# *Generalizing Over Uncertain Dynamics for Online Trajectory Generation*

# Generalizing over Uncertain Dynamics for Online Trajectory Generation

Beomjoon Kim, Albert Kim, Hongkai Dai, Leslie Kaelbling, Tomas Lozano-Perez

**Abstract** We present an algorithm which learns an online trajectory generator that can generalize over varying and uncertain dynamics. When the dynamics is certain, the algorithm generalizes across model parameters. When the dynamics is partially observable, the algorithm generalizes across different observations. To do this, we employ recent advances in supervised imitation learning to learn a trajectory generator from a set of example trajectories computed by a trajectory optimizer. In experiments in two simulated domains, it finds solutions that are nearly as good as, and sometimes better than, those obtained by calling the trajectory optimizer on line. The online execution time is dramatically decreased, and the off-line training time is reasonable.

## 1 Introduction

Given a known deterministic model of the dynamics of a system, a start and goal state, and a cost function to be minimized, trajectory optimization methods [27] can be used to generate a trajectory that connects the start and goal states, respects the constraints imposed by the dynamics, and (locally) minimizes the cost subject to those constraints. A significant limitation to the application of these methods is the computational time required to solve the difficult non-linear program for generating a near-optimal trajectory. In addition, standard techniques [27] require the transition dynamics to be known with certainty.

We are interested in solving problems online in domains that are not completely understood in advance and that require fast action selection. In such domains we will not know, offline, the exact dynamics of the system we want to control. Online, we will receive information that results in a posterior distribution over the domain dynamics. We seek to design an overall method that combines *offline* trajectory optimization and inductive learning methods to construct an *online* execution system that efficiently generates actions based on observations of the domain.

For example, we might wish a robot to move objects along a surface, potentially picking them up or pushing them, and making choices of grasps and contacts. The best way to achieve this depends on properties of the object, such as the coefficient of friction of the robot's contacts with the object and the object's center of mass (COM), which determine the system's dynamics. If we knew the friction and center

MIT, Cambridge, MA, USA, e-mail: beomjoon,alkim,daih,lpk,tlp@mit.edu

of mass, it would be relatively straightforward to find an appropriate trajectory using trajectory optimization, but solving a non-linear optimization with large number of decision variables and constraints generally takes a significant amount of time.

This work builds on recent advances in supervised imitation learning [10, 11] to design a new learning-based online trajectory generation algorithm called TOIL (Trajectory Optimization as Inductive Learning). We present two general problem settings. In the *completely observable* setting, we assume that at execution time the world dynamics will be fully observed; in the manipulation domain, this would correspond to observing the friction and COM of the object. In the *partially observable* setting, we assume that properties of the domain that govern its dynamics are only partially observed; for example, observing the height and shape of an object would allow us to make a "guess" in the form of a posterior distribution over these parameters to the dynamics, conditioned on the online observations. In both cases, we desire the online action-selection to run much more quickly than would be possible if it were necessary to run a traditional trajectory optimization algorithm online.

More concretely, we aim to build a trajectory generator that, for a given initial state and goal, maps the values of the dynamics parameters to a trajectory in the observable setting, or maps from an observation to a trajectory in the partially observable setting. We do this by training a regression function that maps both the dynamics parameters and the current system state to an appropriate control action. The trajectories used for off-line training are generated by using an existing trajectory optimizer that solves non-linear programs. To minimize the number of training trajectories required, we take an active-learning approach based on MMD-IL [11], which uses an anomaly-detection strategy to determine which parts of the state space require additional training data.

The idea of reducing trajectory generation to supervised learning has been suggested before, but it is quite difficult to learn a single regressor that generalizes over a large number of trajectories. Our approach, instead, is to learn a number of local controllers (regressors) based on individual trajectories, for a given value of the dynamic parameter or observation. During training TOIL decides when additional controllers are needed based on a measure of distance between the states reached during execution and the existing set of controllers. During execution, TOIL uses the same distance criterion to select which controller among the learned controllers to use at each time step.

We evaluate TOIL in two domains: aircraft path-finding and robot manipulation. The aircraft domain is a path-planning task in which a sequence of control inputs that drive the aircraft to the goal must be found. For the observable case of this task, we show that TOIL is able to generate a trajectory whose performance is on par with the traditional trajectory optimizer while reducing the generation time by a factor of 44. In the partially observable case, we show TOIL's success rate is better than that of the trajectory optimizer, while the generation time is reduced by a factor of 52.

In the manipulation domain, a robotic hand needs to move a cylindrical object from an initial position to a target position. This involves high dimensional state and control input. We show that in this domain, for both the observable and partially observable settings, TOIL is able to reduce the trajectory generation time by a factor

of thousands. Moreover, we show that TOIL can generalize over different shapes of the cylinder, and generate trajectories whose success rate is almost as same as traditional trajectory optimization.[1]

## 2 Related Work

The idea of combining multiple trajectories to obtain a control policy has a long history, for example [1, 20, 2]. Recently, there has been a surge of interest in learning-based methods for constructing control policies from a set of trajectories obtained from trajectory optimization [3, 4, 9]. These methods generalize a set of relatively expensive trajectory optimizations to produce a policy represented in a form that can be efficiently executed on-line.

Our goal is similar, except that we are trying to generalize over dynamic parameters. Our training examples are trajectories labeled either directly with dynamics parameters or with observations that give a distribution over dynamic parameters.

Our approach is based on the paradigm of imitation learning [21, 25]. This learning paradigm has had many successes in robotics, notably helicopter maneuvering, UAV reactive control, and robot surgery [22, 23, 24]. In imitation learning, the goal is to replicate (or improve upon) trajectories acquired from an "oracle," usually an expert human. The work in this paper can be seen as replacing the oracle in imitation learning with a trajectory optimizer.

DAgger [10] is an influential algorithm that addresses a fundamental problem in standard supervised approaches to imitation learning. Direct application of supervised learning suffers from the fact that the state distribution induced by the learned policy differs from that of the oracle. DAgger adopts intuition from online learning, a branch of theoretical machine learning, to address this problem by iteratively executing the learned policy, collecting data from the oracle, and then learning a new policy from the aggregated data. An important drawback of Dagger is that it queries the oracle at each time step, which would require solving a non-linear optimization program every time step during training in our case.

Our work extends Maximum Mean Discrepancy Imitation Learning (MMD-IL) [11], a recently proposed imitation learning method designed to be efficient in its access to the oracle. MMD-IL learns a set of trajectories to represent a policy and uses the Maximum Mean Discrepancy criterion [17] as a metric to decide when to query the oracle for a new trajectory. In this paper, we also take this approach with a modification that makes it parameter free.

[1] the video of this can be found at: `https://www.youtube.com/watch?v=r9o0pUIXV6w`

## 3 Completely observable dynamics

For clarity of exposition, we begin by defining the learning problem and the operation of TOIL in the completely observable case; in section 4 we extend it to the more realistic partially observable case.

We assume a fixed initial state $x_0$, goal state $x_g$ (or goal criterion) and cost functional $\mathscr{J}$. We also assume that the transition dynamics are drawn from a known class, with a particular instance determined by the parameter $\alpha$ drawn from set $\mathscr{A}$: $\dot{x} = f_\alpha(x,u)$. Our overall aim is to learn to map values of $\alpha$ to trajectories $\tau$ that go from $x_0$ to $x_g$ while respecting the transition dynamics and optimizing the cost functional.

Rather than invent a direct parameterization of trajectories, we will represent a trajectory implicitly as a policy $\pi$ that maps a state $x$ to a control output $u$. Thus, we can think of the problem as learning a mapping $\Pi : \mathscr{A} \to (X \to U)$, which can be rewritten to a more traditional form: $\Pi : \mathscr{A} \times X \to U$. Given a set of example training trajectories of length $H$ for a set of $N$ different $\alpha$ values

$$(\alpha_j, \tau_j) = (\alpha_j, \{(x_t^{(j)}, u_t^{(j)})\}_{t=1}^{H}), \;\; j = 1, ..., N$$

we can use traditional supervised learning methods to find parameters $\theta$ for a family of regression functions, by constructing the training set $\{((\alpha_j, x_t^{(j)}), u_t^{(j)})\}$, and using it as input to a regression method.

Because the training data represent trajectories rather than identically and independently distributed samples from a distribution, we find that it is more effective to use a specialized form of supervised learning algorithm and an active strategy for collecting training data. The remaining parts of this section describe these methods and the way in which the final learned regressor is used to generate action in the on-line setting.

### *3.1 Representation and learning*

The key idea in TOIL is to construct a set of *local trajectory generators* $\pi_k$ and to appropriately select among them based on a two-sample test metric, MMD [17]. These trajectory generators are *local* in the sense that each of them specializes in a particular region of the space $X \times \mathscr{A}$ and can be expected to generalize well to query points that are likely to have been drawn from that same distribution of points. The final policy is a regressor that has the form $\pi(\alpha, x) = \pi_k(\alpha, x)$, where the particular $\pi_k$ is chosen based on the distance between the query point $(\alpha, x)$ and the data that were used to train $\pi_k$. We then iterate between executing the current policy and updating it with the new data. This is to mitigate the problems associated with executing a learned policy without updating it, which has been shown to accumulate error and cause cause the trajectory to be highly erroneous [10]. Pseudo-code for the top level of TOIL is shown in algorithm 1. It takes an initial state $x_0$, a goal state $x_g$

and a sample set $A = \{\alpha_1, \ldots, \alpha_N\}$ of dynamics parameters, and outputs a trajectory generator, $\Pi$, which is a mapping from the state at time $t$ and the uncertain dynamics parameter $\alpha$ to the control to be applied at time $t$.

TOIL comprises three procedures: *LearnLocalTrajGenerator*, *SelectLocalTrajGenerator* and *Optimize*. The procedure *LearnLocalTrajGenerator*$(\tau_i, \alpha_i)$ is simply a call to any supervised regression algorithm on the training set

$$\{((\alpha_i, x_t), u_t)\} \text{ for } (x_t, u_t) \in \tau_i$$

We explain the remaining procedures as we describe algorithm 1 below.

The algorithm begins by generating a set of training trajectories using the procedure *Optimize*. This procedure is responsible for getting a training trajectory, by optimizing a nonlinear program for trajectory optimization. This nonlinear program is discussed in detail in section 3.3. From each of these training trajectories, it builds a local trajectory generator and adds it to the set $\Pi$.

Once the initial training trajectories have been used to train local controllers, we begin an iterative process of ensuring coverage of the input space that will be reached by control actions generated by $\Pi$. For every value of $\alpha_i$, we try to execute the trajectory that would be generated by $\Pi$ starting from $x_0$. At each step of execution we find the local controller $\pi$ that applies to $x_t$ and $\alpha$ using the procedure *SelectLocalTrajGenerator*. This procedure, given in Algorithm 2, is responsible for selecting the most appropriate local trajectory generator based on the similarity metric, called MMD, between the current state and training data $\pi.D$ associated with each of the local controllers. This metric is described in detail in section 3.2. If there is one, we use $f_\alpha$ to simulate its execution and get a new state $x_{t+1}$. If there is no local controller that covers the pair $x_t, \alpha$, then we call the *Optimize* procedure to get a training trajectory from $x_t$ to $x_g$ with dynamics $f_{\alpha_i}$ and use it to train a new local controller, $\pi$, which we add to $\Pi$. This process is repeated until it is possible to execute trajectories for all the training $\alpha_i$ to reach the goal, without encountering any anomalous states. If the $\alpha_i$ have been chosen so that they cover the space of system dynamics that are likely to be encountered during real execution, then $\Pi$ can be relied upon to generate effective trajectories.

## *3.2 Maximum mean discrepancy*

The process of applying trajectory generator $\Pi$ to generate an actual trajectory from an initial $(x_0, \alpha)$, as well as the process of actively collecting training trajectories, depends crucially on identifying when a local trajectory generator is applicable to an observed system state. This decision is based on anomaly detection using the MMD criterion [17, 12], which is a non-parametric anomaly detection method that is straightforward to implement. Other anomaly-detection methods might also be suitable in this context, as surveyed in [30].

**Algorithm 1** TOIL($x_0, f, x_g, A$)

$\Pi = \{\ \}$
**for** $i = 1$ **to** $N$ **do**
  $\tau_i = Optimize(x_0, f, x_g, \alpha_i)$
  $\Pi = \Pi \cup LearnLocalTrajGenerator(\tau_i, \alpha_i)$
**end for**
farPtsExists = *True*
**while** farPtsExists **do**
  farPtsExists = *False*
  **for** $i = 1$ **to** $N$ **do**
    **for** $t = 0$ **to** $H$ **do**
      $\pi_t$ = *SelectLocalTrajGenerator*($\Pi, x_t, \alpha_i$)
      **if** isempty($\pi_t$) **then**
        farPtsExists = *True*
        $\tau = Optimize(x_t, f_{\alpha_i}, x_g)$
        $\pi = LearnLocalTrajGenerator(\tau, \alpha_i)$
        $\Pi = \Pi \cup \pi$
      **end if**
      Generate $x_{t+1}$ using $f_{\alpha_i}(x_t, \pi(x_t, \alpha_i))$
    **end for**
  **end for**
**end while**
**return** $\Pi$

**Algorithm 2** selectLocalTrajGenerator($\Pi, x_t, \alpha$)

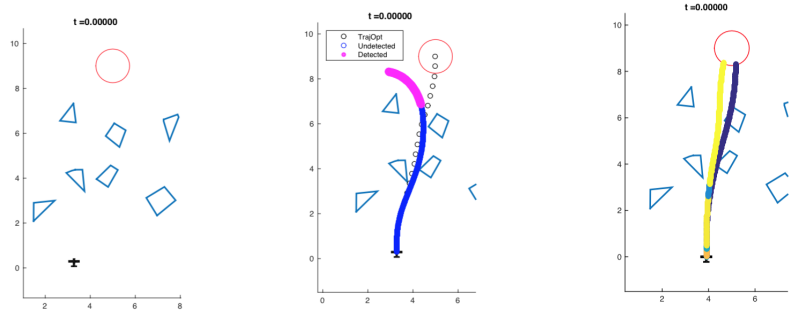
*candidates* = $\emptyset$
**for** $\pi_i \in \Pi$ **do**
  **if** MMD($\pi_i.D, (x_t, \alpha)$) < maxMMD($\pi_i.D$) **then**
    *candidates* = *candidates* $\cup\ \pi_i$
  **end if**
**end for**
**if** *size*(*candidates*) == 0 **then**
  **return** $\emptyset$
**else**
  **return** argmin$_{\hat{\pi} \in candidates}$ MMD($\hat{\pi}, (x_t, \alpha)$)
**end if**

Given two sets of data, $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ..., y_n\}$ drawn i.i.d. from distributions $p$ and $q$ respectively, the maximum mean discrepancy (MMD) criterion determines whether $p = q$ or $p \neq q$, based on an embedding of the distributions in a reproducing kernel Hilbert space (RKHS).

**Definition 1** (from [17]): *Let $\mathscr{F}$ be a class of functions $f : \mathscr{X} \to \mathbb{R}$ and let $p, q, X, Y$ be defined as above. Then MMD and its empirical estimate are defined as:*

$$\text{MMD}(\mathscr{F}, p, q) = sup_{f \in \mathscr{F}}(\mathbb{E}_{x \sim p}[f(x)] - \mathbb{E}_{y \sim q}[f(y)])$$

$$\text{MMD}(\mathscr{F}, X, Y) = sup_{f \in \mathscr{F}}(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i))$$




(a) Initial position of the aircraft (black), obstacles (blue), and the goal region (red).

(b) Training trajectory from a trajectory optimizer (TrajOpt, black), and executed trajectory of TOIL. Magenta indicates the states that are detected as anomalies during the execution, while blue indicates those that were not detected.

(c) Training traj (navy) for an observation, and a traj found by TOIL for that observation (non-navy). Colors indicate the local trajectory generators selected by TOIL. Notice that none of the states on TOIL's trajectory is navy.

Fig. 1: Airplane control

MMD comes with an important theorem which we restate here.

**Theorem** (from [17]): *Let $\mathscr{F}$ be a unit ball in a reproducing kernel Hilbert space $\mathscr{H}$, defined on compact metric space $\mathscr{X}$, with associated kernel $k(\cdot,\cdot)$. Then $MMD(\mathscr{F},p,q)=0$ if and only if $p=q$.*

Intuitively, we can expect $MMD[\mathscr{F},X,Y]$ to be small if $p=q$, and the quantity to be large if distributions are far apart. This criterion can also be used as a metric for anomaly detection, as described in [12]. Given a training dataset $D$ and a query point $x$, we can compute the following:

$$\begin{aligned}\mathrm{MMD}(\mathscr{F},x,D) &= sup_{f\in\mathscr{F}}\left(f(x)-\frac{1}{n}\sum_{x'\in D_i}f(x'))\right)\\ &= \left(k(x,x)-\frac{2}{n}\sum_{x'\in D_i}k(x,x')+\frac{1}{n^2}\sum_{x',x''\in D_i}k(x',x'')\right)^{\frac{1}{2}} \qquad (1)\end{aligned}$$

and define $\mathrm{maxMMD}(D)=\max_{x\in D}\mathrm{MMD}(\mathscr{F},x,D)$. As illustrated in [12], we report $x$ as an anomaly for dataset $D$ if $\mathrm{MMD}(x,D)>\mathrm{maxMMD}(D)$.

Figure 1b shows the result of anomaly detection using the MMD criterion in one of our domains, in which the robot needs to steer to the goal from a given initial position and a forward velocity for the robot.

**Algorithm 3** TOILEx($x_0, \Pi, \alpha$)

**for** $t = 0$ **to** $H$ **do**
  $\pi$ = *SelectLocalTrajGenerator*($\Pi, x_t, \alpha$)
  **execute** $\pi(x_t, \alpha)$
  $x_{t+1}$ = **ObserveState**
**end for**

### *3.3 Trajectory Optimization*

In trajectory optimization, the goal is to produce a locally optimal open-loop trajectory that minimizes a cost function along this trajectory, for a given initial condition $x_0$. The problem of finding an optimal trajectory can be formulated generically as:

$$\begin{aligned} u^*(\cdot) = \underset{u(\cdot)}{\operatorname{argmin}} J(x_0;\alpha) = \underset{u(\cdot)}{\operatorname{argmin}} \int_{t=0}^{T} g(x_t,u_t)\,dt \\ s.t. \quad \dot{x}_t = f_\alpha(x_t,u_t) \quad \forall\, t \quad \textbf{and} \quad x_T = x_g \end{aligned} \tag{2}$$

where, $x_t$ and $u_t$ respectively represent the state and the control input of the system at time $t$, $g(x_t,u_t)$ is the cost function, $\dot{x}_t = f_\alpha(x_t,u_t)$ governs the dynamics of the system, $x_0$ is the initial state and $x_g$ is the goal state.

There are multiple way of solving nonlinear programs of this form [13], any of which would be appropriate for use with TOIL.

### *3.4 Online execution*

Algorithm 3 illustrates the use of a learned $\Pi$ in an on-line control situation. We assume that, at execution time, the parameter $\alpha$ is observable. At each time point, we find the local controller that is appropriate for the current state and $\alpha$, execute the $u$ that it generates, and then observe the next state. Assuming we have $K$ local controllers, each of which is trained with $H$ data points, the worst-case time complexity for computing a control for a given state and model is then $O(HK)$. This can be achieved by storing the Gram matrix of the dataset (i.e. the third part of Eqn. 1) in a database, in which case the computation of the second part of Eqn 1 becomes the dominant term.

## 4 Partially observable dynamics

In more realistic situations, the exact value of $\alpha$ will not be observable online. Instead, we will be able to make observations $o$, which allow computation of a posterior distribution $\Pr(\alpha \mid o)$. The TOIL approach can be generalized directly to this

setting, but rather than selecting the $u_t$ to minimize $J(x_0, \alpha)$ we minimize it in expectation, hoping to obtain a trajectory that "hedges its bets" and performs reasonably well in expectation over system dynamics $f_\alpha$ where $\alpha \sim \Pr(\alpha \mid o)$.

In practice, we use a sampled approximation of the expected value; in particular, we assume that we have $N$ $\alpha$s drawn from $P(\alpha \mid o)$, and we formulate the constrained optimization problem as

$$\begin{aligned} u^*(\cdot) = \underset{u(\cdot)}{\operatorname{argmin}}\, \mathbb{E}_{\alpha \sim P(\alpha|o)}\left[J(x_0;\alpha)\right] \approx \underset{u(\cdot)}{\operatorname{argmin}}\, \frac{1}{N}\sum_{i=1}^{N}\int_{t=0}^{T} g(x_t^i, u_t)\, dt \\ s.t. \quad x_t^i = f_{\alpha_i}(x_t^i, u_t) \quad \forall\, t, i \quad \textbf{and} \quad x_T^i = x_g \quad \forall\, i \end{aligned} \tag{3}$$

Note that there are different state variables $x_t^i$ for each possible dynamics $\alpha_i$, allowing the trajectories to be different, but that there is a single sequence of control variables $u_t$.

The only additional change to the TOIL algorithm is that, instead of conditioning on $\alpha$ in the supervised learning and selection of local trajectory generators, we condition on the observation $o$.

## 5 Experiments

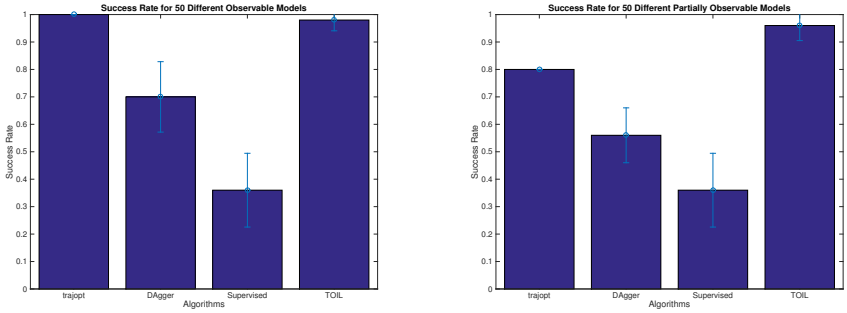
We evaluate our framework on two domains: aircraft path finding and robot manipulation. In all of our experiments, we use random forests [16] as our supervised learner. Specifically, we use the TreeBagger class implemented in MATLAB for the aircraft task, and RandomForestRegressor class implemented in scikit-learn [18] for the manipulation task. We used a Gaussian kernel for the *MMD* metric in both tasks.

To evaluate TOIL, we compute three different measures: success rate, trajectory generation time, and training time. The success rate is the percentage of the time the trajectory generated by TOIL satisfied the constraints of the environment and reached the goal. Trajectory generation time shows how much TOIL decreases the online computation burden and training time measures the off-line computation time required to learn $\Pi$ for a new domain.

We compared TOIL to three different benchmarks: (1) calling a standard trajectory optimization procedure (solving equation 2 using snopt[28]) online in each new task instance, (2) using the initial training trajectories as input to a random forest supervised learning algorithm; and (3) DAgger, which calls trajectory optimization at every time step.

### *5.1 Airplane Control*

This task is to cause an airplane traveling at a constant speed in the plane to avoid obstacles and reach a goal location by controlling its angular acceleration. Figure 1a shows an instance of this task. System states $s_t$ and controls $u_t$ are defined to be:




(a) Observable Model

(b) Partially Observable Model

Fig. 2: Success Rates for Airplane Domain

$$s_t = \left[x_t, y_t, \theta_t, \dot{\theta}_t\right]^T, \quad u_t = \ddot{\theta}_t$$

where $(x,y)$ is the location of the airplane in the 2D plane, $\theta$ is the heading angle, and $\dot{\theta}$ and $\ddot{\theta}$ are the angular velocity and acceleration, respectively. The dynamics of the system is given by:

$$f(x_t, u_t) = \left[\dot{x}_t, \dot{y}_t, \dot{\theta}_t, \ddot{\theta}_t\right]^T = \left[-v \cdot sin(\theta_t), v \cdot cos(\theta_t), \ddot{\theta}_t, u_t\right]^T$$

where $v$ denotes the constant speed of the airplane. The objective function is integrated cost $g(s_t, u_t) = u_t^2 - distToNearestObstacle(s_t)$.

We consider a trajectory to be a "success" if it does not collide with any obstacles, and arrives at the goal.

**Observable Case** The aspect of the dynamics that is variable, corresponding to $\alpha$ in the algorithms, is the speed of the aircraft, $v$; it is correctly observed by the system at the execution time. For training, we sampled 30 different $\alpha$ values from $P(\alpha) = \text{Uniform}(5, 30)$, and then generated training trajectories by solving Eqn 2. For testing, we sampled 50 different $\alpha$ values from $P(\alpha)$.

Figure 2a shows the success rates of the different algorithms. Trajectory optimization always returns a trajectory that is able to arrive at the goal without a collision. DAgger frequently failed to find feasible trajectories, mainly because it has not sampled training trajectories in the relevant parts of the state space. Simple supervised learning was even less successful due to its inability to sample any extra trajectories at training time. TOIL, in comparison, is able to generate trajectories whose performance is almost on par with the trajectory optimizer.

Table 1 shows the online trajectory computation time required by each algorithm for 21 knot points. All of the learning-based methods significantly reduce the online computation time compared to the online trajectory optimization. In terms of training time, we can see that DAgger makes a very large number of calls to the trajectory optimizer, collecting training data at inappropriate regions of state space.

| Algorithm | Trajectory Generation Time | Training Time | Number of Traj Opt Calls |
|---|---|---|---|
| TOIL | 2.73 secs | 64.53 mins | 32 |
| DAgger | 2.10 secs | 375.01 mins | 186 |
| Supervised | 1.57 secs | 60.50 mins | 30 |
| Traj Opt | 121.31 secs | 0 mins | 0 |

Table 1: Observable airplane domain trajectory generation and training times

In contrast, TOIL is able to ask only when necessary, achieving much faster training time. Overall, then, TOIL produces very good trajectories with reasonable time requirements for both training and testing.

**Partially Observable Case** For the partially observable case, the observation $o$ is the mass of the aircraft, $m$. We assume a Gaussian distribution, $P(\alpha \mid o = m) = \mathcal{N}(\frac{1}{m}, 1)$, where $m$ is the mass of the aircraft. During training we pick 30 different $m$ values, and for each $m$ we sample 5 different $\alpha$ values from $P(\alpha \mid o = m)$ and solve Eqn 3 to produce 30 training trajectories, each of which is intended to be robust to varying $\alpha$ values.

For testing, we sampled 50 different new $m$ values, and for each, sampled 5 $\alpha$ values from $P(\alpha \mid o = m)$. In this phase, the robot only sees $m$, but not $\alpha$ or $P(\alpha|o)$. For online trajectory optimization, we computed a trajectory using those 5 $\alpha$ samples by solving Eqn 3. Then, for all learning algorithms, we report the result of evaluating the trajectory they produce on one of the five sampled $\alpha$ values.

Figure 2b shows the success rate of different techniques in these problems. In contrast to the observable case, the robust trajectory optimization cannot always find trajectories that succeed. This is because the non-linear optimizer needs to find a trajectory that is feasible for all 5 sampled $\alpha$ values, which makes the optimization problem much more difficult.

TOIL performs much better than trajectory optimization, which sometimes gets stuck in terrible local optima, such as one that collides with obstacles. TOIL is more robust because the MMD criterion is able to pick appropriate local trajectory generators depending on the initial state. In this task, most of the training trajectories have heading angles facing forward and travel through the middle of the field due to the placement of obstacles. Therefore, it is unlikely for a local trajectory generator whose training data includes traveling towards obstacles to be selected, because $x_0$ is located at the middle of field, facing forward. This is well illustrated in Figure 1c. Here, we can see that the local trajectory generator trained with data that collides with obstacles by traveling to right is never selected during execution.

Now we consider the trajectory generation time and training time, as shown in table 2. As the generation times show, the learning methods reduce the online computation time significantly compared to online optimization. TOIL's training time was again significantly smaller than DAgger's, and comparable with that of supervised learning.

| Algorithm | Trajectory Generation Time | Training Time | Number of Traj Opt Calls |
|---|---|---|---|
| TOIL | 8.40 secs | 273 mins | 37 |
| DAgger | 7.84 secs | 1720.30 mins | 233 |
| Supervised | 1.75 secs | 221.50 mins | 30 |
| Traj Opt | 443.12 secs | 0 mins | 0 |

Table 2: Partially observable airplane trajectory generation and training times

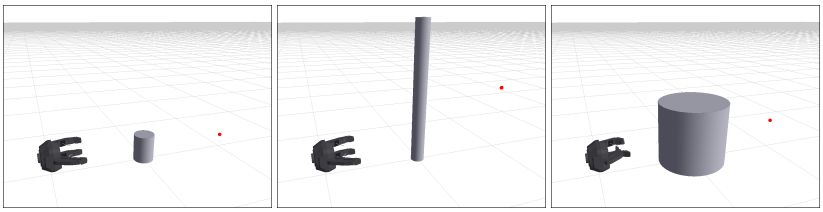


Fig. 3: Examples of the cylinders used in testing. The robotic hand is at its initial position, and the red dot indicates the goal location for the center of the cylinder

## *5.2 Manipulation Control*

In this domain, the task is to move a cylindrical object with a multi-fingered robot hand from an initial position to a goal position. A state of the robot is an element of an 84 dimensional space which consists of: position and orientation of the palm and the cylinder, as well as poses of the other 8 links relative to palm, $q$; associated velocities, $\dot{q}$, and accelerations $\ddot{q}$. We make use of an augmented position controller whose inputs are desired poses and an amount of time that should be taken to achieve them: $u_t = (q_{t+1}, dt_{t+1})$. The aspect of the dynamics that varies is $\alpha = (C_x, C_y, C_z)$, the center of mass of the cylinder, and in the partially observable case, the observation $o = (r, l)$ is the radius and length of the cylinder. We declare a trajectory to be successful if it does not violate the dynamics constraints and moves the cylinder to a desired goal pose. Figure 3 shows some example cylinders used in the testing phase.

The nonlinear program for trajectory optimization has the following components: decision variables $\left\{q_t, \dot{q}_t, \ddot{q}_t, F_t^{(1)}, F_t^{(2)}, F_t^{(3)}\right\}_{t=0}^{T}$ where $F^{(i)}$ is the force exerted by $i^{th}$ finger at a contact point; objective function $g(x_t, u_t) = \alpha \dot{q}_t^2 + \beta \ddot{q}_t^2 + \gamma \sum_{i=1}^{3} {F_t^{(i)}}^2$ where $\alpha, \beta, \gamma \in \mathbb{R}$. are weights on each component; constraints between $x_t$, $u_t$ and $x_{t+1}$ that enforce the physics of the world; constraints on final and initial states; finger-tip contact constraints that require the robot to contact the object with its finger tips only; friction cone constraints between robot and cylinder, and cylinder and the surface; complementarity constraints of the form $F^{(f)} \cdot d(q_t) = 0, \quad \forall f, q_t$, where, $F^{(f)}$ denotes the force being exerted by finger $f$, $q_t$ denotes the location of the hand at time $t$, and $d(q_t)$ denotes the distance from the object to the finger $f$ at time $t$. Our implementation of the physics constraints embodies several approximations to

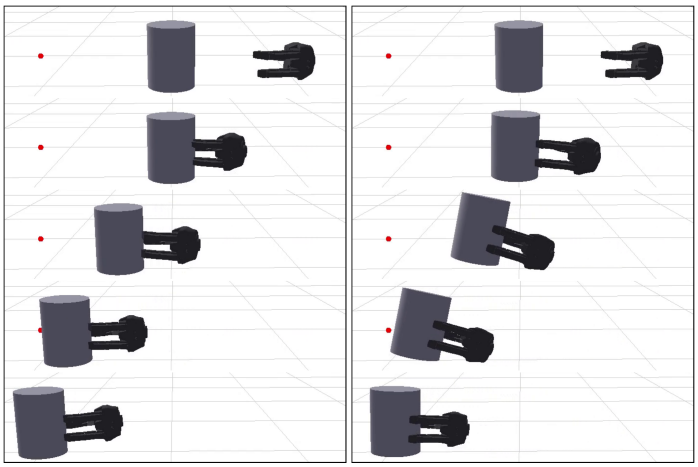


Fig. 4: Trajectories for the observable (left) and partially observable case (right). For the observable case, the robot simply pushes the object to the goal. For the partially observable case, the robot lifts the object to to the goal, as to minimize the risk of tipping the cylinder over.

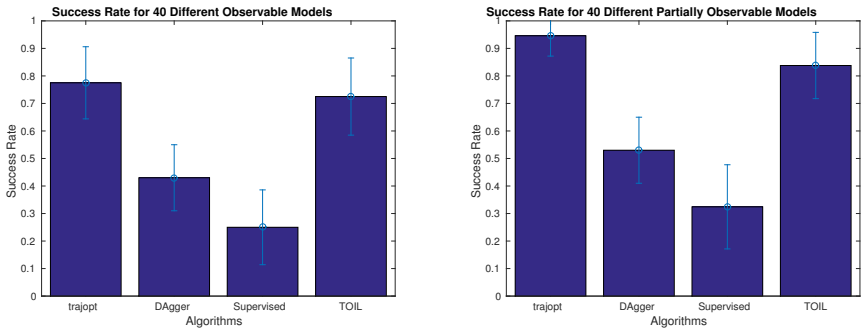
real world physics. However, this still represents a challenging test for the learning methods.

Intuitively, given the objective function and the constraints, the optimal behavior is to simply push the object to the goal, because pushing requires the robot to exert less total force and move along a shorter trajectory than lifting the object. However, when there is uncertainty about the center of mass, pushing the object may be risky: if the height at which it pushes is too far above or below the COM, the cylinder may tip over, and so picking the object up may be preferable, in expectation. We find that when the system dynamics are observable, TOIL selects appropriate pushing trajectories, but when they are only partially observable, TOIL makes more robust choices; an example is illustrated in Figure 4 and a few more examples are shown in the video[2]

**Observable Case** In the observable case, we directly observe $\alpha$. For training, we sample 40 different $\alpha$'s from $P(\alpha = (C_x, C_y, C_z))$, which is defined as a joint uniform distribution with its range defined by the length and radius of the cylinder. For testing, we sample 50 different models from the same distribution.

Figure 5a shows the success rate of the same set of algorithms as for the airplane domain. As the figure shows, even trajectory optimization sometimes fails to satisfy the constraints within the given time limit for optimization, because the problem is quite large. While TOIL again performed just slightly worse than trajectory optimization, DAgger and supervised learning performed relatively poorly. Table 3 shows the training and trajectory computation times. The learning approaches are

[2] `https://www.youtube.com/watch?v=r9o0pUIXV6w`




(a) Observable Model (b) Partially Observable Model

Fig. 5: Success Rates for Manipulation Domain

| Algorithm | Trajectory Generation Time | Training Time | Number of Traj Opt Calls |
|---|---|---|---|
| TOIL | 1.10 secs | 1012 mins | 44 |
| DAgger | 1.04 secs | 1320 mins | 60 |
| Supervised | 0.45 secs | 880 mins | 40 |
| Traj Opt | 1302 secs | 0 mins | 0 |

Table 3: Trajectory computation times and training times for various algorithms

much more efficient at generating trajectories online than the optimizer is; Again, DAgger makes extra calls to the trajectory optimizer, while the supervised learner makes too few.

**Partially Observable Case** In a more realistic scenario, the robot only gets to observe length and radius of the cylinder, but not the exact center of mass. For training, we sampled 40 different observations, and sampled two different $\alpha$'s from the conditional distribution $P(\alpha = [C_x, C_y, C_z] | \mathbf{o} = [r, l])$, which is defined as a joint normal distribution centered at the center of the cylinder (i.e. $\mu_\alpha = (r_x, r_y, l/2)$) and variance defined as half of radius for $(x, y)$ direction, and length in $z$ direction. For testing, we sampled 50 different observations and tested the algorithms on one of the $\alpha$'s sampled from the same distribution. The robot gets to see only $\mathbf{o}$, but not the conditional distribution or $\alpha$.

Figure 5b shows the success rate of the different algorithms. The pattern of performance is similar to the observable case. All the algorithms performed somewhat better than in the observable case, presumably because the trajectories found are less sensitive to variations in the dynamics. Table 4 shows the training and trajectory generation times. For this case, the learning algorithms are even more efficient relative to trajectory optimization, because the optimization problem is so difficult. As before, DAgger gathered much more data, while TOIL collected just enough to perform almost as well as the trajectory optimizer.

| Algorithm | Trajectory Generation Time | Training Time | Number of Traj Opt Calls |
|---|---|---|---|
| TOIL | 1.16 secs | 1978 mins | 46 |
| DAgger | 1.04 secs | 2580 mins | 60 |
| Supervised | 0.45 secs | 1720 mins | 40 |
| Traj Opt | 2628 secs | 0 mins | 0 |

Table 4: Trajectory computation times and training times of various algorithms

## 6 Conclusion

We proposed TOIL, an algorithm that learns an online trajectory generator that can generalize over varying and uncertain dynamics. When the dynamics is certain, our generator is able to generalize across various model parameters. If it is partially observable, then it is able to generalize across different observations. It is shown, in two simulated domains, to find solutions that are nearly as good as, and sometimes better than, those obtained by calling the trajectory optimizer on line. The online execution time is dramatically decreased, and the off-line training time is reasonable.

A significant concern about TOIL, as well as other supervised learning based algorithms for trajectory generation [3, 4, 9], is that the resulting controller has no guarantee of stability. In contrast, controllers synthesized from a set of local stabilizing controllers, such as LQRs, can guarantee that the controller would stabilize to the goal state [2]. Investigating the stability guarantees of supervised learning based trajectory generators would be an interesting research avenue for the future.

## 7 Acknowledgment

This work was supported in part by the NSF (grant 1420927). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from the ONR (grant N00014-14-1-0486 ), from the AFOSR (grant FA23861014135), and from the ARO (grant W911NF1410433).

## References

1. C. Atkeson, Using local trajectory optimizers to speed up global optimization in dynamic programming, In Neural Information Processing Systems, 1994.
2. R. Tedrake, LQR-Trees: Feedback motion planning via sums of squares verification, In International Journal of Robotics Research, 2010.
3. S. Levine and V. Koltun, Guided policy search, In International Conference on Machine Learning, 2013.
4. S. Levine and V. Koltun, Learning complex neural network policies with trajectory optimization, In International Conference on Machine Learning, 2014.
5. S. Levine N. Wagener, P. Abbeel, Learning contact-rich manipulator skills with guided policy search, In International Conference on Automation and Control, 2015.

6. A. D. Marchese, R. Tedrake, and D. Rus, Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator, In International Conference on Automation and Control, 2015.
7. H. Dai, A. Valenzuela, R. Tedrake, Whole-body motion planning with centroidal dynamics and full kinematics, In International Conference on Humanoid Robots, 2014.
8. M. Posa, C. Cantu, R. Tedrake, A direct method for trajectory optimization of rigid bodies through contact, In International Journal of Robotics Research, 2014.
9. I. Mordatch and E. Todorov, Combining the benefits of function approximation and trajectory optimization, in Robotics:Science and Systems 2014.
10. S.Ross, G. J. Gordon, and J. A. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in International Conference on Artificial Intelligence and Statistics 2011.
11. B.Kim and J. Pineau, Maximum mean discrepancy imitation learning, in Robotics:Science and Systems 2013.
12. N. Cristianini and J. Shawe-Taylor, Kernel methods for pattern analysis, Cambridge University Press, 2004.
13. J. Betts, Practical methods for optimal control using nonlinear programming, SIAM Advances in Design and Control, 2001.
14. O.V. Stryk, R. Bulirsch, Direct and indirect methods for trajectory optimization, Annals of Operations Research, 1992.
15. P. T. Boggs and J. W. Tolle, Sequential quadratic programming, Acta Numerica, 1995.
16. L. Breiman, Random forests, Machine Learning, 2001.
17. A. Gretton, K. Borgwardt, M. Rasch, B. Schlkopf, and A. Smola, A kernel method for the two sample problem, In Neural Information Processing Systems, 2007.
18. Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., Scikit-learn: machine learning in Python, In Journal of Machine Learning Research, 2011.
19. R. Tedrake, Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2014, `http://drake.mit.edu`.
20. C. Atkeson, C. Liu., Trajectory-based dynamic programming, In Modeling, Simulation, and Optimization of Bipedal Walking, 2013.
21. B. Argall, S. Chernova, M Veloso, B. Browning., A survey of robot learning from demonstration, In Robotics and Autonomous Systems, 2009.
22. P. Abbeel, A. Coates, and A. Y. Ng., Autonomous helicopter aerobatics through apprenticeship learning, in International Journal of Robotics Research, 2010.
23. S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert., Learning monocular reactive UAV control in cluttered natural environments. In International Conference on Robotics and Automation 2013.
24. 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 International Conference on Robotics and Automation 2010.
25. J. A. Bagnell., An invitation to imitation, In Tech Report CMU-RI-TR-15-08, Robotics Institute, Carnegie Mellon University, 2015.
26. H. Daume, J. Langford, D. Marcu., Search-based structured prediction, In Machine Learning Journal, 2009.
27. J. T. Betts., Survey of numerical methods for trajectory optimization, In Journal of Guidance, Control, and Dynamics, 1998.
28. P. E. Gill, W. Murray, and M. A. Saunders., Snopt: An sqp algorithm for large-scale constrained optimization, In SIAM Journal on optimization, 2002.
29. T. J. Perkins and A. G. Barto., Lyapunov design for safe reinforcement learning, Journal of Machine Learning Research, 2002.
30. V. Chandola, A. Banerjee, V. Kumar., Anomaly detection: a survey, In ACM Computing Surveys, 2009.