Self-Supervised Robot Learning

Dhiraj Prakashchand Gandhi

CMU-RI-TR-19-36

July 2, 2019

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Abhinav Gupta, *chair*
Martial Hebert
Lerrel Pinto

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

Supervised learning has been used in robotics to solve various tasks like navigation, fine manipulation, etc. While it has shown a promising result, in most cases the supervision comes from the human agent. However, relying on human is a huge bottleneck to scale up these approaches. In this thesis, we try to take the human out of the loop and try to solve the task in a self-supervised manner. More specifically, we have applied self-supervised learning to 2 tasks, for drone navigation and exploration in reinforcement learning. In drone navigation, drone first collides with lots of objects and based on it, it learns a policy to avoid them. In reinforcement learning, we try to learn exploration policy in a self-supervised manner for both stochastic and deterministic environment.

How do you learn to navigate an Unmanned Aerial Vehicle (UAV) and avoid obstacles? One approach is to use a small dataset collected by human experts: however, high capacity learning algorithms tend to overfit when trained with little data. An alternative is to use simulation. But the gap between simulation and real world remains large especially for perception problems. The reason most research avoids using large-scale real data is the fear of crashes! In this work, we propose to bite the bullet and collect a dataset of crashes itself! We build a drone whose sole purpose is to crash into objects: it samples naive trajectories and crashes into random objects. We crash our drone 11,500 times to create one of the biggest UAV crash dataset. This dataset captures the different ways in which a UAV can crash. We use all this negative flying data in conjunction with positive data sampled from the same trajectories to learn a simple yet powerful policy for UAV navigation. We show that this simple self-supervised model is quite effective in navigating the UAV even in extremely cluttered environments with dynamic obstacles including humans.

Exploration has been a long standing problem in both model-based and model-free learning methods for sensorimotor control. There have been major advances in recent years demonstrated in noise-free, non-stochastic domains such as video games and simulation. However, most of the current formulations get stuck when there are stochastic dynamics. In this work, we propose a formulation for exploration inspired from the work in active learning literature. Specifically, we train an ensemble of dynamics models and incentivize the agent to maximize the disagreement or variance of those ensembles. We show that this formulation works as well as other formulations in non-stochastic scenarios, and is able to explore better in scenarios with stochastic-dynamics. Further, we show that this objective can be leveraged to perform differentiable policy optimization. This leads to a sample efficient exploration policy. We show experiments on a large number of standard environments to demonstrate the efficacy of this approach. Furthermore, we implement our exploration algorithm on a real robot which learns to interact with objects completely from scratch.

Acknowledgments

I would like to thank my advisor, Professor Abhinav Gupta, for all his support throughout my masters program. I am grateful for having many helpful discussions with him and his unique insights had deep impact on my research. I am also very thankful for Lerrel Pinto, Adithya Murali and Deepak Pathaks help and advice on my projects. Futhermore, I would like to thank all my lab colleagues and friends for the help with the research and life.

Contents

List of Figures

List of Tables

Chapter 1

Self-Supervised Drone Navigation Policy

1.1 Introduction

How do you navigate an autonomous system and avoid obstacles? What is the right approach and data to learn how to navigate? Should we use an end-to-end approach or should there be intermediate representations such as 3D? These are some of the fundamental questions that needs to be answered for solving the indoor navigation of unmanned air vehicle (UAV) and other autonomous systems. Most early research focused on a two-step approach: the first step being perception where underlying occupancy/obstacle map is estimated; the second step is to use the predicted depth map to issue motor commands to travel in freespace [52], [65], [26]. While the two step-approach seems reasonable, the cost of sensors and unrecoverable errors from perception make it infeasible.

Another alternative is to use a monocular camera and learn to predict the motor commands. But how should we learn the mapping from input images to the motor commands? What should be the right data to learn this mapping? One possibility is to use imitation learning [49]. In imitation learning setting, we have a user who provides trajectories to train the flying policy. In order to visit the states not sampled by expert trajectories, they use the learned policy with human corrective actions to train the policy in iterative manner. But learning in such scenarios is restricted to small datasets since human experts are the bottlenecks in providing the training data. Therefore, such approaches cannot exploit high-capacity learning algorithms to train their policies.

Recently, there has been a growing interest in using self-supervised learning for variety of tasks like navigation [68], grasping [46] and pushing/poking [3]. Can we use self-supervised learning to remove the labeling bottleneck of imitation learning? But how do we collect data for self-supervised learning? In contemporary work, Sadeghi and Levine [51] use Reinforcement Learning (RL) in simulation to train the navigation policy of the drone. They focus on using simulations to avoid collisions that are inevitable since RL-techniques involve a trial-and-error component. They also demonstrate how a policy learned in simulation can transfer to real world without any retraining. But is it really true that simulation-based training can work out of box in real world? Most approaches in computer vision suggest otherwise and require small amounts of real-world data for adaptation. We note that the testing scenarios in [51] consist mostly of empty corridors where perspective cues are sufficient for navigation [7]. These perspective cues are captured in simulation as well. However, when it comes to navigation in cluttered environment (such as one shown in figure ??), the gap between real and simulation widens dramatically. So, how can we collect data for self-supervised learning in real-world itself? Most approaches avoid self-supervised learning in real-world due to fear of collisions. Instead of finding ways to avoid collisions and crashes, we propose to **bite the bullet** and collect a dataset of crashes itself! We modified the publicly available Parrot Ar-Drone 2.0 and developed a software to enable it to crash autonomously into obstacles and recover from the impact. This enabled us to collect data for around 11,500 crashes. This dataset represents the different ways in which UAV can collide with obstacles. It also encodes the information about how a drone should not fly. Based on how far (in terms of time) data was collected from the time of collision, we annotated the data into a positive and a negative class. Using this self-annotated data we learned a simple binary classifier network which a given image tries to predict whether the drone should move forward or not. We show that this simple self-supervised paradigm is quite effective in navigating the UAV even in extremely cluttered environments with dynamic obstacles like humans.

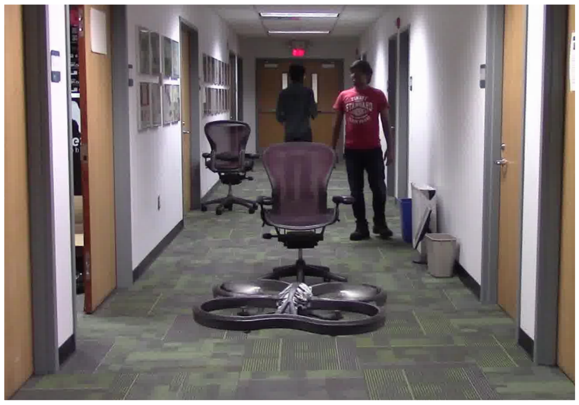


Figure 1.1: In this paper, we focus on learning how to navigate an UAV system. Specifically, we focus on indoor cluttered environment for flying. Instead of using small datasets of imitation learning or performing navigation via intermediate representations such as depth; we collect a large-scale dataset of drone crashes. This dataset acts as a negative instruction and teaches the drone how NOT to crash. We show our simple learning strategy outperforms competitive approaches using the power of large data.

1.2 Related work

This work, which combines self supervised learning with flying a drone in an indoor environment, touches upon the broad fields of robot learning and UAV control. We briefly describe these works and their connections to our method.

1.2.1 UAV control

Controlling UAVs has been a widely studied area motivated by applications in surveillance and transportation. The most prevalent approaches make use of onboard range sensors to fly autonomously while avoiding obstacles [9, 48, 52]. This is however not practical for publicly available drones that often have low battery life and low load carrying capacity. One can also use stereo vision based estimation [2, 18] with light and cheap cameras. However stereo matching often fails on plain surfaces like the white walls of an office.

Most of the methods described so far use multiple sensors to decide control parameters for an UAV. This often leads to higher cost, poor realtime response and bulkier systems. [50] used cheap, light optical flow sensors to control an UAV in indoor setting, but was not able to achieve full autonomy without human intervention. [6] showed promising use of optical flow for full autonomy. However results have only shown on limited outdoor environment (which are not as cluttered as the one we face in indoor setting). Monocular camera based methods [7] use vanishing points as a guidance for flying drone in indoor environment, but still rely on range sensors for collision avoidance.

A more recent trend of approaches has been in using learning based methods to infer flying control for the UAVs. Researchers [49] have used imitation learning strategies to transfer human demonstrations to autonomous navigation. This however fails to collect any negative examples, since humans never crash drones into avoidable obstacles. Because of this data bias, these methods fail to generalize to trajectories outside the training demonstrations from human controllers. [30] has shown successful onboard implementation of self-supervised depth estimator from images to control miniature UAV. However results are shown in single, small indoor environment.

Another recent idea is to use simulators to generate this drone data [51, 58]. However transferring simulator learned policies to the real world works well only in simplistic scenarios and often require additional training on real-world data.

1.2.2 Deep learning for robots

Learning from trial and error has regained focus in robotics. Self supervised methods [3, 33, 45, 46], show how large scale data collection in the real world can be used to learn tasks like grasping and pushing objects in a tabletop environment. Our work extends this idea of self supervision to flying a drone in an indoor environment. Deep reinforcement learning methods [39] have shown impressive results, however they are too data intensive (order of million examples) for our task of drone flying.

A key component of deep learning, is the high amount of data required to train these generalizable models [22, 29]. This is where self-supervised learning comes into the picture by allowing the collection of high amounts of data with minimal human supervision. To the best of our knowledge, this is the first large scale effort in collecting more than 40 hours of real drone flight time data which we show is crucial in learning to fly.

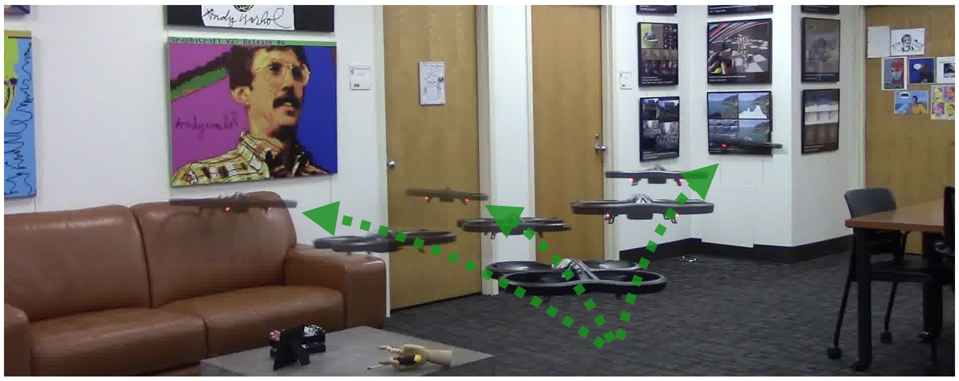


Figure 1.2: We randomly hit objects with our drone more than 11,500 times over a diverse range of environments, This collection of collision trajectories is performed completely autonomously.

1.3 Approach

We now describe details of our data driven flying approach with discussions on methodologies for data collection and learning. We further describe our hardware setup and implementation for reproducibility.

1.3.1 Hardware Specifications:

A important goal of our method is to demonstrate the effectiveness of low cost systems for the complex task of flying in an indoor environment. For this purpose, we use the Parrot Ar-Drone 2.0 which, due to its inaccuracies, is often run in outdoor environments as a hobby drone. We attach no additional sensors/cameras in the flying space in the data collection process. The key required components for the drone is it's inbuilt camera which broadcast 720p resolution images at 30 hz, it's inbuilt accelerometer and a safety hull to collide with objects without damaging the rotors. We externally attach a small camera of the same specification as of drones inbuilt camera to aid in localization during data collection.

1.3.2 Data Collection:

We use a two-step procedure for data collection. First, we sample naive trajectories that lead to collisions with different kind of objects. Based on these sampled we then

Algorithm 1 Data Collection

```
1: Init: Track position error using PTAM from intial Take Off location
2: while No Crash do
3:     Choose a random direction
4:     while No Collision do
5:         Continue in the same direction
6:     end while
7:     while Position error > $\epsilon$ do
8:         calculate control command based on position error
9:     end while
10: end while
```

learn a policy for navigation. These initial naive trajectories and collisions provide a good initialization for reward leading to sampling more complex trajectories. This policy is then used to collect more example akin to hard example mining approach. At this stage, we can obtain much better trajectories that only collide when the learned strategy fails. This sampling of hard negatives has been shown to improve performance [46, 59].

Collecting collision data

Most methods for learning with drones [7, 49] often have very few examples of colliding with objects. Our focus is however to collect as much of collision information as possible. This large amount of crashing data should teach our learning model how not to crash/collide with objects.

But how should we collect this collisions data? One way would be to manually control the drone and crash into objects. But this would severely bias the dataset collected due to human intuition. Another way to collect this data is by commanding the drone to autonomously navigate the environment by SLAM based approaches [40, 64] and collect failure cases. But we again have dataset bias issues due to failure modes of SLAM along with sparse collision data. What we need is a method that has low bias to objects it collides with and can also generate lots of collision data. Our proposed method for data collection involves naive random straight line trajectories described in Figure 1.2.

Algorithm 1 describes our method of data collection in more detail. The drone is first placed randomly in the environment we desire to collect collision data. The drone then takes off, randomly decides a direction of motion and is commanded to follow a straight line path until collision. After a collision, the drone is commanded to go back to its original position followed by choosing another random direction of motion. This process is continued until the drone cannot recover itself from a collision. After N_{env_trial} crashes, the drone collision data collection is restarted in a new environment. For each trajectory, we store time stamped images from the camera, estimated trajectories from IMU readings and accelerometer data. The accelerometer data is used to identify the exact moments of collision. Using this, we create our dataset $D = \{d_i\}$ where $d_i = \{I_t^i\}_{t=0}^{N_i}$. For the i^{th} trajectory d_i, image I_0^i is the image at the beginning of the trajectory far away from the collision object, while image $I_{N_i}^i$ is the image at the end of the trajectory after N_i timesteps when the drone hits an object.

A key component of this data collection strategy is the ability of the drone to come back to its initial position to collect the next datapoint. However due to cheap low accuracy IMU, it isn't possible to do accurate backtracking using these naive sensors. For this purpose, we use PTAM [27] module that localizes the robot and helps it backtrack.

We collect 11,500 trajectories of collisions in 20 diverse indoor environments (Figure 1.3). This data is collected over 40 drone flying hours. Note that since the hulls of the drone are cheap and easy to replace, the cost of catastrophic failure is negligible.

Data processing

We now describe the annotation procedure for the collected trajectories. The trajectories are first segmented automatically using the accelerometer data. This step restricts each trajectory upto the time of collision. As a next step, we further need to segment the trajectory into positive and negative data i.e. $d_i = d_i^+ \bigcup d_i^-$. Here d_i^+ is the part of the trajectory far away from the collision object while d_i^- is the part of the trajectory close to the colliding object. Note the positive part of the dataset correspond to images where the control signal should be to continue forward. This segmentation is done heuristically by splitting the first N_+ timesteps of the trajectory

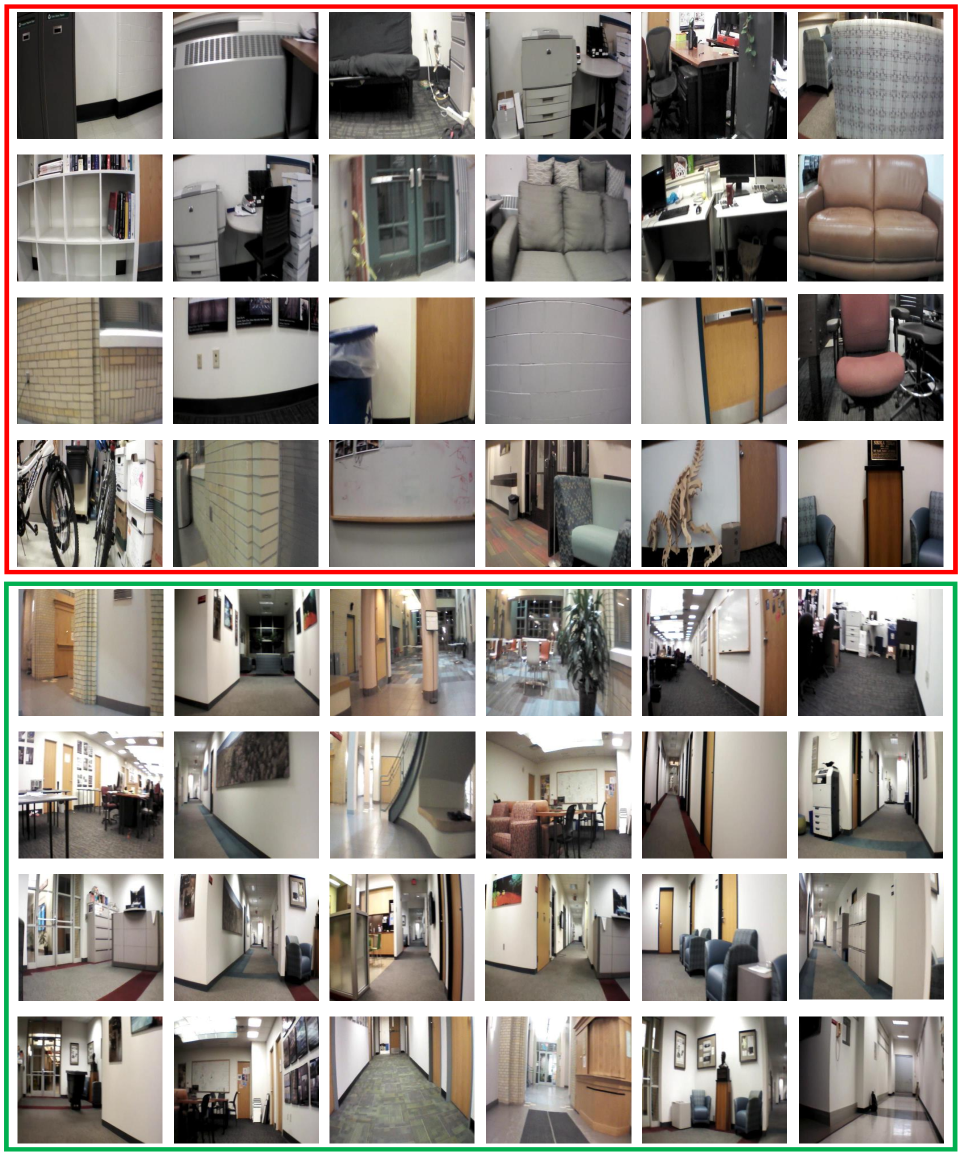


Figure 1.3: Given the data collected from drone collisions, we can extract portions of trajectories into two sections; first one very close to the objects (red box) & second far away from the objects (green box).

as positive and the last N_{-} timesteps as negative trajectories. We ignore the images in the middle part of the trajectory. We now have a dataset with binary classification labels.

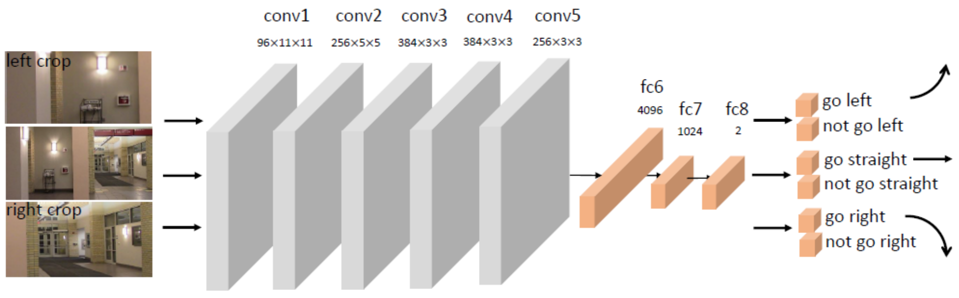


Figure 1.4: We employ deep neural networks to learn how to fly. The convolutional weights of our network (in grey) are pretrained from ImageNet classification [29], while the fully connected weights (in orange) is initialized randomly and learnt entirely from the collision data. At test-time, crops of the image are given as input, and the network outputs the probability of taking control actions.

1.3.3 Learning Methodology

We now describe our learning methodology given the binary classification dataset we have collected. We will first describe our learning architecture and follow it with description of test time execution.

Network architecture

Given the recent successes of deep networks in learning from visual inputs, we employ them to learn controls to fly. We use the AlexNet architecture [29]. We use ImageNet-pretrained weights as initialization for our network [22]. This network architecture is represented in Figure 1.4. Note that the weights for last fully connected layer is initialized from a gaussian distribution. We learn a simple classification network which given an input image predicts if the drone should move forward in straight line or not. Therefore, the final layer is a binary softmax and the loss is negative log-likelihood.

Test time execution

Our model essentially learns if going straight in a specific direction is good or not. But how can we use this model to fly autonomously in environments with multiple obstacles, narrow corridors and turns? We employ a simple approach to use our binary classification network to fly long distances indoors. Algorithm 2 succinctly

Algorithm 2 Policy for flying indoor

```
1: while No Crash do
2:     Input :   Real time image
3:     P(L), P(S), P(R) = Network{image, left & right crop}
4:     if P(S) > $\alpha$ then
5:         Linear Velocity = $\beta$
6:         Angular Velocity $\propto$ P(R) - P(L)
7:     else
8:         Linear Velocity = 0
9:         if P(R) > P(L) then
10:            while P(S) < $\alpha$ do
11:                P(S) = Network{image}
12:                Take Right turn
13:            end while
14:        else
15:            while P(S) < $\alpha$ do
16:                P(S) = Network{image}
17:                Take Left turn
18:            end while
19:        end if
20:    end if
21: end while
```

describes this strategy which evaluates the learned network on cropped segments of the drone's image. Based on the right cropped image, complete image and left cropped image network predicts the probability to move in right, straight and left direction. If the straight prediction (P(S)) is greater than α, drone moves forward with the yaw proportional to the difference between the right prediction (P(R)) and left prediction (P(L)). Intuitively, based on the confidence predictions of left and right, we decide to turn the robot left and right while moving forward. If the prediction for moving straight is below α (going to hit the obstacle soon), we turn the drone left or right depending on which crop of the image predicts move forward. Intuitively, if the network is fed with only the left part of the image, and the network believes that it is good to go straight there, an optimal strategy would be take a left. In this way we can use a binary classification network to choose more complex directional movements given the image of the scene. This cropping based strategy can also be extended to non planar flying by cropping vertical patches instead of horizontal ones.

Table 1.1: Average distance and average time before collision

	Glass door		NSH 4th Floor		NSH Entrance	
	Avg Dist (m)	Avg Time (s)	Avg Dist (m)	Avg Time (s)	Avg Dist (m)	Avg Time (s)
Best Straight	3.3	3.0	6.2	7.0	2.7	3.0
Depth Prediction	3.1	5.0	14.0	28.3	13.4	22.6
Our Method	27.9	56.6	54.0	120.4	42.3	78.4
Human	84.0	145.0	99.9	209.0	119.6	196.0

	Hallway		Hallway With Chairs		Wean Hall	
	Avg Dist (m)	Avg Time (s)	Avg Dist (m)	Avg Time (s)	Avg Dist (m)	Avg Time (s)
Best straight	6.2	6.6	2.7	3.1	4.2	4.6
Depth Prediction	24.9	26.6	25.5	43.5	11.6	22.1
Our Method	115.2	210.0	86.9	203.5	22.4	47.0
Human	95.7	141.0	69.3	121.0	70.6	126.0

1.4 Experimental Evaluation

We now describe the evaluation procedure of our method on indoor environments and compare with strong depth driven baselines.

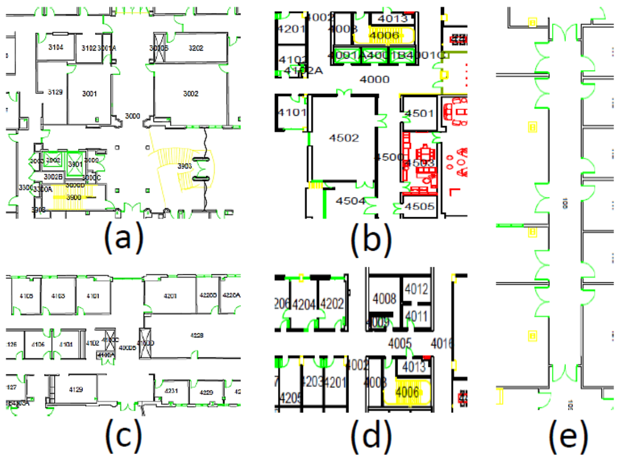


Figure 1.5: Floor plans for the testing environment are shown here: (a) NSH Entrance, (b) Wean Hall, (c) NSH 4^{th} Floor, (d) Glass Door and (e) Hallway. Note that 'Hallway with Chairs' environment has the same floorplan as 'Hallway' but with chairs as additional obstacles.

1.4.1 Baselines

To evaluate our model, we compare the performance with a Straight line policy, a Depth prediction based policy and a human controlled policy.

Straight line policy

A weak baseline for indoor navigation is to take an open-loop straight line path. For environments that contain narrow corridors, this will fail since errors compound which results in curved paths that will hit the walls. To strengthen this baseline, we choose the best (oracle) direction for the straight line policy.

Depth prediction based policy

Recent advances in depth estimation from monocular cameras [15] have shown impressive results. These models for depth estimation are often trained over around 220k indoor depth data. A strong baseline is to use these depth prediction network to generate a depth map given a monocular image. From this depth map, 3 crops, similar to test time execution of our algorithm, are extracted. Based on the average depth of these crops, the direction of movement is calculated.

Human policy

One of the strongest baseline is to use a human operator to fly the drone. In this case, we ask participants to control the drone only given the monocular image seen by the drone. The participants then use a joystick to give the commanded direction of motion to the drone. We also allow the participants to fly the drone in a test trial so that they get familiar with the response of the drone.

1.4.2 Testing Environments

To show the generalizaility of our method, we test it on 6 complex indoor environments: 'Glass Door', 'NSH 4th Floor', 'NSH Entrance', 'Wean Hall', 'Hallway' and 'Hallway with Chairs'. Floor plans for these environments can be seen in Figure 1.5. These environments have unique challenges that encompas most of the challenges faced in general purpose indoor navigation. For each of these environments, our method along with all the baselines are run 5 times with different start orientations and positions. This is done to ensure that the comparisons are robust to initializations.

Glass Door

This environment is corridor with a corner that has transparent doors. The challenge for the drone is to take a turn at this corner without crashing into the transparent door. Most depth based estimation techniques are prone to fail in this scenario (depth sensing is poor with transparent objects). However data driven visual techniques have been shown to perform reasonably with these tough transparent obstacles.

NSH 4^{th}Floor

The second environment we test on is the office space of the 4^{th} floor of the Newell Simon Hall at Carnegie Mellon University. In this environment, the drone has the capacity to navigate through a larger environment with several turns in narrow corridors.

NSH Entrance

In this environment, the drone is initialized at the entrance of the Newell Simon Hall at Carnegie Mellon University. Here the challenge is manoeuvre through a hall with glass walls and into an open space (atrium) that is cluttered with dining tables and chairs.

Wean Hall

This environment has small stretches of straight path which are connected at 90 degree to each other. Drone needs to take required turn to avoid collision at the intersection.

Hallway

This is a narrow ($2m$) dead end straight corridor where drone has keep in the center to avoid the collision with the wall. At the dead end, the drone needs to take a turn and fly back. While flying back, the drone will again meet the dead end and can turn back again. This environment tests the long term flight capacity of the drone.

Hallway With chairs

To the explicitly test the robustness of the controller to clutter and obstacles, we modify ‘Smith Hallway’ with chairs. The controller has to identify the narrow gaps between the chairs and the walls to avoid collisions. This gap can be less than 1m at some points.

We would like to point out that 2 out of 6 environments were also seen during training. These correspond to the NSH 4th Floor and NSH Entrance. The remaining 4 environments are completely novel.

1.4.3 Results

To evaluate the performance of different baselines, we used average distance and average time of flight without collisions as the metric of evaluation. This metric also terminates flight runs when they take small loops (spinning on spot). Quantitative results are presented in Table 1.1. We also qualitatively show in Figure **??** the comparison of trajectories generated by our method vs the depth prediction based baseline.

On every environment/setting we test on, we see that our method performs much better than the depth baseline. The best straight baseline provides an estimate of how difficult the environments are. The human controlled baselines are higher than our method for most environments. However for some environments like ‘Hallway with Chairs’, the presence of cluttered objects makes it difficult for the participants to navigate through narrow spaces which allows our method to surpass human level control in this environment.

A key observation is the failure of depth based methods to (a) glass walls and doors, and (b) untextured flat walls. Glass walls give the depth based models the impression that the closest obstacle is farther away. However our method, since it has been seen examples of collisions with glass windows may have latched onto features that help it avoid glass walls or doors. Another difficulty depth based models face are in untextured environments like corridors. Since our model has already seen untextured environments during training, it has learned to identify corners and realise that moving towards these corridor corners isn’t desirable.

The results on the ‘Hallway’ environments further cements our method’s claim by autonomously navigating for more than 3 minutes (battery life of drone in flight use is 5 minutes).

1.5 Conclusion

We propose a data driven approach to learn to fly by crashing more than 11,500 times in a multitude of diverse training environments. These crashing trajectories generate the biggest (to our knowledge) UAV crashing dataset and demonstrates the importance of negative data in learning. A standard deep network architecture is trained on this indoor crashing data, with the task of binary classification. By learning how to NOT fly, we show that even simple strategies easily outperforms depth prediction based methods on a variety of testing environments and is comparable to human control on some environments. This work demonstrates: (a) it is possible to collect self-supervised data for navigation at large-scale; (b) such data is crucial for learning how to navigate.

Chapter 2

Self-supervised Exploration via Disagreement

2.1 Introduction

Exploration for sensorimotor control have been addressed in literature using both model-free and model-based methods. Most recent successes in Reinforcement Learning (RL) have been achieved when exploration is done in the context of external reward function which is dense and well-shaped, e.g., a running "score" in a video game [38]. However, designing a well-shaped reward function is a challenging engineering problem. Therefore, an alternative approach has been to focus on using "intrinsic" rewards for exploration i.e. rewards computed based on the agents model of the environment. These rewards are denser compared to external rewards and hence provide early feedback to the exploration policy. Some examples of intrinsic rewards include "curiosity" [42, 44, 53] where prediction error is used as reward signal, "diversity rewards" [16, 31, 32] which discourages the agent from revisiting the same states (or similar states).

Intrinsic reward-based exploration requires building a predictive model of the world. However, there is a critical issue with building predictive models: how should the stochastic nature of agent-environment interaction be handled? Stochasticity could be caused by several sources: (1) stochastic observations, (2) noise in execution

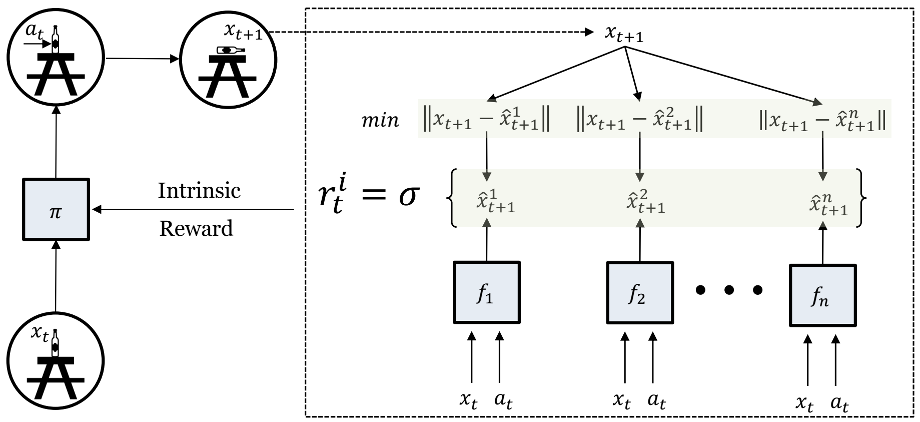


Figure 2.1: Learning to Explore via Disagreement: At time step t, the agent in the state x_t interacts with the environment by taking action a_t sampled from the current policy π and ends up in the state x_{t+1}. The ensemble of forward models $\{f_1, f_2, ..., f_n\}$ takes this current state x_t and the executed action a_t as input to predict the next state estimates $\{\hat{x}^1_{t+1}, \hat{x}^2_{t+2}, ..., \hat{x}^n_{t+1}\}$. The variance over the ensemble of network output σ is used as intrinsic reward r^i_t to train the policy. In practice, we encode the state x into an embedding space $\phi(x)$ for all the prediction purposes.

of agent's action (slipping) (3) stochasticity in the effect of the agent's action (noisy TV, coin flipping). One straightforward way is for the agent to learn a "stochastic" forward model itself! There have been several works in the literature [11, 25], but we know that learning such models from high dimensional image space is difficult to scale and extremely challenging. An alternative way to handle stochasticity is to build a deterministic prediction model and use its error as reward. Recent work proposed building such models in inverse model space that handle noise in observations but fail on stochastic nature of actions [e.g. [10]].

Beyond handling the stochastic nature, there is a bigger issue in the current formulations for intrinsic rewards. The agent performs an action and then computes the reward based on its own model and environment behavior. For example, in curiosity [44], if the internal model and the observed environment disagree, then the policy is rewarded. From an exploration viewpoint, this seems like a good formulation i.e. rewarding actions whose effects are poorly modeled. But the reward is a function of environment dynamics with respect to the performed action. Since the environment behavior function is unknown, it is treated as black-box and hence the gradients have to be estimated using approaches like REINFORCE [66] which are sample inefficient.

We address both challenges by taking inspiration from active learning. In the current curiosity formulation, one has to compare the predicted future state to "ground-truth" future state by performing the action. However, in active learning the rewards are not computed by looking at the ground-truth but rather by looking at the state of the model itself. In classical active learning, we decide what sample to label next based on confidence of the classifier or the entropy/variance of the output. Since, most of the high-capacity networks tend to overfit, confidence is not a good measure of uncertainty. Instead, inspired by the classical Query-by-Committee algorithm [57], we use a simple approach: we train an ensemble of forward dynamics models and incentivize the agent to explore the action space where there is maximum disagreement or variance among the models of this ensemble. We show that this formulation works as well as other formulations such as curiosity in non-stochastic scenarios. On the other hand, our approach is able to explore better in stochastic-dynamics scenarios because all the models in the ensemble converge to mean, reducing the variance of the ensemble. But more importantly, we show that our new objective is a differentiable function so as to perform policy optimization using likelihood maximization – much like supervised learning instead of reinforcement learning. This leads to a sample efficient exploration policy. We show experiments on large number of standard environments to demonstrate the efficacy of this approach.

2.2 Related Work

The problem of exploration is a well-studied problem in the field of reinforcement learning. Early approaches focused on studying exploration from theoretical perspective [61] and proposed Bayesian formulations [13, 28] which are hard to scale in higher dimensions. In this paper, we focus on the specific problem of exploration using intrinsic rewards. A large family of approaches use "curiosity" as a reward for training the agents. A good summary of early work in curiosity-driven rewards can be found in [42, 43]. Most approaches use some form of prediction-error between the learned model and environment behavior [44]. This prediction error can also be formulated as surprise [1, 53, 63]. Other forms of curiosity can be to explore states and actions where prediction of a forward model is highly-uncertain [25, 60]. Finally, approaches such as [36] try to explore state space which help improve the prediction model most. However, most of these efforts have still studied the problem in context of external rewards. These intrinsic rewards just guide the search to the space where forward model is uncertain or likely to be wrong.

Another approach for intrinsic rewards is using explicit visitation counts [4, 19]. These exploration strategies guide the exploration policy to "novel" states [4]. A closely related work uses diversity as a reward for exploration and skill-learning [16]. However, both visitation counts or measuring diversity requires learning a model which keeps the distribution of visited states. Learning such a model does not seem trivial. Another issue is the transferable properties and generalization of such approaches unless the state features are transferable themselves.

Finally, apart from intrinsic rewards, other approaches include using an adversarial game [62] where one agent gives the goal states and hence guiding exploration. Gregor et.al. [23] introduce a formulation of empowerment where agent prefers to go to states where it expects it will achieve the most control after learning. Researchers have also tried using perturbation of learned policy for exploration [17, 47] and using value function estimates [41]. Again these approaches have mostly been considered in context of external rewards and turn out to be sample inefficient.

Our work is mostly inspired from large-body of work in active learning (AL). In AL setting, given a collection of unlabeled examples, a learner selects which samples will be labeled by an oracle [56]. Common selection criteria include entropy [12], uncertainty sampling [34] and expected informativeness [24]. Our work is inspired from the classical work of [57], where they use disagreement among classifiers to select the datapoint. In this work, however, we apply the ideas in a completely different setting of exploration and show its applicability to environments with stochastic dynamics and improving sample-efficiency.

2.3 Exploration by Disagreement

Consider an agent interacting with the environment $\mathcal{E}$. At time t, it receives the observation x_t and then takes an action predicted by its policy, i.e., $a_t \sim \pi(x_t; \theta_P)$. Upon executing the action, it receives, in return, the next observation x_{t+1} which is 'generated' by the environment. Our goal is to build an agent that chooses its action in order to maximally explore the state space of the environment in an efficient manner. There are two main components to our agent: an intrinsic model that captures the agent's current knowledge of the states explored so far, and a policy to output actions. The policy is incentivized to take actions that lead to states where the intrinsic model is uncertain.

As our agent explores the environment, we learn a forward model that predicts the consequences of its own actions. The prediction uncertainty of this model is used to generate intrinsic rewards that incentivize the agent to visit states with maximum uncertainty. Both *measuring* and *maximizing* model uncertainty are challenging to execute with high dimensional raw sensory input (e.g. images). More importantly, the agent should learn to deal with 'stochasticity' in its interaction with the environment. This could either be caused by noisy actuation of the agent's motors, or the environment could be inherently stochastic. A deterministic prediction model will always end up with a non-zero prediction error allowing the agent to get stuck in the local minima of exploration.

A similar behavior would occur if the task at hand is too difficult to learn. Consider a robotic arm manipulating a keybunch and observing its outcome. Predicting the change in pose and position of the keys in the keybunch is extremely difficult. Although the behavior is not inherently stochastic, our agent could easily get stuck in playing with the same keybunch and not try other actions, or even other objects. Existing formulations of curiosity reward or novelty-seeking count-based methods (tabular or continuous space extensions) would also suffer in such scenarios. Learning probabilistic predictive models to measure uncertainty [25], or measuring learnability by capturing change in prediction error [42, 53] have been proposed as solutions, but have been demonstrated in low-dimensional state space inputs and are difficult to scale to high dimensional image inputs.

2.3.1 Disagreement as Intrinsic Reward

Instead of learning a single model and using its prediction error as the measure of intrinsic rewards, we propose an alternate formulation that avoids the pitfalls of prior approaches inspired from the active learning [57] literature. This is also known as optimal experimental design [8] in statistics. In active learning scenario, the goal is to find the optimal training examples to label such that the accuracy is maximized at minimum labeling cost. Exploration-driven learning is concerned with policy optimization while active learning deals with minimizing optimal cost with analytic policy. While the two might look different, we argue that active learning objectives could inspire powerful intrinsic reward formulations. In this work, we leverage the idea of model-variance maximization to propose exploration formulation.

As our agent interacts with the environment in a rollout trajectory, it collects transition data of the form $\{x_t, a_t, x_{t+1}\}$. After each rollout in the environment, the collected transition trajectories are used to train an ensemble of multiple prediction models (aka forward models) $\{f_{\theta_1}, f_{\theta_2} \ldots, f_{\theta_k}\}$ of the environment. Each of the model is trained to map a given tuple of current observation x_t and the action a_t to the resulting state x_{t+1}. These models are trained using straightforward maximum likelihood estimation that minimizes the prediction error, i.e, $\|f(x_t, a_t; \theta_F) - x_{t+1}\|_2$. To maintain the diversity across the individual models of ensemble, we train each of them on different subsets sampled randomly with replacement (bootstrap).

A good intrinsic reward formulation would be the one that encourages the agent to perform actions that lead to most informative examples. Each model in our ensemble is trained to predict the ground truth next state. Hence, the parts of the state space which have been explored by the agent well will have gathered enough data to train each model leading to agreement between the models. Since the models are learned (and not tabular), this property should generalize to unseen but similar parts of the state-space. However, the areas which are novel and unexplored would have high prediction error for each model with disagreement about the next state prediction because none of them are yet trained on such examples. Hence, we use this disagreement as an intrinsic reward. Concretely, the intrinsic reward r_t^i is defined as the *variance* across the prediction of different models in the ensemble:

$$r_t^i \mathbb{E}_\theta \Big[\|f(x_t, a_t; \theta) - \mathbb{E}_\theta[f(x_t, a_t; \theta)]\|_2^2 \Big] \tag{2.1}$$

Note that the expression on the right does not depend on the next state x_{t+1} — a property which will exploit in Section 2.3.3 to propose efficient policy optimization.

Given the agent's rollout sequence and the intrinsic reward r_t^i at each timestep t, the policy is trained to maximize the sum of expected reward, i.e., $\max_{\theta_P} \mathbb{E}_{\pi(x_t;\theta_P)} \big[\sum_t \gamma^t r_t^i \big]$ discounted by a factor γ. The agent policy and the forward model ensemble are simultaneously trained in an online manner on the data collected by the agent during exploration. This objective can be maximized by a number of policy optimization techniques. In the experiments of this paper, we use proximal policy optimization (PPO) [55] to train the policy using exploration reward.

2.3.2 Exploration in Stochastic Environments

Consider a scenario where the next state x_{t+1} is stochastic with respect to the current state x_t and action a_t. The source of stochasticity could be noisy actuation, difficulty or inherent randomness. Given enough samples, a dynamic prediction model should learn to predict the mean of the stochastic samples. Hence, the variance of the outputs in ensemble will drop preventing the agent from getting stuck in stochastic local-minima of exploration. Note this is unlike prediction error based objectives [44, 54] which will settle down to a constant value after large enough samples. And because the constant value is different from the ground-truth state the prediction error remains high and therefore the agent remains curious about the stochastic behavior. We empirically verify this intuition by comparing the behavior of variance of output with respect to a prediction error based objective in an MNIST based toy-prediction task, described in Section 2.5.2.

2.3.3 Differentiable Exploration for Policy Optimization

One commonality between different exploration methods [4, 25, 44], is that the forward model is usually learned in a supervised manner and the agent's policy is trained using reinforcement learning either in on-policy or off-policy manner. Despite several formulations, the optimization procedure for training policies to maximize these intrinsic rewards has more or less remained the same – i.e. – treating the intrinsic reward as a "black-box" even though it is generated by the agent itself.

Let's consider an example to understand the reason behind the status quo. Consider a robotic-arm agent trying to push multiple objects kept on the table in front of it by looking at the image from an overhead camera. Suppose the arm pushes an object such that it collides with another one on the table. The resulting image observation will be the outcome of complex real-world interaction, the actual dynamics of which is not known to the agent. More importantly, note that this resulting image observation is a function of the agent's action (i.e., push in this case). Most commonly, the intrinsic reward $r^i(x_t, a_t, x_{t+1})$ is function of the next state (which is a function of the agent's action), e.g., prediction error, information gain, etc. This dependency on the unknown environment dynamics absolves the policy optimization of computation of analytical gradients with respect to the intrinsic rewards. Hence, the standard way is to optimize the policy to maximize sequences of intrinsic rewards using reinforcement learning, and not make any use of the structure present in the design of r^i_t.

We formulate our proposed intrinsic reward as a differentiable function so as to perform policy optimization using likelihood maximization – much like supervised learning instead of reinforcement. If possible, this would allow the agent to make use of the structure in r^i_t explicitly, i.e., the rewarder could very efficiently inform the agent to change its action space in the direction where forward prediction loss is high, instead of providing a *scalar* feedback as in case of reinforcement learning. Explicit reward (cost) functions are one of the key reasons for success stories in optimal-control based robotics [14, 21], but they don't scale to high-dimensional state space such as images and rely on having access to a good model of the environment.

We first discuss the one step case and then provide the general setup. Consider the intrinsic reward shown in Equation (2.1), and note that our formulation does not depend on the environment interaction at all. It is purely a mental simulation of the rewarder based on the current state and the agent's prediction action. Hence, instead of maximizing the intrinsic reward in expectation via PPO (RL), we can optimize for policy parameters θ_P using direct gradients by treating rewarder as a differentiable loss function. The objective for a one-step reward horizon is:

$$\begin{aligned} &\min_{\theta_1,\ldots,\theta_k} \; (1/k)\sum_{i=1}^{k} \|f_{\theta_i}(x_t, a_t) - x_{t+1}\|_2 \\ &\max_{\theta_P} \; (1/k)\sum_{i=1}^{k}\left[\|f_{\theta_i}(x_t, a_t) - (1/k)\sum_{j=1}^{k} f_{\theta_j}(x_t, a_t)\|_2^2\right] \\ &\text{s.t.} \;\; a_t = \pi(x_t; \theta_P) \end{aligned} \tag{2.2}$$

This is optimized in an alternating fashion where the forward predictor is optimized keeping the policy parameters frozen and vice-versa. Note that both policy and forward models are trained via maximum likelihood in a supervised manner, and hence, efficient in practice.

Generalization to multi-step reward horizon To optimize policy to maximize a discounted sum of sequence of future intrinsic rewards r_t^i in a differentiable manner, the forward model would have to make predictions spanning over multiple time-steps. The policy objective in Equation (2.2) can be generalized to the multi-step horizon setup by recursively applying the forward predictor as $\sum_t r_t^i(\hat{x}_t, a_t)$ where $\hat{x}_t = f(\hat{x}_{t-1}, a_t; \theta)$, $\hat{x}_0 = x_0$, and $r_t^i(.)$ is defined in Equation (2.1). Alternatively, one could use LSTM to make forward model itself multi-step. However, training a long term multi-step prediction model is challenging and an active area of research.

2.4 Implementation Details and Baselines

Learning forward predictions in the feature space It has been shown that learning forward-dynamics predictor f_{θ_P} [10, 44] in some feature space leads to better generalization instead of making predictions in raw pixel space. Our formulation is trivially extensible to any representation space because all the operations can be performed with $\phi(x_t)$ instead of x_t. Hence, in all of our experiments, we train our forward prediction models in feature space. In particular, we use random feature space in all video games and navigation, classification features in MNIST and ImageNet-pretrained ResNet-18 features in real wolrd robot experiments. We use 5 models in the ensemble.

Back-propagation through forward model To directly optimize the policy with respect to the loss function of the forward predictor, as discussed in Section 2.3.3, we need to backpropagate all the way through action sampling process from the policy. In case of continuous action space, one could achieve this via making policy deterministic, i.e. $a_t = \pi_{\theta_P}$ with epsilon greedy sampling [35]. For discrete action space, we found that straight-through estimator [5] works well in practice.

Baseline Comparisons 'Disagreement' refers to our exploration formulation optimized using PPO [55] as discussed in Section 2.3.1, unless mentioned otherwise.

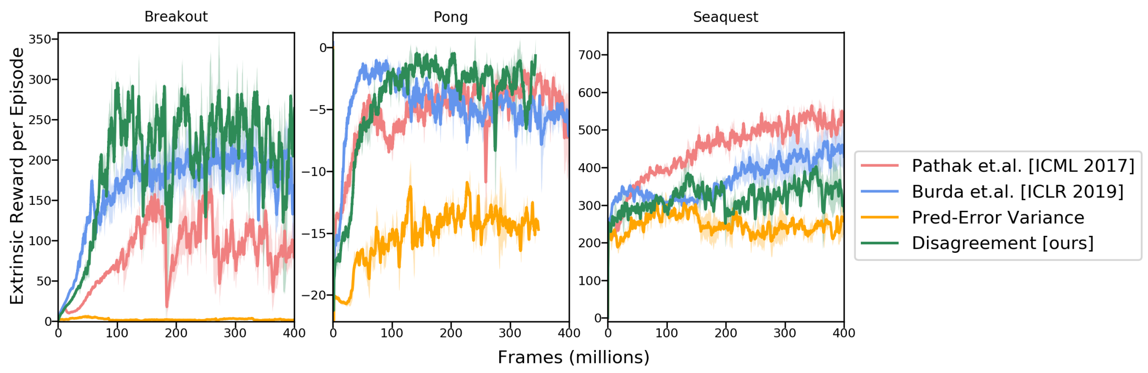


Figure 2.2: Sanity Check: Comparing our policy trained using different intrinsic reward formulation on 3 Atari games. Our approach compares favorably and does not loose accuracy in non-stochastic scenarios.

'Disagreement [Differentiable]' refers to the direct policy optimization for our formulation as described in Section 2.3.3. 'Pathak et.al. [ICML 2017]' refers to the curiosity-driven exploration formulation based on prediction error of the learned forward dynamics model in inverse model action space [44]. 'Burda et.al. [ICLR 2019]' refers to the random feature based prediction-error [10]. Final 'Pred-Error Variance' is an alternate ablation of our method where, instead of measuring variance of the output of models (i.e., disagreement), we train the agent to maximize variance of prediction error as opposed to variance of output itself.

2.5 Experiments

In this paper, we evaluate our approach on several environments including atari games, 3D navigation in unity, MNIST, object manipulation in mujoco and real world robotic manipulation task using sawyer. Our experiments can be divided into 3 parts: a) verifying the performance on standard non-stochastic environments; b) comparison on environments with stochasticity in either dynamics or observations; and c) validating the efficiency of differentiable policy optimization facilitated by our objective.

2.5.1 Sanity Check in Non-Stochastic Environments

We first verify whether proposed formulation of incentivizing agent via disagreement is able to maintain the performance on standard environment as compared to state of the art exploration techniques. Although the primary advantage of our approach is in handling stochasticity and improving efficiency via differentiable policy optimization, it should not come at the cost of performance in nearly-deterministic scenarios. We run this sanity check on standard Atari benchmark suite, as shown in Figure 2.2. These games are not completely deterministic and have some randomness as to where the agent starts when ever the game resets [38]. The agent is trained with only intrinsic reward, without any external reward from the game environment. The external reward is only used as a proxy to evaluate the quality of exploration and not shown to the agent.

The performance is compared to curiosity formulation [44], curiosity with random features [10] and methods which use variance of prediction error. We also compare our performance to Bayesian Neural Networks. In particular, we compared to Dropout NN [20]. In our method, the ensemble of models is trained in the embedding space of random network as discussed in Section 2.4. As seen in the results, our method is as good as or slightly better than state-of-the art exploration methods in most of the scenarios. Overall, these experiments suggest that our exploration formulation which is only driven by disagreement between models output compares favourably to state of the art methods and significantly better than variance of prediction error. We now discuss the more interesting setups: (a) deal with stochasticity and (b) real world exploration using robots.

2.5.2 Exploration in Stochastic Environments

Most of the exploration algorithms show evaluation on standard benchmarks which are nearly deterministic, for instance, Atari benchmark suite or mujoco gym environments. Impressive results have been shown on these environments where an agent learns skills like playing games etc, only using the intrinsic reward [16, 44]. However, these approaches collapse to the degenerate behaviours in the presence of stochasticity in the transitions [10].

A) Noisy MNIST. We first build a toy task on MNIST to intuitively demonstrate superiority compared to a prediction error based method [44]. The states are represented as the raw images. In this task the agent spawns randomly either with an image of label 0 or label 1. The dynamics of the environment are defined as follows: 1) images with label 0 always transition to another image of label 0. 2) Images with label 1 always transition to a randomly chosen image of label 2 to 9. This ensures that transitions from states of images with label 0 has low stochasticity (transitions to same label). On the other hand, states of images with label 1 have high stochasticity (transition to 1 out of 8 labels). The ideal intrinsic reward function should give almost similar importance (reward) to both the states after the agent has observed significant number of transitions. The prediction error based model assigns more importance to the states with higher stochasticity i.e. images with label 1. This behaviour is detrimental since the transition from states of images with label 1 cannot ever be perfectly modeled and hence the agent will be stuck. We compare our ensembles based approach to the prediction error based method quantitatively. Figure 2.3 shows the performance of these methods on test set of MNIST as a function of number of states visited by agent. Even after convergence, the prediction error based approach assigns higher importance/reward to state associated with the images label 1, however ensemble based method gives almost same reward to both the states.

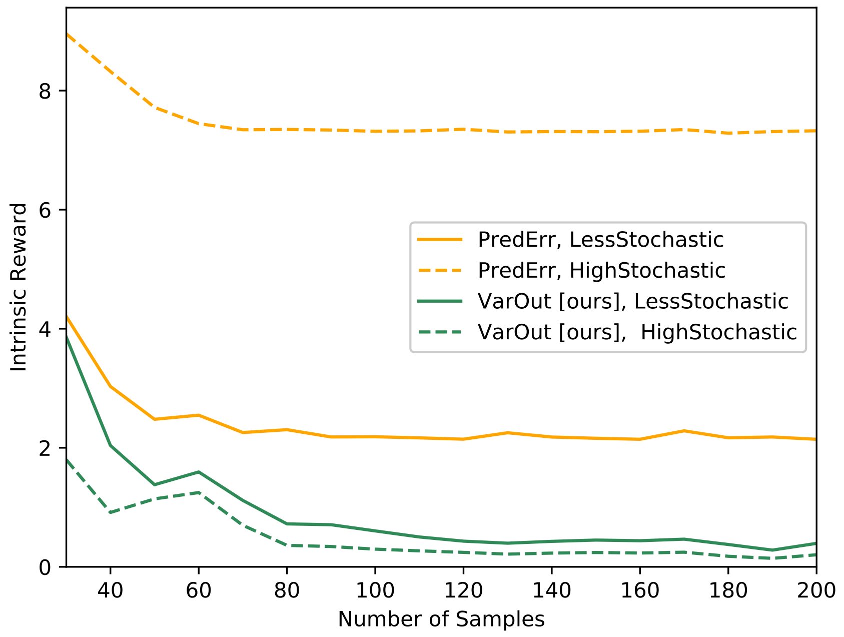


Figure 2.3: Performance of disgreement across ensemble vs prediction error based reward function on Noisy MNIST environment. This environment has 2 sets of state with different level of stochasticity associated with them. The disagreement based reward function achieves the ideal behaviour of assigning the same reward value for both states. However, the prediction error based reward function assigns a high reward to states with high stochasticity.

B) 3D Navigation. In this 3D navigation, the goal is to train the agent to reach a target location in the maze. The agent receives sparse reward of +1 on reaching the goal. For all the methods, we train the policy of the agent to maximize the joint combination of intrinsic with extrinsic reward. This particular environment is replica of VizDoom-MyWayHome environment in unity ML-agent and was developed in Burda et.al. [10]. Interestingly, this environment has 2 variants, one of which has a TV on the wall. Agent can change the channel of the TV but the content is stochastic (random images appear after pressing button). The agent can start randomly anywhere in the maze in each episode, but the goal location is fixed. We compare our proposed method with state-of-the-art prediction error-based exploration [10]. The results are shown in Figure 2.4. Our approach performs similar to the baseline in the non-TV setup, and outperforms the baseline in the presence of the TV by a significant margin. This result demonstrates that an ensemble-based disagreement could be a viable

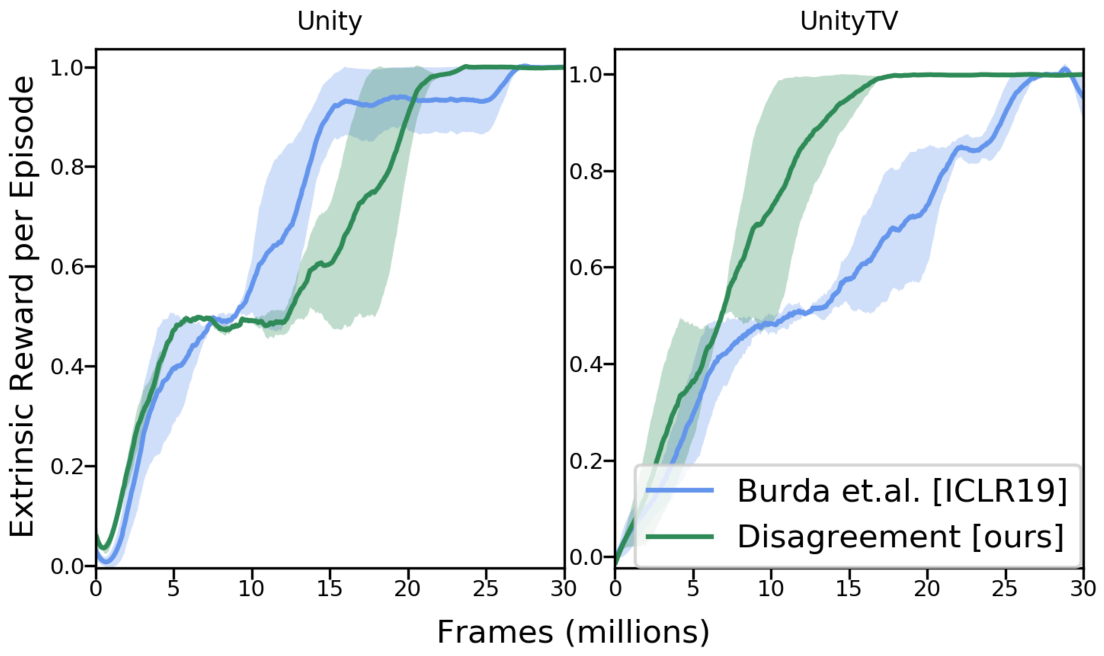


Figure 2.4: Comparing prediction error based curiosity to disagreement based exploration on 3D navigation task in Unity with and without the presence of TV.

alternative in realistic setups.

C) Atari with Sticky Actions. As discussed in the previous section, the usual Atari setup is nearly deterministic. Therefore, a recent study [37] proposed to introduce stochasticity in Atari games by making actions 'sticky'. That is, the agent's intended action is executed with half the probability at each step and otherwise the previous executed action is repeated. As shown in Figure 2.5, our disagreement based exploration approach out performs previous state-of-the-art approaches. In Pong, our approach starts slightly slower than Burda et.al. [10], but eventually achieves higher score. These experiments along with the navigation experiment, demonstrate the potential of ensembles in the face of stochasticity,

2.5.3 Differentiable Exploration in Structured Envs

We now evaluate the differentiable exploration objective proposed in Section 2.3.3. As discussed earlier, the policy is optimized via direct analytic gradients from the

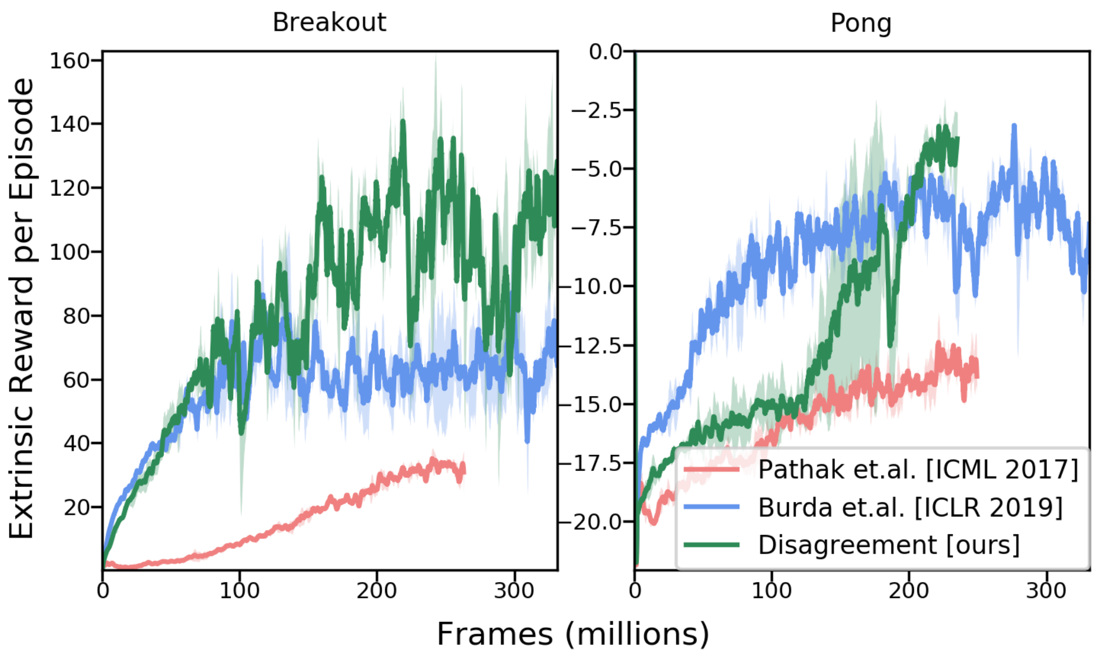


Figure 2.5: Evaluating the robustness of disagreement-based exploration when there is stochasticity in the Atari('sticky') environment.

exploration module. Therefore, the horizon of exploration depends directly on the horizon of the module. Since training long-horizon models from high dimensional inputs (images) is still an unsolved problem, we evaluate our proposed formulation on relatively short horizon scenarios. However, to compensate for length of horizon, we test on large action space setups for real world robot manipulation task.

A) Enduro Video Game. In this game, the goal of the agent is to steer the car on racing track to avoid enemies. We trained our agent via purely intrinsic rewards and extrinsic reward is only used for evaluation. In order to steer the car, the agent doesn't need to model long range dependencies. Hence, in this environment we combine our differentiable policy optimization with reinforcement learning (PPO) to maximize our disagreement based intrinsic reward. The RL captures discounted long term dependency while our differentiable formulation should efficiently take care of short horizon dependencies. We compare this formulation to purely PPO based optimization of our intrinsic reward. As shown in Figure 2.6, our differentiable exploration expedites the learning of the agent significantly. This suggests the efficacy

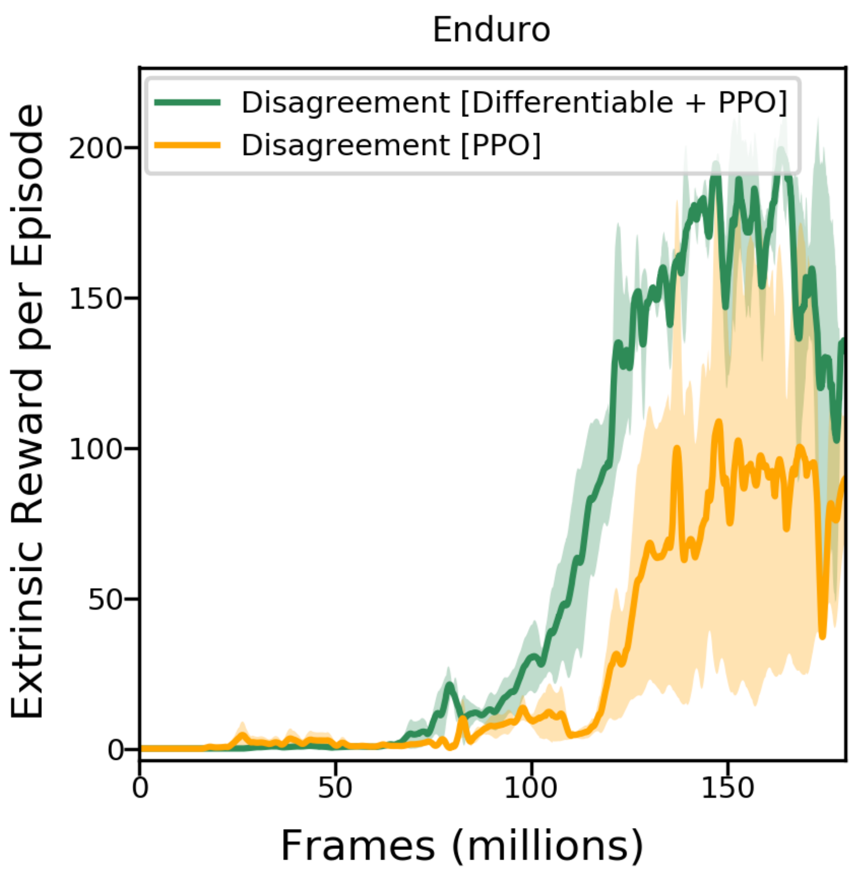


Figure 2.6: Performance on Enduro Atari Environment. We compare the performance of with or without the differentiable policy optimization.

of direct optimization. We now evaluate the performance of only differentiable exploration (without reinforcement) in short-horizon and large-structured action space setups.

B) Object Manipulation by Exploration.

We consider the task of object manipulation in complex scenarios. Our setup consists a 7-DOF robotic arm that could be tasked to interact with the the objects kept on the table in front of it. The objects are kept randomly in the workspace of the robot on the table. Robot's action space is 4 dimensional: a) location (x, y) of point on the surface of table, b) angle of approach θ and c) gripper status, a binary value indicating whether to grasp (open the gripper fingers) or push (keep fingers close).

All of our experiments use raw visual RGBD images as input and predict actions as output. Note that, to accurately grasp or push objects, the agent needs to figure out an accurate combination of location, orientation and gripper status.

The action space is discretized into 224×224 locations, 16 orientations for grasping (fingers close) and 16 orientations for pushing [67]. The policy takes as input a 224×224 RGBD image and produces push and grasp action probabilities for each pixel. Instead of adding the 16 rotations in the output, we pass 16 equally spaced rotated images to the network and then sample actions based on the output of all the inputs. This exploits the convolutional structure of the network. The task has a short horizon but very large state and action spaces.

We make no assumption about either the environment or the training signal. Our robotic agents explore the work-space purely out of their own intrinsic reward in a pursuit to develop useful skills. We have instantiated this setup both in simulation and in the real world.

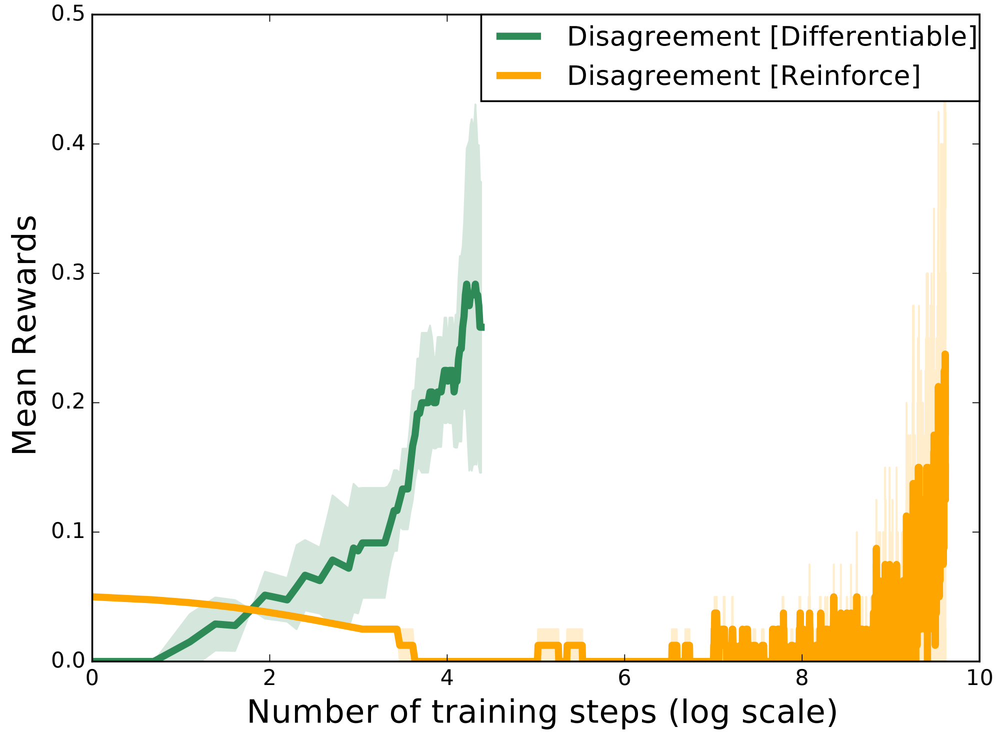
B1) Object Manipulation in MuJoCo. We first carry out a study in simulation to compare the performance of differentiable variant of our disagreement objective against the reinforcement learning based optimization of disagreement. We used MuJoCo to simulate the robot performing grasping and pushing on table top environment as described above.

To evaluate the quality of exploration, we measure the frequency at which agent the agent interacts with the object. Note, this is just a proxy and is not used for training the agent. It represents how quickly our policy learns to explore interesting part of space. Figures 2.7 show the performance when the environment consist of single object. Our approach is able to exploit the structure in the loss, resulting in order of magnitude faster learning than REINFORCE.

B2) Real-World Robotic Manipulation. We now deploy our sample-efficient exploration formulation on real-world robotic setup. The real-world poses additional challenges unlike simulated environments in terms of behavior, dynamics of varied objects available in the real world. Our robotic setup consisted of a Sawyer-arm with a table placed in front of it. We mounted KinectV2 at a fixed location from robot to receive RGBD observations of the environment. In every run, the robot starts with 3 object placed in front of it. If either the robot completes 100 interactions or there are no objects in front of it, objects are re-placed manually.

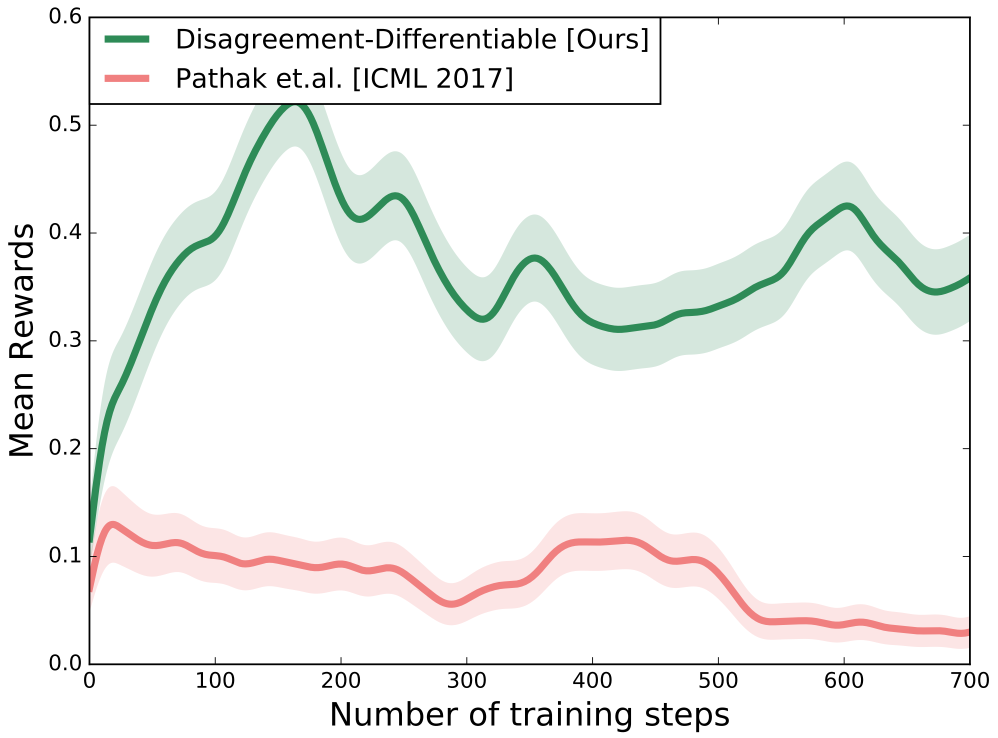
We again use the same metric (number of object interactions) as before to measure effectiveness of the exploration policy. We monitor the change in the RGB image to see if the robot has interacted with objects. Figure 2.8 shows how effective our differentiable curiosity module is and how it learns to interact with object even in less than 1000 examples.

We further test the skills learned by our robot during its curious exploration by measuring interaction frequency on a set of held-out objects. Out of total of 30 objects, we created set of 20 objects for training and 10 objects for testing. Both, our method and reinforce were trained for 700 robot interaction with the environment. Both models were evaluated based on the 80 robot interaction. During testing, environment reset was done after every 10 robot steps. Our final trained exploration policy interacts approximately **67%** of times with unseen objects whereas random performs **17%**. On the other hand, it seems that REINFORCE just collapses and only **1%** of actions involve interaction with objects. Please see robot videos in the supplementary.



[b]0.32

Figure 2.7: Mujoco



[b]0.3

Figure 2.8: Real Robot

Figure 2.9: Interaction Rate vs. Number of Samples for mujoco and real-world robot. We measure the exploration quality by evaluating the object interaction frequency of the agent. Note that the Mujoco plot is in log-scale. For both the environments, our differentiable policy optimization explores much efficiently.

Bibliography

[1] Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv:1703.01732*, 2017. 2.2

[2] Markus Achtelik, Abraham Bachrach, Ruijie He, Samuel Prentice, and Nicholas Roy. Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments. In *SPIE Defense, security, and sensing*, pages 733219–733219. International Society for Optics and Photonics, 2009. 1.2.1

[3] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016. 1.1, 1.2.2

[4] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprintarXiv:1606.01868*, 2016. 2.2, 2.3.3

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013. 2.4

[6] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous robots*, 27(3):201–219, 2009. 1.2.1

[7] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 5776–5783. IEEE, 2011. 1.1, 1.2.1, 1.3.2

[8] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004. 2.3.1

[9] Adam Bry, Abraham Bachrach, and Nicholas Roy. State estimation for aggressive flight in gps-denied environments using onboard sensing. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012. 1.2.1

[10] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and

Alexei A Efros. Large-scale study of curiosity-driven learning. *ICLR*, 2019. 2.1, 2.4, 2.4, 2.5.1, 2.5.2, 2.5.2, 2.5.2

[11] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018. 2.1

[12] I. Dagan and S. Engelson. Committee-based sampling for training probabilistic classifiers. *ICML*, 1995. 2.2

[13] Marc Deisenroth and Carl Rasmussen. Pilco: A model-based and data-efficient approach to policy search. *ICML*, 2011. 2.2

[14] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011. 2.3.3

[15] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014. 1.4.1

[16] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv:1802.06070*, 2018. 2.1, 2.2, 2.5.2

[17] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *arXiv:1706.10295*, 2017. 2.2

[18] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4557–4564. IEEE, 2012. 1.2.1

[19] Justin Fu, John D Co-Reyes, and Sergey Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. *NIPS*, 2017. 2.2

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015. 2.5.1

[21] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016. 2.3.3

[22] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages

580–587, 2014. 1.2.2, 1.3.3

[23] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *ICLR Workshop*, 2017. 2.2

[24] N. Houlsby, F. Huszr, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. *arXiv*, 2011. 2.2

[25] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *NIPS*, 2016. 2.1, 2.2, 2.3, 2.3.3

[26] Yoshifumi Kitamura, Fumio Kishino, Takaaki Tanaka, and W Yachida. Real-time path planning in a dynamic 3-d environment. In *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 2, pages 925–931. IEEE, 1996. 1.1

[27] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007. 1.3.2

[28] Z. Kolter and A. Ng. Near-bayesian exploration in polynomial time. *ICML*, 2009. 2.2

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. (document), 1.2.2, 1.4, 1.3.3

[30] Kevin Lamers, Sjoerd Tijmons, Christophe De Wagter, and Guido de Croon. Self-supervised monocular distance learning on a lightweight micro air vehicle. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1779–1784. IEEE, 2016. 1.2.1

[31] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 2011. 2.1

[32] Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011. 2.1

[33] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *ISER*, 2016. 1.2.2

[34] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. *ACM SIGIR*, 1994. 2.2

[35] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016. 2.4

[36] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*, 2012. 2.2

[37] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009, 2017. URL `http://arxiv.org/abs/1709.06009`. 2.5.2

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015. 2.1, 2.5.1

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 1.2.2

[40] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002. 1.3.2

[41] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016. 2.2

[42] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 2009. 2.1, 2.2, 2.3

[43] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *Evolutionary Computation*, 2007. 2.2

[44] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 2.1, 2.1, 2.2, 2.3.2, 2.3.3, 2.4, 2.4, 2.5.1, 2.5.2, 2.5.2

[45] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. *CoRR*, abs/1609.09025, 2016. URL `http://arxiv.org/abs/1609.09025`. 1.2.2

[46] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *ICRA*, 2016. 1.1, 1.2.2, 1.3.2

[47] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y

Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv:1706.01905*, 2017. 2.2

[48] James F Roberts, Timothy Stirling, Jean-Christophe Zufferey, and Dario Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, number LIS-CONF-2007-006, 2007. 1.2.1

[49] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772. IEEE, 2013. 1.1, 1.2.1, 1.3.2

[50] Franck Ruffier and Nicolas Franceschini. Aerial robot piloted in steep relief by optic flow sensors. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1266–1273. IEEE, 2008. 1.2.1

[51] Fereshteh Sadeghi and Sergey Levine. (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL http://arxiv.org/abs/1611.04201. 1.1, 1.2.1

[52] Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain, and Mike Elgersma. Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 27(5):549–574, 2008. 1.1, 1.2.1

[53] Jürgen Schmidhuber. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pages 1458–1463. IEEE, 1991. 2.1, 2.2, 2.3

[54] Jurgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, 1991. 2.3.2

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. 2.3.1, 2.4

[56] B. Settles. Active learning literature survey. *U Madison Tech Report*, 2010. 2.2

[57] H.S. Seung, M. Opper, and H. Sompolinsky. Query by committee. *COLT*, 1992. 2.1, 2.2, 2.3.1

[58] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Aerial Informatics and Robotics platform. Technical Report MSR-TR-2017-9, Microsoft Research, 2017. 1.2.1

[59] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016. URL http://arxiv.org/abs/1604.03540. 1.3.2

[60] Susanne Still and Doina Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 2012. 2.2

[61] A. Strehl and M. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 2008. 2.2

[62] Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *ICLR*, 2018. 2.2

[63] Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *AGI*, 2011. 2.2

[64] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, 23 (7-8):693–716, 2004. 1.3.2

[65] Nicolas Vandapel, James Kuffner, and Omead Amidi. Planning 3-d path networks in unstructured environments. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4624–4629. IEEE, 2005. 1.1

[66] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992. 2.1

[67] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *CoRR*, abs/1803.09956, 2018. URL `http://arxiv.org/abs/1803.09956`. 2.5.3

[68] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*, 2016. 1.1