Leveraging Proprioceptive Measurements and Temporal Dependencies to Refine RGB Camera-based Pose Estimators

Josiah Wong
Stanford University
450 Serra Mall, Stanford, CA 94305
jdwong@stanford.edu

Abstract

Learning effective robot manipulation strategies is an ongoing and exciting area of active research. Specifically, learning how to coordinate multiple arms in a decentralized manner is especially challenging, as the problem formulation now becomes a set of POMDPs that must be solved on top of the classic RL task-solving problem! As part of an end-to-end solution seeking to address the multi-agent robot manipulation RL task setting, we leverage raw camera inputs and the active robot's proprioceptive observations to estimate the unseen states from the active robot arm's point of view. We show in simulation that a temporally dependent neural network can serve as an effective state estimator for estimating both static as well as dynamic states. Additionally, we show that leveraging proprioceptive observations improves the accuracy of the fully learned estimator.

1. Introduction

Because of robots' increasing ubiquity within the domestic sector, learning scalable and efficient approaches for solving common tasks within this domain is becoming similarly essential. Many of these tasks are manipulation-centric, and as such, contain peculiar properties and require more nuanced approaches than the typical reinforcement learning (RL) vanilla approach. As the demands and tasks become more complex, the necessity for multi-armed robotic interaction will become increasingly prevalent. We consider the case of such environments, which exhibit the important properties briefly described below:

- **High-dimensional, continuous action space**. Unlike the typical RL framework, which typically deals with low-dimensional discrete actions, learning to solve robotic manipulation tasks instead deal with controlling end effector(s), which are often parameterized by either joint or pose coordinates. The dimensionality also increases directly proportional to the number of arms being applied to the task, and is especially susceptable to the curse of dimensionality [2].

- **Dynamic dependency**. Unlike typical single agent RL tasks, which simply condition actions based on the given state, this multi-agent setting must take into account the (potentially non-optimal) policy of the other agents, resulting in an inherently non-stationary environment from the perspective of each individual robot. This can often lead to unstable / poor performance during training [5].

- **Multi-modal performance landscape**. Because manipulation tasks are fundamentally interaction-oriented, defining optimal grasps and other interaction metrics are both heavily dependent on the object(s) being interacted with and is highly sensitive to end effector pose [7].

From these properties, it is evident that learning accurate representations of other robot arms' poses and trajectories is both a crucial and nontrivial task. Therefore, the goal of this project is to learn a specific embodiment of such a representation, namely, the *end effector pose* of other robot arms from the perspective of a single robot arm. Our explicit goal for this project can be concisely described as follows:

Problem Statement. Consider the bimanual robot arm manipulation setting, in which one robot arm (which we will refer to as A^0) does not have explicit access to the entire state space and seeks to predict the other arm's (A^1) end effector pose, $X^1 = [X_p^{1\top}, X_\theta^{1\top}]^\top$, where X_p is the Cartesian coordinate tuple $[x_p, y_p, z_p]$, and X_θ is the quaternion tuple $[x_\theta, y_\theta, z_\theta, w_\theta]$. Given a history of proprioceptive measurements $\bar{X}^0_{0:t}$ (specifically, the noisy end effector pose self-measurements from A^0) and RGB camera observations from some arbitrary source $Y_{0:t}$ from time $\tau = 0, ..., t$, the goal is to predict A^1's true end effector pose state X^1_t at time $\tau = t$. We consider the *model-free* setting, in which we do not assume any a priori information about the object of interest we are measuring.

1.1. Related Work

Vision-based pose estimation has been extensively studied in the past, and has resulted in impressive results across a wide domain of applications. The authors of [17] published a canonical pose estimator framework dubbed PoseCNN which leverages a single RGB camera image to estimate the 3D translation and rotation of detected objects within the image with respect to the camera coordinate system. Of note is their usage of deconvolutions to re-inject resolution into their extracted features, and feeding components of the mid-level features as inputs into their network regressor. The results are relatively impressive, boasting $\sim 90\%$ accuracy with $\sim$ 1cm translation error for symmetric objects, though this metric deteriorates when generalized across the entire suite of objects used. However, unlike [17], we consider the case where we are able to leverage a relative frame of reference (via the noisy self-measurement from A^0) to better refine the camera RGB images.

Like PoseCNN, many of the current state-of-the-art methods leverage 3D models of the perceived objects in order to construct correspondence points in order to best fit a 3D orientation for the object, and refines the estimated 6D object pose over multiple network passes [18] [12]. However, such methods require accurate a priori knowledge of detected objects' geometric models, and requires multiple iterations per estimation. In contrast, we consider a model-free method, and simply infer the objects' geometric properties via the ground truth pose signals provided during training.

We also consider prior work that is explicitly relevant to robotic pose estimation task. The authors of [3] consider the identical task to ours, though the authors parameterize the robot state as joint angles instead of end effector state. Leveraging a purely synthetic training method that uses prelabeled depth images, the proposed framework classifies the robot joint links and then estimates each joint position via a weighted voting scheme. Similar to the prior case, the results show roughly $\sim 90\%$ accuracy with $\sim$ 5cm mean average precision threshold. Impressively, the authors were also able to show that their model is able to perform on real robots as well, despite being trained exclusively on simulated images. Interestingly, the authors do not use any convolutional networks but instead use a random forest classifier to classify image pixels to their respective joint classes. In contrast, [11] propose a single-shot regressor that solely uses RGB images, and leverages key points to learn a camera rotation matrix transforming these key points into the robot coordinate frame. However, both of these cases consider static camera locations with temporal independence between observations; in contrast, we use a non-stationary

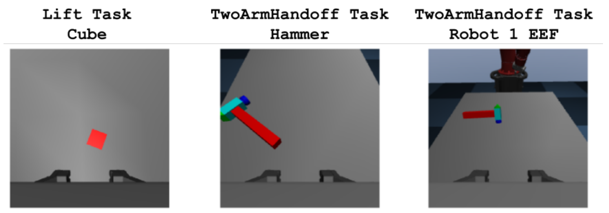

Figure 1. Visual Observations from each `robosuite` Task

camera and consider advantages from assuming temporal dependency between subsequent samples.

It should be noted that leveraging recurrent neural networks has been explored in the past, and has been shown to improve single-shot pose estimators within a dynamic or multi-estimator setting. [10] utilizes an LSTM to model the interdependency between joints when seeking to estimate the multiple 3D joint configurations from a human pose, while [13] explores a more similar case to ours, applying the LSTM to video stream to improve temporal estimator coherence and reduce flicking between images. Their results strongly suggest the efficacy of using such recurrent networks, and provides an empirical motivation for applying a similar model to our framework.

Lastly, we consider a prior work that explores a synthesis of classic state estimation with contemporary learning methods. The authors of [4] propose learning observation functions within the context of deterministic discriminative state estimators to directly estimate the posterior given a history of observations. Similarly, [1] considers learning Kalman Filter noise covariance matrices, as they are often difficult to accurately model beforehand. However, as Kalman Filters are a model-based approach, direct usage of the authors' approach requires a priori knowledge of the system dynamics, and implicitly assumes a self-governing system (i.e.: direct knowledge about the observed system). In our case, we assume no prior knowledge of the unobserved system dynamics of A^1.

It is important to note that the parameterization of the problem plays as essential role in both its tractability and properties during training. Indeed, orientation is especially delicate, as gradient-based methods may easily suffer from either singularities (e.g.: Euler angles) or over-parameterization of orientation space (e.g.: Quaternions / Rotation matrices) [8]. However, empirically, when directly employing geometric-based losses for orientation estimation, it seems that applying a quaternion-based method augmented with tuned heuristics performs well [16] [6].

2. Technical Approach

The guiding overview of our approach can be explained as follows: beginning with the most naive of assumptions, we evaluate simple model architectures and continually improve/refine the architecture based on iterative analysis of both quantitative and qualitative performance. Our initial thought is that accurate vision-based pose state estimators are difficult to learn on their own, but can be substantially improved by leveraging a relative frame of reference within the image (i.e.: proprioceptive measurements of a visually-observed agent). Intuitively, this is analogous to the case where we attempt to estimate the size of an object in a photograph (say, a sculpture). On its own, and without any prior knowledge, it may be difficult to infer its size; however, if a person or other familiar object can also be seen within the picture, it provides a relative frame of reference from which a more well-informed estimate can be anchored.

2.1. Model Architecture

Our proposed framework consists of two components: the **Feature Extraction** step and the **Pose Estimation** step. Both components are concisely described below, and can be seen graphically 2:

- **Feature Extraction.** We use a ResNet-50 model pre-trained on ImageNet to extract features from our RGB images at each timestep. Because ResNet's main outputs often fail to capture spatial information, we augment the main outputs features with additional auxiliary features from ResNet's BatchNorm layer after Conv1 layer. We then compress these multiple auxiliary feature channels into a single channel via a Conv1D layer, and append its flattened vector to the other features from ResNet's output, resulting in final feature vector z_t. Per ResNet's training protocol, we resize and normalize images before passing them as inputs. In contrast to typical feature extraction methods using pre-trained ResNets, all layer weights are updated during backprop to encourage learning spatial awareness.

- **Pose Estimation.** We append the proprioceptive measurement $\bar{x}_t^0$ from A^0 to the feature vector z_t and pass this as input to an LSTM module in order to capture the temporal dependencies between observations over time. At each timestep, the hidden state from the LSTM module h_t is passed to a final fully-connected (FC) network, whose output is the estimated state of the other robot's end-effector, $\tilde{x}_t^1$.

Our loss function is crucial to efficient and effective training. We consider the straightforward geometric losses for position:

$$L_{1,t} = \frac{1}{N}\sum_{i=1}^{N}\sum_{x,y,z} |\tilde{x}_t^{1(i)} - x_t^{1(i)}| \tag{1}$$

$$L_{2,t} = \frac{1}{N}\sum_{i=1}^{N} ||\tilde{x}_t^{1(i)} - x_t^{1(i)}||_2 \tag{2}$$

$$L_{\infty,t} = \frac{1}{N}\sum_{i=1}^{N}\max_{x,y,z} |\tilde{x}_t^{1(i)} - x_t^{1(i)}| \tag{3}$$

where N is the batch size. We also consider a combined version of the position loss, which is an equally weighted sum of l_1, l_2, and l_∞ losses:

$$L_{comb,t} = L_{1,t} + L_{2,t} + L_{\infty,t} \tag{4}$$

For orientation loss, we consider the loss presented in [16], that is, a simplified (non-trigonometric) version of quaternion distance, as well as an additive penalty to encourage outputted quaternions in the positive half-sphere ($x_{\theta,w} \geq 0$). Note that this assumes the ground-truth orientations to also exclusively exhibit quaternions in this half-sphere as well. The orientation loss is shown below:

$$L_{\theta,t} = \frac{1}{N}\sum_{i=1}^{N}\Big(1 - <\tilde{x}_{\theta,t}^{1(i)}, x_{\theta,t}^{1(i)}>^2 + \max(-x_{\theta,w}^{1(i)}, 0)\Big) \tag{5}$$

where $< q_1, q_2 >$ is the dot product between vectors q_1 and q_2.

The final loss is a weighted sum of the position and orientation losses:

$$L_t = L_{position,t} + \alpha L_{\theta,t} \tag{6}$$

where α is a scaling hyperparameter balancing the weight of the orientation loss component relative to the position component.

As our performance metric, we consider two components: the averaged per-step Euclidean L_2 norm between $\tilde{x}_{\theta,t}^{1(i)}$ and $x_{\theta,t}^{1(i)}$, and the averaged per-step angle between the corresponding quaternions.

2.2. Dataset

Our dataset consists of purely synthetic data drawn from simulations from multi-arm environments found in the open-source repository `robosuite`[1]. Specifically, we use the `TwoArmHandoff` environment, which consists of two robot arms around a table with a hammer placed between them. We instantiate this environment with a Franka Panda and Rethink Sawyer robots. Because real-time synthetic data is easily accessible, at train time we grab data in an online fashion, and run a set number of episodes (the "batch

[1]`https://github.com/StanfordVL/robosuite`

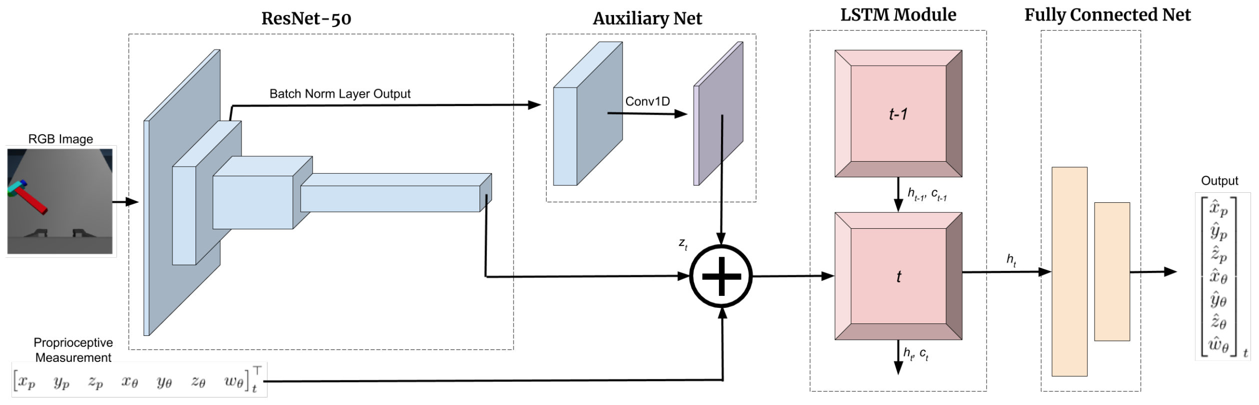
ResNet-50
Auxiliary Net
LSTM Module
Fully Connected Net
Batch Norm Layer Output
Conv1D
RGB Image
t-1
h_{t-1}, c_{t-1}
z_t
t
h_t
h_t, c_t
Output
$\begin{bmatrix} \hat{x}_p \\ \hat{y}_p \\ \hat{z}_p \\ \hat{x}_\theta \\ \hat{y}_\theta \\ \hat{z}_\theta \\ \hat{w}_\theta \end{bmatrix}_t$
Proprioceptive Measurement
$\begin{bmatrix} x_p & y_p & z_p & x_\theta & y_\theta & z_\theta & w_\theta \end{bmatrix}_t^\top$

Figure 2. Proposed Pose Estimator Model

size") with a fixed horizon and partition them into training and validation subsets to be used for that specific epoch. Examples of visual observations of each environment are shown in Figure 1. At a given time t during an episode, the training data consists of a single ($244 \times 244 \times 3$) RGB camera observation y_t attached to A^0's end effector, A^0's noisy and ground truth end effector state $\bar{x}_t^0 = x_t^0 + v_t$, and A^1's ground truth end effector state x_t^1, where noise $v_t \sim \mathcal{N}(0, \Sigma_{noise}) \in \mathbb{R}^7$. As the raw quaternions are both unrestricted and unnormalized (due to injected measurement noise), the end effector quaternion states are normalized and constrained to the positive half-sphere. The latter is accomplished by forcing $x_{\theta,w}$ to be non-negative.

We also consider the traditional object pose detection task, utilizing the same environment `TwoArmHandoff` in order to detect the hammer object pose. As this is a complex object, we also explore pose estimation of a much more simple cube object via the `Lift` environment. Note that unlike the end-effector pose estimation task, both of these objects serve as static variants of the general pose estimation task. However, each object is varied in its shape and size between episodes, providing an additional source of non-trivial stochasticity during training.

Episodes are dynamically generated and sampled as follows. Upon reset, both robots return to an environment-specific initial joint position configuration, perturbed by random uniform noise of 0.2 rad, while the object is placed with a random pose anywhere on the table. Both robots are controlled with an operational space position controller [14]. At each timestep, A^0 sends a delta position command $\Delta x_{cmd}^0 = (0, 0, 0.05)m$, effectively resulting in a uniform rising trajectory during the duration of the episode. At the beginning of the episode, A^1 initially uniformly samples some random delta position command, $(-0.05, -0.05, -0.05)m < \Delta x_{cmd}^1 < (0.05, 0.05, 0.05)m$ as well as a trajectory duration $\tau_{traj} \in 5, ..., 15$. Then, for $t = 1, ..., \tau_{traj}$, A^1 executes the same delta position command, after which it resamples a new command and trajectory duration, repeating the process.

2.3. Baselines

To isolate the significance of our proposed method, we construct and evaluate two ablated versions of our full-scale model, as described below:

- **Non-temporal.** We remove the LSTM module and instead simply directly pass the features and proprioceptive measurement into a larger multi-layered fully-connected (FC) network.
- **Image-only.** We do not leverage the proprioceptive measurements, and instead solely use the RGB camera observations to predict pose.

3. Experiments

Our evaluate our baselines as well as full model on the proposed environments above, which we can reference as *Cube*, *Hammer*, and *Robot1EEF* for concicesness. We use an episode horizon size of 20, using 5 and 2 episodes for training and validation per epoch, respectively. We use a noise scale $\Sigma_{noise} = 0.001I_7$, and loss scaling factor $\alpha = 0.5$. We choose output latent dimension of 512 for the ResNet-50 model, LSTM hidden dimension size of 512, and FC hidden layer sizes of [128, 32]. However, for our non-temporal baseline, we use a FC network with hidden layer sizes [1024, 256, 64]. Note that the input to our LSTM module is of size $n_{latent} + n_{aux} + 7 = 512 + 56^2 + 7 = 3655$. At the beginning of each episode, we set the initial LSTM hidden and cell states to zero. We use Adam optimizer [9], with momentum of 0.9 and loss rate of 0.001. We run each experiment for 5000 epochs or until the loss seems to converge. This results in up to 25000 training episodes / 500000 training steps. We evaluate each trained models' performance

Table 1. Model Performance on Predicting A^1's EEF Position

Model	$\bar{L}_2$ (cm)	σ_p (cm)	$\bar{\theta}$ (°)	σ_θ (°)
Full	3.186	1.629		
Image-only	4.410	1.682	3.740	1.756
Non-temporal	11.916	5.419	4.246	1.804

Table 2. Full Model Performance Across Various Tasks

Task	$\bar{L}_2$ (cm)	σ_p (cm)
Robot1EEF	3.186	1.629
Hammer	3.331	1.367
Cube	3.352	0.903

on the same random batch of 10 evaluation episodes generated from the same random seed, and measure the average per-step position error $\bar{L}_2$, angle distance orientation error $\bar{\theta}$, and their corresponding standard deviations σ_p, σ_θ. Our implemented models are publicly available[2] and were implemented in PyTorch [15].

While we tested all position losses, we empirically found the L_{comb} loss to perform the best, and use this loss component when evaluating and comparing our baselines to our proposed model. Moreover, we find that our orientation evaluation metric was inherently flawed, failing to constrain angles to be $+/-\pi$; thus, our orientation errors were heavily skewed and failed to convey the model's true performance. For example, an error of $|6.211| = 6.122$ rad error should instead be correctly interpreted as $|-0.072| = 0.072$ rad error, which is a significantly different! Fixing this issue results in substantially improved performance. Unfortunately, we were only able to re-train our baselines using this fixed method. Thus, while we provide the orientation error results for completeness for our baselines, in order provide consistent and informative analysis, we only consider the position component of the loss / error during our discussion, leaving the orientation-specific analysis for future work.

3.1. Results

Our results comparing our proposed model to the baselines can be seen in Table 1, with visualized rollouts shown at our online repo. Our full model reduces the positional error of the Image-only baseline by 27.7%, and of the Non-temporal baseline by 73.3%. While we expect our full model to strongly outperform the non-temporal baseline, we were surprised to observe only a marginal improvement over the image-only baseline. Such an observation may be readily explained, and will be discussed in our **Analysis** section below. Though we were unable to acquire accurate orientation metrics for our full model, our baseline orientation errors suggest that our proposed model is effectively able to learn desired objects' orientations. Interestingly, both the image-only and non-temporal baselines exhibit similar orientation performance, despite a significant positional difference. This is due to our robots' end effectors being controlled with a Position controller, which seeks to exclusively translate the end effector without rotation. Thus, a full Pose controller may be a more effective method to distinguish between each of our models' strengths.

Our full model's performance across various tasks can be seen in Table 2. As expected, as the object of interest increases in complexity, the estimation variance increases as well. Interestingly, the overall per-step mean accuracy in fact improves slightly with increasing complexity. A potential explanation is that complex objects have a greater visual set of features relative to simple or symmetric objects, allowing the model to infer pose with greater accuracy.

4. Analysis

4.1. Visualization

We first seek to qualitatively observe what our model is exactly learning. This can first be accomplished by visualizing our model's outputs, as shown in Figure 3. It is clear that in all cases, leveraging the early-stage outputs of the ResNet model via the auxiliary features are crucial, as they directly capture the pixel-wise spatial significance of the object being estimated. For the static cases (*Cube* and *Hammer*), the auxiliary feature output map is nearly binary, with the geom corresponding to the object's pose being exclusively activated. As these directly result from the early-stage ResNet batch normalization layer, this suggests that an excessively deep CNN model (such as ResNet) may perhaps be unnecessary in the static case, and instead only require a sufficient number of layers to extract spatial information.

However, as the estimated object becomes both dynamic and increases in complexity, the corresponding feature map appears to become non-stationary, as can be seen in the *Robot1EEF* feature maps. This suggests that these early convolutional layers are insufficient to directly predict pose in a one-shot manner, for the activation strength corresponding to the each respective visual feature changes in magnitude over time. Observing these dynamic feature maps further justify the usage of a temporally-dependent model (in our case, an LSTM), as such recurrent networks can directly capture the interdependence between individual per-step feature maps over time.

4.2. Comparison to Baseline Performance

As expected, our full model outperforms both of our baselines; however, it is interesting to note that the Image-only baseline only performs marginally worse. This can perhaps be explained by two factors. On one hand, due to the sheer dimensional magnitude of the feature inputs into the LSTM, it is difficult for the network to effectively learn

[2]`https://github.com/cremebrule/rgb-proprioceptive-pose-estimator`

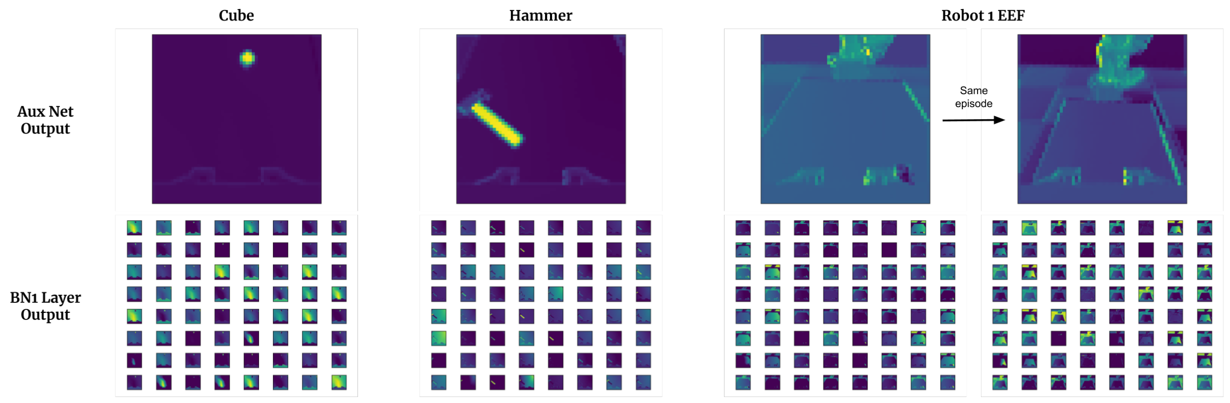

Figure 3. Visualization of auxiliary and early stage ResNet-50 Batch Normalization layer outputs across tasks

Table 3. Model Test-time Performance Over Varying Noise Levels

Noise Level (m^2)	$\bar{L}_2$ (cm)	σ_p (cm)
1.000	3.366	1.627
0.100	3.208	1.623
0.010	3.188	1.628
0.001 (Trained)	3.186	1.629
0.000	3.185	1.630

to heavily weight the proprioceptive measurements accordingly, which only composes 0.2% percent of the input! A perhaps more efficient structure would be to leverage separate LSTM modules for both the visual features and proprioceptive measurements, and then synthesizing each module's output during the final FC network phase. Secondly, proprioceptive measurements provide the most useful information when the visual inputs may become unreliable over time – for example, dynamic object tracking or unknown frames of reference. Unfortunately, our experimental setup has confounded these issues: while we account for localized random initial poses, we fail to explore global randomization for each robot's initial joint configurations. Since learning global frames of reference from high-dimensional observations (i.e.: camera observations) is an inherently difficult task, this type of randomization may serve as an effective method for highlighting proprioceptive measurements' ability to robustly improve pose performance.

4.3. Robustness to Randomization

In fact, this naturally leads us into discussion about the impact of randomization on our model's performance. During evaluation, we varied the noise variance applied to the proprioceptive measurements, and record the results, as shown in Table 3. Despite orders of magnitude difference in noise levels, the test-time performance of the model only improves marginally, supporting our inference that the model is not effectively leveraging the proprioceptive measurements. Moreover, we find that initializing the robots at test-time to configurations outside of the limited range used during training time results in significantly deteriorated performance, further highlighting the necessity for global randomization during training to increase model robustness.

4.4. Comparison to Model-Based Methods

We provide a brief qualitative and quantitative comparison to other model-based pose estimator methods that similarly use L_2 Euclidean metrics. [3] considers an RGB image-only method which is composed of an instance segmentation network followed by a pose estimator network, training exclusively on synethetic data and evaluating from a pre-defined real dataset. Our quantitative results for positional errors are competitive to theirs, which range between 3-4cm, and our preliminary baseline results for orientation error are comparable as well. [11] also considers an RGB-image only approach, and explores estimating robot keypoint poses via a transformation matrix. For their simulation dataset, they report an 88% accuracy within a 40mm threshold, which is comparable to our results. These brief comparisons suggest that our method (albeit, with a more smartly tuned network) exhibits potential to be competitively applied as an alternative model-free method to pose detection, with advantages within dynamic or temporally dependent settings.

5. Conclusion

In this project, we provide a novel model-free pose estimation framework that is both simple and fully learned end-to-end, and can effectively estimate both static and dynamic objects' position over time. We show that including temporal dependencies within our model architecture helps significantly improve overall performance when applied within a dynamic setting, and convey the potential for proprioceptive measurements to further refine pose accuracy. While the scope of our results are both limited and not satisfactorily robust, we suggest several possible remedies and adjustments to improve our method. Lastly, we provide qualitative insight into our model's performance and highlight the significance of including temporal dependencies within our model.

As the end goal is to apply such a general model in the real-world setting, maximizing the transfer sim2real performance is crucial. As mentioned before, a promising future work lies in increasing training randomization and adjusting our model architecture to allow for easier leveraging of the proprioceptive measurements. Additionally, domain randomization has proven to be an effective method for accelerating sim2real transfer learning, and serves as another potential avenue to explore. Our hope is that this simple, intuitive architecture serves as an alternative but effective method for estimating both static and dynamic poses within a model-free setting.

5.1. Acknowledgements

As part of PAIR lab, we would like to thank the lab for usage of their computer cluster and corresponding GPUs, on which we were able to accelerate our models' training significantly.

References

[1] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Ng, and Sebastian Thrun. Discriminative training of kalman filters. pages 289–296, 06 2005.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2012.

[3] J. Bohg, J. Romero, A. Herzog, and S. Schaal. Robot arm pose estimation through pixel-wise part classification. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3143–3150, 2014.

[4] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators, 2016.

[5] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity, 2017.

[6] H. Hsu, T. Wu, S. Wan, W. H. Wong, and C. Lee. Quatnet: Quaternion-based head pose estimation with multiregression loss. *IEEE Transactions on Multimedia*, 21(4):1035–1046, 2019.

[7] C. Huang and B. Mutlu. Learning-based modeling of multimodal behaviors for humanlike robots. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 57–64, 2014.

[8] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning, 2017.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[10] Kyoungoh Lee, Inwoong Lee, and Sanghoon Lee. Propagating lstm: 3d pose estimation based on joint interdependency. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 123–141, Cham, 2018. Springer International Publishing.

[11] Timothy E. Lee, Jonathan Tremblay, Thang To, Jia Cheng, Terry Mosier, Oliver Kroemer, Dieter Fox, and Stan Birchfield. Camera-to-robot pose estimation from a single image, 2019.

[12] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. 2018.

[13] Yue Luo, Jimmy Ren, Zhouxia Wang, Wenxiu Sun, Jinshan Pan, Jianbo Liu, Jiahao Pang, and Liang Lin. Lstm pose machines, 2017.

[14] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27:737, 06 2008.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[16] Jimmy Wu. Robotic object pose estimation with deep neural networks, 2017.

[17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes.

[18] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. Dpod: 6d pose object detector and refiner, 2019.