Terrain and Adversary-Aware Autonomous Robot Navigation

Aniekan Ufot Inyang
University of Denver

Terrain and Adversary-Aware Autonomous Robot Navigation

Abstract

In autonomous robot navigation, the robot is able to understand the environment around it for intelligent navigation. From its world model of this environment, it generates a global plan for navigation from a position to a goal based on different factors. This research aims to implement autonomous robot navigation by learning terrain affordances: traversability (moving quickly) and concealment (staying hidden from an adversary) using the Preference-based Inverse Reward Learning (PbIRL) methodology. The PbIRL methodology reduces the barrier of generating initial demonstration data to learn the terrain affordances by using a human expert's preferences to learn individual weights over the terrain types in the environment. These weights are then combined into a costmap, which is passed to a planner for path generation and navigation to a goal.

This thesis extends prior research in learning terrain costs for robot navigation using an active rewards learning Python package, APReL. The novel contribution of this thesis is that the robot not only considers traversability affordances of different terrains for navigation, but also considers concealment from an adversary in the environment. This research work also augments the APReL package with wrappers suitable for the use-case. The model will be evaluated by comparing its performance with an already existing Inverse Optimal Control (IOC) model trained from human demonstrations as a baseline learning algorithm for autonomous robot navigation.

Document Type

Masters Thesis

Degree Name

M.S.

First Advisor

Christopher Reardon

Second Advisor

Daniel Baack

Third Advisor

Maggie Wigness

Keywords

Adversary aware navigation, Autonomous robot navigation, Context aware navigation, Preference based learning, Robot learning, Terrain aware navigation

Subject Categories

Artificial Intelligence and Robotics | Computer Engineering | Computer Sciences | Engineering | Physical Sciences and Mathematics | Robotics

TERRAIN AND ADVERSARY-AWARE AUTONOMOUS ROBOT NAVIGATION

A Thesis

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Aniekan Ufot Inyang

August 2023

Advisor: Dr. Christopher Reardon

Author: Aniekan Ufot Inyang
Title: TERRAIN AND ADVERSARY-AWARE AUTONOMOUS ROBOT NAVIGATION
Advisor: Dr. Christopher Reardon
Degree Date: August 2023

Abstract

In autonomous robot navigation, the robot is able to understand the environment around it for intelligent navigation. From its world model of this environment, it generates a global plan for navigation from a position to a goal based on different factors. This research aims to implement autonomous robot navigation by learning terrain affordances: traversability (moving quickly) and concealment (staying hidden from an adversary) using the Preference-based Inverse Reward Learning (PbIRL) methodology. The PbIRL methodology reduces the barrier of generating initial demonstration data to learn the terrain affordances by using a human expert's preferences to learn individual weights over the terrain types in the environment. These weights are then combined into a costmap, which is passed to a planner for path generation and navigation to a goal.

This thesis extends prior research in learning terrain costs for robot navigation using an active rewards learning Python package, APReL. The novel contribution of this thesis is that the robot not only considers traversability affordances of different terrains for navigation, but also considers concealment from an adversary in the environment. This research work also augments the APReL package with wrappers suitable for the use-case. The model will be evaluated by comparing its performance with an already existing Inverse Optimal Control (IOC) model trained from human demonstrations as a baseline learning algorithm for autonomous robot navigation.

Acknowledgments

This research project is possible because of the support and guidance of numerous people. First, I will like to thank my advisor, Dr. Christopher Reardon, for trusting me with the implementation of this, and for taking the time to read through, make corrections, and guide me. I also thank Dr. Maggie Wigness from the US Army Research Laboratory (ARL) for constantly offering support, sharing resources, and collaborating with me throughout the research process. Thanks to my other committee members, Dr. Matthew Rutherford and Dr. Daniel Baack for offering insights and their time towards this thesis completion. I extend my gratitude to the staff from ARL that I collaborated with, my lab mates at ARISE Lab, my classmates, and Faculty of the Department of Computer Science, Ritchie School of Engineering and Computer Science, University of Denver. It has been a pleasure meeting, studying with, collaborating, working with, and learning from you all. I am very grateful to my parents, my sisters, Bami, Valerie, and all my friends for constantly stepping up and holding me when I was weary, cheering me on, encouraging me always, and supporting me in every way throughout this process. Finally, I thank myself for showing up daily, doing the work, persevering, and not giving up. I have seen myself grow through it all, by the grace of God. It was all worth it and I am extremely privileged to be afforded all these opportunities that have contributed to making me who I am today.

Contents

List of Algorithms

List of Figures

List of Tables

1 Introduction

If a robot is in an adversarial environment, how does the robot know enough about the environment and the terrains around it to advance and yet stay safe?

To solve this problem, this research project proposes context-aware stealth maneuvering based on two attributes of the terrain—traversability (moving fast) and concealment (not being detected by the adversary). Context-aware stealth maneuvering is the ability of a robot platform to navigate an environment successfully taking into account the context of the environment.

This research hopes to solve the problem of how a robot can

1. Understand the terrain affordances of the environment - i.e., which terrain features provide concealment and traversability. Traversability is afforded when an agent is co-located with a terrain feature (e.g., on a road) while concealment is when the feature is in between the agent and the adversary's likely location.

2. Adapt to smartly maneuver the environment based on different contexts–the location of the adversary in the environment and the terrains around the robot at that time.

This is done using context-aware global path planning via an active preference-based learning algorithm to train a model which outputs weights for concealment and traversability per terrain type.

Autonomous robot navigation is an active research area. However, while there has been active ongoing research in this field, many recent approaches focused on different learning methods like learning from demonstration [1] [2] [3], imitation learning [4] [5] [6], and reinforcement learning [7] of a policy or reward function that enables the robot to navigate autonomously in an environment. Because this initial demonstration data might also not always be available or easy to collect/access for some environments, this presents a challenge to using this method to learn navigation over that environment. Research has also been carried out on learning for different goals, such as learning traversability of a robot in an environment without additional context [8], or learning autonomous robot navigation with a policy in a specific environment type, for example, cluttered rough terrain [9] [10].

The premise of the research is that the human expert has knowledge of the terrains in the environment, adversary location, and this could be represented as a hidden model and learned for the robot to navigate an environment. The goal is to learn this hidden model from the human expert and I use Preference-based Inverse Reinforcement Learning (PbIRL) to achieve this.

1.1 Terrains

An environment is often made up of multiple terrains and objects. Some examples are water, sand, trees, and buildings, among others in the environment. These terrains have different attributes that determine traversability to a goal position or concealment of a robot from an adversary in the environment. The global planner uses the costmap derived from these learned affordances of the terrains to generate a plan which the robot uses to maneuver around the environment. The plan generated is based on the terrain feature costs and optimizing for either traversability, concealment, or both based on real-time context.

1.2 Proposed approach

This thesis extends previous work done by [8] on learning terrain costs from human preferences via a visual interface. The human expert was shown navigation trajectories over the different terrains in an environment via the visual interface for robot learning. This previous research work focused on learning terrain costs for autonomous robot navigation without requiring prior demonstrations (and moving the robot) for four terrains: road, sidewalk, grass, and mud. The previous work was built on an adapted version of APReL, a unified Python package for active reward learning algorithms [11]. [8] tested the learned behavior with human subjects in a simulated user study to measure the performance of the learned model and the responses from the human subject study showed a high confidence in the paths generated by the learned model.

Building up on this prior research, I propose implementing an active preference-based learning method to learn the terrain feature costs over traversability and concealment and optimizing these costs into a costmap passed to the global planner for path planning. This is a different approach from learning from demonstrations and imitation learning which requires an initial teleoperated demonstration for the robot to learn from. Active preference-based learning also helps the model adapt and learn the weights on these features converging faster by optimizing the queries to learn as much information on the features' costs based on previous responses by the user. An advantage of query optimization is the minimization of the time and effort needed to train the model [12].

This method shows two proposed trajectories from the robot to a goal position, iteratively asks the user to select what trajectory they prefer, and updates the costs for each terrain feature based on the user's response to the query. The active querying step is done separately for traversability and concealment, and the final weights are combined in a function to generate the costmap.

1.3 Contributions

The contribution of this research work is introducing concealment–an additional terrain affordance that improves robot autonomous navigation behavior in an environment with an adversary. This is relevant in a military mission application where a robot might be in an environment without any humans and it is important for it to traverse the environment and stay hidden from the adversary's sight as much as possible. This contribution implements learning of costs for two terrain affordances and provides a way to combine these costs to generate a global plan based on mission context.

Another contribution of this research work is that it extends prior work done by [8], scaling to a larger, more realistic, 3D environment, with more terrain types present in the environment. This research also exhibits generalizability to new affordances and new terrain types. Given that the learning is done individually, additional affordances can be learned by defining the features extraction function of the trajectory to learn over for the new affordance.

This thesis also compares the experimental performance of the PbIRL method to another learning method, Inverse Optimal Control (IOC), an inverse reinforcement learning method, in learning the terrain costs for traversability and concealment. The PbIRL model implemented in this thesis does not require initial demonstration to learn while the IOC model learns from demonstrations.

2 Related work

2.1 Autonomous robot navigation

Autonomous robots are robots with some sort of intelligence to carry out a task or a set of tasks without major human intervention. Autonomous robots must be able to perceive obstacles and boundaries, plan motion to a goal, and successfully navigate their environment, based on the planned mission, [13] successfully. Autonomous robot navigation could either be reactive or deliberative. Reactive navigation is when the objects in the environment are only known after navigation begins to a goal position [14]. Deliberative navigation, on the other hand, constructs a prior global map of the environment and updates the map as the robot navigates. The robot carries out deliberations while navigating to update the global map. These deliberations begin from sensing, goal reasoning, learning, planning, acting, and then monitoring [15].

Autonomous robot navigation, as the name implies, is the ability of a robot to navigate in an environment without control from a human. This is not limited to Unmanned Ground Vehicles (UGV), autonomous robot navigation is also a capability of Unmanned Aerial Vehicles (UAV), Unmanned Surface Vehicles (USV), and underwater robots. Research on autonomous robot navigation have explored different scenarios and with additional contexts to increase intelligence and improve robot navigation behavior. Based on the nature of the environment, research has focused on autonomous robot navigation in highly populated areas [16], and cluttered rough terrain environments [9]; and depending on human [17] and social factors in the environment [18] [19] [20]

[21]. This thesis addresses the implementation of autonomous robot navigation in an environment with numerous terrains and the presence of an adversary that the robot aims to not be detected by.

2.2 Robot learning

Typically in robotics, learning is used to find a good reward function or policy that maps the observation space/state of the robot to actions to form an optimal behavior. This reward function can be specified by a human expert, however, this could introduce errors as the human expert might experience difficulties in defining the right reward function. To mitigate these errors, recent research explores how to learn these reward functions instead. Jeon et al. proposed a formalism for reward function learning where the robot learns a reward function from human behavior and maps trajectories to scalar rewards, with each human behavior being modeled as a choice from a set of options. [22]

Robot learning is applied for various tasks from object manipulation, perception and object detection, and robot navigation. This learning can be done by learning from demonstration [1] [2] [3], imitation learning [4] [5] [6], reinforcement learning [7], deep reinforcement learning [23], inverse reinforcement learning [24], active learning [25] [26], or preference-based learning [11] among others. Machine learning and deep learning paradigms are used in robotic learning applications because hand-engineering desired trajectory or reward functions for complicated tasks could be difficult to meet specific requirements and get extremely complex [2]. PbIRL algorithms fall under these paradigms, specifically, as an inverse reinforcement learning technique. In this research, I train a PbIRL model to learn traversability and concealment affordances of terrains in an environment for a robot to autonomously navigate through these terrains to a goal position successfully.

2.3 Context awareness

Context awareness is a crucial aspect of real-time learning and human-robot interaction. It is an active research area and work has been done utilizing context awareness in different research areas. Different methods to gain contextual awareness have been used in previous research.

Wang et al. research using deep learning (deep CNN) by processing video images to recognize a human's action and understand the context of that action for effective human-robot collaboration in assembling an automotive engine [27]. Ziebart et al. modeled a road network as Markov Decision Process (MDP) with states and actions. The drivers in this work are trying to reach a goal while optimizing other trade-offs like stress, fuel costs, safety, etc. which the authors called a cost value (negative reward). They also assumed that the reward is independent of the destination so the same reward can be used for different trips [28].

To gain context awareness, Liu et al. [29] implemented real-time collision sensing by viewing all tracked robots and humans as 3D point clouds in a virtual environment and checking for collision in MoveIt! [30]. Cho et al. use an ontology repository as a context handler to understand the context of the robot's environment. [31].

In this research work, I implement context awareness (detection by the adversary) by ray casting from the adversary to the robot position to measure sensing of the robot by the adversary.

2.3.1 Multi-context and multi-objective inverse reinforcement learning

Previous works cited have focused on a singular context awareness, however, this thesis solves the problem on a multi-context level. Unlike singular context awareness, multi-context is more complex, as there is the need to represent and combine the multiple contexts optimally.

Kollmitz et al. approach multi-context learning by proposing an overall reward function that sums up individual reward terms in a human-aware delivery robot that learns to navigate via inverse reinforcement learning. These reward terms are the reward function for reaching the goal with an additional weight for the distance deviation from the shortest path, the reward function with a very large negative penalty for being in an obstacle state, and the reward function to consider a human's personal space [17]. But does this method take into consideration context hierarchy, causality, and dependence? These are why Wei Gao et al. introduced the Context-Hierarchy Inverse Reinforcement Learning (CHIRL) algorithm to solve [32]. Here, the reward functions are represented as neural network modules, with the different contexts being different layers. This enables data sharing among contexts when learning the reward functions, as the set of all context-hierarchy reward functions is learned jointly. The authors also applied MaxQ decomposition [33] for state abstraction, to remove state dimensions that are irrelevant for a sub-task to decrease training time.

Similarly, Chanyoung Jung and David Hyunchul Shim [34] trained a Convolutional Long Short-Term Memory (ConvLSTM)-based Deep Neural Network (DNN) to approximate the reward function and predict the traversability map for autonomous driving in an urban environment. The network is able to extract multiple contexts from the LiDAR sensor and route data represented in a local map grid which is eventually incorporated into the output reward map. My research work covers a variant of the multi-context/multi-objective inverse reinforcement learning solution with the objectives being traversability and concealment.

2.4 Inverse reinforcement learning

Inverse Reinforcement Learning (IRL) is a research area that has been applied in robotics since the early 2000s. Andrew Ng and Stuart Russell proposed a technique for learning a reward function behavior from observing optimal behavior. The inverse reinforcement learning problem was defined as an MDP of a tuple: (S, A, P_{sa}, γ) with S being a finite set of N states, A being a set of actions, P_{sa} being the state transition probabilities when taking action a in state s, γ is the discount factor between 0 and 1 (not inclusive of 1), which is used to learn a reward function or a set of reward functions, R, of which π is an optimal policy for the MDP (S, A, P_{sa}, γ, R) [24].

There have been a number of extensions to this, with new methods proposed, and advanced research work built upon this in different areas of application. One of which is the Bayesian inverse reinforcement learning introduced by Ramchandran & Amir in 2007. [35] state the need for a probability distribution to represent the uncertainty in the reward function that explains the agent's behavior. The Bayesian IRL model derives a posterior probability distribution for the rewards from a prior distribution and then a probabilistic model of the expert's actions given the reward function [35]. Ziebart et al. also introduce a probabilistic approach to solving the problem of recovering the reward function from mimicking demonstrations [28] based on the principle of maximum entropy and this considers a distribution over all possible behaviors and seeks to resolve ambiguities in choosing the distributions.

Following the IRL problem formulation, I model the problem setup of this research as an MDP in line with the implementation library used. The states of the MDP are the position of the robot in the environment and the action set is a continuous set of possible accelerations and directions to take to get to the next position.

2.5 Active learning

Active learning and perception is a more recent research focus and is applied to robot navigation, object detection, and human-robot interaction, among others. Noseworthy et al. describe active learning as a technique for identifying unlabeled instances that are most informative in learning a target concept [36].

Active learning model performance can be improved by the techniques used to sample and generate the queries. Previous research works have explored a complete query sampling strategy [36] that assumes a uniform sampling over the possible queries. Other methodologies explored are a greedy method that selects the top k queries that individually optimize the acquisition function albeit redundantly [36] and a Thomas sampling method that aims to minimize regret during the querying process [37]. Regret is the error ratio between the weights while learning to find an optimal behavior [38]. Other active learning query methods are sequential sampling which approaches the query sampling by scoring the plans in these queries and presenting queries that tend to provide more information based on the probability of a certain action being taken [36], volume removal which uses the strategy of selecting queries from the set that maximizes the expected difference between the prior weights and the not normalized posterior weight values [11], incremental only subsequently considers queries that meet an initial prefix condition [36], and information gain which is an extension from volume removal that does not focus on reducing the uncertainty for the robot learning [39], but instead focuses on maximizing the human's ability to respond clearly and accurately to the queries for the most information gain.

2.6 Preference-based learning

With reinforcement learning, the human hard-codes the reward function. However, it is difficult for a human to assign how much numerical reward an action or trajectory should get [12]. While Biyik et al. model the problem as an MDP [40] in the Active Preference-based Reward Learning (APReL) library, Tucker et al. model the learning problem for Preference Optimization and Learning Algorithms for Robotics (POLAR) using Gaussian processes [41] with new actions to execute selected via regret minimization or information gain. Some learning algorithms were applied in POLAR from prior research work; regret minimization was accomplished using the CoSpar algorithm [37]. This was extended in LineCoSpar [42] for regret minimization in high-dimensional action spaces, and ROIAL for the Region Of Interest Active Learning via information gain [43].

In preference-based learning, it is essential to optimize learning as much information as possible. Biyik et al. hope to achieve this using their novel and unified Python library, APReL, for experimenting with various active preference-based reward learning techniques [11]. The approaches for learning a preference-based policy could be learning a policy, learning a preference model, or learning a utility function [44].

Although many researchers in the field assume the reward function to be linear, as it is easier to learn it this way, this may not capture non-linearity properly and could lead to high accuracy and likelihood variances when tested. To solve this, [26] proposed modeling the reward function as a Gaussian Process (GP) and using GP regression in active preference-based setting to learn the reward function.

Dhruva Bansal et al. opted to approach preference-based learning as a human-in-the-loop dual representation where the robot learner keeps two representations; one for learning the low-level control policy, and the other as a mental model of human trainers [45]. The authors used scene graphs as a form of abstract representation of the states of the environment. Objects are represented as nodes and the edges in the graph denote the relationships between objects. Although using abstract representations in robotics learning tasks could lead to losing critical information needed to learn the policy, the researchers avoided this by only using it as a mental model of human trainers and not for the low-level control policy.

However, preference-based learning suffers from slow learning due to getting feedback from binary queries, and optimal querying is NP-hard [46]. Different authors have tried out other methods to solve this inefficiency, like using demonstrations to learn the prior reward function and ground the queries [47]. There is active research in the area of optimizing queries, making queries more human-friendly, and asking easy questions that allow the model to learn the reward function optimally without asking too many queries [39].

Preference-based algorithms could either learn state preferences, action preferences, or trajectory preferences [44]. State preferences compare two states and choose that $s_a \succ s_b$ for two states shown to the user to provide their preference. Action preferences compare two actions and ask the user to provide their more preferable action to take based on the current state. However, since for some problems, individual states and actions do not provide the most information for the model to learn long-term, this introduces the need for trajectory preferences. Trajectory preferences are when a user is provided with a set of state and action pairs, from which a feature set is derived, and the user is asked to choose their preferred trajectory. The APReL library, used in this research, implements this trajectory preferences learning [11].

2.6.1 APReL

This thesis work utilizes APReL as the package for training the preference-based inverse reinforcement learning model. APReL is a library published by Erdem Biyik, Aditi Talati, and Dorsa Sadigh for active preference-based reward learning algorithms. It supports various techniques and prior research works related to preference-based learning. The library builds on previous research in active learning to generate queries to capture human preferences; they also employ batch active learning to reduce the time spent to generate these queries. [11]. Batch active learning increases the number of queries needed to learn the reward function, so to solve this, Biyik et al. introduced heuristics methods [48] and Determinantal Point Processes (DPP) methods [49] to maximize the information from fewer queries. These have also been implemented in the APReL library. Researchers can also include expert demonstrations into the learning process to improve performance.

The APReL library is built based on the OpenAI Gym environment and the progress of the agent is structured as an MDP. APReL is a preference-based algorithm that uses trajectory preferences. The aim is to learn the reward function which is assumed to be a linear function of state-action features [11]. Based on this, weights are learned for features of trajectories generated and not states or actions individually. This is because features of a states or actions could be interpreted differently at different time steps in a trajectory.

APReL is built in a modular structure to allow for switching methodologies and techniques in the model for different goals and use cases. For the query optimization process in batch active learning, the authors implemented different acquisition functions to optimally select the query to present to the user for maximum learning. The acquisition functions implemented in the package are mutual information [39] which seeks to select the query based on which can provide maximum information based on the learned weights and the user's previous response(s), disagreement, which is implemented based on [50], and this returns a pair of trajectories in the query that maximizes disagreement in their feature weights, regret [38] calculates the regrets between each pair of trajectory feature weights and returns the pair with highest regret, volume removal, which returns the expected volume removal from the un-normalized belief distribution,[51] methods.

In summary, APReL hopes to use human preferences and expert demonstrations to learn a reward function, optimizing to generate queries actively that increase information learned, yet batch generating these queries to reduce the computational capacity needed. While there are papers that have built on this and cited this work [52] [8], there is not published research at this time applying APReL as an active preference-based learning library in multi-context applications.

For this research, I utilized APReL in the learning implementation, exploring mutual information gain as a query sampling strategy. I also extended the APReL library, writing wrappers to expose the needed methods and attributes, and modifying the package to suit my environment, input cases, and learning goals.

2.7 Semantic map

A semantic map is a map of the environment that not only shows the presence or absence of items in the map, it also gives extra context and semantic reasoning to these items in the map. Most semantic map representation research has focused on spatial arrangements and topological maps, with some authors working on additional scene context about a map [53], for example, if the room is an office, a shop, or a kitchen. Semantic maps also provide the robot with factors that are comprehended by humans [54]. In this research work, the terrains in the environment are represented in a semantic map, with each terrain category represented uniquely.

Semantic maps can also be generated by extracting features from the environment using additional sensors, or manual labeling. Semantic maps are essential in robot navigation and task planning as it helps the robot learn better and behave more suitably. The authors in [55] used semantic mapping to improve trajectory classification to understand the behavior of objects that were being monitored. In their paper, feature selection from features extracted from trajectories was used to improve trajectory classification. This is because, sometimes, the raw trajectories may not give sufficient context that extracting certain features from the trajectory might give. Semantic labeling is not always prior, but can also be done in real-time, continuously, as the data becomes available from a data stream[56]. Semantic terrain classification has been explored for autonomous robot navigation [57]. I adopt a prior semantic labeling approach. Given that the environment is static and completely known, the semantic representation of the ground truth map is generated at the beginning.

2.8 Traversability map and analysis

Traversability maps are vital for off-road navigation because while the sensors might not detect an obstacle on a path in the map, that path might not be totally traversable for the robot [57] due to many factors. Some of these factors include the navigation goal, platform material, or velocity constraints of the robot. The authors in [58] generate the traversability map from a 3D lidar sensor and a camera, combining both. This is a preferable method for a static environment, that does not change much, as this method can be used to generate a ground truth map. The combination of data from both sensors also increases the accuracy.

Suger et al. propose a semi-supervised learning approach for the robot to learn a traversability map from the human operating it over an environment [59]. This method results in sparse positive labels and a large amount of unlabeled data. While this is handled by Positive Naive Bayes (PNB) [60] and Learning Classifiers from Only Positive and Unlabeled Data (POS) [61] learning methods, [62] suggests a self-training algorithm using neural networks for traversability mapping. The advantage of using the self-training algorithm is that it eliminates the need for labeled data.

Still on classification approaches, [63] treats the traversability map learning problem as a height classification problem. The authors build a Convolutional Neural Network (CNN) that predicts the traversability value of a terrain patch in an environment, given its height map.

2.9 Inverse Optimal Control (IOC) from human demonstration

As early as 1997, Atkeson and Schaal carried out research on robot learning from demonstration [1]. Inverse Optimal Control, which is similar to inverse reinforcement learning, is the technique of learning the cost function from an MDP system from human demonstrations [64]. Ziebart et al. propose this cost learning as a non-linear generalization of maximum entropy [28]. Wigness et al. adopt this principle of maximum entropy to learn reward functions from minimal human examples from robot navigation [65]. This learned reward function is tested using the Modified Hausdorff distance (MHD) metric [66] to compare performance against the ideal ground truth trajectory specified by the human expert. IOC is used to learn the rewards in linear MDP solutions [67] and non-linear systems [68]. An instance is [69] implementing a graph-based IOC algorithm for robot manipulation with 7 degrees of freedom (DoF). This algorithm uses a graph-based discretization of the trajectory space for learning from demonstrations.

While the IOC learning method from human demonstration faces the drawback of generating sufficient demonstrations for optimal learning, Finn et al. in [70] adopt a guided cost learning algorithm for non-linear cost functions based on policy optimization. The upside of this approach is that learning the costs can be combined with learning the policies for the costs which can improve the learning process for more complex tasks with few demonstrations. The guided cost learning algorithm is built on combining sample-based maximum entropy IOC [71] with forward reinforcement learning.

In summary, this thesis work builds on previous research using preference-based IRL algorithms to learn terrain costs and applies this to traversability and concealment affordances of terrains in the environment. I also conduct an experimental comparison between the PbIRL method and the IOC method [65] modified to learn the traversability and concealment affordances of terrains.

3 Methods

3.1 Problem statement

For a robot navigating autonomously in an adversarial environment, the goal is not to only maximize moving fast to a given position without running into an obstacle (traversability), it is also pertinent for the robot to stay undetected as much as possible (concealment) while doing so. How can this robot learn the affordances of the terrains in the environment to maximize traversability and concealment from an adversary based on the mission context?

This research thesis seeks to implement autonomous robot navigation in an adversarial environment, maximizing traversability and concealment, based on the terrain types in the environment and the location of an adversary, using preference-based inverse reinforcement learning technique. Traversability of a path is the ability of the robot to move over the terrains in that trajectory successfully without running into an obstacle, while concealment of a path is the ability of the terrains between that path and the adversary to keep the robot out of sight of the adversary.

I also experimentally compare the preference-based inverse reinforcement learning technique to the IOC learning method that is also applied in learning terrain affordances in an environment.

3.2 Conceptual approach

I propose to solve this problem by using active preference-based inverse reinforcement learning to train a model that learns the costs (terrain weights) for traversability and concealment, as outlined in Fig. 3.1. This learning method involves querying a human expert for feedback on their preferred trajectory (that best depicts the learning goal, either traversability or concealment) out of two choices and updating the model weights based on their selected preferred query. Traversability feature of a terrain is the ability of the robot to successfully and safely move across the environment on the different terrain types and the concealment is the ability of an object to stay undetected by the adversary given that the terrain is between that object and the adversary. After training, I combine the learned terrain weights, to generate a costmap for the robot's path planning from a start position to a goal.

3.3 Problem formulation

I formulated the problem as a set up with input as a set of random pre-generated trajectories from which the model queries a user and learns their preferences. A trajectory is a set of positions in the environment from a start position to a goal position. It can be modeled as a set of transitions between state, actions pairs where $(s_1, a_1) - > s_2$ with s_1 being the current state and a_1 being the current action when taken on s_1 that leads to the next state, s_2. The state space for traversability is the set {x, y} where x and y are the x, y positions on the occupancy grid, while the state space for concealment is the set of the {x, y} positions on the Occupancy grid and the terrains between the cell and the adversary, as described in Sec. 4.4. The action space is the continuous variable set of the velocity of the robot, based on different speeds and directions the robot can move in based on the current state position. Each trajectory has a feature

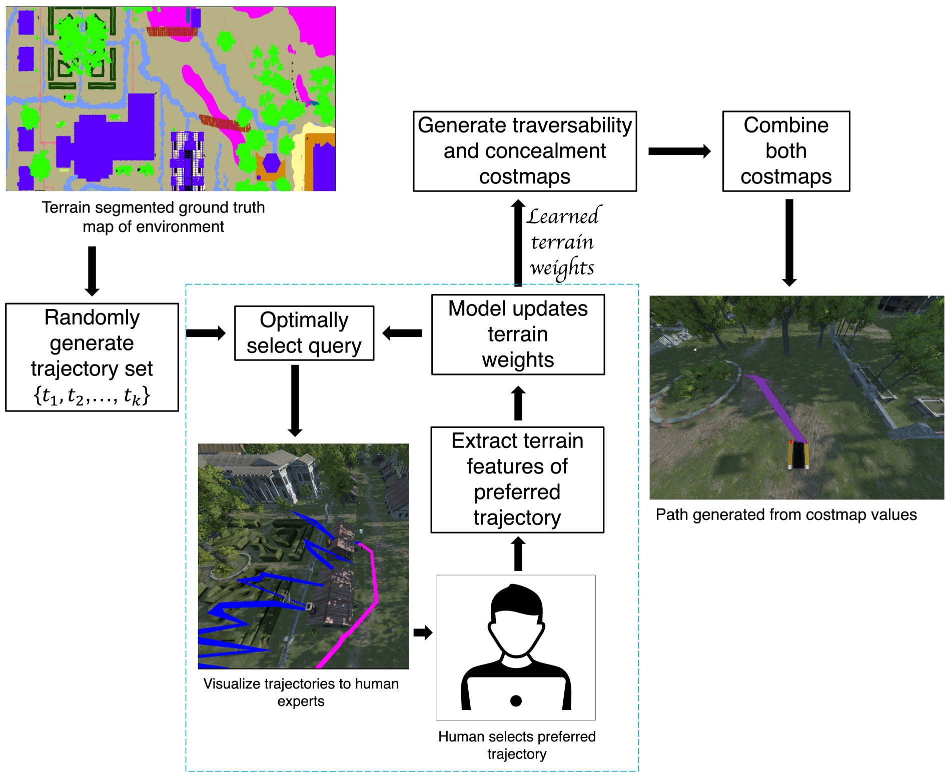

Terrain segmented ground truth map of environment
Randomly generate trajectory set $\{t_1, t_2, \ldots, t_k\}$
Optimally select query
Model updates terrain weights
Learned terrain weights
Generate traversability and concealment costmaps
Combine both costmaps
Path generated from costmap values
Extract terrain features of preferred trajectory
Visualize trajectories to human experts
Human selects preferred trajectory

Figure 3.1: Design of the Preference-based learning system for terrain and adversary-aware autonomous robot navigation

set (the terrain types in the trajectory) and the model tries to learn weights over the features by querying the users and getting their preferred trajectory at each point.

3.4 Path planning

Path planning is an aspect of robot navigation that generates a path from the robot's position to a goal region or position. This is done by first receiving the goal position and any other related parameters, such as the allowed deviation from the exact goal position that still counts as successful goal navigation and the algorithm to use in calculating the path. Path planning is done by the navigation stack which is usually made up of the global planner and the local planner. This thesis is implemented on the Army Research Laboratory (ARL) Autonomy Stack, explained in Sec. 4.1.5, which encompasses the planners and costmaps as discussed in Sec. 3.4.1 and Sec. 3.4.2.

The global and local planners use the costmap values for each cell and try to generate a path, as seen in Fig. 3.2, from the start position to the goal with minimal cumulative terrain costs over the path. This costmap is based on the ROS data class as described in Sec. 3.4.2. In the base case–where the costmap is derived from the obstacles in the environment, the costmap values are either the minimum value of the valid ROS2D costmap values (to denote the absence of an obstacle) or the maximum value (for presence of an obstacle). This devolves into finding the shortest euclidean path from a start position to a goal. The costs, in this case, are not based on the terrain types, only to determine if such terrain is an obstacle (lethal) or not. In my approach, the costmap values are derived from the traversability affordance of the terrain in that position and the concealment affordance of the terrains between that position in the environment and the adversary.

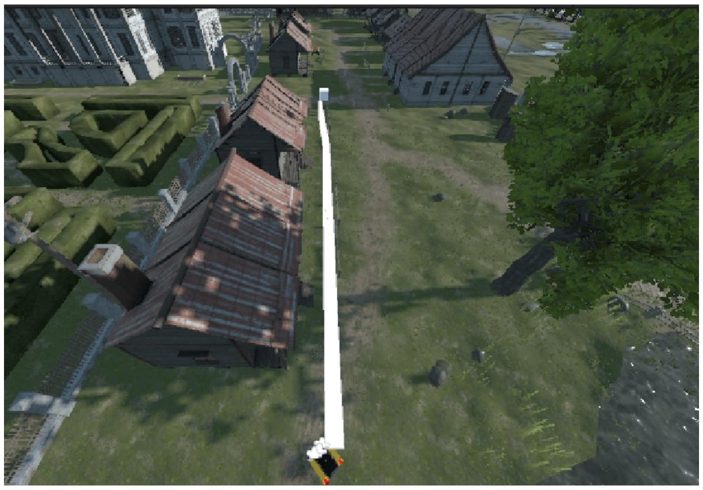


Figure 3.2: Path from robot to goal position visualized in simulation environment

3.4.1 Global and local planner

The global planner tries to generate a feasible and collision-free path from the robot to the goal location while the local planner receives the path from the global planner and generates a local path (which is a sequence of shorter paths) to the goal. The global planner takes in account the general constraints of the platform, e.g. path curvature, maximum speed. The path produced by the global planner serves as a guide for the robot on the direction to navigate and for the local planner to create a local path that is more detailed and adapts as the robot moves closer and obtains a better perception of the environment in that area if obstacles in the environment move. The local planner generates a segment to best match the next portion of the global path that is feasible and executable and the motor controller attempts to execute this local path. If the path generated by the global planner is too close to an obstacle, is not feasible, or something in the environment changes after the global planner has generated the global plan, the local planner deviates from the global plan and generates a new local plan to avoid it. While the global planner is not always super detailed in accuracy, the local planner focuses on a smaller portion of the map per time and gives more detailed accuracy of obstacles around the robot in a certain location of the map. The local planner changes more frequently and updates with new information as the robot moves.

3.4.2 Costmap

The costmap is a 2D or 3D representation of the world map as an occupancy grid. It is a representation of an environment showing the obstacles in the map, the areas around these obstacles that are inflated based on the inflation radius of the robot, and the free areas of the map that the robot can navigate through. The obstacles are inflated so that the global planner can simplify path planning by treating the robot as one single point in the path generation. The inflation also ensures that no part of the robot collides with the obstacle at any point.

The costmap values are a set of values ranging from the minimum representing the probability that the corresponding position in the environment is definitely free to the maximum representing the probability of the position being definitely occupied. These positions are discretely split into portions and can be represented in different forms, such as, grid cells. A cell with a costmap value denoting that it is definitely occupied is called lethal. An inscribed cell is a cell within the robot's inscribed radius around a lethal cell. The inscribed radius of a robot is the distance from the robot in which an obstacle needs to be for no part of the robot to collide with any part of the obstacle.

The costmap is passed into the global planner which the robot uses to plan a global path from one position to another through the map as described in Sec. 3.4.1.

3.5 Assumptions

For my approach, there were some assumptions made in designing the problem. These assumptions are on the robot platform type, the environment, and the adversary. Most of these are outside the scope of this work so I built upon these assumptions. I assume a perfect perception and knowledge of the terrain, a prior map of the environment, and elevation map. This is vital as affordances for different terrains can also be affected by elevation and depth. Another assumption is that the robot has a 360 degrees field of view and a perfect perception of the adversary's location. The adversary, on the other hand, is assumed to be static and has a 360 degrees field of view as well.

3.6 Learning

My approach is to use an active preference learning method to learn the terrain affordances. As mentioned in Sec. 2.5, active learning is appropriate because it is suitable for maximizing information from unlabeled data in learning a target concept by querying a human expert [36]. For this work, I chose to use APReL library, and modify it for my purposes.

The learning problem is set up as an MDP and random trajectories are generated, by generating deviating paths from the global planner, from a start position to a goal position to be passed into APReL for learning. APReL initially was built to work with OpenAI Gym Environment [72], so to utilize APReL in the the autonomy stack (described in Sec. 4.1.5) and simulation environment used in this research, I created wrappers to expose the objects and methods I intend to use. The randomly generated trajectories make up the discrete trajectory set for learning. For each cell in the environment, I define a linear function over the terrain weights for each of the cells. The linear function represents the distribution of the terrains present in each cell, summing up to 1. This accounts for cells that contain multiple terrains, although such was not present in my implementation.

The terrain occurrence of a cell, $\phi(s_t)$, a vector with a dimension of the number of terrains for learning. This vector is obtained from this linear function and it represents the occurrences of all the terrains in that cell, as outlined in Eqn. 3.1.

$$\phi(s_t) = \sum Tr_{s_t} \tag{3.1}$$

where Tr_{s_t} is the array of terrain types for cell, s_t with values assigned to each terrain type based on its presence in that cell in the terrain ground truth maps. With the trajectory set being a set of k trajectories, $\{T_1,\ T_2,...,T_k\}$, the traversability and concealment costs are defined separately for the trajectories. The traversability cost is calculated as a function of the cells a trajectory passes through while the concealment cost is calculated as a function of the cells in a trajectory and every terrain/object in between that cell and the adversary.

Iteratively, the APReL package [11] uses a batch active learning method by query optimizing to present a human expert with a query of two trajectories from the trajectory set, asking for their preference. Based on the human expert's response each time, the belief model over the weights for the terrains is updated, and APReL optimally selects the next query from the trajectory set based on what the model has learned so far and the human expert's responses over time.

After each query iteration, the terrain features for the preferred trajectory selected by the human expert are calculated, and the model updates the traversability and concealment weights based on this.

The traversability cost for a trajectory is the sum of the normalized dot product of the learned traversability weights of each terrain type and the terrain type of each cell in the trajectory.

$$T_{traverse\_cost} = \frac{1}{N}\sum_{t=1}^{N} \omega^{TC}\phi(s_t) \tag{3.2}$$

In Eqn. 3.2, N is the size of the trajectory, ω^{TC} is the learned traversability weight of the terrain in $\phi(s_t)$ which is the terrain type of a cell s_t in trajectory, T.

For concealment, $\phi(s_t)$ is based on the terrains/objects in between the cell s_t and the adversary in the environment, Eqn. 3.3.

$$T_{conceal\_cost} = \frac{1}{N}\sum_{t=1}^{N} \omega^{CC}\gamma(s_t) \tag{3.3}$$

N is the size of the trajectory, ω^{CC} is the learned concealment weight of the cell $\gamma(s_t)$ which is the concealment obtained from the terrains between the cell, s_t, in trajectory, T, and the adversary.

The overall procedure for learning these costs is described in Alg. 1

Algorithm 1 Learning traversability and concealment weights for terrains

```
Precondition: $m$ ground truth terrain maps, start and goal positions, adversary location
Postcondition: learned weights for traversability and concealment for each terrain

  Initialize $A$, a set of $p$ randomly generated trajectories from robot position to a goal position
  Initialize $X$, a vector of random weights with dimension $m$.    ▷ m is the number of terrains
  X:= [$x_1$, $x_2$, $x_3$, ... , $x_m$]
  function LEARNING($A, X, N$, mode)
      Select two random trajectories from the trajectory set, $A$
      for $i \leftarrow 1$ to $N$ do                ▷ N = number of query loops for training
          Query human expert for preferred trajectory from the two selected
          if mode == traversability then
              Extract the terrains of cells in the preferred trajectory
          else if mode == concealment then
              Extract terrains in between the cells in the trajectory and the adversary
          $X \leftarrow$ update_terrain_weights()    ▷ Based on the terrain features of the preferred trajectory
          Next query $\leftarrow$ mutual_information($A$)    ▷ Optimally select the top two trajectories in $A$, based on $X$, as the next query
      return $X$
```

Traversability and concealment weights are learned separately and will be combined to generate a costmap.

3.6.1 System design for learning

Since the problem is designed as an MDP, each transition step has a state, and taking an action from a set of possible actions leads to the next step.

- States: For traversability, the state is the position of the cell (x, y, z, ψ) in the environment. For concealment, the state is the position of the cell in the environment, the adversary pose, and the ray cast data of all objects between that cell to the adversary position.

- Possible actions: The possible actions to be taken at each transition step are defined as a continuous range of acceleration from $0m/s^2$ to the maximum acceleration of the platform in the direction $\in$ [0°, 360°] inclusively.

- Features: Each trajectory has a feature set. For traversability, the features of a trajectory are the terrains present in the cell positions in the trajectory. However, for concealment, these features are the terrains and objects present from ray casting from each cell in that trajectory to the adversary location. This specifies the items in between the cell and the adversary.

- Output: Learned knowledge of traversability and concealment affordances for each cell, based on the terrains, context, and adversary location in the environment. These learned terrain weights are then used to build the costmap which can then be passed to the global planner to generate paths to goal positon, as discussed in Sec. 3.4.1.

3.7 Traversability and concealment weights

Traversability and concealment have separate weights for each terrain. These weights are learned from the preference-based active querying loop. The final weight is a vector of m normalized weights, each per terrain type. The values of these weights go over a continuous range. These terrain weights are used to generate the traversability and concealment costmaps.

3.8 Costmap generation

From the learned terrain weights, I use a `weights_to_cost` function in Alg. 2 to compute a dot product between the terrain weights and the terrain feature maps to obtain costs for each cell making up the traversability costmap. For the weights to follow the same distribution, I normalize the learned traversability and concealment weights using a `normalize_weights` function that normalizes both weight vectors to a scale of [0,1] for the corresponding [min, max] weights in the vector. This is important for the subsequent step of combining and optimizing these costmaps. For the concealment costmap, the cost for each cell is calculated by combining the weights of the terrain types in between the cell and adversary (from the concealment values obtained by ray casting, as explained in Sec. 4.4) and normalizing the result.

The process for generating the traversability and concealment costmaps is described in Alg. 2

The costmap is published as an Occupancy grid type which is then converted as a costmap layer described in Sec. 4.3. Normalizing the weights learned before computing the costmap values from the weights allows for the traversability and concealment data to follow the same distribution. This results in a non-skewed/balanced combination of the costmaps.

3.9 Balancing the costmaps layers

Given that the traversability and concealment costmaps have been generated, the next phase in the process is to combine these costmaps, allowing for balancing traversability and concealment. For this thesis, I combined the costmaps by assigning weights to the traversability and concealment costmap, and summing both to get the final cost. I chose this combination method because it is generalizable and the weights

Algorithm 2 Generating costmaps from learned terrain weights

```
Precondition: $m$ ground truth terrain feature maps, vector of learned terrain weights,
  $X$, with dimension, $m$
Postcondition: traversability and concealment costmaps of the environment

  Initialize costmap, $C$ with dimensions, w x h

  function WEIGHTS_TO_COST($X$, $m$ terrain maps, mode)

    $m\_sum \leftarrow \sum X$

    $X \leftarrow \frac{x_i}{m\_sum} \; \forall \; x_i \in X$                ▷ Normalize the weight vector

    $S \leftarrow$ inverse mapping of the normalized reward values to costmap values

    for $i \leftarrow 1$ to $w$ do

      for $j \leftarrow 1$ to $h$ do
        if mode == traversability then
          C[i][j] = $S \cdot \phi$  ▷ $\phi$ is the terrain maps for each cell in the environment
        else if mode == concealment then
          C[i][j] = $S \cdot \gamma$          ▷ $\gamma$ is the terrain(s) between each cell in the
  environment and the adversary
    return $C$
```

assigned to traversability vs concealment can easily be adjusted based on the mission context and goal. After the costmaps have been combined, a single costmap is produced and passed to the global planner for path planning and navigation.

3.10 Model performance evaluation/assessment

After training to learn the terrain weights for traversability and concealment, the respective costmaps are generated, and the performance of the PbIRL model from my research work will be evaluated. I evaluate the traversability and concealment behavior of the PbIRL model and experimentally compare the PbIRL model with an already-existing IOC learning from demonstration model [65] trained on the same terrains as well.

The preference-based model will be evaluated against the IOC model on the following metrics.

- Proportion of the path in which the robot is being observed by the adversary. Aim to minimize exposure/unconcealed portion of the robot path.
- Distance covered by the robot in trying to reach a goal position. Aim to minimize traversability distance.
- Rate of failure/collision: Number of times the robot crashes into an obstacle or the number of times the global plan fails and the global planner needs to re-plan.

4 Experiments

4.1 Experimental implementation

4.1.1 Experimental environment

The environment used to train the model in simulation is the Flooded Grounds environment in Unity, as seen in Fig. 4.1. This environment is a 1280m x 1280m plane made up of water plane, trees, buildings, sand, stones, rock, walls, fences, boats, pipes, and other artificial objects, with minimal elevation.

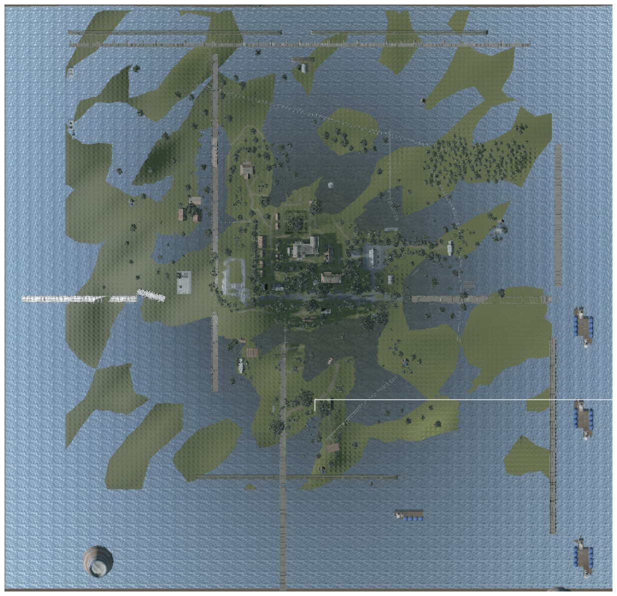
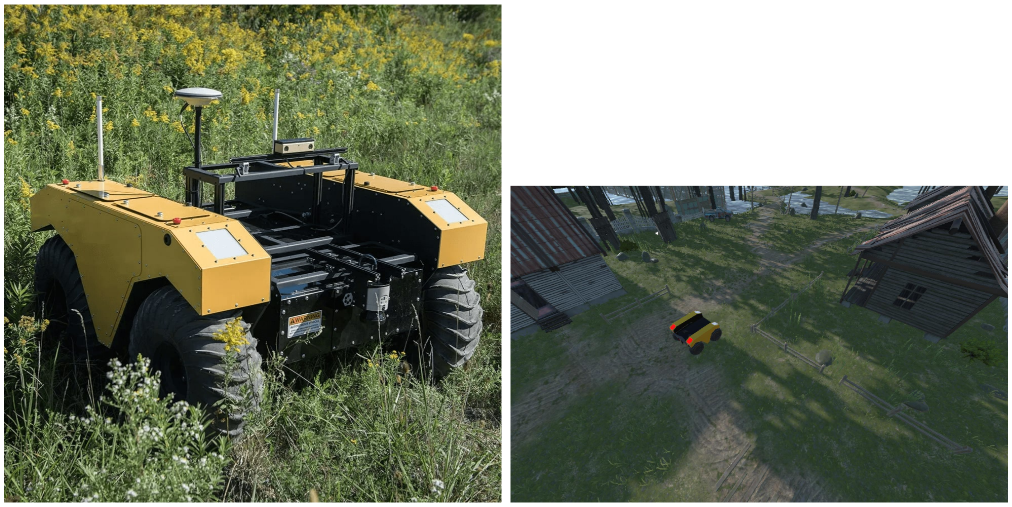


Figure 4.1: Orthogonal view of the Flooded Grounds environment in Unity

4.1.2 Robot platform

The simulation robot used in my implementation is the Clearpath Warthog Unmanned Ground Vehicle robot platform, Fig. 4.2. Its dimensions are 1.52 x 1.38 x 0.83 m, with a base weight of 620 lbs inclusive of the battery, a gross weight of 1300 lbs, and a max speed of 11 mph. For training, the robot is treated as one single agent.



(a) Clearpath Warthog Unmanned Ground Vehicle Robot in a real world field (b) Warthog robot in the simulation environment

Figure 4.2: Clearpath Warthog Unmanned Ground Vehicle Robot

4.1.3 Unity

The environment for training and testing of the research is visualized in simulation in Unity. Unity is a cross-platform development platform for simulating and developing 2D and 3D games and environments. The simulation and development was done in the Unity Editor version 2020.3.36f1. The environment is loaded into Unity from the Unity Assets and there are scripts that execute when the engine is run to carry out different tasks. Some of the scripts added to the Unity project include the script to generate a semantic segmented image of different terrains in the environment and another script for obtaining the ray cast data from each cell in the environment to the adversary position.

Because most of the model development and robot software engineering is done in ROS and Python, Transmission Control Protocol (TCP) server connection was created to link Unity to the ROS Master server for the exchange of data between both systems. This TCP server connection sends data from Unity to ROS and vice versa. The connection is called to visualize trajectories in Unity based on ROS poses, and also carry out analysis and processing of the environment in real-time in ROS nodes. However, since ROS and Unity operate in different coordinate systems[1], Unity positions in the environment needed to be converted to ROS positions, and vice versa, as seen in Table 4.1:

Table 4.1: System coordinates conversion from Unity to ROS

Unity pose	ROS pose
x	-y
y	z
z	x

4.1.4 Adversary

The adversary is a single, static agent with a known exact location in the environment and relative to the robot. The adversary has a 360 degrees field of view with infinite line of sight. For the purpose of my experiment, the adversary was represented in simulation as a second ground platform model. This was done to facilitate computation of terrain concealment (see Sec. 4.4 on raycasting) and performance metrics (see Sec. 3.10).

[1]https://github.com/siemens/ros-sharp/wiki/Dev_ROSUnityCoordinateSystemConversion

4.1.5 Autonomy stack

The autonomy stack used in this implementation is the ARL Autonomy Stack. This is a set of algorithms and ROS nodes that provide the basic building blocks, like mapping, perception, intelligence, and planning, that make up an autonomous system for a ground platform. These building blocks are developed as subsystems and integrated into a monolithic system for cross-functionality of the entire architecture. This research work interfaces with the planning subsystem by generating the costmap which is passed into the global planner for path planning and autonomous robot navigation in the environment.

4.1.6 Global planner and costmap

The selected global planner is the Search-Based Planning Library (SBPL) [73], which is a wrapper built around the ROS SBPL package[2]for global path planning, and the default local planner is Model Predictive Path Integral (MPPI) [74]. The SBPL planner generates a path to the target location with the smallest cumulative cost irrespective of path length, and not the shortest unoccluded path. The SBPL planner also has a `max_envelope` parameter that sets how wide robots can deviate from the path without causing a replanning.

4.2 Terrain ground truth map

As one of the assumptions in this research is a prior ground truth map of the environment and terrains, the terrain ground truth map was generated categorizing all the terrains present in the environment on a map image for an accurate representation of the environment in the ground truth maps. This was done by attaching an overhead camera that takes an overhead snapshot image of the environment and attached

[2]http://wiki.ros.org/sbpl

an available C# script to the camera objects in Unity for semantic segmentation that categorizes each object and terrain type based on set segmentation labels in the output image. The resulting image was used to generate a terrain map of the environment which was loaded as the ground truth map. Of the 4,700 objects present in the environment, over 2,000 were untagged, so I manually tagged the untagged objectes, added semantic layers categorization for objects that were missing those, and created additional segmentation labels for the terrain labels that were not originally listed in the semantic segmentation manager asset.

After generating the segmentation image, as seen in Fig. 4.3 from the overhead camera with each unique segmentation label being assigned to each unique terrain/object in the environment, a Python script was implemented to split the image into individual images per terrain type. These images serve as the ground truth images for each of the terrains. The images were converted from `jpg` images to `pgm` images to achieve compression without losing pixel quality, with corresponding configuration files of the image parameters such as resolution, origin, image file name, etc. to properly load the images as maps and publish them as Occupancy Grids to ROS topics. Then, a base map server node in the ROS launch file linked to the configuration files for each terrain type to load these terrain ground truth maps. This publishes the terrain base maps to ROS topics encoded as binary occupancy grid maps as described in [65].

4.2.1 Overhead camera

To achieve maximum details of the environment and capture minute terrain objects, the ground truth map image was generated by taking overhead snapshots of sixteen different portions and merging the sixteen images into one final overhead image. The camera was placed 10m above the environment and the x and z position values in Unity were adjusted before each photo to traverse the environment totally from bottom

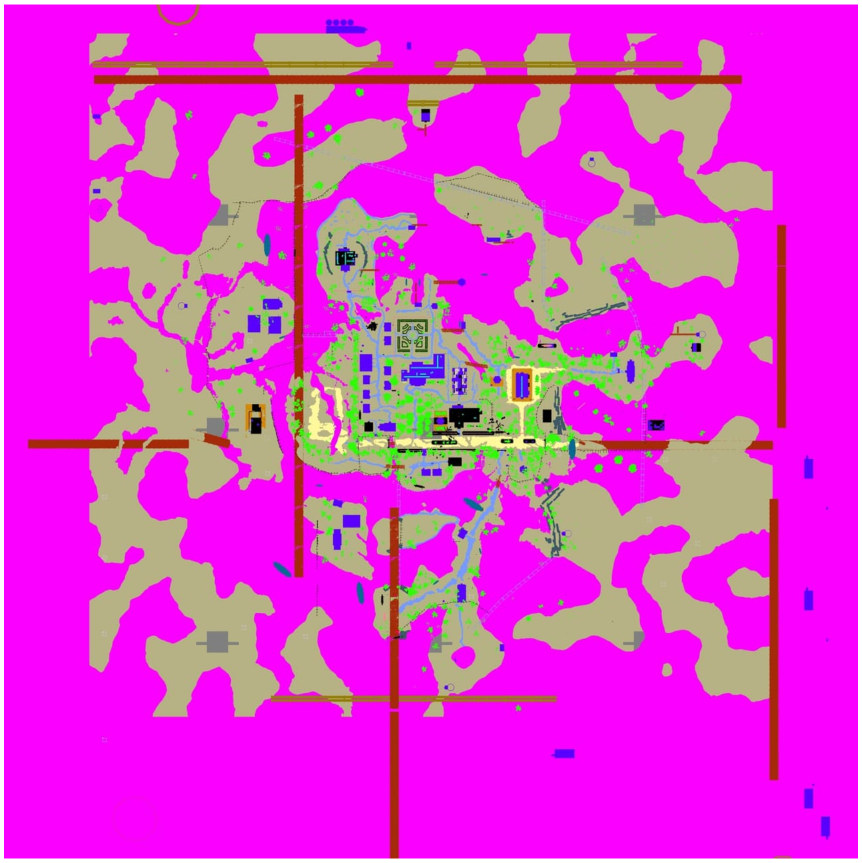


Figure 4.3: Ground truth map of the Flooded Grounds environment (seen in Fig. 4.1) from Unity

left to top right. The camera had a Field of View (FoV) of 100° and the overhead images produced were 3200 pixels x 3200 pixels images. After obtaining the overhead images of the entire portion of the environment, the images were combined into the ground truth map of 1280 pixels x 1280 pixels dimensions, as seen in Fig. 4.3.

4.2.2 Segmentation map

The terrain segmentation maps are individual image files extracted from the ground truth map generated by the overhead camera. These terrain segmentation maps are created by splitting the ground truth map into individual images per terrain type in the ground truth map. Each terrain type was represented in the ground truth map image by a unique set color and in the individual terrain map images, an image mask was applied, so white color pixels show the presence of that terrain in the position, while the rest of the image is black.

The ground truth map consisted of twenty-five terrain types: tree, bush, building, wall, fence, sidewalk, pipe, pole, boat, bench, grass, flower box, factory, roundabout, rock, gray sand, blue sand, yellow sand, water, bridge, tower, billboard, door, vehicle, stones. Fig. 4.4 shows an upclose snapshot of the segmentation map with trees, bush, buildings, water, and sand present in that portion.

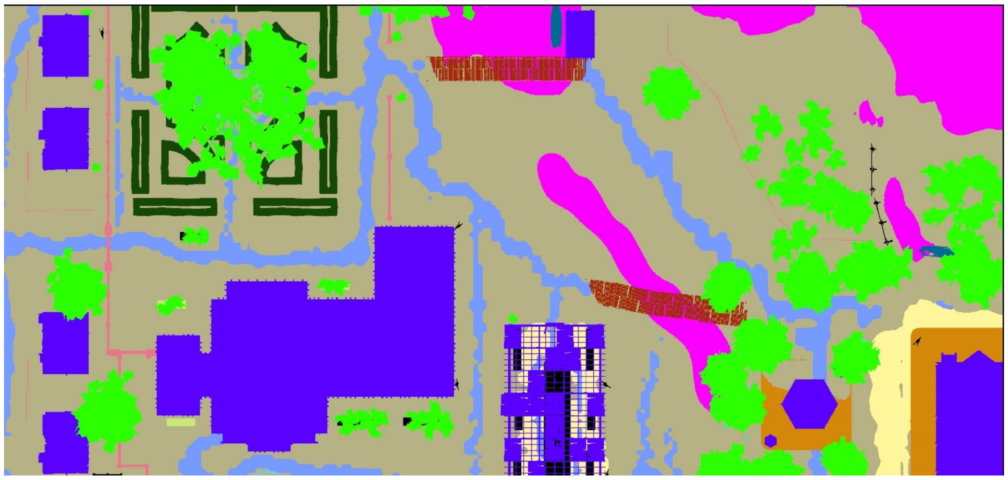


Figure 4.4: Upclose semantic segmented image a portion of the environment showing the terrains closely

Out of these twenty-five terrains, the weights of these five groups of terrains were learned:

1. Tree and bush
2. Building, sidewalk, wall, and fence
3. Water
4. Gray sand, yellow sand, and blue sand

5. Rock

The other terrain types were extremely sparse in the environment and insufficient for model training.

4.2.3 Depth map

To obtain elevation and depth in the environment, there is an overhead depth camera alongside the Red Green Blue (RGB) overhead camera that generates the semantic segmented images, seen in Fig. 4.5. This overhead depth camera utilizes a depth shader to represent the progression of elevation across the environment by rendering depth of 3D image in 2D. This depth shader represents the different elevations in the environment using a gradient of color intensities.

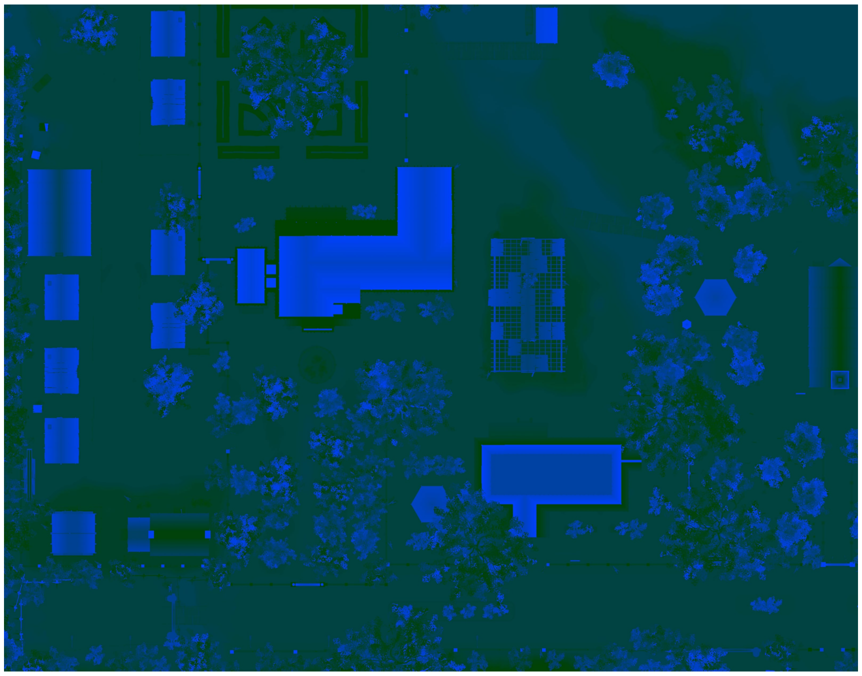


Figure 4.5: Depth map of a portion of the environment showing elevation

Considering that the depth map generated is an image representation and does not return the discrete values of the elevation, a function in the ray cast script was implemented to return the exact elevation/depth of each position in the environment. This function goes through the entire environment in Unity and ray casts directly from positive infinite height position to the ground, recording when the ray hit the ground surface, to get the height of each cell in the environment. This function moves a game object across the entire environment, placing the camera at each cell, sends a line cast at 90° vertically, and returns the distance at which the line cast hit the ground/base terrain. This determines the height of that position in the environment.

4.2.4 Occupancy Grid

The ground truth map and terrain maps are loaded in a ROS launch file and published to base map topics as Occupancy grids. Typically, ROS occupancy grids are used to show the presence or absence of an item on the map. Occupancy grids are a 2D array[3] with a size of $width \times height$. This map height and width can be static (when there is a prior entire map) or dynamic (when the map gets built as the robot scans an environment). For dynamically built maps, the occupancy grid size also changes as the map grows. The ground truth maps loaded are static and do not change in size.

For the terrain maps, the occupancy grids have either values of 0 or 100. A value of 0 in the terrain map grid represents the presence of that terrain in the cell, while a value of 100 represents the absence of the terrain in that cell. The occupancy grid is a 1280×1280 static array, with each cell representing a $1m \times 1m$ grid cell of the environment. The occupancy grid is fixed and does not change in size.

4.3 Implementation process

The environment is loaded in simulation in the Unity Editor and the overhead image with segmentation labels for each terrain is generated. This image is converted

[3] http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html

into a map and loaded into ROS as the ground truth map with each terrain published to its map topic with the exact positions of each terrain on the map known.

For concealment, the adversary is placed in the environment at a known location, then a C script ray casts from each cell in the environment to the adversary, assuming a 360° FoV of the adversary. This script saves the ray cast data for each cell in the environment based on that given location of the adversary. The current assumption is a static adversary, dynamic adversaries is possible future work that can be extended from this thesis. The robot spawns in the environment at a specified location and the model initializes the preference-based learning by pre-generating trajectories from the robot position to a goal position, as seen in Fig. 4.6.

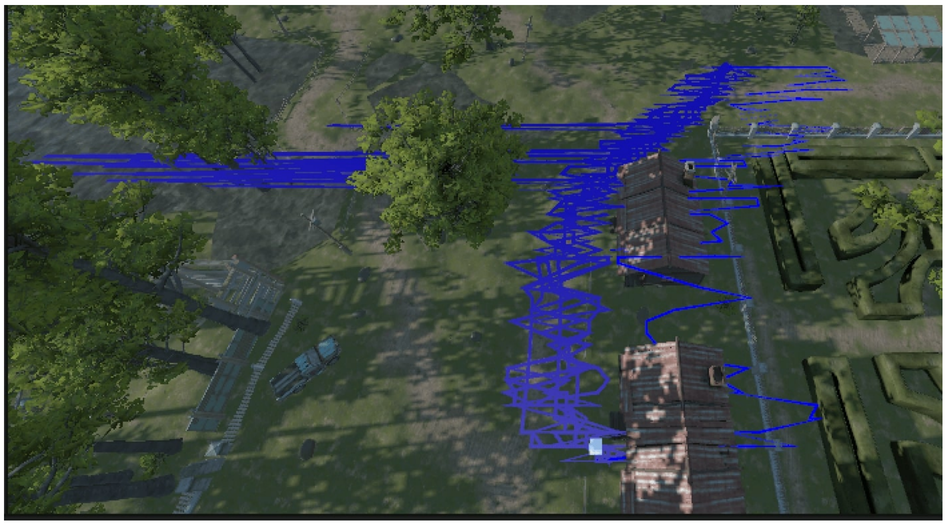


Figure 4.6: Random trajectories generated from robot to goal position for terrain learning

Each trajectory has a feature set over which the model needs to learn feature vector weights. To do this, the model iteratively visualizes two queries per round and asks for the user's preference, updating the belief model based on the user's response until the model learns the weights over these features and the weights converge.

The learning is done differently for traversability and for concealment for five select terrains (trees/bush, buildings/walls/fences, water, sand, stones/rock). Fifteen random trajectories are generated from a start position to a goal position as the trajectory set. Each trajectory has an average of eighty unique positions in the trajectory. APReL optimally selects the top two trajectories from the trajectory set in each query loop that can maximize the information learned for a faster model convergence. When the model has learned, these weights are used to generate a traversability costmap and a concealment costmap. These costmaps are passed as costmap layers to the global planner for path planning.

To combine these costmap layers, it was imperative to define rules for obtaining the final cost of each cell from these various layers. The rules could be determined on a map/environment/terrain basis, based on a mission goal, or be dynamic (adapt this mid-way as goals/priorities may change). For my approach, I placed the traversability costmap layer above the concealment layer because the traversability costmap is used as the base obstacle costmap before compounding with the concealment cost. Dynamic adjustment of the costmap layers rules is in consideration for possible future work.

For testing and experimentation, the path generation is based on the learned terrain weight. The experimentation process begins with loading the terrain ground truth map and extracting terrain features of each cell in the environment. A dot product of these terrain features and their corresponding weights for each terrain feature is calculated for each cell in the map. This results in the traversability and concealment costmaps which are combined into one costmap as passed to the global planner, as explained in Sec. 3.4.2.

4.4 Ray casting

A point on the map is unconcealed if the point is in line of sight of the adversary. 3D object detection could be used to determine this from the adversary to a target location. 3D detection of objects in an environment is essential for autonomous robot navigation [75] and research has explored using machine learning and deep learning algorithms to detect objects in 3D for feature learning from point clouds [76]. Advanced object detection, classification, and perception are outside the scope of this work, so ray casting in Unity was applied to get the concealment features of the different cells in the environment. The ray casting mechanism was adopted because it is more precise, given it can perfectly tell if a ray can pass from the adversary and hit at the target location, in simulation. It is also seamless to integrate with my environment and simulation engine.

The concealment values of each cell are dependent on the adversary's location and the values change if the adversary changes position in the environment. Concealment values of a cell are the terrains and objects between a cell in the environment and the adversary found by ray casting. This data is obtained by ray casting directly to the ground in each cell to get terrain height, placing a game object $1m$ above terrain height, because the robot platform used for training is 1m in height, and we assume that the tallest point on the robot will be most detectable. Then, a ray is cast from each cell to the adversary position to get the ray cast values for the cells. A cell is 1m, so this is performed for all 1280×1280 $1m$ cells by moving the game object by 1m until the entire environment was covered.

This ray casting function returns every object and terrain between the cell and adversary, excluding the robot parts like wheels, etc., and the objects or terrains within 10m of the adversary location. This is based on the assumption that if a terrain is within 10m of the adversary, and provides concealment for the robot, it is of low utility for concealment, as the adversary can quickly move around this terrain because of the close proximity and sight to the robot. The ray cast function also returns for each terrain hit by the ray that was cast, the terrain tags and layers from Unity, the position point of the terrain, distance from the cell which the ray was cast from, alongside the terrain name. The final ray cast data was saved to a text file which the ROS node read from for concealment learning and concealment costmap generation.

4.5 APReL

APReL, a Python package for implementing active preference-based reward learning algorithms [11] was used for learning. I downloaded the source code from GitHub and extended it by writing wrappers for this package to expose certain classes and methods in the library to work with our use-case and input data format. APReL is configured for simpler environments, so it had to be modified to work with a much more complex environment and application. The adapted methods for this research work were the feature extraction function to extract the terrain features of a given trajectory, the attributes of a trajectory (i.e, the poses in the trajectory), and the visualize method to visualize the trajectories in Unity while learning from the user's preferences.

4.5.1 Human model

APReL either works with a simulated human or a real human giving their preferences. The PbIRL model was trained with a real human expert giving the responses to the queries. The selected trajectories are visualized and the human expert is presented a binary query, which they respond to, selecting their preferred trajectory from the visualized trajectories in that interaction loop.

The user model used to learn the reward function from the human responses is a softmax human response model. This means that the responses from the human follow the softmax choice rule [77], where the human expert chooses their preferred trajectory in direct proportion with the maximum reward they expect to be learned from that trajectory [78] [79]. In this case, either for traversability or concealment, the human selects their preferred trajectory out of the two that is shown to them, that maximizes either traversability or concealment depending on the learning goal.

4.5.2 Belief model

The belief model used is the Sampling Based belief model which inherits a linear reward belief model. The sampling-based belief model is a posterior distribution model that keeps all user feedback to learn the posterior values of the terrain weights [11]. Based on these posterior values [80], samples of the weight parameters are sampled from these using the Metropolis-Hastings algorithm [81] for Bayesian learning [82]. The mean of this belief distribution, obtained from the mean of the samples generated by the Metropolis-Hastings algorithm, is returned as the terrain weights after each learning loop.

4.5.3 Query and preference

The query type used was the preference query which presents the human expert with two trajectories and the human is asked for their preferred trajectory from both. While APReL allows for this query type to present the user with as many trajectories as they want, I opted for two trajectories to reduce the cognitive load on the human [83] to rank multiple trajectories or compare multiple trajectories to choose their favorite. The preference class contains the query presented to the user and the response the user gave.

4.5.4 Query optimizer

APReL implements a query optimizer for the active learning process that optimally selects the top queries to present to the user for which the model can get the most information and learn the reward function faster [39] [11]. This query optimization is computed over the discrete set of trajectories. The optimizer returns a trajectory from the trajectory set that best fits the user based on the user model given.

To find the top queries, the acquisition function works with the query optimizer to decide the information value. Mutual information [39] was the acquisition function used in thesis. This acquisition function takes in the belief model and a query of two trajectories to ask the user. It returns the maximum information value as a float. Maximizing this value is necessary for efficient learning. The query optimizer greedily searches exhaustively for the top queries based on this value from the acquisition function. After each query and response loop from the human expert, the belief model is updated, and for the next active querying loop, the query optimizer calls the acquisition function to find the trajectories that will maximize information learned and greedily searches the trajectory set to select the trajectories for the next query based on that.

4.6 Visualizing trajectories

In order for the human expert to have accurate representations of the trajectories and give the best responses to the learning query, there was a visual interface representation of the environment and trajectories presented to the human expert when querying them. Each trajectory runs over positions in the environment, each position having a terrain type. Each querying loop extracts the terrain features of the trajectories presented and updates the model weights based on the terrain features of the selected preferred trajectory.

A new custom ROS message type was created, imported into the Unity Editor, and built. This enables me to send data in the format of this custom ROS-Unity message between ROS and Unity servers. The custom message was named, Poslist, of type

```
float32[] positions
```

which it is an array of float32 positions which are (x, y) positions from ROS. Then, a ROS Transmission Control Protocol/Internet Protocol (TCP/IP) connector was built that connects to the ROS master server, subscribes to the custom ROS messages created, and publishes the data received in Unity. This connector and the custom message generation in Unity were built using the ROS TCP connector package[4].

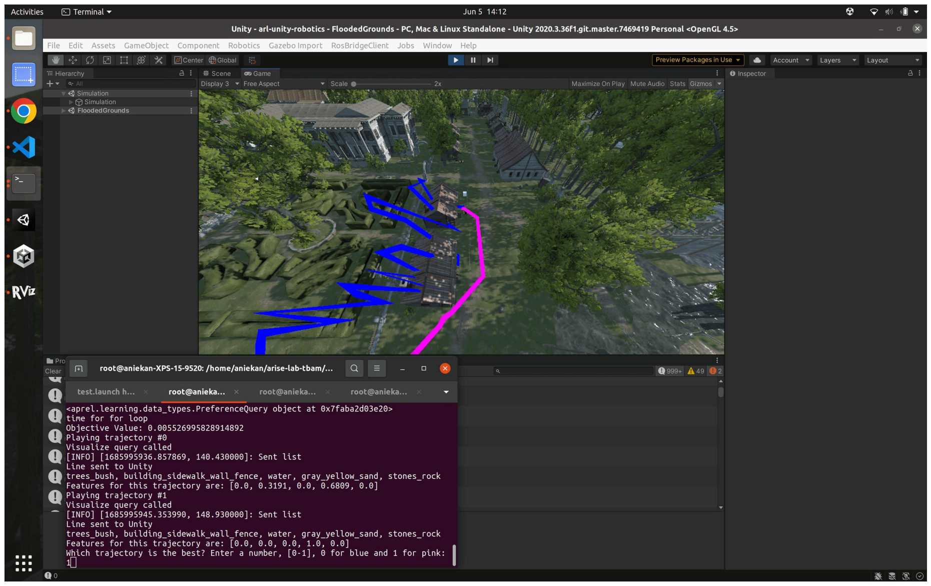
In each querying loop, when the model actively selects the two trajectories from the trajectory set, the positions in the selected trajectories are sent via the custom message to the Unity editor, and two lines are visualized in the environment in Unity in two different colors (blue and pink), which can be seen in Fig. 4.7.

The human expert observes the lines of the trajectories for that query visualized in Unity and responds to the query by typing in their preferred trajectory as input in a command line terminal. The model, then, extracts the terrain features of the selected trajectory to update the model weights and learn as seen in Fig. 3.1.

4.7 Experimental setup

For experimentation, the Preference-based trained model was compared in performance to a previously published IOC model [65] which learns from demonstration. The PbIRL model is evaluated against the IOC model on the set metrics as listed in Sec 3.10.

[4]https://github.com/Unity-Technologies/ROS-TCP-Connector


Unity - arl-unity-robotics - FloodedGrounds - PC, Mac & Linux Standalone - Unity 2020.3.36f1.git.master.7469419 Personal <OpenGL 4.5>
root@aniekan-XPS-15-9520: /home/aniekan/arise-lab-tbam/...

```
<aprel.learning.data_types.PreferenceQuery object at 0x7faba2d03e20>
time for for loop
Objective Value: 0.005526995828914892
Playing trajectory #0
Visualize query called
[INFO] [1685995936.857869, 140.430000]: Sent list
Line sent to Unity
trees_bush, building_sidewalk_wall_fence, water, gray_yellow_sand, stones_rock
Features for this trajectory are: [0.0, 0.3191, 0.0, 0.6809, 0.0]
Playing trajectory #1
Visualize query called
[INFO] [1685995945.353990, 148.930000]: Sent list
Line sent to Unity
trees_bush, building_sidewalk_wall_fence, water, gray_yellow_sand, stones_rock
Features for this trajectory are: [0.0, 0.0, 0.0, 1.0, 0.0]
Which trajectory is the best? Enter a number, [0-1], 0 for blue and 1 for pink:
1
```

Figure 4.7: Two trajectories visualized from the ROS node in Unity for querying the human expert

The experiments were performed in simulation in Unity Editor version 2020.3.36f1 and ROS Noetic on Ubuntu 20.04. The laptop used for the experiment has a 12th Gen Intel Core i9-12900HK $\times$ 20 CPU, 32GB of memory, and 1TB of disk space.

4.8 IOC model

The IOC model demonstrations were collected by moving the robot through the environment using a joystick controller and bagging the joystick teleoperation topics and terrain map topics. We collected 5 demonstrations D_1,...,D_5 each, for traversability and for concealment, in two rosbags file. The demonstrations for traversability and concealment were collected one after the other in two rosbag files, with five demonstrations for learning traversability in one bag file, and five demonstrations for learning concealment in the other. The demonstrations were collected by driving the robot around the environment using a controller, with the trajectory the robot drove on represented by the red line in Fig. 4.8. Each demonstration ran for about 30 seconds and the robot was driven around on traversable terrains, covering the different terrains during this time to have sufficient data representation for the learning. The trajectories collected in the demonstrations are on terrains that the robot is able to drive through successfully.

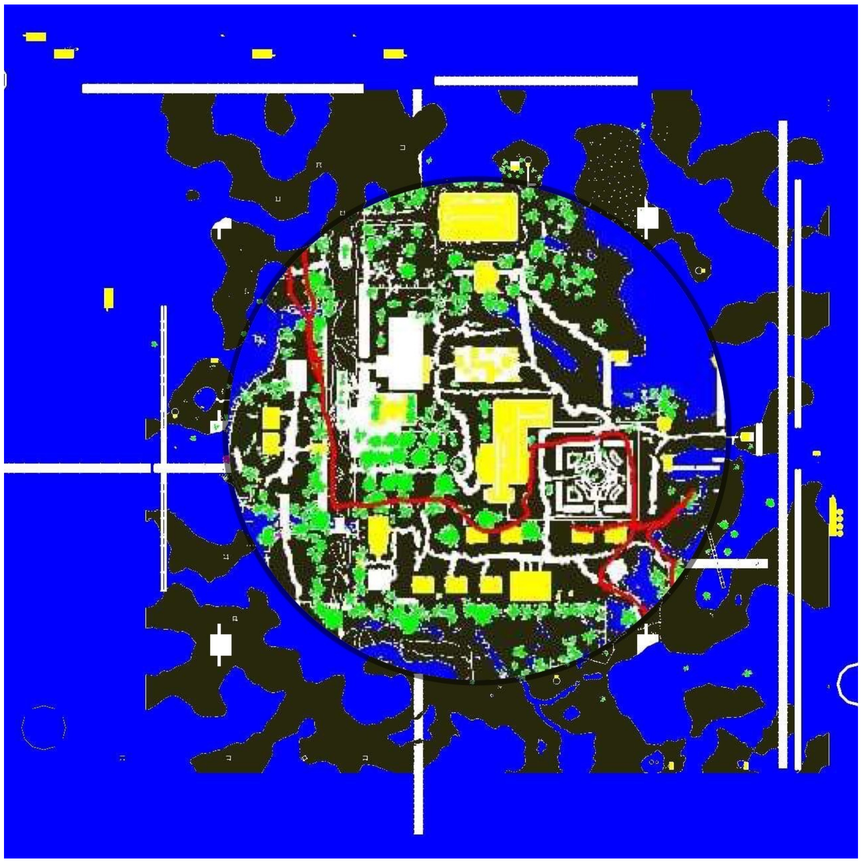


Figure 4.8: Demonstration trajectory driven by the robot for the IOC model training in red line markings

A rosbag is a file in which subscribed ROS topics data are stored in serializable messages format [5]. The rosbag is recorded by a bagging script which takes in the robot name and a configuration file of the topics to record. The terrain feature map topics (*trees_bush_map*, *water_map*, *rock_map*, *building_sidewalk_wall_fence_map*,

[5]http://wiki.ros.org/rosbag

gray_yellow_sand_map) are added to the config file for the rosbag to record. Other topics recorded in the rosbag file are *tf* and the joystick controller teleoperation topics, and the trajectory of the robot as an array of poses.

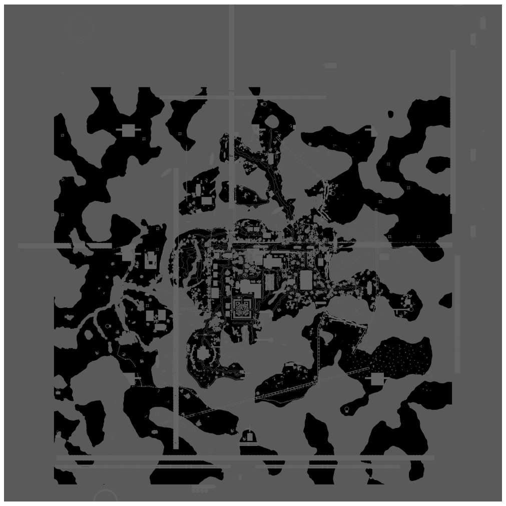
After the collection of the demonstrations in the bag file, the file is passed through a parser script. This parser script extracts images of the maps saved in the bag file and the demonstration trajectories.

The extracted trajectories from the demonstration bag files are then used to train the IOC model to learn the reward function for the five terrains. As in the PbIRL model, a separate IOC model is trained for traversability and for concealment.

4.9 PbIRL model

After the PbIRL model has been trained to learn the terrain weights, and these weights have been converted to occupancy grid values, for traversability and concealment individually, as seen in Fig. 4.9, these costmaps are passed as costmap layers into the global planner.

The default costmap layer passed to the global planner is the obstacle layer. This layer assigns costs based on the presence or absence of an obstacle in a position in the environment. Traversability and concealment are two additional costmap layers. The order of listing these costmap layers determines priority. For the experiment, the costmap layers were the traversability layer, concealment layer, and external constraint layer which defines regions the robot should stay out of. The traversability layer was placed over the concealment layer to give priority to the traversability layer, because if a cell is lethal in the traversability layer, that cost needs to be taken note of first.


(a) Traversability costmap generated from PbIRL model

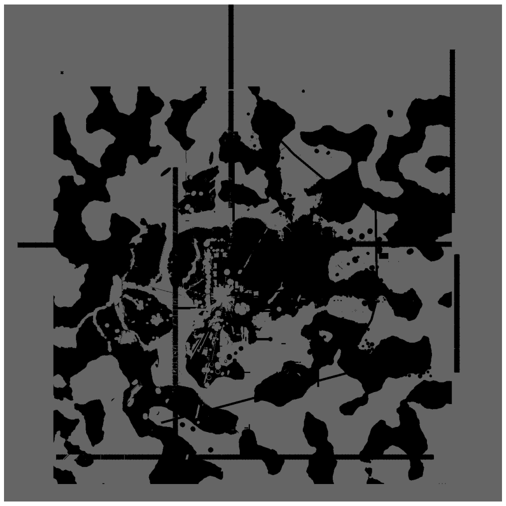

(b) Concealment costmap generated from PbIRL model

Figure 4.9: Traversability and concealment costmaps generated from the PbIRL model

4.10 PbIRL experiments

To test the preference-based reward learning model for learning traversability and concealment affordances of the terrains in the environment, the paths generated by the planner from the robot position to a goal position for different mission contexts were evaluated. Table 4.2 outlines the mission contexts tested for in the experiments. These are traversing fast, concealing fully and balancing traversability and concealment. These different mission contexts have different cosft factors for the weighted addition of the costmap values from the traversability and concealment costmap layers.

Table 4.2: Traversability and concealment cost factors in combining the costmaps for different mission contexts

Mission context	Traversability	Concealment
Traverse fast	1	0
Conceal	0	1
Balance both	0.5	0.5

4.11 IOC vs PbIRL experiments

To compare the performance of the PbIRL model and the IOC model, as the baseline, the paths generated by both models in tested in different simulated scenarios and compared against each other. For these scenarios, what the IOC model did was compared to what the PbIRL model did.

This experiment takes different anecdotes with different missions and compares the behavior of these models in the different scenarios presented. These comparison experiments are grouped into traversability, concealment, and balancing traversability and concealment.

4.11.1 Traversability

- Distance covered by the robot in trying to reach a goal position. Aim to minimize traversability distance.
- Rate of failure: Number of times the robot crashes into an obstacle or the number of times the global plan fails and the global planner needs to re-plan.

4.11.2 Concealment

- Proportion of the path in which the robot is being observed by the adversary. Aim to minimize exposure/unconcealed portion of the robot path.

4.11.3 Balancing traversability and concealment

- Proportion of the path in which the robot is being observed by the adversary. Aim to minimize exposure/unconcealed portion of the robot path.
- Distance covered by the robot in trying to reach a goal position. Aim to minimize traversability distance.

These are measured over three trials with the same start point for both models. The robot is started at the same position and the same end position is passed to the different models, then the trajectory generated by the global planner and the robot navigation are recorded and visualized. A program I created, implemented as a ROS node, tracks the total distance covered to get to a goal, collision/failure rate, and the concealment proportion for each path generated by the costmaps.

These metrics do not consider the robot physics attributes such as velocity and speed of the robot, as the scope of this thesis does not include local planning and local controllers, which are factors that could affect these attributes. The experiments are also run in simulation and computing capacity can affect these physics properties of the robot in experimentation.

5 Evaluation

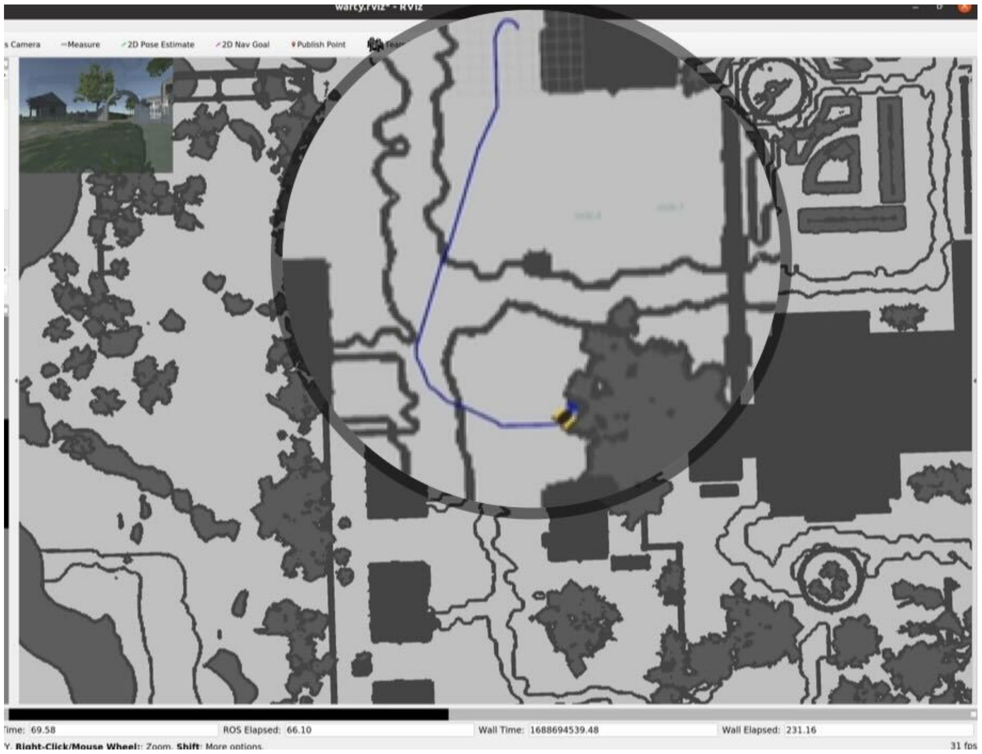
5.1 IOC vs PbIRL experiments

5.1.1 Traversability experiments

The traversability affordances learned by both models are evaluated based on the distance traveled on the path to get to the goal position and the number of crashes/failures that occur while following that path. These values are computed by sending a goal to the global planner, which uses the costmap (generated by either the PbIRL model or the IOC model) to generate a path from the robot location to this given goal position, as seen in Fig. 5.1, and logging the magnitude of the path. I perform multiple runs with the robot by executing the path in simulation from the same start position to the target location. Each goal to navigate to for each model was run three times independently and the mean of the three results was obtained to account for biases.

The robot pose in Table 5.1 is the (x,y) ROS coordinates of the robot platform in the environment. This is the starting point from which the robot begins navigation. Both models were started at the same position, each time, and passed the same goal positions. The goal position are the (x, y) distances in meters relative to the robot's starting position.

In the second goal experiment, the IOC model has a significantly shorter distance for the path generated by global planner. However, the local planner found it difficult to follow that path successfully and it caused some crashes.



Time: 69.58 ROS Elapsed: 66.10 Wall Time: 1688694539.48 Wall Elapsed: 231.16

Right-Click/Mouse Wheel:: Zoom. Shift: More options. 31 fps

Figure 5.1: Path robot navigated through from the start position to the goal position based on the PbIRL traversability costmap

Table 5.1: Measure of distance traveled from robot to goal position using costmaps from each of the models

Robot pose	Goal*(m)*	IOC1	IOC2	IOC3	Avg	PbIRL1	PbIRL2	PbIRL3	Avg
(370.6, 86)	(-30.06, -3.18)	43.5	43.1	44.3	43.6	44.5	44.4	44.5	44.5
(370.6, 86)	(-50, -20.18)	71.0	64.4	71.0	68.8	114.3	96.2	112.4	107.6

The PbIRL model experienced about an equal number of collisions/failure rates compared to the IOC model, on average. Although these collisions occurred at different points, figures 5.2 and 5.3 imply that these collisions were caused by the local planner deviating from the global path in generating the local path. In robot navigation, for optimal performance, global and local planners have to generate optimal paths. However, local planning is outside the scope of this work.

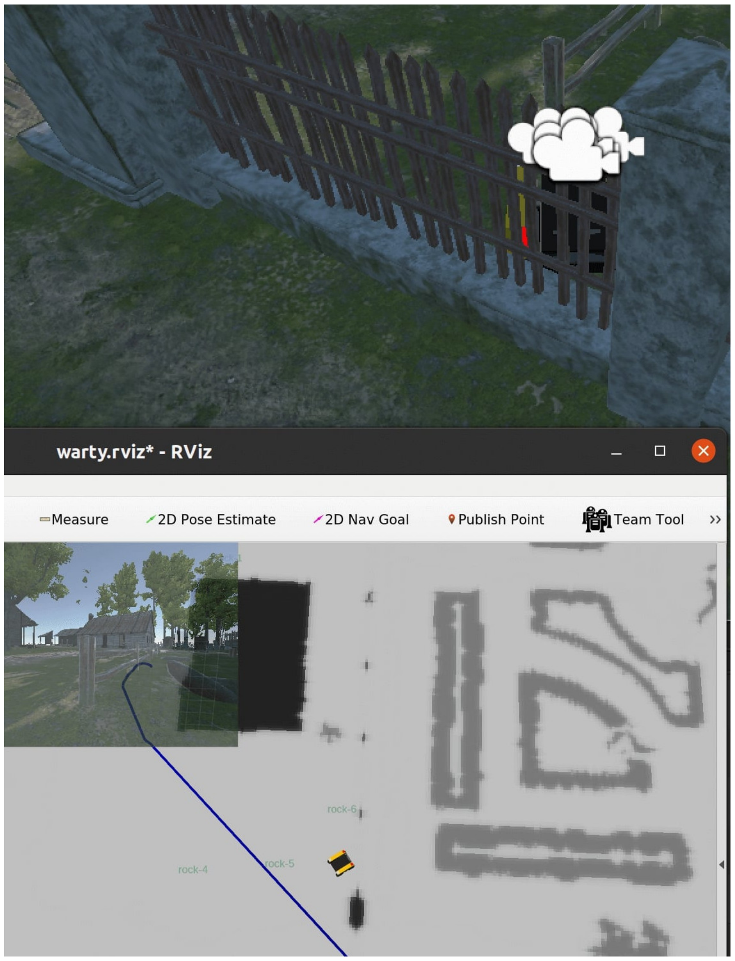


Figure 5.2: Collision of the robot in navigating based on the IOC traversability costmap

Table 5.2: Percentage of collisions out of three runs that occurred while navigating to a goal position based on the trained models

Robot pose	Goal(m)	IOC collisions	PbIRL collisions
(370.6, 86)	(-30.06, -3.18)	0%	0%
(370.6, 86)	(-50, -20.18)	66.7%	66.77%

5.1.2 Concealment experiments

Concealment is tested similarly to traversability by sending a goal to the planner to generate paths from the different learned models. However, concealment is evaluated

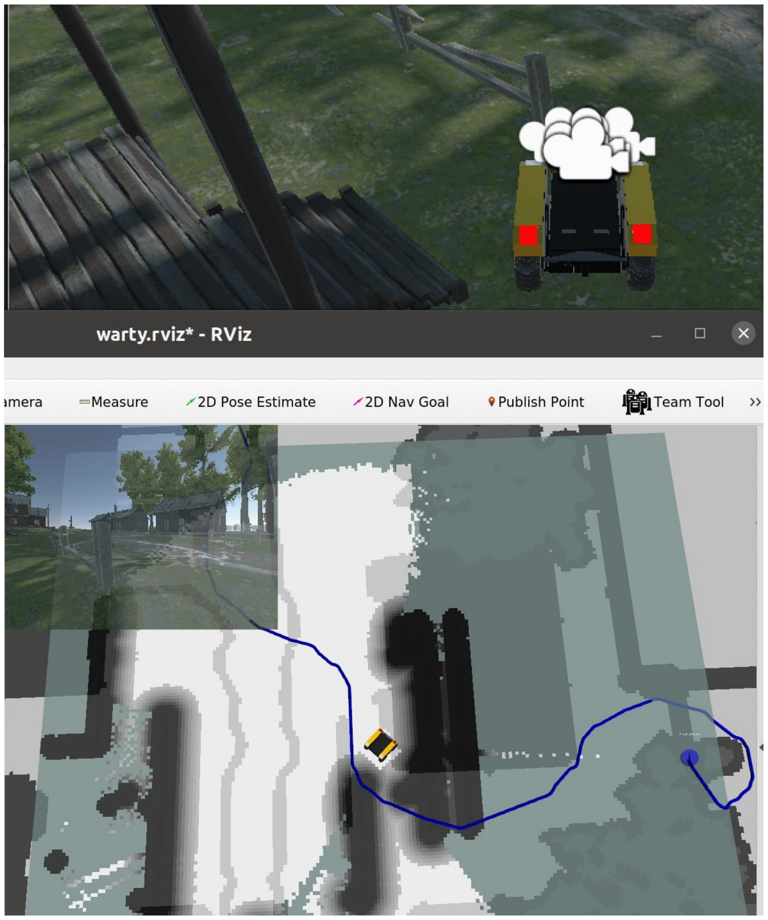


Figure 5.3: Collision of the robot in navigating based on the PbIRL traversability costmap

based on the unconcealed time in navigating to a goal position, so I wrote a C# script that takes in the adversary location as input, and the positions of each cell in the path generated from the robot to the goal position. Then, I run a ray cast function in the Unity editor that casts a ray from the adversary location to the positions on the path to count how much of the robot, while navigating via that path to the goal, was in the line of sight of the adversary.

To test the learned concealment weights, I generate paths from the robot to goal positions based on only the concealment costmaps generated by the IOC model and the

PbIRL model. Table 5.3 shows the rate of unconcealment from the start position to the goal position.

In this experiment, the adversary was placed 20m away from the robot, in direct line of sight, and we tested how the robot was able to navigate out of the low concealment zone and attain concealment while navigating to the goal.

Table 5.3: Percentage of unconcealed path distance in navigating to a goal position

Robot position	Adversary position	Goal	IOC unconcealed	PbIRL unconcealed
(370.6, 86)	(350, 85)	(-30.06, -3.18)	99.3%	49.97%
(370.6, 86)	(350, 85)	(-50, -20.18)	88.9%	51.7%

From Table 5.3, I noticed that the IOC model concealment costmap has a higher unconcealed rate. This highlights one of the advantages of the preference-based learning for multiple contexts over learning from demonstrations. Learning from demonstrations tries to learn and mimic ideal behvaior from the trajectories in the given demonstrations. However, since concealment is not learned directly, but based on the context of the adversary location, it is difficult to learn with demonstration methods, such as IOC in a way that generalizes.

5.1.3 Balancing traversability and concealment

The concealment costmap is not sufficient for successful autonomous navigation of the environment as a cell might have very high concealment from the adversary, which is good, but may be traversably lethal due to an obstacle. The aim of this thesis is to also learn robot navigation based on traversability and concealment of terrains.

To carry out this goal, I combined the traversability and concealment layers together, as a weighted sum, and passed the result to the global planner for path generation. The advantage of this approach is that the weights can be adjusted easily based

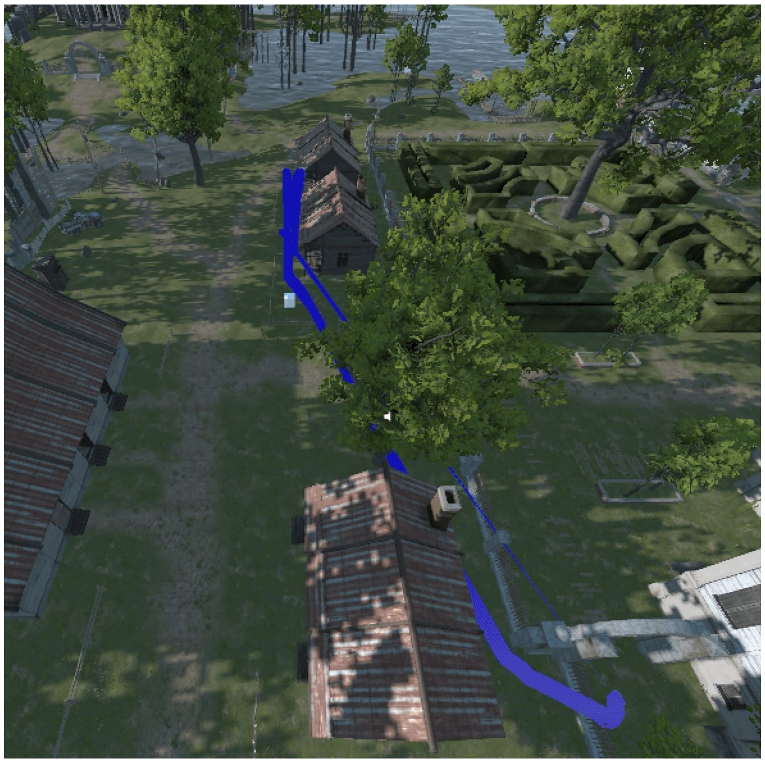


Figure 5.4: Path generated by the global planner to goal position based on the PbIRL concealment costmap. The generated path directly runs through right beside the adversary, denoted by light cube in the environment, in clear line of sight of the adversary

on mission context before running the ROS node to navigate to a goal and further research can be carried out in efficiently optimizing this combination.

For this experiment, I set the weight as 0.5 for the traversability layer and the concealment layer respectively.

Table 5.4 shows the results of balancing traversability and concealment for the PbIRL model and comparing the performance to the IOC model. I measure the average distance it took the robot to navigate to the goal based on each of these models' costmaps as the traversability property. I also measured the concealment of each of the models. For this experiment, the adversary stayed static and I placed the robot at different locations in the environment with different goal positions each time. Both models,

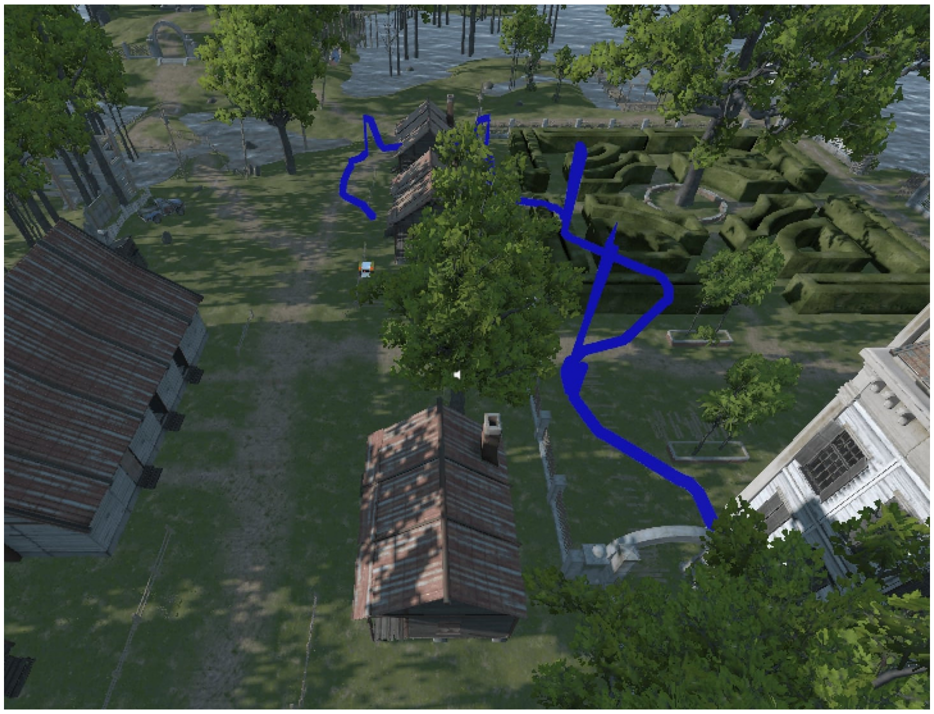


Figure 5.5: Path generated by the global planner to goal position based on the PbIRL concealment costmap

however, started at the same position and were given the same goal position for a fair comparison.

Table 5.4: Average traversability and concealment measure of the balanced IOC and PbIRL models

Robot position	Goal	IOC distance	Unconcealed	PbIRL distance	Unconcealed
(370.6, 86)	(-30.06, -3.18)	48.2	89.1%	171.4	33.8%
(371, -37)	(-15, -5)	20.8	61.5%	24.8	52.6%
(263, -79)	(10, 10)	18.9	0%	31.9	0%
(298, -6)	(90, 80)	165.4	0.7%	214.2	19.5%
(332, 145)	(0, -20)	22.5	0%	45.5	0%
(Mean performance)		55.16	30.26%	97.56	21.18%

These results show that the IOC on average generates significantly shorter paths compared to the PbIRL model. Conversely, the PbIRL model generates longer distance paths to the goal position, but with higher concealment.

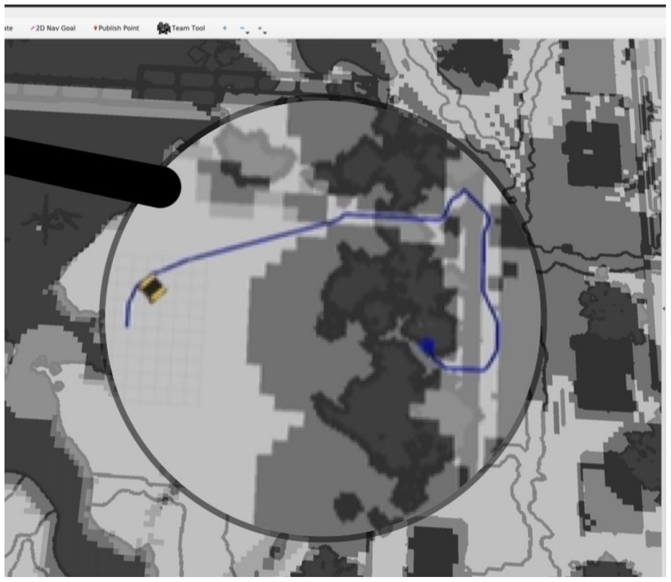
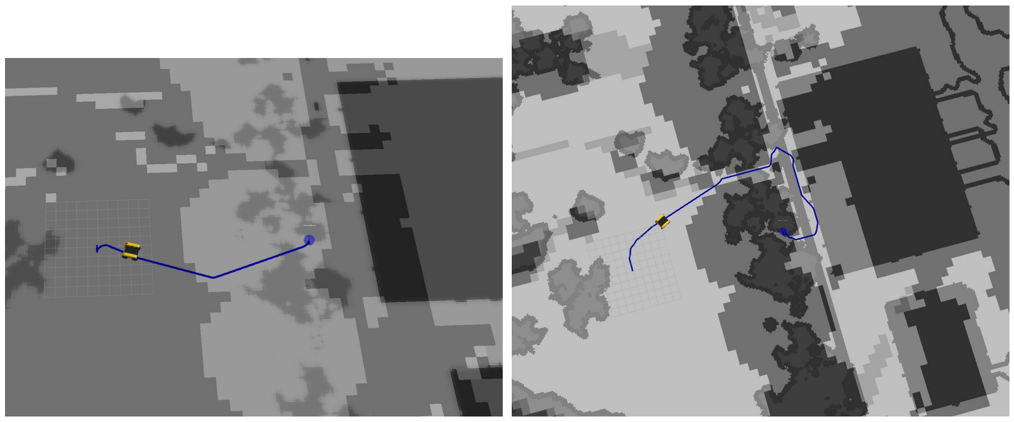


Figure 5.6: Combined PbIRL traversability and costmap layers with 0.5 weight assigned to each



(a) IOC combined costmap global plan (b) PbIRL combined costmap global plan

Figure 5.7: Paths generated from a start position to the same target location based on the IOC and PbIRL combined (traversability and concealment) costmaps

Fig. 5.7 shows the paths generated by the IOC combined costmap, see Fig. 5.7a, and the PbIRL combined costmap, Fig 5.7b. The IOC costmap generates a shorter and more straight forward path ($23m$) to the target location. The PbIRL costmap, on the other hand, generates a longer path ($46m$). I presume that the PbIRL model generates a longer path due to the fact that it learned concealment weights better than the IOC model, and accounts for the concealment of the cells for the path generation.

5.2 Summary

The preference-based learning model is able to learn traversability and concealment affordances of terrain types in the environment for robot navigation. While the IOC model learned the traversability affordances well, and was able to navigate to a goal position, with shorter paths, the concealment terrain weights were not totally representative of the expected behavior.

5.3 Limitations

From these experiments, one of the limitations of this work so far, is that the scope of this thesis only encompasses global planning. Evidently, robot navigation and path planning is a combination of the global planner + local planner + local controllers. Some of the experiment runs produced accurate global paths to the goal position, but the robot was unable to navigate through because the local planner path generated ran into obstacles slightly.

Another limitation was the exclusion of the inflation layer in the costmap layers passed into the global planner. Due to this, the global planner, therefore, did not take into account the C-space of the robot (i.e., the space of all the possible positions the robot might occupy in treating the robot as a point, irrespective of the shape and size) while planning the paths. This led to some paths being too close to an obstacle and the robot getting stuck midway, as seen in Figure 5.8.

Another limitation of the work is the number of terrains for which the model learned weights. Due to the huge size of the environment, number of terrains in the

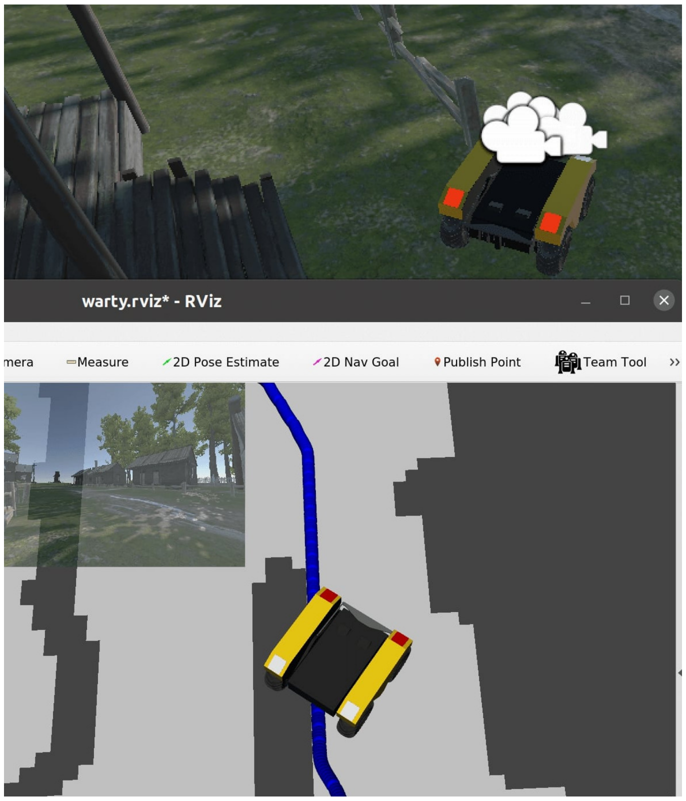


Figure 5.8: Path generated by the global planner to goal position too close to an obstacle causing the robot to get stuck

environment (twenty five unique terrains), data sparsity of some terrains in the environment, and limited computational capacity, I had to limit the terrains learned to under ten. However, since this model and method is generalizable, new terrains can be additionally learned by generating the trajectories for the training trajectory set in an area of the environment where this specific terrain type is available and well represented. The human effort required for initial training is another limitation. While the IOC model took 10 demonstrations of 30 seconds length each, the PbIRL model took 60 interactions to learn traversability and concealment. This, however, does not account for the cognitive load on the human expert training both models, which is something future work can explore further.

5.4 Possible future work

Subsequent to the findings and work done for this thesis research work, a possible extension could be introducing adversarial uncertainty. One of the major assumptions driving this research work currently is a prior knowledge of the exact location of the adversary in the environment.

There is also room for a user study to compare how a human perceives the behavior of this trained model. For different mission contexts, either prioritizing traversability or concealment, a human user study could be carried out to measure how well they think the PbIRL model carried out the mission successfully. This can be taken further to compare the PbIRL model vs the IOC model.

One other major advantages of this research work is the ability to combine multiple contexts in robot navigation. In this case, traversability and concealment. The experiments compare performance of the model for different mission contexts, prioritizing traversability over concealment, and prioritizing only concealment, among others. This mission context is defined prior to running the robot.

Finally, further research can explore dynamic adjustments of these mission contexts based on the environment, terrain, adversary location, or progress through the environment. More experiments can also be carried out to compare the performances of the models assigning different weights for traversability and concealment.

6 Conclusion

In this thesis research, I utilize a preference-based learning algorithm to learn traversability and concealment affordances of different terrains in an environment for autonomous robot navigation. I conduct an experimental evaluation in a feature-rich, large-scale, complex simulation environment, and compare the performance of this Pbrl model to an already deployed IOC model trained on human-collected demonstrations. In balancing traversability and concealment, the IOC model generated shorter traversable paths to the goal with a higher rate of unconcealment, while the PbIRL model generated longer traversable paths with higer concealment. For traversability, the Pbrl and IOC models performed comparatively with the IOC model generating shorter global paths to the goal, and for concealment, the Pbrl model produced better concealment paths to the goal with lower unconcealed portions of the global paths generated. The limitation of this work is the human effort required and learning performance for scaling to a large number of terrains. The results from the experiments show multi-context navigation capabilities of an autonomous robot based on terrains in the environment, adversary location, and mission context, by the Pbrl model. The need for no demonstrations to learn reduces the training efforts required, defines additional context that might not be accurately represented and explained in demonstrations, and increases the generalizability of the model to learn for new/added terrains and contexts.

Bibliography

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

[2] Z. Xie, Q. Zhang, Z. Jiang, and H. Liu, “Robot learning from demonstration for path planning: A review,” *Science China Technological Sciences*, vol. 63, no. 8, pp. 1325–1334, 2020.

[3] K. S. Sikand, S. Rabiee, A. Uccello, X. Xiao, G. Warnell, and J. Biswas, “Visual representation learning for preference-aware path planning,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 11 303–11 309.

[4] S. Schaal, “Is imitation learning the route to humanoid robots?” *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[5] P. Bakker, Y. Kuniyoshi, *et al.*, “Robot see, robot do: An overview of robot imitation,” in *AISB96 Workshop on Learning in Robots and Animals*, vol. 5, 1996.

[6] J. Hua, L. Zeng, G. Li, and Z. Ju, “Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning,” *Sensors*, vol. 21, no. 4, p. 1278, 2021.

[7] V. Gullapalli, J. A. Franklin, and H. Benbrahim, “Acquiring robot skills via reinforcement learning,” *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 13–24, 1994.

[8] K. Velagapudi, *Terrain cost learning from human preferences for robot path planning using a visual user interface*, [Unpublished Master's thesis], University of Denver, 2023.

[9] H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. Agia, and G. Nejat, "A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6569–6576, 2021.

[10] N. Islam, K. Haseeb, A. Almogren, I. U. Din, M. Guizani, and A. Altameem, "A framework for topological based map building: A solution to autonomous robot navigation in smart cities," *Future Generation Computer Systems*, vol. 111, pp. 644–653, 2020.

[11] E. Bıyık, A. Talati, and D. Sadigh, "Aprel: A library for active preference-based reward learning algorithms," in *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2022, pp. 613–617.

[12] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, *Active preference-based learning of reward functions*. 2017.

[13] F. Fahimi, *Autonomous robots*. Springer, 2009.

[14] H. Hexmoor, "Reactive navigation," in *Essential Principles for Autonomous Robotics*. Cham: Springer International Publishing, 2013, pp. 81–91, ISBN: 978-3-031-01563-2. DOI: 10.1007/978-3-031-01563-2_9. [Online]. Available: https://doi.org/10.1007/978-3-031-01563-2_9.

[15] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 2017, Special Issue on AI and Robotics, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.003.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001350.

[16] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "Autonomous robot navigation in highly populated pedestrian zones," *Journal of Field Robotics*, vol. 32, no. 4, pp. 565–589, 2015.

[17] M. Kollmitz, T. Koller, J. Boedecker, and W. Burgard, "Learning human-aware robot navigation from physical interaction via inverse reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 11 025–11 031.

[18] Y. Cui, H. Zhang, Y. Wang, and R. Xiong, "Learning world transition model for socially aware robot navigation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 9262–9268.

[19] T. Kruse, P. Basili, S. Glasauer, and A. Kirsch, "Legible robot navigation in the proximity of moving humans," in *2012 IEEE workshop on advanced robotics and its social impacts (ARSO)*, IEEE, 2012, pp. 83–88.

[20] T. Randhavane, A. Bera, E. Kubin, A. Wang, K. Gray, and D. Manocha, "Pedestrian dominance modeling for socially-aware robot navigation," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 5621–5628.

[21] H. Q. T. Ngo, V. N. Le, V. D. N. Thien, T. P. Nguyen, and H. Nguyen, "Develop the socially human-aware navigation system using dynamic window approach and optimize cost function for autonomous medical robot," *Advances in Mechanical Engineering*, vol. 12, no. 12, p. 1 687 814 020 979 430, 2020.

[22] H. J. Jeon, S. Milli, and A. Dragan, "Reward-rational (implicit) choice: A unifying formalism for reward learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4415–4426, 2020.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] A. Y. Ng, S. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, 2000, p. 2.

[25] B. Settles, *Active learning literature survey*, 2009.

[26] E. Biyik, N. Huynh, M. Kochenderfer, and D. Sadigh, "Active preference-based gaussian process regression for reward learning," in *Robotics: Science and Systems*, 2020.

[27] P. Wang, H. Liu, L. Wang, and R. X. Gao, "Deep learning-based human motion recognition for predictive context-aware human-robot collaboration," *CIRP annals*, vol. 67, no. 1, pp. 17–20, 2018.

[28] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *Aaai*, Chicago, IL, USA, vol. 8, 2008, pp. 1433–1438.

[29] H. Liu, Y. Wang, W. Ji, and L. Wang, "A context-aware safety system for human-robot collaboration," *Procedia Manufacturing*, vol. 17, pp. 238–245, 2018.

[30] I. A. Sucan and S. Chitta, *Moveit!* 2013.

[31] Y. Cho, J. Choi, J. Choi, and Y.-J. Ryoo, "Robot software platform for iot-based context-awareness," *International Journal of Humanoid Robotics*, vol. 14, no. 02, p. 1 750 012, 2017.

[32] W. Gao, D. Hsu, and W. S. Lee, "Context-hierarchy inverse reinforcement learning," *arXiv preprint arXiv:2202.12597*, 2022.

[33] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[34] C. Jung and D. H. Shim, "Incorporating multi-context into the traversability map for urban autonomous driving using deep inverse reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1662–1669, 2021.

[35] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning.," in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[36] M. Noseworthy, I. Brand, C. Moses, *et al.*, *Active learning of abstract plan feasibility*, 2021.

[37] M. Tucker, E. Novoseller, C. Kann, *et al.*, "Preference-based learning for exoskeleton gait optimization," in *2020 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2020, pp. 2351–2357.

[38] N. Wilde, D. Kulić, and S. L. Smith, "Active preference learning using maximum regret," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 952–10 959. DOI: 10.1109/IROS45743.2020.9341530.

[39] E. Biyik, M. Palan, N. C. Landolfi, D. P. Losey, and D. Sadigh, "Asking easy questions: A user-friendly approach to active reward learning," in *Conference on Robot Learning*, 2019.

[40] M. L. Puterman, "Chapter 8 markov decision processes," in *Stochastic Models*, ser. Handbooks in Operations Research and Management Science, vol. 2, Elsevier, 1990, pp. 331–434. DOI: https://doi.org/10.1016/S0927-0507(05)80172-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801720.

[41] M. Tucker, K. Li, Y. Yue, and A. D. Ames, "Polar: Preference optimization and learning algorithms for robotics," *arXiv preprint arXiv:2208.04404*, 2022.

[42] M. Tucker, M. Cheng, E. Novoseller, *et al.*, "Human preference-based learning for high-dimensional optimization of exoskeleton walking gaits," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 3423–3430.

[43] K. Li, M. Tucker, E. Bıyık, *et al.*, "Roial: Region of interest active learning for characterizing exoskeleton gait preference landscapes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 3212–3218.

[44] C. Wirth, R. Akrour, G. Neumann, J. Fürnkranz, *et al.*, "A survey of preference-based reinforcement learning methods," *Journal of Machine Learning Research*, vol. 18, no. 136, pp. 1–46, 2017.

[45] D. Bansal, Y. Hao, A. Hiranaka, R. Martin-Martin, C. Wang, and R. Zhang, *Dual representation for human-in-the-loop robot learning.*

[46] N. Ailon, "An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity.," *Journal of Machine Learning Research*, vol. 13, no. 1, 2012.

[47] M. Palan, G. Shevchuk, N. Charles Landolfi, and D. Sadigh, "Learning reward functions by integrating human demonstrations and preferences," in *Robotics: Science and Systems*, 2019.

[48] E. Biyik and D. Sadigh, "Batch active preference-based learning of reward functions," in *Conference on robot learning*, PMLR, 2018, pp. 519–528.

[49] E. Bıyık, K. Wang, N. Anari, and D. Sadigh, "Batch active learning using determinantal point processes," *arXiv preprint arXiv:1906.07975*, 2019.

[50] S. M. Katz, A.-C. Le Bihan, and M. J. Kochenderfer, "Learning an urban air mobility encounter model from expert preferences," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, IEEE, 2019, pp. 1–8.

[51] E. Bıyık, D. A. Lazar, D. Sadigh, and R. Pedarsani, "The green choice: Learning and influencing human decisions on shared roads," in *2019 IEEE 58th conference on decision and control (CDC)*, IEEE, 2019, pp. 347–354.

[52] R. Zhang, D. Bansal, Y. Hao, *et al.*, "A dual representation framework for robot learning with human guidance," in *Conference on Robot Learning*, PMLR, 2023, pp. 738–750.

[53] C. Galindo, J.-A. Fernández-Madrigal, J. González, and A. Saffiotti, "Robot task planning using semantic maps," *Robotics and autonomous systems*, vol. 56, no. 11, pp. 955–966, 2008.

[54] I. Kostavelis and A. Gasteratos, "Semantic mapping for mobile robotics tasks: A survey," *Robotics and Autonomous Systems*, vol. 66, pp. 86–103, 2015.

[55] R. Saini, P. Kumar, P. P. Roy, and U. Pal, "Trajectory classification using feature selection by genetic algorithm," in *Proceedings of 3rd International Conference on Computer Vision and Image Processing: CVIP 2018, Volume 2*, Springer, 2020, pp. 377–388.

[56] M. Wigness and J. G. Rogers, "Unsupervised semantic scene labeling for streaming data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4612–4621.

[57] A. Shaban, X. Meng, J. Lee, B. Boots, and D. Fox, "Semantic terrain classification for off-road autonomous driving," in *Conference on Robot Learning*, PMLR, 2022, pp. 619–629.

[58] J. Sock, J. Kim, J. Min, and K. Kwak, “Probabilistic traversability map generation using 3d-lidar and camera,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 5631–5637.

[59] B. Suger, B. Steder, and W. Burgard, “Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 3941–3946.

[60] F. Denis, R. Gilleron, and M. Tommasi, “Text classification from positive and unlabeled examples,” in *Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU’02*, 2002, pp. 1927–1934.

[61] C. Elkan and K. Noto, “Learning classifiers from only positive and unlabeled data,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 213–220.

[62] H. Lee and W. Chung, “A self-training approach-based traversability analysis for mobile robots in urban environments,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 3389–3394.

[63] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, “Learning ground traversability from simulations,” *IEEE Robotics and Automation letters*, vol. 3, no. 3, pp. 1695–1702, 2018.

[64] S. Levine and V. Koltun, “Continuous inverse optimal control with locally optimal examples,” in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 475–482.

[65] M. Wigness, J. G. Rogers, and L. E. Navarro-Serment, “Robot navigation from human demonstration: Learning control behaviors,” in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 1150–1157.

[66] M.-P. Dubuisson and A. K. Jain, “A modified hausdorff distance for object matching,” in *Proceedings of 12th international conference on pattern recognition*, IEEE, vol. 1, 1994, pp. 566–568.

[67] K. Dvijotham and E. Todorov, “Inverse optimal control with linearly-solvable mdps,” in *Proceedings of the 27th International conference on machine learning (ICML-10)*, 2010, pp. 335–342.

[68] M. Johnson, N. Aghasadeghi, and T. Bretl, “Inverse optimal control for deterministic continuous-time nonlinear systems,” in *52nd IEEE Conference on Decision and Control*, IEEE, 2013, pp. 2906–2913.

[69] A. Byravan, M. Monfort, B. Ziebart, B. Boots, and D. Fox, “Graph-based inverse optimal control for robot manipulation,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[70] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, Jun. 2016, pp. 49–58. [Online]. Available: `https://proceedings.mlr.press/v48/finn16.html`.

[71] B. D. Ziebart, *Modeling purposeful adaptive behavior with the principle of maximum causal entropy.* Carnegie Mellon University, 2010.

[72] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. arXiv: `1606.01540 [cs.LG]`.

[73] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *2010 IEEE international conference on robotics and automation*, IEEE, 2010, pp. 2902–2908.

[74] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, Feb. 2007.

[75] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 3354–3361.

[76] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, "3d object detection with pointformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7463–7472.

[77] S. Thrun, *Handbook for intelligent control: Neural, fuzzy and adaptive approaches, chapter the role of exploration in learning control*, 1992.

[78] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[79] J. D. Cohen, S. M. McClure, and A. J. Yu, "Should i stay or should i go? how the human brain manages the trade-off between exploitation and exploration," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 362, no. 1481, pp. 933–942, 2007.

[80] E. Bıyık, D. P. Losey, M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh, "Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences," *The International Journal of Robotics Research*, vol. 41, no. 1, pp. 45–67, 2022.

[81] S. Chib and E. Greenberg, “Understanding the metropolis-hastings algorithm,” *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995, ISSN: 00031305. [Online]. Available: `http://www.jstor.org/stable/2684568` (visited on 06/19/2023).

[82] R. A. Jacobs and J. K. Kruschke, “Bayesian learning theory applied to human cognition,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 2, no. 1, pp. 8–21, 2011.

[83] G. N. Yannakakis and J. Hallam, “Ranking vs. preference: A comparative study of self-reporting,” in *Affective Computing and Intelligent Interaction: 4th International Conference, ACII 2011, Memphis, TN, USA, October 9–12, 2011, Proceedings, Part I 4*, Springer, 2011, pp. 437–446.