ABSTRACT

Title of Thesis: LEARNING AUTONOMOUS UNDERWATER NAVIGATION WITH BEARING-ONLY DATA

James Robertson
Master of Science, 2024

Thesis Directed by: Professor Ramani Duraiswami
Department of Electrical and Computer Engineering

Recent applications of deep reinforcement learning in controlling maritime autonomous surface vessels have shown promise for integration into maritime transportation. These could have the potential to reduce at-sea incidents such as collisions and groundings which are majorly attributed to human error. With this in mind the goal of this work is to evaluate how well a similar deep reinforcement learning agent could perform the same task in submarines but using passive SONAR rather than the ranging data provided by active RADAR aboard surface vessels.

A simulated submarine outfitted with a passive spherical, hull-mounted SONAR sensor is placed into contact scenarios under the control of a reinforcement learning agent and directed to make its way to a navigational waypoint while avoiding interfering surface vessels. In order to see how this best translates to lower power autonomous vessels (vice warship submarines), no estimation for the range of the surface vessels is maintained in order to cut down on computing requirements. Inspired by my time aboard U.S. Navy submarines, the agent is provided with simply the simulated passive SONAR data.

I show that this agent is capable of navigating to a waypoint while avoiding crossing, overtaking, and head-on surface vessels and thus could provide a recommended course to a submarine contact management team in ample time since the maneuvers made by the agent are not instantaneous in contrast to the assumptions of traditional target tracking with bearing-only data.

Additionally, an in-progress plugin for Epic Games' Unreal Engine is presented with the ability to simulate underwater acoustics inside the 3D development software. Unreal Engine is a powerful 3D game engine that is incredibly flexible and capable of being integrated into many different forms of scientific research. This plugin could provide researchers with the ability to conduct useful simulations in intuitively designed 3D environments.

LEARNING AUTONOMOUS UNDERWATER NAVIGATION WITH BEARING-ONLY DATA

by

James Robertson

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2024

Advisory Committee:
Professor Ramani Duraiswami, Chair/Advisor
Professor Behtash Babadi, Co-Advisor
Professor Nirupam Roy, Committee Member

Acknowledgments

Firstly I would like to thank my family. My wife has done nothing but support me through my education here at UMD. Rachel and my two wonderful daughters, Peyton and Dallas, have graciously put up with long days without me around and my high levels of stress throughout my time at UMD and in the Navy.

I also must thank the Navy for allowing me to take the time between sea duties to continue my education and giving me the freedom to choose which college to attend to best suit the needs of my family as well as my educational goals.

I owe many thanks to Professor Ramani Duraiswami for introducing himself to me at a new student welcome event. Afterwards, he generously allowed me into his lab giving me guidance and assistance as well as a support network of fellow graduate students. I would also like to thank Professor Behtash Babadi first for teaching my favorite course I took at UMD introducing me to the field of machine learning and also for his advisor meetings with me continuing to provide assistance after I was out of his classroom even though I was not a member of his lab.

Finally, I would like to take the time to thank my fellow students especially Montee, Armin, Ryan, Phil, Meenakshi and Shoken. I am amazed at the intelligence, work ethic and enthusiam I encountered from all of you at UMD. All of you were there at one point when I needed to talk through a problem out loud, ask advice or just express some frustration. I would not have made it without a fantastic support system like the one you all provided.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASV	Autonomous Surface Vessel
AUV	Autonomous Underwater Vessel
BRG	Bearing
BRGRT	Bearing rate
COLREGS	International Regulations for Preventing Collisions at Sea
CPA	Closest Point of Approach
CR	Collision Risk
CRS	Course
DRM	Direction of Relative Motion
GUI	Graphical User Interface
LOS	Line of Sight
MDP	Markov Decision Process
MLE	Maximum Likelihood Estimation
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RADAR	Radio Detection and Ranging
RL	Reinforcement Learning
RNG	Range
RNGRT	Range rate
RNN	Recurrent Neural Network
SPD	Speed
SSP	Sound Speed Profile
SONAR	Sound Navigation and Ranging
TD	Temporal Difference
TGT	Target / Contact
TMA	Target Motion Analysis
TRPO	Trust Region Policy Optimization

Chapter 1: Introduction

1.1 Motivation

During my time in the US Naval submarine community I spent countless hours training on how to manage surface contacts while underwater with only passive SONAR detections. Submarines currently employ algorithms that estimate a contact's "solution". This solution consists of a range, speed, and course. We then utilize this information to select a course and speed to maintain CPA at an acceptable range. However, one thing we are taught very early on is that these generated solutions are "lies" and we need to be able to drive off the raw SONAR data.

Once I began my graduate studies at University of Maryland I quickly learned about fascinating new applications of machine learning and realized that the submarine fleet may be behind the curve on incorporating these new tools. I realized that employing a type of neural network could be extremely beneficial in the bearing-only TMA problem.

While thinking about what I wish I had onboard the submarine I recalled that if you are using the aforementioned algorithms to estimate the target solution you then had to use a separate program to trial new courses and/or speeds to drive. This can be time consuming especially for less experienced operators and can be made more challenging by other operational constraints (i.e. a navigational waypoint you are trying to reach, how much waterspace you are allowed to operate in, etc.). I decided I would want an algorithm that would inform the operator of what

course they should drive taking into account at minimum a navigational goal.

There were examples of algorithms for ASVs but for that methodology to work in the same way I would still need to use the more classical algorithms to estimate a full target solution since a RADAR onboard a surface vessel provides more information than passive SONAR. Recalling the ability of proficient operators to drive off the raw SONAR data I decided to explore a RL agent that took as input only information available through raw SONAR data and decide upon a course to drive.

1.2 Objectives

The goal of my research was to use Reinforcement Learning to accomplish navigation with bearing-only data without maintaining an estimate of the target's solution. For the purpose of this thesis a "solution" will be defined as the Course, Speed, and Range of a contact.

Additionally, I desired to create a realistic simulator for passive SONAR in both a 2D and 3D environment. These simulators could be used as a tool for future endeavors in the field of bearing-only target motion analysis. I wanted to not only simply calculate bearings between two points in a coordinate system but also use reliable simulations of acoustics in the ocean to provide even more data.

1.3 Outline of Thesis

In Chapter 2 background knowledge of how a submarine operates is presented including how sound propagates in the ocean and the traditional TMA algorithms used. Furthermore, there is discussion of the method by which this acoustic propagation is simulated. An introduction

to the general topic of RL is presented before looking at the specifics of Actor-Critic methods leading to the development of Proximal Policy Optimization. Finally, current work in the field of autonomous surface vehicles is presented providing some of the ideas leading to the implementation of this work.

In Chapter 3 the methodology used is presented first by discussing how an open ocean contact scenario is set up. In addition, my development of a plugin to introduce underwater acoustic simulations to Unreal Engine is presented. Afterwards, problem-specific RL terms are defined including the state space, action space and reward function.

In Chapter 4 the results from using a PPO agent on the RL environment described in Chapter 3 are presented.

Chapter 5 sums up the conclusions of this research and discusses possible future work to enhance or modify using RL as a tool for bearings only TMA.

Chapter 2: Background

2.1 Underwater Acoustic Propagation

Water as an acoustic waveguide differs from the air in several ways. Chiefly, sound speed(c) underwater is about 5 times the sound speed in air. Secondly, sound speed is normally related to density and compressibility but in the ocean it depends on temperature(T) in degrees centigrade, depth(z) in meters, and salinity(S) in parts per thousand of the water according to Equation 2.1 [1].

$$c = 1449.2 + 4.6T - 0.055T^2 + 0.00029T^3 + (1.34 - 0.01T)(S - 35) + 0.016z \qquad (2.1)$$

This equation is for a single point in a water volume while the ocean is a large volume with many factors. In the majority of the ocean temperature effects dominate the SSP for the first 1000 meters and afterward the effect of hydrostatic pressure due to depth dominates. The effect of the salinity term is minor unless there is an interface of large bodies of water mixing (i.e. Gibraltar).

A typical SSP in the ocean is shown in Figure 2.1 and displays different profiles near the surface that depend on how the temperature of the water is mixed prior to the thermocline. If the water surface is very warm the sound speed can drop quickly whereas with a mixed layer the sound speed can remain relatively constant before the thermocline unless the mixed layer results

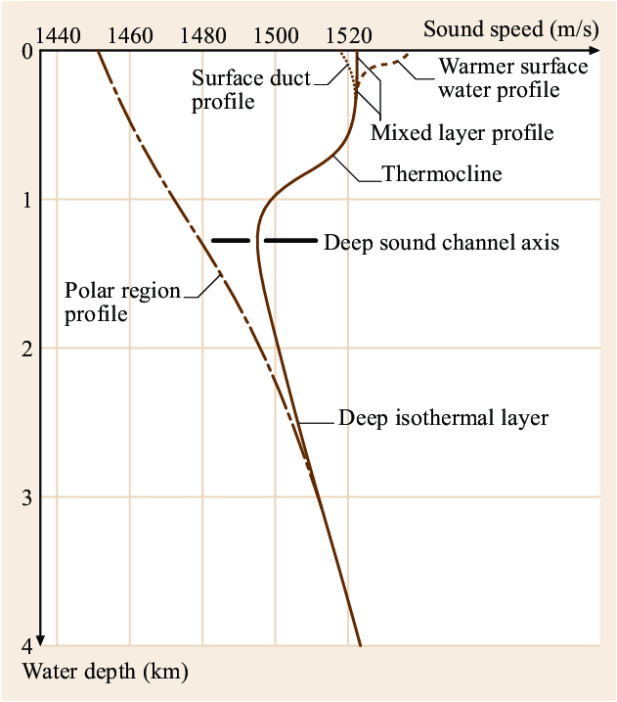


Figure 2.1: Generic SSP in Ocean

in a surface duct. If there is a surface duct then the sound speed is dominated by the changing depth since the temperature of the water remains constant. The resulting surface duct can trap sound inside it and this becomes particularly apparent when simulating broadband SONAR signals from surface ships to a submarine inside the surface duct as range decreases. Moving below the surface duct the changing temperature of the water dominates the SSP until reaching a deep isothermal layer. At this point beneath the thermocline the temperature of the water becomes constant at about 2°C due to thermodynamic properties of salt water at high pressure. The polar region is also shown. It is worth noting that as you travel closer to the poles the SSP shifts towards the polar region profile.

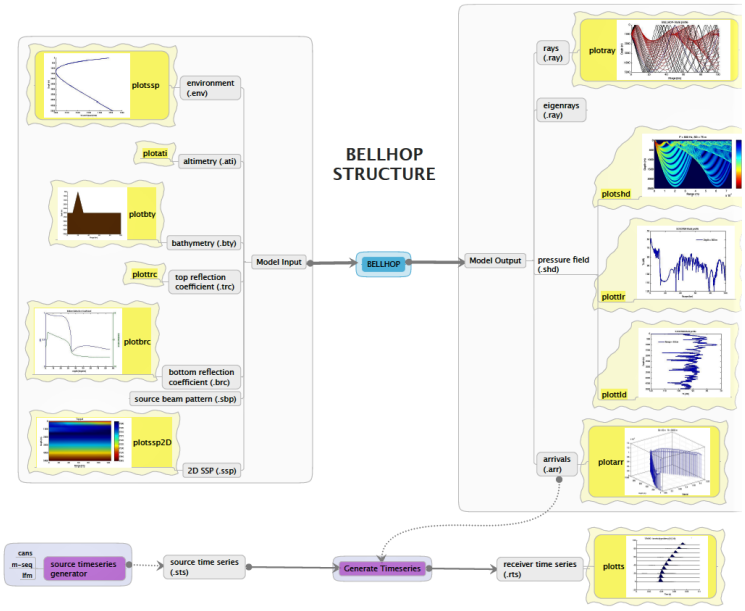


Figure 2.2: BELLHOP Structure [3]

2.1.1 BELLHOP

BELLHOP is one of four acoustic models included in the Acoustic Toolbox from Ocean Acoustics Library [2]. Specifically, it is a beam/ray trace code used in predicting acoustic pressure fields in the ocean and is attributed to Michael B. Porter. Information in this section (unless otherwise noted) was gleaned from the BELLHOP Manual and User's Guide [3] and the BELLHOP3D User Guide [4]. Figure 2.2 shows the structure of the BELLHOP program. In essence, you provide the program with various input files whose formats are delineated in these guides. These files contain information about the ocean environment including source/receiver depths and ranges, ocean floor bathymetry, and SSP. They also specify what the program's outputs for the run, i.e. arrivals, rays, pressure fields.

Nowadays, BELLHOP has various beam methods to perform its calculations. Originally, the beams were based off work in the field of seismology [5] and are termed 'Cerveny Beams'.

These Cerveny beams were in a ray-centered coordinate system (arclength s and normal distance from the ray n) and spread physically. BELLHOP also uses beams (including Cerveny) in a Cartesian coordinate system (range r and vertical distance from the ray δz). In addition, geometric spreading functions were implemented where the beams spread according to the expansion of a ray-tube. This led to the other beams currently available in the software listed below:

- **Geometric Hat-Beams**: The hat-shape (triangle) was inspired by finite-element shape functions [6]
- **Geometric Gaussian Beams**: The hat-shaped beams are replaced by Gaussian beams [7] while maintaining the geometric spreading function.
- **Geometric Hat-Beams in Cartesian Coordinates**: The results here are essentially identical to the implementation in the ray-centered coordinate system but provide better efficiency.

2.1.2 BELLHOP Arrivals

BELLHOP's use in this work focuses on its ability to produce arrivals. You can specify that you want arrivals calculated in the input environment file to the program which will result in a .arr output text file. BELLHOP iterates over each source specified and performs the calculations for all the receivers delineated in the environment file. Multiple receivers are specified with their range (in 2D) or range and bearing (in 3D) [4].

For each receiver delineated by the user in the environment file the arrivals are calculated. The 'arrival information' is in a text file and an example of the information contained is displayed in Table 2.1. The 'SrcDeclAngle' and 'RcvrDeclAngle' refers to the angle of departure and angle of arrival, respectively, in the vertical plane from a line parallel to the ocean surface. The

Frequency (Hz)						50
Source Depth (m)						5
Receiver Depth (m)						150
Range (m)						10050
Amplitude	Phase	Time Delay	SrcDeclAngle	RcvrDeclAngle	NTopBnc	NBotBnc
1.973E-06	0	9.258	43.0	-44.015	0	1
1.900E-06	180	9.392	44.5	45.232	1	1

Table 2.1: Example BELLHOP Arrival Information

'NTopBnc' correlates with the number of times the sound ray bounced off the surface of the ocean while 'NBotBnc' refers to the number of times the ray bounced off the ocean floor. The same information would be printed to the output file for each receiver specified before moving onto the next source if applicable.

For a passive SONAR sensor all this information would obviously not be available to the receiver. If it were then there would be no need to perform TMA in order to estimate range. There is no way to know the number of top and bottom bounces, the time delay, or the declination angle that the sound left the source. The remaining data (amplitude, phase, receiver declination angle) as well as the bearing of the arrival would be available and can be used to conduct TMA.

2.1.3 Beam Tracing

Beam tracing is a foundation of the BELLHOP program and is done in both two- and three-dimensions with the mathematics presented in BELLHOP3D user's guide [4] as well as a separate entry from Porter in The Journal of the Acoustical Society of America [8]. The beams

mentioned in the previous section are each built around a central ray that satisfies [3] [1]

$$\frac{dr}{ds} = c\xi(s), \frac{d\xi}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial r} \tag{2.2}$$

$$\frac{dz}{ds} = c\zeta(s), \frac{d\zeta}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial z} \tag{2.3}$$

where $c(r, z)$ is the sound speed and $[r(s), z(s)]$ is the ray trajectory as a function of arclength, s. Hence we can also write the soundspeed, c, as a function of s. The ray starts at some position (r_0, z_0) and has a specified takeoff angle, $\alpha \in [-90°, +90°]$. α is varied with discrete launch angles. In BELLHOP you can specify a narrower range of takeoff angles than $[-90°, +90°]$ and the number of discrete launch angles to simulate. For each α

$$r = r_0, \xi = \frac{\cos\alpha}{c(0)} \tag{2.4}$$

$$z = z_0, \zeta = \frac{\sin\alpha}{c(0)} \tag{2.5}$$

The central ray has a pressure field defined as a function of s and the normal distance from the receiver to the central ray of the beam, $n(s)$.

$$P(s, n) = A(s)\phi(s, n)e^{-i\omega\tau(s)} \tag{2.6}$$

where ω is angular frequency and $\tau(s) = \int_0^s \frac{1}{c(s')}ds'$ is the phase delay. The functions $A(s)$ and $\phi(s, n)$ vary depending on which type of beams are used.

The rays also satisfy the dynamic ray equations given by 2.7 and 2.8 where c_{nn} is the derivative of the sound speed in a direction normal to the path of the ray given by Equation 2.9.

$\Delta\alpha$ is the spacing between rays at the origin and r is the cylyndrical range.

$$\frac{dq}{ds} = cp(s) \tag{2.7}$$

$$\frac{dp}{ds} = -\frac{c_{nn}}{c^2(s)}q(s) \tag{2.8}$$

$$c_{nn} = c^2(\frac{\partial^2 c}{\partial r^2}\zeta^2 - 2\frac{\partial^2 c}{\partial r \partial z}\zeta\xi + \frac{\partial^2 c}{\partial z^2}\xi^2) \tag{2.9}$$

2.10 sets up the initial conditions of the dynamic ray equations for geometric beams. Equation 2.12 is the amplitude of a geometric beam and the hat-shaped beam is constructed via 2.11.

$$q(0) = 0, p(0) = 1 \tag{2.10}$$

$$\phi(s,n) = \begin{cases} \frac{W(s)-n}{W(s)} \text{ for } n \leq W(s) \\ \\ 0 \text{ otherwise} \end{cases} \tag{2.11}$$

$$A(s) = \sqrt{\frac{cos\alpha}{r}\frac{c(s)}{q(s)}} \tag{2.12}$$

Equations 2.13 and 2.15 are for geometric gaussian beams. The scaling factor of $1/\sqrt{2\pi}$ in the amplitude function is taken from the integral of the Gaussian beams in Equation 2.14.

$$\phi(s,n) = e^{-1/2(n/W(s))^2} \tag{2.13}$$

$$\frac{1}{c}\int_{-\infty}^{\infty} e^{-1/2(n/c)^2} dn = \sqrt{2\pi} \tag{2.14}$$

$$A(s) = \frac{1}{\sqrt{2\pi}}\sqrt{\frac{\cos\alpha}{r}\frac{c(s)}{q(s)}} \tag{2.15}$$

Defining the interval between takeoff angles as $\Delta\alpha$ the formula for the geometric width of the beam, $W(s)$, is given Equation 2.16 and does not change for Gaussian beam calculations.

$$W(s) = |\frac{q(s)\Delta\alpha}{c(0)}| \tag{2.16}$$

where the $q(s)$ is the ray-tube spreading function.

These formulations produce the results received from BELLHOP. Once the data regarding the sound's arrival to the receiver is calculated it is then up to an algorithm to estimate other information about the source, i.e. the range, utilizing that data.

2.2 Bearing-Only Target Motion Analysis

Due to the nature of passive SONAR it was necessary to develop a way to interpret data consisting only of bearings from own ship in order to estimate the range to the target which subsequently can be used to determine course and speed with a range timeseries. The methods in this chapter are presented in the context of underwater acoustics but can be used in other applications such as passive RADAR that detects other RADARs but does not emit any radiowaves. These passive methods allow you to track a target without giving away your position. The

notation, definitions and mathematical concepts of this section are taken from [9] unless otherwise delineated.

Firstly the following definitions and notation are established with time instants $t_0, t_1, \ldots$ and the subscripts $i, j = 0, 1, \ldots$ refer to their corresponding time instant and a double subscript ij refers to a change from $t_i \to t_j$.

- $u :=$ target speed, $c :=$ target true course relative to 000
- $b_i :=$ bearings from the receiver to target at t_i, $r_i :=$ range at t_i
- $t_{ij} = t_j - t_i$, $b_{ij} = b_j - b_i$
- $DOA_{ij} :=$ distance the receiver moves from $t_i \to t_j$ across the line of sight meaning perpendicular to b_i
- $DOI_{ij} :=$ distance the receiver moves from $t_i \to t_j$ into the line of sight meaning in the direction of b_i
- DTA_{ij} and DTI_{ij} are the same defintions but for the target's distance from $t_i \to t_j$

The change in bearings, b_{ij}, is measured from -180°to +180°with a clockwise change being positive. Another important definition is the direction of relative motion (DRM) which is the target's direction relative to the receiver when the receiver is on a linear track (see Figure 2.3) This differs from target relative course in that it is measured relative to true north instead of the receiver's course.

Assuming the receiver has a linear track and there is no error in bearing measurements, obtaining bearings at 3 distinct times will allow you to calculate all bearings by giving you DRM.

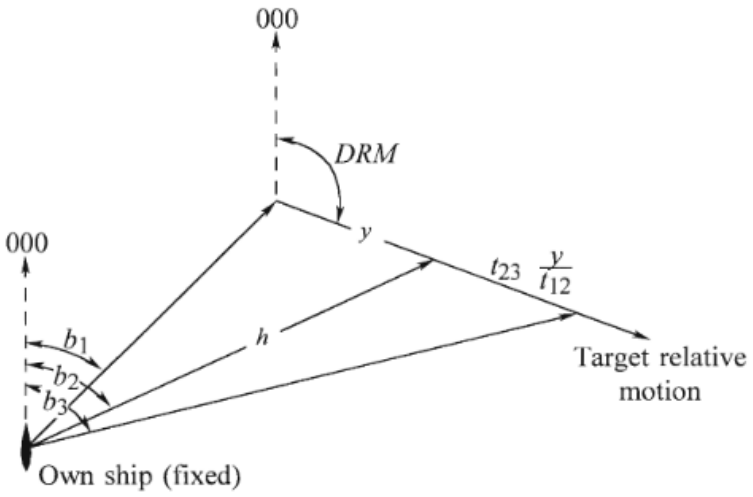


Figure 2.3: Direction of Relative Motion [9]

Furthermore, the course and speed of the receiver are irrelevant to estimating solution again with the caveat that a linear track is maintained. Importantly, you cannot obtain a definitive solution (range, speed, course) from bearings only on a linear track. There is not a unique solution that produces the observed bearing history. In order to determine the best solution the receiver must perform a maneuver or use some other ranging technique such as a triangulation range if a second sensor is available. The TMA problem is made more complex in that the above does not account for sensor error. It is true that 3 distinct bearings provide you with DRM; however, this is assuming that you have obtained the true bearing to the contact. If the 3 bearings have changed significantly more than the measurement error then the DRM will be of high confidence. On the other hand if the 3 bearings obtained have a small change in bearing more observations would be needed to know what the contact will do once range is closed.

Referring to Figure 2.3 we solve for DRM as follows with t_1, t_2, t_3 as distinct times and uniformly spaced ($t_{23} = t_{12}$)

$$t_{23}\cot(DRM - b_1) = t_{13}\cot b_{13} - t_{12}\cot b_{12} \tag{2.17}$$

$$\cot(DRM - b_1) = 2\cot b_{13} - \cot b_{12} \tag{2.18}$$

Solving Equation 2.18 or 2.17 (if the times are not uniformly spaced) will yield two values. To establish the true DRM add or subtract a multiple of 180 to each so that they lie between 0 and 360. Whichever of the 2 values is on the side of b_1 where the bearings are drawing is the DRM. We can similarly obtain the bearing at CPA, b_0, with Equation 2.19

$$t_{23}\tan b_{01} = t_{13}\cot b_{13} - t_{12}\cot b_{12} \tag{2.19}$$

2.2.1 Maximum Likelihood Estimation

MLE is a common way to conduct bearing-only TMA. It is useful to a human operator in that it can show different possible solutions and provide an estimate for how "good" they are usually in the form of a contour plot. MLE evaluates numerous solutions for a starting range and a current range of the contact by calculating the course and speed necessary for the contact to meet those ranges. Subsequently, the bearings that would have been received on this proposed track are calculated and compared to the actual received bearings. The differences at each timestep are evaluated and the contour plot displays how good a solution is for all the possible staring and current ranges. These starting and current ranges can be displayed as the $x-$ and $y-$axis or converted to a (x, y) position relative to the sensor as shown in Figure 2.4 where 2 sensors are used.

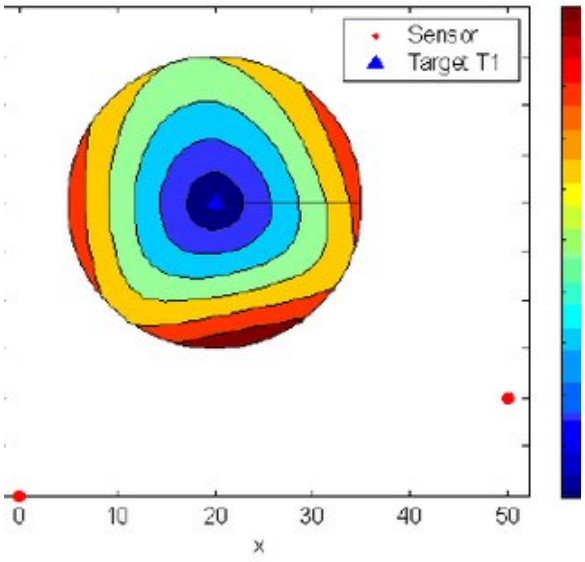


Figure 2.4: Example contour plot conducting bearing-only TMA with MLE using 2 sensors [10]

To estimate the current position of the target take our initial position $x_{OS}^{(0)}$ and $y_{OS}^{(0)}$, the initial bearing observed, $b_{obs}^{(0)}$, as well as the initial range of the target, $r_{TGT}^{(0)}$ and calculate the initial position of the target, $x_{TGT}^{(0)}$ and $y_{TGT}^{(0)}$ according to Equations 2.20 and 2.21. Afterwards, knowing the target's course, CRS_{TGT}, and speed, SPD_{TGT}, we can calculate the target position at time t via Equations 2.22 and 2.23.

$$x_{TGT}^{(0)} = x_{OS}^{(0)} + \sin(b_{obs}^{(0)}) r_{TGT}^{(0)} \tag{2.20}$$

$$y_{TGT}^{(0)} = y_{OS}^{(0)} + \cos(b_{obs}^{(0)}) r_{TGT}^{(0)} \tag{2.21}$$

$$x_{TGT}^{(t)} = x_{TGT}^{(0)} + SPD_{TGT} \sin(CRS_{TGT}) t \tag{2.22}$$

$$y_{TGT}^{(t)} = y_{TGT}^{(0)} + SPD_{TGT} \cos(CRS_{TGT}) t \tag{2.23}$$

Initial position can be reset at any time to treat t as a change in time from the reference

position. We can perform this analysis on different course and speeds and generate the estimated bearings, $b_{est}^{(t)}$ according to Equation 2.24.

$$b_{est}^{(t)} = \arctan(\frac{x_{TGT}^{(t)} - x_{OS}^{(t)}}{y_{TGT}^{(t)} - y_{OS}^{(t)}}) \tag{2.24}$$

We can then define a cost function between the vector of estimated bearings and the vector of received bearings, for example the Euclidean distance between the 2. The cost function for each trial course and speed would be displayed on the contour plot.

As mentioned before, MLE is beneficial for a human user since the contour plot can be designed to be interactive so that clicking on a position in the contour plot displays the estimated bearings and can be overlaid onto the actual SONAR screen. However, it is computationally expensive compared to recursive solutions and may not be the ideal method for AUVs. In addition, the choice of the initial time is crucial and requires human intervention. If a contact maneuvers in the middle of the time history the algorithm is relying on completely useless information from the bearings prior to the maneuver in the cost function calculation.

2.2.2 Kalman Filtering

The Kalman Filter [11] was a major breakthrough in developing a recursive solution to linear filtering problems but it is challenging to apply it to nonlinear problems such as TMA. Other nonlinear systems make use of the extended Kalman Filter which approximates nonlinear functions as linear but it is only reliable on nearly linear systems [12]. Another downside to using a Kalman filter variant on the TMA problem is that, unlike MLE, it provides a single solution. This makes the process faster but provides less flexibility to a human operator.

Early applications of the Kalman filter on bearing-only TMA using the Cartesian coordinate representation were very unstable due to the interaction of bearing and range estimation errors [13]. In order to combat the filter instability attempts were made on using modified polar coordinates [14, 15]. Cartesian coordinates initially appeared attractive since the state equations of the system (Equation 2.25) are intuitive, linear, and partially decoupled. x and y represent the relative position of the target with v_x and v_y as the relative velocity of the target and u_x, u_y as the components of the receiver's acceleration.

$$\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{2.25}$$

As you can see the observable quantity, bearing, is not found in Equation 2.25 and it cannot even be calculated with a linear combination of the state variables. Instead, it is calculated via $\arctan$ in the same way as Equation 2.24. In order to create the filter problem in a modified polar coordinate system we must map 2.25 into polar coordinate form and then solve for the desired θ, θ' and θ'' which are the bearing of the target as well as the first and second derivatives of the bearing. Doing so also involves the first and second derivatives of the range, $r = \sqrt{x^2 + y^2}$, and results in Equation 2.26 where u_c and u_d are the crossrange and downrange acceleration of the receiver, respectively.

$$\begin{bmatrix} 2r'\theta' + r\theta'' \\ r'' - r\theta'^2 \end{bmatrix} = - \begin{bmatrix} u_c \\ u_d \end{bmatrix} \tag{2.26}$$

Solving for an observable quantity, θ'', yields 2.27 and then if the receiver is not accelerating we can have a new observable quantity shown in 2.28 and redesignate this quantity, r'/r, as ρ'

$$\theta'' = -\frac{2r'\theta'}{r} - \frac{u_c}{r} \tag{2.27}$$

$$\frac{r'}{r} = -\frac{\theta''}{2\theta'} \tag{2.28}$$

Defining $s = \frac{1}{r}$ we can define the modified polar coordinates state equations 2.29. Note that while now the measurement equation is strictly linear the state equations are not; therefore, we have not removed the nonlinearity of the problem so an extended Kalman filter (or other Kalman filter variation designed to compensate for nonlinearity) is still required. The properties still differ from the Cartesian system since now covariance linearization errors are independent of range errors over ownship maneuvers [14].

$$\begin{bmatrix} \theta' \\ \theta'' \\ \rho'' \\ s' \end{bmatrix} = \begin{bmatrix} \theta' \\ -2\rho'\theta' \\ \theta'^2 - \rho'^2 \\ -\rho' s \end{bmatrix} + \begin{bmatrix} 0 \\ -su_c \\ -su_d \\ 0 \end{bmatrix} \tag{2.29}$$

Extended Kalman filters using this modified polar coordinate system have been shown to outperform those with Cartesian systems [16]. However, further work on other Kalman filter variations such as the unscented Kalman filter [12] have shown that a Cartesian system filter can do better than a modified polar coordinate system [17].

2.3 Reinforcement Learning

Reinforcement Learning is a branch of machine learning that differs in that the algorithm is expected to maximize a numerical reward value rather than perform more traditional generative or discriminative tasks. The algorithm is not provided any information about which action to take. Usually, each action is initialized with equal probability of being chosen at each step. Gradually, these probabilities are shifted based on the reward received throughout training. In summary, RL "is much more focused on goal-directed learning from interaction than are other approaches to machine learning." [18]

Let us define the agent in a RL application as done in the 2 main sources for this section, [18] and [19], as the learner and decision maker. This agent interacts with its environment which is everything in the problem outside the agent. How the environment is represented to the agent can be chosen to provide key details and information that influence the agent's decision making.

2.3.1 Markov Decision Process

We can detail the interactions between an agent and environment as an infinite horizon, discounted Markov Decision Process (MDP), $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$. Using the definitons of these variables as put forth in [19]

- $\mathcal{S}$:= state space which is the information/details decided upon to be presented to the agent. This can be finite or infinite. For this thesis the state space will be finite or countably infinite.

- $\mathcal{A}$:= action space. This action space can also be finite (discrete) or infinite. This thesis

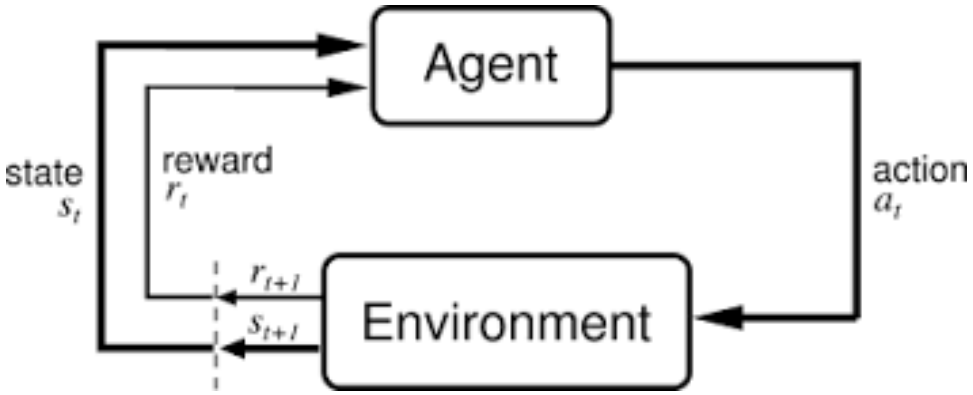


Figure 2.5: Agent and Environment Interaction

will assume a discrete action space.

- $P :=$ a transition function which denotes the probability of transitioning to another state, s', by taking action a in current state s, i.e. $P(s' \mid s, a)$.
- $r :=$ is the reward function. In general the reward function could be a random variable whose distribution depends on (s, a). The reward function could be deterministic or stochastic.
- $\gamma :=$ discount factor which defines a horizon for problem, $0 \leq \gamma \leq 1$
- $\mu :=$ an initial state distribution

A simple interaction in a MDP is shown in Figure 2.5. At each time step, t, the agent gets some information from the environment represented by $s^{(t)} \in \mathcal{S}$ and subsequently selects an action, $a^{(t)} \in \mathcal{A}$. At time step $t+1$ the agent receives a reward, $r^{(t+1)}$, based on the user-defined reward function. This reward function does not need to be explicitly defined in the terms of the state space provided to the agent. There could be variables in the environment hidden to the agent that affect the desired outcome. An example of this is the context of this thesis' navigational problem where the range to a contact is unknown explicitly but needs to be kept outside a certain

value using bearing only information.

In RL we focus on the rewards received after a timestep t and denote this sequence of rewards as $r^{(t+1)}, r^{(t+2)}, r^{(t+3)} \ldots$. Generally, the objective will be to maximize the expectation of this sequence's return, $G^{(t)}$. The simplest form of the return will be a summation of the rewards as shown in Equation 2.30 where T is the final timestep.

$$G^{(t)} = r^{(t)} + r^{(t+1)} + r^{(t+2)} + \cdots + r^{(T)} \tag{2.30}$$

In applications like the focus of this thesis we break up the interactions into episodes. Episodes in general are some repeated interaction, i.e. navigating to the goal, playing through a game of chess. The episode ends in a terminal state before the environment is reset. The reset of the environment is independent of the terminal state from the previous episode. The reset could be going back to the same starting chess board every time or the environment could have multiple starting points in which case the reset is randomly selected from these based on some user-defined distribution. The time of termination, T, can change for each episode; however, there could also be a time limit enforced on the agent that ends an episode even if a terminal state is not reached such as in Proximal Policy Optimization which will be discussed in detail in a later section.

The discount factor, γ, is used to discount the sum of future rewards. The idea is that due to the nature of the problem the calculation of expectation should be weighed more heavily while close to the current timestep. This is especially important in infinite-horizon problems and is termed the expected discounted return:

$$G^{(t)} = r^{(t+1)} + \gamma r^{(t+2)} + \gamma^2 r^{(t+3)} + \cdots = \sum_{k=0}^{\infty} \gamma^k r^{(t+k+1)} \tag{2.31}$$

Provided that $\gamma < 1$, the sum in 2.31 is bounded as long as r is bounded at each timestep. In the case that $r^{(t)} \leq 1 \forall t$ this upper bound is $\frac{1}{1-\gamma}$. As γ approaches 1 the agent takes future timesteps more into account.

An agent in RL is expected to produce a policy, π. Policies in RL can come in different forms. The aforementioned actions, states, and rewards can be stored in a trajectory. A learned policy could be based off a subset of that trajectory whether that be the last k time steps or a random sampling of the trajectory. Conversely, a stationary policy is one that is based off the last time step only. Going further the policy can be deterministic or stochastic. In a deterministic policy there is a chosen action depending on the state space or chosen trajectory while a stochastic policy has a probability of choosing an action given the state s. Accordingly, deterministic policies are represented as $\pi(s)$ while stochastic are represented as $\pi(a \mid s)$. As expected the following equation holds in a stochastic policy $\forall s \in \mathcal{S}$

$$\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1$$

In the event that the environment is a POMDP a stochastic policy can be expected to produce a higher average return than a deterministic policy [20].

In order to numerically define how "good" a policy is we define value functions. The first, $V_M^\pi : \mathcal{S} \to \mathbb{R}$, measures how good it is to be in a given state regardless of the action taken.

$$V_M^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid \pi, s_0 = s] \tag{2.32}$$

This mathematically defines the value function as the discounted sum of future rewards.

Secondly, we have the action-value function more commonly known as the Q-value function, $Q_M^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which measures how good an action is given the current state.

$$Q_M^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid \pi, s_0 = s, a_0 = a] \tag{2.33}$$

2.3.2 Bellman Equations

The learner attempts to find an optimal policy, π^*, which solves one of the following optimization problems

$$\max_\pi V^\pi(s)$$

$$\max_\pi Q^\pi(s,a)$$

In order to find this optimal policy the expected return needs to be calculated and thus require the use of the Bellman equations [21]. The Bellman equations were instrumental in the field of dynamic programming and have been adapted for RL. The Bellman equation for the value

function is derived as follows

$$
\begin{aligned}
V_M^\pi(s) &= \mathbb{E}[\sum_{k=t+1}^{\infty} \gamma^{k-t-1} r(s^{(k)}, a^{(k)}) \mid \pi, s^{(t)} = s] \\
&= \mathbb{E}[r^{(t+1)} + \gamma \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r^{(k)} \mid \pi, s^{(t)} = s] \\
&= \sum_{a \in \mathcal{A}} \pi(a, s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a)(r(s', a) + \gamma \mathbb{E}[\sum_{k=t+1}^{\infty} \gamma^{k-t-1} r^{(k)} \mid \pi, s_t = s']) \\
&= \sum_{a \in \mathcal{A}} \pi(a, s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a)(r(s', a) + \gamma V_M^\pi(s'))
\end{aligned}
$$

Similarly a derivation of the Bellman Equation for the Q-value function is the following

$$
Q_M^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a)(r(s', a) + \gamma \max_{a'} Q_M^\pi(s', a'))
$$

In order to solve these equations the dynamics of the environment need to be known since it uses P and r. This is typically not the case; therefore, techniques such as Monte Carlo methods are needed or approximating the value functions with some parameterized function. The latter is commonly accomplished using a deep neural network.

2.3.3 Temporal Difference Learning

Temporal Difference (TD) learning is a model-free RL technique that combine ideas from Monte Carlo methods and dynamic programming. It is model-free like Monte Carlo methods and updates based partly on other learned estimates like dynamic programming [22]. In the simplest TD learning method the update is shown in 2.34 after receiving reward $r^{(t+1)}$ and estimate of the

value function of the next state $V(s^{(t+1)})$.

$$V(s^{(t)}) = V(s^{(t)}) + \alpha[r^{(t+1)} + \gamma V(s^{(t+1)}) - V(s^{(t)})] \tag{2.34}$$

This is known as the one-step TD since it updates the value function estimation based only on the next state whereas the n-step TD updates based on the next n states discounted. The quantity, $r^{(t+1)} + \gamma V(s^{(t+1)}) - V(s^{(t)})$, is known as the TD error designated $\delta^{(t)}$. Applying TD error to the Q-value function led to the development of Q-learning [23] which was an early breakthrough in the field of RL. The update function of this algorithm is 2.35

$$Q(s^{(t)}, a^{(t)}) = Q(s^{(t)}, a^{(t)}) + \alpha[r^{(t+1)} + \gamma \max_{a} Q(s^{(t+1)}, a) - Q(s^{(t)}, a^{(t)})] \tag{2.35}$$

Q-learning is a strong RL algorithm; however, it can only be applied when the action space is discrete. For continuous action spaces a policy gradient method becomes necessary which led to the development of actor-critic methods.

2.3.4 Actor-Critic Methods

Critic-only methods approximate the value functions as discussed in the previous section through an approximation architecture. One of the downsides of these methods is that even if the value functions are accurately approximated there is not a reliable guarantee that the resulting policy will be optimal or near-optimal. Actor-only methods [24, 25, 26, 27] develop a parameterized set of policies. Through simulation, the performance is estimated with regards to the policy parameters allowing for a gradient update method to change the policy parameters.

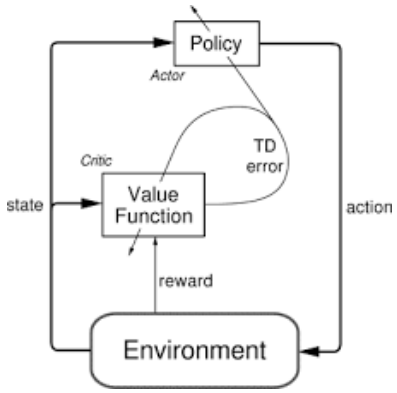


Figure 2.6: Actor-Critic Method

Actor-Critic methods [28] are designed to combine the benefits of the actor-only and critic-only based methods. The ”critic” approximates the value function with some parameterized model (i.e. a deep neural network). This approximation is then used to update the policy parameters of the ”actor”. These methods have reliable convergence properties as opposed to critic-only methods where convergence is guaranteed only in quite limited settings. In addition, they reduce the variance of the policy compared to actor-only methods which can yield faster convergence [29]. A diagram of the Actor-Critic architecture is shown in Figure 2.6.

2.3.5 Proximal Policy Optimization

In 2017 Schulman et. al. introduced proximal policy optimization (PPO) [30], an actor-critic method that promised to have the benefits of another method called trust region policy optimization (TRPO) [31] while having a simpler implementation and thereby scale better to higher dimensions.

The idea of both TRPO and PPO is to constrain the policy update. In TRPO this involves calculating the Kullback-Leibler divergence of the proposed updated policy and the current policy. This divergence calculation is the core of the reasoning that TRPO does not scale to higher

dimensions. The objective function (2.36 and 2.37) is termed the "surrogate" objective and involves the old and new policy parameters and the current estimate of the advantage function, $\hat{A}^{(t)}$

$$\max_{\theta} \mathbb{E}[\frac{\pi_\theta(a^{(t)} \mid s^{(t)})}{\pi_{\theta_{\text{old}}}(a^{(t)} \mid s^{(t)})}\hat{A}^{(t)}] \tag{2.36}$$

$$\text{subject to } \mathbb{E}[\mathbf{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s^{(t)}), \pi_\theta(\cdot \mid s^{(t)})]] \leq \delta \tag{2.37}$$

PPO proposes to simplify the surrogate objective by designating a hyperparameter, ϵ, in order to "clip" the policy update. Designating the probability ratio $r^{(t)}(\theta) = \frac{\pi_\theta(a^{(t)}|s^{(t)})}{\pi_{\theta_{\text{old}}}(a^{(t)}|s^{(t)})}$ the main objective proposed in [30] is Equation 2.38

$$L^{CLIP}(\theta) = \mathbb{E}^{(t)}[\min(r^{(t)}(\theta)\hat{A}^{(t)}, \text{clip}(r^{(t)}(\theta), 1-\epsilon, 1+\epsilon)\hat{(A)}^{(t)})] \tag{2.38}$$

This results in the final objective being a lower bound on the unclipped objective. PPO has become widely used in many RL applications which speaks to its ability to handle a wide range of tasking. It has also been extended to coordinate between multiple agents [32] and to adaptively adjust the hyperparameter ϵ within a trust region [33].

2.4 Autonomous Surface Vessels

In recent years autonomous navigation has been rapidly developing especially using RL algorithms. Maritime autonomous navigation is desirable since the majority of collisions at sea are caused by human decision making errors [34]. Initially in the 1980s a dynamic programming approach was explored [35] as well as a fuzzy controller [36]. Both of these methods laid

the groundwork for incorporating the International Regulations for Preventing Collisions at Sea (COLREGS) and defining a collision risk (CR). CR uses both distance and time to the closest point of approach to time the ship maneuvers. These early methods used speed and heading angle as action spaces. Other methods developed included utilizing particle swarm optimization [37], velocity obstacles [38], and the rudder angle action space in place of heading angle [39].

Recently, RL has taken the forefront in autonomous maritime navigation first with Q-learning [40, 41, 42] and then with various deep learning algorithms such as Deep Q-learning [43] and PPO that included a virtual sensor [44]. Limitations of Q-learning and Deep Q-learning included restricting the action space to discrete only vice continuous. As these methods continued to develop it led to the introduction of a recurrent neural network (RNN) in continuous action spaces using PPO [45].

The COLREGS contain a definition for risk of collision and so these models need to understand the rules in order to operate safely with human piloted vessels. If you are not in a collision risk maneuvering not according to some navigational track can cause confusion to other vessels. The time component is important as well; it is not advisable to maneuver to ”avoid collision” when the collision would not occur for say, over an hour. This time to CPA also may be a more reliable indicator of the danger than range due to the fact that ship’s geometry changes the range at which you should be concerned. The simplest example of this would be a head-on situation where all of both ships’ speed are directly at one another.

An advantage to making autonomous submerged vessels is that the requirements of the COLREGS no longer apply. Since the surface ships can presumably not see or detect you they have no reason to know you are there. This allows a RL agent of a submerged vessel more autonomy than its surface counterparts. It can maneuver early to avoid collision, gather more

bearing data to more accurately assess the other vessel's movement, or simply for navigational purposes.

Chapter 3: Methodology

3.1 Simulating an Open Ocean Contact Scenario

In order to accomplish any RL task the first thing to do was to create a navigational problem. The basic idea was to set up ownship (the submarine) in a coordinate system and surface contacts that ownship had to avoid. I based the scenarios I used on the Imazu problem [46]. Some of these were not especially useful for a submarine since the starting ranges were very low and not for an open ocean type scenario. The Imazu scenarios are shown in Figure 3.1 while my chosen scenarios generated in pygame [47] are shown in Figure 3.2. In order to maintain a level of randomness and ensure the agent didn't just learn a path to take that would avoid all scenarios the courses and speeds were randomly selected in a range that still met the overall situation the ship was in (i.e., crossing, overtaking, etc.) Furthermore, each of the 4 scenarios were flipped on the vertical axis randomly. North was chosen as the direction for the waypoint from the starting position for simplicity. In real-world scenarios the coordinate system could be easily set up to treat the starting direction of the next waypoint as North and translate the contacts' bearings relative to that initial direction.

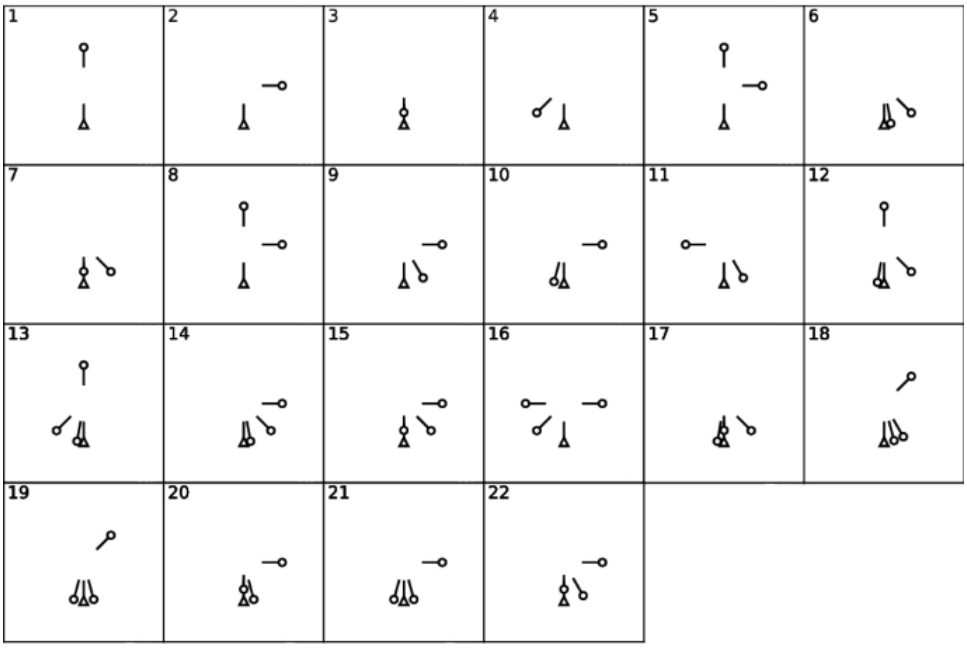
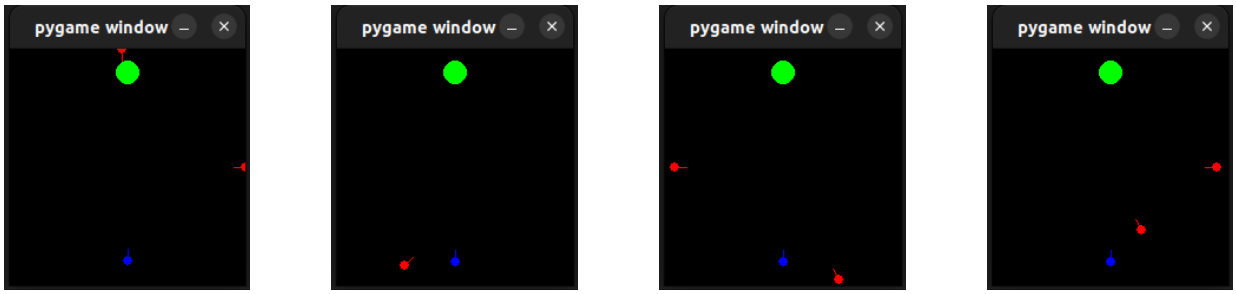


Figure 3.1: Imazu Problem

3.1.1 2-Dimensional Environment

The 2D environment was set up using pygame as previously mentioned. This 2D rendering of a contact situation was simple as the game screen was a canvas that stored all the ships' coordinates and used a line with length proportional to the vessel's speed pointed in the direction that vessel was traveling. The canvas also stored and displayed the waypoint information.

In order to make the environment more realistic I utilized the ARLpy [48] library to effectively simulate the 3rd dimension in a straight line from ownship to each of the targets. This library calculated the arrival angles of a source frequency from the target to the ship. The specifics of what was calculated and provided to the agent are discussed in detail in Section 3.2.1. For this 2D environment a flat ocean floor was assumed and one of BELLHOP's default SSPs, Munk, was used with a slight modification to have a surface duct (Figure 3.3).



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3 (d) Scenario 4

Figure 3.2: Chosen Open Ocean Contact Scenarios

3.1.2 Simulation in Unreal Engine (UnrealBELLHOP)

Unreal Engine [49] is a popular 3D game development software. The software provides in game audio simulation but does not have underwater acoustic simulation. There is the ability to muffle game sound underwater to the human ear but not any type of signal simulation ability.

BYU's FRoSt Lab has developed an underwater simulator called HoloOcean [50] built on Unreal Engine 4; however, the SONAR sensors used for those simulations are active and include imaging SONAR, depth sensors and acoustic communication arrays. My work desired to focus more on signals that would be received by a passive SONAR sensor.

In order to accomplish this I imported a C++ port of BELLHOP [51] into Unreal Engine by way of a dynamic linked-library. Importing this library into a custom Unreal Engine plugin allows use of all the underlying functionality of BELLHOP inside Unreal Engine. It also has the added convenience of not needing to produce the output files shown in Figure 2.2. The outputted data is stored as C++ structs and can be passed to Unreal Engine.

The plugin adapts to the provided environment by calculating the bathymetry of the ocean floor from source to receiver. Figure 3.4 displays how the bathymetry is determined within Unreal Engine. The range of each line from the source is printed to a bathymetry text file along with the

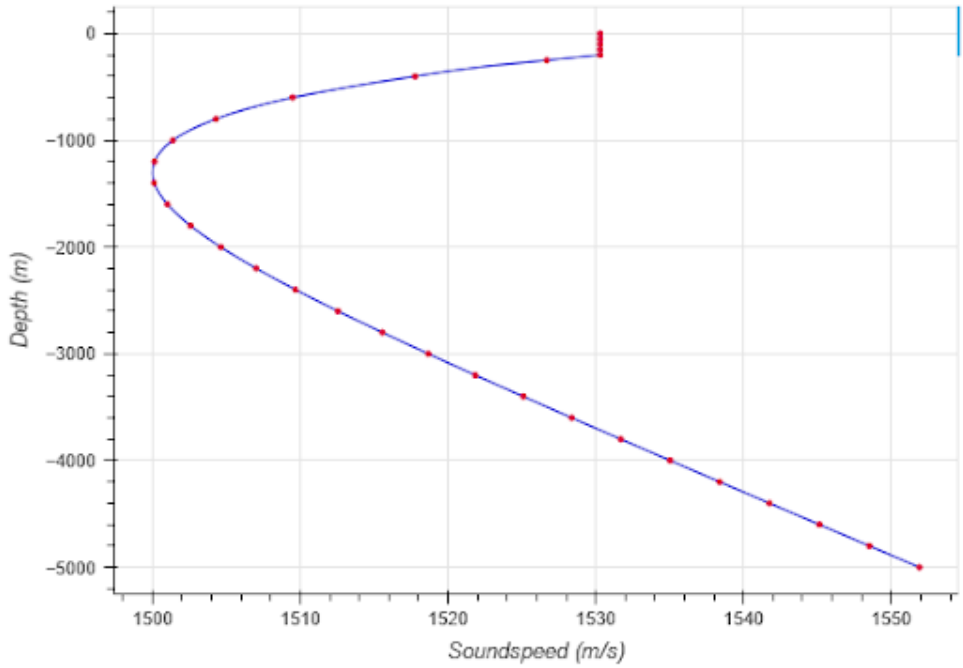


Figure 3.3: The SSP chosen for my simulations in 2D.

depth determined by the line trace. The user inputs their desired SSP by selecting from a list of defaults taken from BELLHOP examples or has the option to enter a custom profile. A Python GUI is used to change the SSP shown in Figure 3.5. Importantly, the SSP must be selected prior to runtime in the editor. By default, the run is done with geometric hat-beams which sacrifices some accuracy for speed but this can be changed to geometric Gaussian beams if desired.

This developed plugin can provide more interactive use of BELLHOP. It allows the user to place sources in the Unreal Engine editor with a ActorComponent class that contains the BELLHOP functionality attached to the desired receiver in order to determine how this specified underwater acoustic environment would behave. The functions can also be called during runtime allowing for a dynamic environment with sources and receivers in motion. The default functionality is to perform an arrivals run of BELLHOP and pass all the arrival information to the user in real time but the user has the ability to use all the functionalities of BELLHOP. Specifically, all the

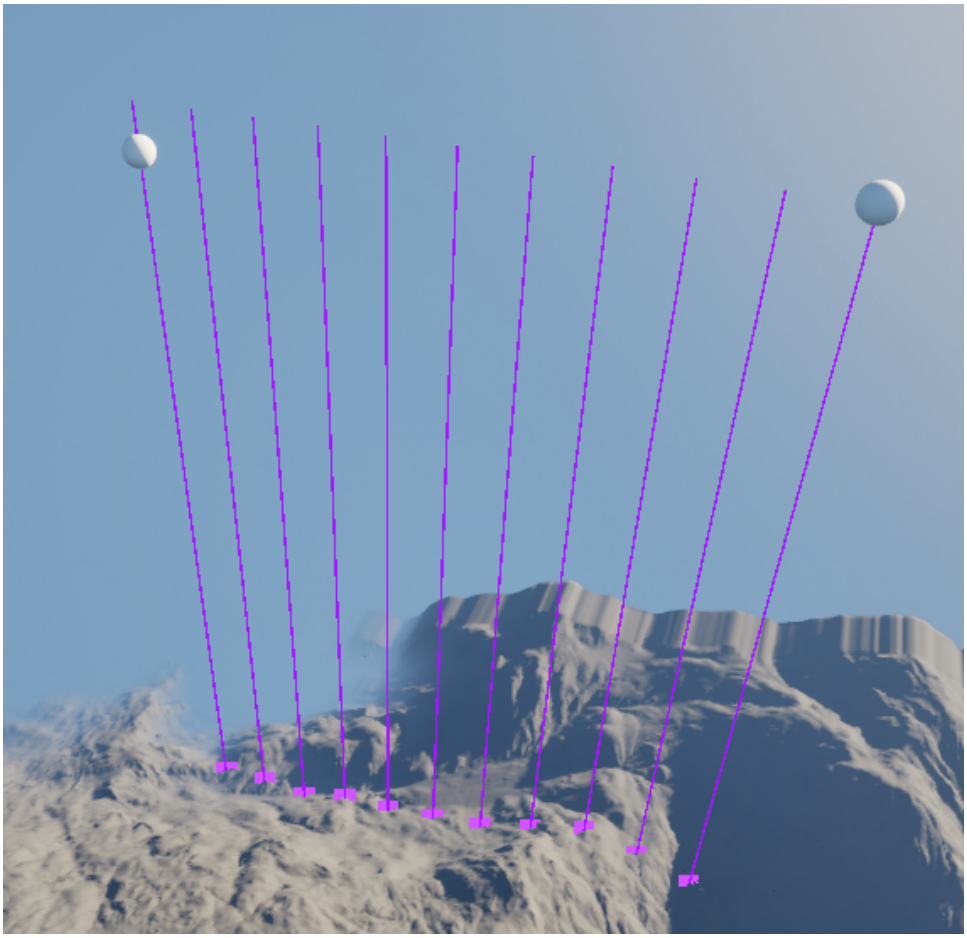
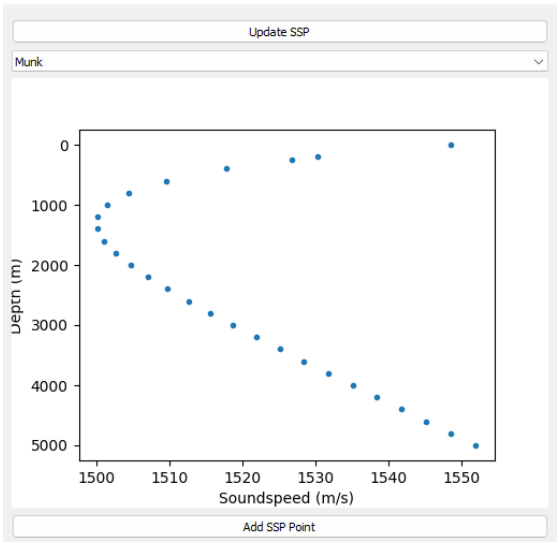
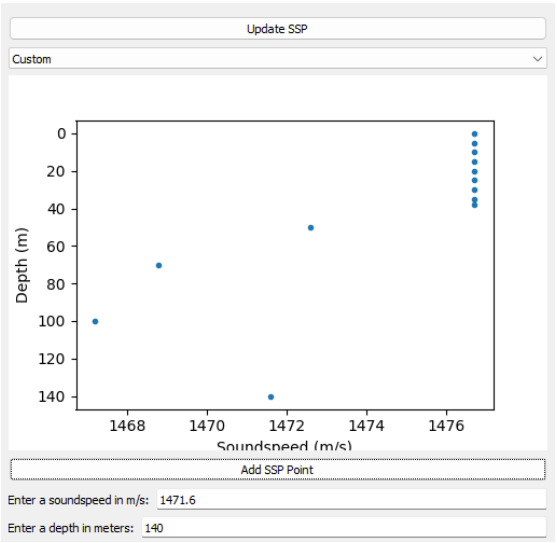


Figure 3.4: Line tracing in Unreal Engine used to determine bathymetry of the environment. The purple lines return their distance from the starting point to when they collide with the landscape below. Water was removed from the scene for visibility and the source/receiver are represented by the sphere meshes.



(a) GUI showing the default 'Munk' SSP



(b) GUI showing an in progress customized SSP

Figure 3.5: Python GUIs in Unreal Engine

data provided by a BELLHOP .arr file is sent to the engine meaning that for each arrival between the source and receiver the number of top bounces, number of bottom bounces, source declination angle, receiver declination angle, amplitude and phase. Azimuth angles for source and receiver could be gained by utilizing the 3D version of BELLHOP as well.

The downside to using this 3D version comes from the formatting of the environment file in a multi contact situation. You are able to specify bearings between the multiple sources and the receiver and their ranges; however, the algorithm will calculate for contacts that are not really there since for each bearing it is given it processes all the ranges provided. For instance, if you put in one contact bearing 030 at 5000 yards and one bearing 180 at 2000 yards the program would run for four total contacts. These would be the two true contacts as well as nonexistent ones bearing 030 at 2000 yards and bearing 180 at 5000 yards. If only one contact is provided then this problem would not appear and a solution to the multiple contact problem could be to handle each contact in its own environment file like I have done in the 2D case in order to account

for the bathymetry possibly being different from the receiver to separate sources. However, this would be slower than 2D and given the lag already introduced some parallel computing would need to be implemented prior to testing; furthermore, the way of generated a bathymetry input file would need to be modified for 3D.

3.2 Setting up Environment for Learning

3.2.1 State and Action Spaces

An essential component of any RL application is the definition of the state and action spaces. There were a few ways forward with representing information coming from passive SONAR. Classic TMA algorithms solely use bearing history of a contact. While bearing history of a single contact would likely yield good outcomes from a RL agent, I wanted the agent to be able to handle multiple contacts at once. With an unknown number of contacts assumed in the environment and the desire to not alter the dimensionality of the state space as the number of contacts changed it became clear that I needed to represent the 360°circle around the submarine.

The next step was to figure out what data to pack into a finite array that represented ”bins” of degrees around the submarine. Ultimately, my experiments tested two different types of data to pass. The first one simply calculated the BRGRT of the contact and placed that value in the array index corresponding to the bin the contact was in. If there was no contact in that bin the array’s default value was zero. If the absolute value of the BRGRT got too low it was altered to have an absolute value of 0.3 in order to clearly differentiate from a no contact to a very low BRGRT contact which is often the most dangerous. Furthermore, the maximum absolute value was capped at 10. Since you do not have ranging information some history is also necessary and

while BRGRT provides that I decided to implement frame stacking. This stacked the arrays of bearing rates on top of each other in order to display the history of all the TGTs. Accordingly, we can define the state space, $\mathcal{S}$, as follows where $f :=$ the number of frames to stack, $b :=$ the number of bins, and Ω is the ordered pairs of frames and bins where a contact is detected.

$$\mathcal{S} := \mathbb{R}^{f \times b}$$

$$s^{(t)} \in \mathcal{S}$$

$$0.3 \leq |s_{ij}^{(t)}| \leq 10.0, (i,j) \in \Omega$$

$$s_{ij}^{(t)} = 0, (i,j) \notin \Omega$$

The standard b I chose was 720 for the state space represented by the BRGRTs of contacts. This meant that each bin accounted for half a degree of bearing coming from the submarine. The bearing calculation was done knowing the coordinates of both the submarine and the contact with noise added according to a zero-mean Gaussian distribution with variance 0.25 degrees.

In order to represent the arrival angles the library ARLpy [48] was used. The arrival corresponding to the largest impulse response magnitude was used, denoted as θ. Passing θ to the agent in an array similarly to the bearing rates was unsuccessful so the solution became representing the angles in an image. In order to accomplish this the max arrival angle possible was also calculated, denoted as $\theta_{\max}$. The columns of the image still represented bins as before and the rows represented time history. The color of the pixel where arrivals were detected, c_{ij},

Figure 3.6: Example Observation Space as an Image emulating a SONAR screen with true bearings

was calculated according to Equation 3.1 (color values were normalized, i.e., $\in [0, 1]$). Different formulations of this equation were also used but the main idea of a linear relationship between the received angle and the maximum remained the same.

$$c_{ij} = \begin{cases} 1 - \frac{1}{2}\frac{|\theta_{ij}|}{|\theta_{\max}|}, & (i,j) \in \Omega \\ 0, & (i,j) \notin \Omega \end{cases} \tag{3.1}$$

The default image size chosen was 40×360. This was a lower bearing resolution than the bearing rate state space but since a CNN was able to be used the convolutions on the image would group nearby pixels together regardless so lowering the dimensionality of the image was seen to speed up training while not reducing performance. Bearing rate information about the contacts could be deduced by the CNN since there was a time history element to the passed image. An example image that was passed to the agent is shown in Figure 3.6.

Another thing to consider in our SONAR data is that bearings to a contact can be represented by their ”true” or ”relative” bearings. The ”true” bearing, θ_T is calculated in a 2-D plane according to Equation 3.2

$$\theta_T = \arctan(\frac{\Delta x}{\Delta y}) \tag{3.2}$$

The ”relative” bearing, θ_R, is then calculated as if ownship's heading, CRS, was North

according to Equation 3.3 assuming that $\theta_T, \text{CRS} \in [-180, +180)$.

$$\theta_R = \theta_T - \text{CRS} \tag{3.3}$$

When calculating relative bearings, the arrays and images described above were slightly altered. Ownship's CRS was the center index of the array and then relative bearings of the contacts were added to this center index in order to place them on the left or right side of the array/image accordingly. Neither method performed considerably better than the other.

In addition to the SONAR data as described above the agent also received the BRG and RNG to the navigational waypoint as well as its own position. This was accomplished in a Gymnasium [52] custom environment through the Dict space which allowed for multiple types of observations to be passed to the agent.

The action space, $\mathcal{A}$, chosen was discrete. The agent would choose an integer $a \in [0, C)$ where C represented the number of courses it could choose. The actual CRS corresponding to the action chosen was calculated by multiplying by $\frac{360}{C}$, i.e. , with $C = 12$ the corresponding CRS options were $\{000, 030, 060, 090, \ldots, 330\}$. While some autonomous surface vessels applications have used continuous action spaces [45] to represent rudder angle, it felt appropriate to force the agent to make bigger turns since small turns do not readily change the bearing rate of a contact. Accordingly, once an action was chosen the environment would simulate the conduction of the turn while still moving the contacts and complete the turn before passing another observation to the agent or allowing another action to be chosen. This was another key difference between this work and the MLE and Kalman Filter methods of TMA since those methodologies assume an instantaneous maneuver.

Another option for the action space if relative bearings were being sent to the submarine as described above was choosing a relative course as well. Instead of choosing a specific course it could choose to remain on course or turn a certain number of degrees to the left or right. Denoting D as the increment of degrees the agent was allowed to turn the discrete action space becomes defined as follows

$$C = \lfloor \frac{180}{D} \rfloor$$

$$a \in [-C, C)$$

$$\text{CRS}_{new} = \text{CRS}_{old} + a * D$$

Furthermore, a third option was to turn to a course plus or minus 180 degrees relative to the direction to the waypoint. This meant that if the agent chose 0 it would turn to point the waypoint and then if it chose otherwise it would move left or right from the bearing to the waypoint in increments as described above.

3.2.2 Reward Function

My reward function has two parts: the navigational reward, r_{nav} and the terminal reward, r_{term}. The terminal reward is 0 until the terminal state is reached while the navigational reward is a sum of factors involving distance to the goal, cross track error, and lateness to the waypoint.

$$r_{\text{goal}} = \frac{\Delta d_{\text{goal}}}{\max \Delta d_{\text{goal}}} \tag{3.4}$$

$$r_{\text{xerr}} = \frac{d_{\max \text{xerr}}/2 - d_{\text{xerr}}}{d_{\max \text{xerr}}/2} \tag{3.5}$$

$$r_{\text{late}} = -10^{-4} \lfloor \frac{t}{200} \rfloor \tag{3.6}$$

$$r_{\text{nav}} = r_{\text{goal}} + r_{\text{xerr}} + r_{\text{late}} \tag{3.7}$$

$$r_{\text{term}} = -20 \text{ if collision} \tag{3.8}$$

$$r_{\text{term}} = 0 \text{ if out of map} \tag{3.9}$$

$$r_{\text{term}} = 20 \text{ if goal achieved} \tag{3.10}$$

3.4 is a simple measure of closing distance on the waypoint. The $\max \Delta d_{\text{goal}}$ was calculated at every iteration and assumed that ownship drove straight to the goal at current speed. 3.5 measured how far to the left or right of the original track ownship was (in these cases a track of North from the starting position.)

Adding in 3.6 proved to be vital to see a logical navigational path by ownship. This term penalized ownship for taking too long to reach the waypoint incentivizing the agent to find a more straightforward route to reach the goal. Prior to adding that term ownship would circle around the waypoint after avoiding contacts presumably due to that technique resulting in accumulating favorable reward returns from r_{goal}.

Chapter 4: Results

Following training each agent with different parameters such as slightly altering state and action spaces, a total of 200 evaluation episodes were run with one of my chosen scenarios randomly selected for each episode. The overall success rate of reaching the target was 88.375% in 5600 simulations. Notably there were 2 out of the 28 different agents that could be considered outliers as they were the only ones below 80%. I was able to monitor the average reward throughout training in order to determine if the agent was converging on some policy as show in Figure 4.1. This particular agent achieved a 95.5% success rate over the 200 evaluation episodes.

As discussed in Chapter 4 two different observation space types were evaluated with their success rates plotted in Figure 4.2a.

The n_steps parameter for PPO determines how many steps to take in an environment before the current environment is terminated and the update to the policy and value networks takes place was also varied with the results shown in Figure 4.2b.

Due to the choice of a discrete action space the size of the course increments available to the agent were also varied with results in Figure 4.2c. As previously stated in Chapter 3, bigger course changes are generally desirable in bearing-only situations in order to see changes in BRGRT.

Another key aspect of PPO is the clip parameter, ϵ, that can limit the policy network update.

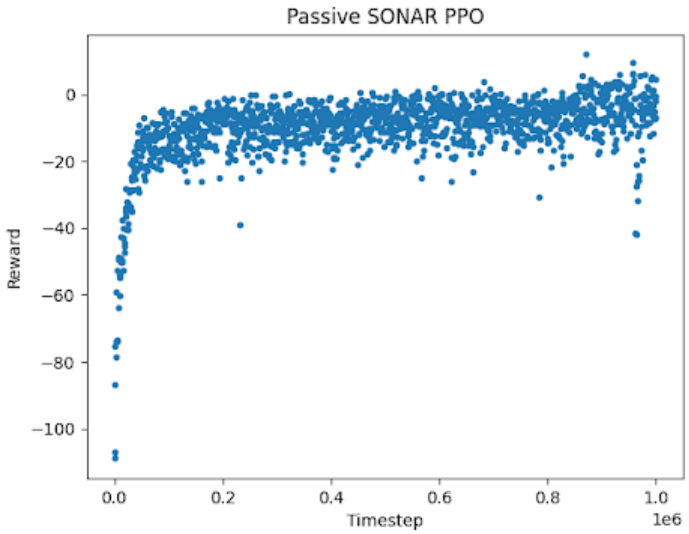
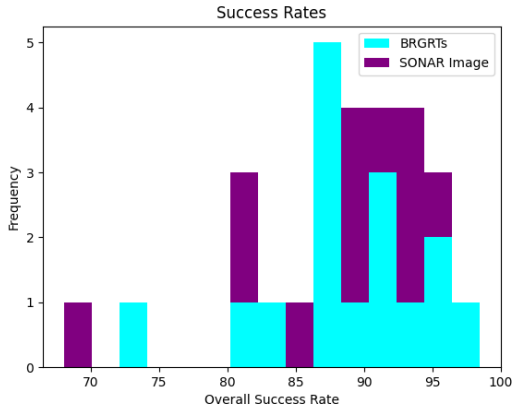


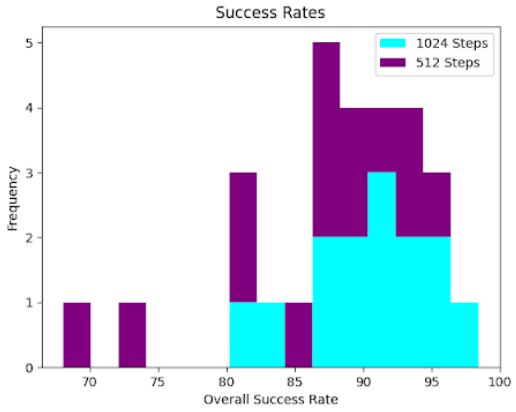
Figure 4.1: Graph of reward received throughout training of a PPO agent

I utilized 2 common values of 0.1 and 0.2 with the varying results depicted in Figure 4.2d.

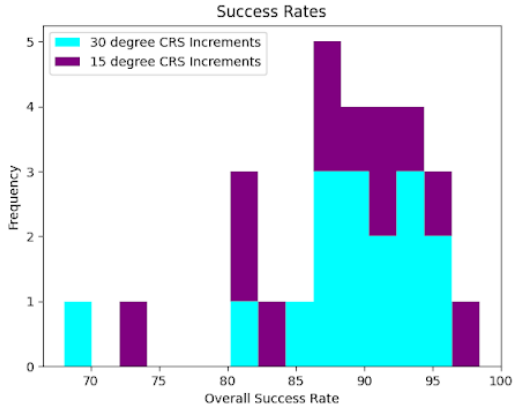
Overall, varying these parameters did not lead to a drastic reduction in the ability of the agent to learn with the majority of agents landing in the 85 to 95 percent range. However, it is notable that the 2 worst performing agents were using a smaller ϵ and a smaller n_steps parameter.



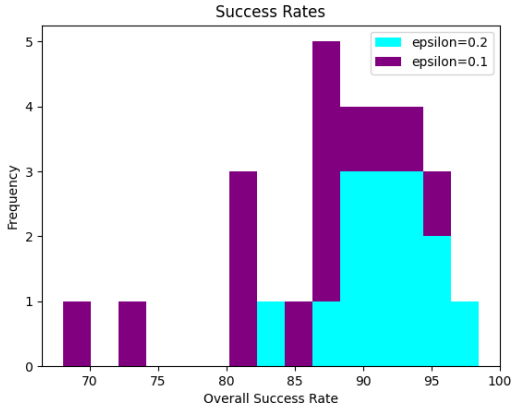
(a) Altering the type of observation space.



(b) Altering the steps parameter.



(c) Altering the course increment available.



(d) Altering the PPO clip parameter ϵ

Figure 4.2: Success Rate Histograms

Chapter 5: Conclusion

Overall the ability of the agent to avoid collision using my simulated data was impressive; however, it is not comparable to current submarine implementations of bearing-only target motion analysis (with human guided direction.) While it is more accurate than blindly following a traditional algorithm as described in Chapter 2 that is never something that actually occurs onboard submarines since there is always human input.

The ability of UnrealBELLHOP to run the BELLHOP calculations inside Unreal Engine is a major step forward to reliable underwater acoustic simulation in a user controlled 3D environment. This is promising as a user would have the ability to design an environment more intuitively than just typing up a text file to match their idea as well as allowing a user-defined dynamic environment.

The agent's performance when providing arrival angle data could also be affected by changing the SSP used. The arrival angles in my chosen ocean environment provided an obvious trend with range shown in Figure 5.1 but could change based upon differing bathymetry and SSP.

5.1 Future Work

In light of the analysis provided above the next logical step forward would be to combine the RL agent with one of the traditional TMA algorithms (or a modern neural network trained to

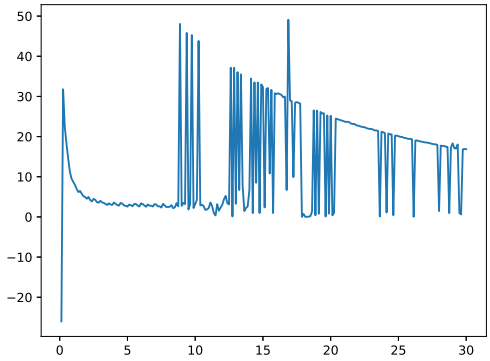


Figure 5.1: Absolute value of arrival angles, $|\theta|$, plotted vs range (in kyds) in my ocean environment

conduct TMA such as [53]) where the agent would replace the human interaction that is present onboard a submarine.

Additionally, it is possible that the agent's performance could improve with better training data. This would mean gaining access to official submarine simulators or even SONAR data from a submarine's deployment correlated with a known contact picture from another source such as AIS. Unfortunately, both of these options are unavailable at the unclassified level and would need to be conducted in a secure facility.

Keeping in mind that this research focused on avoiding close proximity to other vessels the objective could shift to one of intercepting a target with the same available data. These implementations could benefit in areas such as torpedoes and missile defense systems.

The UnrealBELLHOP plugin is still in development to improve user experience. Currently, the background calculation done by the C++ BELLHOP port is running in the Engine thread; therefore, there is still a need to parallelize these BELLHOP calculations so that performing them at runtime does not cause the display to lag. Furthermore, the plugin is designed to give

arrival information but that is not the only type of calculations that BELLHOP could perform. It could be useful to add the functionality to display information such as the rays, eigenrays, and pressure field shown in Figure 2.2.

Once the plugin is optimized it could be used to simulate the same Reinforcement Learning scenario as done in the 2D environment. Unreal Engine has a Learning Agents plugin [54] in beta and there are some options outside of offical Epic Games content such as MindMaker [55].

Bibliography

[1] Finn B Jensen, William A Kuperman, Michael B Porter, Henrik Schmidt, and Alexandra Tolstoy. *Computational Ocean Acoustics*, volume 2011. Springer, 2011.

[2] MB Porter. Bellhop gaussian beam/finite element beam code, ocean acoustics library.

[3] Michael B Porter. The bellhop manual and user's guide: Preliminary draft. *Heat, Light, and Sound Research, Inc., La Jolla, CA, USA, Tech. Rep*, 260, 2011.

[4] Michael B Porter. Bellhop3d user guide, 2016.

[5] V. Červený, M. M. Popov, and I. Pšenčík. Computation of wave fields in inhomogeneous media — Gaussian beam approach. *Geophysical Journal International*, 70(1):109–128, 07 1982.

[6] RH Gallagher, JT Oden, C Taylor, and OC Zienkiewicz. Finite element. *Analysis: Fundamentals. Prentice Hall, Englewood Cliffs*, 1975.

[7] Michael B. Porter and Homer P. Bucker. Gaussian beam tracing for computing ocean acoustic fields. *The Journal of the Acoustical Society of America*, 82(4):1349–1359, 10 1987.

[8] Michael B Porter. Beam tracing for two- and three-dimensional problems in ocean acoustics. *The Journal of the Acoustical Society of America*, 146(3):2016–2029, 10 2019.

[9] Daniel H Wagner and W Charles Mylander. *Naval operations analysis*. Naval Institute Press, 1999.

[10] Xuezhi Wang, Yongqiang Cheng, and Bill Moran. Bearings-only tracking analysis via information geometry. pages 1 – 6, 08 2010.

[11] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[12] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. Spie, 1997.

[13] Vincent J. Aidala. Kalman filter behavior in bearings-only tracking applications. *IEEE Transactions on Aerospace and Electronic Systems*, AES-15(1):29–39, 1979.

[14] HD Hoelzer, GW Johnson, and AO Cohen. Modified polar coordinates-the key to well behaved bearings only ranging. *IR & D Report*, pages 78–M19, 1978.

[15] V. Aidala and S. Hammel. Utilization of modified polar coordinates for bearings-only tracking. *IEEE Transactions on Automatic Control*, 28(3):283–294, 1983.

[16] Sanjeev Arulampalam. *A comparison of recursive style angle-only target motion analysis algorithms*. DSTO Electronics and Surveillance Research Laboratory, 2000.

[17] D. Laneuville and C. Jauffret. Recursive bearings-only tma via unscented kalman filter: Cartesian vs. modified polar coordinates. In *2008 IEEE Aerospace Conference*, pages 1–11, 2008.

[18] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[19] Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms, 2022.

[20] Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

[21] Richard Bellman. On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, 38(8):716–719, 1952.

[22] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[23] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[24] Peter W Glynn. Stochastic approximation for monte carlo optimization. In *Proceedings of the 18th conference on Winter simulation*, pages 356–365, 1986.

[25] Tommi Jaakola, Satinder P. Sing, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352, 1995.

[26] Peter Marbach and John N Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

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

[28] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, pages 834–846, 1983.

[29] V. R. Konda and J. N. Tsitsiklis. On actor-critic algorithms. *SIAM J. Control Optim.*, 42(4), 2003.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[32] Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, Hankz Hankui Zhuo, et al. Coordinated proximal policy optimization. *Advances in Neural Information Processing Systems*, 34:26437–26448, 2021.

[33] Yuhui Wang, Hao He, Xiaoyang Tan, and Yaozhong Gan. Trust region-guided proximal policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[34] Christine Chavin, Salim Lardjane, Gael Morel, Jean-Pierre Clostermann, and Benoit Langard. Human and organisational factors in maritime accidents: Analysis of collisions at sea using the hfacs, 2013.

[35] Hayama Imazu and T Koyama. The optimization of the criterion for collision avoidance action. *Journal of Japan Institute of Navigation*, 71, 1984. (in Japanese).

[36] Kazuhiko Hasegawa and Akihiko Kouzuki. Automation collision avoidance system for ships using fuzzy control. 1987.

[37] Liang Hu, Wasif Naeem, Eshan Rajabally, Graham Watson, Terry Mills, Zakirul Bhuiyan, and Ivor Salter. Colregs-compliant path planning for autonomous surface vehicles: A multiobjective optimization approach. *IFAC-PapersOnLine*, 50(1):13662–13667, 2017.

[38] Yoshiaki Kuwata, Michael T Wolf, Dimitri Zarzhitsky, and Terrance L Huntsberger. Safe maritime autonomous navigation with colregs, using velocity obstacles. *IEEE Journal of Oceanic Engineering*, 39(1):110–119, 2013.

[39] Shinya Nakamura and Naoki Okada. Development of automatic collision avoidance system and quantitative evaluation of the manoeuvring results. *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, 13(1):133–141, 2019.

[40] Kunihiko Mitsubori, Takeshi Kamio, and Takahiro Tanaka. Finding the shortest course of a ship based on reinforcement learning algorithm. *The Journal of Japan Institute of Navigation*, 110:9–18, 2004.

[41] Tomohiro Tanigawa, Takeshi Kamio, Kunihiko Mitsubori, Takahiro Tanaka, Hisato Fujisaka, and Kazuhisa Haeiwa. Modified multi-agent reinforcement learning system to find ships' courses. *IEICE Proceedings Series*, 2(487):487–490, 2013.

[42] Masahiro Nakayama, Takeshi Kamio, Kunihiko Mitsubori, Takahiro Tanaka, and Hisato Fujisaka. Reinforcement learning based search for ships' courses controlled by safety. In *The 2014 proceedings of international symposium on nonlinear theory and its applications*, pages 28–31, 2014.

[43] Haiqing Shen, Hirotada Hashimoto, Akihiko Matsuda, Yuuki Taniguchi, Daisuke Terada, and Chen Guo. Automatic collision avoidance of multiple ships based on deep q-learning. *Applied Ocean Research*, 86:268–288, 2019.

[44] Ryohei Sawada. Automatic collision avoidance using deep reinforcement learning with grid sensor. In *Proceedings of the 23rd Asia pacific symposium on intelligent and evolutionary systems*, pages 17–32. Springer, 2020.

[45] Ryohei Sawada; Keiji Sato; Takahiro Majima. Automatic ship collision avoidance using deep reinforcement learning with lstm in continuous action spaces. *Journal of Marine Science and Technology*, 26(2):509–524, 2021.

[46] H. Imazu. Research on collision avoidance maneuver (in japanese). 1987.

[47] Pete Shinners. Pygame. `http://pygame.org/`, 2011.

[48] Mandar Chitre. arlpy. `https://github.com/org-arl/arlpy`, 2016.

[49] Epic Games. Unreal engine.

[50] Easton Potokar, Spencer Ashford, Michael Kaess, and Joshua G. Mangelson. Holoocean: An underwater robotics simulator. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3040–3046, 2022.

[51] Jules Jaffe. bellhopcuda. `https://github.com/A-New-BellHope/bellhopcuda`, 2023.

[52] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[53] Carl Kolon. Seeing underwater with neural networks. `https://carlkolon.com/2023/08/06/seeing-underwater.html`, 2023. [Accessed 11-06-2024].

[54] Brendan Mulcahy and Daniel Holden. Learning to drive: Tutorial, Aug 2023.

[55] Aaron Krumins. Mindmaker machine learning ai plugin, Nov 2020.