NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTOMATIC ROUTING OF SUBMARINE ELECTRICAL CABLES USING MACHINE LEARNING

by

Katelyn M. Damaso

June 2023

Thesis Advisor: Mark Karpenko
Second Reader: Jessica L. Herman

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2023	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
AUTOMATIC ROUTING OF SUBMARINE ELECTRICAL CABLES USING MACHINE LEARNING

5. FUNDING NUMBERS

6. AUTHOR(S) Katelyn M. Damaso

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

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

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

During the design process of a submarine, there are hundreds of cables that have a large diameter and a corresponding large minimum bend radius. These cables must be modeled using 3D CAD to ensure they are not bent past the minimum bend radius and to reduce the space consumed. This process is labor-intensive and sub-optimal. The objective of this thesis is to determine the feasibility of using the model-based reinforcement learning algorithm MuZero to automatically route the cables. The specific implementation of MuZero used is MuZero-General in conjunction with a custom-built Gymnasium environment. As a proof-of-concept, a 2D model successfully routed a representation of a cable from a start location to an end location after completing training. An object, a representation of an already routed cable, was added into the model. The goal was for the agent to find a path from start to finish while avoiding the obstacle. This single obstacle significantly increased the training requirements. In this model, the agent completed a route from start to finish while avoiding obstacles. However, its path was not optimal and will require additional training, which is left for future work. Overall, it was determined that it is feasible to apply reinforcement learning techniques to route cables, although challenging. In future work, the model can be expanded, making it more representative of a real-world design scenario.

14. SUBJECT TERMS
reinforcement learning, machine learning, automated design, MuZero, Gymnasium, Python 3

15. NUMBER OF PAGES
65

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

AUTOMATIC ROUTING OF SUBMARINE ELECTRICAL CABLES USING MACHINE LEARNING

Katelyn M. Damaso
Ensign, United States Navy
BS, United States Naval Academy, 2022

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2023

Approved by: Mark Karpenko
Advisor

Jessica L. Herman
Second Reader

Brian S. Bingham
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

During the design process of a submarine, there are hundreds of cables that have a large diameter and a corresponding large minimum bend radius. These cables must be modeled using 3D CAD to ensure they are not bent past the minimum bend radius and to reduce the space consumed. This process is labor-intensive and sub-optimal. The objective of this thesis is to determine the feasibility of using the model-based reinforcement learning algorithm MuZero to automatically route the cables. The specific implementation of MuZero used is MuZero-General in conjunction with a custom-built Gymnasium environment. As a proof-of-concept, a 2D model successfully routed a representation of a cable from a start location to an end location after completing training. An object, a representation of an already routed cable, was added into the model. The goal was for the agent to find a path from start to finish while avoiding the obstacle. This single obstacle significantly increased the training requirements and created challenges in designing the reward function. In this model, the agent completed a route from start to finish while avoiding obstacles. However, its path was not optimal and will require additional training, which is left for future work. Overall, it was determined that it is feasible to apply reinforcement learning techniques to route cables, although challenging. In future work, the model can be expanded, making it more representative of a real-world design scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Thank you to VADM (Ret.) Phil Sawyer for helping to get the ball rolling on this work and for connecting me to Dr. Kaitlynn Castelle.

I would like to thank Dr. Kaitlynn Castelle for her mentorship and guidance throughout the process. She was always encouraging and willing to lend an ear to any ideas I had. I would also like to thank her for the numerous introductions she gave, including to Dr. Tom Rando.

I owe this thesis topic to Dr. Tom Rando. It was his idea to use machine learning to route electrical cables on submarines. He aided me tremendously during my literature review by supplying me with many relevant papers and answering any questions I had. He also proposed most of the ideas discussed in my future work section and created the very interesting 3D representation of a cable.

Thank you to Dr. Mark Karpenko for being my thesis advisor and for sharing my excitement for a somewhat out of the box mechanical engineering thesis topic. He kept me sane throughout the process by keeping me on track and helping me scale down the thesis, making it possible to complete it on time. I would also like to thank him for his perspective to treat the path of cable as the path of a car driving. This idea was fundamental to this work.

Thank you to my second reader Jessica Herman for her thoughtful comments and suggestions while reviewing my work.

Thank you, Matthew Norton, from the NPS Graduate Writing Center, for helping improve the clarity of my thesis and teaching me how to improve my writing.

Last but not least, I would like to thank my husband, Aran Damaso, for his support throughout the process. I couldn't have completed this thesis without his loving encouragement and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

During the design process of a U.S. submarine, 70,000 to 80,000 electrical cables must be incorporated into the design. Cables traverse the submarine in a series of cable support hangers referred to as cableways. The cross-sectional areas of the cableways determine how much space will be consumed by the cables. The cables that take up the most space, referred to as major cables, are those with a significant diameter and a corresponding minimum bend radius with a large value. The minimum bend radius is the smallest radius of a figurative circle that the cable can be bent around without causing damage to the cable [1] as shown in Figure 1.

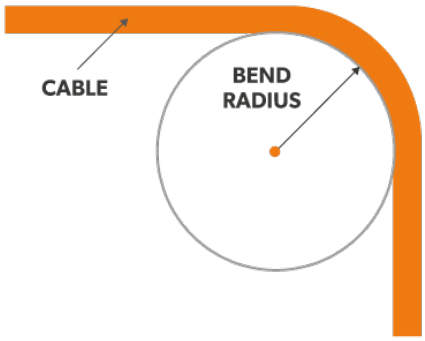

Figure 1. Depiction of the minimum bend radius of a cable. Source: [1].

The minimum bend radius is influenced by the diameter of the cable, the conductor type, the sheathing, the insulation, and other various materials used in cable construction [1]. The minimum bend radius is typically quantified as a multiple of the diameter of the cable, such as 5D. The major cables are only a fraction of the total electrical cables on the submarine. There are hundreds of major cables, while the total cables are in the thousands. However, properly routing the major cables can significantly reduce the overall space consumed by all the cables, because the major cables tend to consume a large amount of space. Today, the process for routing the major cables on-board submarines is a labor-intensive, sub-optimal manual process. Each cableway containing major cables must be

modeled using 3D computer aided design to ensure the minimum bend radius is not exceeded during cable placement. This process can be tedious and time-consuming.

The focus of this thesis is to study how the cable lay design process might be improved by using machine learning to automatically route cables. Specifically, the MuZero reinforcement learning algorithm in conjunction with a custom gymnasium environment will be used.

A. PREVIOUS STUDIES

Previous studies have considered automating similar design processes such as 3D pipe routing in industrial systems [2]. Pipe routing is similar to cable lay design because a path from a start location to an end location must be found while avoiding obstacles and using as little material as possible to reduce cost. However, the pipe-routing problem contains several more constraints such as the number of bends, the support costs, and the pipe's performance under stress during the design process, making it a much more complex problem than cable routing. There is a need to automate the design process for routing pipes because it is tedious and time-consuming similar to cable routing. The algorithm used in [2] planned a route by searching a binary conflict tree. During the search, conflicts are resolved using priorities set by the user. The algorithm used does not allow a pipe to collide with objects already present in an engineering plan. The search terminates after a user defined time-limit has been reached or the search has exhausted the tree. It was found that this method can provide solutions to complex industrial design processes, even ones with hundreds of pipes.

While the algorithm proved successful, it required expert knowledge to determine priorities that would lead to desired results [2]. The success of a plan was quantified by scoring it based on the amount of pipe used, the number of bends (bends are expensive), the support costs, and the pipe's performance during a stress and flexibility analysis. This has parallels to the reinforcement learning method proposed in this thesis where each plan created by the reinforcement learning agent will be scored according to a reward function. However, unlike [2], the calculated score in this thesis is used autonomously to train the model.

Automating engineering design has also been considered for routing the layout of tubes, hoses, and cable harnesses in commercial trucks [3]. Like the 3D pipe routing study described above, the commercial truck study did not make use of reinforcement learning (RL). This particular study used a graph-based routing algorithm that rapidly created a solution [3]. It is important in the automotive industry to design quickly because changes to other parts of the design are made often, requiring a rework of all or part of the truck's design. The algorithm planned by creating "branch routes" by mapping paths from a terminal to a "common starting point" which is the point that that "minimizes the sum of the Euclidean distance between this point and each terminal at each branch" [3]. Each branch route met at the common stating point. Routing was conducted iteratively. With each iteration the algorithm focused on minimizing the score map, which considered the distance to components already present in the design [3]. The goal of this study was not to create a final plan but to create a starting point for the engineers to further iterate during the automotive design process.

For these previous studies, the exact algorithm used differed, but the problem statement and the solutions contained several commonalities. In both cases, routing the objects manually was tedious and time-consuming. Both algorithms also used user defined priorities. Additionally, both studies used techniques other than reinforcement learning. There are many other studies that explored automatic planning without the use of RL such as routing multiple pipelines [4] and a distance-field-based pipe routing [5].

Most previously conducted studies on automating engineering design used non-reinforcement learning algorithms. However, an article was recently published on using reinforcement learning for automatically routing the pipes of ships [6]. This was published well after the work for this thesis began. Their goal was to quickly adjust the pipe routing after modifications had been made to the ship's design and to provide an engineer with an initial design, saving time and money. In their training method, they used curriculum learning, a method of training the agent gradually by adding complexity or more constraints with each training session. This method of using RL and curriculum learning is also explored in this thesis. However, the RL algorithm used in this thesis is the MuZero algorithm which consists of a learned model while [6] used Proximal Policy Optimization,

a model-free RL algorithm. The RL environment they used was created using a built-in machine learning library in Unity. They chose a node-based environment where each node is represented by x, y, and z coordinates. The visualization was supplied by the Unity game engine. They found even if equipment (obstacles) were changed or the number of pipes changed the proposed method could quickly respond and find new routes [6].

B. THESIS OBJECTIVE AND SCOPE

The objective of this thesis is to determine the feasibility of using reinforcement learning combined with a curriculum learning method to route electrical cables onboard a submarine. As a proof-of-concept, only 2D single-agent models are considered. A total of three RL models are examined, each with increasing complexity. Model 1 consists of the agent finding a path from a start location to a specified end location. The goal of Model 1 is to verify that the reward function has been properly defined. Model 2 contains the addition of obstacles for the agent to avoid, representing the real world more closely. The goal of Model 2 is for the agent to find a path from start to end, while avoiding obstacles. Model 3 has a more complex action space. The goal of Model 3 is to allow the agent to have more control over the curvature of the path found. The successful creation of these models provides proof of concept and lays the foundation for future work.

It is beyond the scope of this thesis to create a model that fully automates the cable lay process. Complete automation will require additional development to create an RL network capable of planning a collision-free 3D route from a given start location to a specified end location without violating the minimum bend radius. A collision-free route is one where no cables are intersecting. Complete automation is left for future work.

C. THESIS OVERVIEW

This thesis begins in Chapter II with a discussion of RL in an engineering context. Here, the basics of RL are explained, and an example of RL used in engineering is given. Next, in Chapter III the modeling of real-world cables is presented. An overview of the three models, Model 1, Model 2, and Model 3 is discussed. In Chapter IV, the use of MuZero, a model-based RL algorithm, in an engineering application is discussed. The details of the algorithm are also explained. In Chapter V, the implementation of the MuZero

algorithm used in this thesis, MuZero-General, is introduced along with the custom Gymnasium environment created to interact with MuZero-General. In Chapter VI, the training of the three models and their results are discussed. While training the original Model 1 and Model 2, it was found the reward function for the models had not yet been formulated correctly, and the models were adjusted to address some issues. RL often requires iterations of the environment to achieve desired results. To distinguish the iterations of Model 1 and Model 2 the naming scheme is expanded to Model 1A, Model 1B, Model 2A, Model 2B, Model 2C, Model 2D, and Model 2E. Lastly, in Chapter VII, this thesis is concluded and future work is discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

II. REINFORCEMENT LEARNING IN AN ENGINEERING CONTEXT

Artificial intelligence (AI) is a broad term that refers to a machine that can solve problems with human-like intelligence. Machine learning is under the umbrella of AI and refers to an algorithm that can mimic humans' ability to learn. For instance, humans learn how to identify the objects around them as young children. Computers can also identify objects in photographs through machine learning algorithms. Machine learning algorithms consist of neural networks which are approximations for mathematical functions. Neural networks can approximate functions by receiving an input that will pass through the network layers. As the input is passed through each layer, its value is adjusted by the weights of the neural network. The result is then outputted through the output layer. The neural network becomes better at approximating the output by adjusting the weights. These weights are adjusted during training. There are many training methods, the most common being supervised, unsupervised, and reinforcement learning.

Supervised learning uses a dataset that contains inputs with the matching correct outputs. During the training process, an input from the training data is supplied to the neural network and the output of the neural network is compared with the correct output provided by the dataset. The weights of the neural network are then updated according to a specified loss function. This process is repeated until the output of the neural network matches the output of the dataset within a certain threshold or the dataset has been exhausted. Supervised learning has a wide range of applications such as natural language processing and image classification.

Unsupervised learning is like supervised learning because it uses a dataset. However, this dataset contains only the inputs. This method is primarily used by data scientists to analyze data by recognizing patterns and grouping.

This thesis focuses on the use of RL. RL does not use datasets and in this way is unlike supervised and unsupervised learning. With RL, the computer learns to adjust its behavior over time by receiving a reward after an action or an action sequence, in order to map actions to rewards. If the reward is low, the agent will know to not take that action or

action sequence again. After a sufficient number of training episodes, the agent will have learned the action or action sequence that maximizes the long-term cumulative reward.

RL has been used in several different areas such as image processing, gaming, and robotics [7]. An RL algorithm typically consists of at least one agent, an environment, a way to interact with that environment (the action space), and a means to describe or observe the state the agent is in. A depiction of how these components interact in basic RL can be seen in Figure 2.

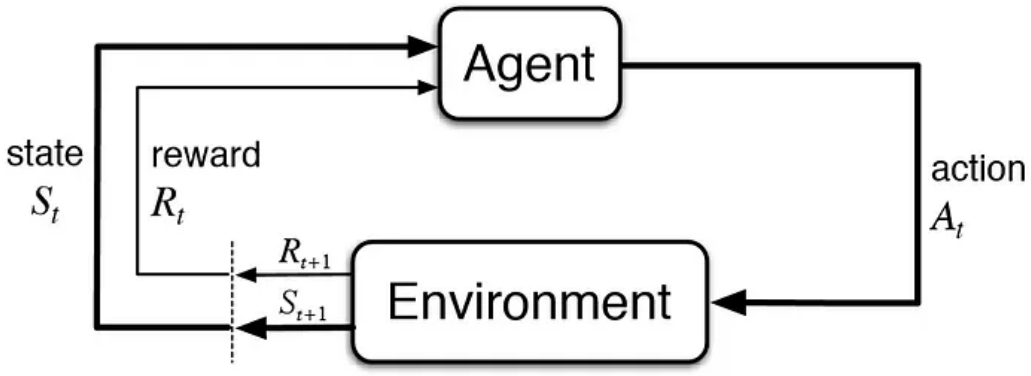

Figure 2. Depiction of the RL sequence. Source: [8].

Referring to Figure 2, based on the current state S_t and the reward R_t, the agent chooses an action, A_t which is inputted into the environment. The environment returns the state after the action has been taken, S_{t+1}, which may be the same as S_t. Additionally, the environment returns a numeric reward, R_{t+1}, for taking the chosen action and landing in the new state. The agent learns how good or bad the action was based on the reward. It updates its understanding and continues this cycle until the agent knows how to choose actions that result in the greatest reward.

As an example, in 2022 UC Berkeley researchers used a model-free reinforcement learning algorithm to teach a robotic dog how to walk [9]. In this case, the dog is the agent, the real-world surroundings are its environment, and the agent interacts with the environment by moving its mechanical joints. The agent's state contains information about

the root orientations, the root velocity, root linear velocity, joint angles, joint velocities, binary foot contacts, and the previous action [9]. Each time the agent takes an action the agent transitions to a new state and a corresponding numerical reward for taking said action and landing in the new state is received. The reward is determined from a user defined reward function. The reward function is one of the most important, if not the most important, aspect of RL because it drives the behavior of the agent. A poorly designed reward function will result in unexpected behavior of the agent. The computer will almost always find a way to exploit the system to receive a large long-term reward, even if the actions taken are not what the programmer intended. Thus, creating a reward function that gives desirable behavior is one of the most difficult aspects of applying the RL technique.

In general, RL is particularly useful for cases where automation is desired and there is not a clear way to teach the machine with pre-recorded examples of what actions are correct for a given situation (supervised learning). For example, to train the UC Berkeley robot dog how to walk without the use of RL would have required a dataset that contained information about what the dog should do when it is in a particular state. This quickly becomes impractical because the dataset would have to be substantial, as there are seemingly an infinite number of states the dog could be in and correspondingly an infinite number of correct actions for the dog to take given the current state. With the use of RL, the only information needed to train the robot dog (during the real-world portion) was the current state and the previous state [9].

In this thesis, there is a need to automate cable lay design, but, like the UC Berkley dog, collecting and using a large dataset for training is impracticable. Without the use of RL, training the model would require a dataset containing information about the unrouted environment of the cable and the corresponding answer of how the cable should be routed in that environment. To create a model that is generalized, thousands of these unrouted environments and routed environments pairs would be required. These mappings would likely be in the form of CAD drawings, which consume a significant amount of memory. This dataset would be too large to efficiently train with without the use of costly supercomputers. Additionally, creating this dataset would be time-consuming. As can be seen, training the cable-routing model with supervised learning is not practical. For these

reasons, RL was chosen as the learning method. Real-world characteristics and mathematical values can still be incorporated in the model via the reward function. For instance, the numerical value of the minimum bend radius can be used as a factor in the calculation of the reward.

In the following chapters the RL principles discussed in this chapter are applied to the cable-routing problem.

III. MODELING THE ELECTRICAL CABLES

The first challenge is how to represent real-world electrical cables computationally. Three models are implemented in this thesis, Model 1, Model 2, and Model 3. To reduce computational requirements and to provide a simpler model for study, the 3D environment of the real world is represented in 2D in all Models.

Model 1 is the most basic representation of a real-world cable. A depiction of Model 1 is shown in Figure 3. Referring to Figure 3, the cable is modeled as the black line connecting the points in gray. The points in gray are defined by x and y coordinates. Each point is placed by simulating a point object moving through 2D space with a specified velocity and initial position. At each step (1 s intervals), the position of the point object is recorded. The positions of the point object (the gray circles) are then connected to create a path. This process is considered the routing of the cable.

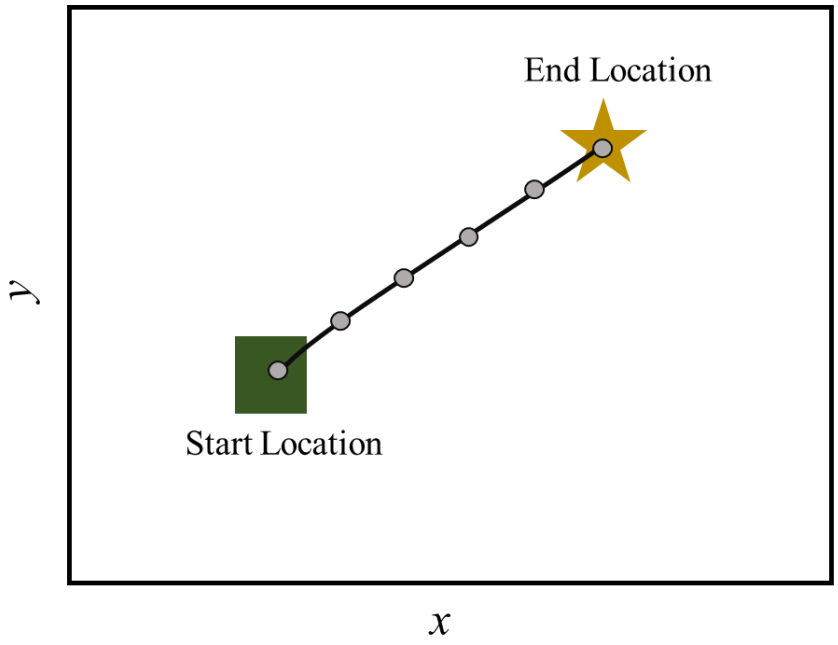

Figure 3. Depiction of Model 1

In terms of RL, this point object moving through space is the agent. The agent is indirectly choosing where to go at each step by choosing the cardinal direction to move in while the magnitude of its velocity remains constant. A depiction of the cardinal directions the agent can move in is shown in Figure 4.

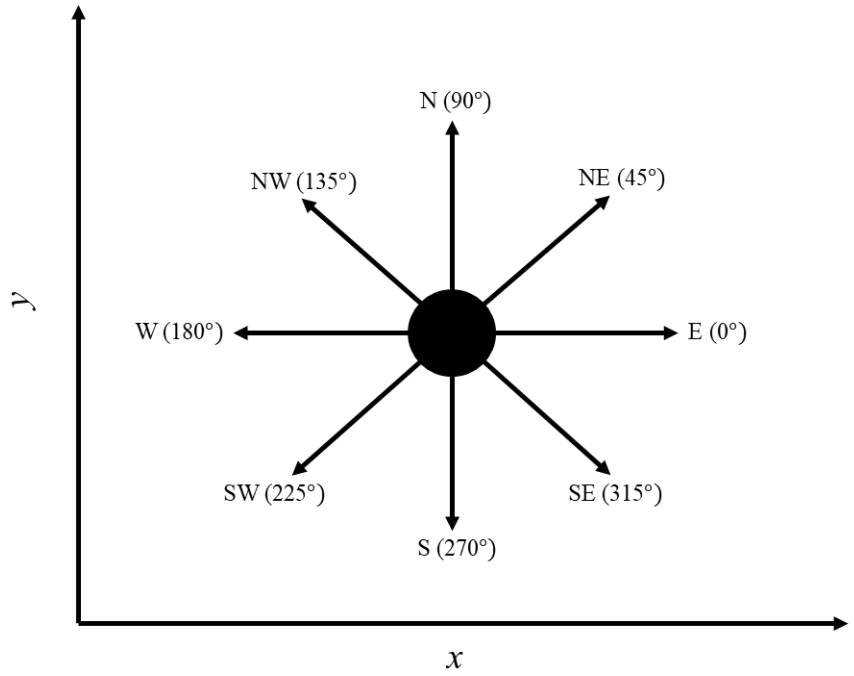

Figure 4. Direction the agent can move in for Model 1 and Model 2

Model 2, shown in Figure 5, is the same as Model 1 apart from the addition of obstacles. Obstacles are represented as point objects placed at a location defined by x and y coordinates. These obstacles are representative of a cable that is already present in the engineering drawing. The new cable, the black line, must be routed from the start location to the end location while avoiding obstacles.

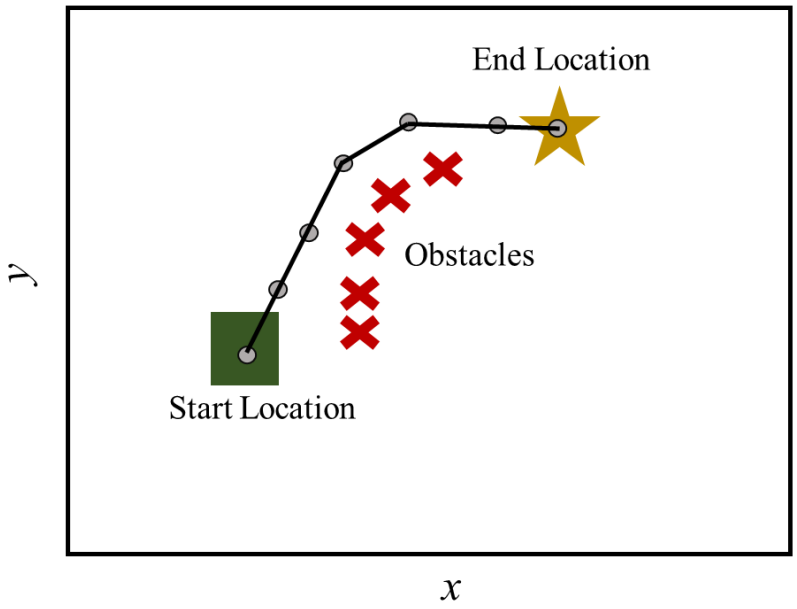

Figure 5. Depiction of Model 2

Model 3 is the same as Model 1 except for the way that the points of the cable's path are placed. Allowing the agent to move using only cardinal directions, as in Model 1 and Model 2, results in a path with possibly sharp turns instead of a smoothly curved path. The sharp turns do not model the curvature of the path of the cable, which is important because in the real world the curvature of the cable must not exceed the minimum bend radius. Therefore, the method of placing points in Model 3 was changed from the agent choosing cardinal directions to choosing how much to vary the direction angle. At each step, the agent chooses to add to, subtract from, or keep its current angle. For example, if the agent was traveling at an angle of 90° and it chose to add 1°, the agent's new angle would be 91°. This change may appear subtle, but it allows the agent to choose actions that produce granular adjustments. An example of a path that could be created with this method of placing the points is shown in Figure 6.

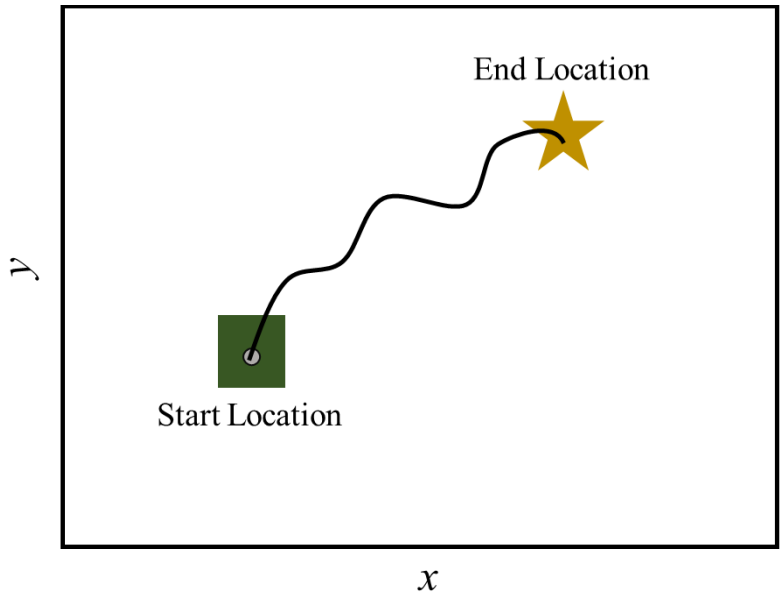

Figure 6. Depiction of Model 3

It should be noted that in Figure 6 the individual points were removed for clarity. The individual line segments between the connecting points are still straight lines as in Model 1 and Model 2. However, after the distance between the points becomes infinitesimal, the agent varying its angle with each step can result in a path with the overall appearance in Figure 6. Later in this thesis, RL is applied to each of the three models developed here.

IV. THE MuZero ALGORITHM IN AN ENGINEERING APPLICATION

A. OVERVIEW

The RL algorithm used in this thesis is MuZero, developed by the AI research company DeepMind [10]. It was designed to master complex games such as Atari without being given the rules of the game. Because the game rules are not included in the algorithm, it is highly generalized, making it a powerful tool for many applications, not just games. In engineering, there is a wide range of design processes that can be tedious, repetitive and have potential to be automated with machine learning. Since MuZero knows nothing about the details of the problem that needs to be solved, it can likely be applied to any repetitive design process, making its use ideal for engineers. Additionally, it is trained only from random plays against itself, meaning no outside data is required. As discussed previously, obtaining data can be quite difficult for complex environments. Its use is also ideal for engineers because it requires little knowledge of the complicated interworks of the algorithm. It requires only basic Python programming skills to set up an environment to interact with the implementation of MuZero used in this thesis.

B. THE MuZero ALGORITHM

The algorithm begins with the current observation which could be in the form of a picture of a game board or an array of data describing the current state. MuZero simulates action sequences, chooses an action, interacts with the environment, and then uses feedback from the environment (observation and reward) to improve its predictions. This process continues until the model is properly trained. The result is a model that knows how to receive the highest reward. A depiction of the overview of the algorithm can be found in Figure 7.

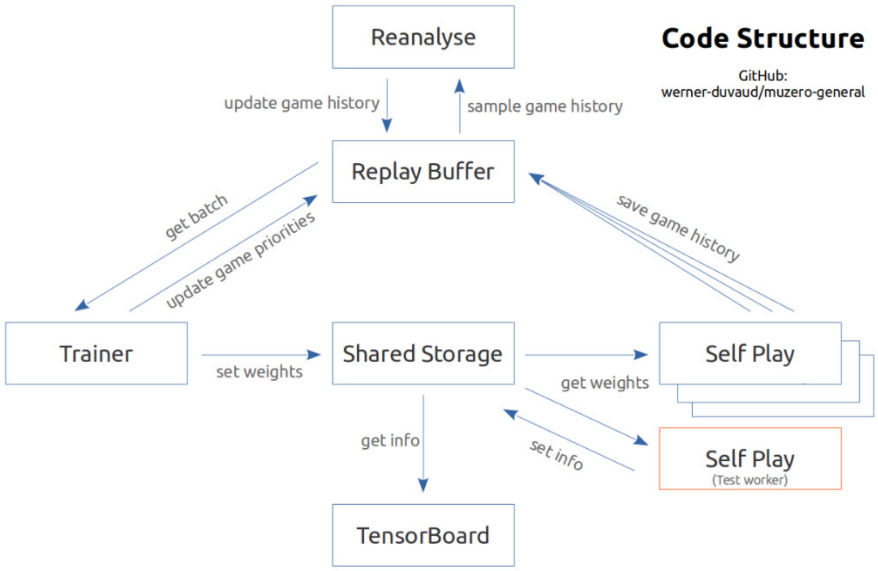

Figure 7. Overview of the MuZero algorithm. Source: [11].

The simulation portion is a combination of a learned-model and a tree-based search. The simulation is akin to a chess player thinking several moves ahead to decide what action should be taken right now. The MuZero agent can look ahead any specified number of steps and make a prediction about what action should be taken.

The learned-model is composed of three functions: the representation, prediction, and dynamics function [10]. These functions are represented as neural networks. From these three functions, the agent plans the path to take by predicting, at each time step, the policy, the value function, and the reward that would be received [10]. The policy function maps states to actions. For a given state (the input) it will output the action to take. The value function is an estimate of the future rewards.

The representation, prediction, and dynamics function are used in conjunction to build a Monte Carlo Tree Search (MCTS). A schematic of an example of a MCTS is shown in Figure 8.

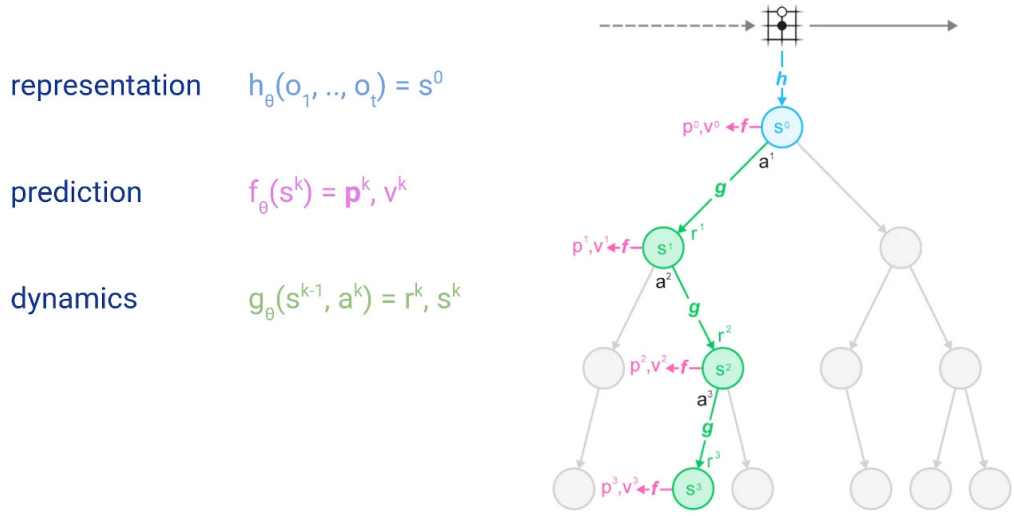

Figure 8. Schematic of a representation of a MCTS. Source: [12].

A Monte-Carlo Tree Search is conducted at each timestep, and the search begins at the root of the tree and proceeds down the tree [10]. Each time a node is visited, its visit count for that node is incremented by one. The agent continues searching until it reaches a leaf (a node with no child) [10]. The leaf is then expanded. As new paths are explored, the tree grows, allowing for a deeper search.

For a given observation, the representation function determines the initial hidden state, which may not contain all the information of the original state, because only the information needed for planning is considered [10]. The prediction function computes the policy and value function for the given hidden state. The policy function and the value function are used in conjunction to determine the next hypothetical action. The dynamics function then computes the hypothetical reward and the hypothetical new state after taking the hypothetical action.

The dynamics function is necessary because MuZero has no knowledge of the rules of the game, or in more general terms the dynamics of the environment. The rules of the game provide information about the next state. For instance, if the game is chess and the algorithm had the rules of the game, then it would know what the board would look like after moving a pawn to the D2 location on the board, and there would be no need to predict

what the next state would be. After the MCTS is explored, an action from the search policy, proportional to the visit for each action from the root node, is chosen [10].

The visit count can be altered using a constant value ranging from 1 to a very small positive number. If the parameter is small, the action choice will more likely be one that has a low visit count (has not been fully explored). If the value is 1, then the best-known action, the one with the highest visit count, will most likely be chosen. This is considered the tradeoff between exploration and exploitation. It is important to focus more on exploration in the early stages of training to ensure the true best action is found. Correspondingly, the agent should focus more on exploitation while nearing the end of training because it has already explored all or almost all the actions and has found the best one.

The chosen action is then inputted into the environment and the reward for transitioning from the current state to the new state is given along with the new observation. A particular path down the tree is considered a trajectory and is saved into a replay buffer for later use as shown in Figure 7. MuZero improves its learned-model by sampling a trajectory from the replay buffer. Using the real action taken and the observation and reward received, the weights of the representations, prediction, and dynamics neural networks are updated via supervised learning. To be clear, MuZero is an RL algorithm, but it also uses supervised learning to update the learned-model as a means to improve the simulation performance. Even though MuZero uses supervised learning in part of its algorithm, the dataset used during the supervised learning portion is created by the computer during self-play. In other words, the algorithm uses RL principles to create data (the trajectories) and then uses that data with supervised learning to improve the performance of the learned-model. The overall loss function used to update the learned-model is described in Equation (1) [10],

$$l_t(\theta) = \sum_{k=0}^{K} l^r\left(u_{t+k}, r_t^k\right) + l^v\left(z_{t+k}, v_t^k\right) + l^p(\pi_{t+k} + p_t^k) + c\|\theta\|^2 \ , \qquad (1)$$

where l_t is the total loss, l^r, l^v, and l^p are loss functions for reward, value, and policy respectively. u_{t+k}, z_{t+k}, π_{t+k} are the target reward, value, and policy respectively while r_t^k, v_t^k, p_t^k, are the predicted reward, value, and policy respectively. θ is the model parameters, which are regularized by the last term in Equation (1).

The goal during training is minimize the overall loss which is akin to the overall error of the reward, value, and policy functions. This process is repeated until the learned-model has been properly trained. Proper training means that during simulation, the agent can make accurate predictions about the actions that will lead to the highest rewards.

Overall, MuZero is a powerful algorithm. Even though it was designed to master games such as Atari and chess [10], it has the potential to be used in a wide variety of applications outside of games including the cable-routing problem in this thesis. In the next chapter the specific implementation of the MuZero algorithm used in this thesis is introduced.

THIS PAGE INTENTIONALLY LEFT BLANK

V. THE CABLE ENVIRONMENT

A. OVERVIEW

The implementation of MuZero used in this thesis is named MuZero-General and was published by Werner Duvaud on GitHub [11]. This implementation of MuZero is intended for educational purposes only. Additionally, at the time of writing this thesis, MuZero-General is supported only on a Linux machine. If a Linux machine is not available, cloud computing platforms are an easy-to-use option. Google Collaboratory's GPU was used to run MuZero-General in this thesis.

As discussed previously in Chapter II, RL consists of an agent, an environment, and the interaction between the two. MuZero-General supplies the agent and interacts with the environment by supplying the agent's chosen action and receiving the new observation and the reward in return from the environment. The environment used in this thesis is a custom-built Gymnasium environment, which MuZero-General is designed to work seamlessly with.

Gymnasium is an application programming interface for single-agent RL environments [13]. A Gymnasium environment is an instance of a Python class and consists of four key functions: make, reset, step, and render [13]. The make function creates an instance of a Gymnasium environment. The reset function returns the agent to an initial state at the beginning of each episode. The step function takes in the action chosen by the agent and transitions it to a new state and returns the new observation, the reward, and a boolean specifying if the agent has reached a terminal state. A terminal state is a state that immediately results in the episode ending. The render function is how the environment is visualized. It allows humans to see the world that the agent is interacting with.

B. CUSTOM-BUILT GYMNASIUM ENVIRONMENT FOR INTERACTION WITH MuZero-General

In the environment built for this thesis, the "game" is finding a path from a given start location to a goal location. The agent must find the path through a bounded 2D space

by choosing discrete numbered actions. The necessary functions in the Python class are __init__, step, reset, and render function and are discussed in detail below.

1. The __init__ Function

The __init__ function is standard in almost all Python classes. Within the __init__ function, the observation space and the action space can be set. These are required Gymnasium settings. The observation space is the specified area that the agent is allowed to exist in. The chosen observation space for Model 1A and all Models except Model 1A is shown in Figure 9. The observation space is a Gymnasium "box" with a lower bound of 0 and an upper bound of 10 for Model 1A. The bounds for all other models are a lower bound of -20 and an upper bound of 20. The reason for this change is related to the RL training process and is discussed in Chapter VI.

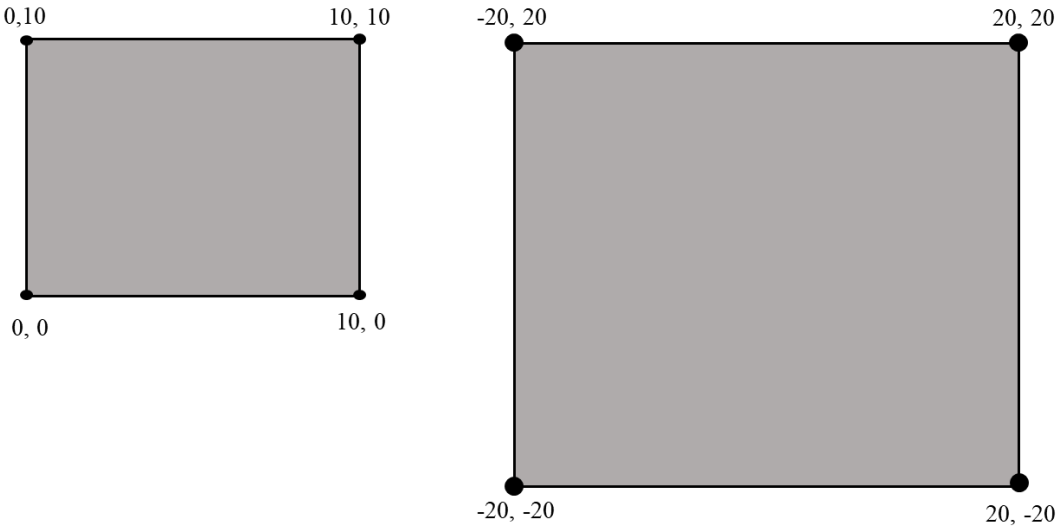

Figure 9. The observation space: a) Model 1A; b) All other models

The action space is the list of action choices for the agent. For the cable environment, the action space is set as discrete numbers 0 through 7 for Model 1 and Model

2. The action space is 0 through 8 for Model 3. Each number choice is mapped to a specific action. The action that corresponds to each number for each model is shown in Table 1.

Table 1. The action that corresponds to each number for Model 1, Model 2, and Model 3

Number	Model 1 and 2 Action	Model 3 Action
0	E (0°)	Add 1°
1	NE (45°)	Add 5°
2	N (90°)	Add 10°
3	NW (135°)	Add 25°
4	W (180°)	Subtract 1°
5	SW (225°)	Subtract 5°
6	S (270°)	Subtract 10°
7	SE (315°)	Subtract 25°
8	N/A	No Change

The start location and the goal location are hard coded at locations (0, 0) and (9, 9), respectively for all models. For Model 2A, there are five obstacles hard coded into the environment at (*x, y*) location (4, 2), (4, 4), (4, 6), (6, 6), and (6, 8). These obstacles are placed in these positions to achieve the appearance of a cable that has already been routed in the environment. For Model 2B, Model 2C, Model 2D, and Model 2E there is a single object located at (4, 4).

2. Step Function

The step function takes in an action and returns the observation, the reward, and the status of meeting termination criteria. The reward function can be defined within the step function. However, in this study, it was defined separately and then called from within the step function to allow for unit testing of the reward function prior to training. For the same reason, the observation calculation and the check for terminal states were defined separately and then called from within the step function. The interaction between the step function and its components is shown in Figure 10.

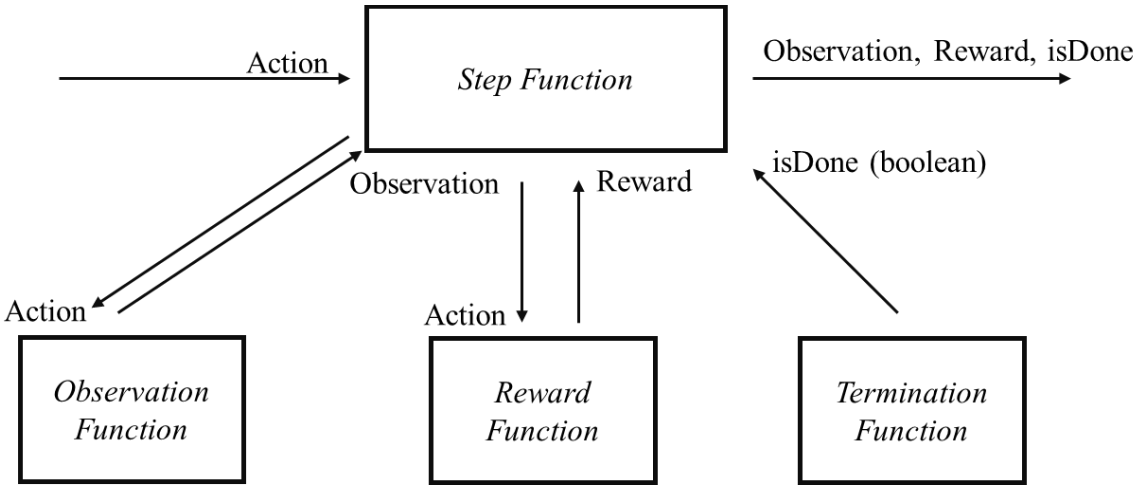

Figure 10. The step function and its components

a. Observation Function

To represent the path of the cable mathematically, the cable was envisioned as a point object moving through space with a certain velocity. Recording the position of the point in space at each time step and then plotting the recorded positions of the point results in a path.

Translating this concept to the cable lay environment, the point is considered the agent, and the observation is the x and y position of the agent at each step. The magnitude of the agent's velocity is constant, but the x and y components of its velocity change based on the angle it is traveling at. The angle the agent is traveling at is directly chosen in Model 1 and Model 2 or is modified in Model 3 using the action choices described in Table 1. The previous angle of the agent is then updated accordingly.

The position of the agent is then updated using its previous location, new angle, total velocity, and the time per step. The velocity of the agent was broken down into its x and y components using the velocity triangle found in Figure 11.

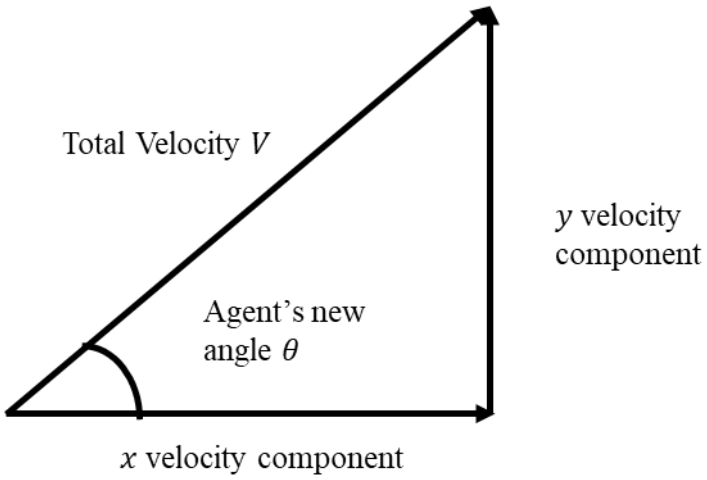

Figure 11. Agent's velocity triangle

Equations for updating the x and y position were developed using the velocity triangle and basic principles of motion,

$$x_{t+1} = x_t + Vtcos(\theta_t), \tag{2}$$

and

$$y_{t+1} = y_t + Vtsin(\theta_t), \tag{3}$$

where x_{t+1} is the x position after a step was taken, x_t was the x position before taking a step, V is the magnitude of the velocity, t is the amount of time it takes to complete one step, and θ_t is the new angle of the agent. The magnitude of the velocity was set to a constant of 0.5 m/s. The time per step was set to 1 s. These quantities were chosen to make the calculations simple. In reality, cables are usually not several meters long. To the agent, there is no difference between 0.5 m/s and 0.5 cm/s. Once calculated, the new and x and y locations are returned to MuZero-General as an observation.

b. Reward Function

The reward function in this environment is simple. It is a summation of rewards received if the agent's action resulted in certain criteria being met. For all models, to encourage the agent to move toward the goal, the agent is given a positive reward for

entering a particular zone defined by the Euclidean distance to the goal. Each time the agent moves up a zone it gets closer to the goal and is given an additional positive reward. However, if it goes down a zone, the agent receives a negative reward equal in magnitude to the reward given for entering the zone.

It is crucial that the positive reward for moving into a zone is negated with a negative reward of equal or greater magnitude for leaving the zone. If it is not negated, the agent will continually leave the zone and enter the zone until reaching some other terminal state (such as a step limit). This will occur because the agent can receive a larger long-term reward by bouncing back and forth instead of ending the game. In total, there are six zones defined. Once the agent enters the final zone, zone 0, it has reached the goal, receives a large positive reward, and then the episode terminates. The range to the goal for each zone, and the magnitude of the reward for entering or leaving the zone can be found in Table 2.

Table 2. The range for each zone and the magnitude of the reward received by entering or exiting the zone.

Zone Number	Distance to Goal	Value of Reward Received by Entering or (Exiting) the Zone
5	<= 6	20 (-20)
4	<=5	30 (-30)
3	<=4	40 (-40)
2	<=3	50 (-50)
1	<=2	60 (-60)
0	<=1	200 (-200)

Additionally, the agent receives a small negative reward of -0.1 for each step it takes to incentivize the agent to reach the goal in as few steps as possible. This translates in the real world to using the shortest length of cable possible, which reduces cost.

In addition to the previously mentioned rewards, certain models contain other rewards. In Model 2A, Model 2B, Model 2C, and Model 2D if the agent is within 0.25 m (Euclidean distance) of an obstacle then a reward of -10 is given.

For Model 2E, there are zones placed around the object similar to the zones around the goal. If the agent is within 0.3 m of the obstacle it receives a reward of -10, -20 for being within 0.2 m, and -30 for being within 0.1.

c. Termination Function

The episode terminates if any of the following conditions are met: the agent is in the final zone, the agent is within a specified distance to the boundary (for all models except Model 1A), or the agent has reached the step limit. The episode is terminated after reaching the final zone because the agent has completed its mission, so there is no need to continue. For this environment, it was found that unexpected behavior occurs when the agent is at the boundary. For this reason, the episode is terminated before reaching the boundary for all models except Model 1A. A depiction of the area the agent can operate in without termination is depicted in Figure 12.

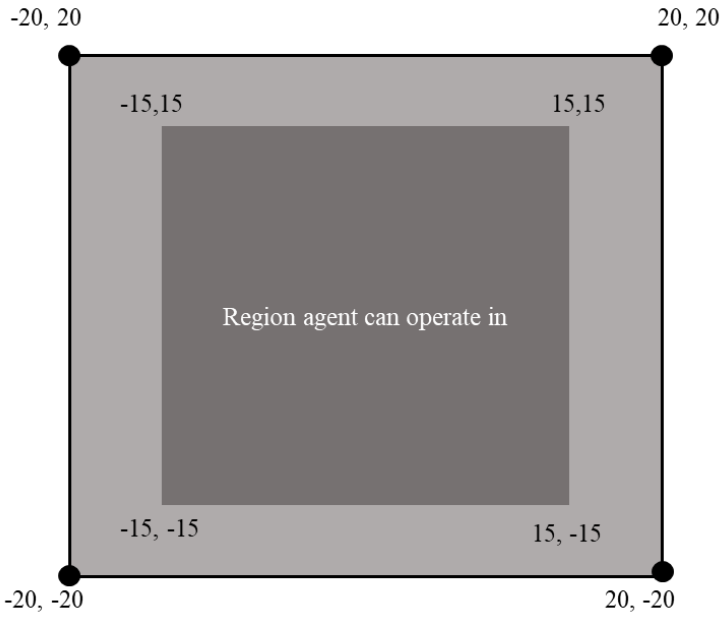

Figure 12. The region the agent is allowed to operate in before termination for all models except Model 1A

Additionally, it is important to limit the agent's training episode to a certain number of steps to prevent the agent from becoming too off track. The maximum number of steps

per episode was set to 150. The distance from the start location to the goal location is 9 m. The agent moves 0.5 m with each step. This means the agent can move 66 m more than is required to reach the goal. This is believed to be enough leeway for the agent to explore the environment without deviating from the goal too much.

3. Reset Function

The reset function is called after the current episode terminated and a new episode has begun. The reset function ensures that every episode begins the same way. In the reset function, the current position is set to (0,0), the current angle is set to 90°, the step counter is set to 0, the current zone is set to 6.

4. Render Function

The render function is how the environment is visualized. It allows the user to see the world that the agent is interacting with. Rendering can be simple, such as an *x*-*y* plot or complex using a 2D game engine such as pybox2D. In this thesis, a simple *x*-*y* plot was chosen. After an action is completed, the *x* and *y* location are plotted. An image of the plot is then saved for later viewing.

In the next chapter, the RL model designed here is used to train the cable agent.

VI. TRAINING AND RESULTS

A. MuZero-General TRAINING CONFIGURATION

Each model was trained on a Google Collaboratory GPU with a batch size of 200 (the number of parts of games to train on at each training step), simulated 50 moves ahead, kept 500 self-play games in the replay buffer, and had a discount reward factor of γ =0.997. These parameters are set within the MuZeroConfig class portion of MuZero-General. The value of γ adjusts the agent's emphasis on either immediate rewards or long-term rewards. The range for the value of the factor is $\gamma \in [0,1)$. If it is set to 0 then the agent would be focused on instant gratification. On the other hand, if the reward is very close to 1 (not useful be set to exactly 1) then it would focus heavily on achieving the greatest future rewards. In this thesis, to achieve the desired results the agent needs to be focused on receiving the future reward of reaching the goal. For this reason, the discount factor was set close to 1 at 0.997.

Additionally, each model's tradeoff between exploration and exploitation was determined using the same temperature function. The temperature function returns the parameter that alters the visit count distribution to ensure that the action selection focuses more on exploitation as training progresses. A smaller parameter means the agent is more likely to choose the action with the highest visit count (the best-known action). The parameter that is returned is determined by how many training steps have been completed and how many remain. The value of the parameter is chosen using the criteria shown in Table 3.

Table 3. The value of the parameter returned by the temperature function

Temperature Parameter Value Returned	Number of Training Steps Completed
1.00	Less than 50% of total steps
0.50	Less than 75% of total steps
0.25	Greater than 75% of total steps

All training settings are the same for each model apart from the total number of training steps. Additionally, some models were trained using curriculum learning while others were not. Curriculum learning is the method of adding additional constraints or complexity with each training session. Curriculum learning can sometimes reduce overall training time because the agent typically responds better to learning one new constraint at a time. The training steps and the distinction between those trained using the curriculum learning method for each model is shown in Table 4.

Table 4. Total training steps for each model

Model Number	Total Training Steps	Curriculum Learning (Y/N)
1A	10,000	N
1B	20,000	N
2A	40,000	Y
2B	40,000	Y
2C	20,000	N
3	20,000	N

B. MODEL 1

Model 1A is the first model considered. The path taken by the agent prior to training is shown in Figure 13a. The corresponding reward at each step for this path is shown in Figure 13b. Model 1A was then trained for 10,000 steps, and the path taken after training is shown in Figure 13c. The corresponding reward at each step taken is shown in Figure 13d.

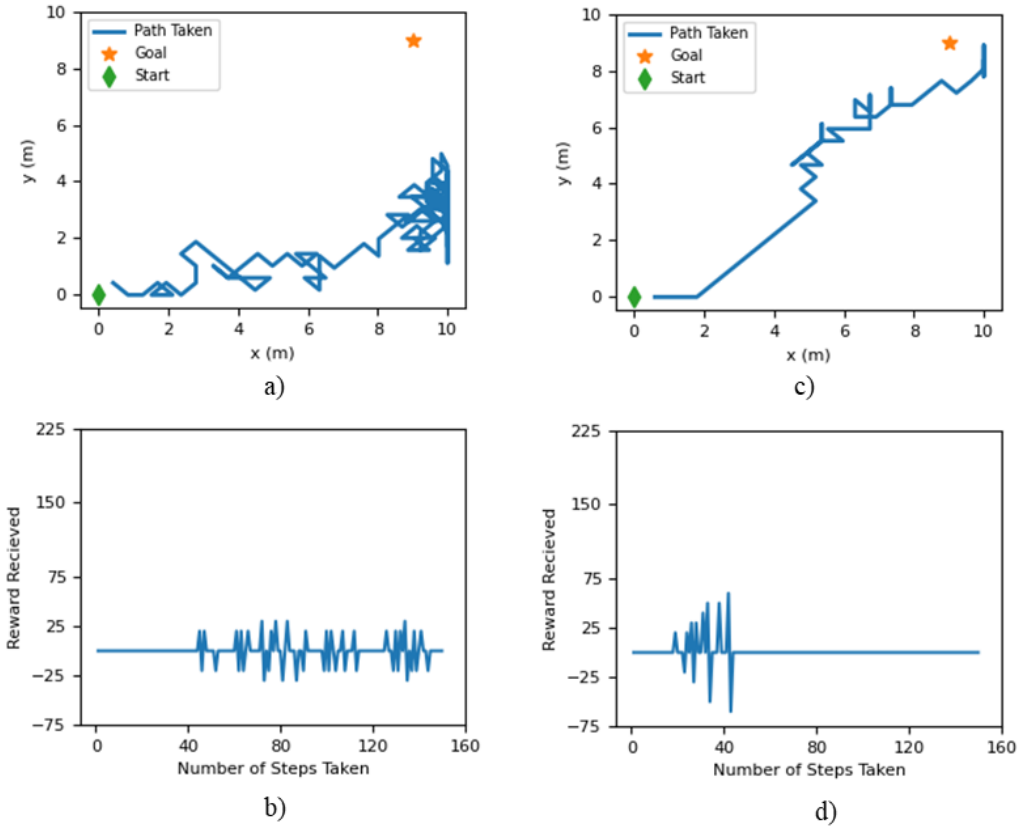

Figure 13. Results for Model 1A: a) Before training; b) Reward at each step before training; c) After training; d) Reward at each step after training

Referring to Figure 13c, the agent learned to move more in the direction of the goal after training. However, it never made it to the goal because it began alternating its action choice between Northeast and South on the boundary until it reached the maximum number of steps per episode. This undesired behavior at the environment boundaries led to the conclusion that it is best practice to not allow the agent to operate on the edge of the boundary.

To attempt to resolve this issue, the size of the environment of Model 1A was increased from a range between 0 m and 10 m to a range between -20 m and 20 m and was labeled as Model 1B. A termination criteria for terminating the episode when the agent is within 5 meters of the boundary was added to Model 1B. Model 1B was originally trained with 10,000 steps. However, the model did not appear properly trained. As a result, the

training steps were increased from 10,000 to 20,000. The graphs of Model 1B before and after completing training are shown in Figure 14.

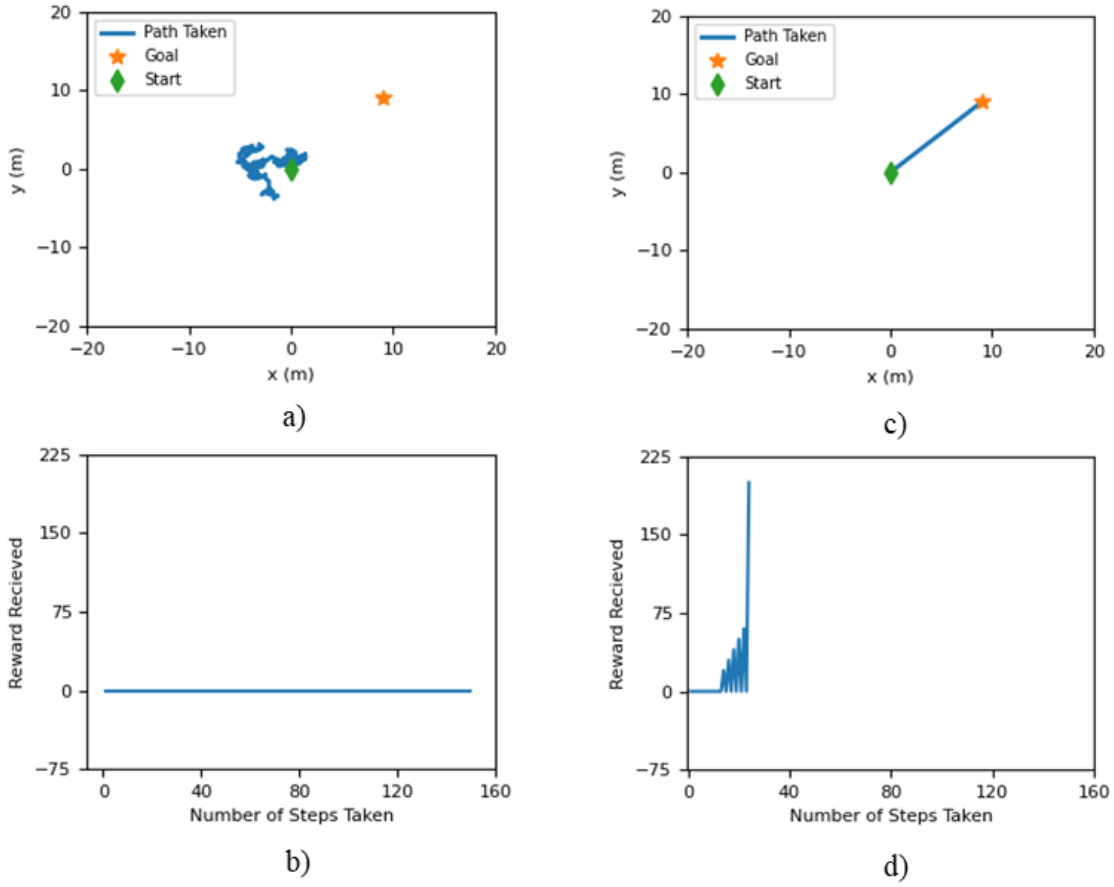

Figure 14. Results for Model 1B: a) Before training; b) Reward at each step before training; c) After training; d) Reward at each step after training

Looking at Figure 14a, the actions of the agent in Model 1B prior to training resulted in a sporadic path. In Figure 14c, after training, the agent takes the shortest path from the start location to the end location, achieving the greatest long-term reward possible. These results demonstrate that the undesired behavior at the environment boundary can be avoided by not allowing the agent to approach it. Additionally, these results confirm that the reward function that was defined is sufficient to teach the agent to reach the goal.

C. MODEL 2

For Model 2, obstacles were added as an additional consideration. With the additional constraint to avoid obstacles, the goal for the agent in Model 2 is to take actions that not only take it from the start to the finish as in Model 1, but to do so while avoiding an obstacle(s). Within Model 2, there are five versions Model 2A, Model 2B, and Model 2C, Model 2D, and Model 2E.

Model 2A's training began where Model 1B's training left off, i.e. curriculum learning. Model 2A was trained for an additional 20,000 training steps for a total of 40,000 training steps. A depiction of Model 2A before and after training is shown in Figure 15.

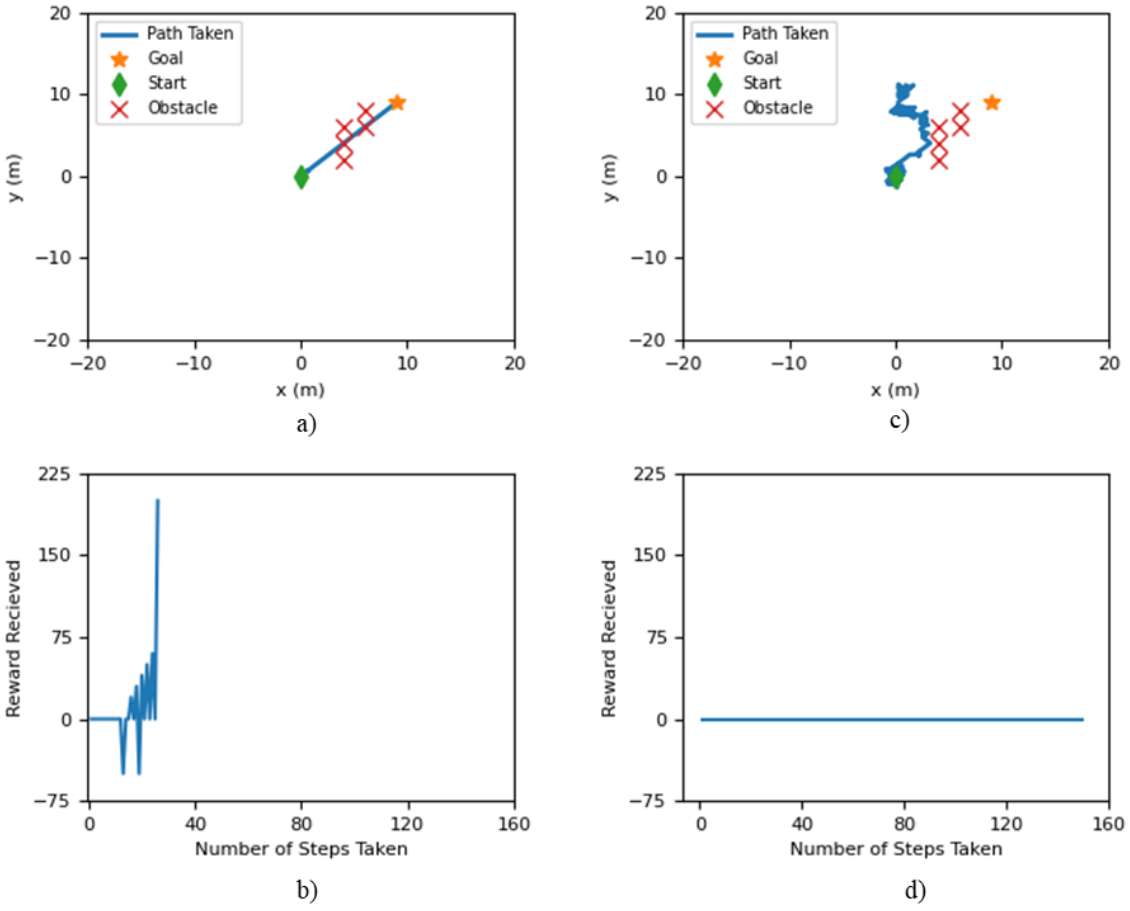

Figure 15. Results for Model 2A: a) Before training; b) Reward at each step before training; c) After training; d) Reward at each step after training

Referring to Figure 15, Model 2A before training took the same direct path from start to finish, as it did in Model 1B after training. The key difference between the trained Model 1B and the untrained Model 2A is the rewards received. The path the resulted in the greatest reward for Model 1B, is now not the optimal path for Model 2A because of the altered reward function. Taking the path that the agent learned in Model 1B resulted in several rewards of -10 along the way. After training for an additional 20,000 training steps, the agent has not yet found a path from the start to the end while avoiding the obstacles. However, it is evident that the agent had begun learning to avoid the obstacles. It was expected that the agent would have performed better than it did after a total of 40,000 training steps. Looking at Figure 15d, the agent appears to never have received any positive reward, indicating that there may be an issue with how the reward function was defined. It is believed that adding five obstacles at once was overly ambitious. For this reason, four out of the five obstacles were removed from Model 2A, and the resulting model was labeled as Model 2B.

The Model 2B was trained the same way as Model 2A. The results are shown in Figure 16.

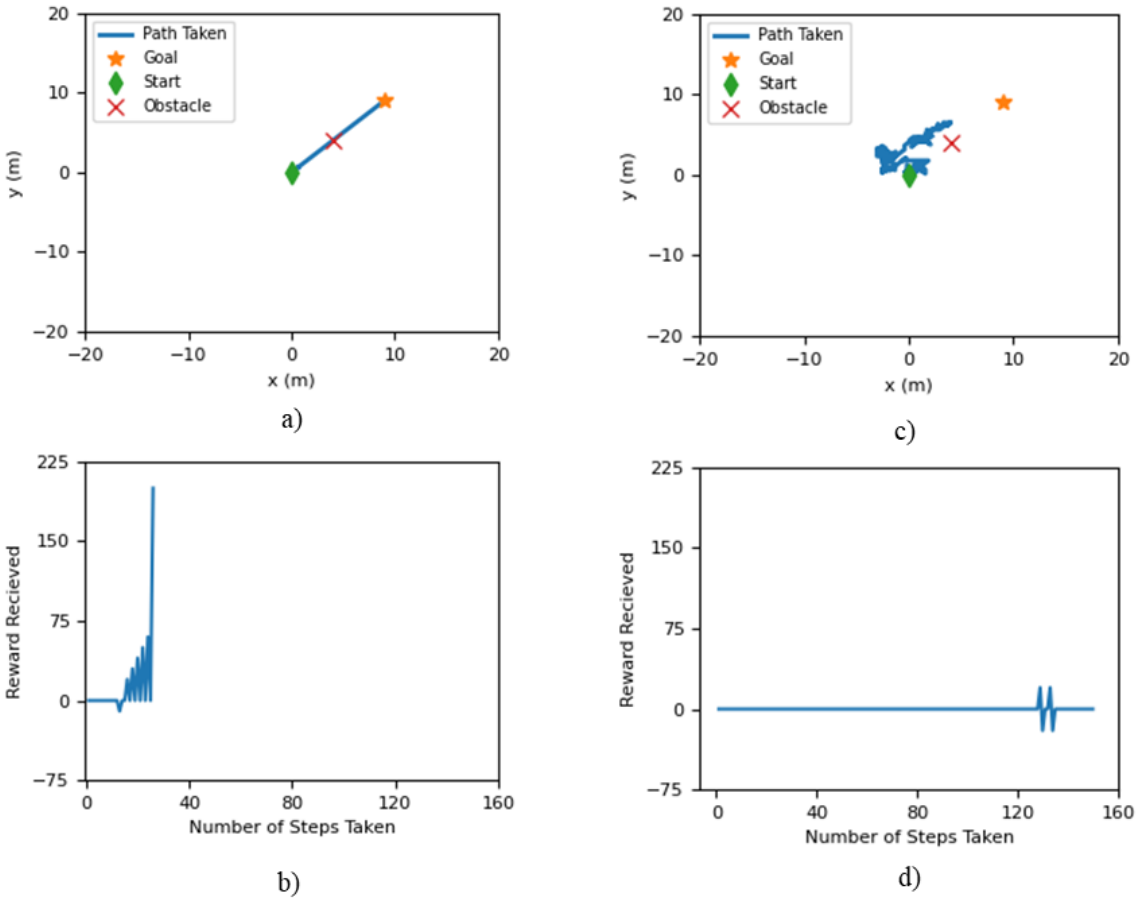

Figure 16. Results for Model 2B: a) Before training; b) Reward at each step before training; c) After training; d) Reward at each step after training

Referring to Figure 16, the agent was still unable to reach the goal while avoiding an obstacle. However, Model 2B appears to be further along in the training process than Model 2A. A side-by-side comparison of the two Models can be found in Figure 17, where its seen that the agent does receive some positive reward at the end of training.

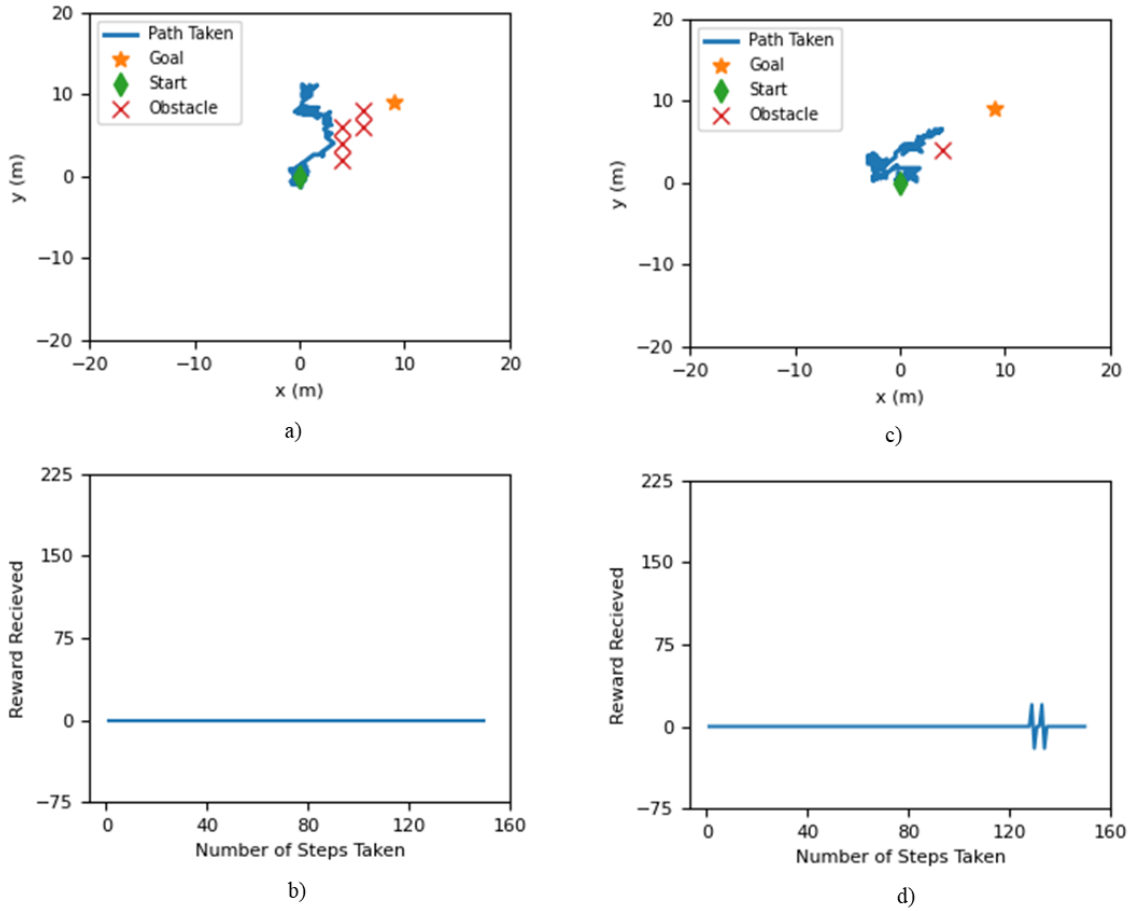

Figure 17. Comparison of Model 2A and Model 2B after training: a) Model 2A; b) Model 2A reward received at each step; c) Model 2B; d) Model 2B reward received at each step

It was then hypothesized that the lack in performance is because of the curriculum learning method used as well as issues with the reward function. To further debug, Model 2B was trained without curriculum learning and labeled as Model 2C. Model 2C was trained for a total of 20,000 training steps. The results for Model 2C can be found in Figure 18.

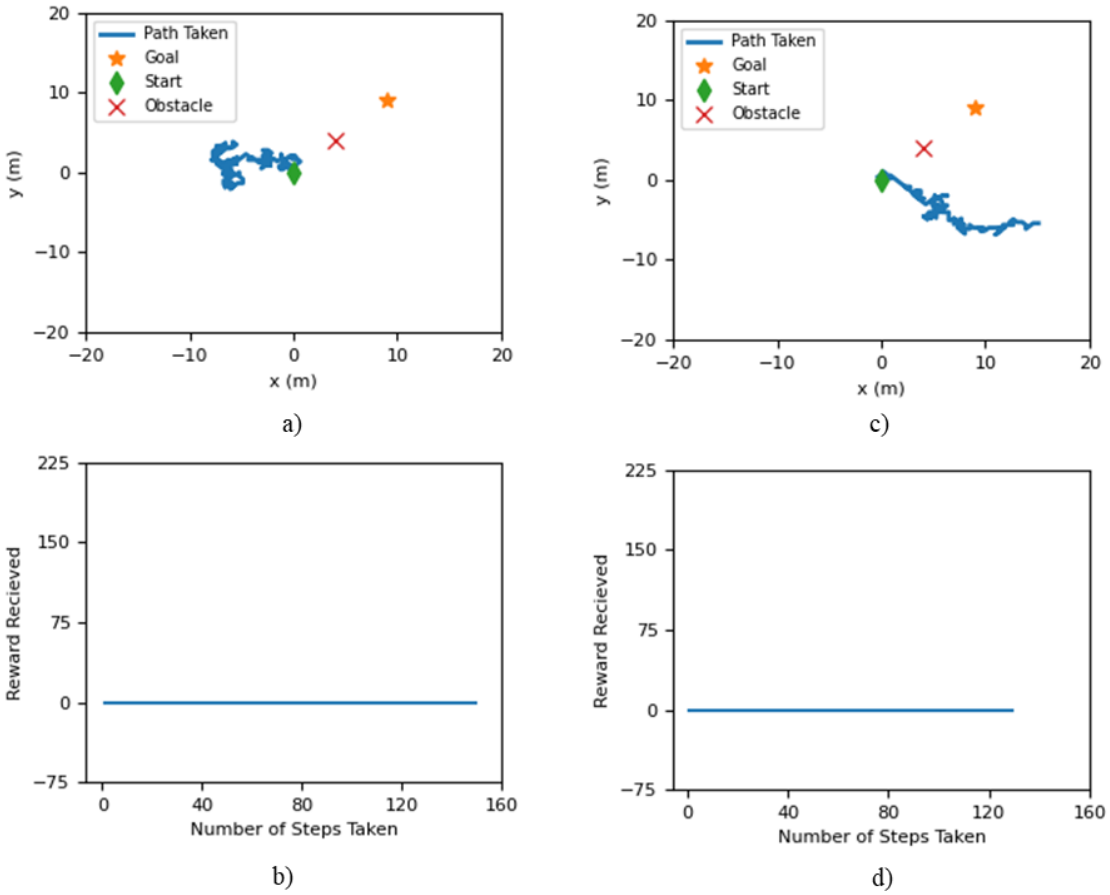

Figure 18. Results for Model 2C: a) Before training; b) Reward at each step before training; c) After training; d) Reward at each step after training

These results show the belief that the curriculum learning method was the culprit for the low performance in Model 2A and Model 2B is incorrect. Model 2B trained using curriculum learning achieved better performance than Model 2C which was trained without the use of curriculum learning. It is now believed that the training steps chosen for all Model 2 models were not sufficient for proper training due to how the reward function was constructed. In future work, it is expected that the performance would improve if the reward function is suitably refined.

It is important to note that Model 1B was trained properly in 20,000 training steps while Model 2C was not. The only difference between the two models is the single obstacle located at (4,4) in Model 2C. These findings show that adding in what appears to be a simple addition can lead to significantly different requirements for the reward function.

In Model 2D, the obstacle in Model 2C was moved from (4, 4) to (-9, 1) so that the obstacle would not intersect the optimal path from the start to the goal. This was done to see if the position of the obstacle affects the training outcome. Model 2D was trained for 20,000 steps, the same as Model 2C. A side-by-side comparison of Model 2C and Model 2D after training is shown in Figure 19. Comparing Figure 19a and Figure 19c, it is clear that the location of the obstacle did affect the training outcome. Model 2D appears to be further along in the training process than Model 2C does. Even so, the agent in Model 2D is still not receiving positive rewards after training.

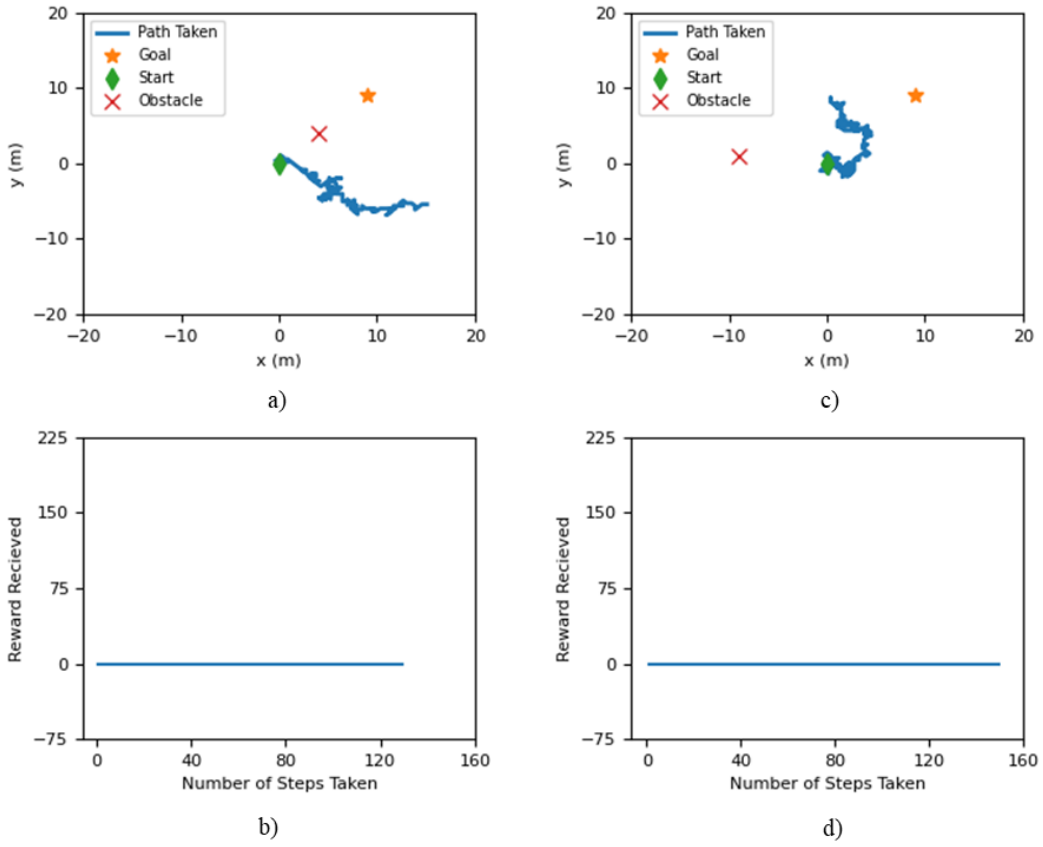

Figure 19. Comparison of Model 2C and Model 2D after training: a) Model 2C; b) Model 2C reward received at each step; c) Model 2D; d) Model 2D reward received at each step

To attempt to resolve the issues with the reward function, the reward function for Model 2E was altered. For Model 2E, the reward for intersecting an obstacle was changed

from the single reward of -10 for being within 0.25 m of the obstacle to a series of zones around the obstacle as described in Chapter V. Model 2E was trained for 20,000 training steps. A side-by-side comparison of Model 2D and Model 2E after training is shown in Figure 20. Referring to Figure 20c, the agent made it to the goal in just over 120 steps. Compared to Model 2D after 20,000 training steps shown in Figure 20a, it is clear that the reward function used in Model 2D is better suited to teach the agent to avoid obstacle than the reward function of Model 2B.

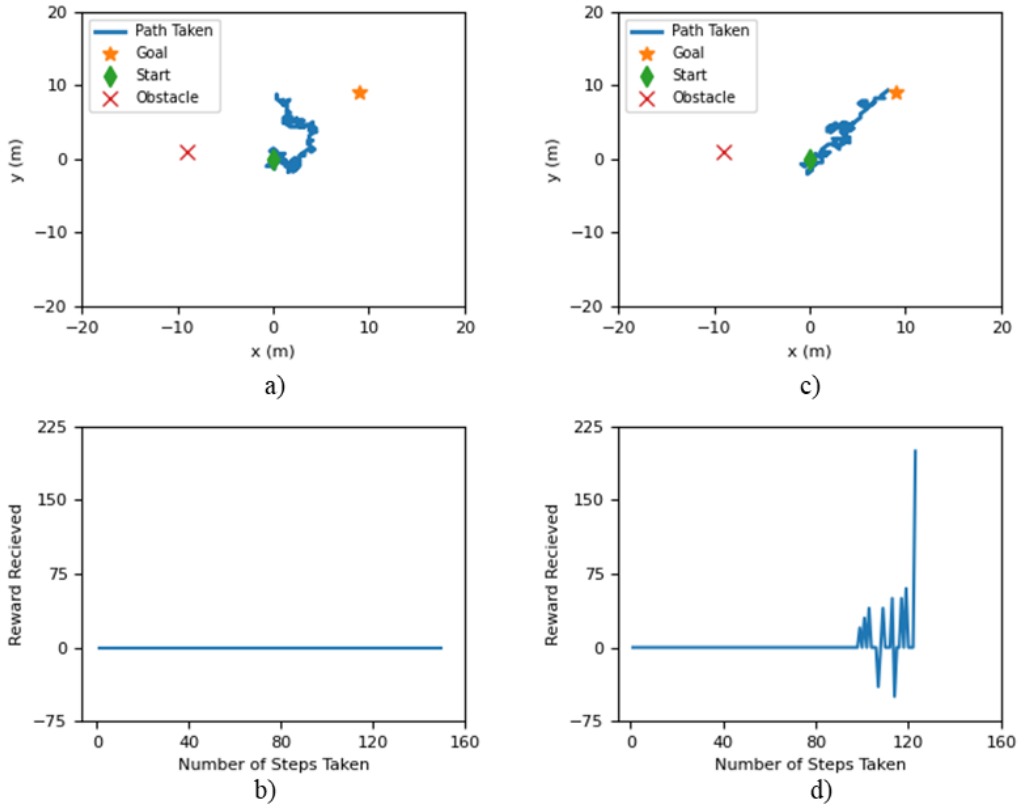

Figure 20. Comparison of Model 2D and Model 2E after training: a) Model 2D; b) Model 2D reward received at each step; c) Model 2E; d) Model 2E reward received at each step

In Model 2E and Model 1B both agents found paths from the start to the goal location after 20,000 training steps. However, the agent in Model 1B, which does not contain any obstacles, was able to find a path in about 30 steps while the agent in Model

2E took just over 120 steps to reach the goal. It is likely that if the reward function for Model 2E is also altered to give a larger penalty for taking a long path then it would achieve similar results to Model 1B.

D. MODEL 3

Model 3, the model with a different action space than Model 1 and Model 2, was trained without the use of curriculum learning for 20,000 training steps. The results can be found in Figure 21.

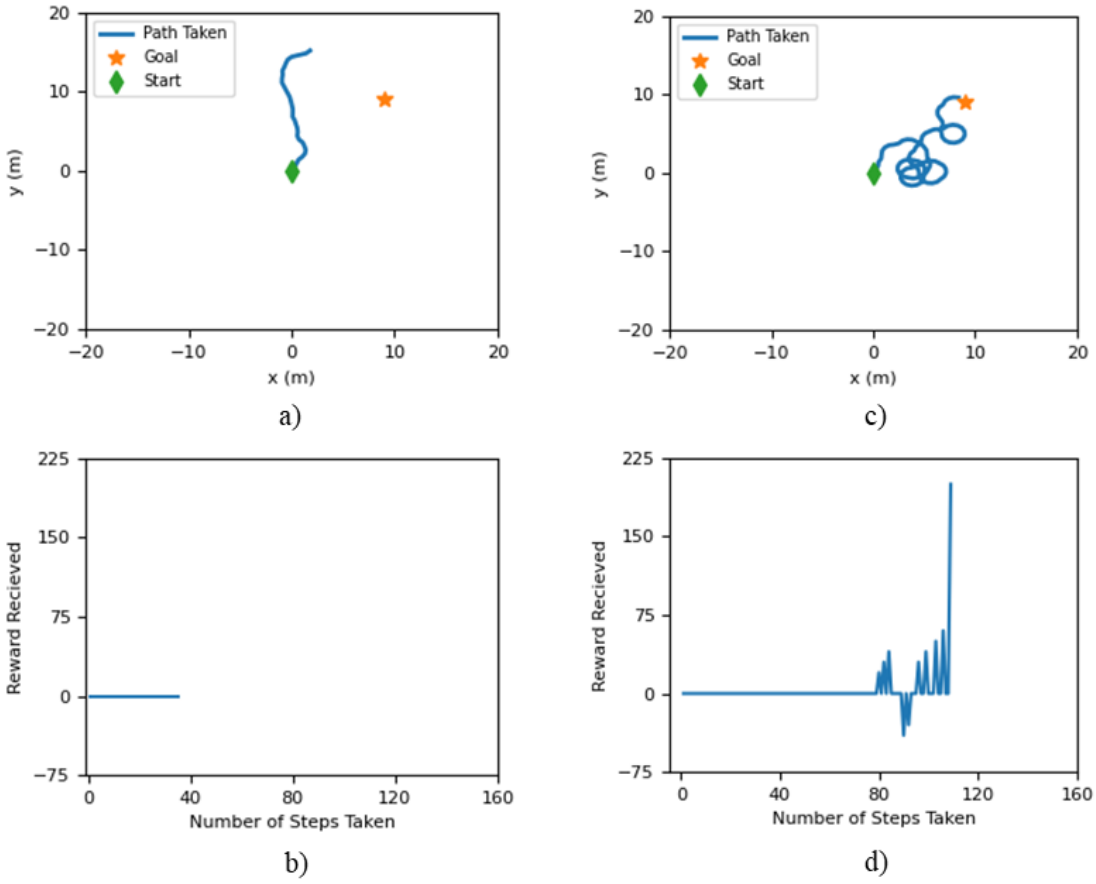

Figure 21. Results for Model 3 a) Before training b) Reward at each step before training c) After training d) Reward at each step after training

It is evident that prior to training the agent did not reach the goal. After training, the agent made it to the goal but did so by taking a longer than needed path. This could be

for several reasons. For one, 20,000 training steps may not have been enough to adequately train the agent. With time permitting, training for additional steps may result in a path that reaches the goal in fewer steps. Secondly, the -0.1 reward for each step may be insignificant compared to the 200 reward received for reaching the goal. The agent may not be properly incentivized to reach the goal in as few steps as possible because there isn't much difference between a long-term reward of 389 (received for the actual path taken), vs 397.6 (what would be received for a straight line). That is a 2.1% decrease in the overall reward for a 124.4% increase in the number of steps taken. Perhaps increasing the penalty for each step taken or decreasing the reward received for reaching the goal will give the agent more incentive to take the optimal path. This new incentive may reduce training time and allow the agent to be properly trained with the 20,000 training steps originally selected.

E. SUMMARY

Overall, it was determined that a reward function may be suitable for a particular environment, but become unsuitable after altering that environment even slightly, such as adding in a single obstacle. When the reward function is no longer suitable, training requirements can drastically increase, especially as the environment becomes more complex.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION AND FUTURE WORK

A. CONCLUSION

Automatically routing objects such as pipes, cables, tubes, and hoses without the use of reinforcement learning has been widely studied. However, research on using RL methods to automate these tasks has been studied limitedly. In this thesis, it was proposed that electrical cables could be routed automatically using the model-based RL algorithm MuZero. Specifically, the MuZero-General implementation of MuZero was used in conjunction with a custom-build gymnasium environment. The goal of this thesis was to provide proof of concept. From the models created and examined, it was determined that using RL to automate cable lay design is feasible but challenging. From Model 1, it was determined the reward function defined is sufficient for training a single-agent 2D model to find a path from a given start location to a given end location. In Model 2 an obstacle was added for additional complexity. After training Model 2, it was concluded that adding in what appears to be a simple constraint can significantly increase the training requirements and complicate the selection of a suitable reward function. From Model 3 it was found that increasing the complexity of the action space also results in increased training requirements. As a recommendation for reducing training requirements, the reward function could be updated to balance the -0.1 reward for taking each step and the 200 reward for reaching the goal. This imbalance resulted in a 2.1% decrease in the overall reward for a 124.4% increase in the number of steps taken. In Future work, there are many considerations that could improve and increase the complexity of the models, making it a practical solution to routing real-world electrical cables. Perhaps in the future, this method can be used to create an initial draft of a cable routing plan that can save engineering time and money during the cable lay design for a submarine.

B. FUTURE WORK

In future work, the minimum bend radius must be considered as a constraint for routing. This constraint is vital for creating a routing layout that can be implemented in the

real world. To quantify the curvature of the path taken by the agent, the curvature formula in Equation (4) can be used,

$$K = \frac{\left\|\overrightarrow{r'}(t) \times \overrightarrow{r''}(t)\right\|}{\left\|\overrightarrow{r'}(t)\right\|^3} \quad , \tag{4}$$

where K is the curvature and $\overrightarrow{r'}(t)$ and $\overrightarrow{r''}(t)$ are the first and second derivatives, respectively, of the position vector of the agent $\vec{r}(t)$. $\vec{r}(t)$ and its second and third derivative can be found in Equation (5), Equation (6), and Equation (7), respectively.

$$\vec{r}(t) = x\hat{\imath} + y\hat{\jmath} \tag{5}$$

$$\overrightarrow{r'}(t) = \dot{x}\hat{\imath} + \dot{y}\hat{\jmath} \tag{6}$$

$$\overrightarrow{r''}(t) = \ddot{x}\hat{\imath} + \ddot{y}\hat{\jmath} \tag{7}$$

Variables $\dot{x}$ and $\dot{y}$ can be approximated using the two-point forward difference method, which in this case is the change in position per step. The agent's change in position per step can be defined using the velocity,V, of the agent. The resulting equations for $\dot{x}$ and $\dot{y}$ are shown in Equation (8) and Equation (9),

$$\dot{x} \sim \frac{x_{i+1} - x_i}{h} = \mathrm{V}cos\theta_i \tag{8}$$

and

$$\dot{y} \sim \frac{x_{i+1} - x_i}{h} = \mathrm{V}sin\theta_i \quad , \tag{9}$$

where h is the step size and θ is the angle the agent is traveling at. Taking the derivative of $\dot{x}$ and $\dot{y}$ results in Equation (10) and Equation (11),

$$\ddot{x} = \mathrm{V}'cos\theta - \dot{\theta}\mathrm{V}sin\theta \tag{10}$$

and

$$\ddot{y} = \mathrm{V}'sin\theta + \dot{\theta}\mathrm{V}cos\theta \, , \tag{11}$$

where $\dot{\theta} = \omega$, the rate of change of the angle of the agent. ω is the parameter that the agent would alter through its actions. The resulting curvature equation that can be used to determine if the agent's choice for ω results in violating the minimum bend radius of the cable is shown in Equation (12).

$$K = \frac{\dot{\theta}V + 2\dot{V}sin\theta cos\theta}{V^2} \tag{12}$$

The minimum bend radius can be directly compared to $1/K$. The value of $1/K$ is the radius of the circle with curvature K. Therefore, if $1/K$ is smaller than the minimum bend radius of the cable, then the agent would be penalized in the reward function.

Another real-world concern that was not considered was the orientation at the end location. For instance, finishing at a 90° angle as opposed to a 45° angle.

A helpful addition to the model would be making it more generalized by starting each episode with a random start and end location. This would teach the agent how to find a path in any situation. If the model was generalized then it could be trained once and then applied to any cable way, regardless of the start and end locations.

Additionally, the model can be changed from single-agent to multi-agent to allow for the routing of more than one cable at a time. There are numerous cables contained within one cable way. Routing all the cables simultaneously would likely reduce the time it takes to converge on a conflict-free solution.

To increase the real-world similarities of the model, it can be converted from 2D space to 3D space. Instead of a path in only x and y coordinates, it is believed the path can be represented as a 3D point cloud. An example is shown in Figure 22.

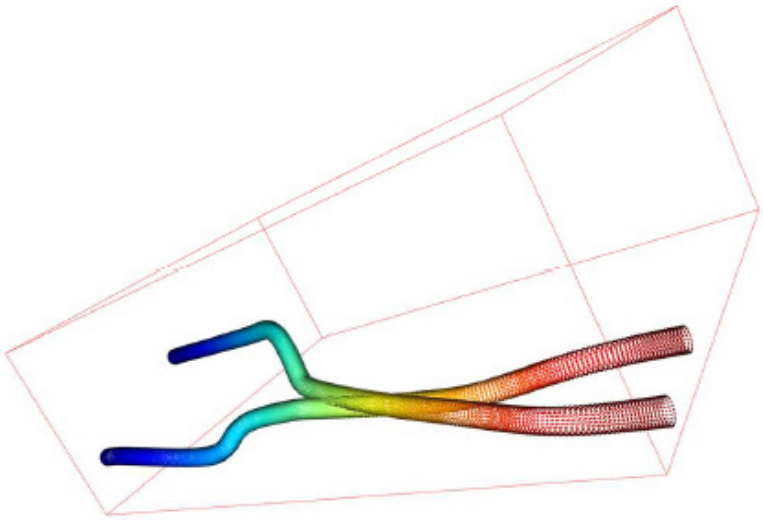

Figure 22. An example of representing an electrical cable in 3D. Source: [15]

The curvature of the path can be defined using a cubic Bézier curve. Points can then be filled in around the Bézier curve bounded by a specified radius to create a 3D object [15]. It is foreseen that detecting collisions between two cables each defined as a separate point cloud will be a difficult challenge. One proposed method is to convert the 3D point clouds into octrees and to use existing methods for detecting collisions between octrees [16]. Another future consideration is to use a 3D convolutional neural network such as an octree-based convolution neural network [17] in conjunction with the MuZero algorithm to create a uniform input size regardless of the number of cables in the environment.

LIST OF REFERENCES

[1] Eland Cables, "How is the minimum bending radius determined for cables?" Accessed: Apr. 2, 2023 [Online]. Available: https://www.elandcables.com/the-cable-lab/faqs/faq-how-is-the-minimum-bending-radius-determined-for-cables

[2] G. Belov, W. Du, M. G. De La Banda, D. Harabor, S. Koenig, and X. Wei, "From multi-agent pathfinding to 3D pipe routing," in *Thirteenth International Symposium on Combinatorial Search,* 2021 [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18530

[3] S. Kim, S. Kim, T. Choi, T. Kwon, T. Hee Lee, and K. Lee, "Automatic design system for generating routing layout of tubes, hoses, and cable harnesses in a commercial truck," *Computational Design and Engineering*, vol. 8, no. 4, pp. 1098–1114, Aug. 2021.

[4] Y. Ando and K. Hajime, "Automatic routing system for multiple pipe-lines," *Society of Naval Architects and Ocean Engineers*, vol. 20, pp. 221–230, Jan. 2014.

[5] S. Ueng and H. Huang, "A distance-field-based pipe-routing method," *Materials*, vol. 15, no. 5376, Aug. 2022 [Online]. Available: https://www.mdpi.com/1996-1944/15/15/5376

[6] Y. Kim, K. Lee, B. Nam, and Y Han, "Application of reinforcement learning based on curriculum learning for the pipe auto-routing of ships," *Computational Design and Engineering*, vol. 10, no. 1, pp. 318–328, Feb. 2023 [Online]. Available: https://academic.oup.com/jcde/article/10/1/318/6972374

[7] Santa Clara University, "9 real-life examples of reinforcement learning." Accessed: Mar. 15, 2023 [Online]. Available: https://onlinedegrees.scu.edu/media/blog/9-examples-of-reinforcement-learning

[8] S. Bhatt, "Reinforcement learning 101," Towards Data Science, Mar. 19, 2018 [Online]. Available: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[9] L. Smith, I. Kostrikov, and S. Levine, "A walk in the park: learning to walk in 20 minutes with model-free reinforcement learning," arXiv:2208.07860, Aug. 2022 [Online]. Available: https://arxiv.org/abs/2208.07860

[10] W. Duvaud, A. Hainaut, and P. Lenoir, "MuZero-General," GitHub, May 31, 2022 [Online]. Available: https://github.com/werner-duvaud/MuZero-general

[11] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. "Mastering Atari, Go, chess and Shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, Dec. 2020 [Online]. Available: https://doi.org/10.1038/s41586-020-03051-4

[12] J. Schrittwieser. "MuZero – ICAPS 2020." (October 6, 2020). Accessed: Apr 10, 2023 [Online Video]. Available: https://www.youtube.com/watch?v=L0A86LmH7Yw

[13] J. Schrittwieser, "MuZero intuition," Furidamu, Dec. 22, 2020 [Online]. Available: https://www.furidamu.org/blog/2020/12/22/MuZero-intuition/

[14] Gymnasium Documentation, "Gymnasium is a standard API for reinforcement learning, and a diverse collection of reference environments." Accessed Apr. 13, 2023 [Online] Available: https://gymnasium.farama.org/

[15] T. Rando, email, Jan. 2023

[16] B. Lucchesi, D. Egbert, and F. Harris, "A parallel linear octree collision detection algorithm," *IJCA*, vol. 21, no. 4, Dec. 2014.

[17] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D Shape Analysis", *ACM Transaction on Graphics*, vol. 36, no. 4, Jul. 2017.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California