Maximum Mean Discrepancy Imitation Learning

Beomjoon Kim
School of Computer Science
McGill University
Montreal, Canada
Email: bkim43@cs.mcgil.ca

Joelle Pineau
School of Computer Science
McGill University
Montreal, Canada
Email: jpineau@cs.mcgill.ca

***Abstract*—Imitation learning is an efficient method for many robots to acquire complex skills. Some recent approaches to imitation learning provide strong theoretical performance guarantees. However, there remain crucial practical issues, especially during the training phase, where the training strategy may require execution of control policies that are possibly harmful to the robot or its environment. Moreover, these algorithms often require more demonstrations than necessary to achieve good performance in practice. This paper introduces a new approach called Maximum-Mean-Discrepancy imitation learning that uses fewer demonstrations and safer exploration policy than existing methods, while preserving strong theoretical guarantees on performance. We demonstrate empirical performance of this method for effective navigation control of a social robot in a populated environment, where safety and efficiency during learning are primary considerations.**

I. Introduction

Imitation learning is a class of algorithms for efficiently optimizing the behaviour of an agent using demonstrations from an expert (or oracle). It is particularly advantageous in complex robotics tasks, where it can be difficult to specify, or directly learn, a model of the dynamics of the robot and its environment. Imitation learning has been used successfully in recent years for a large number of robotic tasks, including helicopter manoeuvring [1], car parking [2], autonomous navigation [14], and robotic surgery [7].

There are several different methods for imitation learning [4]. The most common approach is to use supervised learning to match the actions of the expert, by explicitly minimizing loss with respect to the demonstration data. However, one important difference between imitation learning and standard supervised learning is that in imitation learning, the assumption that the data is *independent and identically distributed (IID)* is generally violated, due to the fact that the state distribution induced by the learned policy is different from that of the oracle. Ignoring this fact means that typical supervised approaches to imitation learning may suffer a quadratic loss in the task horizon T [16], rather than having a loss linear in T as in standard supervised learning tasks.

Recently, an imitation learning algorithm known as Dataset Aggregation (DAgger) [17] was proposed, and this work stands out among other imitation learning algorithms for two reasons. First, the authors showed a reduction from imitation learning to online learning, thereby opening the door to this rich literature. The second major contribution was to show that DAgger has strong theoretical guarantee, and is in fact able to achieve a loss linear in T.

Despite its important insights, DAgger has some drawbacks that can limit its usefulness in some robotic application. First, we note that in DAgger, as in other imitation learning approaches, the expert's actions are stored and used for computation, but they are not directly executed by the robot. The actions carried-out in the environment are always from the robot's own (partially optimized) policy, and the oracle is not allowed to intervene. Unfortunately this poses unnecessary risk, as the robot could use an exploration policy that is not be safe, carrying out actions that are dangerous, either to itself or its environment. This is particularly problematic in the early phases of training, and of particular concern for tasks with high-risk states, for example in robot-assisted surgery [7]. The second drawback of DAgger for practical deployment is its naive data aggregation strategy. Since it queries the oracle at every time step and treats each query equally, it can require large amounts of oracle demonstrations to learn to recover from the mistakes of early policies. Requiring excessive demonstrations from an oracle can be quite expensive, especially if the queries involves highly skilled personnel as in helicopter tasks [1].

The primary contribution of our paper is to propose a new practical imitation learning algorithm to achieve safe and efficient learning in challenging robotic domains. Like DAgger, our algorithm iteratively train policies. Unlike DAgger, however, we propose a query metric that depends on how close an encountered state is from the distribution of data gathered so far, and query the oracle only if this metric is above some threshold. To recover faster from mistakes of policies of previous iterations, we aggregate a data point if it is distributed differently than the datasets gathered so far, *and* if the current policy has made a mistake, and then train a separate policy for each distribution of a dataset. As a result, we learn multiple policies, each specializing in particular regions of the state space where previous policies made mistakes, while focusing on one particular distribution of data. At a run time, we check which one of the policies is trained with the dataset that is most closely distributed with the encountered state. The distribution discrepancy metric that we employ is Maximum Mean Discrepancy (MMD) [10], a criterion that was originally proposed to determine whether two sets of data are from the same or different probability distributions.

The other contributions of our paper are to show that our approach, Maximum Mean Discrepancy Imitation Learning (MMD-IL), has similar theoretical guarantees as DAgger, as well as having better empirical performance. We investigate empirical performance of MMD-IL on two contrasting domains, one a simulated car driving domain, where we can extensively characterize the performance in terms of efficiency, safety and loss. The second set of experiments is performed using a mobile robot intended for personal assistance where we confirm that MMD-IL can efficiently learn to imitate how an expert drives the robot through a crowd, using significantly fewer demonstrations than DAgger.

A noteworthy caveat is that the improved safety and data efficiency observed with MMD-IL is obtained by trading-off data efficiency with computational efficiency. Indeed, there is an extra $O(m)$ computational cost (m = # data points) for computing the MMD criteria necessary to choose the agent's policy[1]. Therefore the method is particularly intended for domains where data is more expensive than computation, which is often the case for imitation learning applications.

II. Technical Background

We denote a state space as S, an action space as A, and a policy space as Π. Given a task horizon T, the agent follows a policy $\pi : S \rightarrow A$ and suffers an immediate loss $L(s, a)$, which we assume bounded in $[0, 1]$. After taking an action the agent reaches another state s' with a particular probability. We denote d_π^t the state distribution at time t after executing π from time 1 to $t-1$, and $d_{\pi_t} = \frac{1}{T}\sum_{t=1}^{T} d_\pi^t$ the average distribution of states over T steps. The expected loss in these T steps is denoted by $J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}[L(s, \pi(s)] = T\mathbb{E}_{s\sim d_\pi}[L(s, \pi(s))]$. The demonstration data is defined by the tuple $< s, a >$, and a trajectory is a sequence of demonstration data. The goal in imitation learning is to find a policy $\pi \in \Pi$ such that the policy minimizes $J(\pi)$. We assume that Π is a closed, bounded and non-empty convex set in Euclidean space.

A. Imitation via Supervised Learning

The standard approach in imitation learning is to use demonstration data as supervised learning data, and learn a policy via regression techniques to map states to actions [4]. During a time step t, we collect a training example from the oracle $(s_t, \pi^*(s_t))$ where $\pi^*(s_t)$ denote action of the oracle at state s_t. Since we do not have access to the true loss $L(s, a)$ in imitation learning, we denote $l(s_t, \pi, \pi^*)$ as the surrogate loss of executing policy π at state s_t with respect to π^*. This can be any convex loss function, such as squared or hinge loss. We learn a policy by solving:

$$\hat{\pi} = argmin_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^*}}[l(s, \pi, \pi^*)] \tag{1}$$

One drawback of this simple approach is that it ignores the fact that the state distributions of the oracle and the learner are different. This occurs because when the learner makes a mistake, the wrong action choice will change the following state distribution such that the states induced are not necessarily distributed from same distribution as the oracle's training data. Ross and Bagnell [16] proved that this could lead to a quadratic loss in task horizon, compared to the linear loss that is observed in standard supervised learning with IID data.

[1]This could likely be reduced to a constant by keeping only a fixed set of points, or else reduced to a log factor by using appropriate data structures.

B. Dataset Aggregation

Dataset Aggregation (DAgger) tackles the non-IID problem by iteratively learning a policy [17]. In its simplest form, it proceeds as follows. Let s_π denote a state visited by executing π, π^* the oracle's policy, and π_i the policy learned at iteration i. Initially, the learner is given an oracle dataset $D_0 = \{(s_{\pi^*}, \pi^*(s_{\pi^*}))\}$, and learns a policy π_0 by any supervised learning method which minimizes the loss of π_0 with respect to π^* at s_{π^*} in D_0. We execute π_0 and collect data from oracle, $D_1 = \{(s_{\pi_0}, \pi^*(s_{\pi_0}))\}$. Policy π_1 is learned from the aggregated dataset, $D_0 \cup D_1$. This iterative process of policy execution, data aggregation, and policy optimization continues for N iterations. The purpose of iterative learning is to recover from mistakes of the previous policy. Since the state distribution is not IID, the learned policy is likely to make mistakes in regions of the state space that are unfamiliar. Hence by aggregating the oracle's demonstrations at states encountered by the previous policy, the hope is that the agent will learn to recover from those mistakes.

Let $Q_t^{\pi'}(s, \pi)$ denote the t-step loss of executing π in the initial state and then executing π'. We state here an important theorem from Ross and Bagnell [17].

Theorem 1 (from [17]): *Let π be such that $\mathbb{E}_{s\sim d_\pi}[l(s, \pi, \pi^*)] = \epsilon$. If $Q_{T-t+1}^{\pi^*}(s, \pi) - Q_{T-t+1}^{\pi^*}(s, \pi^*) \leq u$ for all action a,t $\in \{1, 2, ..., T\}$, then $J(\pi) \leq J(\pi^*) + uT\epsilon$.*

This theorem basically that if π has expected loss of ϵ in state distribution induced by π, differs from π^* by one action at time step t, and if the cumulative cost is bounded by u, then the task cost with respect to π^* is $O(uT)$.

One of the major contributions of DAgger is its reduction of imitation learning to online learning. In an online learning problem, the algorithm executes a policy and suffers a loss $l_i(\pi_i)$ at iteration i. The algorithm then modifies its policy to produce a new policy, π_{i+1}, suffers loss $l_{i+1}(\pi_{i+1})$, and so on. A class of online algorithms known as no-regret algorithms guarantees the following:

$$\frac{1}{N}\sum_{i=1}^{N} l_i(\pi_i) - min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N} l_i(\pi) \leq \delta_N \tag{2}$$

where $lim_{N\rightarrow\infty}\delta_N = 0$. If we assume a strongly convex loss function, then Follow-The-Leader is a no-regret algorithm where in each iteration, it uses the policy that works the best so far: $\pi_{i+1} = argmin_{\pi\in\Pi}\sum_{j=1}^{i} l_j(\pi)$ [13]. DAgger can be seen as a variant of a Follow-The-Leader algorithm in that it chooses the policy that minimizes the surrogate loss on data gathered so far. In the reduction of imitation learning to online learning, we let $l_i(s, \pi, \pi^*) = \mathbb{E}_{s\sim d_{\pi_i}}[l(s, \pi_i, \pi^*)]$,

N be the number of training iterations in DAgger, and $\epsilon_N = min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} l_i(s, \pi, \pi^*)$ be the minimum loss we can achieve in the policy space Π in hindsight. Assume $l(s, \pi, \pi^*)$ is strongly convex and denote $\pi_{1:N}$ the sequence of learned policies. Then DAgger guarantees the following if infinite samples per iteration are available:

Theorem 2 (from [17]): *For DAgger, if N is $O(uTlogT)$ and $Q^{\pi'}_{T-t+1}(s, \pi) - Q^{\pi'}_{T-t+1}(s, \pi^*) \leq u$ then there exists a policy $\pi \in \pi_{1:N}$ such that $J(\pi) \leq J(\pi^*) + uT\epsilon_N$.*

The important result here is that DAgger guarantees linear loss in terms of the task horizon T.

III. Max Mean Discrepancy imitation learning

The first step towards reducing unsafe and unnecessary exploration is to query the oracle and execute the oracle's action (i.e. oracle intervention) in cases where the state encountered is distributed differently from data gathered so far, and the learner is about to make a mistake. We formally define a mistake as the discrepancy between the policy of oracle and that of the agent:

$$||\pi(s) - \pi^*(s)||_1 > \gamma \tag{3}$$

where γ is a pre-defined parameter. Note that in the case of discrete actions, the metric needs to be defined. This also reduces the problem of unnecessary exploration in states that are not reachable in reality, because the oracle is able to direct the learner to states that are reachable in reality. We explain below how this is computed and used in our algorithm.

We now introduce the criterion known as Maximum Mean Discrepancy, which is used as a distance metric to determine when to query the expert for demonstration, and which of the policies to follow out of multiple policies learned so far.

A. Maximum Mean Discrepancy

Given two sets of data, $X := \{x_1, ..., x_m\}$ and $Y := \{y_1, ..., y_n\}$ drawn IID from p and q respectively, Maximum Mean Discrepancy criterion determines whether $p = q$ or $p \neq q$ in reproducing kernel Hilbert space (RKHS).

Definition 1 (from [10]): *Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ and let p, q, X, Y be defined as above. Then MMD and its empirical estimate are defined as:*

$$MMD[\mathcal{F}, p, q] := sup_{f \in \mathcal{F}}(\mathbb{E}[f(x)] - \mathbb{E}[f(y)])$$
$$MMD[\mathcal{F}, X, Y] := sup_{f \in \mathcal{F}}(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i))$$

MMD comes with important theorem which we restate here.

Theorem 3 (from [10]): *Let $\mathcal{F}$ be a unit ball in reproducing kernel Hilbert space $\mathcal{H}$, defined on compact metric space $\mathcal{X}$, with associated kernel $k(\cdot, \cdot)$. Then $MMD[\mathcal{F}, p, q] = 0$ if and only if $p = q$.*

Algorithm 1 MMD Imitation Learning

```
Initialize D_0 ← D*
Train π̂_0 = Train(D_0)
for i = 0 to N − 1 do
  D_{i+1} = ∅
  for j = 1 to T do
    π̂^(i)_MMD(s_j) = argmin_{π̂_{0:i}} MMD[F, s_j, D_{0:i}]
    if min MMD[F, s_j, D_{0:i}] > α then
      π*(s_j) = Query the oracle
      if |π*(s_j) − π̂^(i)_MMD(s_j)| > γ then
        Execute π*(s_j)
        Collect data: D_{i+1} = D_{i+1} ∪ (s_j, π*(s_j))
      else
        Execute π̂^(i)_MMD(s_j)
      end if
    else
      Execute π̂^(i)_MMD(s_j)
    end if
  end for
  Train π̂_{i+1} = Train(D_{i+1})
end for
```

Intuitively, we can expect $MMD[\mathcal{F}, X, Y]$ to be small if $p = q$, and the quantity to be large if distributions are far apart. Note that $\mathcal{H}$ is an RKHS endowed with common kernels, such as Gaussian kernel (for more choices of kernels, see [10]).

B. MMD-IL Algorithm

The MMD-IL algorithm (Algorithm 1) is an iterative procedure that train multiple policies. Initially, the oracle provides D^*, a demonstration of the given task. We use this data to train the first policy, $\hat{\pi}_0$. The agent then goes into successive rounds during which it executes one of the learned policies over trajectories of length T, aggregating new data points to train a new policy if they satisfy conditions we discuss below.

During a policy execution, the agent must decide at every time step whether to apply one of the learned policies, $\{\hat{\pi}_0, \ldots, \hat{\pi}_i\}$, or to query the oracle for a demonstration. Let $MMD[\mathcal{F}, s, D_{0:i}]$ be the set of MMDs evaluated between s and each of $\{D_1, .., D_i\}$, evaluated according to Equation 4, the empirical MMD for state s and dataset D_i, in RKHS endowed with kernel $k(\cdot, \cdot)$:

$$\begin{aligned} &MMD[\mathcal{F}, s, D_i] \qquad (4)\\ &= sup_{f \in \mathcal{F}}(f(s) - \frac{1}{n}\sum_{s' \in D_i} f(s')) \\ &= \Big[k(s,s) - \frac{2}{n}\sum_{s' \in D_i} k(s, s') + \frac{1}{n^2}\sum_{s', s'' \in D_i} k(s', s'')\Big]^{\frac{1}{2}} \end{aligned}$$

The MMD criterion is first used to determine which of the available policies, $\{\hat{\pi}_0, \ldots, \hat{\pi}_i\}$, is trained with data that has minimum MMD value with the current state. The idea is to use the policy that is trained with the dataset whose distribution is closest to the current state. Then, if the minimum MMD between current state and datasets gathered so far is larger than α, the agent queries the oracle for a demonstration, $\pi^*(s)$. This action gets compared with $\pi^{(i)}_{MMD}(s)$ using the

threshold γ, to decide whether to execute $\pi^*(s)$ or $\pi^{(i)}_{MMD}(s)$. If the encountered state has discrepancy bigger than α and the current policy makes a mistake, MMD-IL aggregates the data point, $(s, \pi^*(s))$. This allows the agent to learn new policies specifically targeting unexplored states that are reachable from current policy and the previous learned policies are still making mistakes, while controlling the frequency of queries.

The MMD-IL algorithm comes with two pre-defined parameters. The first, γ as introduced above, controls the frequency of usage of oracle actions. If γ is set to a small value, the oracle's queried action will be executed and its demonstration will be aggregated even for small mistakes. The parameter γ can be set using domain knowledge. Note that we do not execute the oracle's action even after it is queried, if the current policy is not making a mistake. This allows the agent to explore states that it will encounter during an execution of its policy, as long as the policy does not differ from that of the oracle's. The parameter α, on the other hand, allows the user to control how much the oracle monitoring (i.e. frequency of queries to the oracle) is necessary. For instance, if the robot is learning a risk sensitive task, α can be set to 0 in which case the oracle will provide an action at every step, and the agent uses oracle's action if the current policy is making a mistake. If α is set to a high value, we only query the oracle when the encountered state is significantly far away from distributions of data gathered so far. This is more suitable for cases where the oracle demonstration data is expensive, and the oracle is not present with the robot in every step of the policy execution.

C. Space and Time Complexities

Assume MMD-IL trained N policies, $\{\hat{\pi}_1, \ldots, \hat{\pi}_N\}$, from datasets $\{D_1, .., D_N\}$. In calculating the MMD value between each dataset and the given state, the third term in Eqn 4 is most expensive to compute, requiring $O(m^2)$ time (where m is the number of points in a given dataset D_i). But this term can be pre-computed once D_i is fully collected, and so at run time we only evaluate the second term, which is $O(m)$. Thus, for each query, our algorithm takes $O(mN)$ time, which corresponds to the total number of datapoints gathered so far. We also need a $O(mN)$ space to store the data. In practice, this may be reduced in a number of ways, for instance, by imposing a stopping criterion for the number of iterations based on MMD values evaluated with new states, or limiting the number of data points by discarding (near-)duplicate data points.

D. Theoretical Consequences

In order to prove that our policy is robust, we require a guarantee that our policy achieves a loss linear in T as N gets large, like supervised learning algorithms[16]. We first assume that for i^{th} iteration, the oracle interventions have bounded effect on total variation distance by some constant $C_i \leq 2$:

$$||d^{(i)}_{\pi_{MMD}} - d^{(i)}_{\hat{\pi}_{MMD}}||_1 \leq C_i$$

Here, $d^{(i)}_{\pi_{MMD}}$ denotes the average empirical distribution of states when the oracle intervened at least once, and $d^{(i)}_{\hat{\pi}_{MMD}}$ denotes the average empirical distribution of states when the MMD policy ($\hat{\pi}^{(i)}_{MMD}$) is used throughout the task horizon. We assume that C_i reaches zero as N approaches infinity (i.e. the oracle intervenes less as the number of iterations grows), and the policy at the latest iteration achieves minimum expected loss out of all the policies learned so far. Denote l_{max} the upper bound on loss.

Theorem 4 *For MMD policy $\hat{\pi}_{MMD}$,*

$$\mathbb{E}_{s\sim d_{\hat{\pi}_{MMD}}}[l(s, \hat{\pi}_{MMD})] \leq \epsilon_n + \delta_N + \frac{l_{max}}{N}\sum_{i=1}^{N} C_i,$$

for δ_N the average regret of $\hat{\pi}_{MMD}$ (as defined in Eqn 2).

Proof

$$\begin{aligned}
&\mathbb{E}_{s\sim d^{(N)}_{\hat{\pi}_{MMD}}}[l(s, \hat{\pi}^{(N)}_{MMD}, \pi^*)] \\
&\leq min_{\hat{\pi}\in\hat{\pi}^{(1:N)}_{MMD}} \mathbb{E}_{s\sim d_{\hat{\pi}}}[l(s, \hat{\pi}, \pi^*)] \\
&\leq \frac{1}{N}\sum_{i=1}^{N}[\mathbb{E}_{s\sim d^{(i)}_{\hat{\pi}_{MMD}}}(l(s, \hat{\pi}^{(i)}_{MMD}, \pi^*))] \\
&\leq \frac{1}{N}\sum_{i=1}^{N}[\mathbb{E}_{s\sim d^{(i)}_{\pi_{MMD}}}(l(s, \hat{\pi}^{(i)}_{MMD}, \pi^*)) + l_{max}C_i] \\
&\leq \delta_N + min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N} l_i(\pi) + \sum_{i=1}^{N} l_{max}\frac{C_i}{N} \\
&= \delta_N + \epsilon_N + \frac{l_{max}}{N}\sum_{i=1}^{N} C_i
\end{aligned}$$

The first two lines follow from simple algebra. The third line follows from our first assumption above, and using the definition of expectation. The fourth line follows from Eqn 2. The last line follows from the definition of ϵ_N (Thm 2). Applying Theorem 1, we can see that the MMD policy will yield a bound linear in the task horizon T as well.

IV. Experiments

A. Car Brake Control Simulation

We applied MMD-IL for the vehicle brake control simulation introduced in Hester and Stone [11]. The task here is to go from an initial velocity to the target velocity, and maintain the target velocity. The robot can either press the acceleration pedal or the brake pedal, but cannot press both of them at the same time. The difference between the experiment in [11] and ours is that we pose it as a regression problem, where the action of the robot can be any positive positions of pedals pressed rather than discretizing the position of pedals. A state is represented as four continuous valued features: target velocity, current velocity, current positions of brake and acceleration pedals. Given the state, the robot has to predict a two-dimensional action: position of acceleration pedal and brake pedal. The reward is calculated as -10 times the error in velocity in m/s. The initial velocity is set to 2m/s, and the target velocity is set to 7m/s.

The oracle was implemented using the dynamics between the amount of pedals pressed and output velocity from which it can calculate the optimal velocity at the current state. The

robot has no knowledge of these dynamics, and receives only the demonstrations from the oracle. We used a mean-squared loss based decision tree regressor for learning policies, as implemented in the Scikit-learn machine learning library [15]. A Gaussian kernel was used for computing the MMD, γ was set to 0.1, and α was set to various values. During the test phase (from which we computed the results in the plots below), we added random noise to the dynamics to simulate a more realistic scenario where the output velocity is governed by other factors such as friction and wind. We repeated each iteration's policy 15 times, gathering mean and standard deviation of each iteration's result.

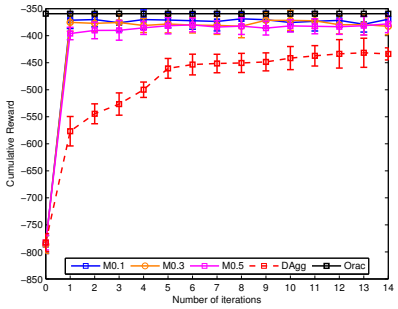

Fig. 1: Cumulative reward of DAgger and MMD-IL. M$\{0.1, 0.3, 0.5\}$ indicates MMD-IL with $\alpha = 0.1, 0.3, 0.5$.

Figure 1 shows the average cumulative reward after execution over the task horizon, $T = 100$. As can be seen from the plot, MMD-IL learns to recover from necessary mistakes after the first iteration. The cumulative reward after the first iteration stays about the same, even though there are few mistakes observed. In essence, we could have stopped training MMD-IL after the first iteration, because its reward is very close to that of oracle's. DAgger, on the other hand, struggles to learn from mistakes and struggles to gain reward comparable to MMD-IL. This is due to two reasons. First, as mentioned above, DAgger explores many states that are not likely to occur in reality. For instance, we observed that it was exploring parts of the state space where the velocity of the robot is above 16m/s, when the target was 7m/s. Second, it is possible that the function space for our regressor is not rich enough to represent a good enough function for the entire task; recall that DAgger uses a single policy, whereas MMD-IL uses a collection of policies. This is investigated further below.

Next, in Figure 2 we consider the number of oracle demonstrations used by both methods at each iteration. As expected, the number of oracle demonstrations required by DAgger increases linearly in the task horizon, whereas MMD-IL requires much less. MMD-IL is shown to perform much better than DAgger even using only around 200 oracle demonstration actions (one iteration), whereas DAgger needed at least 800 data points (eight iterations) to achieve near competitive performance. We also show results for various settings of the parameter α, and as expected, as α gets larger, MMD-IL is more conservative in querying the oracle.

Finally, we investigated the complexity of regression function space required by MMD-IL. Here we fixed the number of iterations at five, α at 0.3, and changed the *maximum* depth of decision trees, to determine how complex the fitted function should be. As seen in Figure 3, at around a tree depth of seven, DAgger begins to receive consistent cumulative rewards (suggesting appropriate function complexity has been reached), which is still less than what MMD-IL achieves with a tree of depth 3. We do note again that MMD-IL uses multiple decision trees, instead of one like in DAgger. We conjecture that the better performance of MMD-IL here can be explained via the notion of weak learnability, which suggests that training weak policies such that they focus on mistakes of other weak policies could lead to a stronger final policy if the final policy is an aggregation of these weak policies [18]. This may be particularly noteworthy for tackling more complex robotics tasks, where a single regressor could be computationally infeasible.

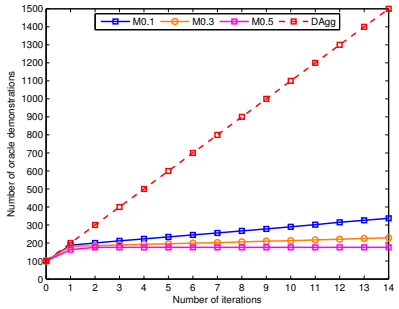

Fig. 2: Number of demonstration data required at each iteration for DAgger and MMD-IL. M$\{0.1, 0.3, 0.5\}$ indicates MMD-IL with $\alpha = 0.1, 0.3, 0.5$.

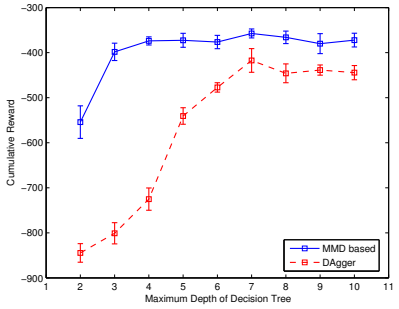

Fig. 3: Depth of tree vs Cumulative Rewards

B. Robot Navigation Task

In a second set of experiments, we compared MMD-IL and DAgger for real robot navigation in static and dynamic environments. The navigation tasks considered focus on situations where unsafe exploration may be a severe issue, for example where there is a risk that the robot will collide with pedestrians or static obstacles. To avoid major accidents in this early phase of development, we do restrict our experiments to a controlled laboratory environment. We begin with a first set of experiments involving only static (but unmapped) obstacles, and then move to a second set of experiments involving

dynamic obstacles, in the form of moving pedestrians. In this case, the robot needs to not only avoid the pedestrians, but also maintain a socially acceptable distance, therefore the use of imitation learning is particularly appropriate since notions such as socially acceptable movement are difficult to define using a standard cost function.

The robot used for these experiments is a robotic wheelchair equipped with an RGB-D (Kinect) sensor [6], as shown in Figure 4. Unlike the previous set of experiments, the state of the system here is not known precisely, and must be inferred in real-time from the sensor data. The camera provides a point cloud containing 3D coordinates of the surrounding obstacles, as well as RGB information for each point in the cloud. For the static environment experiment, we set up a

Fig. 4: Picture of the robotic wheelchair equipped with Kinect

track that is approximately 3m wide and 15m long, in which three static obstacles were placed. The distance from the goal and the initial position of the robot was approximately 13m. While getting to the goal, the robot had to avoid all three obstacles and stay inside the track. The dynamic environment experiment was also conducted in the same track, but the environment involved a static obstacle and a pedestrian that is trying to reach the robot's initial position from robot's goal (i.e. they need to exchange positions in a narrow hallway). The robot therefore has to simultaneously avoid the static obstacle and the moving pedestrian, while staying inside the track and getting to its goal.

The action space is a continuous two dimensional vector space representing angular and linear velocities of the robot. The state space is defined by environmental features such as obstacle densities and velocities extracted from the point cloud. We used 3×7 grid cells, each grid cell being 1m by 1m, to represent the environmental feature information in each cell. To compute density of a cell, we counted the number of points in the cloud belonging to that cell. Computing the velocity in a cell involved several steps. First, we used 2D optical flow algorithm [12] on two consecutive RGB images provided by Kinect, which gives us correspondence from each pixel of RGB image at time $t-1$ to a pixel of RGB image at time t. The correspondence information between two frames was then transferred to 3D space by mapping each pixel to a point in the point cloud at $t-1$ and t accordingly. Then, the velocity of a point was calculated by simply subtracting the 3D coordinate of a point in $t-1$ from corresponding point in t. Finally, velocity of each cell was calculated by averaging the velocity of each point in that cell. For dynamic environment experiment, this gave us a 63 continuous dimensional state space (3×7 grid cells * 3 environmental features(density, and 2 horizontal velocities). For the static environment, we did not use the velocity information (=21 dimensional state space).

To train the robot, we followed Algorithm 1 and DAgger. In the initial iteration, both MMD-IL and DAgger were given a demonstration trajectory of how to avoid static and dynamic obstacles and get to the goal by a human pilot (oracle) who tele-operated the robot. For MMD-IL, we allowed the maximum oracle monitoring (i.e. $\alpha = 0$) as the collision with dynamic obstacles needed to be prevented (for the safety of the pedestrian). At each iteration, we executed the previous iteration's policy while applying π^* if it is different from the robot's policy by γ, that we set using domain knolwedge[2]. As a result, the oracle was able to guide the robot when it was about to go out of the track or collide with an obstacle. For DAgger, the policy was executed and the oracle provided actions, but did not intervene (as per the algorithm's definition) until a collision was imminent, at which point the exploration was manually terminated by the operator and the dataset was aggregated.

To test each iteration's policy π_i, we stopped gathering data and executed the policy. If the policy was going to make a mistake (i.e. hit one of the obstacles or go outside of the track), the oracle intervened and corrected its mistake so that the robot could get to the goal. The purpose of the oracle intervention in the test phase was to be able to numerically compare the policies throughout the whole task horizon T. Otherwise, as these policies have not finished training, it is hard to numerically compare their performances throughout the task horizon as they tend to collide with obstacles and go out of the track. To compare two algorithms, we used the oracle intervention ratio calculated by $\frac{|S^*|}{T}$, where S^* is the set of states in which the oracle intervened. We repeated the navigation task using each iteration's policy four times, to get the mean and standard deviation of the ratio. Note that this ratio is analogous to classification error, because S^* is the set of mistakes of the policy and T is the total number of test set.

1) Static Environment: Figure 6 shows the trajectory results of MMD-IL and DAgger. We observe that MMD-IL almost never makes the same mistake twice. For instance, in the first iteration it made mistakes close to the goal, but in second iteration there are no mistakes at this region. Likewise, in the second iteration it makes mistakes when avoiding the third obstacle, but in the third iteration this is prevented. In the final iteration, it successfully avoids all three obstacles and reaches the goal, behaving very similarly to the oracle.

In contrast, DAgger struggles to learn to recover from mistakes. For instance, in the first iteration it started making

[2]The linear and angular velocities of the robot are mapped to 0 to 1.0 scale. We set γ to 0.2.

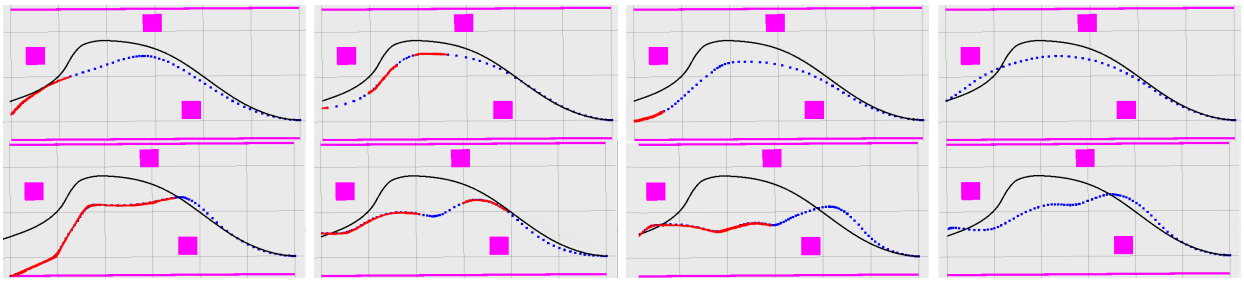

Fig. 6: Example trajectories executed by policies in iterations 1 to 4, left to right, for MMD-IL(top) and for iterations 1,3,5 and 8 DAgger (bottom). Initial human demonstration, D^* (black - this is the same in all panels), each iteration's policy (dotted blue), and oracle's interventions during the execution of the policy (red). Obstacles (pink squares), and track(pink lines). The initial position of the robot is at the right and the goal position is at the left.

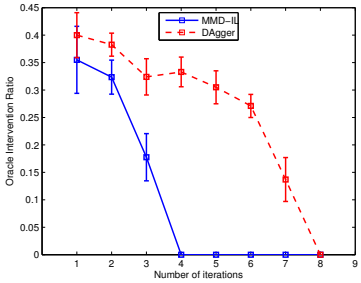

Fig. 5: Oracle intervention ratio during the execution of each iteration's policy for static environment.

mistakes near the second obstacle until the end. The mistakes at these regions persisted until iteration three. We would also like to note that for DAgger, it was harder for the oracle to provide the optimal demonstration than MMD-IL because the oracle's input actions were not executed. For instance, when the robot was getting closer to an obstacle, it was hard for the oracle to know how much to turn because the oracle controls the robot not only by how close the robot is to the obstacle, but also by direction and speed of the robot. Since the robot was not executing the oracle's actions, these factors were hard to consider and the oracle ended up giving suboptimal policy. This lead to a suboptimal final policy for DAgger as shown in the bottom right-most panel of Figure 6, which features "wiggling" motions.

Figure 5 shows how often the oracle had to intervene during the navigation to make the robot get to the goal. As the plot shows, MMD-IL learns to navigate by itself after iteration 3, compared to iteration 7 for DAgger. This again supports our claim that MMD-IL requires much less oracle demonstrations.

2) Dynamic Environment: This experiment involved a pedestrian that interfered with the robot's trajectory. The pedestrian was asked to navigate freely to get to its goal, except to move towards the static obstacle. While the pedestrian showed irregular trajectories, the general trajectories were either moving to the right to avoid the robot, or moving forward and then slightly to the right. It all depended on who moved first; if the pedestrian moved first, it was usually the case that the pedestrian moved to the right to avoid the robot. If the robot moved first to the right to avoid the static obstacle and pedestrian, the pedestrian simply moved forward.

In the initial oracle demonstration step, the pedestrian happened to move forward so the oracle turned to the right in an attempt to avoid both approaching pedestrian and the static obstacle on the left. Due to this reason, the policy learned have a tendency to move towards right first, but learns to stop this motion and go straight if the pedestrian is moving towards the right. The final policies' trajectories are shown in Figure 7.

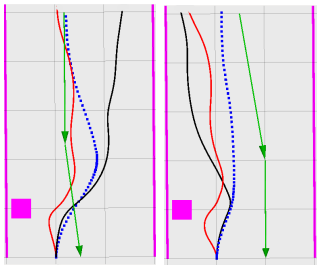

Fig. 7: Left: Case where the pedestrian moved forward. Right: Case where the pedestrian moved to the right. Oracle's initial trajectory (black), final policy of MMD-IL (dotted blue) , and final policy of DAgger (red) are shown. The green arrows are the approximate pedestrian trajectories (manually annotated).

As the figure shows, MMD-IL showed a trajectory that is much more socially adaptive when trying to avoid the pedestrian, by imitating the oracle almost perfectly at these situations. Once both pedestrian and static obstacles are avoided, it deviates from the oracle's trajectory but this is unimportant as there are no more obstacles. In contrary, DAgger's final policy involved moving towards the static obstacle, and waiting until the pedestrian passed. This is a less efficient policy, and is a direct result of the fact that when the oracle's policies are not being executed, the oracle cannot keep the demonstrations within the distribution of the optimal policy. This policy is not only suboptimal but also socially ineffective, in that it strayed too close to the pedestrian.

Figure 8 shows the oracle intervention ratio plot. Again, MMD-IL learned to navigate in this environment with less demonstration data than DAgger. Note that the high variance was caused by the irregular trajectories of the pedestrian.

V. Related Work

There are other supervised imitation learning algorithms which query the oracle incrementally, besides DAgger. In [8],

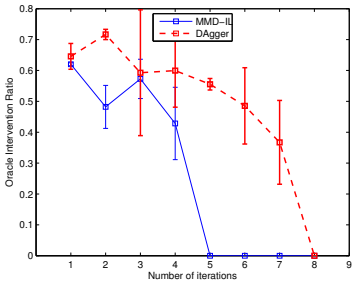

Fig. 8: Oracle intervention ratio during the execution of each iteration's policy for dynamic environment.

a confidence-based imitation learning is proposed. The idea is to employ a Gaussian Mixture Model per action label, and then use certainty of a classification output to determine when to query the oracle. The limitation of this work is that the query metric restricts the choice of classifier to a probabilistic one. In [9], this work was extended to use other metrics such as distance to the closest point in a training dataset and distance to the decision boundary. Another line of work is feedback based imitation learning, in which the agent receives feedback from the oracle as to correct the behaviour of the current policy [3, 5]. The idea is to use an Markov Decision Process, in which rewards are specified by the oracle feedback, and to correct the current policy such that it maximizes the return. Our intuition behind MMD-IL is that these additional queries and feedbacks are required because state distribution is different when policy changes. The key difference between MMD-IL and these works is that it queries the oracle only when there is a significant discrepancy in state distributions, as measured via MMD, and corrects the policy only when the current policy makes a mistake. Moreover, MMD-IL provides theoretical guarantee using the reduction from imitation learning to no-regret online learning.

VI. Conclusion

In this work we proposed MMD-IL, an imitation learning algorithm that is more data efficient and safer to use in practice than the state-of-art imitation learning method DAgger, albeit at the cost of some extra computation. We also showed that MMD-IL guarantees a loss linear in terms of the task horizon, as in standard supervised learning tasks. Moreover, MMD-IL comes with pre-defined parameters, through which the user can control how much demonstration data is collected, and when the oracle's policy should be used. MMD-IL showed good empirical performance in a variety of tasks, including navigating in the presence of unmodeled static and dynamic obstacles using standard sensing technology. MMD-IL outperformed DAgger in all tasks considered, mainly due to its ability to learn to quickly recover from mistakes by using the oracle to guide the learning, though limiting interventions to key areas by using the MMD criterion. The improvement in performance, however, was achieved at a extra computational cost of O(m). In future, we hope to integrate imitation learning with learning by trial-and-error, or reinforcement learning, as to consider the case where we only have small number of oracle data, or when the oracle is sub-optimal.

Acknowledgement

Authors would like to thank Stephane Ross and Doina Precup for helpful discussions. Many thanks to A. Sutcliffe, A. el Fathi, H. Nguyen and R. Gourdeau for their contributions to the empirical evaluation. Funding was provided by NSERC (NCNFR), FQRNT (INTER and REPARTI) and CIHR (CanWheel)

References

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *NIPS 19*, 2007.

[2] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun. Apprenticeship learning for motion planning, with application to parking lot navigation. In *IROS*, 2008.

[3] B.D. Argall, B. Browning, and M. Veloso. Learning by demonstration with critique from a human teacher. 2009.

[4] B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57, 2009.

[5] B.D. Argall, B. Browning, and M. Veloso. Policy feedback for refinement of learned motion control on a mobile robot. *International Journal of Social Robotics*, 4, 2012.

[6] A. Atrash, R. Kaplow, J. Villemure, R. West, H. Yamani, and J. Pineau. Development and validation of a robust speech interface for improved human-robot interaction. *International Journal of Social Robotics*, 1, 2009.

[7] J. Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X. Fu, K. Goldberg, and P. Abbeel. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *ICRA*, 2010.

[8] S. Chernova and M. Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *AAMAS*, 2007.

[9] S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34, 2009.

[10] A. Gretton, K. Borgwardt, M. Rasch, B. Schlkopf, and A. Smola. A kernel method for the two sample problem. In *NIPS*, 2007.

[11] T. Hester, M. Quinlan, and P. Stone. RTMBA: A real-time model-based reinforcement learning architecture for robot control. In *ICRA*, 2012.

[12] B. Horn and B. Schunck. Determining optical flow. *Artif. Intell.*, 17, 1981.

[13] S. M. Kakade and S. Shalev-shwartz. Mind the duality gap: Logarithmic regret algorithms for online optimization. In *NIPS*, 2008.

[14] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. Off-road obstacle avoidance through end-to-end learning. In *NIPS*, 2005.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[16] S. Ross and J. A. Bagnell. Efficient reductions for imitation learning. In *AISTATS*, 2010.

[17] S. Ross, G. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[18] R. Schapire. The strength of weak learnability. *Machine Learning*, 5, 1990.