Chapter 1

Introduction

Robots today are capable of a wide variety of tasks, which they achieve by two broad types of methods. The first set of methods involves directly programming the robot to execute a specific task. This form of explicit programming of the robot is suitable for situations in which repetitive tasks are to be performed in known environments with low amounts of uncertainty.

The second type of method involves the robot learning how to execute the task, or meta-programming. In this case, the robot does not have explicit instructions of exactly what to do, rather it is provided with a general framework that may adapt to different scenarios in its surrounding environment, or uncertainty in its world.

Incorporating such learning based approaches into robots is crucial for generalizing robots to much wider ranges of applications, or creating versatile robots that can complete a variety of tasks. Some fields and applications that would benefit greatly from general purpose robots include personal robots that assist humans in daily activities, industrial robots required to perform general purpose tasks, and robots that are required to act in collaboration with humans and other robots alike.

These scenarios call for a high level of understanding on the robot's part, and the ability to act without being explicitly instructed on what to do in a particular situation. Such generalization also allows the robot to handle unforeseen or new circumstances, and thus act more intelligently than previously capable.

Chapter 2

Literature Review

Recent advancements made in computer vision and machine learning literature have facilitated the development of semantic understanding techniques [9]. Robotics has benefited from the application such contributions to concepts including Visual SLAM [7],[topological and semantic SLAM], active navigation [10, 18], object recognition [13,14,15,16], and manipulation [3,6]. Semantic understanding has also been successfully applied to robots assisting humans in daily activites [5,15].

Of particular interest to us is recent work involving active searches for objects [1,2,3]. Anand et. al. [1] employed probabilistic heat maps of potential object locations, based on a graphical model of the scene constructed from visual and geometric features of co-occurring objects. While Koppula et. al. [5] exploited object affordances to anticipate human activities, the contextually guided search in [1] did not make use of affordances. It may also be noted that the robot navigation used was directly guided by the probabilistic heat map of a single object.

Aydemir et. al. [2] defines an active visual search for robots to locate objects in both known and unknown environments. While the authors' robot

is capable of exploring the scene, its knowledge is in the form of the environment map, and a position of the object, rather than higher level notions of affordances and associated actions. The authors also provide a comparison of the robots performance against a human in a similar setting, although they do not leverage the implicit understanding gained by the human to guide the robot.

In [3], Wong et. al. integrate robotic manipulation with active perception, by manipulating objects in the scene to achieve a better view of the surrounding scene. The authors' work focuses on target objects that are potentially occluded inside 'containers' of given sizes. By manipulating extraneous objects in the vicinity, they demonstrate a planner on a simulated PR2 robot, that explores containers to locate the target object.

A common trend in recent advancements involves enabling robots to implicitly understand the environments they perceive, and go about tasks armed with this understanding, in much the same way humans do. We note that a largely unused resource in this regard comes in the form of human demonstration and understanding. An intuitive method to infuse robots with this implicit understanding of the scene is to provide the robots with access to a demonstration by a human executing a task in a similar environment.

Learning from demonstration (LfD) techniques have explored the concept of a robot imitating motion from a human expert. While LfD techniques have been successfully applied to a variety of robotic tasks, such as aerial manoeuvres for UAVs [R1], manipulation of typical objects [R2,R6,R8], such approaches are based on kinematic or geometric features inherent to the robot or its surrounding environment. Work tying learning from demonstration to semantic understanding is relatively nascent [10]. We propose a system which incorporates expert demonstrations to help a robot learn the implicit rewards

associated with exploring a scene. More details are provided in section 3.

Recent work in LfD [R1,R2,R3] is closely tied to inverse reinforcement learning [R4]. Such approaches utilize the demonstrations provided by an expert user to estimate an unknown reward function. The majority of these techniques do so by maximizing the margin between the expected reward obtained by expert policy and the current robot policy. These approaches thus attempt to bound the sub-optimality of a robot policy, by iteratively updating the reward function till convergence.

In this paper, we propose a robotic system that understands object-object and object-concept relationships in the scene.

[The figure will be displayed here.]

Figure 2.1: The correlation coefficient as a function of ρ

Chapter 3

Semantic Relationships

3.1 Semantic Relationships

For constructing a semantic understanding of a scene, we consider spatial relationships between objects, as well as object-affordance relationships. Spatial relationships exist between pairs of objects present in the scene. We analyse the relative spatial positions of objects that occur together in a scene. If these objects occur with proximity to each other in a particular scene, we claim they are spatially related, and tag this scene with a spatial relationship of these objects. We also make use of object-affordance relationships. An affordance typically refer to a quality of an object that enables it to perform a certain action, or to be used for performing a certain action. Throughout this paper, we disregard the temporal domain of actions, and hence utilize the term *concepts* to refer to these actions. Thus we first understand the knowledge of what concepts the object in question may be used for. By identifying objects with common affordances, we are then able to understand what additional objects may be required to execute a given concept, and thus enable the robot to search for such objects.

3.2 Notation

Throughout this document, we make use of the following notation. We consider m number of objects, and n number of potential affordances, and s number of scenes for training our system.

x_i refers to object i, varying from $1 : m$

y_j refers to affordance j, varying from $1 : n$

k refers to the index over scenes used for training

$p(x_i)$ refers to the probability of object i occuring in the scene

$p(y_j)$ refers to the probability of affordance j occuring in the scene

3.3 Spatial object relationships

The spatial object relationships may be quantified by the distance between a particular pair of objects. Other qualitative relationships include relative location, such as front, back, etc. However, such features are not robust to changes in camera view point. Traditionally, relative distances between objects is not scale invariant, and depends on the distance of the camera from the objects. In may be noted, however, that we make use of the Kinect as a 3D depth sensor, thus providing us with access to absolute pose information of the objects. This permits us to use the relative distance, or radius feature between two objects. The feature is clearly robust to changes in camera position as well as scale changes. The problem of spatial object relationships is thus split into two sub-components that of learning and querying. Thus we first learn the spatial relationships of all possible object pairs over each of the s training scenes, and subsequently generate a corresponding predictive model.

3.3.1 Learning spatial relationships

In order to learn the spatial relationships, we first parse the scenes present. Thus for each scene k, over each pair of objects i and j, we have:

$$r_{ij}^{k} = \|\mathbf{x_i}^{k} - \mathbf{x_j}^{k}\|_2 \tag{3.1}$$

And we generate the predictive model for spatial relationships as r_{ij}^{*} :

$$r_{ij} = \arg\min_{r^*} \sum_{k=1}^{s} \|r^* - r_{ij}^{k}\|_2 \tag{3.2}$$

We may exploit the single dimensional nature of the radius feature, to reduce the $\arg\min_{r^*}$ problem to a simple computation of the mean of all such r_{ij}^{k}:

$$r_{ij}^{*} = \frac{1}{s} \sum_{k=1}^{s} r_{ij}^{k} \tag{3.3}$$

We also compute the standard deviation about the mean for each of the object pairs.

$$\sigma_{ij}^{*} = \sqrt{\frac{1}{s} \sum_{k=1}^{s} (r_{ij}^{k} - r_{ij}^{*})^2} \tag{3.4}$$

3.3.2 Querying spatial relationships

Given an estimate of the 3D pose of all objects detected in the scene, we then proceed to calculate the potential location of a particular object.

To compute this efficiently, we make use of a sampling based approach. We discretize the current field of view of the Kinect into a fixed number of 2D grid points. This discretization helps reduce the number of points we must consider to the order of the grid size, instead of the original input point-cloud.

In order to account for the uncertainty in position of the object we are looking to find, we use a value function based approach in the form of a truncated Gaussian distribution. Thus for a given pair of objects, we define a value function based on the distance between these objects, and make the assertion that a higher function value at a certain radius corresponds to a higher likelihood of the object occurring at that particular radius away from the alternate object. Such a distribution may be defined as:

$$G(r, \mu_{ij}, \sigma_{ij}, a, b) = \frac{\frac{1}{\sigma_{ij}}\phi(\frac{r-\mu_{ij}}{\sigma_{ij}})}{\Phi(\frac{b-\mu_{ij}}{\sigma_{ij}}) - \Phi(\frac{a-\mu_{ij}}{\sigma_{ij}})}$$

where $\phi(r)$ is the probability density function, and $\Phi(r)$ represents the cummulative distribution function. Here, the limits a and b are the points of truncation for our distribution. We have $a = 0$ and $b \to \infty$, thus under these conditions, we have $\Phi(\frac{b-\mu}{\sigma}) \to 1$.

It may also be noted that since the radius is independent of the relative direction in which the objects occur, using one pair of objects is not sufficient to estimate a unique location where the object is likely to occur.

The framework we use thus considers multiple pairs of objects, to assign a value to a particular sampled point. This value corresponds to the likelihood of the desired object occurring at that sampled point. For completeness, we consider *all* such object pairs, noting that the contribution of objects that are very far away is diminished by the Gaussian trait of the value function. The value of a given sample point, to find object i is then computed as:

$$V(\mathbf{r}_{sample}) = \sum_{j=1}^{m} G((\mathbf{r}_{sample} - \mathbf{r_j}), \mu_{ij}, \sigma_{ij}) p(\mathbf{x_i}) \tag{3.5}$$

Where $p(\mathbf{x_i})$ is the belief that object i occurs at location $\mathbf{x_i}$, provided to

us as a confidence estimate by our detector. The sample point chosen to be our best estimate of the current position of the desired object may thus be defined as:

$$\mathbf{z}^*_{obj} = \arg\max_{\mathbf{r} \in S} V(\mathbf{r}_{sample}) \tag{3.6}$$

Where the arg max is taken over all such points belonging to the discretized sample space.

3.4 Object Affordance relationships

The second type of object property we are interested in understanding is that of object affordances. An affordance is typically considered as a relationship between the object and its surrounding environment, that permits the object to perform an action, or to be used to perform an action. We consider common day to day executable concepts such as eating, writing, reading, as potential affordances that an object may have.

Human beings have an implicit understanding of what concepts may occur in a scene based on the objects present. To enable a robot the same capabilities, we leverage information of scenes labelled with concepts that potentially occur in the scene. We then note which objects occur in this scene, and then draw up a correlation between these objects and concepts for each scene. We split the object affordance relationships problem into two components, learning and querying.

3.4.1 Learning object affordance relationships

In the learning phase, we would like to first estimate the probability of a concept occurring in the scene given the confidence of our object detector in detecting some subset of objects present in the scene. In order to do this,

we must create a transformation that takes us from the object space to the concept space, i.e., it operates on a vector of object detection confidences, and outputs a vector of confidences of each of the concepts. We consider a set of scenes $k = 1 : s$, each labelled with:

x_i^k - Whether object i occurs in scene k, over all i and k.

y_j^k - Whether concept j occurs in scene k, over all j and k.

We define a set of coefficients w_{ji}^*, which correspond to whether object i is required to execute concept j. For convenience, we also define an m x n matrix $M = [w_{ji}^*]$. We first compute a series of such coefficients w_{ji}^k and a corresponding matrix M_k for each scene k in the training dataset. We wish to learn the predictive affordance model, w_{ji}^*, from w_{ji}^k.

w_{ji}^k thus takes on a non zero value when object i is required for concept j, and is 0 otherwise. Here, w_{ji}^* and w_{ji}^k alike are both normalized for each j, i.e.

$$\sum_{i=1}^{m} w_{ji}^k = \sum_{i=1}^{m} w_{ji}^* = 1 \tag{3.7}$$

Note here that while w_{ji}^k may appear analogous to $p(y_j^k|x_i^k)$, w_{ji}^k is to be treated as a transform from the object space to the affordance space rather than a series of conditional probability expressions. Thus for each scene k, we have:

$$\overline{y_j}^k = \sum_{i=1}^{m} w_{ji}^k x_i^k \tag{3.8}$$

and

$$y_j^k = \frac{\overline{y_j}^k}{\sum_{j=1}^{n} \overline{y_j}^k} \tag{3.9}$$

In order to learn w_{ji}^*, we find an optimal set of weights w_{ji} such that our predictive model of concepts occurring in the scene, $\sum_{i=1}^{m} w_{ji}^* x_i^k$ is as close

to the labelled set of concepts y_j^k as possible. However, for a given scene k, it is impossible to learn the $n \times m$ elements from just the single constraint $y_j^k = \sum_{i=1}^{m} w_{ji}^k x_i^k$. To simplify the problem, we consider one concept at a time. By trying to solve for a consistent w_{ji}^k for a fixed concept j over all scenes k, the problem may be rearranged to find an efficient solution method. We first define $\mathbf{w_j}^*$, $\mathbf{x}^k$, $\mathbf{y_j}$ and X as -

$$\mathbf{w_j}^* = [w_{j1}^*, w_{j2}^*, \dots w_{ji}^*, \dots w_{jm}^*]$$

$$\mathbf{x}^k = [x_1^k, x_2^k, \dots x_i^k, \dots x_m^k]$$

$$\mathbf{y_j} = [y_j^k, y_j^2, \dots y_j^k, \dots y_j^s]$$

$$X = [x_i^k]_{s \times m}$$

We thus have an overdetermined system of linear equations, under the assumption our dataset is sufficiently large. For each such concept j,

$$\mathbf{y_j} = \mathbf{w_j}^{*T}.\mathbf{x}^k \tag{3.10}$$

Thus the desired w_{ji}^*, or $\mathbf{w_j}^*$ for all such concepts j, may be defined by a linear least squares problem-

$$\mathbf{w_j}^* = \arg\min_{\mathbf{w_j}^*} \sum_{k=1}^{s} \|y_j^k - \mathbf{w_j}^{*T}.\mathbf{x}^k\|_2 \tag{3.11}$$

This may be solved using the Moore Penrose Pseudoinverse.

$$(X^T X).\mathbf{w_j}^* = X^T.\mathbf{y_j} \tag{3.12}$$

$$\mathbf{w_j}^* = (X^T X)^{-1} X^T.\mathbf{y_j} \tag{3.13}$$

$$\mathbf{w_j}^* = X^+ y_j \tag{3.14}$$

This gives us an efficient method of learning the w^*_{ji} values from a dataset of labelled scenes. For convenience, we define an Object-Affordance matrix M, equivalent to the transformation provided by w^*_{ji}.

$$M = [w^*_{ji}]_{n \times m}$$

3.4.2 Querying object affordance relationships

Once we have learnt the object affordance relationships from the labelled dataset, we must create a system to determine what objects and concepts occur in a new scene. We derive from the literature to determine what objects occur in the scene, making use of a 3D semantic scene labelling provided by Kopulla et. al. Such an object detector provides us with a confidence of a particular object occurring in a scene, $p(x_i)$, for each such object i. Our first task is thus to retrieve the probability with which a certain activity occurs in the scene, $p(y_j)$, for all concepts j. Thus given $\mathbf{x}$, we would like to estimate $\mathbf{y}$, where:

$$\mathbf{x} = [p(x_1), p(x_2), \ldots p(x_m)]^T$$

$$\mathbf{y} = [p(y_1), p(y_2), \ldots p(y_n)]^T$$

This may be easily computed using the Object-Affordance matrix M.

$$[p(y_j)]_{n \times 1} = M_{n \times m}[p(x_i)]_{m \times 1} = [w^*_{ji}]_{n \times m}[p(x_i)]_{m \times 1} \tag{3.15}$$

Which may be simply written as

$$\mathbf{y} = M\mathbf{x} \tag{3.16}$$

Equation 16 is notably powerful - it simplifies a portion of our semantic model into one cohesive equation. Obtaining the probabilities of each activity occurring, defines half of the semantic understanding - we now understand what concepts are likely occurring in the scene, and with what likelihood they are present. The second phase of our understanding requires us to determine which objects are required to execute the most likely concepts in the scene, introducing the notion of correlated objects.

A simple example is if the scene contains a plate, a fork, and a knife, the robot identifies one potential activity that could occur is eating. The robot now ideally understands, from what it has learnt, that a spoon is also required to perform the activity.

Traditional approaches typically follow a Maximum-a-posteriori estimation for such a problem. This involves choosing a single most likely action that is occurring in the scene, disregarding other possibilities that may be only slightly less likely.

In order to develop a more robust approach, we consider that multiple concepts may be occurring in the scene, and proceed with identifying objects required to execute this *set* of concepts, rather than a single activity. In order to retrieve these objects that are required, we must return from the activity space y to the object space x. Specifically, given the vector of probabilities of concepts $\mathbf{y}$, we need a transformation that provides us with a set of objects that is required for these concepts, $\mathbf{x}$. We observe that the terms of the object affordance matrix M, i.e. w^*_{ji} express the correlation of activity j with object i. Utilizing the commutativity of such correlation, the transpose of this matrix, M^T, is observed to be the desired transform from the action space to the object space. Thus our set of correlated objects, $\mathbf{x}^*$, may be computed as

$$\mathbf{x}^* = M^T\mathbf{y} \tag{3.17}$$

Equation 17, the 'inverse' counterpart of equation 16, is equally powerful. It permits us to calculate the set of objects required to perform a set of concepts which are likely occurring in the scene. We note with emphasis that we may already have a high confidence of detection of a subset of these objects, represented in the original object vector $\mathbf{x}$. In a meaningful setting, we would like a robot to smartly overlook objects in which we have a high confidence of detection. An assistive robot would thus search only for those objects which are not already present, or obvious. To identify such objects, we subtract the initial object confidences, $\mathbf{x}$ from our set of correlated objects, $\mathbf{x}^*$, and assign these "neediness" values to z.

$$\mathbf{x} = \mathbf{x}^* - \mathbf{x} \tag{3.18}$$

$$\mathbf{z} = \mathbf{x}^* - \mathbf{x} \tag{3.19}$$

This may be equivalently expressed as

$$\mathbf{z} = M^T\mathbf{y} - \mathbf{x} \tag{3.20}$$

$$\mathbf{z} = M^T M\mathbf{x} - \mathbf{x} \tag{3.21}$$

$$\mathbf{z} = \{M^T M - I\}\mathbf{x} \tag{3.22}$$

In order to determine an order in which we ought to search for these objects, we simply rank the object in decreasing order of values of their

"neediness" for all $q = 1 : z$:

$$\mathbf{z}^* = rank_q z_q \tag{3.23}$$

Chapter 4

Decision Making

4.1 Decision Making

In order for the robot to autonomously explore the environment without explicit human commands, it is critical for the robot to be able to navigate its surroundings on its own. There are two approaches that may be taken towards autonomous navigation of the robot. The first is to explicitly provide the robot with a goal location, and use classical planners from path planning literature to drive the robot to that location. The second approach, while more complicated in formulation, makes use of stochastic decision making techniques to find optimal motions, or actions, for the robot.

Stochastic approaches to planning or decision making are established to be more robust to uncertainty in the environment. Rather than providing a robot with a sequence of actions, or controls, they provide the robot with a policy. A policy is a description of what action to take in every possible state the robot may be in. Thus to achieve its objective of reaching a particular goal, the robot would take the action specified by the policy, estimate what

state it is in, and repeat the procedure.

By providing the robot with an idea of what action to execute in every possible scenario, rather than a deterministic set of actions to follow, the planning exercise is made considerably more robust. Such approaches are then only restricted by the uncertainty of the state estimation itself. Here, some well established frameworks for decision making are presented for subsequent use.

4.2 Markov Decision Processes

A typical formulation in formal decision making theory is to utilize a Markov Decision Process (MDP). An MDP consists of:

1. State $s \in$ State space S - A description of the state the system is currently in.

2. Action $a \in$ Action space A - The list of potential actions the system may take in its current state.

3. State Transition Matrix $T(s, a, s')$ - The probability of the state transitioning from s to s' upon taking action a.

4. Reward Function $R(s, a, s')$ - A reward value associated with the state s upon taking action a.

The fundamental premise of an MDP is that actions taken by an agent or a robot are stochastic - there is some probability with which action a will cause a transition from state s to s'.

A policy $\pi : S \rightarrow A$ may be defined as a mapping from state space S to action space A. Thus $a = \pi(s)$ represents the action to be taken at state

s under policy π. The MDP also considers a time horizon T over which an expected value of reward may be computed as:

$$E[V^{\pi}] = \sum_{t=1}^{T} \gamma^{t} R(s, \pi(s)) \tag{4.1}$$

where t is the index over time, γ is a discount factor reduce the value of decisions far into the future.

Thus we say a policy π^* is optimal if there is no gain in expected reward by defecting from the action specified by π^*, for every possible state:

$$R(s, \pi^*(s), s') \geq R(s, a, s') \quad \forall a \in A \quad \forall s \in S \tag{4.2}$$

We assume the access to an MDP solver, such as SARSOP.

4.3 Learning from demonstration

Learning from demonstration (LfD), or Imitation Learning, is a concept that makes use of demonstrations of a particular task to understand how the task is performed. When employed on a robot, it then allows the robot to 'imitate' the demonstration, thus accomplishing complex tasks without explicit programming. Traditionally, LfD uses access to a demonstration by an expert user, as an "expert policy" π^E, and attempts to create a policy π^* that performs as well as the expert policy.

4.4 Inverse Reinforcement Learning

While the MDP decision making framework is suitable for an explicitly defined reward function, it may occur that the Reward function is unknown, or too complex to define. Similarly, the transition matrix may also be unknown, in cases where a model of the robot is difficult to create. In such cases, it is desirable that a robot learns this reward function on its own. Inverse Reinforcement Learning (IRL) provides us a well established framework for this purpose. LfD may be coupled with IRL to estimate the reward function, by assuming that the expert user is maximizing the expected reward in taking actions. The LfD and IRL approaches are formally defined in section 5.1.

4.5 Scene exploration

For semantic context based robot exploration, we define the problem as follows. The robot is provided with a map of the environment, as well as an estimate of the spatial likelihood of occurrences of various objects. The robot uses the semantic knowledge explained in sections 2-4, to understand the priority of searching for these objects, based on the "action / concept" it believes is occurring in the scene. Its task is then to navigate this environment in an optimal manner, such that it maximizes the probability of finding these objects. It may be noted that the robot would ideally follow the priority order of locating the objects, and also avoid static obstacles present while navigating its environment.

Formulation

In the context of robot exploration, we assume an environment to be a discrete rectangular grid world, consisting of of $length_{env} \times breadth_{env}$ number of

1×1 grid 'cells'. An ideal choice of state s is the robot location $[x_{robot}, y_{robot}]^T$ in its environment. Thus x_{robot} and y_{robot} correspond to the indices of the robot in the grid world. For simplicity, we assume the robot may fit inside one grid cell. Note here that the state space S is thus the entire grid world.

We consider a ground robot exploring this environment, which can move forward, backward, left or right. Our action space A then becomes $\{forward, backward, left, righ$ It follows all robot actions a must then belong to A.

We would like to estimate a reward function $R(s, a, s')$, given N_d number of demonstrations by a user, of navigating the robot in its environment. These demonstrations may be represented as an expert policy π^E.

We recall the previously calculated set of value functions $V_i(s)$ from equation #, which correspond to the likelihood of object i occurring at location s. We additionally consider a set of costmaps $C_h(s) \quad \forall h \in \{1, 2, ...H\}$. These costmaps are retrieved from the 3D map of the environment created in section #, and have a progressively higher value around obstacles, and have lower values in open spaces. This costmap provides us information of where the robot may physically move in the real world, without colliding with obstacles.

We assume our user is trying to maximize a reward function with respect to these value functions as well as the costmaps. value functions and costmaps. We thus label these functions $V_i(s)$ and $C_h(s)$ as s set of basis functions $\boldsymbol{\phi}(s)$. This gives us a number of basis functions $N_b = N_{basis} = N_{value\ functions} + N_{costmaps}$. A logical step is to approximate the reward func-

tion as a linear combination of these basis functions.

$$R(s, a, s') = \sum_{k=1}^{N_b} \omega_k \phi_k(s) \tag{4.3}$$

This may be concisely represented as:

$$R(s) = \boldsymbol{\omega}^T \boldsymbol{\phi}(s) \tag{4.4}$$

Where $\boldsymbol{\omega}$ is defined as:

$$\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_{N_b}]^T$$

and $\boldsymbol{\phi}(s)$ is the collection of basis functions:

$$\boldsymbol{\phi}(s) = [V_1(s), V_2(s), \ldots, V_{N_{value\ functions}}, C_1(s), C_2(s), \ldots, C_{N_{costmaps}}]^T$$

Our task then becomes to estimate the weights $\boldsymbol{\omega}$, and hence the reward function, given the expert policy. Existing approaches such as "Apprenticeship Learning via Inverse Reinforcement Learning", Abbeel and Ng, 2004 iteratively consider policies π_k, and maximize the difference of expected reward between the expert policy π^E and the considered policy π_k.

Such approaches utilize the demonstrations provided by an expert user to estimate an unknown reward function. The majority of these techniques do so by maximizing the margin between the expected reward obtained by expert policy and the current robot policy. These approaches thus attempt to bound the sub-optimality of a robot policy, by iteratively updating the reward function till convergence.

It may be noted that this method is limited in the following capacity. By iteratively solving an MDP with an updated estimate of the rewards, the authors do not provide a guarantee on whether the output policy provides a greater expected reward than the existing policies previously considered. For an initial sub-optimal policy, margin maximization between the expert and the current policy provides no new information, and solving the MDP with the resultant rewards is unnecessary. Furthermore, in order to evaluate the expected reward for the expert policy, access to the entire expert policy is assumed. In an actual setting of apprenticeship learning, we note that such access is not guaranteed; rather, the user provides an expert policy for a subset of the states present.

In order to overcome these limitations, we present a method that leverages the fact that an expert user would try to maximize the expected reward at every step. We further postulate a tighter bound on such a reward; the expert π^E maximizes the increase in reward at every time step. We also note that our approach is more intuitive to the reader, and also runs with significantly reduced time complexity, which we demonstrate in section 23232.

The reader may note that a single demonstration consists of a series of ordered pairs $<s, a, s'>_q^t$. Here, t is an index over time, till the horizon T_q of demonstration q. A demonstration may alternately be considered as a train of these ordered pairs indexed over time. The algorithm we propose is as follows:

Our objective then becomes to solve:

$$\arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{T_q} \gamma^t \left[R({}_q^t s') - R({}_q^t s)\right] \tag{4.5}$$

Algorithm 1 LfD-IRL for SAREx

procedure LEARN REWARD WEIGHTS
    Initialize $\boldsymbol{\omega}^*$
    for $q \in [1, 2, \ldots, N_d]$ **do**:
        $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}^*$
        $\boldsymbol{\omega}^* \overset{\alpha}{\leftarrow} \arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{T_q} \gamma^t \left(R({}^t_q s') - R({}^t_q s)\right)$
        $\alpha \leftarrow 0.9\alpha$
    return $\boldsymbol{\omega}^*$

We also recall $R(s) = \boldsymbol{\omega}^T.\boldsymbol{\phi}(s)$, this may be represented as:

$$\arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{T_q} \gamma^t \left[\boldsymbol{\omega}^T.\boldsymbol{\phi}({}^t_q s') - \boldsymbol{\omega}^T.\boldsymbol{\phi}({}^t_q s)\right] \tag{4.6}$$

Noticing that $\boldsymbol{\omega}$ is independent of the time index, we have:

$$\arg\max_{\boldsymbol{\omega}} \ \boldsymbol{\omega}^T.\sum_{t=1}^{T_q} \gamma^t \left[\boldsymbol{\phi}({}^t_q s') - \boldsymbol{\phi}({}^t_q s)\right] \tag{4.7}$$

In order for the problem to lead to a consistent solution that may subsequently used by the robot to navigate its environment, we note the following trivial corner case. If the basis functions are defined to be non-negative over the state-space domain, the specified arg max problem would simply individually maximize each component of the weights, ω_j.

To avoid this pitfall, we map each of the basis functions from their respective ranges, to a common range of $[-R_{min}, 1]$, where R_{min} is an ϵ small penalty received over the entire space, i.e. $R_{min} \in [0, \epsilon]$. By enforcing the additional constraints $\omega_k \in [0, 1]\ \forall k \in [1, 2, \ldots, N_b]$, and subsequently $\|\boldsymbol{\omega}\|_1 = 1$, we direct the arg max problem to then choose a vector $\boldsymbol{\omega}$ such that the expected increase in reward at each step of the process is maximized. This formulation also restricts the generated reward function to a similar range of $[-R_{min}, 1]$.

For clarity, we concisely state the problem as follows:

Objective function:

$$\max_{\boldsymbol{\omega}} \boldsymbol{\omega}^T . \sum_{t=1}^{T_q} \gamma^t \left[\boldsymbol{\phi}({}_q^t s') - \boldsymbol{\phi}({}_q^t s) \right]$$

Decision Variables:

$$\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_{N_b}]^T$$

Constraints considered:

$$\|\boldsymbol{\omega}\|_1 = 1$$

$$\omega_k \in [0, 1] \; \forall k \in [1, 2, \ldots, N_b]$$

$$R(s|\boldsymbol{\omega}) \in [-R_{min}, 1] \; \forall s \in S$$

We propose algorithm 1 as an iterative process to solve for $\boldsymbol{\omega}$, using finite difference methods to approximate the gradient of the weighted sums of the basis functions, and subsequently applying gradient based searches over the space of $\boldsymbol{\omega}$.

Evaluating weights of the reward function

In this section we present the stochastic gradient based search used for estimating the weights of the reward function.

Our iterative algorithm considers each demonstration sequentially, and estimates the set of weights $\boldsymbol{\omega}$ that maximize the expected increase in reward for a given time step as:

$$\max_{\boldsymbol{\omega}} \boldsymbol{\omega}^T . \sum_{t=1}^{T_q} \gamma^t \left[\boldsymbol{\phi}({}_q^t s') - \boldsymbol{\phi}({}_q^t s) \right]$$

We first initialize the $\boldsymbol{\omega}$ to $\boldsymbol{\omega}^{(0)}$, such that $\omega_j^{(0)} = 1/N_b \; \forall j \in [1, N_b]$.

We update the ω_j values as:

$$\omega_j = \omega_j - \alpha \frac{\partial U_i}{\partial \omega_j} \tag{4.8}$$

Represented as a vector equation:

$$\boldsymbol{\omega} = \boldsymbol{\omega} - \alpha \nabla U_i \tag{4.9}$$

4.6 Introduction

4.7 Related Work

Semantic Understanding

Recent advancements in computer vision and machine learning literature have facilitated the development of semantic understanding techniques. [9]. Such techniques have also been applied in conjunction with a variety of robotic concepts, including SLAM [7],

Apprenticeship Learning & Reinforcement Learning

Recent work in apprenticeship learning [1], [2] and [3] is closely tied to inverse reinforcement learning []. Such approaches utilize the demonstrations provided by an expert user to estimate an unknown reward function. The majority of these techniques do so by maximizing the margin between the expected reward obtained by expert policy and the current robot policy. These approaches thus attempt to bound the sub-optimality of a robot policy, by iteratively updating the reward function till convergence.

It may be noted that this method is limited in the following capacity. By iteratively solving an MDP with an updated estimate of the rewards, the authors do not provide a guarantee on whether the output policy provides a greater expected reward than the existing policies previously considered. For an initial sub-optimal policy, margin maximization between the expert and the current policy provides no new information, and solving the MDP with the resultant rewards is unnecessary. Furthermore, in order to evaluate the expected reward for the expert policy, access to the entire expert policy is assumed. In an actual setting of apprenticeship learning, we note that such access is not guaranteed; rather, the user provides an expert policy for a subset of the states present.

4.8 Section-1 Name

Definition 4.8.1. Some definition....

Remark 4.8.2. Some remark.....

Theorem 4.8.3. *Some theorem.....*

Proof. Proof is as follows.... □

4.9 Section-2 Name

Definition 4.9.1. Some definition....

Remark 4.9.2. Some remark.....

4.9.1 Subsection name

Theorem 4.9.3. *Some theorem.....*

Proof. Proof is as follows.... □