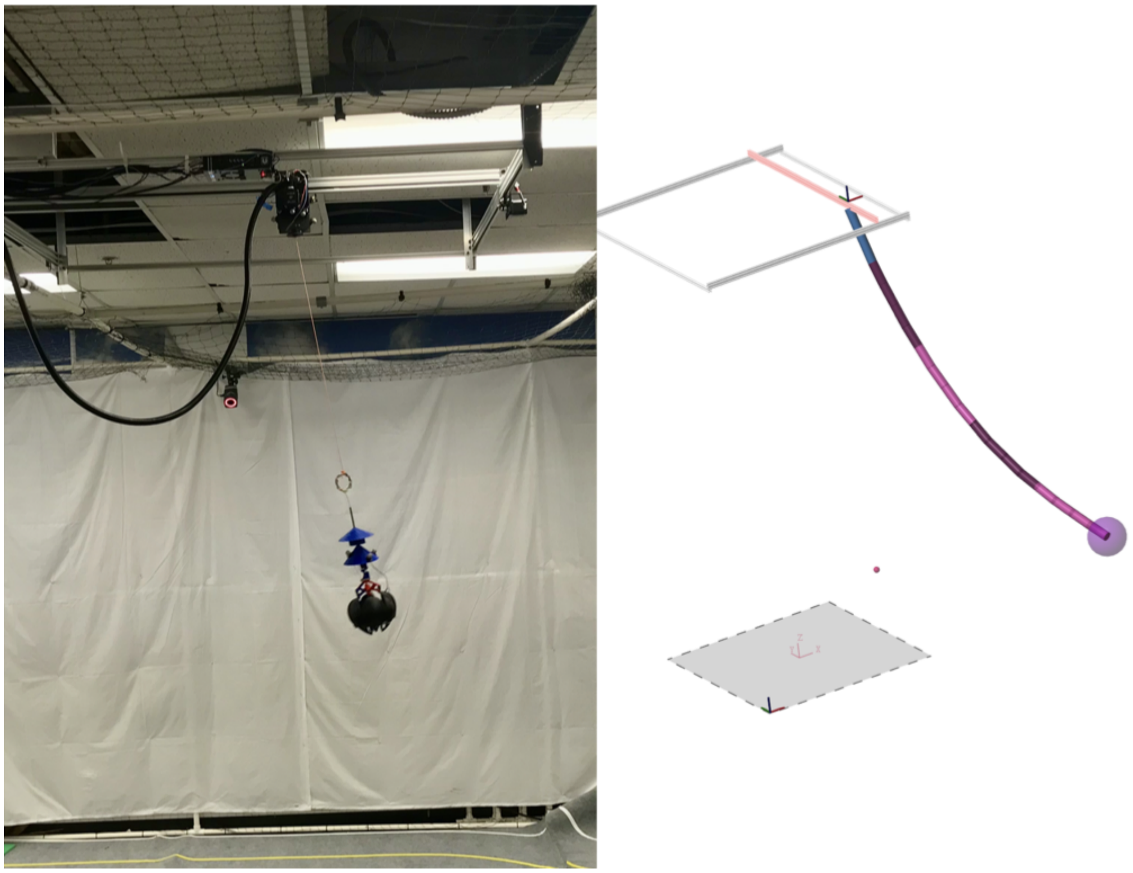


Figure 5.2: On the left, we see the leaf sampler connected to the real CNC using a flexible rope. On the right, we see the leaf sampler connected using a simulated flexible rope. The simulated rope is made by connecting rigid links together using universal joints. The length of the links on the simulated CNC has been exaggerated to show the difference, we use a much higher number of links with much smaller lengths in our simulated experiments to improve simulation accuracy.

the other hand is connected with a flexible rope that gets kinetic energy when the CNC starts moving it as seen in Figure 5.2. This dynamics causes the leaf sampler to swing similar to a pendulum and causes it to overshoot the target when the X-Y axes movement speed is very high. At higher sample rates where CNC is moving at an effectively lower speed, we see the inertia imparted on the leaf sampler is also lower.

Also as the sampling rate increases, the RL agent is able to see the position or overall movement characteristics of the leaf sampler more frequently and take corrective actions to maximize the reward. However, the leaf sampler oscillating more in the real CNC than in the simulation might also be due to the rope design in the simulation not faithfully modeling the real-world rope and might need further tuning. For leaf sampling tasks using CNC at the current configuration, these results indicate higher precision at lower sample times. If the accuracy of the real CNC is improved by tuning the system, we might also see higher accuracy at low sample times. For the UAS leaf sampling tasks, as the agility of the system increases, it will likely perform better trajectory control at lower sample rates.

Leaf sampler performance on the Z-axis matches well between the simulated and the real CNC. However, the response time seems to be much higher as we only see a transient state response for the whole episode. Increased response time compared to the X-Y axes is most likely due to the acceleration for the Z-axis being set to $0.5\,\mathrm{m/s^2}$ instead of $1.0\,\mathrm{m/s^2}$ on the X-Y axes. This design decision was to reduce sudden movement in Z-axis which might pull the leaf sampler in through a branch faster and get itself stuck before the RL agent recognizes it. We investigate the effect of increasing acceleration in the Z-axis at Section 5.2.1. For leaf sampling tasks, these results indicate we need to have a much slower trajectory for this axis in the current configuration.

5.1.2 Circle Trajectory

For the second trajectory, we chose a sinusoidal input with different phases for all three axes. The trajectory looks like a circle in the X-Y axes. While the RL agent was trained on static input, evaluating it on this trajectory does a ‘stress test‘ of the agent to see its robustness to change in the operating environment. The change of rate was also much higher as the target position moved around the circle twice in the same 30 s episode length of the square trajectory. The results for the circle

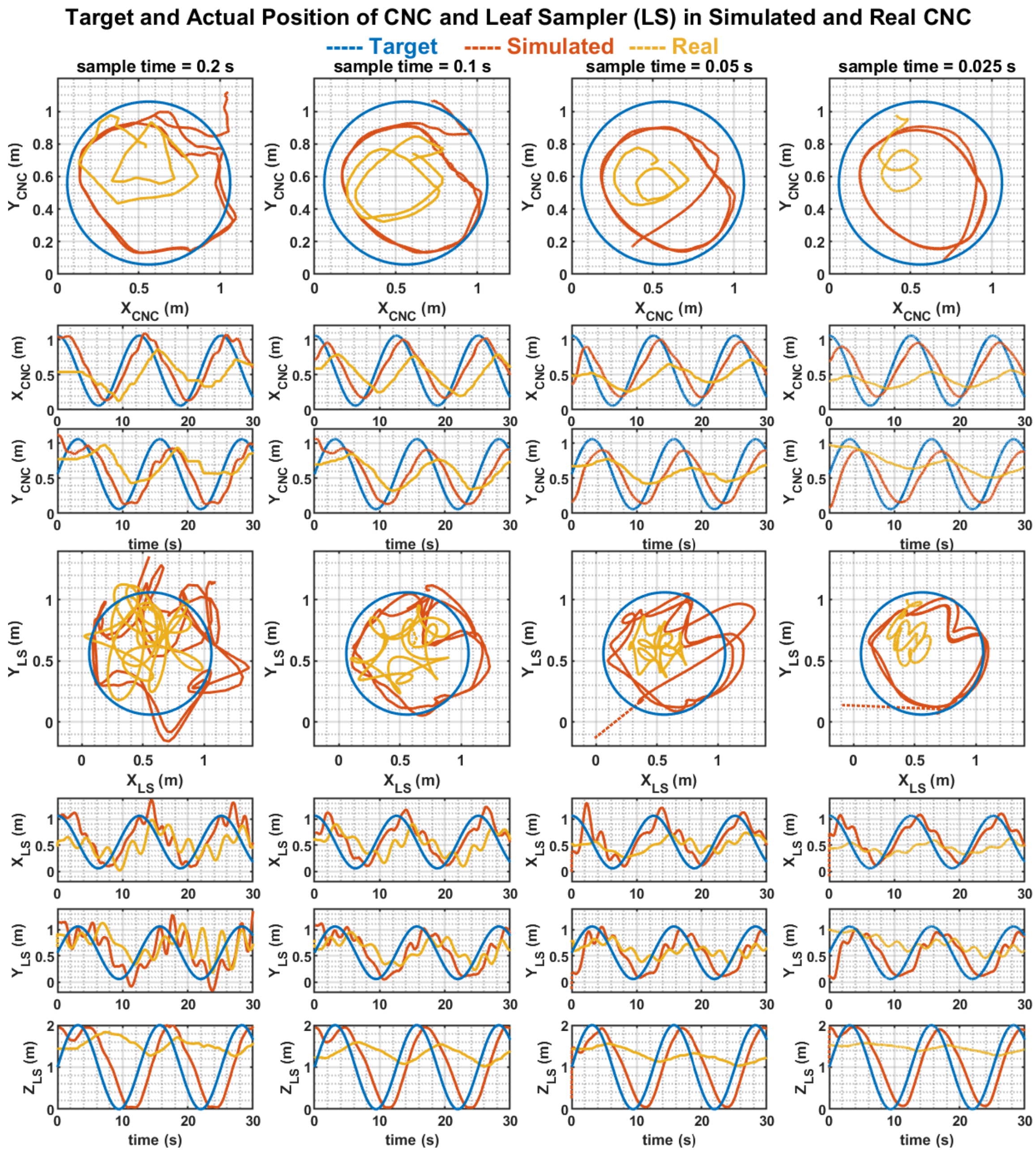


Figure 5.3: Graph of CNC and leaf sampler positions for a 30 s episode in response to a circle trajectory input. Each column represents experiments done at a specific sample rate as indicated at the top. The top three rows show CNC's X-Y position and timeseries, while the bottom four rows show the leaf sampler's X-Y position and X-Y-Z timeseries. Each graph shows two continuous circle trajectories performed in the period of 30 s

trajectory following described below show that the accuracy is worse than that of the step input trajectory but highlight the importance of response time when following a fast-moving target.

Figure 5.3 shows the output of the circle trajectory for a single episode of 30 s. Each column represents the same trajectory done at different sample rates of 0.2 s, 0.1 s, 0.05 s and 0.025 s from left to right respectively. The first three represent the X-Y position of the CNC and the corresponding timeseries. The bottom four rows represent the X-Y position and X-Y-Z timeseries of the leaf sampler.

We break our result discussion into the CNC X-Y, and the Leaf sampler X-Y-Z in the following text.

5.1.2.1 CNC X-Y

We compare the performance of the RL agent for the CNC X-Y positions. RL agent performed similarly as it did with square step input trajectory but with a decrease in accuracy. The increased response time we saw for lower sample times in the step input graphs explains the lack of accuracy: the rate of change in the target position is so high for the circle trajectory that the CNC position never reaches a steady state. This graph also shows that even the simulated CNC would perform at a lower accuracy if the rate of change in the target position is higher than the response time. Looking at the timeseries plots of the X-Y axes in rows three and four we see the higher response time manifests as lower amplitude for sinusoidal inputs. However, the positions still follow the leaf sampler predictably as they did for the step input. For leaf sampling tasks using CNC, these results indicate that dynamic trajectories will need to have a lower rate of change that accounts for the response time for the chosen sample rate. Using the UAS for leaf sampling tasks should be able to use the RL agent without much tuning as the response times would be much lower for UAS.

5.1.2.2 Leaf sampler X-Y-Z

If we look at the leaf sampler graphs, oscillation makes the X-Y plot harder to represent a circle. But looking at the timeseries plots in the bottom three rows, we see the leaf sampler performs pretty much on par with the simulated one for 0.2 s and 0.1 s sample time. The effect of decreased accuracy in CNC x-y axis for 0.05 s and 0.025 s sample time causes a significant decrease in the diameter of the circle (0.4 m while the target is 1 m). The oscillation on the leaf sampler axes still remains throughout the episode but the magnitude of the oscillation is decreased from that of the step input trajectory. Leaf sampler Z-axis illustrates the very high response time causing the amplitude to be only 0.6 m to 0.2 m for a sample time of 0.2 s to 0.025 s while the amplitude of the target position is 2 m. This illustrates that our target positions are highly ambitious and do not properly account for the limitation imposed by the real CNC. For leaf sampling tasks using CNC, we would need to consider reducing the oscillation of the leaf sampler by either tuning the reward function and retraining the RL agent or reworking the way waypoints are being sent to the CNC such that movements can be made smoother.

5.2 Simulation Calibration Results

This section aims to evaluate the performance of the RL agent in controlling the real CNC machine and to understand how to calibrate the simulation system to match the real system dynamics. We conduct three experiments to address the following research questions: 1) How do changes in dynamics such as acceleration settings affect the accuracy and responsiveness of the real CNC machine? 2) How long does it take for the RL agent to reach the target position at low sample times in the real CNC machine? 3) How do these settings influence the Simulink software? The results

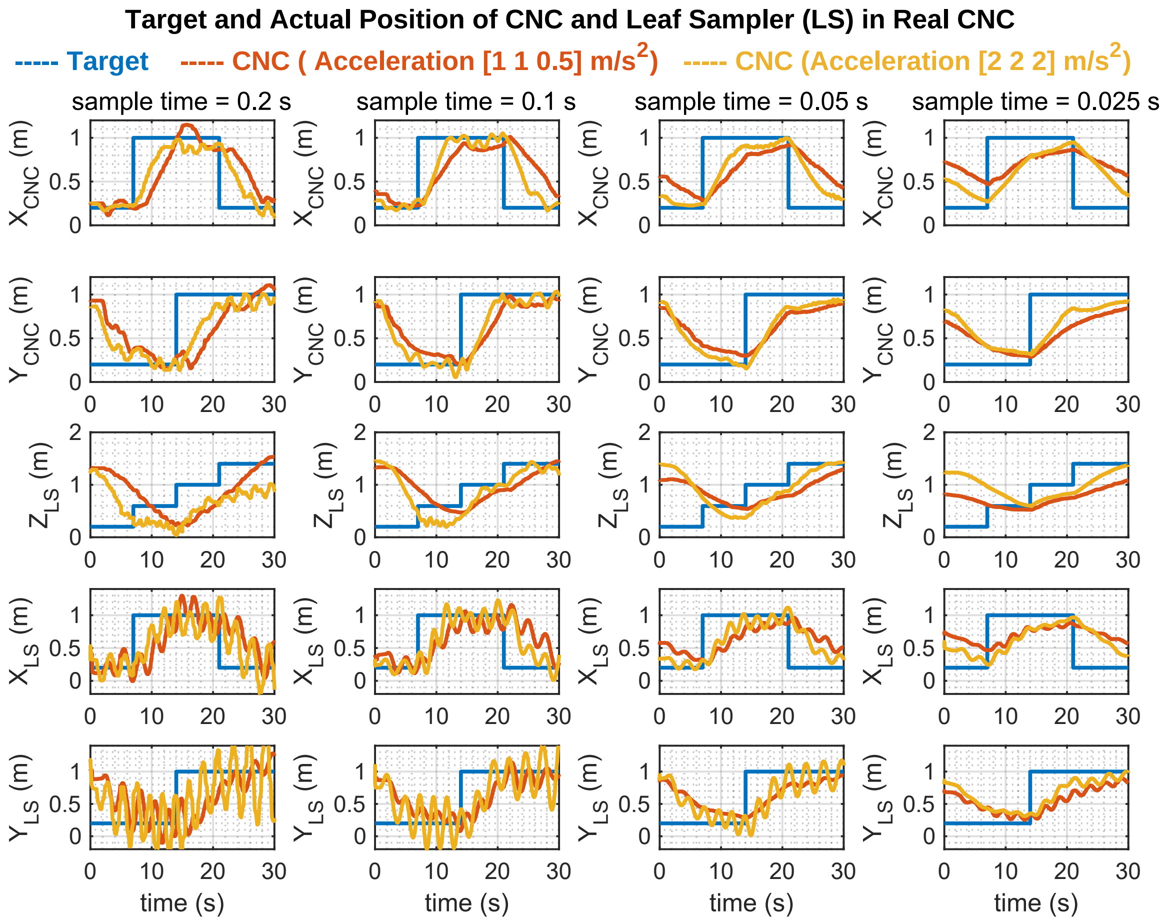


Figure 5.4: Graph of CNC and leaf sampler positions for a 30 s episode in response to a square step input trajectory in Real CNC for regular acceleration and two times acceleration. Each column represents experiments done at a specific sample rate as indicated at the top. The top two rows show CNC's X-Y position timeseries, while the bottom three rows show the leaf sampler's X-Y-Z position timeseries.

of these experiments will help us fine-tune both the real and simulation systems to match their response.

5.2.1 Effect of Change in Dynamics

As we have seen in the previous sections, the performance in real CNC seems to be bottlenecked by the dynamic response of the system. While we could work to match the simulation more to the real world by modifying simulation dynamics, here we

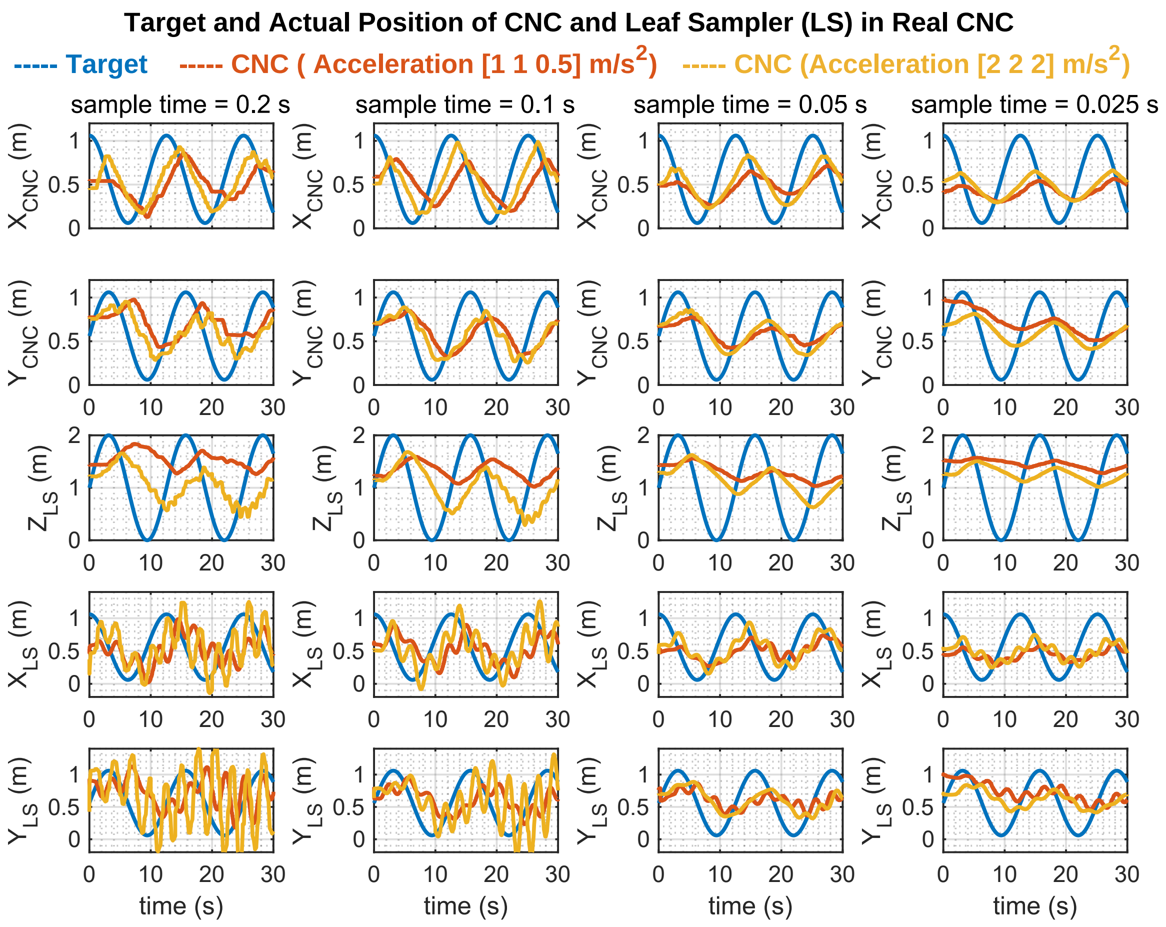


Figure 5.5: Graph of CNC and leaf sampler positions for a 30 s episode in response to a circle trajectory input in Real CNC for regular acceleration and two times acceleration. Each column represents experiments done at a specific sample rate as indicated at the top. The top two rows show CNC's X-Y position timeseries, while the bottom three rows show the leaf sampler's X-Y-Z position timeseries.

investigate what happens if we change the physical dynamics of the real CNC without re-training or tuning the RL agent. More specifically, we change the acceleration parameter for X-Y-Z axes from [1 m/s^2, 1 m/s^2 and 0.5 m/s^2] to [2 m/s^2, 2 m/s^2 and 2 m/s^2] and compare the performance using the same RL agent. Our results show that these changes increase accuracy but have other consequences that we need to consider as described below.

We show the comparison for square step trajectory in Figure 5.4, and the circle trajectory response in Figure 5.5. The first two rows in each figure show the timeseries of the CNC X-Y axis, bottom three rows show the timeseries of the X-Y-Z axis of the leaf sampler. The experiments are done with the same trajectory as presented in the previous section with the same episode length of 30 s.

Increasing the acceleration should reduce the response time to change in input as the speed-up or slow-down period for the CNC motion is reduced. We indeed see the improvement in response time in all three axes as evident from the square step input trajectory in Figure 5.4. However, as expected we see more oscillations on the leaf sampler as the higher acceleration increases the jerk. While the RL agent performs robustly in the CNC X-Y axes, further tuning or training might benefit the RL agent. Adding a domain randomization method that includes changes in acceleration and velocity limits during training might also make the model more robust to these changes.

Increasing the acceleration should also increase the accuracy as the model can reach a steady state faster. This is also verified with the response for sinusoidal input shown in Figure 5.5. We see the amplitude of the response increases across all three axes for all the sample rates. The oscillation in the leaf sampler also seems to get worse than it did for the square step trajectory. This is an expected behavior for the leaf sampler as it is essentially getting imparted with higher moments of inertia.

While increasing the acceleration improved the response, it still does not match the simulated CNC performance yet. We may need to fine-tune other aspects of the CNC dynamics such as speed, timestep of the commands, etc. We may also need to modify the ROS interface software to change how we send the commands to the CNC to optimize the response time as well if we want to match the responses better.

5.2.2 Tracking Dynamic Target

In the previous sections, we saw that real CNC has high precision and low accuracy due to having a very high response time. Specially at low sample times, we only see the transient response and not the steady state. We investigate whether the target position would be achieved if given sufficient time.

Figure 5.6 shows the timeseries graph of the X-Y axes for CNC and the X-Y-Z axes for the leaf sampler. We only do this experiment at the sample rate of 0.5 s for an episode length of 60 s. We experiment with a dynamic target position that we move only after the CNC achieves that target position. Dynamic target is tracked using Vicon motion capture systems and connected to Simulink through ROS.

As expected, we see the real CNC can indeed reach the target positions in all three axes. We do this experiment on four times lower sample time and two times higher episode length than what the RL agent was trained at. This further confirms the robustness of the RL agent and confirms the performance in simulation indeed predicts the performance in the real system. As long as the simulation and real CNC response times are comparable, we should see better accuracy overall.

5.2.3 Performance of the Simulation Software

We used a Simulink block that introduces a delay to keep the simulation time as close as possible to the real-time clock. This is important when controlling the real-world system from Simulink, we need to have a predictable step size to ensure the frequency of control actions stays within the range of the real system. The real-time pacing block outputs pacing error as the deviation from real-time. We investigate how changing the sample times affect the Simulink software.

We show the pacing error for the circular trajectory and square step for all the

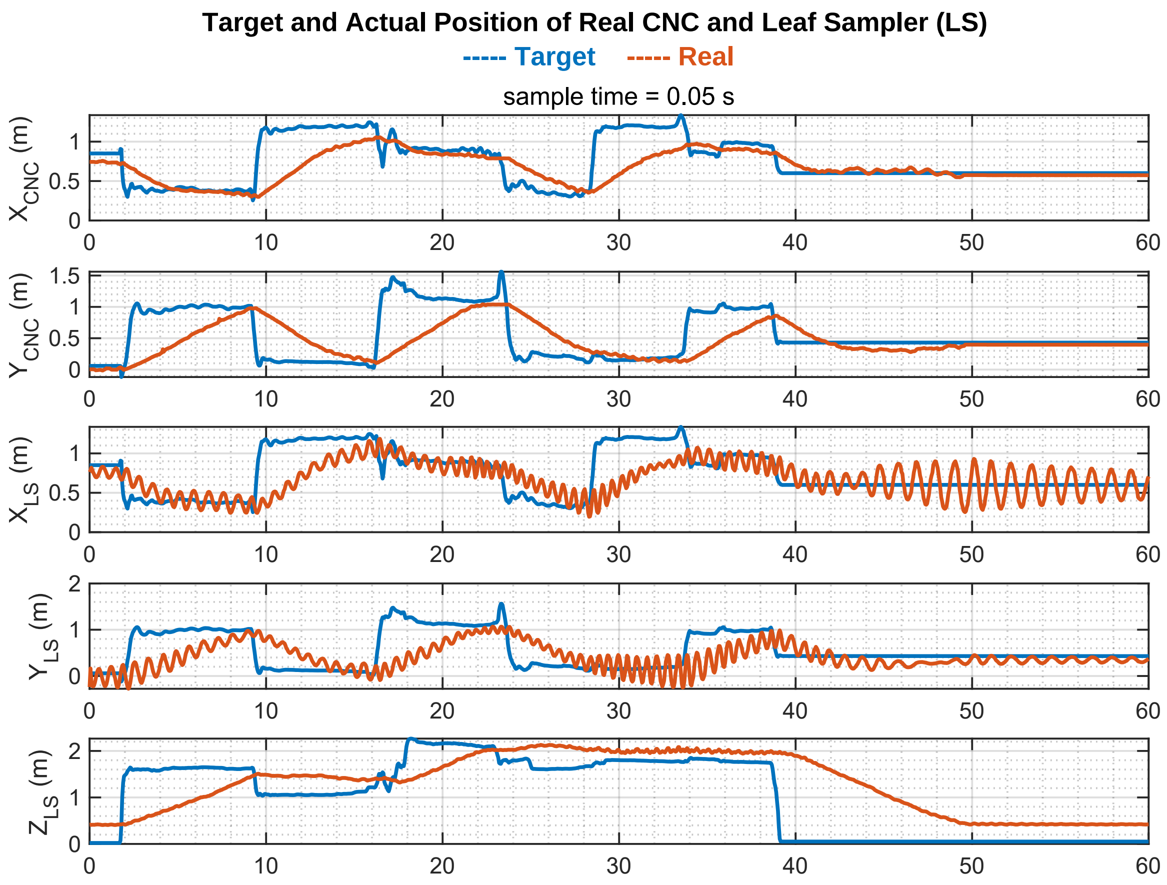


Figure 5.6: Timeseries of Real CNC following a dynamic target tracked by Vicon motion captures through ROS interface. The top two rows show the X-Y position timeseries of the CNC and the bottom three row shows the X-Y-Z position timeseries of the leaf sampler.

sample times in Figure 5.7. The first and second column shows the pacing error related to the circle trajectory and square step respectively. The sample time for each plot is indicated at the top of each subplot.

We see the performance of the real-time pacing block stays the same for all the sample times in the real system. This is important for control tasks. This performance however drops significantly when it comes to the simulation environment at sample rates of 0.5 s and 0.025 s. The pacing error is 10 s and 40 s for 0.5 s and 0.025 s respectively. This would be unacceptable when controlling real-time systems. This slowdown does not affect the training since we only use real-time pacing blocks during

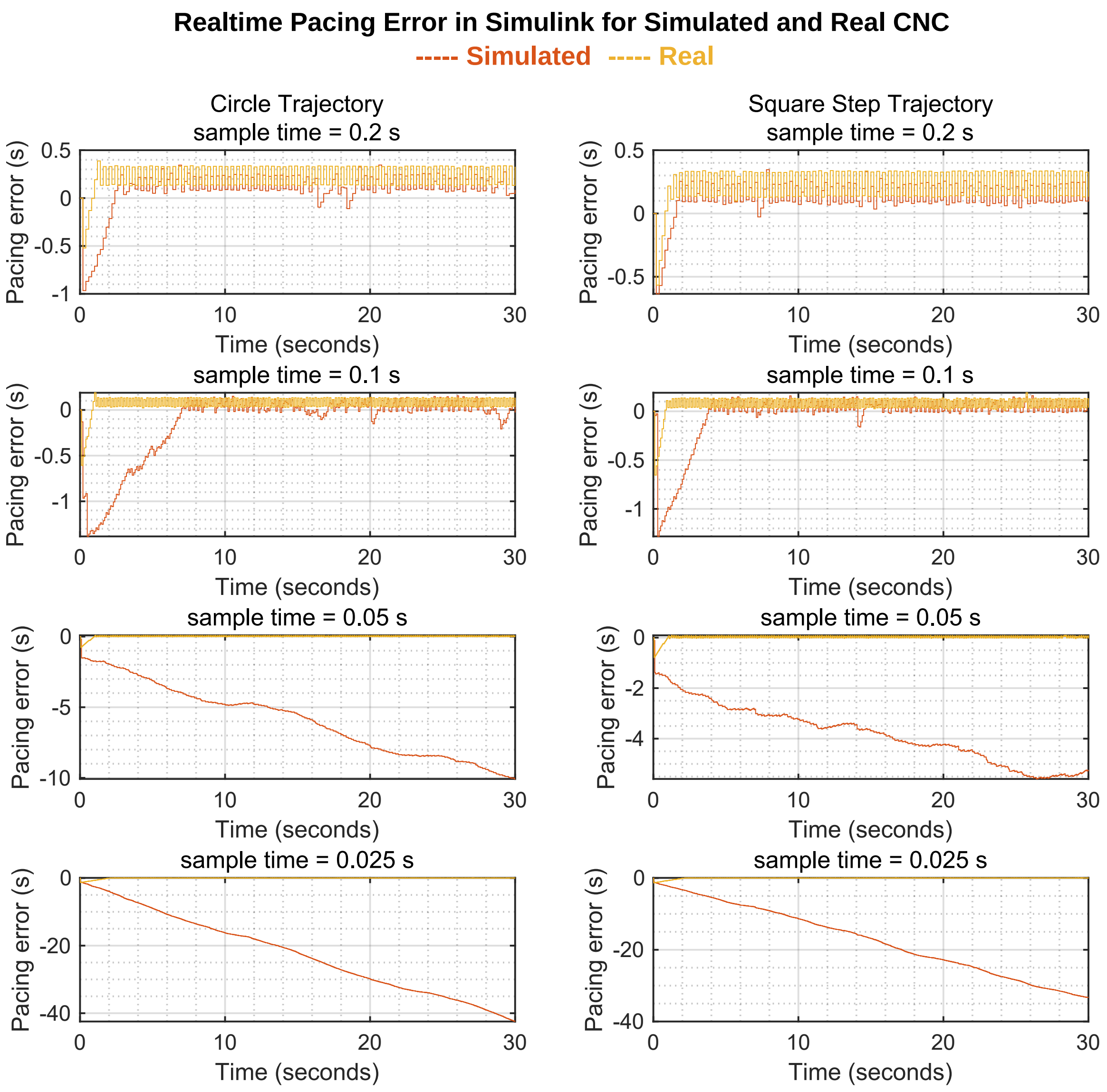


Figure 5.7: Timeseries of all the pacing errors in real and simulation at different sample times. The first and second column shows pacing error for the circle and square step trajectory respectively. Sample time for each subplot is indicated at the top of each subplot

testing or evaluation or when we are controlling the real CNC. As such, care must be taken not to have the simulated system activated when the Simulink is controlling the real system.

Chapter 6

Conclusion and Future work

We presented an indoor sim-to-real platform using CNC for training and testing reinforcement learning algorithms for leaf sampling tasks. This thesis provides a good first step towards using UAS for autonomous aerial leaf sampling using sim-to-real reinforcement learning. Autonomous aerial leaf sampling would allow the collection and new analytics of unreachable leaf samples that are inside the tree canopy without needing multiple skilled operators.

We contributed a sim-to-real framework for training and deploying an RL agent in a simulated CNC. We tested the RL agent in real CNC for a square step input and circle trajectory input. Our evaluation shows RL agent performance matches well between the simulated and the real CNC for the training sample times of 0.2 s. At lower sample times, the RL agent performance is still predictable but shows a lower accuracy in real CNC due to the increase in response time at lower sample times.

We also contributed a hardware and software interface that allows fast deployment of simulation-trained agents without any additional tuning required. We used observations that are available in both CNC and UAS with similar control inputs so that the framework can be easily extended to controlling the UAS in the future.

We also noticed some limitations in our work, such as the slow response time in the real CNC, the slow down of the simulation software for lower sample rates, and other simulated environment limitations, etc., that we discuss next.

6.1 Limitations and Future Work

We discuss some of the limitations and possible future work to address those.

6.1.1 Response Time Limitation of the real CNC

In the simulated CNC we see the response time stays consistent regardless of the sample times. However, in the real CNC, the response times increase when the sample time is decreased. This causes a reduction in accuracy. One possible reason for that is the CNC plans each motion with a speed-up, constant speed, and slow-down period. This is to reduce the sudden start and stop that can introduce jerk or cause the stepper motor to lose steps. The CNC uses a planner buffer that can store up to 15 movements and can plan the motions to optimize the movements. However, if the two consecutive motion is not in the same direction, we can not avoid the slow-down for the current movement and speed-up for the next movement. When looking at the time series response trajectory in the X-Y-Z axes, the outputs look similar to a PID control. However, if we investigate the RL agent action output (not presented here), we see the RL agent switches back and forth between $+ve$ and $-ve$ velocity frequently similar to the 'bang bang' control method. While bang-bang control is simple and faster, it is prone to oscillation and error as we see in our case.

One possible future extension would be to experiment with a reward function to encourage the RL agent action output to be more like a PID control as well which would make the motion smoother by reducing the magnitude of output velocity instead of switching to the opposite state.

6.1.2 Simulation speed Limitation of the simulation software

We chose Matlab as it provides seamless integration between the RL agent, simulator, training, real CNC, and ROS. However, we noticed that the accuracy of the simulation comes at the cost of a high time penalty for simulating each time step. We use variable timestep simulation in Matlab to speed up simulations as Matlab can optimally simulate physical systems with continuous states at a pretty high step size. Lowering the sample time on the RL agent, however, forces the simulator to take steps that are less than or equal to the sample time. This means that between 0.2 s sample rate, and 0.025 s sample rate- there are at least 8 times more steps required for Matlab to simulate for the same 30-second period. This slows down the simulation environment significantly where training the model would be unrealistic for such low sample times as the reinforcement learning usually takes a lot of episodes to learn, and tuning the reward becomes much harder due to waiting times between each training period. One way to improve this is by training the RL agent using parallel training but the performance diminishes after a certain number of agents.

One possible future extension is the use of an external rendering engine such as Unreal Engine or Unity Engine for the simulation of physical bodies. Since these are game engines optimized for rendering speed, the physics accuracy might not be the same as Simulink but rather it is realistic enough. Alternatively, we can explore the use of the Gazebo simulation software interfaced with Simulink through ROS.

Some particle-based simulation exists that limit the particles to a cartesian joint while allowing no rotation in any axis. Simulation of deformable objects such as the rope or tree branches can be designed using this method called position-based dynamics (PBD) [97], and the improved version Extended Position Based Dynamics (XPBD) simulation [98]. This technique is capable of simulating deformable soft bodies, as well rigid bodies at a very large scale [99]. One issue with this approach is that it does not simulate actual physics, but rather a close visual resemblance of it. Domain adaption technique or fine-tuning of the RL agent for the real system might be required when using these techniques.

6.1.3 Actuation of the Leaf Sampler

In the scope of this thesis, we move the leaf sampler using the movement of the CNC. While this worked for validation of the leaf sampling framework, we noticed precise control of the leaf sampler is harder with this approach. The presence of disturbances such as wind would make the oscillation we see on the leaf sampler even worse.

One possible extension to the leaf sampler would be to use active propulsion on the leaf sampler to counter this oscillation but also to be able to position the leaf sampler more precisely. Success has been shown for navigating the tree canopy using a propelled mechanism [96] connected through the rope. The RL agent can be easily adapted to either have increased action space or use in a multi-agent setup where two RL agents collaborate in controlling the CNC, and the leaf sampler.

6.2 Future Extensions

The next step of the project would be to evaluate the RL agent performance on the UAS. The simulated environment would need to be extended to have trees/branches, and model contact forces/collision dynamics for interaction with the leaf sampler for various leaf sampling tasks. The real CNC can also be used for hardware-in-the-loop training of the RL agent instead of simulation if certain aspects of the leaf sampler become harder to model in simulation.

Bibliography

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[2] Guillaume Charron, Thomas Robichaud-Courteau, Hughes La Vigne, Samantha Weintraub, Andy Hill, Douglas Justice, Nicolas Bélanger, and Alexis Lussier Desbiens. The DeLeaves: A UAV device for efficient tree canopy sampling. *Journal of Unmanned Vehicle Systems*, 8(3):245–264, September 2020.

[3] James Kutia. *Aerial Manipulation for Canopy Sampling.* Thesis, ResearchSpace@Auckland, 2019.

[4] Wei Zhu, Xian Guo, Dai Owaki, Kyo Kutsuzawa, and Mitsuhiro Hayashibe. A survey of sim-to-real transfer techniques applied to reinforcement learning for bioinspired robots. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[5] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, December 2020.

[6] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[7] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv:1912.06680 [cs, stat]*, December 2019.

[8] Jan Matas, Stephen James, and Andrew J. Davison. Sim-to-Real Reinforcement Learning for Deformable Object Manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, pages 734–743. PMLR, October 2018.

[9] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. *arXiv:1808.00177 [cs, stat]*, January 2019.

[10] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell,

Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. *arXiv:1910.07113 [cs, stat]*, October 2019.

[11] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-To-Real via Sim-To-Sim: Data-Efficient Robotic Grasping via Randomized-To-Canonical Adaptation Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[12] Fan Shi, Timon Homberger, Joonho Lee, Takahiro Miki, Moju Zhao, Farbod Farshidian, Kei Okada, Masayuki Inaba, and Marco Hutter. Circus ANYmal: A Quadruped Learning Dexterous Manipulation with Its Limbs. *arXiv:2011.08811 [cs]*, December 2020.

[13] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *arXiv:1804.10332 [cs]*, May 2018.

[14] Bangyu Qin, Yue Gao, and Yi Bai. Sim-to-real: Six-legged Robot Control with Deep Reinforcement Learning and Curriculum Learning. In *2019 4th International Conference on Robotics and Automation Engineering (ICRAE)*, pages 1–5, November 2019.

[15] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Automated aerial suspended cargo delivery through reinforcement learning. *Artificial Intelligence*, 247:381–398, June 2017.

[16] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a Quadrotor With Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, October 2017.

[17] Riccardo Polvara, Massimiliano Patacchiola, Sanjay Sharma, Jian Wan, Andrew Manning, Robert Sutton, and Angelo Cangelosi. Autonomous Quadrotor Landing using Deep Reinforcement Learning. *arXiv:1709.03339 [cs]*, February 2018.

[18] Pedro Lucas Franca Albuquerque. Domain Adaptation in Unmanned Aerial Vehicles Landing using Reinforcement Learning. Master's thesis, University of Nebraska-Lincoln, 2019.

[19] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*, October 2018.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[21] Mathworks. Reinforcement Learning for Control Systems Applications - MATLAB & Simulink, 2021.

[22] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

[24] G. A. Rummery and M. Niranjan. On-Line Q-Learning Using Connectionist Systems. Technical report, 1994.

[25] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[26] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*, June 2016.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1889–1897. PMLR, June 2015.

[29] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

[30] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905 [cs, stat]*, January 2019.

[31] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2019.

[32] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[33] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation - GECCO '10*, page 119, Portland, Oregon, USA, 2010. ACM Press.

[34] Manuel Kaspar, Juan David Munoz Osorio, and Jürgen Bock. Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization. *arXiv:2002.11635 [cs]*, February 2020.

[35] Fabio Muratore, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger, and Jan Peters. Robot Learning from Randomized Simulations: A Review. *arXiv:2111.00956 [cs]*, November 2021.

[36] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. *arXiv:1709.07857 [cs]*, September 2017.

[37] Irina Higgins, Arka Pal, Andrei A. Rusu, Loic Matthey, Christopher P. Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. *arXiv:1707.08475 [cs, stat]*, June 2018.

[38] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-To-Image Translation With Conditional Adversarial Networks. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition, pages 1125–1134, 2017.

[39] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional Domain Adaptation for Sim2Real Transfer of Embodied Navigation Agents. *IEEE Robotics and Automation Letters*, 6(2):2634–2641, April 2021.

[40] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv:1703.10593 [cs]*, August 2020.

[41] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real Single-Image Flight without a Single Real Image. *arXiv:1611.04201 [cs]*, June 2017.

[42] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *arXiv:1703.06907 [cs]*, March 2017.

[43] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979, May 2019.

[44] Eugene Valassakis, Zihan Ding, and Edward Johns. Crossing The Gap: A Deep Dive into Zero-Shot Sim-to-Real Transfer for Dynamics. *arXiv:2008.06686 [cs]*, August 2020.

[45] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2016.

[46] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, October 2012.

[47] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, May 2018.

[48] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-Real Transfer with Neural-Augmented Robot Simulation. In *Proceedings of The 2nd Conference on Robot Learning*, pages 817–828. PMLR, October 2018.

[49] Unity Technologies. Unity Real-Time Development Platform | 3D, 2D VR & AR Engine, 2021.

[50] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, September 2004.

[51] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, page 5. Kobe, Japan, 2009.

[52] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorS—A modular gazebo MAV simulator framework. *Studies in Computational Intelligence*, 625:595–625, 2016.

[53] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *arXiv:1705.05065 [cs]*, July 2017.

[54] Ajay Shankar, Sebastian Elbaum, and Carrick Detweiler. In-air exchange of small payloads between multirotor aerial systems. In *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 511–523. Springer, 2020.

[55] Daniel A Rico, Francisco Muñoz-Arriola, and Carrick Detweiler. Trajectory selection for power-over-tether atmospheric sensing uas. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2321–2328. IEEE, 2021.

[56] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *arXiv:1603.02199 [cs]*, August 2016.

[57] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[58] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *arXiv:1806.10293 [cs, stat]*, November 2018.

[59] Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-Real Robot Learning from Pixels with Progressive Nets. *arXiv:1610.04286 [cs]*, May 2018.

[60] Wenshuai Zhao, Jorge Peña Queralta, Li Qingqing, and Tomi Westerlund. Towards Closing the Sim-to-Real Gap in Collaborative Multi-Robot Deep Reinforcement Learning. *arXiv:2008.07875 [cs, stat]*, August 2020.

[61] Lei M. Zhang, Matthias Plappert, and Wojciech Zaremba. Predicting Sim-to-Real Transfer with Probabilistic Dynamics Models. *arXiv:2009.12864 [cs]*, September 2020.

[62] Geoffrey G. Parker, Alan P. Smith, and Kevin P. Hogan. Access to the Upper Forest Canopy with a Large Tower Crane. *BioScience*, 42(9):664–670, 1992.

[63] Nigel E. Stork. Australian tropical forest canopy crane: New tools for new frontiers. *Austral Ecology*, 32(1):4–9, 2007.

[64] Martin G. Barker and Michelle A. Pinard. Forest canopy research: Sampling problems, and some solutions. In K. Eduard Linsenmair, A. J. Davis, B. Fiala, and M. R. Speight, editors, *Tropical Forest Canopies: Ecology and Management*, volume 69, pages 23–38. Springer Netherlands, Dordrecht, 2001.

[65] Thayer Walker. Last Tree Standing, November 2016.

[66] F. T Portlock. *Field Guide to Collecting Cones of British Columbia Conifers.* 1996.

[67] Maurice Leponce. Investigating the Biodiversity of Soil and Canopy Arthropods: Canopy access, 2021.

[68] Will Apse. Rainforest Trees for Beginners, 2019.

[69] Florian Käslin, Thomas Baur, Philip Meier, Patrick Koller, Nina Buchmann, Petra D'Odorico, and Werner Eugster. Novel Twig Sampling Method by Unmanned Aerial Vehicle (UAV). *Frontiers in Forests and Global Change*, 1:2, October 2018.

[70] Canopy Crane Papua New Guinea. KAKOBA Canopy Crane | Papua New Guinea, 2021.

[71] Francis Hallé. Francis Hallé and the Canopy Raft, December 2017.

[72] Tawanda W. Gara, Roshanak Darvishzadeh, Andrew K. Skidmore, and Tiejun Wang. Impact of Vertical Canopy Position on Leaf Spectral Properties and Traits across Multiple Species. *Remote Sensing*, 10(2):346, February 2018.

[73] Anna K. Schweiger, Alexis Lussier Desbiens, Guillaume Charron, Hughes La Vigne, and Etienne Laliberté. Foliar sampling with an unmanned aerial system (UAS) reveals spectral and functional trait differences within tree crowns. *Canadian Journal of Forest Research*, June 2020.

[74] Hughes la Vigne, Guillaume Charron, Samuel Hovington, and Alexis Lussier Desbiens. Assisted Canopy Sampling Using Unmanned Aerial Vehicles (UAVs). In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1642–1647, June 2021.

[75] Christopher Alexander Maximilian Busch, Karl A. Stol, and Wannes van der Mark. Dynamic tree branch tracking for aerial canopy sampling using stereo vision. *Computers and Electronics in Agriculture*, 182:106007, March 2021.

[76] Daniel Orol, Jnaneshwar Das, Lukas Vacek, Isabella Orr, Mathews Paret, Camillo. J. Taylor, and Vijay Kumar. An aerial phytobiopsy system: Design,

evaluation, and lessons learned. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 188–195, Miami, FL, USA, June 2017. IEEE.

[77] Domen Finžgar, Marko Bajc, Jernej Brezovar, Andraž Kladnik, Rok Capuder, and Hojka Kraigher. Development of a patented unmanned aerial vehicle based system for tree canopy sampling. *Folia biologica et geologica*, 57(2):35–39, 2016.

[78] Bailey Wu, Bryce Marshall, Colin Andrews, Max Davitt, and James Fanchiang. Sampler Drone for Plant Physiology and Tissue Research, 2018.

[79] Patricia Ventura Diaz and Steven Yoon. High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles. In *2018 AIAA Aerospace Sciences Meeting*, page 1266, Kissimmee, Florida, 2018. American Institute of Aeronautics and Astronautics.

[80] James R. Kutia, Karl A. Stol, and Weiliang Xu. Aerial Manipulator Interactions With Trees for Canopy Sampling. *IEEE/ASME Transactions on Mechatronics*, 23(4):1740–1749, August 2018.

[81] Changliang Xu, Zhong Yang, Yuhong Jiang, Qiuyan Zhang, Hao Xu, and Xiangrong Xu. The Design and Control of a Double-saw Cutter on the Aerial Trees-pruning Robot. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2095–2100, December 2018.

[82] Javier Molina and S. Hirai. Aerial pruning mechanism, initial real environment test. *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2017.

[83] Basaran Bahadir Kocer, Boon Ho, Xuanhao Zhu, Peter Zheng, André Farinha, Feng Xiao, Brett Stephens, Fabian Wiesemüller, Lachlan Orr, and Mirko Ko-

vac. Forest Drones for Environmental Sensing and Nature Conservation. In *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIR-PHARO)*, pages 1–8, October 2021.

[84] Adam Savage's Tested. Jamie Hyneman's 'Arborist' Quadcopter Test, April 2015.

[85] Chris Colin. How Mythbuster Jamie Hyneman Hacked a Drone to Trim His Trees, April 2017.

[86] Samantha Weintraub. TOS PROTOCOL AND PROCEDURE: CANOPY FOLIAGE SAMPLING. Technical report, National Ecological Observatory Network (NEON), 2019.

[87] Forest Pathology and Mycology Lab, UC Berkeley. Sampler Drones for Forestry Research, 2015.

[88] Tawanda W. Gara, Andrew K. Skidmore, Roshanak Darvishzadeh, and Tiejun Wang. Leaf to canopy upscaling approach affects the estimation of canopy traits. *GIScience & Remote Sensing*, 56(4):554–575, May 2019.

[89] Tawanda W. Gara, Roshanak Darvishzadeh, Andrew K. Skidmore, Tiejun Wang, and Marco Heurich. Evaluating the performance of PROSPECT in the retrieval of leaf traits across canopy throughout the growing season. *International Journal of Applied Earth Observation and Geoinformation*, 83:101919, November 2019.

[90] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed Distributional Deterministic Policy Gradients. *arXiv:1804.08617 [cs, stat]*, April 2018.

[91] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[92] Mathworks. Design and Train Agent Using Reinforcement Learning Designer - MATLAB & Simulink, 2021.

[93] OpenBuilds. OpenBuilds ACRO CNC 1515 60" x 60", 2021.

[94] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe Exploration in Continuous Action Spaces. *arXiv:1801.08757 [cs]*, January 2018.

[95] Ajay Shankar, Sebastian Elbaum, and Carrick Detweiler. Freyja: A full multirotor system for agile & precise outdoor flights. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 217–223. IEEE, 2021.

[96] Steffen Kirchgeorg and Stefano Mintchev. Multimodal aerial-tethered robot for tree canopy exploration. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6080–6086. IEEE, 2022.

[97] Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber. Position-based simulation of continuous materials. *Computers & Graphics*, 44:1–10, 2014.

[98] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. XPBD: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pages 49–54, Burlingame California, October 2016. ACM.

[99] Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. Detailed Rigid Body Simulation with Extended Position Based Dynamics. *Computer Graphics Forum*, 39(8):101–112, December 2020.