A Data-Efficient Framework for Training and Sim-to-Real Transfer of Navigation Policies

Homanga Bharadhwaj[1*], Zihan Wang[2*], Yoshua Bengio[3] and Liam Paull[3]

***Abstract*— Learning effective visuomotor policies for robots purely from data is challenging, but also appealing since a learning-based system should not require manual tuning or calibration. In the case of a robot operating in a real environment the training process can be costly, time-consuming, and even dangerous since failures are common at the start of training. For this reason, it is desirable to be able to leverage *simulation* and *off-policy* data to the extent possible to train the robot. In this work, we introduce a robust framework that plans in simulation and transfers well to the real environment. Our model incorporates a gradient-descent based planning module, which, given the initial image and goal image, encodes the images to a lower dimensional latent state and plans a trajectory to reach the goal. The model, consisting of the encoder and planner modules, is first trained through a meta-learning strategy in simulation. We subsequently perform adversarial domain transfer on the encoder by using a bank of unlabelled but random images from the simulation and real environments to enable the encoder to map images from the real and simulated environments to a similarly distributed latent representation. By fine tuning the entire model (encoder + planner) with only a few real world expert demonstrations, we show successful planning performances in different navigation tasks.**

I. INTRODUCTION

Applying machine learning - and specifically deep reinforcement learning - to robotics algorithm development has shown great promise recently [1]–[3]. However, state-of-the-art methods still require a lot of experiments on the physical robot [4], which is very expensive and possibly even dangerous if the robot is learning a task where wrong execution can cause harm or damage. Furthermore, there are few guarantees that a policy learned by one robot in a particular environment will *transfer* to another (even slightly) different robot or another (even slightly) different environment. The recently popularized theory of "meta-learning" [5]–[7] offers a methodology for overcoming the policy transfer issue, but at the expense of an even higher data requirement.

In practice, a roboticist has two potential tools to aid in reducing the number of real on-policy rollouts that are needed on the real robot. The first is a *simulator*. A simulator requires development effort to build, but there are now incredible tools to facilitate this. However, there will always be a discrepancy between the simulator and the real world, both in terms of the world dynamics and the perception of the environment. This will induce a distributional shift between training and test data which is problematic for deep learning. The second resource that is likely readily available is *off-policy* rollouts from the real robot. The most common example could be data collected while the robot is being teleoperated safely by a person.

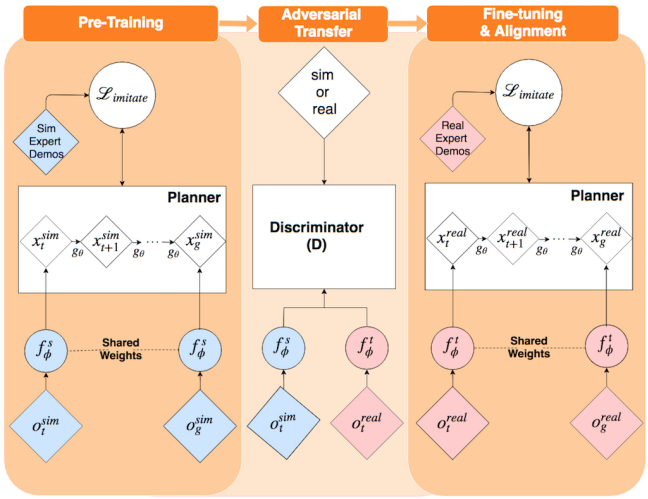


Fig. 1: **Sim-to-real transfer of navigation policies**: Our method comprises three main elements. First, a planner is trained in simulation. This process includes a learned image encoding f^s_ϕ and a learned dynamics model g_θ. Second, an adversarial discriminative transfer approach is used to allow real images to be encoded in the same way as the simulated ones. Finally, a small amount of fine-tuning is performed on the real environment.

In this work, we propose a novel procedure for combining these two resources (simulation and off-policy data) to efficiently train a physically embodied agent to complete a task in the real world. In short, we use the simulation environment to learn a policy for navigation in a meta-learning setup and then transfer the learned policy to the real world using an adversarial domain adaptation approach [8]. We use as a basis for our planner the Universal Planning Network [9] but make several improvements that make our approach particularly well-suited to the transfer learning scenario and show the impact of these improvements by rigorous experiments on a real robot in the easily reproducible Duckietown environment [10].

Also of note is that most of the approaches in the literature related to the transfer from a simulation to a real robot that we are aware of consider a robot agent that is fully observed from an off-board camera. None of them consider the task of mobile robot navigation [8], [11]–[14]. In this work, we consider the case of a mobile robot with an onboard camera. This is an important consideration because the robot must now additionally implicitly infer its own state from partial observations over time rather than having the luxury to be able to infer its state fully from one observation. It is also more challenging from a visuomotor policy learning perspective since the camera itself is moving and therefore many of the pixels will change, rather than just the agent as in other works [9].

*Equal contribution. Author ordering determined by Liam holding a duckie behind his back and making Homanga guess "right" or "left".

[1] Homanga Bharadhwaj is with the Department of Computer Science and Engineering, IIT Kanpur, India

[2] Zihan Wang is with the Division of Engineering Science, University of Toronto, Canada

[3] Yoshua Bengio and Liam Paull are with Mila, Universite de Montreal

We also generalize the adversarial domain transfer method for sim-to-real transfer of an end-to-end gradient-descent based planner, where separate supervisory signals are not available for the perception and control modules separately. We first train using expert trajectories in simulation and then perform adversarial transfer on the encoder's output space to learn mappings from the real environment that are similar to the mappings from the simulation environment. In particular, we claim the following contributions:

- We develop a stable and efficient planning model for navigation through incorporation of a meta-learned loss function, latent space regularization terms and a stochastic forward dynamics model in the planning objective.
- We demonstrate on a real robot that the developed policy (encoder + planner) trained in simulation can transfer to a real environment (by using very few real expert demonstrations for fine-tuning) through an adversarial transfer approach.

II. BACKGROUND AND RELATED WORKS

Our work draws inspiration from recent developments in meta-learning and sim-to-real policy transfer.

A. Meta Learning

Meta-learning models are trained by being subjected to a variety of tasks in training and are then tested in their ability to learn new tasks. The concept is not new [15], [16], but has become increasingly relevant in modern deep reinforcement learning and imitation learning algorithms [5], [17]–[23]. Model-Agnostic Meta Learning (MAML) [5]–[7] provides a framework for rapidly adapting gradient-based planners to different (new) tasks by performing a few gradient steps. On a high level, our approach is inspired by MAML in the sense that we have a two-stage computation through gradient descent during training. The inner stage computes a plan given the planner, while the outer stage updates the parameters of the planner, including the weights of the neural network used as the inner stage loss function.

1) Universal Planning Networks: The UPN [9] framework considers the problem of finding a plan $\hat{a}_{t:t+T}$ given an initial image o_t and a goal image o_g as inputs. Similar to MAML it employs a two-tiered approach: 1) optimize the trajectories (sequence of actions) with gradient decent given a planner (inner loop) and 2) optimize the representations in the planner (outer loop) using expert trajectories. The planning module consists of a forward dynamics model g_θ (a fully connected neural network) and an encoder f_ϕ (a convolutional neural network) with θ and ϕ being neural network parameters respectively, which are learned in an end-to-end manner.

In each iteration, for a fixed planning horizon T, the current and goal images are encoded into a latent space X:

$$\begin{aligned} x_t &= f_\phi(o_t) \\ x_g &= f_\phi(o_g) \end{aligned} \tag{1}$$

The latent representation at the end of the horizon, x_{t+T+1} is calculated by recursively applying the learned forward dynamics model and the current estimate of the actions, $\hat{a}_{t\ldots t+T}$ in the planned trajectory:

$$\hat{x}_{t+1} = g_\theta(\hat{x}_t, \hat{a}_t) \tag{2}$$

starting from the latent encoding of the initial image x_t. The inner loop planning loss is then calculated as the discrepancy between the direct encoding of the goal image and the latent space estimate generated by propagating the initial image encoding through the learned dynamics model T times.

$$\mathscr{L}^{(i)}_{plan} = ||\hat{x}^{(i)}_{t+T+1} - x_g||^2 \tag{3}$$

This loss is backpropagated to find the best *actions* given the encoding parameters ϕ and the dynamics model parameters θ. This process repeats until convergence (gradient descent). Once a trajectory has been converged upon, it is compared with an expert trajectory, $a^*_{t:t+T}$, using an outer-loop imitation loss:

$$\mathscr{L}_{imitate} = ||\hat{a}_{t:t+T} - a^*_{t:t+T}||^2_2 \tag{4}$$

This loss is back-propagated into the planner and used to update the parameters of the planner ϕ and θ. This process continues over a batch of expert demonstrations until convergence in the hope that the resulting latent space encoding and dynamics model parameters will be automatically learned.

This setup is elegant since it is able to learn a latent encoding without wasting additional optimization effort on reconstruction as is the case in a variational autoencoder setup such as DARLA [24]. However, in our experience it suffers from the following shortcomings:

1) It is data inefficient and requires a lot of expert trajectories to train,
2) The inflexible planning loss constrains the learning process because it is not necessarily suitable for every task, since what is a good representation to model state transitions may not be best to measure discrepancy to the goal,
3) While it is able to adapt to new dynamics models (this is shown in an RL context in [9]) it is not able to adapt to changes in the perceptual environment, which limits its ability to transfer from a simulator to a real robot,
4) The learned dynamics model lacks the robustness to be used on a real robot since it is devoid of any notion of stochasticity.

In Sec. III we detail how our method overcomes these shortcomings.

B. Sim-to-Real Transfer

The goal of sim-to-real transfer is to use simulated or synthetic data, which are cheap and easy to be collected, to partially or fully replace the use of real-world data, which are expensive and time consuming to obtain [25]–[27]. The main challenge in effective sim-to-real transfer is that there are aspects of reality which cannot be modelled well in the simulation environment [28]. Hence, a model that has been trained in simulation cannot be directly deployed in the real environment since there is a distributional shift between the test data and the training data [29]. One approach to close the "reality gap" is by matching the simulator to physical reality via dedicated system identification and superior-quality rendering [30]–[32]. However this is very expensive in terms of development effort and, not very effective based on past results [33]. Apart from this, there are broadly two categories of approaches to resolve the

aforementioned issue, 1) learning invariant features and 2) learning a mapping from simulation to real.

1) Learning Invariant Representations: Domain randomization [12], [25]–[27], [34]–[36] bridges the reality gap by leveraging rich variations of the simulation environment during training. The hope is that by adding random variability in the simulator, the real data distribution will be within that of the training data. However, recent results have only been able to successfully use domain randomization for relatively simple tasks like object localization [27] and robotic grasping [37] with no use cases in navigation to the best of our knowledge. Additionally, which parameters to randomize and to what degree is done heuristically and requires significant testing and tuning.

2) Learning the Mapping between Simulation and Real: A second option is to explicitly learn the relationship between the simulated and real data [38]. Then, a policy trained on the simulator can be executed in the real world by pre-processing the real data to make it seem like simulated data. A recent approach [39] proposed a Simulated+Unsupervised (S+U) learning method which utilizes unlabeled real data to learn a model in order to improve the performance of a simulated agent.

Another approach, namely "Adversarial Discriminative Domain Adaptation" [40] has the key advantage over prior methods of not requiring pair-wise labeled data from the two domains. All that is required is batches of data from each domain and labels corresponding to their ground truth domain. The GAN approach builds a representation that attempts to fool a discriminator as to the true origin of the data thereby learning a mapping from one domain to the other.

This was recently applied to sim-to-real transfer for a robotic table-top-reaching task with a 7 DoF arm [8]. The authors show the ability to effectively transfer the learning of visuomotor policies from a simulation environment to the real setup by the use of very few real expert demonstrations for fine-tuning. The architecture consists of two key components:

- A perception module that estimates the object position $\mathbf{x}^*$ from a raw-pixel image I (based on a VGG16 neural network [41]);
- A control module that estimates the optimum joint velocities $\mathbf{v}$ given the position $\mathbf{x}^*$ and joint angles $\mathbf{q}$.

The source encoder is first pre-trained using labelled simulated data of images and corresponding target positions. Then, the source encoder (E_s) is locked and a reference target encoder (E_r) is trained through images sampled from both the simulation (I^s) and the real (I^r) setup. They use an adversarial loss $L_{Ad} = L_D + \gamma L_E$ where

$$\begin{aligned} L_D &= -\frac{1}{2m}\sum_j [\log D(E_s(I_j^s)) + \log(1 - D(E_r(I_j^r)))] \\ L_E &= -\frac{1}{m}\sum_j \log D(E_r(I_j^r)) \end{aligned} \tag{5}$$

Here, D denotes the discriminator and γ is a balancing weight. In practice the authors use a supervised loss over real expert demonstrations in addition to the adversarial loss for successful transfer. This method is appealing since it provides a principled way to transfer learned policies from simulation to the real robot with limited and not necessarily pairwise matched labeled data from the real robot. However, the authors explicitly consider the output of the perception module to correspond to object position and formulate the control module to map from positions to velocities. Letting the image encoding of the perception module correspond to position restricts the wide scope of latent features that can be encoded, and hence we do not explicitly force the encoding in our model to correspond to one particular tangible attribute (like position). However this introduces a difficulty in sim-to-real transfer because there is no ground-truth supervision for the perception module alone. In our proposed method, we train end-to-end in simulation and hence require no ground truth perceptual data, only a select number of expert trajectories to be used in the outer-loop imitation learning loss.

III. METHOD

The basis of our approach is inspired from two areas of recent rapid development: meta-learning for planning, and discriminative policy transfer. An overview of the approach is shown in Fig. 1.

A. Proposed End-to-End Planner

We build our planner, which consists of the encoder f_ϕ, the forward dynamics model g_θ and the planning loss $\mathscr{L}_{plan}$ in a UPN-style framework.

1) Stochastic Forward Dynamics Model: In UPN [9], the forward dynamics model g_θ is fully deterministic, which makes the model inappropriate when applied to a real robot, since transitions are not deterministic (especially if the next state conditioned on the previous state is not unimodal), as well as making the model brittle to slight perturbations in the initial and/or goal image. We capture this intuition for making our model robust by explicitly encoding *noise* in the dynamics model:

$$\hat{x}_{t+1} \sim g_\theta(\hat{x}_t, \hat{a}_t, \varepsilon) \tag{6}$$

where ε is sampled from a zero-mean, fixed variance normal distribution.

2) Learning the Planning Loss Function: Most existing approaches [5], [9], [37], [42] use a fixed loss function, like squared error loss or Huber loss [9]. We alleviate the modelling bias introduced by a fixed loss function by adopting one with tunable parameters. In particular, we use a Multi-Layer Perceptron (MLP) as the planning loss, the parameters of which are "meta-learned" through the outer loop imitation loss. Our new inner loop planning loss becomes:

$$\mathscr{L}_{MLP} = MLP(\hat{x}_g, x_g) \tag{7}$$

The intuition behind using an MLP as the loss function is to let the model suitably adapt the loss function to any particular task by tuning the parameters of the MLP.

3) Faster Convergence Through Regularization: The original UPN framework is relatively data inefficient since all information about the latent encoding parameters and the dynamics model must be learned from the outer loop imitation loss. We propose two forms of regularization to the model to alleviate this.

The first is a "smoothness" regularization which enforces the successive latent states to be "close" to each other in latent

Algorithm 1 Sim-to-Real Transfer of Navigation Policy

Randomly initialize θ, ϕ, ζ
$f^s_\phi,\ g^s_\theta,\ MLP^s_\zeta$ = TRAINING($a^{sim*}_{t:t+T},\ \beta_1,\ \beta_2,\ \beta_3,\ \alpha,\ n_p$)
f^t_ϕ = TRANSFER($\{o^{real}\},\ \{o^{sim}\},\ f^s_\phi,\ k$)
$\phi \leftarrow f^t_\phi, \theta \leftarrow g^s_\theta, \zeta \leftarrow MLP^s_\zeta$
$f_\phi,\ g_\theta,\ MLP_\zeta$ = TRAINING($a^{real*}_{t:t+T},\ \beta_1,\ \beta_2,\ \beta_3,\ \alpha,\ n_p$)

space. Since, the transition from $\hat{x}_t$ to $\hat{x}_{t+1}$ occurs as a result of action $\hat{a}_t$ on a physical robot (i.e., $\hat{x}_{t+1} \sim g_\theta(\hat{x}_t, \hat{a}_t, \varepsilon)$) we should expect that, in order to have a smooth trajectory, the "distance" in latent space between subsequent state encodings should be small. We enforce this by adding the the following term to the planning loss:

$$\mathscr{L}_{smooth} = \sum_{t=t}^{t=g} ||\hat{x}_t - \hat{x}_{t+1}||_p \tag{8}$$

where $||\cdot||_p$ denotes the L_p norm. Note that since $g_\theta(\hat{x}_t, \hat{a}_t, \varepsilon)$ is a distribution, $\hat{x}_t$ is a sample from that distribution.

The second type of regularization enforces "consistency". The original planning loss enforces a notion of consistency but only at the terminal state x_g. By consistency, we mean that the error represents the discrepancy between the terminal latent states calculated two ways: 1) by encoding the goal image and 2) by encoding the initial image and propagating the latent state through the dynamics model T times. However, in practice during training we have the entire sequence of images. Therefore, we can enforce consistency at each timestep *regardless of the policy being executed to generate the data*. This is achieved by considering the two pathways that we can use to arrive at the same latent state: 1) encode image at time t and propagate through the dynamics model and 2) encode the image at time $t+1$. More precisely, we enforce that samples from $g_\theta(f_\phi(o_t), a_t, \varepsilon)$ and $f_\phi(o_{t+1})$ are "close" to each other at every time-step t by adding:

$$\mathscr{L}_{consist} = \sum_{t=t}^{t=g} ||g_\theta(f_\phi(o_t), a_t, \varepsilon) - f_\phi(o_{t+1})||_p \tag{9}$$

to the planner loss function. Here, the first term is a sample from the respective distribution in each rollout. Note that here, a_t is sampled to be either the expert action (with a probability of 80%) or the current action (being optimized) at time-step t and o_{t+1} is the observed image at time-step $t+1$ after the agent takes action a_t in the state with observation o_t. An overview of the training process is outlined in Alg. 2.

B. Policy Transfer to the Real Robot

Although a gradient-descent based planning algorithm is very general and powerful in the sense that it can be applied to different tasks, training through imitation learning is data intensive and requires many demonstrations, something which is not always possible to collect in a real environment. Hence, training in simulation and fine-tuning in the real setup is a promising direction for using such architectures in real robotic tasks like navigation and grasping. However, it is not immediately evident if a sim-to-real transfer architecture can be applied in this framework because the latent encoding does not have an easily interpretable physical meaning.

We propose a method based on pre-training in simulation, using an adversarial discriminative approach for policy transfer, followed by a fine-tuning approach on the real robot as detailed in Alg. 1.

Algorithm 2 Planner Training

procedure TRAINING($a^*_{t:t+T},\ \beta_1,\ \beta_2,\ \beta_3,\ \alpha,\ n_p$)
  for number of training iterations **do**
    Sample a batch of demonstrations $o_t, o_g, a^*_{t:g}$
    Take Randomized guess for the optimal plan $\hat{a}^{(0)}_{t:g}$
    for i from 0 to $n_p - 1$ **do**
      Compute $x_t = f_\phi(o_t),\ x_g = f_\phi(o_g)$
      for j from 0 to T **do**
        $\hat{x}^{(i)}_{t+j+1} = g_\theta(\hat{x}^{(i)}_{t+j}, \hat{a}_{t+j}{}^{(i)}, \varepsilon)$
      end for
      Compute: $\mathscr{L}_{plan} = \mathscr{L}_{MLP} + \mathscr{L}_{smooth} + \mathscr{L}_{consist}$
      Update: $\hat{a}_{t:t+T}{}^{(i+1)} = \hat{a}_{t:t+T}{}^{(i)} - \alpha \nabla_{\hat{a}_{t:t+T}{}^{(i)}} \mathscr{L}^{(i)}_{plan}$
    end for
    Compute $\mathscr{L}_{imitate} = ||\hat{a}_{t:g} - a^*_{t:g}||^2_2$
    Update $\theta := \theta - \beta_1 \nabla_\theta \mathscr{L}_{imitate}$
    Update $\phi := \phi - \beta_2 \nabla_\phi \mathscr{L}_{imitate}$
    Update $\zeta := \zeta - \beta_3 \nabla_\phi \mathscr{L}_{imitate}$
  end for
  return $f_\phi,\ g_\theta,\ MLP_\zeta$
end procedure

Algorithm 3 Sim-to-Real Transfer

procedure TRANSFER($\{o^{real}\},\ \{o^{sim}\},\ f^s_\phi,\ k$)
  for number of training iterations **do**
    for k steps **do**
      Sample a batch of N real images $o^{real}_{1:N}$
      Sample a batch of N sim images $o^{sim}_{1:N}$
      Update Discriminator D : $\nabla_{\theta_d} L_D$
    end for
    Sample a batch of N real images $o^{real}_{1:N}$
    Update Generator (Target Encoder) f^t_ϕ by ascending its stochastic gradient: $\nabla_{\theta_d} L_G$
  end for
  return f^t_ϕ // Target encoder
end procedure

1) Pre-training in simulation: Expert trajectories are very inexpensive to obtain in a simulation (once the simulator has been built) and therefore this represents the bulk of our training phase.

2) Adversarial transfer of encoder from sim-to-real: Once we have a policy that is performing well in the simulator, we aim to learn an encoder that generates the same distribution of latent states over real images as the pre-trained encoder. To achieve this we begin by freezing the source encoder's learned weights. We feed in images sampled randomly from the simulation environment and execute one forward pass through the source encoder to yield a latent embedding $x_{sim} = f^s_\phi(o^{sim})$ where $f^s_\phi(\cdot)$ is the simulator encoder. We initialize the target encoder with the same weights as the source encoder but do not freeze them (i.e. the weights of the target encoder are trainable). The target encoder is fed images randomly sampled from the real environment and we execute one forward pass to yield a latent embedding $x_{real} = f^t_\phi(o^{real})$ where $f^t_\phi(\cdot)$ is the real robot encoder.

We then use a three-layer feedforward neural network

as a discriminator (D) to distinguish between which latent representations are obtained from images of simulation and which are obtained from real images. This is an adversarial learning framework where the generator is the target encoder that tries to generate latent representations from real images which are close to the representations of the trained source encoder on images from simulation. The discriminator and generator losses used in Alg. 3 are:

$$L_D = -\frac{1}{2N}\sum_{i=1}^{N}[\log D(f_\phi^s(o_i^{sim}) + \log(1 - D(f_\phi^t(o_i^{real}))]$$

$$L_G = -\frac{1}{N}\sum_{i=1}^{N}[\log D(f_\phi^t(o_i^{real})]$$

If the process of adversarial domain transfer is perfect, then without changing the rest of the architecture, the forward dynamics model g_θ^s and MLP loss function MLP_ζ^s pre-trained on simulation affixed to the target encoder f_ϕ^t should be able to perform well in the real environment. In practice, due to imperfect convergence of adversarial training, we need to incorporate fine-tuning with some expert demonstrations from the real environment. This is exactly similar to the pre-training phase, except for the fact that expert trajectories are from the real environment.

IV. EXPERIMENT DESIGN

To test the performance of our architecture, we designed two experiments on the Duckietown [10] platform: lane following and left turn. For each test run, we selected different initial poses for the Duckiebot, with each pose being a pair of initial position and initial facing angle.

In simulation, for the lane following test, we select the initial angles from the range -30° to 30° and the initial positions from the center of the right lane to the center of the left lane. For the left turn test, the initial angle ranges from -30° to 30° and the initial position ranges from the center of the right lane to the broken yellow (middle) line. We randomly generate a number of initial poses in the above mentioned ranges during testing and a number of expert trajectories of different horizon lengths during training.

In the real environment we uniformly discretize the space of initial poses. For lane following, there are three initial positions, namely center of the right lane, left lane and yellow line and seven values of initial angles (-45°, -30, -15°, 0°, 15°, 30°, 45°). For the left turn test, there is one intiial position, namely the center of the right lane and five initial angles (-30, -15°, 0°, 15°, 30°). See Figure 2.

A. Dataset Collection

The dataset for training consists of expert trajectories in simulation, expert trajectories in the real setup and images from both the simulator and real setup (sim/real frame data) in any context. The expert trajectories in both sim and real are collected with a joystick. Each trajectory consists of a pair of actions and corresponding observation frames from the agent's point of view.

The sim/real frame dataset contains a list of image-label pairs, where the label corresponds to the domain (either sim or real). The images from the simulator were collected using basic domain randomization with respect to camera

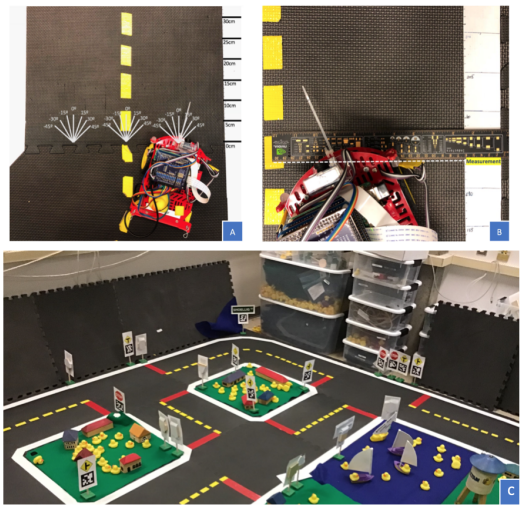


Fig. 2: **The Duckietown Environment**: (a) The initial pose setup for the Duckiebot in the real Duckietown environment. (b) A demonstration of distance measurement. (c) An overview of the Duckietown environment.

height, angle, field of view, floor color, horizon color and pose of the robot. The real images were collected though the front camera of a physical Duckiebot by ensuring capture of different facing angles and positions on the road.

B. Training

For all experiments, we train the model in a curriculum learning style during the pre-training (in sim) and fine-tuning (in real) phases. In practice, this means that while sampling trajectories for each batch, we consider those with shorter horizon lengths before the longer ones and the lane-following trajectories before the turning ones.

V. RESULTS

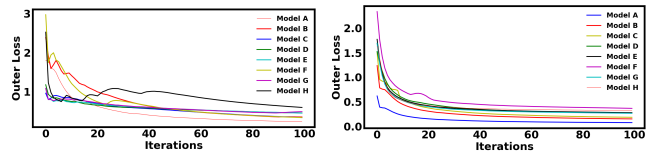
The performance of the framework has been measured by four metrics: **outer loss** ($\mathscr{L}_{imitate}$), **inner loss** ($\mathscr{L}_{plan}$), **average reward per time step** (simulation only), and **average completion rate** (fraction of the total distance to goal travelled by the Duckiebot before falling off the road averaged over all test instances with the same initial conditions). The reward function is given by

$$r = \begin{cases} v \cdot dir - 10|d_c|, & \text{if on the right lane} \\ 0, & \text{otherwise} \end{cases}$$

where v is the velocity of the Duckiebot, dir is the moving direction of the Duckiebot and d_c is the distance of the Duckiebot away from the right lane center.

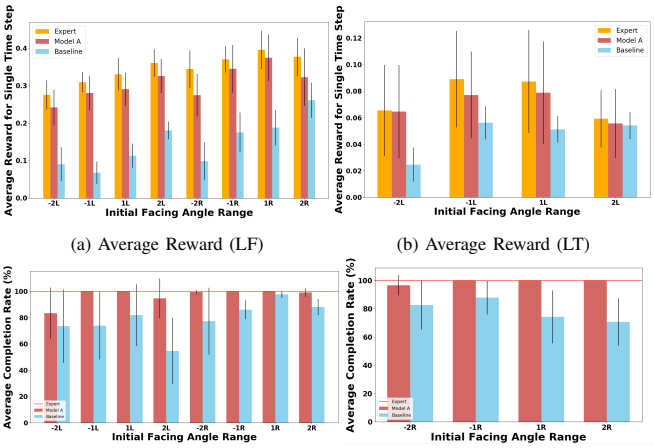
A. Convergence Analysis of the Planner Module

Here we analyze the efficacy of the key components of the planner module proposed in Sec. III. Fig. 3a depicts the convergence of the models during pre-training in simulation through the training procedure in Alg. 2. Fig. 3b shows the convergence of the models during fine-tuning by the use of real expert trajectories. It is evidenced from both the figures that Model A, which is our final model incorporating all the components described in Sec. III has a much steeper convergence rate and also converges to a better optimum.



(a) Model convergence on simulator data (b) Model convergence on real data

Fig. 3: Evaluation of various baselines models on Duckietown simulation (a) and real (b) environment showing the convergence of outer loss as training progresses. (Model A denotes our final planner, B is the version without Stochasticity, C is the version with Huber Loss instead of MLP as the planning loss, D does not incorporate regularization, E is D sans Stochasticity, F is C sans Regularization, G is C sans Stochasticity and H is the vanilla UPN [9] planner).



(a) Average Reward (LF) (b) Average Reward (LT)

(c) Completion Rate (LF) (d) Completion Rate (LT)

Fig. 4: Evaluation of the average time step reward and average completion rate on Duckietown Simulator (Notation: L - left lane, R - right lane; −2 - (-30º, -15º), −1 - (-15º, 0º), 1 - (0º, 15º), 2 - (15º, 30º); LF - lane following; LT - left turn).

B. Evaluation on Duckietown Simulation Environment

We now evaluate the performance of our model after pre-training in simulation through the training procedure described in Sec. IV-B. The results of the lane following test are shown in Fig. 4a and Fig. 4c and that of the left turn test are highlighted in Fig. 4b and Fig. 4d. We observe that Model A significantly outperforms the baseline UPN model. We claim that this improvement in simulation is a crucial stepping stone for effective sim-to-real transfer.

C. Evaluation of the Inner-Loop Loss Function

In our planner, we have a MLP as the inner-loop loss function whose parameters are learned in the outer imitation learning loop as described in Sec. III-A.2. After training the model, we fix the parameters of the MLP inner-loss and test for its value in different positions on the road. Intuitively, the value of the loss inferred by this function should he high near the center of the lane and should increase away from the center. Empirical evaluations in Fig. 5 justify that the loss function conforms to our intuition about its desired behavior.

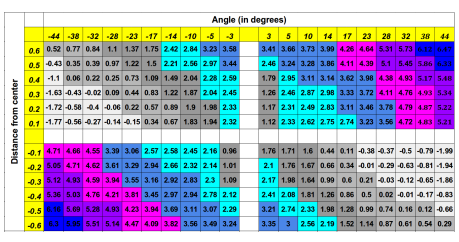


Fig. 5: Evaluation of the inner loop loss function on Duckietown Simulator. The evaluation is of Model A that has been trained on horizons of different lengths from 30 to 300 via curriculum learning.

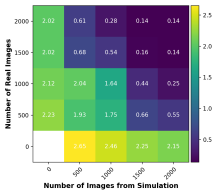
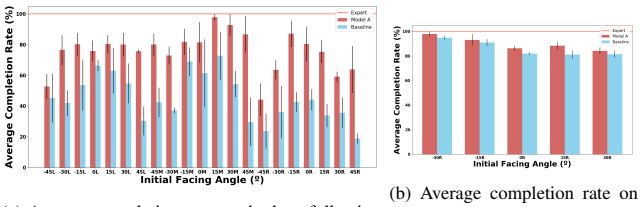


Fig. 6: Evaluation of the performance of adversarial transfer in terms of loss as a function of the number of real and simulated images used in training, after pre-training in simulation with 2000 expert demos.



(a) Average completion rate on the lane following test (real). (b) Average completion rate on the left turn test (real).

Fig. 7: Performance of our method on the real robot (Notation: L - left lane center, M - Middle line center, R - right lane center;)

D. Efficacy of the Transfer to the Real Robot

After pre-training in simulation and performing adversarial domain transfer, we fine-tune the model in the real setup. The architecture used is our final Model A. The results of the lane following test are shown in Fig. 7a and that of the left turn test are highlighted in Fig. 7b. We use domain randomization [12] as baseline against which we compare our sim-to-real transfer architecture[1].

It is interesting to note that our model performs quite well ($> 50\%$ average completion rate) even for the most difficult case of navigation starting from the center of the left lane with an initial facing angle of -45º. Also of note is the fact that the performance on left-turn is quite good for our model. This is indicative of the curriculum learning framework, which first learns lane following followed by turning (in training) yielding noticeable gains during testing. We also evaluated how many real and simulated images were required for convergence of the adversarial loss, with results presented in Fig. 6, and also how many real trajectories were needed to achieve an equivalent outer-loop loss with and without our transfer learning pipeline, with results presented in Table I. From these two results, we see that our method preferentially uses "off-policy" data to save the amount of on-policy expert trajectories needed on the real robot.

TABLE I: The number of real trajectories required in the proposed sim-to-real transfer compared to training the model directly without sim-to-real.

Outer loss	0.10	0.15	0.20	0.25	0.30	0.35
No. of Real Trjs (Direct)	1250	1150	950	750	500	200
No. of Real Trjs (Transfer)	230	180	120	75	50	25

VI. CONCLUSION

We present a framework for gradient-based planning and transfer from sim-to-real. We demonstrated through experimentation that the proposed method achieves significant performance gains in the real environment by learning a robust policy in simulation followed by a successful adversarial transfer.

[1]For a video of the real robot results please refer to this link

REFERENCES

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[2] M. Zhang, X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine, "Deep reinforcement learning for tensegrity robot locomotion," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 634–641.

[3] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2786–2793.

[4] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[5] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv preprint arXiv:1703.03400*, 2017.

[6] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," *arXiv preprint arXiv:1806.02817*, 2018.

[7] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," *arXiv preprint arXiv:1806.03836*, 2018.

[8] F. Zhang, J. Leitner, Z. Ge, M. Milford, and P. Corke, "Adversarial discriminative sim-to-real transfer of visuo-motor policies."

[9] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," *arXiv preprint arXiv:1804.00645*, 2018.

[10] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1497–1504.

[11] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[12] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[13] M. Yan, I. Frosio, S. Tyree, and J. Kautz, "Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control," *arXiv preprint arXiv:1712.03303*, 2017.

[14] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[15] Y. Bengio, S. Bengio, and J. Cloutier, *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1990.

[16] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook," Ph.D. dissertation, Technische Universität München, 1987.

[17] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[18] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl ˜2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[19] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043v2*, 2018.

[20] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "One-shot learning with memory-augmented neural networks," *CoRR*, vol. abs/1605.06065, 2016. [Online]. Available: http://arxiv.org/abs/1605.06065

[21] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths, "Recasting gradient-based meta-learning as hierarchical bayes," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJ_UL-k0b

[22] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel, "Evolved policy gradients," *CoRR*, vol. abs/1802.04821, 2018. [Online]. Available: http://arxiv.org/abs/1802.04821

[23] P. Sprechmann, S. Jayakumar, J. Rae, A. Pritzel, A. P. Badia, B. Uria, O. Vinyals, D. Hassabis, R. Pascanu, and C. Blundell, "Memory-based parameter adaptation," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rkfOvGbCW

[24] I. Higgins, A. Pal, A. A. Rusu, L. Matthey, C. P. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "Darla: Improving zero-shot transfer in reinforcement learning," *arXiv preprint arXiv:1707.08475*, 2017.

[25] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," *CoRR*, vol. abs/1707.02267, 2017. [Online]. Available: http://arxiv.org/abs/1707.02267

[26] F. Sadeghi and S. Levine, "(cad)$ˆ2$rl: Real single-image flight without a single real image," *CoRR*, vol. abs/1611.04201, 2016. [Online]. Available: http://arxiv.org/abs/1611.04201

[27] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 23–30.

[28] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance-a case study for quadrupedal robots," in *Advances in Cooperative Robotics*. World Scientific, 2017, pp. 464–472.

[29] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. I. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *CoRR*, vol. abs/1511.03791, 2015. [Online]. Available: http://arxiv.org/abs/1511.03791

[30] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Model identification via physics engines for improved policy search," *arXiv preprint arXiv:1710.08893*, 2017.

[31] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3888–3895.

[32] U. Viereck, A. ten Pas, K. Saenko, and R. P. Jr., "Learning a visuomotor controller for real world robotic grasping using easily simulated depth images," *CoRR*, vol. abs/1706.04652, 2017. [Online]. Available: http://arxiv.org/abs/1706.04652

[33] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning," *arXiv preprint arXiv:1609.03759*, 2016.

[34] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," *arXiv preprint arXiv:1511.07111*, 2015.

[35] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *CoRR*, vol. abs/1710.06542, 2017. [Online]. Available: http://arxiv.org/abs/1710.06542

[36] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *CoRR*, vol. abs/1511.07111, 2015.

[37] J. Tobin, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," *arXiv preprint arXiv:1710.06425*, 2017.

[38] F. Zhang, J. Leitner, B. Upcroft, and P. I. Corke, "Vision-based reaching using modular deep networks: from simulation to the real world," *CoRR*, vol. abs/1610.06781, 2016. [Online]. Available: http://arxiv.org/abs/1610.06781

[39] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," *CoRR*, vol. abs/1612.07828, 2016. [Online]. Available: http://arxiv.org/abs/1612.07828

[40] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017, p. 4.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[42] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.