Motion Planner Augmented Deep Visuomotor Policy Learning

Venkata Pradeep Kadubandi, Gautam Salhotra, Gaurav Sukhatme, Peter Englert
Robotic Embedded Systems Laboratory
Department of Computer Science
University of Southern California
kaduband@usc.edu, pradeep.v.kadubandi@gmail.com

***Abstract*—In this work, we explore infusing motion planning algorithms as an expert to drive the generalization performance or sample efficiency in visuomotor policy learning. Prior work explored end to end learning, auto encoders or pose based policies in training the visual portion of visuomotor policies. We focus our attention instead on guiding the policy learning using motion planner and combining with an auto encoder to achieve good end to end performance with only a few samples. Through simulated experiments, we demonstrate the effectiveness of such end to end policies and compare their performance with prior work.**

***Index Terms*—deep learning, motion planning, auto encoders**

I. Introduction

Humans effortlessly operate in their environment using vision. Such a capability is an ambitious goal for robots to achieve today. There has been a lot of interest and recent work that aims to learn robotic control directly from raw sensory image inputs. One theme in such efforts is training the perception system jointly with the control policy in an end-to-end fashion using experience. Another common approach has been to use representation learning techniques like Auto Encoders [TODO cite]in an unsupervised fashion to learn latent representations of high dimensional raw sensory input which can then be used for control in various ways. There are two limitations of such methods - training deep neural networks which is the core approach in these methods require huge volumes of training data and acquiring such real world data in robotics is impractical. Simulators can be an effective approach for gathering data, however transferring policies trained in simulation to work well in the real world remains an important challenge. Typically during simulation, in addition to raw sensory input, we may have access to additional information about the robotic environment. For example, this could be a low dimensional but more meaningful state representation like shape and positions of goals and obstacles in the environment. When such a low dimensional environment state is available, existing sampling based motion planning algorithms combined with trajectory optimization techniques are well proven to be able to generate a smooth feasible path in the environment towards reaching the goal state. Motion planning and trajectory optimization however rely on a known system model and also won't work with high dimensional raw sensory input like images in their direct form.

Through this work, we aim to explore the question: Can we use trajectory data generated from a motion planning algorithm with access to a low dimensional environment state as expert behavior to improve either the sample efficiency or transfer of policies from high dimensional raw sensory observations through imitation? At the heart of our approach is learning a mapping from the latent state representation to the input of a policy learned by imitating a motion planner. Our key insight is that known planning algorithms may offer more meaningful cues for representational learning. So, it can prove to be more sample efficient than model free reinforcement approaches. We initially learn a lower dimensional state representation from images using an auto encoder with reconstruction being the cue and a policy that operates on low dimensional environment state using behavior cloning from motion planner generated data. We then learn a mapping from the latent state representation to the policy input to learn an effective end to end policy that yields an action given the image observation of environment. The approach can be extended to work in sim-to-real transfer setup. Image based observations are easy to obtain both in real world setting and simulation whereas access to low dimensional environment state maybe only available in simulation settings. We can learn a policy working with the low dimensional environment state at simulation time and combine it with image auto encoder learned for real world setting to learn an effective policy to work with high dimensional images in real world setting.

We show preliminary results from two experimental settings one using a point robot and goal in a planar environment and another using a 3-joint robotic arm to reach a goal configuration. Our experiments show that effective policies can be learned in a high-fidelity simulation using our method. We intend to extend to scenarios where the data distribution is different in real world setting from that of simulation. Rest of the paper is organized as follows: In section II, we briefly review the related work. Section III elaborates on our approach. We give the details of experiments conducted and results in section IV and end with concluding remarks and future work in the final section.

II. Related Work

Controlling robots directly from raw sensory vision input has been an active area of research. Model free RL methods

have been applied to this problem in various works [1]. Another approach is behavior cloning [2] where a policy is directly learned from expert demonstrations in a supervised end to end fashion. Behavior cloning approach is one of the baseline method we used for our experiments. However these end to end learning methods often require large number of samples and do not learn meaningful latent representations towards solving the problem.

These RL methods formulate the problem as a Markov Decision Process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$ consisting of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition function $P(s' \in \mathcal{S}|s, a)$, reward $R(s, a)$, initial state distribution ρ_0, and discount factor $\gamma \in [0, 1]$. The agent's action distribution at time step t is represented by a policy $\pi_\phi(a_t|s_t)$ with state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$, where ϕ denotes the parameters of the policy. Once the agent executes the action a_t, it receives a reward $r_t = R(s_t, a_t)$. The performance of the agent is evaluated using the discounted sum of rewards $\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$, where T denotes the episode horizon.

Several works also take the route of learning latent representations towards solving the control problem. Embed to control [3] is closely related to our work in learning latent representations, however they don't learn policy directly and instead use known trajectory optimization algorithms on latent space to infer optimal control. [4] took a different route of using self-supervised correspondence as the cue for learning intermediate representations. Our work combines motion planning algorithms with policy learning towards learning latent representations.

Motion planning has been proven to be effective in generating a collision-free path in cluttered environments using explicit models of the robot and environment. Probabilistic roadmaps (PRMs) [] and rapidly-exploring random trees (RRTs) [] are two common sampling-based motion planning techniques. However, these methods typically work with low dimensional state representations and cannot by applied directly to raw sensory input.

Combining motion planning algorithms with robot learning has been attempted in several works before. [5] attempts to adapt motion planning algorithms to high dimensional settings by introducing learning non-linear functions for each motion planning primitives and is orthogonal to our work. Our work uses motion planning in guiding robot learning. More recently [6], attempts to apply RL techniques to motion planner generated data to learn manipulation tasks in cluttered environments. Our work on the contrary studies the problem as supervised learning problem with the help of auto encoders and policies learned from motion planner generated data.

III. Method

We consider the problem of policy search where the goal is to learn a policy $\pi_\theta(u_t|s_t)$ which outputs the actions u_t for an intelligent agent based on the state s_t of the environment. The environment is a markov decision process (MDP) whose state evolves according to unknown non-linear dynamics. We consider two kinds of states of the environment - one a high dimensional raw sensory state I_t like an image from a camera hosted on the agent, second a more specific low dimensional state y_t, for example a specification of goal position and shape and positions of obstacles in the environment. In addition to this, we assume that we always have access to robot internal state x_t for example specified as position or joint angles. Different dynamic models can be represented for robot state, environment state and image state and these are assumed to be unknown. Table I shows a summary of the terminology we use in this paper. In the following sub sections, we explain the input data requirements and overall objective, auto encoder learning, policy learning from motion planner and learning the latent mapping.

symbol	*definition*	*examples/description*
x_t	Robot state	position/orientation or joint angles
y_t	Environment state	position/orientation of goal/obstacles
I_t	state represented as Image	camera image
u_t	Action of agent	velocity or torques
z_t	Latent representation	Intermediate representation of high dimensional image. Typically the output of image auto encoder. This is typically lower dimensional than I_t but higher dimensional than y_t.
g_I	$\mathrm{I}_t \rightarrow z_t$	Image auto encoder
π_y	$(\mathrm{x}_t, y_t) \rightarrow u_t$	Low dimensional policy
π_I	$\mathrm{I}_t \rightarrow u_t$	High dimensional policy
g_z	$\mathrm{z}_t \rightarrow y_t$	Latent Mapping (from latent state to low dimensional policy environment input)

TABLE I
SUMMARY OF NOTATION USED IN THIS ARTICLE.

A. Objective and Inputs

We make use of a motion planning algorithm to generate trajectories using the low dimensional environment state y_t in addition to robot state x_t at training time. We call this Planning data:

$$D_{plan} = \{(x^i_{0:T}, u^i_{0:T}, y^i_{0:T})\}_{i=1...M}$$

We also assume we have some demonstrations available for image observations as well at training time. This data will be used for training the combinations of auto encoder and a policy working with low dimensional data. We call this demonstration data:

$$D_{demo} = \{(x^i_{0:T}, u^i_{0:T}, I^i_{0:T})\}_{i=1...N}$$

Our objective is to learn an effective policy that can directly output the desired action from high dimensional Image observation i.e., to learn a function $\pi_I(u_t|I_t)$.

B. Auto encoder

Towards solving the problem, we learn an auto encoder $E_I(z_t|I_t)$ for learning a latent representation that can be trained in unsupervised fashion from the image observations of environment. Since the auto encoder only uses image observations (without access to the corresponding actions) for

training, we can assume that we have more samples of image observations of the system for learning an auto encoder. Thus, For learning an auto encode, we may use a separate data set of just the image observations of environment:

$$D_{auto} = \{I^i\}_{i=1...L} \; L >> N$$

Auto encoder is learned through reconstruction loss on image using either L1 or L2 reconstruction loss:

$$L_{auto} : ||g_I^{-1}(g_I(x_t, I_t)) - (x_t, I_t)||^*$$

In one setup when we have access to corresponding image I_t in addition to low dimensional environment y_t, we can directly learn an auto encoder that maps from Image to low dimensional environment using the reconstruction loss on y_t i.e.,

$$L_{I-to-y} : ||g_y^{-1}(g_I(x_t, I_t)) - (x_t, y_t)||^*$$

However, we only use this assumption in one of the experiments as a baseline comparison and don't rely on this assumption for our main approach.

C. Low Dimensional Policy

In addition to learning an auto encoder, using the data from motion planner, we learn a policy that maps low dimensional environment state to actions i,e., $\pi_y(u_t|(x_t, y_t)$. Our key insight here is that we can obtain ground truth data for learning such a policy by running existing sampling based motion planning algorithms and collecting ground truth data for robot state, environment state evolution for corresponding robotic actions.

D. Learning Latent mapping

We then learn a latent mapping $g_z(y|z)$ by combining the learned auto encoder and environment policy and train the combination using demonstration data D_{demo}. This gives us an end to end policy to work with images i.e., $\pi_I(u_t|I_t)$. We take the encoder network of pre-trained auto encoder, map it's output to the input of the low dimensional policy using another sub-network (usually another multi-layered fully connected neural net). We freeze the pre-trained encoder and low dimensional policy network weights. Using the demonstration data, we train the sub-neural net that maps from the latent representation z_t of image to the input of low dimensional policy y_t. We considered a few different variations for combining the two networks and training procedure which are outlined in the figure 1.

IV. Experiments

We evaluate our method on two different tasks: The first experimental setup is for a point robot in a planar 2-D grid environment where robot and goal are positions within the grid and robot moves using discrete directions as possible action space. The second experiment uses reacher like MuJoCo environment in which a 3-joint angle robotic arm moves from it's current configuration to reach a desired configuration.

We used several criteria for evaluating our method against the baseline. For evaluation, we use the learned policy with underlying environment as a black box to do a roll out. There are different choices for number of time steps of roll out. We considered roll outs that match the length of ground truth test trajectory and also a fixed horizon (a sufficiently large number is chosen to be higher than all ground truth trajectories) length roll out. While the first choice tells us how well the policy mimics the expert, the second choice signifies whether the agent stays at goal if it reached the goal. There is another potential choice of stopping the roll out when we are near the desired goal but that was not considered. We evaluate different criteria like goal deviation at the end of trajectory (how far is the reached goal from expected goal), trajectory error (the distance between robot and environment states of ground truth trajectory and roll out trajectory) and policy error (the distance between actions chosen by our agent over the trajectory from ground truth actions). Note that when the roll out is lengthier than the ground truth, the latter two criterion consider only the first portion of roll out that matches in length to ground truth. We also consider the number of trajectories that ended near the goal state (determined using a specific threshold value for goal deviation based on experimental setting) among all the test trajectories.

A. Point Robot in Planar Grid

For the point robot experiment, the image observation is a 32 x 32 gray scale image depicting a birds-eye-view where robot is a square centered at the robot position and goal is another square centered at the desired location in grid. x_t and y_t are 2-dimensional positions in the grid. The positions are determined based on dividing the grid into 32 x 32 cells. Robot can take a single step along 8-directions to neighboring grid cells so u_t is also 2-dimensional. We generate environment with a robot position and a goal position sampled randomly and we use a greedy planner which moves the robot greedily towards the goal in the optimal way for generating ground truth trajectory data. A sample environment and a trajectory is shown in Figure 2. The trajectory is generated by first moving the robot to align diagonally to the goal in shortest steps possible and then moving along the appropriate diagonal direction to reach the goal.

For this experiment we assume that we have both I_t and corresponding y_t are available together. So, both our plan and demo data set are of the form:

$$D_{plan/demo} = \{(x^i_{0:T}, u^i_{0:T}, y^i_{0:T})\}_{i=1...M}$$

We generated $M = 10,000$ trajectories for the training data set and the total number of time steps is roughly 130000. We generated a separate test data set with 500 trajectories containing about 6500 total sample time steps.

As we have I_t and y_t pairs together, we trained auto encoder with loss L_{I-to-y} for our baseline performance comparison in addition to the experiment variations mentioned in the method section. Table II shows the results we have observed (the experiment variations which are not shown did not produce decent results).

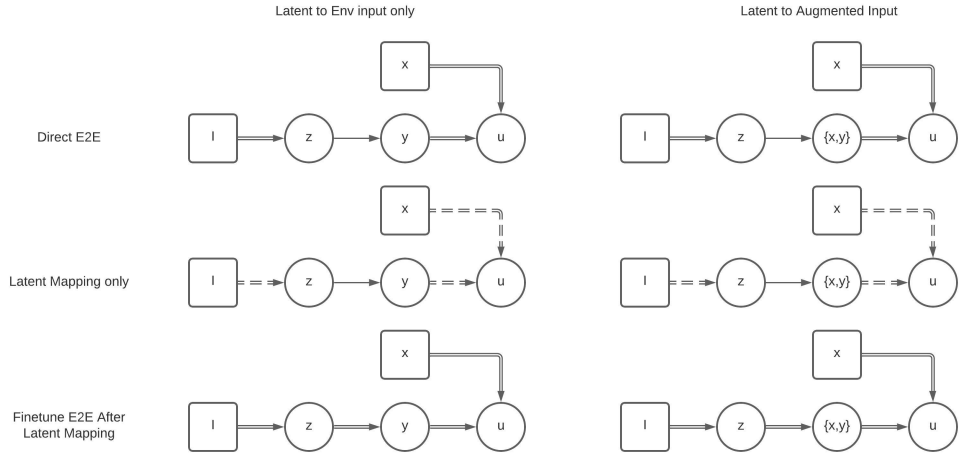

Fig. 1. Experiment Variations for combining and training g_I, g_z, π_y to produce a π_I. Broken arrow represents untrained/frozen links. Double arrow represents pre-initialized (from previous training) links. Rows represent variations in training procedure. In all experiments, latent state maps to the second input of policy and the first input is the ground truth robot state. In left column, we use a policy that takes only env state for second input. In right column, we augment the second input to contain both environment state and robot state (so the latent mapping learns to predict the robot state as well in addition to environment state.

Method/Metric	*Policy Accuracy % (Avg)*	*Policy Accuracy % (Worst)*	*Trajectory Loss (Avg)*	*Goal Deviation (Avg)*	*Goal Deviation (Worst)*	*Success Rate (in Reaching goal)*
Image To Env AE + Policy (baseline)	99.38	60	4.0e-3	2.5e-2	1.0	96.0%
Latent to Env only: Train Decoder only	88.64	0	2.5e-1	9.0e-1	24.5	60.8%
Latent to Env only: Train Decoder and Finetune E2E	99.18	60	1.2e-2	1.6e-2	1.0	97.6%
Latent to Augmented Input: Train Decoder only	95.84	0	7.7e-2	1.2e-1	1.5	79.2%
Latent to Augmented Input: Train Decoder and Finetune E2E	99.06	0	1.3e-2	1.3e-2	1.0	98.6%

TABLE II
COMPARISON OF VARIOUS PERFORMANCE METRICS ACROSS GRID ENVIRONMENT RUNS.

Fig. 2. Grid Environment and Sample Trajectory

B. MuJoCo Reacher

For this experiment, we used a MuJoCo based gym environment similar to open-ai Reacher which contains a robotic arm with 3-planar joints. Robot starts at an initial configuration and the task is to reach a desired configuration specified by goal configuration. Thus both x_t and y_t are 3-dimensional. Action u_t represents a small desired change in configuration at a time step (representing velocity) and is also 3-dimensional. A sample trajectory is shown in Figure 3.

We experimented with different data sets in this configuration: data set with uniformly random sampled start and goal configurations; a fixed start position across the data set; a fixed goal position across the data set; Different image observation sizes among (256, 256) or (128, 128) or (64, 64). We observed good results for the the base line method with fixed goal position for the task and (64, 64) size image observation. The results are only shown for this data set.

For this data set, both D_{plan} and D_{demo} training sets contain 900 trajectories with about 30000 sample time steps. We used a separate test set with 100 trajectories with about 3000 sample time steps. For training image auto encoder, we used a bigger data set D_{auto} with 100000 image samples of environment observations. We trained the auto encoder using reconstruction loss on image.

In this experiment, we used behavior cloning as the baseline for comparison with our approach. Table III presents the results we observed from the base line run and two experiment variations where we have seen good results.

We trained the baseline behavior cloning method for 500 epochs on the data set to get the results. The image auto encoder is trained for TODO epochs. The policy on low

Method/Metric	*Goal Loss (Avg)*	*Goal Loss (Highest)*	*Trajectory Loss (Avg)*	*Policy Loss (Avg)*	*Success Rate (in Reaching goal)*
Behavior Cloning	3.0e-2	8.3e-2	6.1e-2	9.2e-3	100%
Latent To Env only: Train Direct E2E	5.1e-3	2.1e-2	5.5e-2	9.3e-3	100%
Latent To Augmented Input: Train Direct E2E	1.6e-2	3.1e-2	7.9e-2	1.2e-2	100%

TABLE III
COMPARISON OF VARIOUS PERFORMANCE METRICS ACROSS REACHER RUNS.

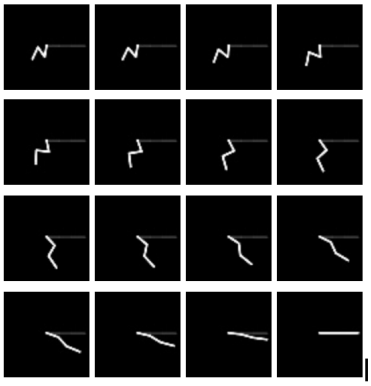

Fig. 3. A sub sample of Reacher Environment Sample Trajectory. The actual trajectory is 44 steps but only uniform intermediate steps are shown for brevity.

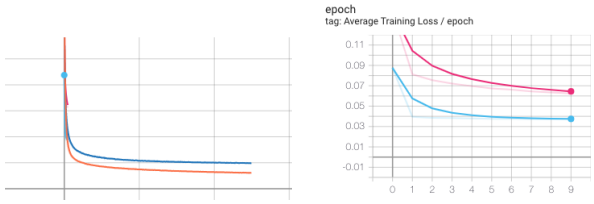

Fig. 4. Reacher Experiment Training Curves. Image on left shows baseline runs which are trained for 500 epochs. Image on right shows our methods which are fine tuned for 10 epochs end to end after using a pre-trained image encoder and low dimensional policy.

dimensional input is trained for TODO epochs. After this, we fine tuned end to end network combined with 10 epochs and seen performance matching or beating the baseline method. Figure 4 presents the comparison of training loss curves from these experiments.

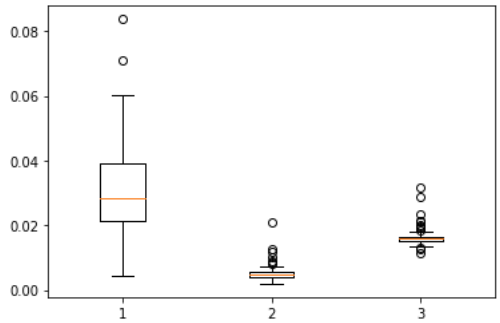

Fig. 5. Reacher Goal Deviation Box Plot Comparison. From left to right: 1.Behavior Cloning; 2.Latent To Env Input only Trained E2E; 3.Latent To Augmented Input Trained E2E

We compared different runs by generating a box plot of goal deviation across the entire test set for different experiments. We observed that our approach has better guarantees for final goal deviation and reaches the goal closer across different test trajectories. Figure 5 presents the comparison of box plots across different experiments.

V. CONCLUSION

In this work, we have shown that effective end to end policies with raw sensory input can be learned by augmenting the latent representation learned through an auto encoder with a policy learned from motion planner generated data. Our experiments show that such policies match in quality with existing end to end learning methods and can be trained faster to achieve the results.

In the current work, we assume that the underlying distribution governing the task where we apply motion planning and the goal task are same. Extending our work to apply to tasks with different underlying data distributions is future work that has important implications for sim-to-real transfer.

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2015. [Online]. Available: http://arxiv.org/abs/1504.00702

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[3] M. Watter, J. T. Springenberg, J. Boedecker, and M. A. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *CoRR*, vol. abs/1506.07365, 2015. [Online]. Available: http://arxiv.org/abs/1506.07365

[4] P. R. Florence, L. Manuelli, and R. Tedrake, "Self-supervised correspondence in visuomotor policy learning," *CoRR*, vol. abs/1909.06933, 2019. [Online]. Available: http://arxiv.org/abs/1909.06933

[5] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[6] J. Yamada, Y. Lee, G. Salhotra, K. Pertsch, M. Pflueger, G. S. Sukhatme, J. J. Lim, and P. Englert, "Motion planner augmented reinforcement learning for obstructed environments," in *Conference on Robot Learning*, 2020.