Visual Reaction: Learning to Play Catch with Your Drone

Kuo-Hao Zeng[1]   Roozbeh Mottaghi[1,2]   Luca Weihs[2]   Ali Farhadi[1]

[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]PRIOR @ Allen Institute for AI

Abstract

In this paper we address the problem of visual reaction*: the task of interacting with dynamic environments where the changes in the environment are not necessarily caused by the agent itself. Visual reaction entails predicting the future changes in a visual environment and planning accordingly. We study the problem of visual reaction in the context of playing catch with a drone in visually rich synthetic environments. This is a challenging problem since the agent is required to learn (1) how objects with different physical properties and shapes move, (2) what sequence of actions should be taken according to the prediction, (3) how to adjust the actions based on the visual feedback from the dynamic environment (e.g., when objects bouncing off a wall), and (4) how to reason and act with an unexpected state change in a timely manner. We propose a new dataset for this task, which includes 30K throws of 20 types of objects in different directions with different forces. Our results show that our model that integrates a forecaster with a planner outperforms a set of strong baselines that are based on tracking as well as pure model-based and model-free RL baselines. The code and dataset are available at* github.com/KuoHaoZeng/Visual_Reaction.

1. Introduction

One of the key aspects of human cognition is the ability to interact and react in a visual environment. When we play tennis, we can predict how the ball moves and where it is supposed to hit the ground so we move the tennis racket accordingly. Or consider the scenario in which someone tosses the car keys in your direction and you quickly reposition your hands to catch them. These capabilities in humans start to develop during infancy and they are at the core of the cognition system [3, 8].

Visual reaction requires predicting the future followed by planning accordingly. The future prediction problem has received a lot of attention in the computer vision community. The work in this domain can be divided into two major categories. The first category considers predicting future actions of people or trajectories of cars (e.g., [5, 22, 25,
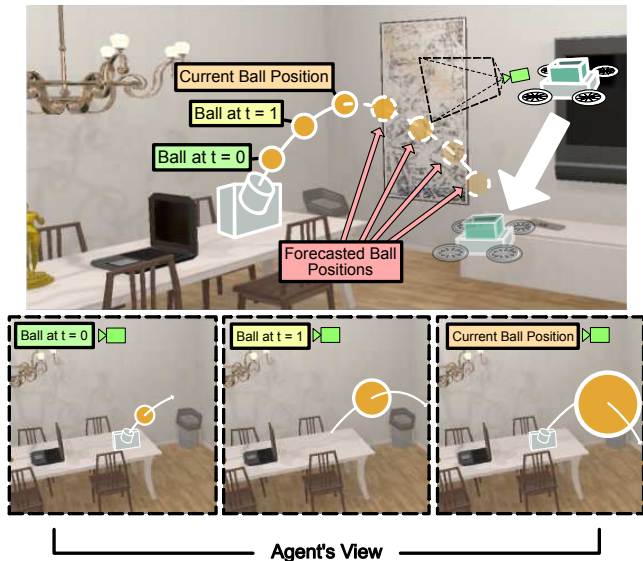

Figure 1: Our goal is to train an agent that can visually react with an interactive scene. In the studied task, the environment can evolve independently of the agent. There is a launcher in the scene that throws an object with different force magnitudes and in different angles. The drone learns to predict the trajectory of the object from ego-centric observations and move to a position that can catch the object. The trajectory of the thrown objects varies according to their weight and shape and also the magnitude and angle of the force used for throwing.

58]). Typically, there are multiple correct solutions in these scenarios, and the outcome depends on the intention of the people. The second category is future prediction based on the physics of the scene (e.g., [27, 32, 60, 66]). The works in this category are mostly limited to learning from passive observation of images and videos, and there is no interaction or feedback involved during the prediction process.

In this paper, we tackle the problem of *visual reaction*: the task of predicting the future movements of objects in a dynamic environment and planning accordingly. The interaction enables us to make decisions on the fly and receive feedback from the environment to update our belief about the

future movements. This is in contrast to passive approaches that perform prediction given pre-recorded images or videos. We study this problem in the context of playing catch with a drone, where the goal is to catch a thrown object using only visual ego-centric observations (Figure 1). Compared to the previous approaches, we not only need to predict future movements of the objects, but also to infer a minimal set of actions for the drone to catch the object in a timely manner.

This problem exhibits various challenges. First, objects have different weights, shapes and materials, which makes their trajectories very different. Second, the trajectories vary based on the magnitude and angle of the force used for throwing. Third, the objects might collide with the wall or other structures in the scene, and suddenly change their trajectory. Fourth, the drone movements are not deterministic so the same action might result in different movements. Finally, the agent has limited time to reason and react to the dynamically evolving scene to catch the object before it hits the ground.

Our proposed solution is an adaptation of the model-based Reinforcement Learning paradigm. More specifically, we propose a forecasting network that rolls out the future trajectory of the thrown object from visual observation. We integrate the forecasting network with a model-based planner to estimate the best sequence of drone actions for catching the object. The planner is able to roll out sequences of actions for the drone using the dynamics model and an action sampler to select the best action at each time step. In other words, we learn a policy using the rollout of both object and agent movements.

We perform our experiments in AI2-THOR [23], a near-photo-realistic interactive environment which models physics of objects and scenes (object weights, friction, collision, etc). Our experiments show that the proposed model outperforms baselines that are based on tracking (current state estimation as opposed to forecasting) and also pure model-free and model-based baselines. We provide an ablation study of our model and show how the performance varies with the number of rollouts and also the length of the planning horizon. Furthermore, we show how the model performs for object categories unseen during training.

The contributions of the paper are as follows: (1) We investigate the problem of *visual reaction* in an interactive, dynamic, and visually rich environment. (2) We propose a new framework and dataset for visual reaction in the context of playing catch with a drone. (3) We propose a solution by integrating a planner and a forecaster and show it significantly outperforms a number of strong baselines. (4) We provide various analyses to better evaluate the models.

2. Related Work

Future prediction & Forecasting. Various works explore future prediction and forecasting from visual data. Several authors consider the problem of predicting the future tra-

jectories of objects from individual [31, 37, 55, 56, 57, 65] and multiple sequential [1, 22, 62] images. Unlike these works, we control an agent that interacts with the environment which causes its observation and viewpoint to change over time. A number of approaches explore prediction from ego-centric views. [36] predict a plausible set of ego-motion trajectories. [39] propose an Inverse Reinforcement Learning approach to predict the behavior of a person wearing a camera. [54] learn visual representation from unlabelled video and use the representation for forecasting objects that appear in an ego-centric video. [26] predict the future trajectories of interacting objects in a driving scenario. Our agent also forecasts the future trajectory based on ego-centric views of objects, but the prediction is based on physical laws (as opposed to peoples intentions). The problem of predicting future actions or the 3D pose of humans has been explored by [6, 14, 25, 49]. Also, [5, 28, 46, 52, 53, 63] propose methods for generating future frames. Our task is different from the mentioned approaches as they use pre-recorded videos or images during training and inference, while we have an interactive setting. Methods such as [13] and [10] consider future prediction in interactive settings. However, [13] is based on a static third-person camera and [10] predicts the effect of agent actions and does not consider the physics of the scene.

Planning. There is a large body of work (e.g., [7, 16, 18, 19, 34, 38, 45, 51, 59]) that involves a model-based planner. Our approach is similar to these approaches as we integrate the forecaster with a model-based planner. The work of [4] shares similarities with our approach. The authors propose learning a compact latent state-space model of the environment and its dynamics; from this model an Imagination-Augmented Agent [38] learns to produce informative rollouts in the latent space which improve its policy. We instead consider visually complex scenarios in 3D so learning a compact generative model is not as straightforward. Also, [59] adopts a model-based planner for the task of vision and language navigation. They roll out the future states of the agent to form a model ensemble with model-free RL. Our task is quite different. Moreover, we consider the rollouts for both the agent and the moving object, which makes the problem more challenging.

Object catching in robotics. The problem of catching objects has been studied in the robotics community. Quadro-copters have been used for juggling a ball [33], throwing and catching a ball [40], playing table tennis [44], and catching a flying ball [47]. [20] consider the problem of catching in-flight objects with uneven shapes. These approaches have one or multiple of the following issues: they use multiple external cameras and landmarks to localize the ball, bypass the vision problem by attaching a distinctive marker to the ball, use the same environment for training and testing, or assume a stationary agent. We acknowledge that experiments

on real robots involve complexities such as dealing with air resistance and mechanical constraints that are less accurately modeled in our setting.

Visual navigation. There are various works that address the problem of visual navigation towards a static target using deep reinforcement learning or imitation learning (e.g., [17, 29, 43, 64, 67]). Our problem can be considered as an extension of these works since our target is moving and our agent has a limited amount of time to reach the target. Our work is also different from drone navigation (e.g., [15, 41]) since we tackle the visual reaction problem.

Object tracking. Our approach is different from object tracking (e.g., [2, 9, 11, 35, 48]) as we forecast the future object trajectories as opposed to the current location. Also, tracking methods typically provide only the location of the object of interest in video frames and do not provide any mechanism for an agent to take actions.

3. Approach

We first define our task, *visual reaction*: the task of interacting with dynamic environments that can evolve independently of the agent. Then, we provide an overview of the model. Finally, we describe each component of the model.

3.1. Task definition

The goal is to learn a policy to catch a thrown object using an agent that moves in 3D space. There is a launcher in the environment that throws objects in the air with different forces in different directions. The agent needs to predict the future trajectory of the object from the past observations (three consecutive RGB images) and take actions at each timestep to intercept the object. An episode is successful if the agent catches the object, i.e. the object lies within the agent's top-mounted basket, before the object reaches the ground. The trajectories of objects vary depending on their physical properties (e.g., weight, shape, and material). The object might also collide with walls, structures, or other objects, and suddenly change its trajectory.

For each episode, the agent and the launcher start at a random position in the environment (more details in Sec. 4.1). The agent must act quickly to reach the object in a short time before the object hits the floor or goes to rest. This necessitates the use of a forecaster module that should be integrated with the policy of the agent. We consider 20 different object categories such as basketball, newspaper, and bowl (see the supplementary for the complete list).

The model receives ego-centric RGB images from a camera that is mounted on top of the drone agent as input, and outputs an action $a_{d_t} = (\Delta_{v_x}, \Delta_{v_y}, \Delta_{v_z}) \in [-25m/s^2, 25m/s^2]^3$ for each timestep t, where, for example, Δ_{v_x} shows acceleration, in meters, along the x-axis. The movement of the agent is not deterministic due to the time dependent integration scheme of the physics engine.
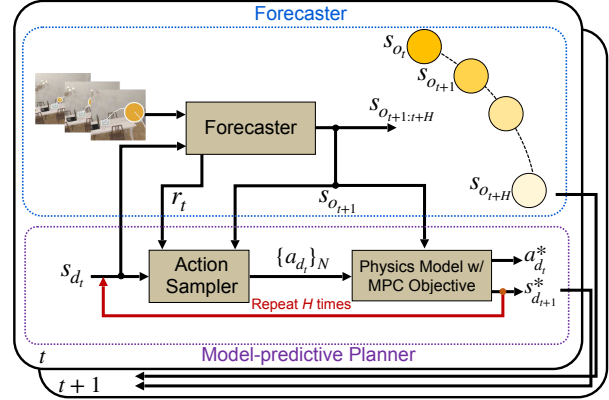

Figure 2: **Model overview.** Our model includes two main parts: Forecaster and Planner. The visual encoding of the frames, object state, agent state and action are denoted by r, s_o, s_d, and a, respectively. t denotes the timestep, and H is the planning horizon.

In the following, we denote the agent and object state by $s_d = [d, v_d, a_d, \phi, \theta]$ and $s_o = [o, v_o, a_o]$, respectively. d, v_d and a_d denote the position, velocity and acceleration of the drone and o, v_o and a_o denote those of the object. ϕ and θ specify the orientation of the agent camera, which can rotate independently from the agent.

3.2. Model Overview

Our model has two main components: a *forecaster* and a *model-predictive planner*, as illustrated in Fig. 2. The forecaster receives the visual observations $i_{t-2:t}$ and the estimated agent state s_{d_t} at time t, and predicts the current state s_{o_t} of the thrown object. The forecaster further uses the predicted object state (i.e., position, velocity and acceleration) to forecast H steps of object states $s_{o_{t+1:t+H}}$ in the future. The model-predictive planner is responsible for generating the best action for the agent such that it intercepts the thrown object. The model-predictive planner receives the future trajectory of the object from the forecaster and also the current estimate of the agent state as input and outputs the best action accordingly. The model-predictive planner includes an action sampler whose goal is to sample N sequences of actions given the current estimate of the agent state, the predicted object trajectory, and the intermediate representation r_t produced by the visual encoder in the forecaster. The action sampler samples actions according to a policy network that is learned. The second component of the model-predictive planner consists of a physics model and a model-predictive controller (MPC). The physics model follows Newton Motion Equation to estimate the next state of the agent (i.e., position and velocity at the next timestep) given the current state and action (that is generated by the action sampler). Our approach builds on related joint model-based and model-free RL ideas. However, instead of an

ensemble of model-free and model-based RL for better decision making [24, 59], or using the dynamics model as a data augmentor/imaginer [12, 38] to help the training of model-free RL, we explicitly employ model-free RL to train an action sampler for the model-predictive planner.

In the following, we begin by introducing our forecaster, as shown in Fig. 3(a), along with its training strategy. We then describe how we integrate the forecaster with the model-predictive planner, as presented in Fig. 2 and Fig. 3(b). Finally, we explain how we utilize model-free RL to learn the action distribution used in our planner, Fig. 3(b).

3.3. Forecaster

The purpose of the forecaster is to predict the current object state s_{o_t}, which includes the position $o_t \in \mathbb{R}^3$, the velocity $v_{o_t} \in \mathbb{R}^3$, and the acceleration $a_{o_t} \in \mathbb{R}^3$, and then, based on the prediction, forecast future object positions $o_{t+1:t+H}$ from the most recent three consecutive images $i_{t-2:t}$. The reason for forecasting H timesteps in the future is to enable the planner to employ MPC to select the best action for the task. We show how the horizon length H affects the performance in the supplementary. Note that if the agent does not catch the object in the next timestep, we query the forecaster again to predict the trajectory of the object $o_{t+2:t+H+1}$ for the next H steps. Forecaster also produces the intermediate visual representation $r_t \in \mathbb{R}^{256}$, which is used by the action sampler. The details are illustrated in Fig. 3(a). We define the positions, velocity, and acceleration in the agent's coordinate frame at its starting position.

The three consecutive frames $i_{t-2:t}$ are passed through a convolutional neural network (CNN). The features of the images and the current estimate of the agent state s_{d_t} are combined using an MLP, resulting in an embedding r_t. Then, the current state of the object s_{o_t} is obtained from r_t through three separate MLPs. The NME, which follows the discretized Newton's Motion Equation ($o_{t+1} = o_t + v_t \times \Delta t$, $v_{t+1} = v_t + a_t \times \Delta t$) receives the predicted state of the object to calculate the future positions $o_{t+1:t+H}$. We take the derivative of NME and back-propagate the gradients through it in the training phase. Note that NME itself is not learned.

To train the forecaster, we provide the ground truth positions of the thrown object from the environment and obtain the velocity and acceleration by taking the derivative of the positions. We cast the position, velocity, and acceleration prediction as a regression problem and use the L1 loss for optimization.

3.4. Model-predictive Planner

Given the forecasted trajectory of the thrown object, our goal is to control the flying agent to catch the object. We integrate the model-predictive planner with model-free RL to explicitly incorporate the output of the forecaster.

Our proposed model-predictive planner consists of a model-predictive controller (MPC) with a physics model, and an action sampler as illustrated in Fig. 3(b). We will describe how we design the action sampler in Sec. 3.5. The action sampler produces a rollout of future actions. The action is defined as the acceleration a_d of the agent. We sample N sequences of actions that are of length H from the action distribution. We denote these N sequences by $\mathbf{a}_{d_{t:t+H-1}}$. For each action in the N sequences, the physics model estimates the next state of the agent $s_{d_{t+1}}$ given the current state s_{d_t} by using the discretized Newton's Motion Equation ($d_{t+1} = d_t + v_{d_t} \times \Delta t$, $v_{d_{t+1}} = v_{d_t} + a_{d_t} \times \Delta t$). This results in N possible trajectories $d_{t+1:t+H}$ for the agent. Given the forecasted object trajectories $o_{t+1:t+H}$, the MPC then selects the best sequence of actions $\mathbf{a}^*_{t:t+H-1}$ based on the defined objective. The objective for MPC is to select a sequence of actions that minimizes the sum of the distances between the agent and the object over H timesteps. We select the first action a^*_t in the sequence of actions, and the agent executes this action. We feed in the agent's next state $s^*_{d_{t+1}}$ for planning in the next timestep.

Active camera viewpoint. The agent is equipped with a camera that rotates. The angle of the camera is denoted by ϕ and θ in the agent's state vector s_d. We use the estimated object and agent position at time $t + 1$, o_{t+1} and d_{t+1}, to compute the angle of the camera. We calculate the relative position $p \in (p_x, p_y, p_z)$ between object and agent by $o - d$. Then, we obtain the Euler angles along y-axis and x-axis by $\arctan \frac{p_x}{p_z}$ and $\arctan \frac{p_y}{p_z}$, respectively. In the supplementary, we also show results for the case that the camera is fixed.

3.5. Action sampler

The actions can be sampled from a uniform distribution over the action space or a learned policy network. We take the latter approach and train a policy network which is conditioned on the forecasted object state, current agent state and visual representation. Model-based approaches need to sample a large set of actions at each timestep to achieve a high level of performance. To alleviate this issue, we parameterize our action sampler by a series of MLPs that learns an action distribution given the current agent state, the forecasted trajectory of the object $o_{t+1:t+H}$ and the visual representation r_t of observation $i_t - 2 : t$ (refer to Sec. 3.3). This helps to better shape the action distribution, which may result in requiring fewer samples and better performance.

To train our policy network, we utilize policy gradients with the actor-critic algorithm [50]. To provide the reward signal for the policy gradient, we use the 'success' signal (if the agent catches the object or not) as a reward. In practice, if the agent succeeds to catch the object before it hits the ground or goes to rest, it would receive a reward of $+1$. Furthermore, we also measure the distance between the agent trajectory and the object trajectory as an additional reward signal (pointwise distance at each timestep). As a result, the
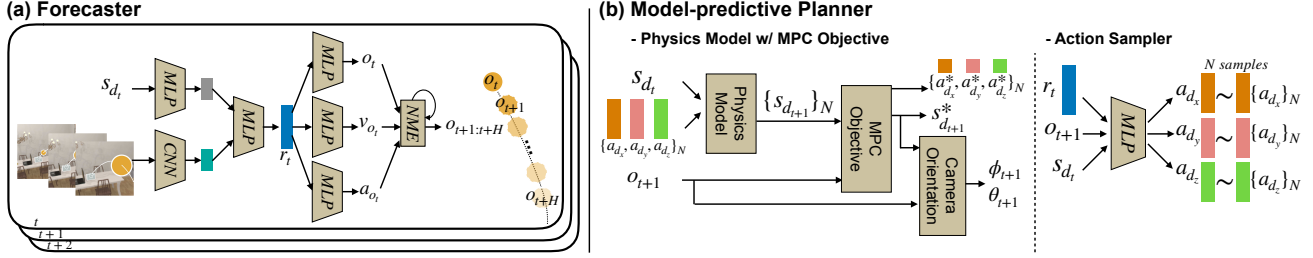
(a) Forecaster

(b) Model-predictive Planner
- Physics Model w/ MPC Objective
- Action Sampler

Figure 3: **Model architecture.** (a) The forecaster receives images and an estimate of the agent state s_{d_t} as input and outputs the estimates for the current state s_{o_t}, including o_t, v_{o_t}, and a_{o_t}. Then it forecasts future positions of the object $o_{t+1:t+H}$ by discretized Newton Motion Equation. Forecasting is repeated every timestep if the object has not been caught. (b) The model-predictive planner includes a MPC w/ Physics model and an action sampler. The action sampler generates N sequences $\mathbf{a}_{d_{t:t+H-1}} = \{(\Delta_{v_x,i}^j, \Delta_{v_y,i}^j, \Delta_{v_z,i}^j)_{i=t}^{t+H-1} | j = 1, ..., N\}$ of actions at each timestep, and an optimal action $(\Delta_{v_x^*}, \Delta_{v_y^*}, \Delta_{v_z^*})$ is chosen such that it minimizes the distance between the agent and the object at each timestep.

total reward for each episode is $R = \mathbb{1}\{\text{episode success}\} - 0.01 \cdot \sum_t ||d_t^* - o_t^*||_2$ where d_t^* and o_t^* are the ground truth positions of the agent and object at time t.

4. Experiments

We first describe the environment that we use for training and evaluating our model. We then provide results for a set of baselines: different variations of using current state prediction instead of future forecasting and a model-free baseline. We also provide ablation results for our method, where we use uniform sampling instead of the learned action sampler. Moreover, we study how the performance changes with varying mobility of the agent, noise in the agent movement and number of action sequence samples. Finally, we provide analysis of the results for each object category, different levels of difficulty, and objects unseen during training.

4.1. Framework

We use AI2-THOR [23], which is an interactive 3D indoor virtual environment with near photo-realistic scenes. We use AI2-THOR v2.3.8, which implements physical properties such as object materials, elasticity of various materials, object mass and includes a drone agent. We add a launcher to the scenes that throws objects with random force magnitudes in random directions.

The trajectories of the objects vary according to their mass, shape, and material. Sometimes the objects collide with walls or other objects in the scene, which causes sudden changes in the trajectory. Therefore, standard equations of motion are not sufficient to estimate the trajectories, and learning from visual data is necessary. The statistics of the average velocity and the number of collisions have been provided in Fig. 4. More information about the physical properties of the objects are in the supplementary.

The drone has a box on top to catch objects. The size of drone is $0.47m \times 0.37m$ with a height of $0.14m$, and the box is $0.3m \times 0.3m$ with a height of $0.2m$. The drone is equipped with a camera that is able to rotate. The maximum acceleration of the drone is $25m/s^2$ and the maximum velocity is $40m/s$. However, we provide results for different maximum acceleration of the drone. The action for the drone is specified by acceleration in x, y, and z directions. The action space is continuous, but is capped by the maximum acceleration and velocity.

Experiment settings. We use the *living room* scenes of AI2-THOR for our experiments (30 scenes in total). We follow the common practice for AI2-THOR wherein the first 20 scenes are used for training, the next 5 for validation, and the last 5 for testing. The drone and the launcher are assigned a random position at the beginning of every episode. We set the horizontal relative distance between the launcher and the drone to be 2 meters (any random position). We set the height of the launcher to be 1.8 meters from the ground which is similar to the average human height. The drone faces the launcher in the beginning of each episode so it observes that an object is being thrown.

To throw the object, the launcher randomly selects a force between $[40, 60]$ newtons, an elevation angle between $[45, 60]$ degree, and an azimuth angle between $[-30, 30]$ degree for each episode. The only input to our model at inference time is the ego-centric RGB image from the drone. We use 20 categories of objects such as basketball, alarm clock, and apple for our experiments. We observe different types of trajectories such as parabolic motion, bouncing off the walls and collision with other objects, resulting in sharp changes in the direction. Note that each object category has different physical properties (mass, bounciness, etc.) so the trajectories are quite different. We use the same objects for training and testing. However, the scenes, the positions, the magnitude, and the angle of the throws vary at test time. We also show an experiment, where we test the model on categories unseen during training. We consider 20K trajectories during training, 5K for val and 5K for test. The number of trajectories is uniform across all object categories.
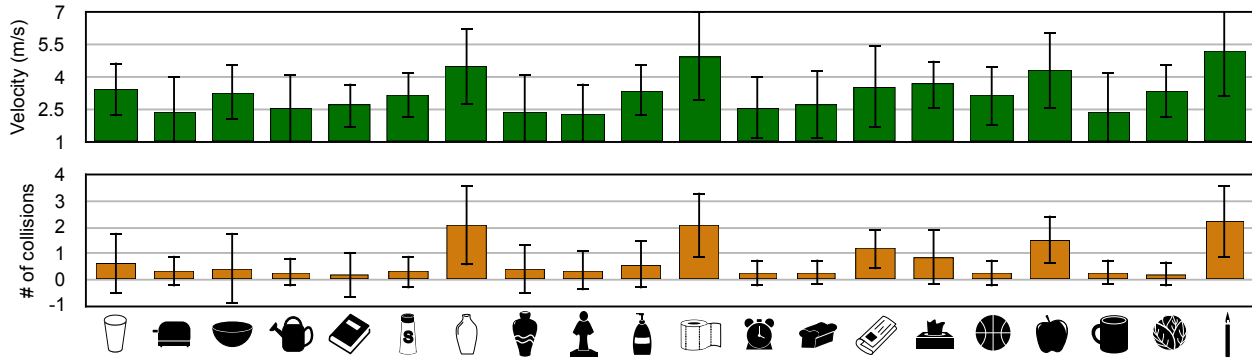
Figure 4: **Dataset statistics.** We provide the statistics for the 20 types of objects in our dataset. We illustrate the average velocity along the trajectories and the number of collisions with walls or other structures in the scene.

4.2. Implementation details

We train our model by first training the forecaster. Then we freeze the parameters of the forecaster, and train the action sampler. An episode is successful if the agent catches the object. We end an episode if the agent succeeds in catching the object, the object falls on the ground, or the length of the episode exceeds 50 steps which is equal to 1 second. We use SGD with initial learning rate of 10^{-1} for forecaster training and decrease it by a factor of 10 every 1.5×10^4 iterations. For the policy network, we employ Adam optimizer [21] with a learning rate of 10^{-4}. We evaluate the framework every 10^3 iterations on the validation scenes and stop the training when the success rate saturates. We use MobileNet v2 [42], which is an efficient and light-weight network as our CNN model. The forecaster outputs the current object position, velocity, and acceleration. The action sampler provides a set of accelerations to the planner. They are all continuous numbers. The supplementary provides details for the architecture of each component of the model.

4.3. Baselines

Current Position Predictor (CPP). This baseline predicts the current position of the object relative to the initial position of the drone in the 3D space, o_t, instead of forecasting the future trajectory. The model-predictive planner receives this predicted position at each time-step and outputs the best action for the drone accordingly. The prediction model is trained by an L1 loss with the same training strategy used for our method.

CPP + Kalman filter. We implement this baseline by introducing the prediction update through time to the Current Position Predictor (CPP) baseline. We assume the change in the position of the object is linear and follows the Markov assumption in a small time period. Thus, we add the Kalman Filter [61] right after the output of the CPP. To get the transition probability, we average the displacements along the three dimensions over all the trajectories in the training set. We set the process variance to the standard deviation of

the average displacements, and measurement variance to 3×10^{-2}. Further, same as CPP, the model-predictive planner receives this predicted position at each time-step as input and outputs the best action to control the agent. This baseline is expected to be better than CPP, because the Kalman Filter takes into account the possible transitions obtained from the training set so it further smooths out the noisy estimations.

Model-free (A3C [30]). Another baseline is model-free RL. We use A3C [30] as our model-free RL baseline. The network architecture for A3C includes the same CNN and MLP used in our forecaster and the action sampler. The network receives images $i_{t-2:t}$ as input and directly outputs action a_t for each time-step. We train A3C by 4 threads and use SharedAdam optimizer with the learning rate of 7×10^{-4}. We run the training for 8×10^4 iterations ($\approx$ 12 millions frames in total). In addition to using the the 'success' signal as the reward, we use the distance between the drone and the object as another reward signal.

4.4. Ablations

We use the training loss described in Sec. 3.3 and the training strategy mentioned in Sec. 4.2 for ablation studies. **Motion Equation (ME).** The forecaster predicts the position, velocity, and acceleration at the first time-step so we can directly apply motion equation to forecast all future positions. However, since our environment implements complex physical interactions, there are several different types of trajectories (e.g., bouncing or collision). We evaluate if simply using the motion equation is sufficient for capturing such complex behavior.

Uniform Action Sampling (AS). In this ablation study, we replace our action sampler with a sampler that samples actions from a uniform distribution. This ablation shows the effectiveness of learning a sampler in our model.

4.5. Results

Quantitative results. The results are summarized in Tab. 1 for all 20 objects and different number of action sequences.

	N = 100000	N = 10000	N = 1000	N = 100	N = 10	Best
Curr. Pos. Predictor (CPP)	22.92±2.3	22.57±2.0	21.04±1.2	18.72±1.8	10.86±0.5	22.92±2.3
CPP + Kalman Filter	23.22±1.29	22.78±0.90	21.88±0.79	19.29±0.81	12.17±1.2	23.22±1.29
Model-free (A3C [30])	-	-	-	-	-	4.54±2.3
Ours, ME, uniform AS	6.12±0.7	6.11±0.7	6.00±0.5	5.99±0.5	5.12±1.0	6.12±0.7
Ours, uniform AS	26.01±1.3	25.47±1.3	23.61±1.5	20.65±0.93	10.58±1.1	26.01±1.3
Ours, full	**29.34±0.9**	**29.26±1.4**	**29.12±0.8**	**29.14±0.8**	**24.72±1.6**	**29.34±0.9**
MPC Upper bound	68.67±1.9	76.00±0.0	78.67±1.9	66.00±3.3	49.33±10.5	78.67±1.9

Table 1: **Quantitative results.** We report the success rate for the baselines and the ablations of our model. N refers to the number of action sequences that the action sampler provides. The model-free baseline does not have an action sequence sampling component so we can provide only one number. The MPC upper bound is the case that model-predictive planner uses perfect forecasting with uniform action sampler. Note that the MPC upper bound must be done in the off-line mode since the perfect forecasting only available after collecting the objects' trajectory.

Ours, uniform AS	20.4	49.8	32.4	13.7	39.2	17.3	12.9	6.0	37.6	26.8	0.0	49.6	34.8	24.0	25.6	61.2	14.0	12.8	39.6	8.4
Ours, full	22.8	65.9	35.2	20.1	37.2	18.5	14.5	12.0	42.8	29.6	0.0	54.4	37.2	24.4	26.4	64.4	18.1	18.0	40.8	10.0

Table 2: **Per category result.** Our dataset includes 20 object categories. We provide the success rate for each object category.

We use success rate as our evaluation metric. Recall that the action sampler samples N sequences of future actions. We report results for five different values $N = 10, 100, 1000, 10000, 100000$. We set the horizon H to 3 for the forecaster and the planner. For evaluation on the test set, we consider $5K$ episodes for each model. For Tab. 1, we repeat the experiments 3 times and report the average.

As shown in the table, both the current position predictors (CPP) and the Kalman Filter (CPP + Kalman Filter) baseline are outperformed by our model, which shows the effectiveness of forecasting compared to estimating the current position. Our full method outperforms the model-free baseline, which shows the model-based portion of the model helps improving the performance. 'Ours, ME, uniform AS' is worse than the two other variations of our method. This shows that simply applying motion equation and ignoring complex physical interactions is insufficient and it confirms that learning from visual data is necessary. We also show that sampling from a learned policy 'Ours - full' outperforms 'Ours, uniform AS', which samples from a uniform distribution. This justifies using a learned action sampler and shows the effectiveness of the integration of model-free and model-based learning by the model-predictive planner.

4.6. Analysis

Per-category results. Tab. 2 shows the results for each category for 'Ours - full' and 'Ours, uniform AS'. The results show that our model performs better on relatively heavy objects. This is expected since typically there is less variation in the trajectories of heavy objects.

Difficulty-based categorization. Tab. 3 shows the performance achieved by 'Ours - full' and 'Ours, uniform AS' in terms of difficulty of the trajectory. The difficulty is defined

	Easy	Medium	Difficult
Proportion	43%	33%	24%
Ours, uniform AS	46.4	18.4	1.2
Ours, full	51.9	20.7	1.6

Table 3: **Difficulty categorization.** We show the categorization of the results for different levels of difficulty.

	100%	80%	60%	40%	20%
Ours, uniform AS	26.0	23.6	16.0	10.5	3.3
Ours, full	29.3	25.1	18.4	10.5	3.5

Table 4: **Mobility results.** We show the results using 100%, 80%, 60%, 40%, 20% of the maximum acceleration.

by how many times the object collides with other structures before reaching the ground or being caught by the agent. We define *easy* by no collision, *medium* by colliding once, and *difficult* by more than one collision. The result shows that even though our model outperforms baselines significantly, it is still not as effective for *medium* and *difficult* trajectories. It suggests that focusing on modeling more complex physical interactions is important for future research.

Different mobility. We evaluate how varying the mobility of the drone affects the performance (Tab. 4). We define the mobility as the maximum acceleration of the drone. We re-train the model using 100%, 80%, 60%, 40%, 20% of the maximum acceleration.

Movement noise. Here, we evaluate the scenarios where the agent has more noisy movements. We perform this by injecting a Gaussian noise to the drone's movement after each action (Fig. 6). We re-train the model using 0.01, 0.05, 0.1, and 0.15 of the standard deviation of the Gaussian noise. As expected, the performance decreases with more noise.

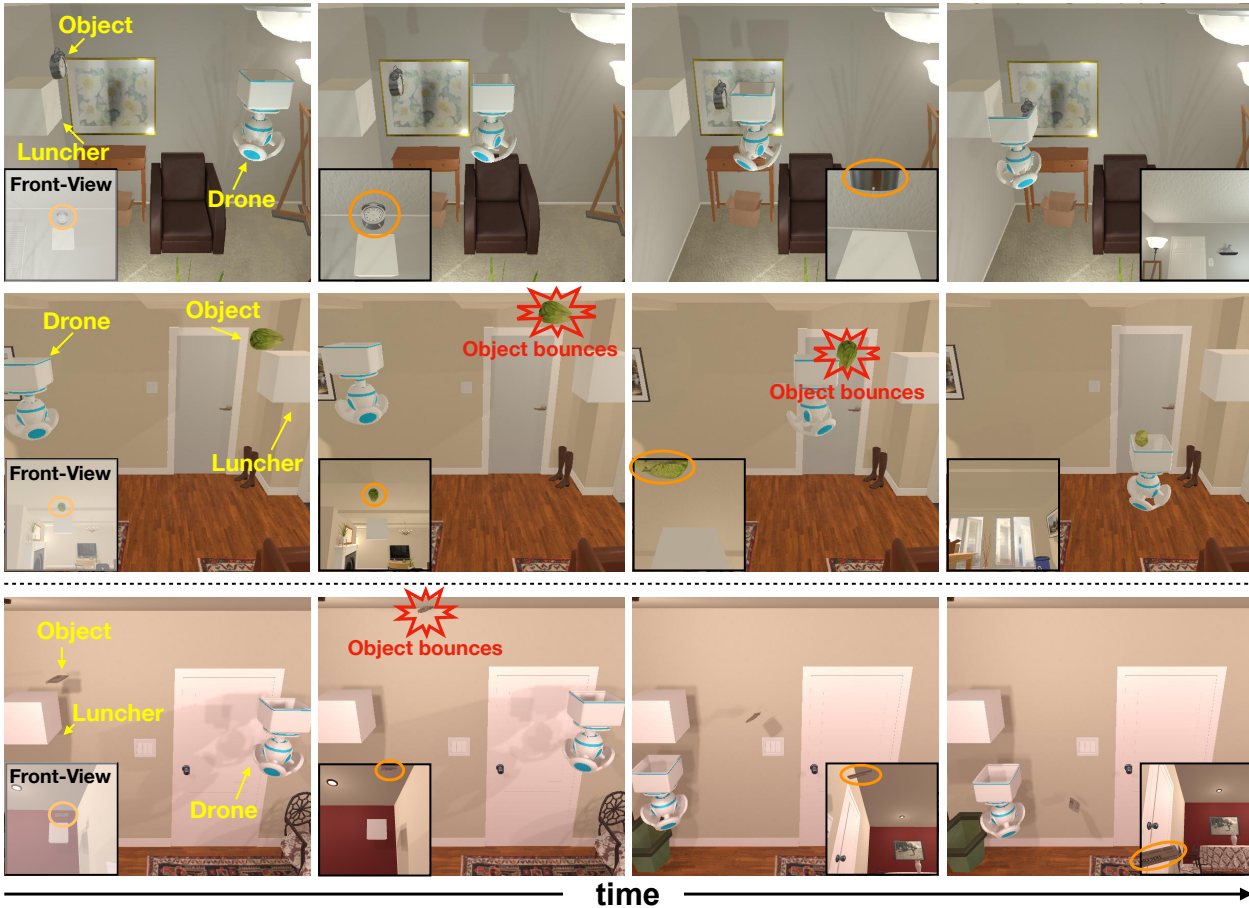**Unseen categories.** We train the best model on 15 object

Figure 5: **Qualitative Results.** We show two successful sequences of catching objects in the first two rows and a failure case in the third row. For instance, in the second row, the object bounces off the ceiling, but the drone is still able to catch it.
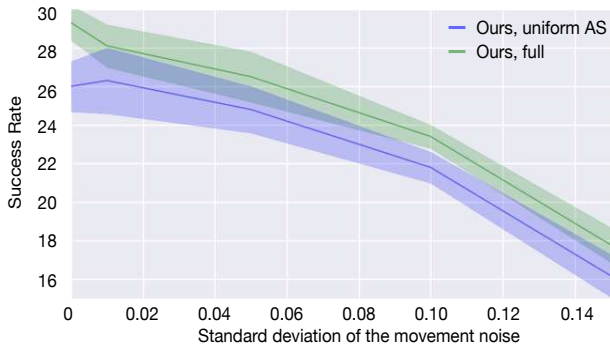

Figure 6: **Movement noise variation results.** We show how the noise in the agent movement affects the performance.

categories (the list is in the supplementary) and evaluate on the remaining categories. The success rate is $29.12 \pm 0.9\%$. This shows that the model is rather robust to unseen categories.

Qualitative results. Fig. 5 shows two sequences of catching the object and a failure case. The sequence is shown from a third person's view and the agent camera view (we only use the camera view as the input to our model). The second row shows the drone is still able to catch the object although there is a sudden change in the direction due to the collision

of the object with the ceiling. A supplementary video[1] shows more success and failure cases.

More analysis are provided in the supplementary.

5. Conclusion

We address the problem of *visual reaction* in an interactive and dynamic environment in the context of learning to play catch with a drone. This requies learning to forecast the trajectory of the object and to estimate a sequence of actions to intercept the object before it hits the ground. We propose a new dataset for this task, which is built upon the AI2-THOR framework. We showed that the proposed solution outperforms various baselines and ablations of the model including the variations that do not use forecasting, or do not learn a policy based on the forecasting.

---

[1] https://youtu.be/iyAoPuHxvYs

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, 2016. 2

[2] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In *ECCV*, 2016. 3

[3] Andreja Bubic, D Yves Von Cramon, and Ricarda I Schubotz. Prediction, cognition and the brain. *Frontiers in human neuroscience*, 4:25, 2010. 1

[4] Lars Buesing, Théophane Weber, Sébastien Racanière, S. M. Ali Eslami, Danilo Jimenez Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning. *arXiv*, 2018. 2

[5] Lluís Castrejón, Nicolas Ballas, and Aaron C. Courville. Improved vrnns for video prediction. In *ICCV*, 2019. 1, 2

[6] Yu-Wei Chao, Jimei Yang, Brian L. Price, Scott Cohen, and Jia Deng. Forecasting human dynamics from static images. In *CVPR*, 2017. 2

[7] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *ICML*, 2017. 2

[8] Andy Clark. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, 36(3):181–204, 2013. 1

[9] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *CVPR*, 2017. 3

[10] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *ICLR*, 2017. 2

[11] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *ICCV*, 2017. 3

[12] V Feinberg, A Wan, I Stoica, MI Jordan, JE Gonzalez, and S Levine. Model-based value expansion for efficient model-free reinforcement learning. In *ICML*, 2018. 4

[13] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *NeurIPS*, 2016. 2

[14] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *ICCV*, 2015. 2

[15] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017. 3

[16] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016. 2

[17] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 3

[18] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv*, 2018. 2

[19] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NeurIPS*, 2015. 2

[20] Seungsu Kim, Ashwini Shukla, and Aude Billard. Catching objects in flight. *IEEE Transactions on Robotics*, 30:1049–1065, 2014. 2

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 6

[22] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In *ECCV*, 2012. 1, 2

[23] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 2, 5

[24] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *ICLR*, 2018. 4

[25] Tian Lan, Tsung-Chuan Chen, and Silvio Savarese. A hierarchical representation for future action prediction. In *ECCV*, 2014. 1, 2

[26] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B. Choy, Philip H. S. Torr, and Manmohan Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. In *CVPR*, 2017. 2

[27] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv*, 2016. 1

[28] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016. 2

[29] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017. 3

[30] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 6, 7

[31] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. In *CVPR*, 2016. 2

[32] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. In *ECCV*, 2016. 1

[33] Mark Muller, Sergei Lupashin, and Raffaello D'Andrea. Quadrocopter ball juggling. In *IROS*, 2011. 2

[34] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018. 2

[35] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 3

[36] Hyun Soo Park, Jyh-Jing Hwang, Yedong Niu, and Jianbo Shi. Egocentric future localization. In *CVPR*, 2016. 2

[37] Silvia L. Pintea, Jan C. van Gemert, and Arnold W. M. Smeulders. Déjà vu: Motion prediction in static images. In *ECCV*, 2014. 2

[38] Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NeurIPS*, 2017. 2, 4

[39] Nicholas Rhinehart and Kris M. Kitani. First-person activity forecasting with online inverse reinforcement learning. In *ICCV*, 2017. 2

[40] Robin Ritz, Mark W. Müller, Markus Hehn, and Raffaello D'Andrea. Cooperative quadrocopter ball throwing and catching. In *IROS*, 2012. 2

[41] Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In *RSS*, 2017. 3

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 6

[43] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018. 3

[44] Rui Silva, Francisco S. Melo, and Manuela M. Veloso. Towards table tennis with a quadrotor autonomous learning robot and onboard vision. In *IROS*, 2015. 2

[45] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, 2017. 2

[46] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015. 2

[47] Kunyue Su and Shaojie Shen. Catching a flying ball with a vision-based quadrotor. In *ISER*, 2017. 2

[48] Chong Sun, Huchuan Lu, and Ming-Hsuan Yang. Learning spatial-aware regressions for visual tracking. In *CVPR*, 2018. 3

[49] Chen Sun, Abhinav Shrivastava, Carl Vondrick, Rahul Sukthankar, Kevin Murphy, and Cordelia Schmid. Relational action forecasting. In *CVPR*, 2019. 2

[50] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 2000. 4

[51] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *NeurIPS*, 2016. 2

[52] Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc V. Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. In *NeurIPS*, 2019. 2

[53] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *ICLR*, 2017. 2

[54] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating visual representations from unlabeled video. In *CVPR*, 2016. 2

[55] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*, 2016. 2

[56] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014. 2

[57] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *ICCV*, 2015. 2

[58] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In *ICCV*, 2017. 1

[59] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *ECCV*, 2018. 2, 4

[60] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter W. Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *NeurIPS*, 2017. 1

[61] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995. 6

[62] Dan Xie, Sinisa Todorovic, and Song-Chun Zhu. Inferring "dark matter" and "dark energy" from videos. In *ICCV*, 2013. 2

[63] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NeurIPS*, 2016. 2

[64] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. In *ICLR*, 2019. 3

[65] Jenny Yuen and Antonio Torralba. A data-driven approach for event prediction. In *ECCV*, 2010. 2

[66] Bo Zheng, Yibiao Zhao, Joey C. Yu, Katsushi Ikeuchi, and Song-Chun Zhu. Scene understanding by reasoning stability and safety. *IJCV*, 2014. 1

[67] 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. In *ICRA*, 2017. 3